UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Departamento de Sistemas Informáticos y Computación

TESIS DOCTORAL

Computación Avanzada en Problemas Estáticos y Dinámicos de Gran Dimensión: Aplicación al Cálculo de Estructuras

Autor: José Miguel Alonso Ábalos

Directores: Dr. Ignacio Blanquer Espert
Dr. Adolfo Alonso Durá

Valencia, enero de 2016

A mi padre

Resumen

El trabajo realizado en esta tesis doctoral ha consistido en la aplicación de la Computación de Altas Prestaciones, las Tecnologías Grid y las Tecnologías Cloud al cálculo estático y dinámico lineal de estructuras de edificación e ingeniería civil mediante el método de los elementos finitos.

Gracias a la eficiencia de los métodos numéricos actuales, plasmados en librerías numéricas de dominio público, y a la disponibilidad de librerías de comunicación y sincronización entre los elementos de proceso, bien sea mediante paso de mensajes o por memoria compartida, la aplicación de la Computación de Altas Prestaciones al cálculo estructural ha permitido llevar a cabo un análisis riguroso y realista de estructuras de gran dimensión y complejidad, con unos tiempos de respuesta razonablemente reducidos. Además de utilizar librerías numéricas paralelas de dominio público (dedicadas a la resolución de sistemas de ecuaciones lineales mediante métodos directos con MUMPS, PaStiX y PARDISO o métodos iterativos con PETSc y *hypre* o la resolución del problema de valores propios generalizado con SLEPc y ARPACK) se ha empleado el paradigma de programación paralela multinivel, el cual combina el desarrollo de algoritmos mediante MPI y OpenMP, a fin de garantizar la portabilidad y obtener las mejores prestaciones en distintos sistemas operativos, como Linux y Windows, sobre diferentes plataformas computacionales paralelas.

El cálculo estático se ha llevado a cabo mediante el Método de la Rigidez, donde se han paralelizado las diferentes etapas que lo componen. En el caso del análisis dinámico, la ecuación dinámica de segundo orden se ha resuelto mediante la paralelización de numerosos métodos de integración directa, así como mediante técnicas de análisis modal, superposición modal y análisis modal espectral. En el caso concreto del análisis modal espectral, se ha paralelizado un amplio número

de métodos de combinación modal de los resultados, acompañados de distintas técnicas de combinación direccional y diferentes alternativas que intentan proporcionar el signo de los resultados.

Con el objetivo de que dicha aplicación paralela desarrollada pueda estar a disposición de una amplia comunidad de arquitectos e ingenieros estructurales, se han implementado y desplegado sendos servicios Grid y Cloud que ofrecen un análisis estructural por internet seguro, fiable y de alta productividad bajo una arquitectura orientada a servicio. Dichos servicios integran componentes, como GMarte, para la planificación de tareas, la recogida de resultados y la tolerancia a fallos en plataformas Grid o incorporan los desarrollos del proyecto europeo VENUS-C, para el despliegue del servicio sobre Microsoft Azure o sobre una infraestructura Cloud "on-premises" gestionada por COMPSs.

La competitividad demostrada por el Simulador Estructural en sus tiempos de respuesta, inclusive a nivel secuencial, con respecto a otros paquetes software disponibles y ampliamente utilizados, ha dado lugar a su incorporación en diferentes aplicaciones. Bajo un entorno de carácter investigador, forma parte de un sistema de optimización estructural, en colaboración con el departamento de Mecánica de los Medios Continuos y Teoría de Estructuras (DMMCTE) de la UPV. En un ámbito profesional, el Simulador se ha incorporado en dos aplicaciones comerciales. Por un lado en Architrave, una aplicación de cálculo estático y dinámico estructural, desarrollada por la UPV y con más de 3550 usuarios, dotada de un atractivo interfaz gráfico que se distribuye por parte de la empresa Preference. A nivel profesional, Architrave se ha descargado en 22 países y cuenta, a nivel académico, con alumnos y profesores de 54 universidades de 8 países diferentes. Adicionalmente, la empresa Preference ha incorporado la versión estática del Simulador a su producto PrefSuite, un paquete software comercializado a nivel mundial en el ámbito de la carpintería metálica y el cálculo estructural en 3D de muros cortina.

Resum

El treball realitzat en aquesta tesi doctoral ha consistit en l'aplicació de la Computació d'Altes Prestacions, les Tecnologies Grid i les Tecnologies Cloud al càlcul estàtic i dinàmic lineal d'estructures d'edificació i enginyeria civil mitjançant el mètode dels elements finits.

Gràcies a l'eficiència dels mètodes numèrics actuals, plasmats en llibreries numèriques de domini públic, i a la disponibilitat de llibreries de comunicació i sincronització entre els elements de procés, bé siga mitjançant pas de missatges o per memòria compartida, l'aplicació de la Computació d'Altes Prestacions al càlcul estructural ha permès dur a terme una anàlisi rigorosa i realista d'estructures de gran dimensió i complexitat, amb uns temps de resposta raonablement reduïts. A més d'utilitzar llibreries numèriques paral·leles de domini públic (dedicades a la resolució de sistemes d'equacions lineals mitjançant mètodes directes amb MUMPS, PaStiX i PARDISO o mètodes iteratius amb PETSc i *hypre* o la resolució del problema de valors propis generalitzat amb SLEPc i ARPACK) s'ha emprat el paradigma de programació paral·lela multinivell, el qual combina el desenvolupament d'algorismes mitjançant MPI i OpenMP, a fi de garantir la portabilitat i obtenir les millors prestacions en diferents sistemes operatius, com Linux i Windows, sobre diferents plataformes computacionals paral·leles.

El càlcul estàtic s'ha dut a terme mitjançant el Mètode de la Rigidesa, on s'han paral·lelitzat les diferents etapes que ho componen. En el cas de l'anàlisi dinàmica, l'equació dinàmica de segon ordre s'ha resolt mitjançant la paral·lelització de nombrosos mètodes d'integració directa, així com mitjançant tècniques d'anàlisi modal, superposició modal i anàlisi modal espectral. En el cas concret de l'anàlisi modal espectral, s'ha paral·lelitzat un ampli nombre de mètodes de combinació modal dels resultats, junt amb de diferents tècniques de combinació direccional i

diferents alternatives que intenten proporcionar el signe dels resultats.

Amb l'objectiu que aquesta aplicació paral·lela desenvolupada puga estar a la disposició d'una àmplia comunitat d'arquitectes i enginyers estructurals, s'han implementat i desplegat sengles serveis Grid i Cloud que ofereixen una anàlisi estructural per internet segur, fiable i d'alta productivitat sota una arquitectura orientada a servei. Aquests serveis integren components, com GMarte, per a la planificació de tasques, la recollida de resultats i la tolerància a fallades en plataformes Grid o incorporen els desenvolupaments del projecte europeu VENUS-C, per al desplegament del servei sobre Microsoft Azure o sobre una infraestructura Cloud 'on-premises" gestionada per COMPSs.

La competitivitat demostrada pel Simulador Estructural en els seus temps de resposta, inclòs a nivell seqüencial, pel que fa a altres paquets de programari disponibles i àmpliament utilitzats, ha donat lloc a la seua incorporació en diferents aplicacions. Sota un entorn de caràcter investigador, forma part d'un sistema d'optimització estructural, en col·laboració amb el departament de Mecànica dels Mitjans Continus i Teoria d'Estructures (DMMCTE) de la UPV. En un àmbit professional, el Simulador s'ha incorporat en dues aplicacions comercials. D'una banda en Architrave, una aplicació de càlcul estàtic i dinàmic estructural, desenvolupada per la UPV i amb més de 3550 usuaris, dotada d'una atractiva interfície gràfica que es distribueix per part de l'empresa Preference. A nivell professional, Architrave s'ha descarregat en 22 països i compta, a nivell acadèmic, amb alumnes i professors de 54 universitats de 8 països diferents. Addicionalment, l'empresa Preference ha incorporat la versió estàtica del Simulador al seu producte PrefSuite, un paquet software comercialitzat a nivell mundial en l'àmbit de la fusteria metàl·lica i el càlcul estructural en 3D de murs cortina.

Summary

The work implemented in this thesis is concerned with the application of High Performance Computing, Grid technologies and Cloud Technologies to the static and dynamic linear analysis of building and civil engineering structures by means of the finite element method.

Thanks to the efficiency of the current numerical methods, embedded in many public domain numerical libraries, and to the availability of communication and synchronization libraries among the processing elements, either by message passing or by shared memory, the application of High Performance Computing to the structural computation has allowed us to carry out a rigorous and realistic analysis of large dimension and complexity structures, with reasonably reduced response times. Besides using state-of-the-art parallel numerical libraries (devoted to the solution of systems of linear equations by means of direct methods with MUMPS, PaStiX and PARDISO or iterative methods with PETSc and *hypre* or the resolution of the generalized eigenvalue problem with SLEPc and ARPACK), the multilevel parallel programming paradigm has been employed, which combines the development of algorithms using MPI and OpenMP in order to ensure the portability and to obtain the best performance in different operating systems, such as Linux and Windows, on different parallel computational platforms.

The linear static calculation has been carried out through of the Stiffness Method where all its different stages have been parallelised. In case of structural dynamics, the second order dynamic equation has been solved by means of the parallelization of several direct time integration methods, as well as through parallel modal analysis techniques, modal superposition time history and response spectrum analysis. In the case of the latter analysis, numerous modal combination methods have been parallelized, accompanied by different directional combination

techniques and distinct alternatives that try to provide us with sign of the results.

With the aim that the above mentioned HPC-based application could be at the disposal of a wide community of architects and structural engineers, Grid and Cloud services have been implemented and deployed. They offer a secure, reliable and high throughput structural analysis under a service oriented architecture model. On the one hand, the Grid service integrates components, such as GMarte, for the task metascheduling, the results retrieval and the failure tolerance on Grid platforms. On the other hand, the Cloud service incorporates the developments of the European VENUS-C project for its deployment on Microsoft Azure infrastructure or on-premises Cloud platform managed by COMPSs.

The competitiveness demonstrated by the structural simulator in its response times, even when launched in a sequential approach, with regard to other well-known software packages, has given place to its incorporation in different software applications. From a research point of view, it is part of a structural optimization system in collaboration with the Continuous Medium Mechanics and Theory of Structures Department (DMMCTE) of the UPV. In a professional environment, the simulator has been incorporated into two commercial applications. First of all, it is one of the three components of Architrave, a software system for the the static and dynamic structural design and analysis, developed by the UPV and with more than 3550 users, equipped with an attractive graphical user interface that is distributed by the Preference company. Professionally, Architrave has been downloaded from 22 countries. Academically, it is being used by students and teachers belonging to 54 universities from 8 different countries. Additionally, the Preference company has incorporated the structural simulator static version to its PrefSuite product, a worldwide commercialized software package in the field of metallic carpentry and 3D structural analysis of curtain walls.

Agradecimientos

En primer lugar, quiero expresar mi más profundo agradecimiento a Ignacio Blanquer y a Adolfo Alonso por la dirección multidisciplinar de esta tesis, por su apoyo, por todo lo que he aprendido a su lado y por todas las horas que me han dedicado. Va por ambos mi admiración como profesionales y como personas.

Por otro lado, deseo manifestar mi enorme gratitud a Vicente Hernández, primer director de esta tesis, por acogerme en su grupo de investigación y por proporcionarme el soporte y las colaboraciones necesarias, a lo largo de todos estos años, para desarrollar este trabajo. Fue él, sin duda, quien marcó mi destino profesional, gracias a su llamada telefónica de aquel domingo por la tarde que con tanto afecto recuerdo. Lamento de corazón el fatal e injusto destino que a él, y a su familia, les toca estar viviendo.

Quiero también expresar mi agradecimiento a todos aquellos que me han ayudado, de un modo u otro y a lo largo del tiempo, a que esta tesis haya salido adelante: A Fernando Alvarruiz, compañero ideal de trabajo en mi incorporación al grupo. A José Román, Andrés Tomás y Eloy Romero, por sus consejos en el uso de librerías numéricas, especialmente en lo que respecta a SLEPc y PETSc. A Carlos de Alfonso, por su trabajo mano a mano en HiperBUILD. A Germán Moltó, por la implementación de GMarte como middleware utilizado en el servicio Grid y a Roberto López, por toda su labor en el desarrollo y despliegue de dicho servicio y por el interfaz gráfico de Grid4Build. Y en especial, a Pedro de la Fuente y a Pau Lozano, por sus valiosas e inmejorables aportaciones y desarrollos al servicio Cloud y al interfaz gráfico de Architrave. Es además digna de agradecer la inestimable ayuda de David Guerrero en el diseño de la portada. No quisiera olvidarme del resto de integrantes del GRyCAP que trabajan día a día por el bien común del grupo.

Doy también las gracias a Agustín Pérez y a Fernando Gómez, compañeros incansables desde hace muchos años en el desarrollo y distribución de Architrave, por tanto esfuerzo desempeñado y tantas horas de trabajo dedicado. Mis agradecimientos además a Pedro Ruiz, Javier Ibáñez, Jesús Peinado, Andrés Terrasa, Ismael García, Joaquín Serralta, José Emilio Patiño y Enrique Arias por sus constantes ánimos y sus mejores deseos, cada vez que nos encontrábamos, para continuar adelante y que esta tesis llegara a buen puerto.

Dejo aquí también recogido el agradecimiento a mis padres y a mi hermano, las personas que siempre me han brindado su apoyo y lo mejor de sí mismas. Sin ellos no hubiera llegado hasta aquí. En especial, este agradecimiento es a mi padre, a quien dedico esta tesis, quien el destino me arrebató hace ya demasiados años y me privó de compartir lo mejor de nuestras vidas. Va por tí, allá donde estés, porque siempre estuviste orgulloso de mí.

Y por último, un millón de gracias a María José, el amor de mi vida, por su apoyo incondicional para no desfallecer, por el tiempo que hemos dejado de estar juntos y por darme a mis hijos, Noelia e Iván, esos dos bichejos que siempre me proporcionan, con sus risas y alegrías, los mejores momentos de cada día.

Gracias a todos.

José Miguel Alonso Ábalos

Valencia, 28 de febrero de 2016

Índice general

Índice de tablas

Índice de figuras

Índice de algoritmos

Capítulo 1

Introducción

Este capítulo incluye la motivación del desarrollo de la presente tesis doctoral, el marco de los proyectos de I+D+i bajo los cuales se ha llevado a cabo y los principales objetivos y aportaciones a alcanzar. Finalmente, se indican los capítulos bajo los cuales está estructurada, detallando el contenido de cada uno de ellos.

1.1. Introducción

El análisis estructural es el proceso que determina la respuesta de un edificio o una obra de ingeniería civil ante un conjunto conocido de cargas o acciones externas. Dicha respuesta, estática en caso de que las cargas no varíen con el tiempo, o dinámica en caso de que sí lo hagan, es habitualmente medida estableciendo previamente la definición de las propiedades estructurales y de las acciones aplicadas sobre la estructura, obteniendo como resultado los desplazamientos, los esfuerzos y las tensiones en múltiples puntos pertenecientes a los elementos que la forman.

El sujeto del análisis no es en sí la propia estructura, sino un modelo de la misma cuya definición depende del tipo de estructura analizada, y pretende no sólo proporcionar una descripción realista de su comportamiento, sino también

desarrollar una serie de relaciones entre las acciones y la respuesta que describe el modelo matemático del problema.

Las características físicas de la estructura a tener en cuenta en la definición de dicho modelo matemático son su rigidez, en el caso de un análisis estático, y la masa y el amortiguamiento, en el caso de un análisis dinámico.

Si bien es cierto que un análisis completo de la estructura supondría determinar la respuesta en el infinito número de puntos que la configuran, así como en un número infinito de instantes (siempre que se trate de un análisis dinámico a lo largo del tiempo), también lo es que ello impediría dar una solución numérica al problema a resolver. Con el objetivo de que dicha solución sí pueda obtenerse, los modelos estructurales concentran en una serie de puntos denominados *nudos*, y a través de una operación denominada *discretización espacial*, las características másicas, elásticas y disipadoras de la estructura, calculando en ellos mismos la respuesta del sistema. Además, definimos los *grados de libertad* de una estructura como aquellos desplazamientos que determinan su posición deformada, siendo su identificación una operación relevante en los resultados del análisis estático y dinámico. Es a la vez preciso, en caso de un análisis dinámico en el tiempo, realizar otra operación denominada *discretización temporal*, al objeto de obtener la respuesta en un número finito de instantes de tiempo.

El análisis estructural juega un papel clave durante la fase preliminar del diseño de un edificio u obra de ingeniería civil, momento en el cual deben considerarse un conjunto de diferentes alternativas (ver figura 1.1) [1]. Cada una de estas distintas variantes supone una consideración de las cargas o acciones que tendrá que soportar la estructura, no sólo en su fase de vida útil, sino también durante su fase de construcción, junto con un análisis estructural necesario para determinar los desplazamientos, esfuerzos y tensiones que van a tener lugar sobre la misma. A menudo, estos análisis preliminares se llevan a cabo utilizando modelos simplificados, con el objetivo de minimizar el tiempo y el esfuerzo empleado en dicha fase.

Aunque a veces todas estas soluciones constructivas son rechazadas, debiendo desarrollarse un nuevo conjunto de diseños iniciales, la situación más común es que una de ellas sea la elegida, continuando con la etapa final del diseño estructural, en

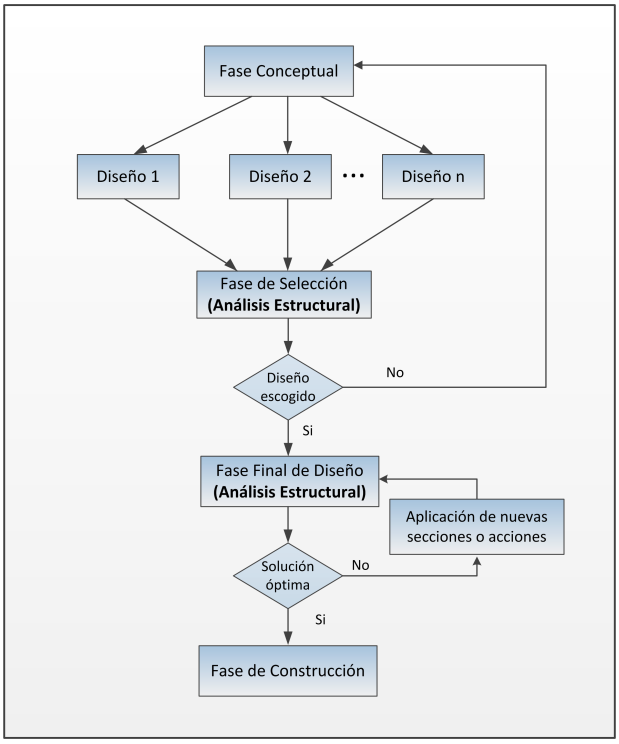


Figura 1.1: Diagrama de flujo describiendo el proceso de diseño estructural.

la cual se desarrolla un proceso de prueba y error a fin de obtener, en el tiempo más breve posible, la solución estructural que siendo económicamente rentable para el constructor sea a la vez funcional y segura. A modo de ejemplo, en dicho proceso de prueba y error se realizan modificaciones en la geometría (asignando diferentes dimensiones, secciones, o materiales a los elementos estructurales, o variando el mallado por elementos finitos, etc.), se analiza la estructura completa y se interpretan los resultados obtenidos. Ahora bien, por motivos de seguridad, el análisis estructural requerido en esta etapa debe ser llevado a cabo con una gran precisión, eliminado gran parte de las simplificaciones tenidas en cuenta en la fase preliminar. Es por ello necesario realizar un análisis espacial realista en 3D, con 6 grados de libertad por nudo y donde no se elimine ninguno de ellos. Resulta obvio que a menos que el modelo empleado sea lo suficientemente correcto, los resultados estructurales obtenidos serán de un valor más que cuestionable.

Sin embargo, el diseño de una estructura de tamaño considerable (máxime si hablamos de un análisis dinámico no lineal) puede superar los recursos computacionales de los computadores disponibles por los usuarios, o conllevar un tiempo

de análisis equivalente a varias horas, e incluso días, mediante los programas de simulación existentes en el mercado.

Por otro lado, es también habitual el simular una misma solución estructural bajo la influencia de diferentes cargas dinámicas. Así, por ejemplo, la Normativa de Construcción Sismorresistente Española (NCSE-02) [2] exige que un edificio se analice, como mínimo, con cinco terremotos diferentes y representativos. Obviamente, esto implica que el número de simulaciones a realizar incremente notablemente la demanda computacional del problema.

De ese modo, los requerimientos de reducción en el tiempo empleado en el cálculo estructural vienen impuestos, por un lado, por la necesidad de tener información completa de la estructura en la fase de diseño preliminar y, por otro, por la necesidad de evaluar en el menor tiempo posible un elevado número de soluciones estructurales, a fin de obtener la alternativa más eficiente en términos constructivos, económicos y de seguridad. Según [3], si bien es cierto que las fases de diseño citadas representan únicamente un 10 % en valor económico, durante esta fase se toman decisiones que afectan al 90 % de las etapas restantes en el proceso de construcción de un edificio.

Finalmente, conviene también destacar que las herramientas de análisis estructural forman parte de aquellas otras dedicadas a procesos de optimización, los cuales pretenden obtener de manera automática la mejor alternativa posible de acuerdo a una función objetivo, como puede ser minimizar el coste de la estructura y la cantidad de material (hormigón, acero, etc.) empleado, garantizando siempre su viabilidad y su cumplimiento con la normativa vigente. Dichas herramientas de optimización invocan a la aplicación encargada del cálculo de la estructura para la ingente cantidad de alternativas estructurales que analizan, siempre en base al algoritmo que guía el proceso de optimización. Si dichas aplicaciones de cálculo no son eficientes y no ofrecen un tiempo de respuesta lo más reducido posible, el uso de las herramientas de optimización no será viable o se verá abocado a trabajar o bien con pequeñas estructuras o bien con pocas variables y con un espacio de búsqueda limitado, pudiendo evaluar tan sólo un pequeño número de soluciones estructurales.

1.2. Motivación

Las pérdidas económicas y de vidas humanas que se producen como consecuencia de los terremotos están relacionadas, en multitud de ocasiones, con un comportamiento deficiente de las estructuras [4, 5]. Son muy numerosos los casos de edificios que han sufrido daños importantes durante terremotos recientes, aun estando construidos de acuerdo a las normativas de diseño y construcción vigentes.

A pesar de ser indudable el avance en los últimos años de los ordenadores, de la investigación en los métodos de cálculo numérico, de las tecnologías de computación avanzada y del diseño de estructuras resistentes, no resulta tan evidente el avance práctico o implementación de dicha investigación teórica, haciendo especial referencia a las normativas de diseño, las cuales incluyen siempre procedimientos de cálculo simplificados, cuya presencia es hoy en día obsoleta, recurriendo a sencillas expresiones matemáticas o disminuyendo en gran medida el número de nudos considerados y el número de grados de libertad asociados a fin de reducir los requerimientos de cálculo y de memoria. Hay que decir que aunque estas simplificaciones han demostrado ser perfectamente válidas para estructuras sencillas y simétricas, no lo son para edificios asimétricos o que posean una notable complejidad.

Es importante destacar que dichos cálculos simplificados se consideran innecesarios dentro del más amplio nivel de conocimiento de gran parte de los calculistas de estructuras. Así por ejemplo, el desarrollo de simplificaciones para la determinación de las fuerzas sísmicas equivalentes tuvo lugar entre los años cuarenta y setenta, cuando la carencia de ordenadores eficientes hacía impracticable el análisis estático o dinámico detallado y realista de la estructura.

Sin embargo, la propia naturaleza intrínseca espacial o tridimensional de los edificios y su constante incremento en complejidad tanto en planta como en altura conlleva el realizar un análisis riguroso, simulando la estructura conjuntamente y reflejando el comportamiento estático y dinámico de la misma de manera más adecuada que el llevado a cabo mediante análisis planos y simplificados, donde cada uno de los elementos planos de la estructura se calculaba, en muchos programas comerciales hasta hace unos años, de manera independiente. Es bien

conocido que los resultados del cálculo estructural son claramente dependientes del modelo empleado, lo que indica la necesidad de emplear modelos analíticos en 3D que sean adecuados y realistas, lo cual repercutiría en edificios más seguros y económicos. La necesidad de emplear dichos modelos aparece claramente expuesta en [6]. Así, cada nudo de la estructura posee, en un análisis en 3D, seis grados de libertad, con capacidad de desplazarse según tres direcciones ortogonales entre sí y de describir tres giros alrededor de las mismas. Por el contrario, el número de grados de libertad por nudo se reduciría a tres en un análisis 2D, con dos desplazamientos longitudinales con respecto a dos ejes perpendiculares y un giro con respecto a uno tercero ortogonal a los mismos.

No obstante, el incremento de complejidad reflejado en el gran número de grados de libertad totales resultantes en un edificio de tamaño singular supone que el análisis 3D riguroso y realista, con seis grados de libertad por nudo, de lugar a problemas algebraicos estáticos y dinámicos con una dimensión equivalente a varias centenas de miles, e incluso varios millones, de grados de libertad.

Si a esto añadimos el que métodos de cálculo dinámico como el de la integración directa suponen la integración de las ecuaciones del movimiento, y por tanto la resolución de dichos problemas algebraicos, en un número elevado de pasos de tiempo, podemos afirmar que el análisis dinámico en 3D de una estructura de gran dimensión da lugar a unos tiempos de cálculo y unos requerimientos de memoria y de disco que tradicionalmente han sido demasiado elevados para un PC, lo que llevó a convertirlos en inabordables. Algo similar, aunque en menor medida, podríamos decir de las técnicas basadas en un análisis modal. Aunque unos pocos modos de vibración pueden ser suficientes para determinar con precisión la respuesta máxima del sistema, el alto número de grados de libertad que conlleva un análisis tridimensional da lugar a un problema con una dimensión que se traduce en unos elevados tiempos de cómputo y unos requerimientos de memoria claramente exigentes.

Es por ello, como decíamos en un principio, que habitualmente se ha recurrido a simplificaciones que reduzcan en gran medida la complejidad computacional del problema, proporcionando sin embargo unos resultados cuya exactitud no puede, en muchos casos, considerarse como satisfactoria.

A modo de ejemplo, numerosos programas comerciales ampliamente utilizados incluían hasta hace pocos años un análisis estático en 2D, donde cada forjado de la estructura se analizaba de manera independiente al resto, como mencionábamos anteriormente. Incluso, el análisis dinámico iba más allá, donde dichas aplicaciones software analizaban la estructura en conjunto, pero aplicaban técnicas simplificadas como el *modelo del edificio a cortante*, en la cual la masa de cada forjado estructural quedaba agrupada en un solo nudo, contemplando tres grados de libertad a lo sumo para el mismo y reduciendo al mínimo esfuerzo los requerimientos computacionales del problema. Dichos programas no incluían entre sus opciones de cálculo métodos de integración directa o de análisis por superposición modal, con la imposibilidad de obtener la respuesta de la estructura a lo largo del tiempo, proporcionando únicamente, gracias al análisis modal y a diferentes métodos estadísticos, la respuesta máxima de la estructura.

Hay que decir que, en gran medida, uno de los mayores inconvenientes asociados al cálculo estructural viene dado por la necesidad de resolver sistemas de ecuaciones lineales dispersos y de gran dimensión (una vez en el caso del cálculo estático lineal o para cada paso de tiempo en el caso del cálculo dinámico mediante métodos de integración directa) o resolver un problema de valores propios generalizado (en el caso del cálculo dinámico mediante técnicas de análisis modal), donde el alto coste computacional y de memoria de los algoritmos utilizados en los mismos hace recomendable el uso de técnicas de computación avanzada.

La justificación en el uso de estas técnicas es considerablemente aún mayor en el caso de un análisis estructural no lineal, donde deben resolverse, para cada incremento de carga (en el caso de un análisis estático) o para cada paso de tiempo (en el caso de un análisis dinámico) un sistema de ecuaciones no lineales, lo que implica la resolución de múltiples sistemas de ecuaciones lineales hasta alcanzar la convergencia.

Afortunadamente, la situación actual es completamente diferente. Si bien es cierto que un análisis dinámico en 3D a lo largo del tiempo de grandes estructuras de edificación mediante métodos de integración directa o mediante técnicas de análisis modal habría sido imposible de llevarse a cabo hace unos cuantos años, hoy en día es perfectamente abordable y capaz de realizarse con unos tiempos de computación que en absoluto pueden tildarse de excesivos. Esto es así gracias

a la posibilidad de emplear Computación de Altas Prestaciones y a la eficiencia de los métodos numéricos actuales, desarrollando y ejecutando herramientas de simulación en arquitecturas computacionales paralelas. Como es evidente, esto supondría el reducir notablemente los tiempos de cómputo, así como el poder abordar estructuras de mayor dimensión de un modo mucho más realista.

Además, debemos tener en cuenta que, a día de hoy, cualquiera de los ordenadores portátiles o de sobremesa disponibles en el mercado es ya, en sí mismo, una máquina paralela, compuesta como mínimo por dos o cuatro elementos de procesamiento. Cabe además la posibilidad de interconectar dichos ordenadores en los denominados clusters de PCs, a fin de dar lugar a una plataforma computacional paralela de bajo coste que destaca por su alta disponibilidad y su excelente relación precio prestaciones.

Sin embargo, puede afirmarse que la amplia mayoría de los programas comerciales de los que disponen los calculistas estructurales no hacen uso ni de la Computación Paralela ni de los últimos avances en el campo de la Computación Numérica, lo que les permitiría abordar de forma eficiente las diferentes simulaciones a realizar.

Conviene también aclarar que los estudios de arquitectura, las empresas de ingeniería dedicadas al cálculo estructural e incluso muchos centro de investigación no disponen de recursos computacionales apropiados, más allá de los equipos de trabajo personales, que les permitan ejecutar aplicaciones paralelas, en caso de disponer de las mismas. Esto justifica la oportunidad de emplear otro tipo de tecnologías avanzadas, como son la computación Grid o la computación Cloud, esta última más conocida como computación en la nube.

Las tecnologías Grid facilitan la compartición y utilización de distintos tipos de recursos (hardware y software) disponibles en la red, a través de las e-infraestructuras propias de una organización o de aquellas que se han desplegando en diferentes proyectos de ámbito regional, nacional o europeo. De esta manera, es posible la creación de una organización virtual científico-tecnológica, en la cual no todos los miembros necesitan adquirir los equipos o licencias de aplicaciones de cálculo en propiedad, sino que basta con realizar acuerdos para el uso de los mismos.

Del mismo modo, la computación en la nube nos permite utilizar recursos software y hardware, generalmente virtualizados, bajo un modelo de pago por uso, con un aprovisionado dinámico y a demanda de los recursos.

Precisamente el proceso de diseño de estructuras, donde deben calcularse múltiples soluciones estructurales o una misma solución bajo la acción de diferentes terremotos, puede beneficiarse sustancialmente al ejecutar todas ellas, o parte de las mismas, simultáneamente en una infraestructura Grid o Cloud de alta productividad, pudiendo utilizar, incluso, una herramienta de cálculo basada en Computación de Altas Prestaciones y reduciendo así, en gran medida, el tiempo dedicado al diseño estructural. El hecho de incrementar el número de soluciones estructurales analizadas por unidad de tiempo permitiría además aumentar el número de alternativas estructurales evaluables, lo que repercutiría en una mejora de la solución final alcanzada, en términos económicos y de seguridad.

1.3. Marco de la tesis

La tesis doctoral que aquí se describe se enmarca dentro de las líneas de investigación del Grupo de Grid y Computación de Altas Prestaciones (GRyCAP) de la Universitat Politècnica de València (UPV), liderado por Ignacio Blanquer Espert. Dicho grupo inició sus actividades en 1986, bajo la tutela de Vicente Hernández García, con el nombre de Grupo de Computación Paralela.

Actualmente, el grupo está compuesto por más de 20 miembros, los cuales, además de tareas docentes en el Departamento de Sistemas Informáticos y Computación (DSIC) de la UPV, desarrollan su labor investigadora en el marco del Instituto de Instrumentación para Imagen Molecular (I3M).

Principalmente, el GRyCAP ha orientado su línea de investigación básica en la dirección de la Computación de Altas Prestaciones, las tecnologías Grid y las tecnologías Cloud, así como a la aplicación de las mismas en el desarrollo de software científico orientado a la simulación en ingeniería, la computación numérica, la informática gráfica o las tecnologías de la salud.

Todo el trabajo desarrollado en esta tesis doctoral, así como su divulgación,

se inició bajo el paraguas de una beca FPI y se ha desarrollado y financiado en el marco de los convenios con empresas y proyectos de investigación y de difusión europeos, nacionales o regionales que a continuación se citan:

1. Introducing High Performance Computing in Small and Medium Size Enterprises (HIPERCOSME)(20059). CEE-ESPRIT. Duración: 01/01/1996 - 31/12/1997.

2. Algoritmos Paralelos Para el Cálculo de Valores Propios en Matrices Dispersas y Estructuradas (CICYT TIC96-1062-C03-C01). Formación de Personal Investigador. Comisión Interministerial de Ciencia y Tecnología. Ministerio de Educación. Duración: 01/08/1996 - 01/08/1999.

3. HIPERCOSME Technology Transfer Node (HIPERTTN) (ESPRIT 20003) junto con los proyectos o estudios de validación llamados Porto Awareness Campaign (PADEM), Valencia Assessments (VASSES), Euskadi Assessments (EASSES), Greece Assessments (GASSES) y Midi-Pyréenes Assessments (MASSES). Duración: 01/04/1997- 31/03/2000.

4. Promoción y Transferencia de Tecnología Basada en Computación de Altas Prestaciones a la Pequeña y Mediana Empresa (CTI 98-1370-E). Comisión Interministerial de Ciencia y Tecnología. Ministerio de Educación. Acción Especial. Duración: 15/06/1998 - 08/09/1999.

5. Introducción de TIC en las PYMES Valencianas (IMTEPA/2000/114). Instituto Valenciano de Competitividad Empresarial. Programa de Actuaciones de Difusión y Transferencia de Tecnología. Duración 09/01/2000 - 09/11/2000.

6. Dinámica de Tipologías Estructurales con Comportamiento Lineal (DINTEL). Universidad Politécnica de Valencia. Vicerrectorado de Investigación, Desarrollo e Innovación. Duración: 27/10/00 - 27/10/02.

7. Cálculo Estructural de Muros Cortina en 3D (Calma3D). Preference, S.L. Duración 01/02/2002 - 01/01/2003.

8. Investigación, Desarrollo e Innovación de Servicios GRID: Aplicación a Modelos Cliente-Servidor, Colaborativos y de Alta Productividad (GRID-IT) (TIC2003-01318). Plan Nacional de Investigación Científica, Desarrollo en Innovación Tecnológica. Ministerio de Ciencia y Tecnología. Duración 01/11/2003 - 01/12/2006.

9. Desarrollo de una Aplicación Grid de Altas Prestaciones para el Análisis Dinámico y la Visualización en 3D de Estructuras de Edificación (GRID4-BUILD) (GV04B/424). 2004-2005. Consellería de Cultura, Educació i Esport, Generalitat Valenciana. Duración 01/01/2004 - 01/01/2006.

10. Colaboración entre los grupos CID del Departamento de Mecánica de los Medios Continuos y Teoría de Estructuras (DMMCTE) de la UPV y el GRyCAP para el desarrollo y la comercialización de Architrave, una aplicación orientada al diseño y al análisis estático y dinámico de elementos finitos y la visualización en 3D de estructuras. Octubre 2007-actualidad.

11. Ampliación y Mejoras en la Librería CALMA3D Para el Cálculo Estructural de Muros Cortina en 3D (CALMA3D-V2). Preference, S.L. Duración: 24/04/2008 - 24/04/2009.

12. Licencia del Derecho de Uso del Sistema Informático Architrave 2011. Duración: 23/02/2011 - Actualidad.

13. Virtual Multidisciplinary Environments Using Cloud Infrastructures (VENUS-C) (261565). Comisión de la Comunidad Europea. Duración: 01/06/2010 - 01/06/2012.

1.4. Objetivos y aportaciones

El objetivo de esta tesis consiste en la aplicación de técnicas computacionales avanzadas, como son la Computación de Altas Prestaciones y las Tecnologías Grid y Cloud, al cálculo lineal estático y dinámico de estructuras de edificación, mediante elementos finitos.

Gracias a la eficiencia de los métodos numéricos actuales y a la disponibilidad de diferentes paradigmas de Computación Paralela (basados en memoria compartida y distribuida), la aplicación de la Computación de Altas Prestaciones permitirá que el análisis estructural se realice de manera colaborativa entre un conjunto de procesos e hilos de ejecución. Esto hará posible que dicho análisis 3D pueda llevarse a a cabo de manera rigurosa y realista, considerando todos los nudos de la estructura e incluyendo seis grados de libertad por cada uno de ellos, en unos tiempos de cómputo razonablemente reducidos. De este modo, será posible simular, de una manera más segura, edificios y estructuras de ingeniería civil con una dimensión y complejidad mucho mayor de lo que venía haciéndose hasta el momento.

En lo que a la aplicación de la Computación Paralela se refiere, e independientemente del tipo de análisis y del método aplicado, se implementarán algoritmos secuenciales y paralelos encargados de:

1. Generar las matrices de rigidez, masa y amortiguamiento representativas de la estructura.
2. Formar los vectores con las cargas externas, estáticas y dinámicas, aplicadas sobre la misma.
3. Calcular los desplazamientos en los nudos (más las velocidades y aceleraciones en el caso del cálculo dinámico), bien sea resolviendo los sistemas de ecuaciones lineales o el problema de valores propios generalizado.
4. Obtener las solicitaciones en los extremos de las barras y las reacciones en los apoyos.
5. Calcular los esfuerzos y las deformaciones en los puntos intermedios de las barras.
6. Determinar los esfuerzos y las tensiones en los nudos a los que confluye algún elemento finito.

En el caso del análisis estático, los algoritmos secuenciales y paralelos a implementar estarán basados en el *Método de la Rigidez*. En el caso del análisis

dinámico [7, 8], dicho análisis se llevará a cabo mediante diferentes metodologías. En primer lugar, calculando la respuesta máxima de la estructura mediante el Análisis Modal Espectral, empleando diferentes métodos de combinación modal (Suma Absoluta, Raíz Cuadrada de la Suma de los Cuadrados, Combinación Cuadrática Completa, etc) y tratando de determinar el signo de los resultados. En segundo lugar, obteniendo la respuesta de la estructura a lo largo del tiempo, bien sea mediante la implementación secuencial y paralela de distintos métodos de Integración Directa (tales como Newmark [9], Wilson-θ [10], etc.), o por medio de las técnicas de Superposición Modal, las cuales, a su vez, incluyen el empleo de técnicas de integración para resolver las ecuaciones diferenciales desacopladas.

Los algoritmos de cálculo se implementarán en el lenguaje C y sobre sistemas paralelos de bajo coste, como son los cluster de PCs. Además, se aplicará conjuntamente el paradigma de memoria compartida, mediante el entorno de programación paralela que proporciona OpenMP [11], y el de memoria distribuida, donde la comunicación entre los procesos será llevada a cabo mediante las rutinas de paso de mensajes implementadas en la librería MPI (Message Passing Interface) [12]. De este modo, la portabilidad de los códigos desarrollados a una amplia variedad de plataformas paralelas y sistemas operativos (como Linux o Windows) quedará garantizada.

A fin de resolver de la manera más eficiente posible los principales problemas computacionales resultantes se estudiarán y se emplearán diferentes librerías numéricas paralelas de dominio público, las cuales proporcionarán robustez, eficiencia y portabilidad a la aplicación a desarrollar.

Así, la aplicación desarrollada estará basada, en su totalidad, en una de las herramientas de desarrollo de computación científica más conocidas, a día de hoy, como es PETSc (*Portable, Extensible Toolkit for Scientific Computation*) [13], además de resolver, en secuencial y en paralelo, los sistemas de ecuaciones lineales dispersos de gran dimensión por medio de los métodos iterativos basados en subespacios de Krylov, que dicha librería proporciona, o por medio de métodos directos, basados en técnicas multifrontales, presentes en otras librerías como MUMPS (*MUltifrontal Massive Parallel Solver*) [14] u otras, directamente invocables desde el propio PETSc.

Por otro lado, se empleará la librería SLEPc (*Scalable Library for Eigenvalue Problem Computations*) [15] para resolver, en secuencial y en paralelo, el problema de valores propios generalizado correspondiente a un análisis modal, bien sea mediante los propios métodos de cálculo presentes en SLEPc o por medio de los existentes en otras librerías numéricas de las cuales SLEPc actúa como pasarela, entre las que cabe citar a ARPACK (*ARnoldi PACKage*) [16]. Mientras que SLEPc proporciona métodos como Krylov-Schur, o Lanczos o Arnoldi con reinicio explícito, ARPACK ofrece a Arnoldi con reinicio implícito como método de resolución.

Con el objetivo de que el trabajo desarrollado resulte accesible a un abanico, lo más amplio posible, de arquitectos e ingenieros, tanto desde un ámbito docente e investigador como comercial, se pretende desarrollar y desplegar servicios Grid y Cloud de cálculo de estructuras fiables, multiusuario y de alta productividad, que incorporen a la aplicación paralela previamente implementada.

De este modo, el calculista dispondrá en su ordenador de una aplicación gráfica desde la cual puede llevar a cabo las tareas de entrada de datos de la estructura y visualización de resultados. Así, desde dicha aplicación gráfica y con un solo clic de ratón, el usuario envía vía Internet las estructuras a analizar, de manera que dichos servicios Grid y Cloud atiendan sus peticiones y le proporcionen los resultados. Estos servicios a desarrollar se encargarán de asignar las diferentes peticiones de cálculo a un conjunto remoto de recursos computacionales, de manera totalmente transparente para el usuario.

Los servicios a implementar emplearán un sistema de acceso restringido, de manera que sólo aquellos usuarios autorizados y autenticados podrán acceder a los mismos, proporcionando la debida privacidad de datos (tanto en su transferencia como en su almacenamiento), así como diferentes niveles de tolerancia a fallos, ante caídas del cliente gráfico, del servicio, del recurso computacional en el que se está ejecutando la simulación o de la red de comunicaciones, garantizando en todo momento que cualquier petición de cálculo enviada por un cliente será atendida satisfactoriamente.

Para el desarrollo del servicio Grid de cálculo estructural, se empleará Globus Toolkit 4 [17] como middleware que ofrezca los servicios Grid básicos, además

de GMarte (Grid Middleware to Abstract Remote Task Execution) [18] como software Grid de más alto nivel capaz de gestionar la planificación eficiente de tareas a los recursos computacionales, el envío de datos y la recogida de resultados entre el recurso y el servicio, así como la tolerancia a fallos ante caídas del recurso computacional.

En lo que respecta al servicio Cloud, estará basado en Microsoft Azure [19] y deberá ajustar dinámicamente la cantidad de recursos computacionales utilizados dependiendo de la cantidad de simulaciones que lleguen al sistema, garantizando además que las simulaciones estáticas no se vean penalizadas por el alto coste de las simulaciones dinámicas.

Estos servicios Grid y Cloud de alta productividad facilitarán, en primer lugar, el acceso a una herramienta paralela por parte de un usuario no experimentado en dichas tecnologías, potenciando el uso de todas ellas en el campo de la ingeniería estructural. En segundo lugar, el usuario tendrá a su disposición una infraestructura de computación distribuida formada por un conjunto de computadores de los que no dispone, pudiendo calcular, concurrentemente, un elevado número de estructuras, reduciendo así los tiempos dedicados al análisis y al diseño estructural. De este modo, el usuario tiene acceso a la última versión de un software especializado, así como a una costosa infraestructura hardware, sin necesidad de adquirir ninguno de ellos en propiedad y sin gastos derivados de mantenimiento, pagando, si procede, exclusivamente por el uso de los mismos.

En consecuencia, los servicios a implementar y desplegar podrán tener los siguientes usos:

1. Un uso público, siendo accesible gratuitamente vía Internet y de manera colaborativa por miembros de una comunidad universitaria e investigadora.

2. Un uso privado, dentro de una empresa o institución. A dicho servicio sólo tendrían acceso los miembros de las mismas, empleando para el cálculo los computadores de los que la empresa dispone e incrementando la tasa de uso de los mismos.

3. Un modelo de negocio basado en el pago por servicio, donde una entidad externa ofrece un servicio de cálculo estructural y pone a disposición de

los usuarios un software avanzado junto con un conjunto de computadores que ejecutan las simulaciones y almacenan los resultados. De este modo, los clientes que envíen sus peticiones de cálculo no necesitarán disponer de una licencia de uso propio de la herramienta paralela, de la cual quizá pudieran no llegan a sacar provecho si no la emplean frecuentemente, sino que pagarán únicamente por el uso puntual de la misma y de la infraestructura computacional utilizada.

Independientemente del modelo de uso, el sistema será beneficioso para todos aquellos usuarios que necesiten analizar una cantidad elevada de estructuras, ya que gran parte las mismas se simularán al mismo tiempo entre los diferentes recursos computacionales que conformen la infraestructura Grid o Cloud distribuida, incrementando así su productividad y volumen de negocio.

Por último, es importante destacar que gran parte de los desarrollos de esta tesis, pese a estar orientados al cálculo estructural de edificios y obra civil, pueden ser fácilmente aplicados a otros ámbitos, como la automoción, la resistencia de materiales, la ingeniería espacial y la aeronáutica.

1.5. Estructura de la tesis

Esta tesis está estructurada en los siguientes capítulos. En primer lugar, el capítulo 2 está dedicado al análisis estructural y los principales tipos de análisis. El capítulo 3 recoge toda la formulación necesaria para implementar un análisis estático lineal mediante barras y elementos finitos, como ha ocurrido en esta tesis doctoral. De igual manera, el capítulo 4 describe la formulación y los diferentes métodos que se ha tenido en consideración para implementar un análisis dinámico lineal mediante métodos de integración directa, superposición modal y por análisis modal espectral, de estructuras compuestas por barras y elementos finitos. El capítulo 5 está dedicado a la Computación Científica y en él se describen la Computación de Alta Prestaciones, los diferentes tipos de supercomputadores que existen hoy en día, además de los distintos paradigmas de programación de dichos supercomputadores y el software disponible para programar aplicaciones

que obtengan las mejores prestaciones de los mismos. En el capítulo 6 detallaremos en qué consisten la computación Grid y la computación Cloud, o en la nube, describiendo las peculiaridades de cada una de ellas y poniendo especial énfasis en el software y en las plataformas que han permitido desarrollar y desplegar los diferentes servicios Grid y Cloud implementados, como son Globus Toolkit, Microsoft Azure y COMPSs. El capítulo 7 describe la implementación del Simulador Estructural basado en técnicas de Computación de Altas Prestaciones para los diferentes tipos de análisis que realiza. A continuación, el capítulo 8 está dedicado a los servicios Grid y Cloud de análisis estructural implementados, incluyendo su arquitectura y los diferentes elementos que los componen, como el Simulador Estructural descrito en el capítulo previo. En el capítulo 9 detallaremos los resultados experimentales obtenidos, a nivel del Simulador Estructural y a nivel de los servicios Grid y Cloud, comparando las prestaciones obtenidas por parte de los mismos con respecto a una aproximación tradicional. Finalmente, el capítulo 10 recoge las conclusiones, los resultados obtenidos en el marco de esta tesis doctoral y los trabajos futuros.

Capítulo 2

El Análisis Estructural y el Sector de la Construcción

Tras una pequeña alusión a la importancia que tienen los edificios en la vida de las personas, la repercusión del sector de la construcción en la actividad económica de nuestro país y la actividad de dicho sector en actividades de I+D+i, este capítulo describe en qué consiste el análisis estructural y el tipo de análisis llevado a cabo en esta tesis doctoral. Se incluye también una breve descripción del Código Técnico de la Edificación, como marco normativo español por el que se regulan las exigencias que deben cumplir los edificios, entre otros muchos, a nivel seguridad estructural. Finalmente se enumeran los diferentes tipos de cargas externas que pueden actuar sobre las estructuras, cuya naturaleza determina el tipo de análisis a realizar.

2.1. Ciudades y edificios

Más del 80 % de los ciudadanos europeos viven en áreas urbanas, donde además de vivir, trabajar o estudiar, tienen acceso a bienes, servicios y actividades lúdicas y culturales, estableciendo sus relaciones sociales. Para proporcionar estas

funciones, las áreas urbanas presentan diferentes elementos estáticos, tales como edificios, infraestructuras de ingeniería civil o espacios verdes, así como elementos dinámicos, como el transporte, la energía, etc.

Como centros de bienestar económico, social y educacional, las ciudades sufren una incesante llegada de personas, bien sea desde áreas rurales próximas o por movimientos migratorios desde lejanos lugares, fenómeno que está llamado a ser cada vez más marcado y creciente en futuras décadas en Europa.

Los edificios y el entorno construido son los elementos principales que definen el entorno urbanístico. Son ellos los encargados de dar a una ciudad su carácter y su identidad, creando entre sus habitantes un sentimiento de arraigo hacia la misma, y haciendo de ella un lugar atractivo donde a la mayoría de la gente le gusta vivir y trabajar. La calidad urbanística y arquitectónica de los edificios tiene una fuerte influencia sobre la ciudad, máxime si tenemos en cuenta que los ciudadanos europeos pasan el 90 % de su tiempo dentro de los edificios, debiendo garantizar su salud, bienestar, seguridad y calidad de vida.

Entre los diferentes factores relevantes dentro del ciclo de vida de los edificios destaca la integración de los procesos de diseño, construcción y gestión durante el uso de los mismos, con clara alusión, entre otras, a la necesidad de disponer de herramientas flexibles y precisas de diseño, modelado analítico y simulación.

2.2. El sector de la construcción como motor económico

El sector de la construcción es el formado por aquel conjunto de empresas cuya actividad consiste en ejecutar directamente obras completas o partes de ellas, tanto de edificación como de ingeniería civil o industrial.

Sin lugar a dudas, la construcción es uno de los sectores productivos más importantes en la economía de los países modernos, convirtiéndose durante muchos años en uno de los principales motores de la economía española. Según datos del Instituto Nacional de Estadística y del Banco de España, el sector aportó al Producto Interior Bruto, a finales del tercer trimestre de 2006, un total del 18 %,

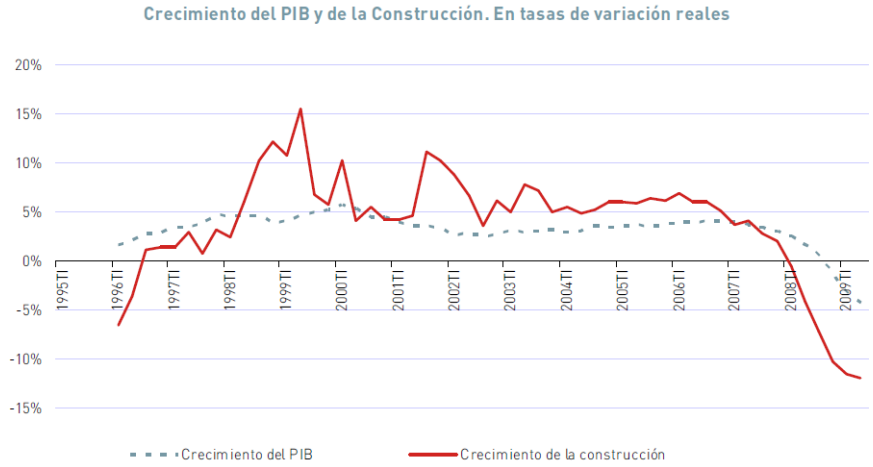


Figura 2.1: Comparativa de la evolución del PIB y del sector de la construcción entre los años 1996 y 2008.

mientras que en 1996 su peso era del 11.7 % (ver figura 2.1, cortesía de [20]).

Con un sector en alza, con una media de crecimiento del 6 % desde 1997, la superficie edificada aumentó de manera considerable. Así, en el año 2005 se alcanzó la cifra de más de 800.000 nuevas viviendas, cantidad récord de construcción en la historia de nuestro país. Dicha edificación de nuevas viviendas suponía el 35 % de la actividad económica del sector, aproximadamente. De este modo, la relevancia del sector en el empleo en España pasó de aportar un 10.3 % en 1991 al 21.9 % en 2005. No en vano, la construcción fue la causante del 24.4 % del crecimiento económico y nada menos que del 40.9 % del incremento del empleo. Con todo ello, el volumen de negocio de las empresas, incluidas todas aquellas dedicadas al cálculo de estructuras, fue muy elevado.

El exceso de oferta residencial, el precio de las viviendas, que llegó a alcanzar un incremento de hasta un 150 %, y el aumento del Euribor dieron lugar a que la demanda mostrara, desde finales de 2006, signos claros de debilidad. Si a esto unimos la crisis financiera internacional que empieza a manifestarse en la segunda mitad de 2007, el resultado es un proceso acelerado de caída abrupta de la actividad, arrastrando al resto de la economía y afectando seriamente al PIB y a las cifras de desempleo. Ello supuso la entrada, de nuestro país, en una durísima etapa de recesión y crisis económica que todavía sufrimos, la cual ha afectado con

especial crudeza a dicho sector de la construcción y, en consecuencia, a todos sus empleados y agentes relacionados.

2.3. El I+D+i en el sector de la construcción

Ya en el año 1997 se apuntaba que el sector de la construcción en general, y las empresas constructoras en particular, con la ayuda de universidades e institutos de investigación, debían prestar mayor atención al desarrollo de tecnologías propias o adaptadas de otros sectores industriales, tecnológicamente más desarrollados, para encontrar mejores soluciones a los problemas que se plantean en una obra [3]. Dicho documento destacaba además en su momento a la *construcción informatizada* como una necesidad tecnológica en el sector de la construcción, donde el uso de sistemas expertos y la simulación mediante herramientas informáticas resultaba de especial interés.

En España, al igual que en la mayoría de países industrializados, la inversión en I+D+i de las empresas de construcción ha estado tradicionalmente en unos niveles bajos, en relación a su facturación, con respecto a otras actividades industriales, y una amplia mayoría no han tenido costumbre de participar en programas nacionales o europeos de investigación [21]. A pesar de que las grandes empresas españolas cuentan con departamentos técnicos dedicados total o parcialmente a actividades de I+D+i, la dedicación de la construcción española ha estado siempre en un nivel inferior al europeo. Además, salvo las grandes empresas y muchas de las especializadas, el resto no han considerado a la innovación como un factor determinante.

Esta tendencia, unida a la crisis económica que estamos viviendo, implica que, en 2012, el gasto en I+D+i de las empresas de la construcción fuera solo de un 2.1 %, frente al total invertido por el sector español empresarial, tal y como muestra la figura 2.2, lo cual supone un porcentaje idéntico al de 2005.

Según esta misma fuente [22], el porcentaje de empresas innovadoras (dígase aquellas que han introducido en el mercado un nuevo producto o significativamente mejorado o han implantado un nuevo proceso o actividad a sus bienes y servicios) del sector de la construcción es del 6.71 %, frente al total de empresas

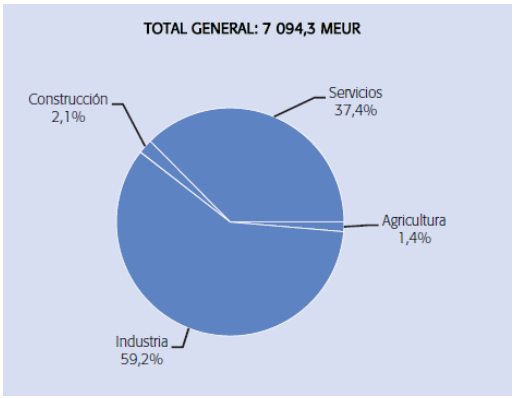


Figura 2.2: Porcentaje de gastos de las empresas españolas en I+D en 2012.

del sector, valor inferior al del año 2005, cuando era de un 20.4 % [23], de acuerdo a lo mostrado en la figura 2.3. Estos valores son muy inferiores a otros sectores donde los resultados de esta tesis pueden ser aplicados, como es el caso de la construcción aeronáutica y espacial, en el cual un 47.50 % de las empresas del sector, en el año 2012, dicen ser innovadoras.

Aun con todo ello, es necesario resaltar la importancia que en los últimos años han tenido las tecnologías de la información y las comunicaciones en la innovación y en la optimización de muchos procesos en el sector de la construcción. Es evidente el esfuerzo de adaptación de los procesos de diseño y control a las nuevas tecnologías, tanto por parte de las grandes empresas como por los profesionales que trabajan en solitario, donde la informatización de los procedimientos de cálculo, diseño, cartografía y topografía ha permitido incrementos de productividad más que significativos.

2.4. El análisis estructural

Podemos definir el análisis estructural [1, 24] como el proceso que determina la respuesta de un edificio o una obra de ingeniería civil ante un conjunto conocido de cargas o acciones externas, con el objetivo de asegurar que cumple con las condiciones adecuadas de seguridad, funcionalidad y durabilidad, a fin de rendir el servicio para el cual ha sido proyectado.

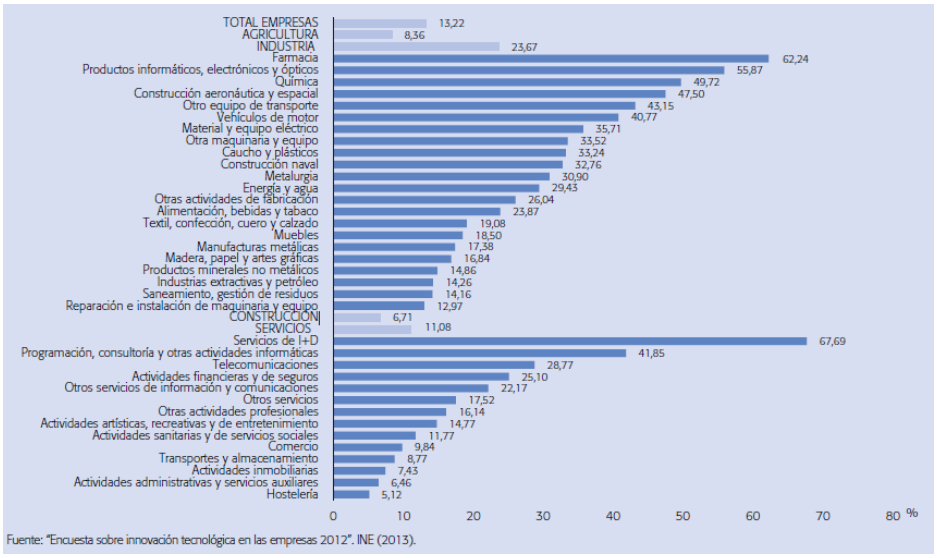
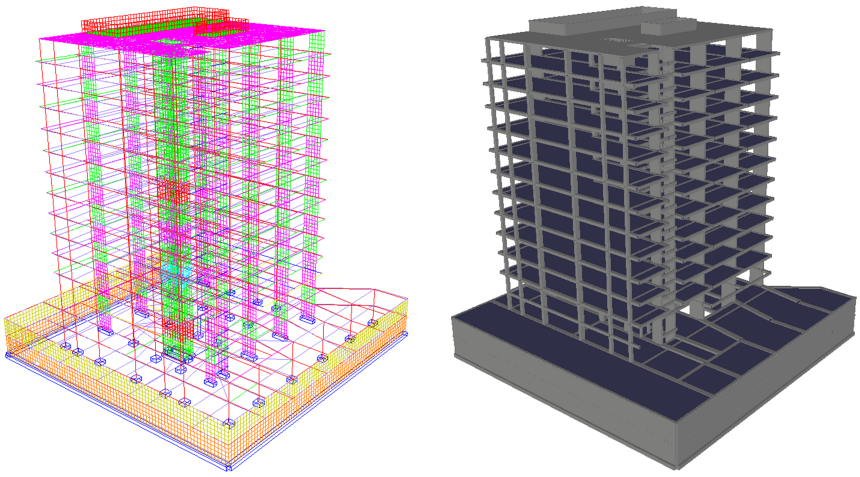


Figura 2.3: Porcentaje de empresas innovadoras en relación al total de las empresas del sector en 2012.

El tipo de metodología de cálculo a seguir en esta tesis es la basada en la teoría de los elementos finitos. Ello supone que en un análisis estructural no se emplea en sí la propia estructura continua, sino un modelo matemático de la misma correspondiente a su discretización, compuesto por elementos estructurales de tamaño finito, como es el caso de la figura 2.4, siendo conocidas las propiedades elásticas y de inercia de estos últimos. Dichos elementos estructurales pueden ser o bien barras (elementos unidimensionales donde la anchura y el espesor son pequeños en comparación con su longitud), elementos finitos en dos dimensiones como triángulos o cuadriláteros (en los cuales el espesor es pequeño en comparación con su superficie) y elementos finitos en tres dimensiones, como prismas triangulares, tetraedros y hexaedros. Todos ellos estarán interconectados mediante *nudos*, entendiéndose como tal aquellos puntos de la estructura que pueden desplazarse bajo la acción de las cargas aplicadas, así como los puntos en los cuales se une la estructura a una sustentación rígida. En el resto de este documento, el término movimiento, o desplazamiento, de un nudo se empleará para referirse al vector columna de seis componentes que define la traslación, en las tres primeras, y el giro, en las tres restantes, de dicho nudo.

El análisis completo de una estructura lleva consigo la determinación de los movimientos, esfuerzos o tensiones en cualquiera de los puntos que la componen.



(a) Visualización en modo alámbrico. (b) Visualización en modo sólido.

Figura 2.4: Discretización de una estructura de edificación formada por barras y elementos finitos en 2D.

Una vez conocidos los movimientos en los nudos de la estructura, el cálculo detallado de movimientos y esfuerzos (en cualquier punto intermedio de una barra) y de tensiones (en cualquier parte de un elemento finito) depende exclusivamente de las características de los mismos y no de la posición espacial que ocupan en la estructura.

En el análisis tridimensional a realizar, utilizaremos dos sistemas de referencia, uno global, a la propia estructura, y otro local, para las barras y los elementos finitos bidimensionales. Como podemos observar en la figura 2.5, el sistema de coordenadas globales de la estructura estará compuesto por tres ejes ortogonales entre sí, donde el eje X será horizontal, el eje Y se obtendrá del anterior girándolo un ángulo de 90º en sentido antihorario, dentro del plano de simetría, y el eje Z será el resultado de efectuar el producto vectorial de los dos ejes anteriores, quedando definido tanto su dirección como su signo. En este sistema de coordenadas globales expresaremos las coordenadas de los nudos, sus movimientos, las cargas que actúan sobre los mismos y la tensión a la que se ven sometidos. Se entiende por ejemplo que los desplazamientos serán positivos cuando coincidan con el sentido positivo de los ejes globales y los giros cuando sean antihorarios.

En el caso de las barras, existirá un sistema local para cada una de ellas,

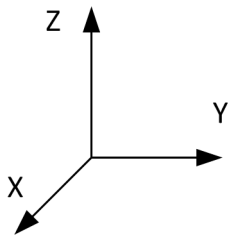


Figura 2.5: Sistema de referencia global empleado en el análisis estructural.

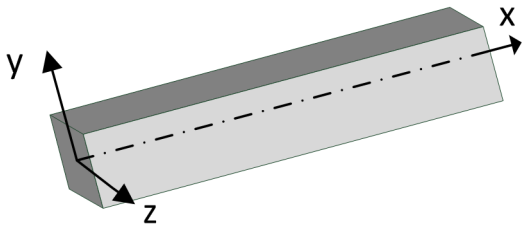


Figura 2.6: Sistema de referencia local de una barra.

el cual presentará su origen de coordenadas en el llamado extremo inicial de la barra (elegido arbitrariamente entre ambos nudos), compuesto también por tres ejes perpendiculares entre sí. El eje X de dicho sistema local quedará alineado con la directriz de la barra, yendo desde el nudo inicial hasta el final, y a los otros dos ejes les exigiremos unas condiciones parecidas a las que hemos impuesto a los dos últimos ejes globales, de manera que el eje Y se corresponderá con los movimientos verticales de barra, mientras que el eje Z quedará orientado expresando sus movimientos horizontales, siendo ortogonal al plano que definen los ejes X e Y, como muestra la figura 2.6. En el caso de barras verticales, el eje Y local de la barra estará alineado con el eje Y de la estructura, lo cual supone que el eje Z local será paralelo, aunque con sentido contrario, al eje X global. Precisamente será con respecto al eje local de cada barra en el que expresaremos las solicitaciones en los extremos de las barras y los esfuerzos y deformaciones en sus puntos intermedios.

En ocasiones, ocurre que las barras podrán presentar un giro sobre el eje X local, el cual siempre es positivo y se consigue girando su eje X en el sentido del eje Y al Z. A dicho ángulo α se le denomina ángulo de rotación o giro y se utilizará en el algoritmo 1 para calcular la matriz de rotación de dicha barra.

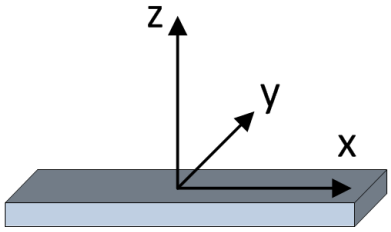


Figura 2.7: Sistema de referencia local de un elemento finito en 2D.

En lo que respecta a los elementos finitos en 2D (triángulos y cuadriláteros), y tal y como podemos ver en la figura 2.7, emplearemos un sistema de referencia local donde los ejes X e Y formarán un plano coplanario con el elemento finito, siendo el eje Z perpendicular a dicho plano. Por el contrario, los elementos finitos en 3D no poseerán un sistema de referencia propio.

De aquí en adelante, los vectores de desplazamientos d y de cargas f en un nudo i estarán formados, respectivamente, por las siguientes componentes:

$$d_i = \begin{bmatrix} \delta_{ix} \\ \delta_{iy} \\ \delta_{iz} \\ \theta_{ix} \\ \theta_{iy} \\ \theta_{iz} \end{bmatrix} \qquad f_i = \begin{bmatrix} f_{ix} \\ f_{iy} \\ f_{iz} \\ m_{ix} \\ m_{iy} \\ m_{iz} \end{bmatrix} \tag{2.1}$$

cuando estén expresados en ejes globales y por estos otros cuando estén expresados con respecto a los ejes locales de algún elemento que a él confluye:

$$d'_i = \begin{bmatrix} \delta'_{ix} \\ \delta'_{iy} \\ \delta'_{iz} \\ \theta'_{ix} \\ \theta'_{iy} \\ \theta'_{iz} \end{bmatrix} \qquad f'_i = \begin{bmatrix} f'_{ix} \\ f'_{iy} \\ f'_{iz} \\ m'_{ix} \\ m'_{iy} \\ m'_{iz} \end{bmatrix} \tag{2.2}$$

Durante años, fue habitual emplear en el cálculo estático modelos simplificados de una realidad física tridimensional, como es dividir una estructura en

elementos planos y analizarlos de manera independiente. No en vano, el hecho de trabajar con seis grados de libertad en un análisis espacial frente a tres en un análisis plano determina que el número de ecuaciones o incógnitas a resolver sea el doble en un análisis espacial frente a un análisis plano, a igualdad de nudos en ambos casos. Sin embargo, cuando se lleva a cabo un análisis tridimensional, se están considerando ya no solo la totalidad de elementos planos que componen la estructura, sino también la conexión entre los mismos por parte de diferentes elementos estructurales no presentes en un análisis 2D, lo cual conlleva una mayor colaboración y participación conjunta ante la actuación de las cargas, además de un incremento sustancial en la cantidad de nudos y en el número de grados de libertad totales contemplados en el proceso de cálculo.

Muy aplicadas fueron también las técnicas de condensación estática, las cuales pretendían reducir la dimensión del problema, identificando un determinado conjunto de grados de libertad como dependientes o secundarios y expresándolos en función de los grados de libertad llamados independientes o primarios. De ese modo, el problema queda expresado únicamente en función de estos últimos.

En el caso del análisis dinámico, también se han venido realizando diferentes simplificaciones a fin de reducir la complejidad de problema. Buen ejemplo de ello es el modelo llamado *edificio a cortante*, o *método de la fuerza lateral equivalente*, en el cual la masa de cada forjado se agrupa en un solo nudo (modelo de masas concentradas), considerando para el mismo 3 grados de libertad, a lo sumo. De la misma manera, es también posible la aplicación de técnicas de condensación dinámica que, como decíamos anteriormente, persiguen reducir el número de grados de libertad totales a considerar en el sistema y el tiempo de cómputo correspondiente.

No obstante, muchas de estas simplificaciones son únicamente válidas en estructuras que presenten una fuerte regularidad tanto en su tipología como en las cargas aplicadas. Por el contrario, cuando no se den estas circunstancias, es de esperar la aparición de esfuerzos de torsión no contemplados.

Es importante aclarar que el comportamiento dinámico de las estructuras no puede asimilarse, en general, al del modelo matemático de uno o varios grados de libertad por planta, citado anteriormente. No olvidemos que cuando la estructura

pueda desplazarse con más grados de libertad de los contemplados, la solución que se obtiene será solamente una aproximación al verdadero comportamiento dinámico. En consecuencia, debe llevarse a cabo un análisis tridimensional realista, con una discretización espacial en el cual el número de grados de libertad del análisis dinámico coincide al utilizado en un análisis estático, compuesto por varios cientos de miles de ecuaciones, e incluso millones, y donde se utilice el modelo de masas consistentes y no el de concentradas.

2.5. Los métodos de análisis

2.5.1. Introducción

Los métodos matriciales [25] representan la herramienta de análisis más potente en la ingeniería estructural, siendo muy adecuados desde el punto de vista práctico para ser implementados en computadores tanto secuenciales como paralelos. Por otro lado, y desde el punto de vista teórico, el empleo de la notación matricial presenta la ventaja de utilizar métodos de cálculo de una forma compacta, precisa y, al mismo tiempo, completamente general.

En contraste con estas ventajas, los métodos matriciales se caracterizan por una gran cantidad de cálculo sistemático, de manera que su aplicación al cálculo práctico de estructuras se reduce a la adecuación de los ordenadores para llevar a cabo el trabajo numérico. Se desprende de esto que el principal campo de aplicación está en el cálculo de grandes y complejas estructuras, en las que los métodos manuales tradicionales requieren una dosis excesiva de esfuerzo humano o del computador.

Las hipótesis que vamos a considerar en el cálculo estructural matricial en esta tesis doctoral son las de *linealidad* y *superposición*. Se dice que una estructura tiene un comportamiento lineal si todos los movimientos y esfuerzos son funciones lineales de las cargas aplicadas sobre ella. Este principio está controlado por la teoría de las pequeñas deformaciones (se supone que la estructura no cambia sustancialmente bajo la aplicación de las cargas), así como por las propiedades físicas de los materiales que forman la estructura.

Esta hipótesis de linealidad presenta dos ventajas importantes. En primer lugar, simplifica notablemente el trabajo real de analizar una estructura bajo un sistema dado de cargas. En segundo lugar, permite la superposición de soluciones, de manera que los esfuerzos y movimientos producidos en una estructura por un conjunto de cargas actuando simultáneamente pueden obtenerse por adición de los esfuerzos y movimientos producidos por cada carga actuando por separado.

Cuando las cargas se aplican sobre una estructura lentamente, y su valor no varía lo largo del tiempo, la estructura se deformará bajo estas cargas y quedará en reposo en su forma final. Desde este instante la estructura no sufre cambios ni en su posición ni en su forma deformada, alcanzando la denominada posición de equilibrio estático de la estructura. Por el contrario, si las cargas se aplican súbitamente, con variaciones de valor a lo largo de su tiempo de actuación, la estructura alcanzará diferentes deformaciones en diferentes instantes.

2.5.2. Análisis por barras

En este tipo de análisis, los elementos de la estructura como vigas y pilares se modelizan mediante las llamadas *barras*, elementos unidimensionales donde la longitud es muy superior a su sección. La figura 2.8 muestra un ejemplo de una sencilla estructura modelizada de este modo.

En un análisis lineal de una estructura formada, en parte o en su totalidad, por barras, hay tres grupos de condiciones que fuerzas y movimientos deben satisfacer:

1. Las fuerzas que actúan en los extremos de las barras y los movimientos de dichos extremos deben satisfacer las *condiciones de comportamiento* elástico y lineal deducido del diagrama *tensión-deformación* del material del cual está compuesta la barra.

2. Los movimientos de los extremos de cada barra coinciden con los de los nudos a los cuales están unidos. Estas son las denominadas *condiciones de compatibilidad*. Se entiende que el desplazamiento de cualquier punto de la estructura es continuo y tiene un solo valor.

3. Las fuerzas que actúan en los extremos de cada barra deben ser tales que

Figura 2.8: Modelización de una estructura mediante barras.

la mantengan en equilibrio. Más aún, la resultante de las fuerzas en los extremos de todas las piezas que coinciden en un nudo cualquiera debe ser igual a la carga exterior aplicada en dicho nudo. Estas son las denominadas *condiciones de equilibrio*.

En un sentido amplio, los métodos matriciales de cálculo de estructuras pueden clasificarse de acuerdo al orden en el cual se aplican las condiciones de comportamiento, compatibilidad y equilibrio. Los métodos en los cuales empleamos inicialmente las ecuaciones de compatibilidad, para posteriormente aplicar las condiciones de comportamiento y, para finalizar, las de equilibrio, se denominan *métodos de la rigidez* o *de los desplazamientos*. Por el contrario, los métodos donde partimos de las ecuaciones de equilibrio, que utilizamos para incorporarlas a las de comportamiento y que, finalmente, empleamos en las ecuaciones de compatibilidad de movimientos se llaman *métodos de flexibilidad* o *de las fuerzas*.

La esencia de los *métodos de la rigidez* o *de los desplazamientos*, aplicados en esta tesis doctoral, consiste en considerar como incógnitas básicas los movimientos de los nudos. En cualquier aplicación de estos métodos, el primer paso es expresar

los esfuerzos en los extremos de las barras en función de sus correspondientes movimientos. Estas expresiones de los esfuerzos en sus extremos resultan, en general, de la integración de las ecuaciones diferenciales apropiadas de elongación, flexión o torsión asociadas. Si la estructura es lineal, como hemos dicho, los esfuerzos en los extremos de la barra serán funciones lineales de los movimientos de los mismos, siendo los coeficientes de los movimientos funciones de las propiedades del material y dimensiones de las barras.

El paso siguiente utiliza las condiciones de compatibilidad para determinar los movimientos de los extremos de las barras en términos de los movimientos de los nudos. Así, obtenemos expresiones de los esfuerzos que actúan en los extremos de las piezas, en función de los movimientos incógnitas de estos extremos, los cuales satisfacen las ecuaciones tensión-deformación y las ecuaciones de compatibilidad. Estas expresiones de los esfuerzos en los extremos de las barras se sustituyen, entonces, en las ecuaciones de equilibrio del nudo. El resultado es un sistema de ecuaciones que relaciona los esfuerzos conocidos de los nudos con los movimientos desconocidos de los mismos. Habrá una ecuación para cada componente del movimiento del nudo (seis componentes si trabajamos en tres dimensiones) y el término independiente será la componente de la carga aplicada. El número de ecuaciones a resolver será, por tanto, igual al número total de movimientos independientes desconocidos o, lo que es lo mismo, al número total de grados de libertad de la estructura.

Resolviendo este sistema de ecuaciones lineales se obtienen los valores de los movimientos incógnita de los nudos. Una vez hecho esto, pueden determinarse los esfuerzos en los extremos de las barras y, a partir de ahí, los esfuerzos y movimientos en cualquier punto intermedio de ellas, así como las reacciones en los apoyos.

2.5.3. Análisis por elementos finitos

2.5.3.1. Generalidades

El Método de los Elementos Finitos (MEF) [26, 27, 28] es una técnica de resolución de ecuaciones en derivadas parciales a partir de una discretización inicial

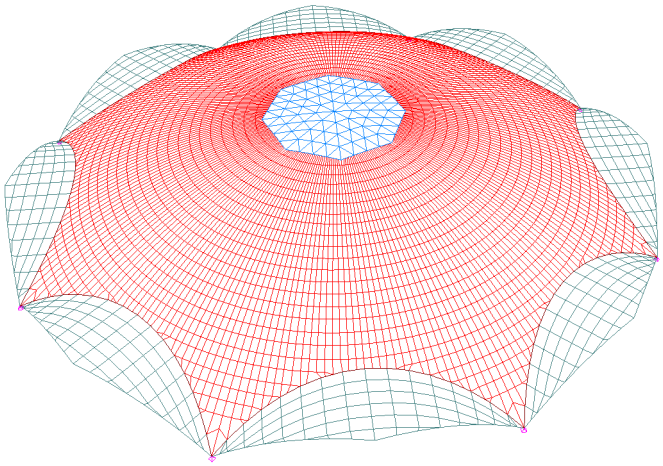


Figura 2.9: Modelización de una estructura mediante elementos finitos bidimensionales.

de dichas ecuaciones en sus dimensiones espaciales. Esta discretización se lleva a cabo empleando pequeñas secciones de forma simple y arbitraria, denominadas *elementos finitos*, capaces de aproximar cualquier cuerpo o estructura siempre que se disminuya de manera adecuada el tamaño de los mismos y, en consecuencia, se aumente su número en la discretización de la estructura continua.

Dicho de otro modo, el MEF es una extensión del método de cálculo matricial donde la estructura, por muy compleja que sea en geometría o forma, se discretiza mediante elementos que no son sólo barras, sino volúmenes de diferentes formas geométricas, rectas e incluso curvas, los cuales los cuales se caracterizan por su mejor adaptación a las zonas perimetrales de las superficies.

Entre las posibles formas a emplear se encuentran los elementos bidimensionales, muy apropiados para reproducir el comportamiento de placas y láminas, donde una de las dimensiones (espesor) es mucho menor que las otras dos. Como elementos bidimensionales, destacan los triángulos (elementos finitos por excelencia), así como los cuadriláteros, los cuales pueden subdividirse a su vez en triángulos. A nivel de elemento constructivos, los elementos finitos en 2D resultan muy útiles para discretizar y simular forjados, losas, muros de contención, bóvedas y cúpulas. A modo de ejemplo, en la figura 2.9 se puede observar la discretización de una estructura mediante triángulos y cuadriláteros.

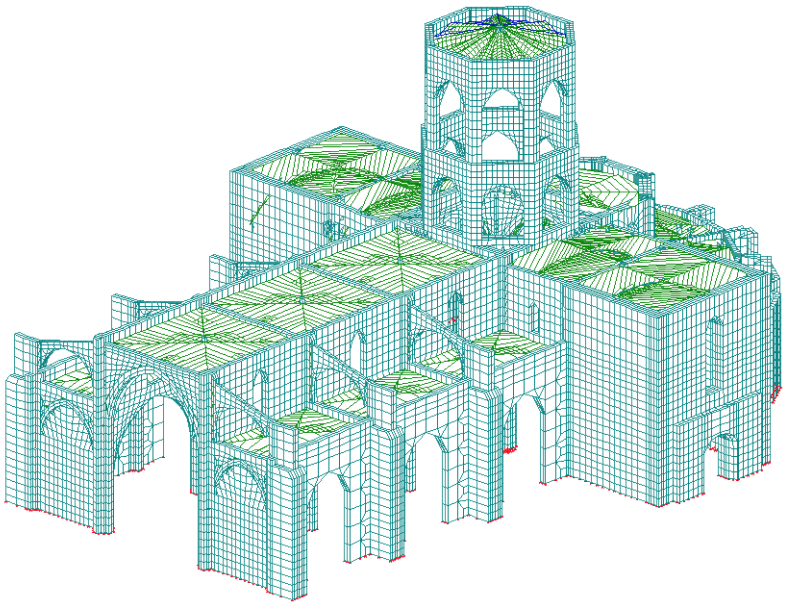


Figura 2.10: Modelización de la catedral de Valencia mediante elementos finitos bidimensionales y tridimensionales.

Hay sin embargo estructuras en las cuales, por su naturaleza tridimensional intrínseca, no resulta apropiada la discretización en elementos unidimensionales o bidimensionales. Ejemplos de ellos pueden ser las estructuras históricas como iglesias, catedrales, etc. (ver figura 2.10) o las presas bóveda y de gravedad, si nos limitamos al ámbito de la edificación y la ingeniería civil, además de un amplio abanico de problemas presentes en la ingeniería aeroespacial e incluso en la ingeniería mecánica, a la hora de simular el comportamiento de las piezas utilizadas. En esos casos, lo que procede es discretizar la estructura empleando elementos tridimensionales, como son los tetraedros, los hexaedros o los prismas triangulares. El uso de estos elementos implica, como es lógico, mayores tiempos de simulación y un incremento en la memoria RAM demandada, motivo por el cual se tendió, siempre que fue posible, a modelizar la estructura de forma que pudiera evitarse el estudio tridimensional, en beneficio de análisis más simplificados [27]. A su vez, dentro de cada elemento se distinguen una serie de puntos representativos, llamados *nodos*, encargados de unir los elementos finitos. Dichos nudos estarán situados, al menos, en los vértices del elemento finito, pudiendo incluso estar en sus lados y en su interior, lo que aumenta notablemente su complejidad.

De este modo, es posible pasar de un sistema continuo inicial, compuesto un número infinito de grados de libertad y gobernado por un conjunto de ecuaciones en derivadas parciales, a un sistema con un número finito de grados de libertad cuyo comportamiento vendrá plasmado en un sistema de ecuaciones lineales o no lineales, a resolver.

A día de hoy, y gracias al avance de los computadores, el MEF se ha convertido en una poderosa e imprescindible herramienta de cálculo aplicable a numerosos sectores en la ingeniería, en los cuales hay que resolver problemas numéricos relacionados, entre otros muchos, con la dinámica de fluidos, el campo magnético, la trasmisión de calor o el análisis estructural, para el cual se ideó. No en vano, son numerosos los programas comerciales orientados a la resolución de problemas por elementos finitos, tales como Abaqus, Ansys, Cosmos o SAP2000.

Al igual que ocurría con la modelización de la estructura mediante barras, las incógnitas fundamentales se materializan sobre los nodos del problema. En el caso que nos atañe, estas incógnitas comienzan siendo los desplazamientos de dichos nodos, obtenidas tras la resolución del citado sistema de ecuaciones. A partir de dichos desplazamientos, es posible determinar el resto de variables de interés, como puede ser la tensión en cada uno de los nodos. Además, gracias a las denominadas *funciones de forma,* y por interpolación, puede obtenerse la deformación y la tensión en cualquier punto intermedio de un elemento finito, conocidos dichos valores en sus nodos.

Dado que este método es un procedimiento aproximado, la precisión alcanzada aumenta con el número de elementos considerados, existiendo mayor similitud entre la estructura discretizada y la estructura continua. Como es lógico, esa mejor similitud requerirá mayor tiempo de cálculo y mayores requerimientos de consumo de memoria del ordenador.

No es posible, a priori, conocer el número de elementos finitos necesarios para proporcionar una solución satisfactoria, ya que depende del problema considerado. La elección de la discretización más apropiada de la estructura debe realizarse de acuerdo a la experiencia del calculista y en base a posibles ensayos realizados. En cualquier caso, se deben analizar soluciones que presenten diferentes discretizaciones, con el fin de garantizar la convergencia de los resultados [29].

Los problemas de análisis estructural están gobernados por las siguientes ecuaciones:

- Relaciones deformaciones - desplazamientos.
- Relaciones tensiones - deformaciones.
- Ecuaciones de equilibrio.

Pasamos a continuación a describir cada una de ellas en las siguientes subsecciones.

2.5.3.2. Funciones de forma

En el MEF, el valor de una variable en cualquier punto de un elemento se obtiene a partir del valor conocido de dicha variable en sus nudos, gracias a las denominadas funciones de forma. Así por ejemplo, el valor de una función ϕ en un punto cualquiera de un elemento, con coordenadas (x,y,z), lo obtendremos del siguiente modo, siendo n el número de nodos del elemento, ϕ_i el valor de la función en el nodo i y $N_i(x,y,z)$ la función de forma i, dependiente de las coordenadas espaciales y entendiendo que hay tantas funciones de forma como nodos tenga el elemento finito.

$$\phi(x,y,z) = \sum_{i=1}^{n} N_i(x,y,z)\,\phi_i \tag{2.3}$$

Dichas funciones de forma deben garantizar determinadas condiciones de continuidad entre los elementos, a fin de asegurar la convergencia a la solución exacta a medida que el tamaño de los elementos tiende a cero. Así, en el caso de la elasticidad es necesario que la interpolación presente continuidad C^0 en las fronteras entre elementos, lo cual supone que únicamente existirá continuidad en las funciones a interpolar, pero no en sus derivadas.

Aunque en la definición anterior ha ocurrido que las funciones de forma se han expresado en función de las coordenadas cartesianas de un punto, cabrá la posibilidad de que también se expresen en función de las llamadas coordenadas

naturales (ξ, η, τ), bajo las cuales el elemento presenta una forma normalizada:

$$\phi(\xi, \eta, \tau) = \sum_{i=1}^{n} N_i(\xi, \eta, \tau)\, \phi_i \tag{2.4}$$

Por último, a la vez de expresar el valor de una función en un punto cualquiera conocido el valor de la función en los nodos, las funciones de forma nos permiten también conocer la coordenadas cartesianas de un punto a partir de sus coordenadas naturales y a partir de las coordenadas cartesianas de los nodos:

$$\begin{aligned} x &= \sum_{i=1}^{n} N_i(\xi, \eta, \tau)\, x_i \\ y &= \sum_{i=1}^{n} N_i(\xi, \eta, \tau)\, y_i \\ z &= \sum_{i=1}^{n} N_i(\xi, \eta, \tau)\, z_i \end{aligned} \tag{2.5}$$

O lo que es lo mismo, expresado matricialmente:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \sum_{i=1}^{n} \mathrm{N_i}\,(\xi, \eta, \zeta) \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \mathrm{N x^e} \tag{2.6}$$

con:

$$\mathrm{N} = \begin{bmatrix} \mathrm{N_1} & \mathrm{N_2} & \ldots & \mathrm{N_n} \end{bmatrix}, \quad \mathrm{N_i} = \begin{bmatrix} N_i\,(\xi, \eta, \zeta) & 0 & 0 \\ 0 & N_i\,(\xi, \eta, \zeta) & 0 \\ 0 & 0 & N_i\,(\xi, \eta, \zeta) \end{bmatrix} \tag{2.7}$$

2.5.3.3. Relación deformaciones unitarias - desplazamientos

Sea d_i^e el vector con los movimientos del nudo i del elemento e y d^e el vector que contiene los movimientos de todos los nodos que definen a dicho elemento. Como hemos visto en el apartado anterior, las funciones de forma nos permitirán calcular los movimientos d en cualquier punto (ξ, η) del elemento tras haber obtenido previamente dichos movimientos en sus nodos. Si N es la matriz que agrupa las

funciones de forma, ocurre que:

$$d = \sum_{i=1}^{n} \mathrm{N_i}(\xi, \eta)\, d_i^e = \mathrm{N} d^e \tag{2.8}$$

A partir de dichos desplazamientos de un punto cualquiera, calcularemos también sus deformaciones unitarias, empleando el operador diferencial matricial ∂, compuesto por derivadas parciales con respecto a las coordenadas cartesianas:

$$\varepsilon = \partial d = \partial \mathrm{N} d^e \tag{2.9}$$

considerando que $B = \partial \mathrm{N}$, siendo B la denominada matriz de deformación, nos queda que:

$$\varepsilon = B d^e \tag{2.10}$$

La relación matricial entre la tensión σ en un punto de un elemento finito y la deformación unitaria a la que se ve sometido tras la aplicación de una carga exterior es de la siguiente forma, considerando un material isótropo y sin deformaciones ni tensiones unitarias iniciales:

$$\sigma = D\varepsilon \tag{2.11}$$

siendo D la matriz constitutiva del material que conforma el elemento, σ el vector de tensiones y ε el vector de deformaciones unitarias. Teniendo en cuenta las relaciones anteriores, es también posible determinar la relación entre la tensión y los desplazamientos:

$$\sigma = DBd^e \tag{2.12}$$

Si el sólido está sometido a un estado de deformación inicial, como puede ser el debido a la deformación térmica, y sobre él actúan unas tensiones iniciales, la relación entre la tensión y las deformaciones unitarias se escribe como:

$$\sigma = D\left(\varepsilon - \varepsilon_0\right) + \sigma_0 \tag{2.13}$$

por lo cual:

$$\sigma = DBd^e - D\varepsilon_0 + \sigma_0 \tag{2.14}$$

2.5.3.4. Ecuaciones de equilibrio

Las ecuaciones de equilibrio de la estructura, en su conjunto, estarán formadas por el ensamblado de las ecuaciones de equilibrio de cada elemento finito, lo cual permite realizar gran parte de los cálculos de forma sistematizada.

2.5.3.5. Principio de los trabajos virtuales

El teorema de los trabajos virtuales dice que un cuerpo o estructura está en equilibrio si el trabajo virtual interno W_i es igual al trabajo virtual externo W_e, para todo campo de desplazamiento d cinemáticamente admisible. Supongamos que el elemento se encuentra en equilibrio sometido a las cargas puntuales q_n^e aplicadas sobre sus nodos, las cargas q_s^e distribuidas en su superficie S^e y las cargas q_v^e distribuidas en su volumen V^e. Si se producen unos desplazamientos nodales virtuales infinitesimales d^e compatibles con las condiciones de contorno, se originan unos desplazamientos virtuales d en los puntos del elemento.

El trabajo interno W_i, en todo el volumen de un elemento, viene dado por el trabajo que realizan las fuerzas internas cuando se producen unas deformaciones virtuales ε compatibles con los desplazamientos virtuales d^e:

$$W_i = \int_{V^e} \varepsilon^T \sigma \, d\nu = d^{eT} \int_{V^e} B^T \sigma \, d\nu = d^{eT} \left(\int_{V^e} B^T D B d^e \, d\nu \right) \tag{2.15}$$

Si existieran deformaciones unitarias y tensiones iniciales, entonces ocurriría que:

$$W_i = \int_{V^e} \varepsilon^T \sigma \, d\nu = d^{eT} \int_{V^e} B^T \sigma \, d\nu =$$

$$= d^{eT} \left(\int_{V^e} B^T D B d^e \, d\nu - \int_{V^e} B^T D \varepsilon_0 \, d\nu + \int_{V^e} B^T \sigma_0 \, d\nu \right) \tag{2.16}$$

Por otro lado, el trabajo externo W_e está relacionado con las fuerzas externas aplicadas sobre el elemento:

$$W_e = d^{eT} q_n^e + \int_{S^e} d^T q_s^e \, ds + \int_{V^e} d^T q_v^e \, d\nu = d^{eT} q_n^e + d^{eT} \int_{S^e} \mathrm{N}^{\mathrm{T}} q_s^e \, ds + d^{eT} \int_{V^e} \mathrm{N}^{\mathrm{T}} q_v^e \, d\nu \tag{2.17}$$

Igualando las expresiones de ambos trabajos llegamos a las ecuaciones de equilibrio del elemento:

$$d^{eT}\left(\int_{V^e} B^T D B d^e \, d\nu - \int_{V^e} B^T D \varepsilon_0 \, d\nu + \int_{V^e} B^T \sigma_0 \, d\nu\right) =$$

$$= d^{eT} q_n^e + d^{eT} \int_{S^e} \mathrm{N}^{\mathrm{T}} q_s^e \, ds + d^{eT} \int_{V^e} \mathrm{N}^{\mathrm{T}} q_v^e \, d\nu \tag{2.18}$$

lo que equivale a que:

$$K^e d^e = f^e \tag{2.19}$$

siendo K^e la matriz de rigidez del elemento, definida por:

$$K^e = \int_{V^e} B^T D B \, d\nu \tag{2.20}$$

y siendo f^e el vector de cargas equivalentes en los nodos del elemento, formado por el vector f_n^e de cargas aplicadas directamente sobre ellos, el vector f_s^e de cargas distribuidas sobre la superficie del elemento, el vector f_v^e de cargas distribuidas sobre su volumen y los vectores $f_{\varepsilon_0}^e$ y $f_{\sigma_0}^e$correspondientes a las deformaciones unitarias y las tensiones iniciales, es decir:

$$f^e = f_n^e + f_s^e + f_v^e + f_{\varepsilon_0}^e - f_{\sigma_0}^e =$$

$$= q_n^e + \int_{S^e} \mathrm{N}^{\mathrm{T}} q_s^e \, ds + \int_{V^e} \mathrm{N}^{\mathrm{T}} q_v^e \, d\nu + \int_{V^e} B^T D \varepsilon_0 \, d\nu - \int_{V^e} B^T \sigma_0 \, d\nu \tag{2.21}$$

Dichas integrales deben resolverse empleando técnicas de cuadratura numérica, siendo el método de la cuadratura de Gauss-Legendre el más utilizado.

Si cada elemento de la estructura está en equilibrio, también lo estará la estructura completa, por lo cual deberemos aplicar el teorema de los trabajos virtuales a la estructura como suma de los trabajos virtuales de cada elemento, lo que da lugar a la condición de equilibrio global de la estructura:

$$K D = F \tag{2.22}$$

siendo K la matriz de rigidez global de la estructura, D el vector con los desplazamientos en los nodos y F el vector de cargas aplicadas.

Si por el contrario deseamos obtener la respuesta dinámica de una estructura, tendremos que incorporar las fuerzas de inercia y de amortiguamiento a las fuerzas internas que tratan de equilibrar la estructura tras la aplicación de las acciones externas. De este modo diríamos que el trabajo interno es:

$$W_i = \int_{V^e} \varepsilon^T \sigma \, d\nu + \int_{V^e} d^T \rho a \, d\nu + \int_{V^e} d^T c \, v d\nu \tag{2.23}$$

donde ρ es la densidad del material que compone el elemento y el parámetro c define el amortiguamiento. De igual manera a como obtenemos los desplazamientos en un punto cualquiera, podemos obtener las velocidades y aceleraciones en dicho punto, conocidos los valores en los nodos:

$$d = \mathrm{N} d^e, \; v = \mathrm{N} v^e, \; a = \mathrm{N} a^e \tag{2.24}$$

Teniendo en cuenta dichas consideraciones, junto con el hecho de que pudieran existir deformaciones unitarias y tensiones iniciales, tendríamos que:

$$W_i = d^{eT} \left(\int_{V^e} B^T D B d^e \, d\nu + \int_{V^e} \rho \mathrm{N}^{\mathrm{T}} \mathrm{N} a^e \, d\nu + \int_{V^e} \mathrm{N}^{\mathrm{T}} c \mathrm{N} v^e \, d\nu \right) -$$

$$- d^{eT} \left(\int_{V^e} B^T D \varepsilon_0 \, d\nu - \int_{V^e} B^T \sigma_0 \, d\nu \right) \tag{2.25}$$

Igualando el trabajo interno con el trabajo externo de la expresión (2.17) llegaríamos a que:

$$K^e d^e + C^e v^e + M^e a^e = f^e \tag{2.26}$$

donde C^e y M^e serían las matrices de amortiguamiento y masa del elemento, obtenidas como:

$$C^e = \int_{V^e} c \; \mathrm{N}^{\mathrm{T}} \mathrm{N} \, d\nu \, d\nu \tag{2.27}$$

$$M^e = \rho \int_{V^e} \mathrm{N}^{\mathrm{T}} \mathrm{N} \, d\nu \tag{2.28}$$

Podemos entender, por tanto, que las fuerzas exteriores aplicadas sobre la estructura se equilibran mediante una combinación de fuerzas elásticas, inerciales y de amortiguamiento. Si dicho principio de trabajo virtuales lo aplicamos conjuntamente a todos los elementos de la estructura, para cada instante de tiempo de aplicación de las fuerzas externas variables, obtendremos la denominada ecuación

del movimiento, la cual se corresponde en realidad con un sistema de ecuaciones diferenciales de segundo orden:

$$KD(t) + CV(t) + MA(t) = F(t) \tag{2.29}$$

siendo K, C y M las matrices globales de rigidez, amortiguamiento y masa de la estructura, *F(t)* el vector con las cargas aplicadas en el instante de tiempo t y *D(t)*, *V(t)* y *A(t)* los vectores con los desplazamientos, las velocidades y las aceleraciones en cada uno de los nudos, para dicho instante.

2.6. El Código Técnico de la Edificación

El Código Técnico de la Edificación (CTE) [30] es el marco normativo actual por el que se regulan las exigencias básicas de calidad que deben cumplir los edificios, incluidas sus instalaciones, para satisfacer los requisitos básicos de seguridad y habitabilidad, en desarrollo de lo previsto en la disposición adicional segunda de la Ley 38/1999, de 5 de noviembre, de Ordenación de la Edificación (LOE).

El CTE establece dichas exigencias básicas para los requisitos básicos de *seguridad estructural*, *seguridad en caso de incendio*, *seguridad de utilización*, *higiene*, *salud y protección del medio ambiente*, *protección contra el ruido* y *ahorro de energía y aislamiento térmico*, establecidos en el artículo 3 de la LOE, y proporciona procedimientos que permiten acreditar su cumplimiento con suficientes garantías técnicas en el proyecto, la construcción, el mantenimiento y la conservación de los edificios y sus instalaciones.

El CTE se aplicará a las obras de edificación de nueva construcción, excepto a aquellas construcciones de sencillez técnica y de escasa entidad constructiva que no tengan carácter residencial o público, que se desarrollen en una sola planta y no afecten a la seguridad de las personas. Igualmente, el CTE se aplicará a las obras de ampliación, modificación, reforma o rehabilitación que se realicen en edificios existentes, siempre y cuando dichas obras sean compatibles con la naturaleza de la intervención y, en su caso, con el grado de protección que puedan tener los edificios afectados.

Junto con las disposiciones y condiciones generales de aplicación del CTE y las exigencias básicas que deben cumplir los edificios, el CTE está formado por los denominados Documentos Básicos (DB), para el cumplimiento de las exigencias básicas del CTE. Entre ellos, los denominados *DB SE Seguridad Estructural*, *DB-SE-AE Acciones en la Edificación*, *DB-SE-C Cimientos*, *DB-SE-A Acero*, *DB-SE-F Fábrica* y *DB-SE-M Madera* especifican parámetros objetivos y procedimientos cuyo cumplimiento asegura la satisfacción de las exigencias básicas de resistencia y estabilidad (en la fase de construcción y uso) y aptitud al servicio, así como la superación de los niveles mínimos de calidad propios del requisito básico de seguridad estructural. Por otro lado, las estructuras de hormigón están reguladas por la Instrucción de Hormigón Estructural vigente (EHE-08) [31].

Dentro de ellos, y en relación con esta tesis doctoral, es de destacar el Documento Básico SE Seguridad Estructural, el cual tiene por objeto establecer reglas y procedimientos que permitan cumplir las exigencias básicas de seguridad estructural. Este requisito básico de seguridad estructural consiste en asegurar que el edificio tiene un comportamiento estructural adecuado frente a las acciones e influencias previsibles a las que pueda estar sometido durante su construcción y uso previsto. De este modo, dicho Documento Básico establece los principios y los requisitos relativos a la resistencia mecánica y la estabilidad del edificio, así como la aptitud al servicio, incluyendo su durabilidad y describiendo las bases y los principios para el cálculo de las mismas.

En palabras del *DB SE*, la comprobación estructural de un edificio requiere:

1. Determinar las situaciones de dimensionado que resulten determinantes, las cuales deben englobar todas las condiciones y circunstancias previsibles durante la ejecución y la utilización de la obra, teniendo en cuenta la diferente probabilidad de cada una.

2. Establecer las acciones que deben tenerse en cuenta, para cada situación de dimensionado, y los modelos adecuados para la estructura.

3. Realizar el análisis estructural, adoptando métodos de cálculo adecuados a cada problema.

4. Verificar que, para las situaciones de dimensionado correspondientes, no se

sobrepasan los estados límite.

Se contemplan las tres siguientes situaciones de dimensionado: *persistentes*, que se refieren a las condiciones normales de uso; *transitorias*, que se refieren a unas condiciones aplicables durante un tiempo limitado; *extraordinarias*, que se refieren a unas condiciones excepcionales en las que se puede encontrar o a las que puede estar expuesto el edificio.

2.7. Acciones en la edificación

Definimos las acciones como todas aquellas cargas generadas por causas internas o externas a la estructura sobre la que actúan. Dichas cargas deben ser estimadas a fin de llevar a cabo el análisis de la estructura, siendo transmitidas desde los distintos forjados que componen el edificio hasta las vigas, de ahí a los pilares o a los muros y desde estos últimos hasta la cimentación y al terreno.

Son múltiples los posibles criterios a seguir a fin de clasificar a las cargas. Si atendemos a la variabilidad del valor de la carga y a la respuesta estructural, las podemos catalogar como estáticas y dinámicas. Las cargas estáticas son aquellas cuyo valor es constante durante su tiempo de actuación. Tanto dicha carga como la deformación correspondiente ocurren lentamente, alcanzándose el valor máximo de deformación cuando la carga estática es máxima. Ejemplo de dichas cargas son las concargas, las sobrecargas, los empujes del terreno, las fuerzas reológicas, las térmicas, etc.

Por el contrario, las cargas dinámicas tienen un valor variable con respecto al tiempo. Su variación con el tiempo es rápida y pueden causar vibraciones en la estructura, motivo por el cual las deformaciones máximas no ocurren con respecto al valor más alto de la carga. A causa de las variaciones en el tiempo, las cargas dinámicas producen fueras de inercia y aceleraciones significativas. Ejemplo de dichas cargas son los terremotos, el viento, las explosiones, los impactos, etc. Tradicionalmente, y de acuerdo a las normativas, algunas acciones dinámicas producidas por el viento, el sismo o los impactos se han representado a través de fuerzas estáticas equivalentes. Dentro del CTE, el Documento Básico *SE-AE,*

Seguridad Estructural, Acciones en la Edificación es el encargado de determinar las acciones que pueden actuar frente a los edificios y que deben considerarse en el cálculo estructural. En dicho documento, las acciones se clasifican, por su variación en el tiempo, de la forma siguiente:

- Acciones permanentes: Son aquellas que actúan en todo momento sobre la estructura con posición constante. Su magnitud puede ser variable o no. En este grupo se incluyen:
  - El peso propio de los elementos estructurales, los cerramientos y elementos separadores, la tabiquería, todo tipo de carpinterías, revestimientos, rellenos y equipo fijo.
  - El pretensado. Se entiende por pretensado la aplicación controlada de una tensión al hormigón mediante el tensado de cables de acero. Se evaluará en base a lo establecido en la instrucción EHE.
  - Las acciones del terreno. Se incluyen aquellas acciones derivadas del empuje del terreno, tanto las procedentes de su peso como de otras acciones que actúan sobre él, o las acciones debidas a sus desplazamientos y deformaciones. Estas acciones se evalúan y tratan según establece el DB-SE-C.

- Acciones variables: Son aquellas que pueden actuar o no sobre la estructura. En este grupo se incluyen:
  - La sobrecarga de uso o peso de todo lo que puede gravitar sobre el edificio por razón de su uso.
  - Las fuerzas sobre barandillas y elementos divisorios (petos, quitamiedos de terrazas, miradores, balcones, escaleras, etc.).
  - Las acciones climáticas tales como el viento, la nieve y la temperatura.

- Acciones accidentales: Son aquellas cuya posibilidad de actuación es pequeña pero de gran importancia. En este grupo se incluyen:
  - Las acciones sísmicas, reguladas en la Norma de Construcción Sismorresistente (NCSE-02) [2].

- Los impactos de un cuerpo, como puede ser un vehículo, sobre un edificio.
- Los incendios, quedando definidas las acciones debidas a la agresión térmica del incendio, dentro del CTE, en el Documento Básico de Seguridad en Caso de Incendio (DB-SI).
- Otras acciones accidentales específicas consideradas en los edificios con usos tales como fábricas químicas, laboratorios o almacenes de materiales explosivos.

A la hora de realizar el cálculo de la estructura, el proyectista establecerá las posibles combinaciones de las acciones citadas. Se entiende como combinación de acciones a aquel conjunto de acciones compatibles que actúan sobre la estructura de manera simultánea.

Capítulo 3

El Análisis Estático Lineal de Estructuras

Este capítulo recoge la formulación necesaria para implementar un análisis lineal y estático por ordenador de una estructura formada por barras y por elementos finitos, tal y como se ha llevado a cabo en esta tesis doctoral. Más concretamente, se detalla la generación de la matriz de rigidez de cada elemento, así como su ensamblaje a la matriz de rigidez de la estructura, además de la generación del vector con las cargas aplicadas y la aportación al mismo por parte de cada elemento.

Tras definir el modo de imponer las condiciones de contorno y obtener los desplazamientos en los nudos, el capítulo describe la obtención de un conjunto de resultados no exclusivos de un análisis estático lineal, entre los cuales se encuentran los esfuerzos en los extremos de las barras, las reacciones en los apoyos, las deformaciones y los esfuerzos en los puntos intermedios de las barras y las deformaciones unitarias, las tensiones y los esfuerzos en los nudos que son vértices de los elementos finitos. Dichos resultados serán por tanto aprovechados en los distintos tipos de análisis dinámico que se detallarán en el capítulo siguiente.

3.1. Introducción

El análisis estático obtiene el comportamiento de una estructura bajo cargas estáticas conocidas, teniendo en consideración, en nuestro caso, la hipótesis de linealidad. El ya citado método de la Rigidez (o de los desplazamientos) emplea las propiedades de rigidez de los elementos estructurales para dar lugar a la ecuación de equilibrio estático (3.1), la cual representa la relación entre las fuerzas externas actuando sobre los nudos de la estructura y lo desplazamientos a los que están siendo sometidos:

$$KD = F \tag{3.1}$$

donde:

- $K \in \mathbb{R}^{nxn}$ es la matriz de rigidez de la estructura, siendo n es el número total de grados de libertad de los que consta, y representa la resistencia de la misma al desplazamiento bajo la influencia de cargas externas. En nuestro caso, al contemplar seis grados de libertad por nudo, n es igual a $6N$, donde N se corresponde con el número total de nudos de la estructura. Esta matriz se obtiene tras el ensamblaje de las matrices de rigidez individuales de los elementos estructurales.
- $F \in \mathbb{R}^{nxm}$ es la matriz de fuerzas sobre los nudos de la estructura, generada a partir de las diferentes cargas aplicadas tanto sobre las barras, los elementos finitos o, directamente, sobre los nudos, siendo m el número total de acciones básicas diferentes aplicadas sobre la estructura y/o cualquier combinación de las mismas.
- $D \in \mathbb{R}^{nxm}$ es la matriz de movimientos incógnita de los nudos de la estructura, para las distintas acciones de fuerzas externas aplicadas.

Por el principio de acción y reacción, al someter una estructura a un conjunto de cargas aparecerán unas reacciones a fin de equilibrar el efecto de las mismas. Durante este proceso, ya que la estructura estará compuesta por materiales deformables, se producirá la deformación, hasta que ésta encuentra su posición de

equilibrio. En consecuencia, la estructura quedará deformada y en equilibrio bajo las correspondientes acciones y reacciones.

Los pasos a seguir para calcular la respuesta lineal estática de una estructura son los siguientes:

1. Generar la matriz de rigidez K.
2. Generar la matriz de cargas externas F sobre los nudos.
3. Imponer las condiciones iniciales o condiciones de contorno del problema.
4. Obtener los desplazamientos D de los nudos, resolviendo la ecuación de equilibrio estático.
5. Calcular las solicitaciones en los extremos de las barras y las reacciones en los apoyos.
6. Determinar los esfuerzos y las deformaciones en cualquier punto intermedio de las barras.
7. Obtener las deformaciones unitarias, las tensiones y los esfuerzos en los nudos a los que confluye algún elemento finito.

3.2. Generación de la matriz de rigidez de la estructura

La matriz K de rigidez de la estructura se compone de la aportación de las matrices de rigidez de cada elemento de la estructura. Es necesario en consecuencia generar previamente las matrices de rigidez de las barras y de los elementos finitos empleados en el proceso de discretización. Se describe por tanto, en este apartado, la matriz de rigidez de los elementos utilizados en esta tesis doctoral en la discretización o mallado de una estructura, tales como barras, triángulos, cuadriláteros, prismas triangulares, tetraedros y hexaedros.

3.2.1. Matriz de rigidez de barras

La matriz de rigidez $K'^b \in \mathbb{R}^{12x12}$, definida en ejes locales, relaciona los esfuerzos presentes en los extremos inicial y final (llamados i y j respectivamente) de una barra b aislada, $f_i'^b$, $f_j'^b$, y los movimientos correspondientes en los mismos, $d_i'^b$, $d_j'^b$, todos ellos vectores columna de 6 componentes expresados con respecto a los ejes locales de la barra:

$$\begin{bmatrix} f_i'^b \\ f_j'^b \end{bmatrix} = K'^b \begin{bmatrix} d_i'^b \\ d_j'^b \end{bmatrix} = \begin{bmatrix} K'_{ii} & K'_{ij} \\ K'_{ji} & K'_{jj} \end{bmatrix} \begin{bmatrix} d_i'^b \\ d_j'^b \end{bmatrix} \tag{3.2}$$

En general, salvo en caso de excentricidades que comentaremos más adelante, dichos extremos i y j de la barra confluirán y se corresponderán con los nudos i y j de la estructura. Si nos restringimos al caso de una barra b de sección constante empotrada en sus dos extremos, su matriz de rigidez K'^b en ejes locales es de la forma:

$$K'^b = \begin{bmatrix} K'_{ii} & K'_{ij} \\ K'_{ji} & K'_{jj} \end{bmatrix} =$$

$$= \begin{bmatrix}
\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & \frac{-EA}{L} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{12EI_z}{L^3} & \frac{12EI_{zy}}{L^3} & 0 & \frac{-6EI_{zy}}{L^2} & \frac{6EI_z}{L^2} & 0 & \frac{-12EI_z}{L^3} & \frac{-12EI_{zy}}{L^3} & 0 & \frac{-6EI_{zy}}{L^2} & \frac{6EI_z}{L^2} \\
0 & \frac{12EI_{zy}}{L^3} & \frac{12EI_y}{L^3} & 0 & \frac{-6EI_y}{L^2} & \frac{6EI_{zy}}{L^2} & 0 & \frac{-12EI_{zy}}{L^3} & \frac{-12EI_y}{L^3} & 0 & \frac{-6EI_y}{L^2} & \frac{6EI_{zy}}{L^2} \\
0 & 0 & 0 & \frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & \frac{-GJ}{L} & 0 & 0 \\
0 & \frac{-6EI_{zy}}{L^2} & \frac{-6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & \frac{-4EI_{zy}}{L} & 0 & \frac{6EI_{zy}}{L^2} & \frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & \frac{-2EI_{zy}}{L} \\
0 & \frac{6EI_z}{L^2} & \frac{6EI_{zy}}{L^2} & 0 & \frac{-4EI_{zy}}{L} & \frac{4EI_z}{L} & 0 & \frac{-6EI_z}{L^2} & \frac{-6EI_{zy}}{L^2} & 0 & \frac{-2EI_{zy}}{L} & \frac{2EI_z}{L} \\
\frac{-EA}{L} & 0 & 0 & 0 & 0 & 0 & \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{-12EI_z}{L^3} & \frac{-12EI_{zy}}{L^3} & 0 & \frac{6EI_{zy}}{L^2} & \frac{-6EI_z}{L^2} & 0 & \frac{12EI_z}{L^3} & \frac{12EI_{zy}}{L^3} & 0 & \frac{6EI_{zy}}{L^3} & \frac{-6EI_z}{L^2} \\
0 & \frac{-12EI_{zy}}{L^3} & \frac{-12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & \frac{-6EI_{zy}}{L^2} & 0 & \frac{12EI_{zy}}{L^3} & \frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & \frac{-6EI_{zy}}{L^2} \\
0 & 0 & 0 & \frac{-GJ}{L} & 0 & 0 & 0 & 0 & 0 & \frac{GJ}{L} & 0 & 0 \\
0 & \frac{-6EI_{zy}}{L^2} & \frac{-6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & \frac{-2EI_{zy}}{L} & 0 & \frac{6EI_{zy}}{L^2} & \frac{6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & \frac{-4EI_{zy}}{L} \\
0 & \frac{6EI_z}{L^2} & \frac{6EI_{zy}}{L^2} & 0 & \frac{-2EI_{zy}}{L} & \frac{2EI_z}{L} & 0 & \frac{-6EI_z}{L^2} & \frac{-6EI_{zy}}{L^2} & 0 & \frac{-4EI_{zy}}{L} & \frac{4EI_z}{L}
\end{bmatrix} \tag{3.3}$$

En dicha matriz, L es la longitud de la barra, E y G son los módulos de elasticidad longitudinal y transversal del material del que se compone la barra, respectivamente, A es el área de la sección transversal a la barra, J representa el módulo de torsión, I_y e I_z expresan los momentos de inercias con respecto a los ejes Y y Z de la barra y, finalmente, I_{zy} es el producto de inercia. Como puede observarse, esta matriz es simétrica y los elementos de la diagonal principal son positivos y no pueden ser nulos. Ello se debe a que el desplazamiento de un extremo de una barra, en un cierto sentido, exige la aplicación en ese extremo de la solicitación correspondiente y en el mismo sentido.

Debemos destacar el hecho de que hay ocasiones en las cuales el extremo de una barra y el nudo correspondiente presentarán una desconexión entre ambos, motivo por el cual no se garantiza la condición de compatibilidad. Ello supone que, en ese caso, la matriz de rigidez citada para la barra no es válida.

De manera genérica, supondremos que las barras están unidas entre sí por uniones rígidas, articuladas o mediante un grado de rigidez intermedio entre 0 (articulada) y 1 (rígida). Cuando la conexión entre nudo y extremo es rígida, las fuerzas o momentos internos de los extremos de las barras se trasmiten con el mismo valor al nudo, siendo por tanto idénticos los movimientos en el extremo de la barra y en el nudo. Sin embargo, cuando están total o parcialmente desconectados, dicha transmisión de esfuerzos será nula o parcial y no se producirá la igualdad de movimientos.

Esta tesis sigue la formulación detallada en [32] que unifica, bajo una misma expresión, los diferentes casos de unión elástica entre el extremo de una barra y el nudo al que confluye. La definición de *grado de rigidez* del extremo de la barra respecto al nudo, en los grados de libertad rotacionales, se realiza relacionando los giros entre ambos, como se muestra en la figura 3.1. Si al aplicar un giro φ en el nudo ocurre que el extremo de la barra gira un ángulo α, el grado de rigidez vendrá dado por:

$$G = \frac{\alpha}{\varphi};\ \ 0 \leq G \leq 1. \tag{3.4}$$

Estos grados de rigidez, como valores comprendidos entre 0 y 1, los aplicaremos únicamente a cada una de las conexiones rotacionales entre el extremo de la barra y el nudo.

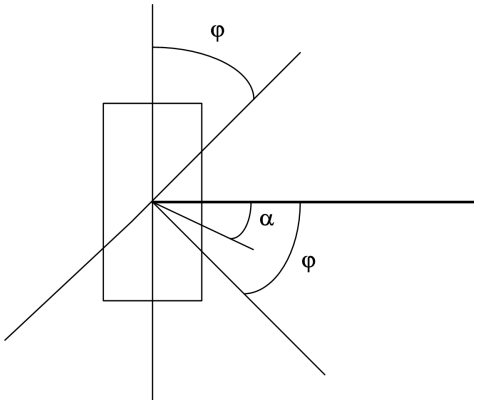


Figura 3.1: Grado de rigidez rotacional entre el extremo de una barra y un nudo.

En las conexiones traslacionales, entenderemos que la barra y el nudo estarán totalmente conectados (1) o desconectados (0), no tomando valores intermedios. En este caso, la definición de grado de rigidez entre el extremo de la barra y el nudo es muy similar a la anterior y viene dada al relacionar los desplazamientos entre ambos, es decir, tras aplicar un desplazamiento δ al nudo y analizar el desplazamiento γ del extremo de la barra:

$$G = \frac{\gamma}{\delta};\ \ G = 0, 1. \tag{3.5}$$

La nomenclatura a emplear será la siguiente:

- Gt_{ix}: Grado de rigidez traslacional del eje X en el extremo i de la barra.
- Gt_{jx}: Grado de rigidez traslacional del eje X en el extremo j de la barra.
- Gt_{iy}: Grado de rigidez traslacional del eje Y en el extremo i de la barra.
- Gt_{jy}: Grado de rigidez traslacional del eje Y en el extremo j de la barra.
- Gt_{iz}: Grado de rigidez traslacional del eje Z en el extremo i de la barra.
- Gt_{jz}: Grado de rigidez traslacional del eje Z en el extremo j de la barra.
- Gr_{ix}: Grado de rigidez rotacional del eje X en el extremo i de la barra.
- Gr_{jx}: Grado de rigidez rotacional del eje X en el extremo j de la barra.

- Gr_{iy}: Grado de rigidez rotacional del eje Y en el extremo i de la barra.
- Gr_{jy}: Grado de rigidez rotacional del eje Y en el extremo j de la barra.
- Gr_{iz}: Grado de rigidez rotacional del eje Z en el extremo i de la barra.
- Gr_{jz}: Grado de rigidez rotacional del eje Z en el extremo j de la barra.

A partir de dichos grados de rigidez, calculamos el valor de las siguientes constantes:

$$C_1 = \frac{\left(1 + \frac{Gr_{jz}}{2}\right) Gr_{iz}}{4 - Gr_{jz}} + \frac{\left(1 + \frac{Gr_{iz}}{2}\right) Gr_{jz}}{4 - Gr_{iz}}; \quad C_2 = \frac{2\left(1 + \frac{Gr_{jz}}{2}\right) Gr_{iz}}{4 - Gr_{jz}}$$

$$C_3 = \frac{\left(1 + \frac{Gr_{jy}}{2}\right) Gr_{iy}}{4 - Gr_{jy}} + \frac{\left(1 + \frac{Gr_{iy}}{2}\right) Gr_{jy}}{4 - Gr_{iy}}; \quad C_4 = \frac{2\left(1 + \frac{Gr_{jy}}{2}\right) Gr_{iy}}{4 - Gr_{jy}}$$

$$C_5 = \frac{3Gr_{iy}}{4 - Gr_{jy}}; \quad C_6 = \frac{3Gr_{iz}}{4 - Gr_{jz}}$$

$$C_7 = \frac{2\left(1 + \frac{Gr_{iz}}{2}\right) Gr_{jz}}{4 - Gr_{iz}}; \quad C_8 = \frac{2\left(1 + \frac{Gr_{iy}}{2}\right) Gr_{jy}}{4 - Gr_{iy}}$$

$$C_9 = Gr_{iy}\left(\frac{2}{4 - Gr_{jy}} + \frac{Gr_{jy}}{4 - Gr_{iy}}\right); \quad C_{10} = \frac{3Gr_{iy}Gr_{jy}}{4 - Gr_{iy}}$$

$$C_{11} = Gr_{iz}\left(\frac{2}{4 - Gr_{jz}} + \frac{Gr_{jz}}{4 - Gr_{iz}}\right); \quad C_{12} = \frac{3Gr_{iz}Gr_{jz}}{4 - Gr_{iz}}$$

$$D_1 = \frac{1}{4 - 3Gt_{iy}}; \quad D_2 = \frac{1}{4 - 3Gt_{jy}}$$

$$D_3 = \frac{1}{4 - 3Gt_{iz}}; \;\; D_4 = \frac{1}{4 - 3Gt_{jz}}$$

$$D_5 = \frac{SIGN(Gt_{iy} - 0.5)}{2 - Gt_{iy}}; \;\; D_6 = \frac{SIGN(Gt_{jy} - 0.5)}{2 - Gt_{jy}}$$

$$D_7 = \frac{SIGN(Gt_{iz} - 0.5)}{2 - Gt_{iz}}; \;\; D_8 = \frac{SIGN(Gt_{jz} - 0.5)}{2 - Gt_{jz}}$$

$$SIGN(x) = \begin{cases} 1 \; si \; x > 0 \\ -1 \; si \; x < 0 \end{cases} \tag{3.6}$$

Una vez obtenidas dichas constantes, las submatrices $K'_{ii}, K'_{ij}, K'_{ji}, K'_{jj}$ que determinan la matriz de rigidez de la barra en ejes locales vienen dadas, respectivamente, por las cuatro siguientes expresiones:

$$\begin{bmatrix} \frac{EAGt_{ix}Gt_{jx}}{L} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{12EI_zC_1Gt_{iy}Gt_{jy}}{L^3} & \frac{12EI_{zy}C_1C_3}{L^3} & 0 & \frac{-6EI_{zy}C_2C_4}{L^2} & \frac{6EI_zC_2Gt_{iy}Gt_{jy}}{L^2} \\ 0 & \frac{12EI_{zy}C_1C_3}{L^3} & \frac{12EI_yC_3Gt_{iz}Gt_{jz}}{L^3} & 0 & \frac{-6EI_yC_4Gt_{iz}Gt_{jz}}{L^2} & \frac{6EI_{zy}C_2C_4}{L^2} \\ 0 & 0 & 0 & \frac{GJGr_{ix}Gr_{jx}}{L} & 0 & 0 \\ 0 & \frac{-6EI_{zy}C_2C_4}{L^2} & \frac{-6EI_yC_4Gt_{iz}Gt_{jz}}{L^2} & 0 & \frac{4EI_yC_5D_3D_4}{L} & \frac{-4EI_{zy}C_5C_6}{L} \\ 0 & \frac{6EI_zC_2Gt_{iy}Gt_{jy}}{L^2} & \frac{6EI_{zy}C_2C_4}{L^2} & 0 & \frac{-4EI_{zy}C_5C_6}{L} & \frac{4EI_zC_6D_1D_2}{L} \end{bmatrix}$$

$$\begin{bmatrix}
\frac{-EAGt_{ix}Gt_{jx}}{L} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{-12EI_zC_1Gt_{iy}Gt_{jy}}{L^3} & \frac{-12EI_{zy}C_1C_3}{L^3} & 0 & \frac{-6EI_{zy}C_7C_8}{L^2} & \frac{6EI_zC_7Gt_{iy}Gt_{jy}}{L^2} \\
0 & \frac{-12EI_{zy}C_1C_3}{L^3} & \frac{-12EI_yC_3^yGt_{iz}Gt_{jz}}{L^3} & 0 & \frac{-6EI_yC_8Gt_{iz}Gt_{jz}}{L^2} & \frac{6EI_{zy}C_7C_8}{L^2} \\
0 & 0 & 0 & \frac{-GJGr_{ix}Gr_{jx}}{L} & 0 & 0 \\
0 & \frac{6EI_{zy}C_9C_{11}}{L^2} & \frac{6EI_yC_9Gt_{iz}Gt_{jz}}{L^2} & 0 & \frac{2EI_yC_{10}D_7D_8}{L} & \frac{-2EI_{zy}C_{10}C_{12}}{L} \\
0 & \frac{-6EI_zC_{11}Gt_{iy}Gt_{jy}}{L^2} & \frac{-6EI_{zy}C_9C_{11}}{L^2} & 0 & \frac{-2EI_{zy}C_{10}C_{12}}{L} & \frac{2EI_zC_{12}D_5D_6}{L}
\end{bmatrix}$$

$$\begin{bmatrix}
\frac{-EAGt_{ix}Gt_{jx}}{L} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{-12EI_zC_1Gt_{iy}Gt_{jy}}{L^3} & \frac{-12EI_{zy}C_1C_3}{L^3} & 0 & \frac{6EI_{zy}C_9C_{11}}{L^2} & \frac{-6EI_zC_{11}Gt_{iy}Gt_{jy}}{L^2} \\
0 & \frac{-12EI_{zy}C_1C_3}{L^3} & \frac{-12EI_yC_3Gt_{iz}Gt_{jz}}{L^3} & 0 & \frac{6EI_yC_9Gt_{iz}Gt_{jz}}{L^2} & \frac{-6EI_{zy}C_9C_{11}}{L^2} \\
0 & 0 & 0 & \frac{-GJGr_{ix}Gr_{jx}}{L} & 0 & 0 \\
0 & \frac{-6EI_{zy}C_7C_8}{L^2} & \frac{-6EI_yC_8Gt_{iz}Gt_{jz}}{L^2} & 0 & \frac{2EI_yC_{10}D_7D_8}{L} & \frac{-2EI_{zy}C_{10}C_{12}}{L} \\
0 & \frac{6EI_zC_7Gt_{iy}Gt_{jy}}{L^2} & \frac{6EI_{zy}C_7C_8}{L^2} & 0 & \frac{-2EI_{zy}C_{10}C_{12}}{L} & \frac{2EI_zC_{12}D_5D_6}{L}
\end{bmatrix}$$

$$\begin{bmatrix}
\frac{EAGt_{ix}Gt_{jx}}{L} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{12EI_zC_1Gt_{iy}Gt_{jy}}{L^3} & \frac{12EI_{zy}C_1C_3}{L^3} & 0 & \frac{6EI_{zy}C_7C_8}{L^2} & \frac{-6EI_zC_7Gt_{iy}Gt_{jy}}{L^2} \\
0 & \frac{12EI_{zy}C_1C_3}{L^3} & \frac{12EI_yC_3Gt_{iz}Gt_{jz}}{L^3} & 0 & \frac{6EI_yC_8Gt_{iz}Gt_{jz}}{L^2} & \frac{-6EI_{zy}C_7C_8}{L^2} \\
0 & 0 & 0 & \frac{GJGr_{ix}Gr_{jx}}{L} & 0 & 0 \\
0 & \frac{6EI_{zy}C_7C_8}{L^2} & \frac{6EI_yC_8Gt_{iz}Gt_{jz}}{L^2} & 0 & \frac{4EI_yC_{13}D_7D_8}{L} & \frac{-4EI_{zy}C_{13}C_{14}}{L} \\
0 & \frac{-6EI_zC_7Gt_{iy}Gt_{jy}}{L^2} & \frac{-6EI_{zy}C_7C_8}{L^2} & 0 & \frac{-4EI_{zy}C_{13}C_{14}}{L} & \frac{4EI_zC_{14}^zD_1D_2}{L}
\end{bmatrix} \tag{3.7}$$

Puesto que cada una de las barras tendrá expresada su matriz de rigidez en ejes locales y la aportación de todas ellas dará lugar a la matriz de rigidez de la estructura, es necesario una transformación dichas matrices de rigidez a ejes globales. De este modo, la matriz de rigidez de una barra expresada con respecto a los ejes globales de la estructura, K^b, es de la siguiente forma, conservando las mismas propiedades que la matriz de rigidez en ejes locales:

$$K^b = \begin{bmatrix} K_{ii} & K_{ij} \\ K_{ji} & K_{jj} \end{bmatrix} = \begin{bmatrix} R^b & 0 \\ 0 & R^b \end{bmatrix} \begin{bmatrix} K'_{ii} & K'_{ij} \\ K'_{ji} & K'_{jj} \end{bmatrix} \begin{bmatrix} R^{bT} & 0 \\ 0 & R^{bT} \end{bmatrix} =$$

$$= \begin{bmatrix} R^b K'_{ii} R^{bT} & R^b K'_{ij} R^{bT} \\ R^b K'_{ji} R^{bT} & R^b K'_{jj} R^{bT} \end{bmatrix} \tag{3.8}$$

y siendo $R^b \in \mathbb{R}^{6x6}$ la matriz de rotación de la barra que nos permite pasar de coordenadas locales a globales. A su vez esta matriz R^b, ortogonal, es de la forma siguiente, con $R_0 \in \mathbb{R}^{3x3}$:

$$R^b = \begin{bmatrix} R_0 & 0 \\ 0 & R_0 \end{bmatrix} \tag{3.9}$$

Sean u, v y w vectores unitarios sobre los ejes locales de la barra cuyas tres componentes serán los cosenos directores de los ejes locales respecto al sistema global de referencia (ver figura 3.2), obtenidos tal y como muestra el algoritmo 1. A partir de dichos vectores, la matriz R_0 es la siguiente:

$$R_0 = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \tag{3.10}$$

Es posible, por tanto, expresar la relación entre los esfuerzos presentes en los extremos de una barra y sus movimientos también en ejes globales:

$$\begin{bmatrix} f_i^b \\ f_j^b \end{bmatrix} = K^b \begin{bmatrix} d_i^b \\ d_j^b \end{bmatrix} = \begin{bmatrix} K_{ii} & K_{ij} \\ K_{ji} & K_{jj} \end{bmatrix} \begin{bmatrix} d_i^b \\ d_j^b \end{bmatrix} \tag{3.11}$$

Gracias a dicha matriz R^b de rotación de la barra, los vectores de esfuerzos y desplazamientos se transforman en ejes globales a partir de esos mismos valores en ejes locales:

$$\begin{bmatrix} f_i^b \\ f_j^b \end{bmatrix} = \begin{bmatrix} R^b & 0 \\ 0 & R^b \end{bmatrix} \begin{bmatrix} f_i'^b \\ f_j'^b \end{bmatrix}, \quad \begin{bmatrix} d_i^b \\ d_j^b \end{bmatrix} = \begin{bmatrix} R^b & 0 \\ 0 & R^b \end{bmatrix} \begin{bmatrix} d_i'^b \\ d_j'^b \end{bmatrix} \tag{3.12}$$

Algoritmo 1 Generación de los vectores unitarios u, v y w que forman la matriz de rotación de una barra.

Entrada: Coordenadas del extremo inicial (x_i, y_i, z_i) y final (x_j, y_j, z_j) de la barra, ángulo de rotación (α) de la barra y su longitud (L).

Salida: Vectores u, v y w.

1: $u_x = \frac{x_j - x_i}{L}$; $u_y = \frac{y_j - y_i}{L}$; $u_z = \frac{z_j - z_i}{L}$
2: **si** $u_x = 0$ y $u_y = 0$ **entonces**
3: $a = [0,\ 0,\ 1]$ // La barra no es vertical
4: **si no**
5: **si** $u_z > 0$ **entonces**
6: $a = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$ // Extremo final situado encima del inicial
7: **si no**
8: $a = \begin{bmatrix} 0 & -1 & 0 \end{bmatrix}$ // Extremo inicial situado encima del final
9: **fin si**
10: **fin si**
11: $w = u \wedge a$
12: $v = w \wedge u$
13: **si** $\alpha > 0$ **entonces**
14: $$v = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \begin{bmatrix} 0 \\ \cos(\alpha) \\ sen(\alpha) \end{bmatrix}$$
15: **fin si**
16: $w = u \wedge v$
17: $v_x = \frac{v_x}{|v|}$; $v_y = \frac{v_y}{|v|}$; $v_z = \frac{v_z}{|v|}$
18: $w_x = \frac{w_x}{|w|}$; $w_y = \frac{w_y}{|w|}$; $w_z = \frac{w_z}{|w|}$

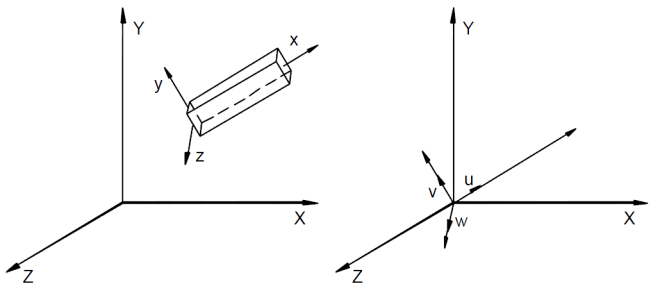


Figura 3.2: Relación entre los sistemas de referencia global, local a la barra y el formado por los vectores unitarios u, v y w.

O viceversa, pasando esfuerzos y desplazamientos de globales a locales:

$$\begin{bmatrix} f_i'^b \\ f_j'^b \end{bmatrix} = \begin{bmatrix} R^{b^T} & 0 \\ 0 & R^{b^T} \end{bmatrix} \begin{bmatrix} f_i^b \\ f_j^b \end{bmatrix}, \quad \begin{bmatrix} d_i'^b \\ d_j'^b \end{bmatrix} = \begin{bmatrix} R^{b^T} & 0 \\ 0 & R^{b^T} \end{bmatrix} \begin{bmatrix} d_i^b \\ d_j^b \end{bmatrix} \tag{3.13}$$

No obstante, la relación entre las fuerzas aplicadas sobre los extremos de una barra y los movimientos experimentados por los mismos en ejes globales no debe definirse únicamente a nivel de barra, sino que debe extenderse a todas las barras, definiendo la relación matricial que relaciona movimientos y fuerzas para todos los nudos de la estructura. Para ello, tendremos que aplicar las condiciones de compatibilidad entre los movimientos de los nudos y los de los extremos de las barras que confluyen a ellos y las condiciones de equilibrio de fuerzas en los nudos.

En cuanto a la condición de compatibilidad, los desplazamientos en los extremos de los elementos que se unen entre sí en un nudo i cualquiera son todos iguales al desplazamiento de ese nudo:

$$d_i = d_i^a = d_i^b = \cdots = d_i^t \tag{3.14}$$

donde a, b, ..., t representan la numeración de las barras unidas por el nudo i.

Por otro lado, atendiendo a la condición de equilibrio en los nudos de la estructura, tendremos que obligar a que la carga exterior f_i que actúa directamente sobre nudo i coincida con la suma de los esfuerzos que ejercen sobre él todas las nb barras conectadas al mismo. Eso supone que para un nudo i:

$$f_i = \sum_{b}^{nb} f_i^b \tag{3.15}$$

Por tanto, la aportación de cada barra de la estructura en su relación esfuerzos-desplazamientos a la ecuación de equilibrio de toda la estructura se llevará a cabo del modo siguiente: En primer lugar, la contribución de la matriz de rigidez de una barra b en ejes globales K^b a la matriz de rigidez completa K de la estructura se obtendrá sumando cada una de sus cuatro submatrices, en las posiciones que relacionan las fuerzas y los movimientos de los nudos i y j. Esto supone que, si un nudo m cualquiera no está unido directamente al nudo i por una barra, entonces

las posiciones correspondientes en la matriz K (fila i y columna m, y fila m y columna i) serán nulas.

Un proceso similar debe seguirse para obtener el vector de cargas externas aplicadas sobre los nudos de la estructura, de manera que los valores que sumaremos en las componentes i y j del vector de fuerzas serán las cargas exteriores f_i y f_j que están actuando directamente sobre dichos nudos i, j.

$$\begin{matrix} \\ i\cdots \\ \\ j\cdots \\ \\ \\ \\ \\ \end{matrix} \begin{bmatrix} 0 \\ \vdots \\ f_i \\ \vdots \\ f_j \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & K_{ii} & \cdots & K_{ij} & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & K_{ji} & \cdots & K_{jj} & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ \vdots \\ d_i \\ \vdots \\ d_j \\ \vdots \\ d_N \end{bmatrix} \tag{3.16}$$
$$\begin{matrix} & & i & & j & & \end{matrix}$$

Esta suma de términos, tanto en lo que se refiere a la matriz de rigidez como al vector de fuerzas, recibe el nombre de ensamblaje, y permite plantear las relaciones de equilibrio interno de todos los nudos de la estructura. Debe quedar claro que la numeración de filas y columnas seguida está expresada en términos de submatrices de 6x6 elementos en la matriz K o, lo que es lo mismo, en términos de bloques de 6x1 elementos en los vectores de desplazamientos y fuerzas.

3.2.2. Matriz de rigidez de barras excéntricas

Aunque siempre se considera que el extremo de una barra confluye y va unido a un nudo, ocurre en ocasiones que dicho extremo de barra y nudo no coinciden en sus coordenadas espaciales. Éste puede ser el caso de pilares de esquina contiguos con diferente sección que no estén centrados, al no coincidir sus ejes.

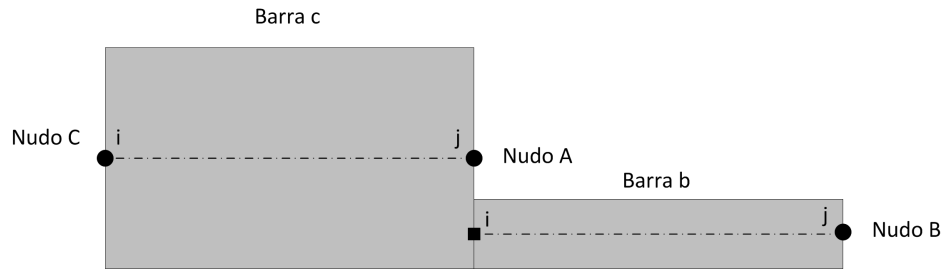


Figura 3.3: Vigas excéntricas.

A modo de ejemplo, la figura 3.3 nos muestra dos barras contiguas de diferente sección y cuyos ejes no están centrados, las cuales confluyen a un mismo nudo A. Como se puede apreciar en la figura, el extremo i de la barra b está situado a una cierta distancia del nudo A. Esto no ocurre en el caso de la barra c, donde el extremo j de la barra sí que coincide con dicho nudo A. Para poder modelizar estas disposiciones constructivas, consideraremos los denominados *nudos extensos*, cuya definición geométrica es un punto que posee un entorno, de rigidez infinita, que alcanza los extremos de los ejes de las barras que a él confluyen.

Sean $[x_i, y_i, z_i]$ las coordenadas espaciales del extremo i de la barra b y sean $[x_A, y_A, z_A]$ las coordenadas espaciales del nudo A. Si restamos ambos vectores, obtenemos este otro vector tal que $[x_{iA}, y_{iA}, z_{iA}]$=$[x_i - x_A, y_i - y_A, z_i - z_A]$, el cual nos permite obtener la siguiente matriz H_{iA}:

$$H_{iA} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -z_{iA} & y_{iA} & 1 & 0 & 0 \\ z_{iA} & 0 & -x_{iA} & 0 & 1 & 0 \\ -y_{iA} & x_{iA} & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

Será precisamente esta matriz H_{iA}, y la que sería la matriz H_{jB} para el extremo j, la que nos permite generar la matriz de rigidez en ejes globales de la barra, partiendo de la relación recogida en la expresión (3.8):

$$K^b = \begin{bmatrix} H_{iA} & 0 \\ 0 & H_{jB} \end{bmatrix} \begin{bmatrix} R^b & 0 \\ 0 & R^b \end{bmatrix} \begin{bmatrix} K'_{ii} & K'_{ij} \\ K'_{ji} & K'_{jj} \end{bmatrix} \begin{bmatrix} R^{b^T} & 0 \\ 0 & R^{b^T} \end{bmatrix} \begin{bmatrix} H^T_{iA} & 0 \\ 0 & H^T_{jB} \end{bmatrix} =$$

$$= \begin{bmatrix} H_{iA} & 0 \\ 0 & H_{jB} \end{bmatrix} \begin{bmatrix} K_{ii} & K_{ij} \\ K_{ji} & K_{jj} \end{bmatrix} \begin{bmatrix} H_{iA}^T & 0 \\ 0 & H_{jB}^T \end{bmatrix} = \begin{bmatrix} H_{iA}K_{ii}H_{iA}^T & H_{iA}K_{ij}H_{jB}^T \\ H_{jB}K_{ji}H_{iA}^T & H_{jB}K_{jj}H_{jB}^T \end{bmatrix} \quad (3.18)$$

Finalmente, podremos también determinar los desplazamientos globales en los extremos de una barra excéntrica, una vez conocidos los desplazamientos globales en los nudos a los que confluye, lo cual vendrá dado por medio de la siguiente formulación:

$$\begin{bmatrix} d_i^b \\ d_j^b \end{bmatrix} = \begin{bmatrix} H_{iA}^T & 0 \\ 0 & H_{jB}^T \end{bmatrix} \begin{bmatrix} d_A \\ d_B \end{bmatrix} \quad (3.19)$$

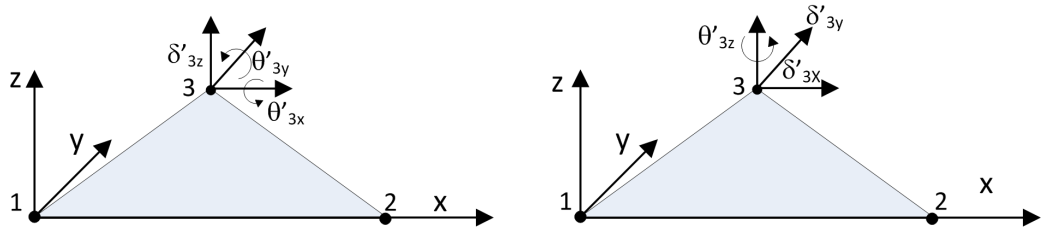
Recordemos que el posterior paso de dichos desplazamientos en los extremos de la barra a sus ejes locales viene recogido en la expresión (3.13).

3.2.3. Matriz de rigidez de elementos triangulares

En este apartado procedemos a describir la formulación del elemento triangular de tres nodos y lados rectos que nos sirva para reproducir el comportamiento de placas y láminas, donde cada elemento estará sometido generalmente a esfuerzos de flexión y de membrana.

De aquí en adelante, consideraremos que los tres nodos del triángulo tienen como numeración global los índices i, j y k y que dicha numeración global se corresponderá con la numeración local, en el propio elemento, *1*, *2* y *3*, respectivamente. Además, utilizaremos la numeración local para obtener las matrices de rigidez de los elementos y la numeración global para expresar su ensamblado en la matriz de rigidez global de la estructura.

En 1980, Batoz, Bathe y Ho [33] estudiaron diferentes elementos triangulares que reproducían el comportamiento de una placa trabajando a flexión, y demostraron que el elemento DKT (Discrete Kirchhoff Triangle) es el elemento triangular más fiable para el análisis de placas delgadas (ver figura 3.4(a)). Se trata de un triángulo con 3 nudos y 3 grados de libertad en cada uno de ellos: una traslación respecto al eje Z local del elemento y dos giros respecto a los ejes locales X e Y. En adelante, diremos que el comportamiento de este elemento es el de placa, teniendo en cuenta que son precisamente los esfuerzos a cortante y a



(a) Grados de libertad del elemento DKT.

(b) Grados de libertad del elemento LST3/9R óptimo.

Figura 3.4: Grados de libertad de diferentes elementos triangulares.

flexión los predominantes en una placa delgada.

Posteriormente, en el año 2003, Felippa [34] desarrolló un elemento triangular con comportamiento de membrana que incorporaba un grado de libertad de rotación alrededor del eje *Z*, normal al plano del elemento, como podemos observar en la figura 3.4(b). A dicho elemento le llamó LST3/9R óptimo, siendo LST las siglas de Linear Strain Triangular, o triángulo de deformación unitaria lineal. Al igual que el anterior posee 3 nodos en los vértices y 3 grados de libertad por cada uno de ellos, los cuales son ahora dos traslaciones en los ejes X e Y locales al elemento y una rotación en el eje Z.

En esta tesis doctoral combinaremos ambos elementos, con el objetivo de reproducir el comportamiento de una lámina, tanto a flexión como a membrana. El elemento empleado estará por tanto formado por 3 nodos, con 6 grados de libertad por nodo. En adelante, utilizaremos la siguiente nomenclatura para expresar, de manera abreviada:

- La diferencia entre las coordenadas cartesianas de los nudos:

$$\begin{aligned} x_{ij} &= x_i - x_j \\ y_{ij} &= y_i - y_j, \quad i,j = 1,2,3 \end{aligned} \tag{3.20}$$

  Conviene resaltar que dichas coordenadas cartesianas no son las coordenadas espaciales de los nodos del triángulo, sino dichas coordenadas cartesianas expresadas en los ejes locales del mismo. Este sistema de referencia local al triángulo tendrá su punto de origen en el nodo *1*.

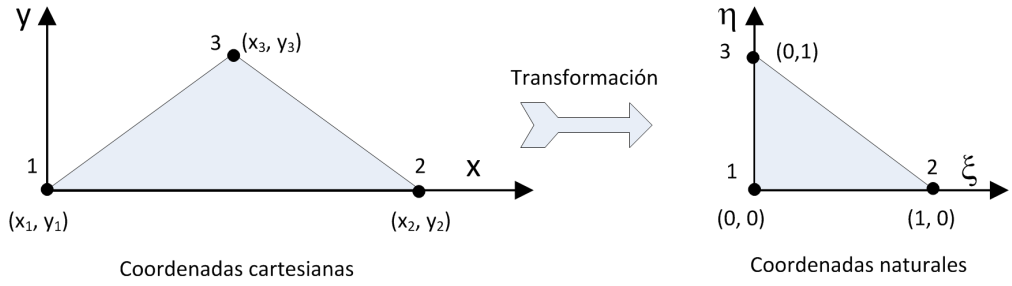


Figura 3.5: Transformación de las coordenadas cartesianas locales de un triángulo lineal a sus coordenadas naturales.

- Las longitudes de los lados:

$$l_{ij} = l_{ji} = \sqrt{x_{ij} + y_{ij}}, \quad i, j = 1, 2, 3, \quad i \neq j \tag{3.21}$$

- El área del triángulo:

$$A = \frac{y_{21}x_{13} - x_{21}y_{13}}{2} \tag{3.22}$$

Antes de continuar, conviene aclarar que la formulación empleada será la denominada isoparamétrica, la cual transforma la geometría real del elemento en una geometría normalizada, bajo la cual se expresan las funciones de forma. Con esto conseguimos, por un lado, independizar la obtención de las funciones de forma de la geometría real del elemento y, por otro lado, facilitar la resolución numérica de las integrales que aparecen en el cálculo de la matriz de rigidez y del vector de cargas de un elemento.

La figura 3.5 nos muestra la transformación de la geometría real de un triángulo de 3 nodos, también conocido como triángulo lineal, expresada de acuerdo a sus coordenadas cartesianas y definidas por sus ejes locales X e Y, a su geometría normalizada, definida en coordenadas naturales con respecto a los ejes ξ y η.

Las funciones de forma del triángulo isoparamétrico lineal son las siguientes:

$$\begin{aligned} N_1(\xi, \eta) &= 1 - \xi - \eta \\ N_2(\xi, \eta) &= \xi \\ N_3(\xi, \eta) &= \eta \end{aligned} \tag{3.23}$$

Por medio de dichas funciones, podremos obtener las coordenadas cartesianas

de un punto a partir de sus coordenadas naturales (ξ, η), conocidas también las coordenadas cartesianas de los nodos:

$$\begin{aligned} x &= \sum_{i=1}^{3} N_i(\xi,\eta)\, x_i = x_1 + \xi x_{21} + \eta x_{31} \\ y &= \sum_{i=1}^{3} N_i(\xi,\eta)\, y_i = y_1 + \xi y_{21} + \eta y_{31} \end{aligned} \tag{3.24}$$

Más adelante, será además necesario conocer sus derivadas parciales con respecto a las coordenadas naturales:

$$\begin{aligned} &\tfrac{\partial N_1}{\partial \xi} = -1, \quad \tfrac{\partial N_2}{\partial \xi} = 1, \quad \tfrac{\partial N_3}{\partial \xi} = 0 \\ &\tfrac{\partial N_1}{\partial \eta} = -1, \quad \tfrac{\partial N_2}{\partial \eta} = 0, \quad \tfrac{\partial N_3}{\partial \eta} = 1 \end{aligned} \tag{3.25}$$

o con respecto a las coordenadas cartesianas, las cuales no serán tan inmediatas como las anteriores y se obtendrán a partir de los siguientes sistemas de ecuaciones lineales:

$$\begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{bmatrix}, \quad i = 1,2,3 \tag{3.26}$$

en los cuales aparece la matriz $J \in \mathbb{R}^{2x2}$, denominada matriz Jacobiana del elemento, expresada en función de las diferencias entre las coordenadas locales:

$$J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \sum_{i=1}^{3} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} x_i & \frac{\partial N_i}{\partial \xi} y_i \\ \frac{\partial N_i}{\partial \eta} x_i & \frac{\partial N_i}{\partial \eta} y_i \end{bmatrix} = \begin{bmatrix} x_{21} & y_{21} \\ x_{31} & y_{31} \end{bmatrix} \tag{3.27}$$

Despejando, las derivadas parciales con respecto a las coordenadas cartesianas valdrán:

$$\begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix} = J^{-1} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{bmatrix} = \frac{1}{|J|} \begin{bmatrix} y_{31} \frac{\partial N_i}{\partial \xi} - y_{21} \frac{\partial N_i}{\partial \eta} \\ x_{21} \frac{\partial N_i}{\partial \eta} - x_{31} \frac{\partial N_i}{\partial \xi} \end{bmatrix}, \quad i = 1,2,3. \tag{3.28}$$

siendo $|J|$ el determinante de la matriz Jacobiana, cuyo valor es el doble del área del triángulo.

Conocidas dichas derivadas parciales, generamos la matriz $B \in \mathbb{R}^{3x6}$ de deformación del elemento, compuesta por tantas submatrices $B_i \in \mathbb{R}^{3x2}$ como número

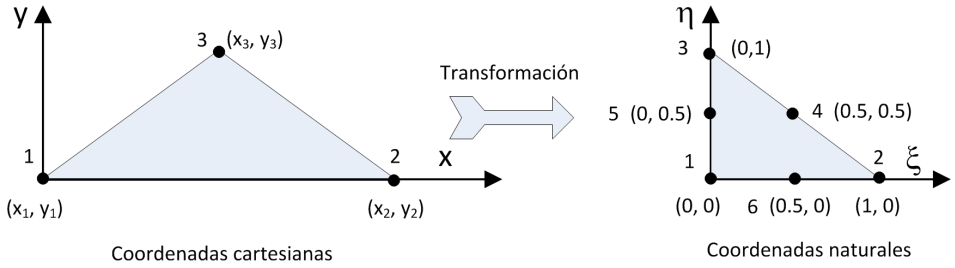


Figura 3.6: Transformación de las coordenadas cartesianas locales del elemento DKT a sus coordenadas naturales.

de nodos tiene el elemento:

$$B = \begin{bmatrix} B_1 & B_2 & B_3 \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{bmatrix} =$$

$$= \frac{1}{2A} \begin{bmatrix} y_{21} - y_{31} & 0 & y_{31} & 0 & -y_{21} & 0 \\ 0 & x_{31} - x_{21} & 0 & -x_{31} & 0 & x_{21} \\ x_{31} - x_{21} & y_{21} - y_{31} & -x_{31} & y_{31} & x_{21} & -y_{21} \end{bmatrix} \tag{3.29}$$

3.2.3.1. Formulación del elemento DKT

Además de los 3 nodos de partida correspondientes a los vértices del triángulo, el elemento DKT tiene en consideración en su formulación a los 3 nodos situados en el punto medio de cada uno de los lados, tal y como podemos observar en la figura 3.6. Se trata por tanto de lo que podríamos denominar como un elemento triangular cuadrático, con las siguientes funciones de forma:

$$\begin{aligned} N_1(\xi,\eta) &= (1-\xi-\eta)(1-2\xi-2\eta) \\ N_2(\xi,\eta) &= \xi(2\xi-1) \\ N_3(\xi,\eta) &= \eta(2\eta-1) \\ N_4(\xi,\eta) &= 4\xi\eta \\ N_5(\xi,\eta) &= 4\eta(1-\xi-\eta) \\ N_6(\xi,\eta) &= 4\xi(1-\xi-\eta) \end{aligned} \tag{3.30}$$

La matriz de rigidez $K_p^{\prime e} \in \mathbb{R}^{9x9}$ de un triángulo e de este tipo, en ejes locales, de acuerdo a la formulación de [33], se expresa como:

$$K_p^{\prime e} = 2A \int_0^1 \int_0^{1-\eta} B^T D_b B \, d\xi \, d\eta \tag{3.31}$$

donde A se corresponde con el área del triángulo y $D_b \in \mathbb{R}^{3x3}$ es la matriz constitutiva del material, o matriz de rigidez a flexión de placa, formada por el coeficiente de Poisson υ del material, su modulo de deformación E y el espesor h del triángulo:

$$D_b = \frac{Eh^3}{12(1-\upsilon^2)} \begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & \frac{1-\upsilon}{2} \end{bmatrix} \tag{3.32}$$

y $B \in \mathbb{R}^{3x9}$ es la matriz de deformación del elemento, definida en función de sus coordenadas naturales (ξ ,η):

$$B(\xi,\eta) = \frac{1}{2A} \begin{bmatrix} y_{31} H_{x,\xi}^T + y_{12} H_{x,\eta}^T \\ -x_{31} H_{y,\xi}^T - x_{12} H_{y,\eta}^T \\ -x_{31} H_{x,\xi}^T - x_{12} H_{x,\eta}^T + y_{31} H_{y,\xi}^T + y_{12} H_{y,\eta}^T \end{bmatrix} \tag{3.33}$$

donde los vectores $H_{x,\xi}$, $H_{x,\eta}$, $H_{y,\xi}$, $H_{y,\eta} \in \mathbb{R}^{9x1}$ dependen también de ξ y η:

$$H_{x,\xi} = \begin{bmatrix} P_6\,(1-2\xi) + (P_5 - P_6)\,\eta \\ q_6\,(1-2\xi) - (q_5 + q_6)\,\eta \\ -4 + 6(\xi+\eta) + r_6\,(1-2\xi) - (r_5 + r_6)\,\eta \\ -P_6\,(1-2\xi) + (P_4 + P_6)\,\eta \\ q_6\,(1-2\xi) - (q_6 - q_4)\,\eta \\ -2 + 6\xi + r_6\,(1-2\xi) + (r_4 - r_6)\,\eta \\ -(P_4 + P_5)\,\eta \\ (q_4 - q_5)\,\eta \\ -(r_5 - r_4)\,\eta \end{bmatrix}, \quad H_{y,\xi} = \begin{bmatrix} t_6\,(1-2\xi) + (t_5 - t_6)\,\eta \\ 1 + r_6\,(1-2\xi) - (r_5 + r_6)\,\eta \\ -q_6\,(1-2\xi) + (q_5 + q_6)\,\eta \\ -t_6\,(1-2\xi) + (t_4 + t_6)\,\eta \\ -1 + r_6\,(1-2\xi) + (r_4 - r_6)\,\eta \\ -q_6\,(1-2\xi) - (q_4 - q_6)\,\eta \\ -(t_4 + t_5)\,\eta \\ (r_4 - r_5)\,\eta \\ -(q_4 - q_5)\,\eta \end{bmatrix}$$

$$H_{x,\eta} = \begin{bmatrix} -P_5\,(1-2\eta) - (P_6 - P_5)\,\xi \\ q_5\,(1-2\eta) - (q_5 + q_6)\,\xi \\ -4 + 6(\xi+\eta) + r_5\,(1-2\eta) - (r_5 + r_6)\,\xi \\ (P_4 + P_6)\,\xi \\ (q_4 - q_6)\,\xi \\ -(r_6 - r_4)\,\xi \\ P_5\,(1-2\eta) - (P_4 + P_5)\,\xi \\ q_5\,(1-2\eta) + (q_4 - q_5)\,\xi \\ -2 + 6\eta + r_5\,(1-2\eta) + (r_4 - r_5)\,\xi \end{bmatrix}, \quad H_{y,\eta} = \begin{bmatrix} -t_5\,(1-2\eta) - (t_6 - t_5)\,\xi \\ 1 + r_5\,(1-2\eta) - (r_5 + r_6)\,\xi \\ -q_5\,(1-2\eta) + (q_5 + q_6)\,\xi \\ (t_4 + t_6)\,\xi \\ (r_4 - r_6)\,\xi \\ -(q_4 - q_6)\,\xi \\ t_5\,(1-2\eta) - (t_4 + t_5)\,\xi \\ -1 + r_5\,(1-2\eta) + (r_4 - r_5)\,\xi \\ -q_5\,(1-2\eta) - (q_4 - q_5)\,\xi \end{bmatrix} \tag{3.34}$$

i	ξ	η	ω
1	1/2	1/2	1/3
2	0	1/2	1/3
3	1/2	0	1/3

Tabla 3.1: Coordenadas naturales y pesos de los 3 puntos de integración empleados en el elemento DKT.

siendo:

$$\begin{aligned} &P_k = -\frac{6x_{ij}}{l_{ij}^2}, \quad q_k = \frac{3x_{ij}y_{ij}}{l_{ij}^2} \\ &t_k = -\frac{6y_{ij}}{l_{ij}^2}, \quad r_k = \frac{3y_{ij}^2}{l_{ij}^2} \\ &k = 4,5,6 \text{ para } ij = 23,\ 31,\ 12, \text{ respectivamente.} \end{aligned} \tag{3.35}$$

Considerando que las propiedades del material y el espesor son idénticos a lo largo del elemento, es posible obtener la solución exacta a la integral de la expresión (3.31) empleando, como puntos de la aproximación de Gauss, aquellos 3 situados en la posición intermedia de cada uno de los lados del triángulo normalizado, todos ellos con un peso $\omega = \frac{1}{3}$, tal y como se muestra en la tabla 3.1. De este modo:

$$K_p'^e = \frac{A}{3}\sum_{i=1}^{3} B(\xi_i,\eta_i)^T D_b B\left(\xi_i,\eta_i\right) \tag{3.36}$$

3.2.3.2. Formulación del elemento LST3/9R óptimo

Según lo recogido en [34], la matriz de rigidez $K_m'^e \in \mathbb{R}^{9x9}$ en ejes locales de un elemento triangular e de este tipo se compone de la suma de las matrices K_{mb} y K_{mh}, respectivamente denominadas matriz de rigidez de membrana básica y matriz de rigidez de membrana de orden superior:

$$K_m'^e = K_{mb} + K_{mh} = \frac{1}{V} L E_m L^T + \int_{V^e} B^T E_m B d\nu \tag{3.37}$$

Si a la hora de resolver numéricamente la integral anterior se emplea la técnica de la cuadratura de Gauss, donde los puntos en los que se evalúa la función son los ubicados en el punto medio de cada lado del triángulo, el resultado de la integral

es exacto, lo que da lugar a que:

$$K_m^{'e} = K_{mb} + K_{mh} = \frac{1}{V} L E_m L^T + \frac{3}{4} \beta_0 \tilde{T}_{\theta u}^T K_\theta \tilde{T}_{\theta u} \tag{3.38}$$

La matriz K_{mb} está formada por el volumen V del elemento, calculado como el producto de su área A por su espesor h, por la matriz $E_m \in \mathbb{R}^{3x3}$ o matriz de elasticidad del material, que se obtiene a partir de su modulo E y su coeficiente de Poisson υ:

$$E_m = \frac{E}{1-\upsilon^2} \begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & \frac{1-\upsilon}{2} \end{bmatrix} \tag{3.39}$$

y por la matriz $L \in \mathbb{R}^{9x3}$ que contiene el parámetro α_b y depende de las diferencias en las coordenadas de los nudos:

$$L = \frac{1}{2} h \begin{bmatrix} y_{23} & 0 & x_{32} \\ 0 & x_{32} & y_{23} \\ \frac{1}{6}\alpha_b y_{23}\left(y_{13}-y_{21}\right) & \frac{1}{6}\alpha_b x_{32}\left(x_{31}-x_{12}\right) & \frac{1}{3}\alpha_b\left(x_{31}y_{13}-x_{12}y_{21}\right) \\ y_{31} & 0 & x_{13} \\ 0 & x_{13} & y_{31} \\ \frac{1}{6}\alpha_b y_{31}\left(y_{21}-y_{32}\right) & \frac{1}{6}\alpha_b x_{13}\left(x_{12}-x_{23}\right) & \frac{1}{3}\alpha_b\left(x_{12}y_{21}-x_{23}y_{32}\right) \\ y_{12} & 0 & x_{21} \\ 0 & x_{21} & y_{12} \\ \frac{1}{6}\alpha_b y_{12}\left(y_{32}-y_{13}\right) & \frac{1}{6}\alpha_b x_{21}\left(x_{23}-x_{31}\right) & \frac{1}{3}\alpha_b\left(x_{23}y_{32}-x_{31}y_{13}\right) \end{bmatrix} \tag{3.40}$$

Dicho parámetro α_b marca la diferencia entre el elemento triangular *CST* (Constant Strain Triangular) y el *LST* (Linear Strain Triangular), reduciendo al primero cuando vale 0, en cuyo caso se anulan las filas y las columnas asociadas a la rotación en el eje Z local al elemento.

Por otro lado, la matriz de rigidez de orden superior K_{mh} depende de estos otros datos. Para comenzar, de la matriz $\tilde{T}_{\theta u} \in \mathbb{R}^{3x9}$, la cual se define como:

$$\tilde{T}_{\theta u} = \frac{1}{4A} \begin{bmatrix} x_{32} & y_{32} & 4A & x_{13} & y_{13} & 0 & x_{21} & y_{21} & 0 \\ x_{32} & y_{32} & 0 & x_{13} & y_{13} & 4A & x_{21} & y_{21} & 0 \\ x_{32} & y_{32} & 0 & x_{13} & y_{13} & 0 & x_{21} & y_{21} & 4A \end{bmatrix} \tag{3.41}$$

Por otro lado, depende de la matriz $K_\theta \in \mathbb{R}^{3x3}$, o matriz de rigidez de orden superior en términos de las rotaciones jerárquicas, la cual es igual a:

$$K_\theta = Ah\left(Q_4^T E_{nat} Q_4 + Q_5^T E_{nat} Q_5 + Q_6^T E_{nat} Q_6\right) \tag{3.42}$$

donde:

$$Q_4 = \frac{1}{2}\left(Q_1 + Q_2\right), \quad Q_5 = \frac{1}{2}\left(Q_2 + Q_3\right), \quad Q_6 = \frac{1}{2}\left(Q_1 + Q_3\right)$$

$$Q_1 = \frac{2A}{3}\begin{bmatrix} \frac{\beta_1}{l_{21}^2} & \frac{\beta_2}{l_{21}^2} & \frac{\beta_3}{l_{21}^2} \\ \frac{\beta_4}{l_{32}^2} & \frac{\beta_5}{l_{32}^2} & \frac{\beta_6}{l_{32}^2} \\ \frac{\beta_7}{l_{13}^2} & \frac{\beta_8}{l_{13}^2} & \frac{\beta_9}{l_{13}^2} \end{bmatrix}, Q_2 = \frac{2A}{3}\begin{bmatrix} \frac{\beta_9}{l_{21}^2} & \frac{\beta_7}{l_{21}^2} & \frac{\beta_8}{l_{21}^2} \\ \frac{\beta_3}{l_{32}^2} & \frac{\beta_1}{l_{32}^2} & \frac{\beta_2}{l_{32}^2} \\ \frac{\beta_6}{l_{13}^2} & \frac{\beta_4}{l_{13}^2} & \frac{\beta_5}{l_{13}^2} \end{bmatrix}, Q_3 = \frac{2A}{3}\begin{bmatrix} \frac{\beta_5}{l_{21}^2} & \frac{\beta_6}{l_{21}^2} & \frac{\beta_4}{l_{21}^2} \\ \frac{\beta_8}{l_{32}^2} & \frac{\beta_9}{l_{32}^2} & \frac{\beta_7}{l_{32}^2} \\ \frac{\beta_3}{l_{13}^2} & \frac{\beta_2}{l_{13}^2} & \frac{\beta_1}{l_{13}^2} \end{bmatrix} \tag{3.43}$$

Finalmente, de $E_{nat} \in \mathbb{R}^{3x3}$ o matriz de deformación unitaria - esfuerzo natural:

$$E_{nat} = T_e^T E_m T_e \tag{3.44}$$

$$T_e = \frac{1}{4A^2}\begin{bmatrix} y_{23}y_{13}l_{21}^2 & y_{31}y_{21}l_{32}^2 & y_{12}y_{32}l_{13}^2 \\ x_{23}x_{13}l_{21}^2 & x_{31}x_{21}l_{32}^2 & x_{12}x_{32}l_{13}^2 \\ \left(y_{23}x_{31} + x_{32}y_{13}\right)l_{21}^2 & \left(y_{31}x_{12} + x_{13}y_{21}\right)l_{32}^2 & \left(y_{12}x_{23} + x_{21}y_{32}\right)l_{13}^2 \end{bmatrix} \tag{3.45}$$

En el caso de que el material sea elástico homogéneo e isótropo, presentando el mismo comportamiento mecánico para cualquier dirección de estiramiento alrededor de un punto, los valores que dan lugar al elemento de membrana óptimo son:

$$\begin{aligned} &\alpha_b = \tfrac{3}{2}, \quad \beta_0 = \tfrac{1}{2}\left(1 - 4\upsilon^2\right), \quad \beta_{1,3,5} = 1 \\ &\beta_2 = 2, \quad \beta_4 = 0, \quad \beta_{6,7,8} = -1, \quad \beta_9 = -2 \end{aligned} \tag{3.46}$$

3.2.3.3. Formulación del elemento triangular de lámina

Tal y como hemos comentario con anterioridad, en esta tesis combinaremos los elementos DKT y LST3/9R óptimo con el objetivo de reproducir el comportamiento de una lámina plana, tanto a flexión como a membrana. Eso supone que dicho elemento estará compuesto 3 nodos, con 6 grados de libertad en cada

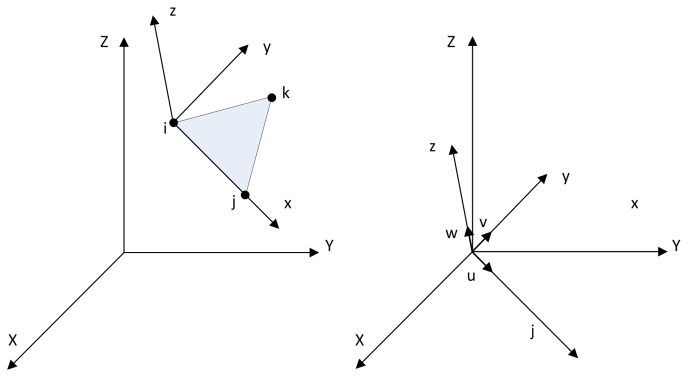


Figura 3.7: Relación entre los sistemas de referencia global, local al triángulo y el formado por los vectores unitarios u, v y w.

uno de ellos. En consecuencia, la generación de la matriz de rigidez en ejes locales $K'^e \in \mathbb{R}^{18x18}$ de un elemento e triangular estará formada por la agrupación, en una misma matriz, de las dos matrices $K'^e_p \in \mathbb{R}^{9x9}$ y $K'^e_m \in \mathbb{R}^{9x9}$ vistas con anterioridad.

Supongamos un triángulo e cuyos nudos se corresponden con la numeración i, j y k, como el mostrado en la figura 3.7. La matriz de rigidez K'^e_p del elemento con comportamiento de placa (DKT) será de la forma:

$$K'^e_p = \begin{bmatrix} K'^p_{ii} & K'^p_{ij} & K'^p_{ik} \\ K'^p_{ji} & K'^p_{jj} & K'^p_{jk} \\ K'^p_{ki} & K'^p_{kj} & K'^p_{kk} \end{bmatrix} \tag{3.47}$$

y establece la siguiente relación entre fuerzas y desplazamientos locales en sus vértices:

$$\begin{bmatrix} f'_{iz} \\ m'_{ix} \\ m'_{iy} \\ f'_{jz} \\ m'_{jx} \\ m'_{jy} \\ f'_{kz} \\ m'_{kx} \\ m'_{ky} \end{bmatrix} = \left[\begin{array}{ccc|ccc|ccc} & & & & & & & & \\ & K'^p_{ii} & & & K'^p_{ij} & & & K'^p_{ik} & \\ - & - & - & - & - & - & - & - & - \\ & & & & & & & & \\ & K'^p_{ji} & & & K'^p_{jj} & & & K'^p_{jk} & \\ - & - & - & - & - & - & - & - & - \\ & & & & & & & & \\ & K'^p_{ki} & & & K'^p_{kj} & & & K'^p_{kk} & \\ & & & & & & & & \end{array}\right] \begin{bmatrix} \delta'_{iz} \\ \theta'_{ix} \\ \theta'_{iy} \\ \delta'_{jz} \\ \theta'_{jx} \\ \theta'_{jy} \\ \delta'_{kz} \\ \theta'_{kx} \\ \theta'_{ky} \end{bmatrix} \tag{3.48}$$

Del mismo modo, la matriz de rigidez K'_{em} del elemento con comportamiento a membrana (LST3/9R óptimo) estará formada por las 9 siguientes submatrices:

$$K'^{e}_{m} = \begin{bmatrix} K'^{m}_{ii} & K'^{m}_{ij} & K'^{m}_{ik} \\ K'^{m}_{ji} & K'^{m}_{jj} & K'^{m}_{jk} \\ K'^{m}_{ki} & K'^{m}_{kj} & K'^{m}_{kk} \end{bmatrix} \tag{3.49}$$

y permite expresar que:

$$\begin{bmatrix} f'_{ix} \\ f'_{iy} \\ m'_{iz} \\ f'_{jx} \\ f'_{jy} \\ m'_{jz} \\ f'_{kx} \\ f'_{ky} \\ m'_{kz} \end{bmatrix} = \left[\begin{array}{ccc|ccc|ccc} & & & & & & & & \\ & K'^{m}_{ii} & & & K'^{m}_{ij} & & & K'^{m}_{ik} & \\ - & - & - & - & - & - & - & - & - \\ & & & & & & & & \\ & K'^{m}_{ji} & & & K'^{m}_{jj} & & & K'^{m}_{jk} & \\ - & - & - & - & - & - & - & - & - \\ & & & & & & & & \\ & K'^{m}_{ki} & & & K'^{m}_{kj} & & & K'^{m}_{kk} & \\ & & & & & & & & \end{array}\right] \begin{bmatrix} \delta'_{ix} \\ \delta'_{iy} \\ \theta'_{iz} \\ \delta'_{jx} \\ \delta'_{jy} \\ \theta'_{jz} \\ \delta'_{kx} \\ \delta'_{ky} \\ \theta'_{kz} \end{bmatrix} \tag{3.50}$$

Agrupando ambas matrices, sumaremos el comportamiento del elemento como placa y como membrana, produciendo el efecto completo de lámina. De este modo, la matriz de rigidez en ejes locales K'^{e} recoge, ahora sí, la relación entre fuerzas y desplazamientos para los 6 grados de libertad de cada nodo del triángulo:

$$\begin{bmatrix} f'^{e}_{i} \\ f'^{e}_{j} \\ f'^{e}_{k} \end{bmatrix} = K'^{e} \begin{bmatrix} d'^{e}_{i} \\ d'^{e}_{j} \\ d'^{e}_{k} \end{bmatrix} = \begin{bmatrix} K'_{ii} & K'_{ij} & K'_{ik} \\ K'_{ji} & K'_{jj} & K'_{jk} \\ K'_{ki} & K'_{kj} & K'_{kk} \end{bmatrix} \begin{bmatrix} d'^{e}_{i} \\ d'^{e}_{j} \\ d'^{e}_{k} \end{bmatrix} \tag{3.51}$$

Cada submatriz de la matriz de rigidez con efecto de lámina, de tamaño 6x6, estará formada por las submatrices correspondientes de las matrices de placa y membrana, de tamaño 3x3, ubicándolas en las posiciones apropiadas de acuerdo a los grados de libertad con los que cada una trabaja. A continuación se muestra un ejemplo de lo que estamos comentando para una submatriz cualquiera, como la K'_{ij}:

$$K'_{ij} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & K'^{p} & K'^{p} & K'^{p} & 0 \\ 0 & 0 & K'^{p} & K'^{p} & K'^{p} & 0 \\ 0 & 0 & K'^{p} & K'^{p} & K'^{p} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} K'^{m} & K'^{m} & 0 & 0 & 0 & K'^{m} \\ K'^{m} & K'^{m} & 0 & 0 & 0 & K'^{m} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ K'^{m} & K'^{m} & 0 & 0 & 0 & K'^{m} \end{bmatrix} =$$

$$= \begin{bmatrix} K'^{m} & K'^{m} & 0 & 0 & 0 & K'^{m} \\ K'^{m} & K'^{m} & 0 & 0 & 0 & K'^{m} \\ 0 & 0 & K'^{p} & K'^{p} & K'^{p} & 0 \\ 0 & 0 & K'^{p} & K'^{p} & K'^{p} & 0 \\ 0 & 0 & K'^{p} & K'^{p} & K'^{p} & 0 \\ K'^{m} & K'^{m} & 0 & 0 & 0 & K'^{m} \end{bmatrix} \tag{3.52}$$

Tras obtener la matriz de rigidez de un elemento en ejes locales, procede ensamblarla a fin de generar la matriz de rigidez global de la estructura. Para ello, debemos transformar a ejes globales dicha matriz de rigidez del triángulo, empleando la matriz de rotación $R^e \in \mathbb{R}^{6x6}$ del elemento:

$$K^e = \begin{bmatrix} R^e & 0 & 0 \\ 0 & R^e & 0 \\ 0 & 0 & R^e \end{bmatrix} \begin{bmatrix} K'_{ii} & K'_{ij} & K'_{ik} \\ K'_{ji} & K'_{jj} & K'_{jk} \\ K'_{ki} & K'_{kj} & K'_{kk} \end{bmatrix} \begin{bmatrix} R^{eT} & 0 & 0 \\ 0 & R^{eT} & 0 \\ 0 & 0 & R^{eT} \end{bmatrix} =$$

$$= \begin{bmatrix} R^e K'_{ii} R^{eT} & R^e K'_{ij} R^{eT} & R^e K'_{ik} R^{eT} \\ R^e K'_{ji} R^{eT} & R^e K'_{jj} R^{eT} & R^e K'_{jk} R^{eT} \\ R^e K'_{ki} R^{eT} & R^e K'_{kj} R^{eT} & R^e K'_{kk} R^{eT} \end{bmatrix} = \begin{bmatrix} K_{ii} & K_{ij} & K_{ik} \\ K_{ji} & K_{jj} & K_{jk} \\ K_{ki} & K_{kj} & K_{kk} \end{bmatrix} \tag{3.53}$$

La citada matriz de rotación R^e es ortogonal y se obtiene a partir de la matriz $R_0 \in \mathbb{R}^{3x3}$, compuesta por los vectores unitarios u, v y w sobre los ejes locales del triángulo, cuyas componentes serán los cosenos directores de los ejes locales respecto al sistema global de referencia, como muestra la figura 3.7 y como se

calcula en el algoritmo 2:

$$R^e = \begin{bmatrix} R_0 & 0 \\ 0 & R_0 \end{bmatrix}, \quad R_0 = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \tag{3.54}$$

Algoritmo 2 Generación de los vectores unitarios u, v y w que componen la matriz de rotación de un triángulo.

Entrada: Coordenadas en ejes globales del nudo i (x_i, y_i, z_i), del nudo j (x_j, y_j, z_j) y del nudo k (x_k, y_k, z_k) que forman el triángulo.

Salida: Vectores u, v y w.

1: $u = [x_j - x_i,\ y_j - y_i,\ z_j - z_i]$
2: $u_x = \frac{u_x}{|u|};\ u_y = \frac{u_y}{|u|};\ u_z = \frac{u_z}{|u|}$
3: $a = [x_k - x_i,\ y_k - y_i,\ z_k - z_i]$
4: $w = u \wedge a$
5: $w_x = \frac{w_x}{|w|}; w_y = \frac{w_y}{|w|}; w_z = \frac{w_z}{|w|}$
6: $v = w \wedge u$

La relación por tanto entre las fuerzas presentes en los vértices del triángulo y sus movimientos, definida en ejes locales en (3.51), puede ahora expresarse en ejes globales:

$$\begin{bmatrix} f_i^e \\ f_j^e \\ f_k^e \end{bmatrix} = K^e \begin{bmatrix} d_i^e \\ d_j^e \\ d_k^e \end{bmatrix} = \begin{bmatrix} K_{ii} & K_{ij} & K_{ik} \\ K_{ji} & K_{jj} & K_{jk} \\ K_{ki} & K_{kj} & K_{kk} \end{bmatrix} \begin{bmatrix} d_i^e \\ d_j^e \\ d_k^e \end{bmatrix} \tag{3.55}$$

Del mismo modo, es también posible transformar a ejes globales los vectores de fuerzas y movimientos en los nudos del triángulo a partir de dichos valores en ejes locales, o viceversa:

$$\begin{bmatrix} f_i^e \\ f_j^e \\ f_k^e \end{bmatrix} = \begin{bmatrix} R^e & 0 & 0 \\ 0 & R^e & 0 \\ 0 & 0 & R^e \end{bmatrix} \begin{bmatrix} f_i'^e \\ f_j'^e \\ f_k'^e \end{bmatrix}, \quad \begin{bmatrix} d_i^e \\ d_j^e \\ d_k^e \end{bmatrix} = \begin{bmatrix} R^e & 0 & 0 \\ 0 & R^e & 0 \\ 0 & 0 & R^e \end{bmatrix} \begin{bmatrix} d_i'^e \\ d_j'^e \\ d_k'^e \end{bmatrix} \tag{3.56}$$

$$\begin{bmatrix} f_i'^e \\ f_j'^e \\ f_k'^e \end{bmatrix} = \begin{bmatrix} R^{eT} & 0 & 0 \\ 0 & R^{eT} & 0 \\ 0 & 0 & R^{eT} \end{bmatrix} \begin{bmatrix} f_i^e \\ f_j^e \\ f_k^e \end{bmatrix}, \begin{bmatrix} d_i'^e \\ d_j'^e \\ d_k'^e \end{bmatrix} = \begin{bmatrix} R^{eT} & 0 & 0 \\ 0 & R^{eT} & 0 \\ 0 & 0 & R^{eT} \end{bmatrix} \begin{bmatrix} d_i^e \\ d_j^e \\ d_k^e \end{bmatrix} \tag{3.57}$$

Al igual como ocurría en el caso de las barras, la aportación de la matriz de rigidez de un triángulo, en ejes globales, a la matriz de rigidez completa se realizará sumando sus 9 submatrices en las posiciones correspondientes donde se expresa la relación entre las fuerzas y los movimientos de los nudos i, j y k del triángulo. Del mismo modo, si hay cargas exteriores f_j, f_j y f_k aplicadas directamente sobre los nudos del triángulo que no se hayan aportado previamente al vector de fuerzas, se sumarán en las posiciones i, j y k de dicho vector:

$$\begin{matrix} \\ \\ i\cdots \\ \\ j\cdots \\ \\ k\cdots \\ \\ \\ \\ \end{matrix} \begin{bmatrix} f_1 \\ \vdots \\ f_i \\ \vdots \\ f_j \\ \vdots \\ f_k \\ \vdots \\ f_N \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & K_{ii} & \cdots & K_{ij} & \cdots & K_{ik} & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & K_{ji} & \cdots & K_{jj} & \cdots & K_{jk} & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & K_{ki} & \cdots & K_{kk} & \cdots & K_{kj} & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ & & \vdots & & \vdots & & \vdots & & \\ & & i & & j & & k & & \end{bmatrix} \begin{bmatrix} d_1 \\ \vdots \\ d_i \\ \vdots \\ d_j \\ \vdots \\ d_k \\ \vdots \\ d_N \end{bmatrix} \tag{3.58}$$

3.2.4. Matriz de rigidez de elementos cuadriláteros

En lugar de implementar la formulación correspondiente a los elementos superficiales con forma de cuadrilátero, se ha optado por dividir a dicho elemento por sus dos diagonales, lo cual da lugar a 4 triángulos que se procesan, independientemente, como elementos triangulares que son, promediando el resultado. En consecuencia, esto supone que si generamos la matriz de un triángulo y éste proviene de un cuadrilátero, multiplicaremos por 0.5 la matriz obtenida. La figura

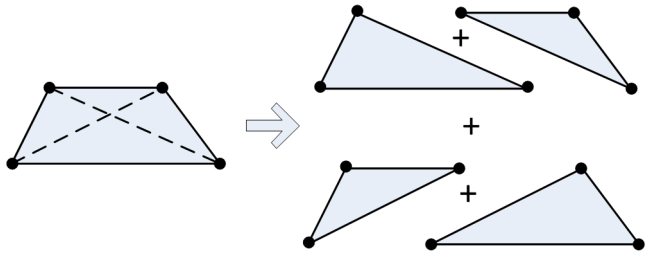


Figura 3.8: División de un cuadrilátero en 4 triángulos.

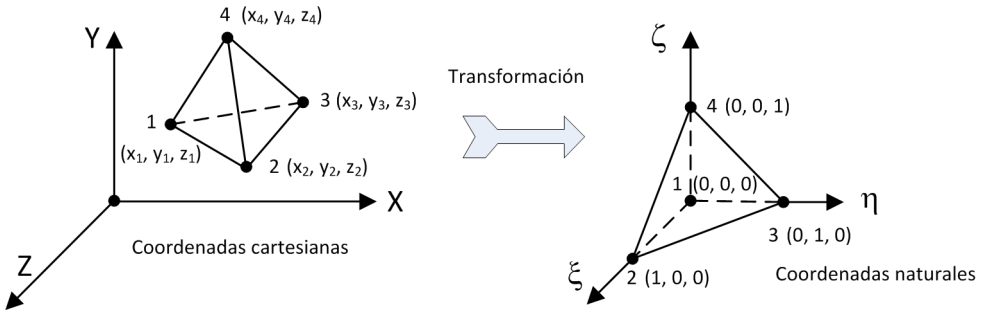


Figura 3.9: Transformación de las coordenadas cartesianas espaciales de un tetraedro a sus coordenadas naturales.

3.8 muestra el proceso a seguir en la descomposición de un cuadrilátero en los 4 mencionados triángulos.

3.2.5. Matriz de rigidez de elementos tetraédricos

El tetraedro utilizado en esta tesis doctoral ha sido el de 4 nodos, con 3 grados de libertad por nodo, recogiendo los movimientos longitudinales pero no los giros de los mismos. A pesar de su aparente mayor dificultad, su formulación sigue la línea de los elementos bidimensionales.

Consideraremos que los nodos del tetraedro presentan, como numeración global, los índices *i*, *j*, *k* y *l*, la cual se corresponderá con la numeración local *1*, *2*, *3* y *4* a nivel del propio elemento. Si bien dicha numeración local la emplearemos para generar la matriz de rigidez del elemento, recurriremos a la numeración global a la hora de realizar su ensamblaje. Emplearemos además la formulación isoparamétrica, en la cual la geometría del elemento quedará normalizada, de acuerdo a sus coordenadas naturales, tal y como se muestra en la figura 3.9. Las funciones

de forma del tetraedro isoparamétrico son las siguientes:

$$\begin{aligned} N_1(\xi,\eta,\zeta) &= 1-\xi-\eta-\zeta \\ N_2(\xi,\eta,\zeta) &= \xi \\ N_3(\xi,\eta,\zeta) &= \eta \\ N_4(\xi,\eta,\zeta) &= \zeta \end{aligned} \tag{3.59}$$

Estas funciones pueden usarse para obtener las coordenadas cartesianas de un punto, a partir de sus coordenadas naturales:

$$\begin{aligned} x &= \sum_{i=1}^{4} N_i(\xi,\eta,\zeta)\, x_i = x_1 + \xi x_{21} + \eta x_{31} + \zeta x_{41} \\ y &= \sum_{i=1}^{4} N_i(\xi,\eta,\zeta)\, y_i = y_1 + \xi y_{21} + \eta y_{31} + \zeta y_{41} \\ z &= \sum_{i=1}^{4} N_i(\xi,\eta,\zeta)\, z_i = z_1 + \xi z_{21} + \eta z_{31} + \zeta z_{41} \end{aligned} \tag{3.60}$$

habiendo empleando las siguientes diferencias entre las coordenadas cartesianas de los nudos:

$$\begin{aligned} x_{ij} &= x_i - x_j \\ y_{ij} &= y_i - y_j \\ z_{ij} &= z_i - z_j, \quad i,j = 1,2,3,4 \end{aligned} \tag{3.61}$$

De igual modo, podemos obtener el desplazamiento d en un punto cualquiera del tetraedro a partir de sus coordenadas naturales y de los desplazamientos en sus nodos:

$$d = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \sum_{i=1}^{4} \mathrm{N_i} d_i^e = \mathrm{N} \begin{bmatrix} d_1^e \\ d_2^e \\ d_3^e \\ d_4^e \end{bmatrix} = \mathrm{N} d^e \tag{3.62}$$

donde:

$$\mathrm{N} = \begin{bmatrix} \mathrm{N_1} & \mathrm{N_2} & \mathrm{N_3} & \mathrm{N_4} \end{bmatrix}, \ \mathrm{N_i} = \begin{bmatrix} N_i(\xi,\eta,\zeta) & 0 & 0 \\ 0 & N_i(\xi,\eta,\zeta) & 0 \\ 0 & 0 & N_i(\xi,\eta,\zeta) \end{bmatrix} \tag{3.63}$$

Si calculamos las derivadas parciales de dichas funciones de forma con respecto a

las coordenadas naturales, obtendremos que:

$$\begin{aligned}
&\frac{\partial N_1}{\partial \xi} = -1, \quad \frac{\partial N_2}{\partial \xi} = 1, \quad \frac{\partial N_3}{\partial \xi} = 0, \quad \frac{\partial N_4}{\partial \xi} = 0 \\
&\frac{\partial N_1}{\partial \eta} = -1, \quad \frac{\partial N_2}{\partial \eta} = 0, \quad \frac{\partial N_3}{\partial \eta} = 1, \quad \frac{\partial N_4}{\partial \eta} = 0 \\
&\frac{\partial N_1}{\partial \zeta} = -1, \quad \frac{\partial N_2}{\partial \zeta} = 0, \quad \frac{\partial N_3}{\partial \zeta} = 0, \quad \frac{\partial N_4}{\partial \zeta} = 1
\end{aligned} \tag{3.64}$$

Si por otro lado pretendemos obtener las derivadas de dichas funciones de forma con respecto a las coordenadas cartesianas, tendremos que resolver estos otros siguientes sistemas de ecuaciones lineales:

$$\begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{bmatrix}, \quad i = 1, 2, 3, 4 \tag{3.65}$$

siendo $J \in \mathbb{R}^{3x3}$ la matriz de coeficientes del sistema o matriz Jacobiana del elemento, expresada en función de las diferencias entre las coordenadas locales:

$$J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} = \sum_{i=1}^{4} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} x_i & \frac{\partial N_i}{\partial \xi} y_i & \frac{\partial N_i}{\partial \xi} z_i \\ \frac{\partial N_i}{\partial \eta} x_i & \frac{\partial N_i}{\partial \eta} y_i & \frac{\partial N_i}{\partial \eta} z_i \\ \frac{\partial N_i}{\partial \zeta} x_i & \frac{\partial N_i}{\partial \zeta} y_i & \frac{\partial N_i}{\partial \zeta} z_i \end{bmatrix} = \begin{bmatrix} x_{21} & y_{21} & z_{21} \\ x_{31} & y_{31} & z_{31} \\ x_{41} & y_{31} & z_{41} \end{bmatrix} \tag{3.66}$$

Dichos derivadas parciales se podrán calcular empleando la inversa de la matriz Jacobiana, la cual es sencilla de obtener. Una vez que hayamos calculado las citadas derivadas parciales, quedará allanado el camino para generar la matriz $B \in \mathbb{R}^{6x12}$ de deformación del elemento:

$$B = \begin{bmatrix} B_1 & B_2 & B_3 & B_4 \end{bmatrix} \tag{3.67}$$

compuesta por estas 4 submatrices $B_i \in \mathbb{R}^{6x3}$:

$$B_i = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_i}{\partial z} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & 0 \\ \frac{\partial N_i}{\partial z} & 0 & \frac{\partial N_i}{\partial x} \\ 0 & \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial y} \end{bmatrix}, \; i = 1, 2, 3, 4 \tag{3.68}$$

A partir de la expresión (2.20) es conocido que la matriz de rigidez $K^e \in \mathbb{R}^{12x12}$ del elemento tetraédrico e la calcularemos del siguiente modo:

$$K^e = \int_{V^e} B^T D B d\nu =$$

$$= \int_0^1 \int_0^{1-\eta} \int_0^{1-\eta-\zeta} B\left(\xi, \eta, \zeta\right)^T D B\left(\xi, \eta, \zeta\right) \left|J\left(\xi, \eta, \zeta\right)\right| \, d\xi \; d\eta \; d\zeta \tag{3.69}$$

donde la matriz $D \in \mathbb{R}^{6x6}$ representa la matriz constitutiva del material, formada por su módulo de elasticidad longitudinal E y su coeficiente de Poisson υ:

$$D = \frac{E}{(1+\upsilon)(1-2\upsilon)} \begin{bmatrix} 1-\upsilon & \upsilon & \upsilon & 0 & 0 & 0 \\ \upsilon & 1-\upsilon & \upsilon & 0 & 0 & 0 \\ \upsilon & \upsilon & 1-\upsilon & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}-\upsilon & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}-\upsilon & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}-\upsilon \end{bmatrix} \tag{3.70}$$

Obsérvese que en la integral aparecen las propias variables en los límites de integración, lo que dificulta su resolución numérica. Empleando 4 puntos de integración y siendo los valores de B y D son constantes en cada punto, resulta que:

$$K^e = V B^T D B \tag{3.71}$$

siendo V el volumen del tetraedro, calculado a partir del determinante de la matriz

Jacobiana del elemento:

$$V = \int_0^1 \int_0^{1-\eta} \int_0^{1-\eta-\zeta} |J(\xi,\eta,\zeta)| d\xi d\eta d\zeta = \frac{1}{6}|J| \tag{3.72}$$

Conviene aclarar que las coordenadas cartesianas empleadas en la transformación isoparamétrica del elemento estaban expresadas en ejes globales. Por ese motivo, la matriz K^e de rigidez está ya formulada en dichos ejes. No en vano, los elementos tetraédricos carecen de un sistema de coordenadas local propio.

Si la numeración global de los nudos del elemento es la *i*, *j*, *k* y *l*, la cual se corresponde, como ya dijimos, con los nudos locales *1*, *2*, *3* y *4* respectivamente, la matriz de rigidez del tetraedro estará formada por las siguientes submatrices:

$$K^e = \begin{bmatrix} K_{ii} & K_{ij} & K_{ik} & K_{il} \\ K_{ji} & K_{jj} & K_{jk} & K_{jl} \\ K_{ki} & K_{kj} & K_{kk} & K_{kl} \\ K_{li} & K_{lj} & K_{lk} & K_{ll} \end{bmatrix} \tag{3.73}$$

donde cada una de ellas, como por ejemplo la K_{ij}, se podrá obtener de manera independiente al resto, del siguiente modo:

$$K_{ij} = V B_i^T D B_j \tag{3.74}$$

Con todo ello, la relación entre fuerzas y desplazamientos en los nodos del tetraedro vendrá dada por la siguiente expresión, ya conocida:

$$\begin{bmatrix} f_i^e \\ f_j^e \\ f_k^e \\ f_l^e \end{bmatrix} = K^e \begin{bmatrix} d_i^e \\ d_j^e \\ d_k^e \\ d_l^e \end{bmatrix} = \begin{bmatrix} K_{ii} & K_{ij} & K_{ik} & K_{il} \\ K_{ji} & K_{jj} & K_{jk} & K_{jl} \\ K_{ki} & K_{kj} & K_{kk} & K_{kl} \\ K_{li} & K_{lj} & K_{lk} & K_{ll} \end{bmatrix} \begin{bmatrix} d_i^e \\ d_j^e \\ d_k^e \\ d_l^e \end{bmatrix} \tag{3.75}$$

El ensamblaje de la matriz de rigidez del elemento y de las fuerzas que actúan sobre sus nodos a la matriz de rigidez de la estructura y al vector de fuerzas se realizará de manera similar a lo que ya hemos comentado para las barras y para los triángulos. Así, la aportación de la matriz de rigidez de un tetraedro a la matriz de rigidez completa se realizará sumando sus submatrices en las filas

y columnas correspondientes a sus nudos i, j, k y l. Del mismo modo, las cargas aplicadas sobre los nudos del elemento se sumarán en las posiciones del vector de fuerzas correspondientes a la numeración de los nodos:

$$
\begin{array}{c}
\\
\vdots \\
...i \\
\vdots \\
...j \\
\vdots \\
...k \\
\vdots \\
...l \\
\\
\\
\\
\end{array}
\begin{bmatrix} f_1 \\ \vdots \\ f_i \\ \vdots \\ f_j \\ \vdots \\ f_k \\ \vdots \\ f_l \\ \vdots \\ f_N \end{bmatrix}
=
\begin{bmatrix}
0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
0 & \cdots & K_{ii} & \cdots & K_{ij} & \cdots & K_{ik} & \cdots & K_{il} & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
0 & \cdots & K_{ji} & \cdots & K_{jj} & \cdots & K_{jk} & \cdots & K_{jl} & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
0 & \cdots & K_{ki} & \cdots & K_{kj} & \cdots & K_{kk} & \cdots & K_{kl} & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
0 & \cdots & K_{li} & \cdots & K_{lj} & \cdots & K_{lk} & \cdots & K_{ll} & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
 & & \vdots & & \vdots & & \vdots & & \vdots & & \\
 & & i & & j & & k & & l & &
\end{bmatrix}
\begin{bmatrix} d_1 \\ \vdots \\ d_i \\ \vdots \\ d_j \\ \vdots \\ d_k \\ \vdots \\ d_l \\ \vdots \\ d_N \end{bmatrix}
\tag{3.76}
$$

3.2.6. Matriz de rigidez de prismas triangulares

La opción elegida para tratar a los prismas triangulares, formados por 6 nodos y 3 grados de libertad en cada nodo, ha sido la de dividir a dichos elementos en 3 tetraedros, tal y como se muestra en la figura 3.10, con los cuales se trabaja de manera independiente. Puesto que no hay ningún tipo de solapado entre los mismos, todos ellos se tratarán de manera idéntica a si fueran elementos originales del mallado y el prisma triangular no hubiera existido. Eso supone que la matriz de rigidez de cada uno de los tres tetraedros se generará y se ensamblará de manera idéntica a lo recogido en la sección anterior.

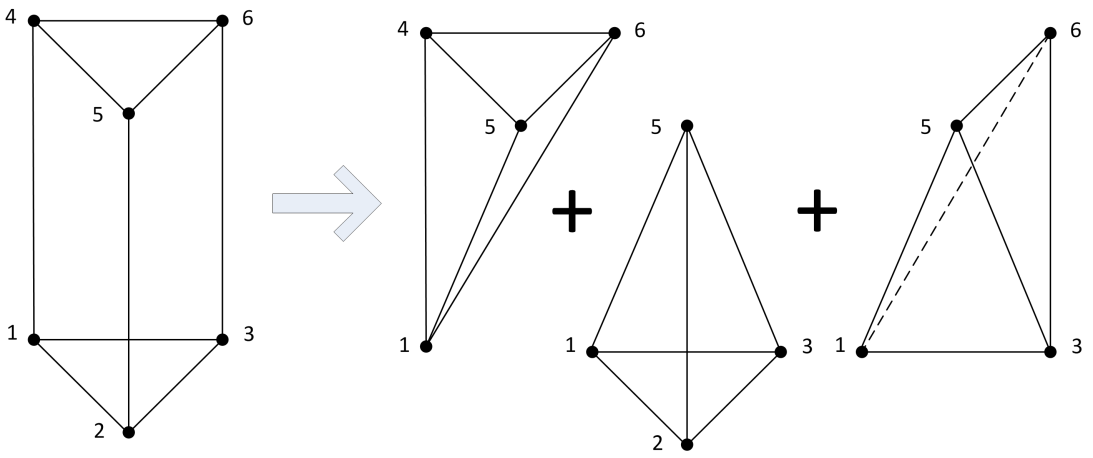


Figura 3.10: División de un prisma triangular en 3 tetraedros.

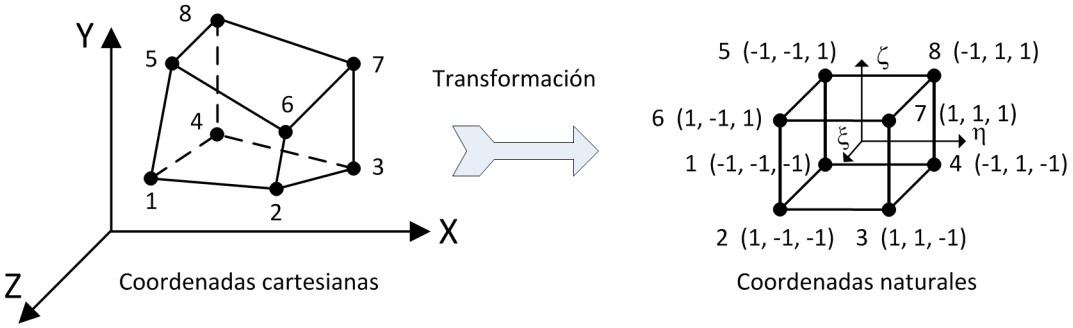


Figura 3.11: Transformación de las coordenadas cartesianas espaciales de un hexaedro a sus coordenadas naturales.

3.2.7. Matriz de rigidez de elementos hexaédricos

El tipo de hexaedro escogido en esta tesis doctoral ha sido el formado por 8 nodos, con 3 grados de libertad que recogen los desplazamientos longitudinales de cada uno de ellos. Al igual que ocurría con el tetraedro, no tendremos en consideración los giros de los nodos.

Los nodos del hexaedro seguirán una numeración local del 1 al 8. Además, emplearemos la formulación isoparamétrica, normalizando la geometría del elemento de acuerdo a sus coordenadas naturales, como se recoge en la figura 3.11. Las funciones de forma del hexaedro isoparamétrico son las siguientes:

$$
\begin{aligned}
N_1(\xi,\eta,\zeta) &= \tfrac{1}{8}(1-\xi)(1-\eta)(1-\zeta) \\
N_2(\xi,\eta,\zeta) &= \tfrac{1}{8}(1+\xi)(1-\eta)(1-\zeta) \\
N_3(\xi,\eta,\zeta) &= \tfrac{1}{8}(1+\xi)(1+\eta)(1-\zeta) \\
N_4(\xi,\eta,\zeta) &= \tfrac{1}{8}(1-\xi)(1+\eta)(1-\zeta) \\
N_5(\xi,\eta,\zeta) &= \tfrac{1}{8}(1-\xi)(1-\eta)(1+\zeta) \\
N_6(\xi,\eta,\zeta) &= \tfrac{1}{8}(1+\xi)(1-\eta)(1+\zeta) \\
N_7(\xi,\eta,\zeta) &= \tfrac{1}{8}(1+\xi)(1+\eta)(1+\zeta) \\
N_8(\xi,\eta,\zeta) &= \tfrac{1}{8}(1-\xi)(1+\eta)(1+\zeta)
\end{aligned}
\tag{3.77}
$$

las cuales nos permiten obtener las coordenadas cartesianas de un punto, a partir de sus coordenadas naturales:

$$
\begin{aligned}
x &= \sum_{i=1}^{8} N_i(\xi,\eta,\zeta)\, x_i \\
y &= \sum_{i=1}^{8} N_i(\xi,\eta,\zeta)\, y_i \\
z &= \sum_{i=1}^{8} N_i(\xi,\eta,\zeta)\, z_i
\end{aligned}
\tag{3.78}
$$

así como obtener el desplazamiento d en un punto cualquiera del elemento a partir

de sus coordenadas naturales y de los desplazamientos en sus nodos:

$$d = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \sum_{i=1}^{8} \mathrm{N_i} d_i^e = \mathrm{N} \begin{bmatrix} d_1^e \\ d_2^e \\ d_3^e \\ d_4^e \\ d_5^e \\ d_6^e \\ d_7^e \\ d_8^e \end{bmatrix} = \mathrm{N} d^e \tag{3.79}$$

siendo N la matriz formada por las mencionadas funciones de forma:

$$\mathrm{N} = \begin{bmatrix} \mathrm{N_1} & \mathrm{N_2} & ... & \mathrm{N_8} \end{bmatrix}, \ \mathrm{N_i} = \begin{bmatrix} N_i(\xi,\eta,\zeta) & 0 & 0 \\ 0 & N_i(\xi,\eta,\zeta) & 0 \\ 0 & 0 & N_i(\xi,\eta,\zeta) \end{bmatrix} \tag{3.80}$$

Si derivamos estas funciones con respecto a sus coordenadas naturales, llegamos a que:

$$\begin{array}{lll}
\frac{\partial N_1}{\partial \xi} = -\frac{1}{8}(1-\eta)(1-\zeta) & \frac{\partial N_1}{\partial \eta} = -\frac{1}{8}(1-\xi)(1-\zeta) & \frac{\partial N_1}{\partial \zeta} = -\frac{1}{8}(1-\xi)(1-\eta) \\
\frac{\partial N_2}{\partial \xi} = \frac{1}{8}(1-\eta)(1-\zeta) & \frac{\partial N_2}{\partial \eta} = -\frac{1}{8}(1+\xi)(1-\zeta) & \frac{\partial N_2}{\partial \zeta} = -\frac{1}{8}(1+\xi)(1-\eta) \\
\frac{\partial N_3}{\partial \xi} = \frac{1}{8}(1+\eta)(1-\zeta) & \frac{\partial N_3}{\partial \eta} = \frac{1}{8}(1+\xi)(1-\zeta) & \frac{\partial N_3}{\partial \zeta} = -\frac{1}{8}(1+\xi)(1+\eta) \\
\frac{\partial N_4}{\partial \xi} = -\frac{1}{8}(1+\eta)(1-\zeta) & \frac{\partial N_4}{\partial \eta} = \frac{1}{8}(1-\xi)(1-\zeta) & \frac{\partial N_4}{\partial \zeta} = -\frac{1}{8}(1-\xi)(1+\eta) \\
\frac{\partial N_5}{\partial \xi} = -\frac{1}{8}(1-\eta)(1+\zeta) & \frac{\partial N_5}{\partial \eta} = -\frac{1}{8}(1-\xi)(1+\zeta) & \frac{\partial N_5}{\partial \zeta} = \frac{1}{8}(1-\xi)(1-\eta) \\
\frac{\partial N_6}{\partial \xi} = \frac{1}{8}(1-\eta)(1+\zeta) & \frac{\partial N_6}{\partial \eta} = -\frac{1}{8}(1+\xi)(1+\zeta) & \frac{\partial N_6}{\partial \zeta} = \frac{1}{8}(1+\xi)(1-\eta) \\
\frac{\partial N_7}{\partial \xi} = \frac{1}{8}(1+\eta)(1+\zeta) & \frac{\partial N_7}{\partial \eta} = \frac{1}{8}(1+\xi)(1+\zeta) & \frac{\partial N_7}{\partial \zeta} = \frac{1}{8}(1+\xi)(1+\eta) \\
\frac{\partial N_8}{\partial \xi} = -\frac{1}{8}(1+\eta)(1+\zeta) & \frac{\partial N_8}{\partial \eta} = \frac{1}{8}(1-\xi)(1+\zeta) & \frac{\partial N_8}{\partial \zeta} = \frac{1}{8}(1-\xi)(1+\eta)
\end{array} \tag{3.81}$$

Por último, las derivadas con respecto a las coordenadas espaciales las obtendremos gracias a la resolución de los siguientes sistemas de ecuaciones lineales:

$$\begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{bmatrix}, \quad i = 1, .2, ..., 8 \tag{3.82}$$

donde $J \in \mathbb{R}^{3x3}$ representa la matriz Jacobiana del elemento:

$$J = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} = \sum_{i=1}^{8} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} x_i & \frac{\partial N_i}{\partial \xi} y_i & \frac{\partial N_i}{\partial \xi} z_i \\ \frac{\partial N_i}{\partial \eta} x_i & \frac{\partial N_i}{\partial \eta} y_i & \frac{\partial N_i}{\partial \eta} z_i \\ \frac{\partial N_i}{\partial \zeta} x_i & \frac{\partial N_i}{\partial \zeta} y_i & \frac{\partial N_i}{\partial \zeta} z_i \end{bmatrix} \tag{3.83}$$

Con todo ello, concluimos que la matriz $B \in \mathbb{R}^{6x24}$ de deformación del elemento es:

$$B = \begin{bmatrix} B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & B_7 & B_8 \end{bmatrix} \tag{3.84}$$

formada por estas 8 submatrices $B_i \in \mathbb{R}^{6x3}$:

$$B_i = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_i}{\partial z} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & 0 \\ \frac{\partial N_i}{\partial z} & 0 & \frac{\partial N_i}{\partial x} \\ 0 & \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial y} \end{bmatrix}, \; i = 1, 2, ..., 8 \tag{3.85}$$

Una vez determinada esta matriz de deformación B y conocida también la matriz $D \in \mathbb{R}^{6x6}$ constitutiva del material, idéntica a la indicada en la expresión (3.70), podremos calcular la matriz de rigidez $K^e \in \mathbb{R}^{24x24}$ de un elemento hexaédrico e:

$$K^e = \int_{V^e} B^T D B d\nu =$$

$$= \int_{-1}^{1} \int_{-1}^{1} \int_{-1}^{1} B\left(\xi, \eta, \zeta\right)^T D B\left(\xi, \eta, \zeta\right) \left| J\left(\xi, \eta, \zeta\right) \right| \, d\xi \; d\eta \; d\zeta \tag{3.86}$$

Empleando la aproximación de Gauss-Legendre con 2 puntos por cada dirección, los cuales tienen como raíces los valores $\pm\frac{1}{\sqrt{3}}$ y un factor de peso ω igual a 1, llegaremos a que:

$$K^e = \sum_{i=1}^{2} \sum_{j=1}^{2} \sum_{k=1}^{2} B\left(\xi_i, \eta_j, \zeta_k\right)^T D B\left(\xi_i, \eta_j, \zeta_k\right) \left| J\left(\xi_i, \eta_j, \zeta_k\right) \right| \tag{3.87}$$

Resulta conveniente aclarar que, para calcular la matriz B en función de las coordenadas naturales que aparecen en la expresión anterior, hay que recalcular

p	ξ	η	ζ	ω
1	-r	-r	-r	1
2	r	-r	-r	1
3	r	r	-r	1
4	-r	r	-r	1
5	-r	-r	r	1
6	r	-r	r	1
7	r	r	r	1
8	-r	r	r	1

Tabla 3.2: Coordenadas naturales de los 8 puntos de integración empleados en los elementos hexaédricos.

el valor las derivadas de las funciones de forma con respecto a las coordenadas locales indicadas, para luego obtener la matriz Jacobiana y sus derivadas parciales con respecto a las coordenadas cartesianas. En realidad, podemos expresar los tres sumatorios que aparecen en la integral en uno sólo, visitando cada punto de integración y empleando las coordenadas naturales de cada uno de ellos, recogidas en la tabla 3.2, donde $r = \frac{1}{\sqrt{3}}$:

$$K^e = \sum_{p=1}^{8} B\left(\xi_p, \eta_p, \zeta_p\right)^T DB\left(\xi_p, \eta_p, \zeta_p\right) \left|J\left(\xi_p, \eta_p, \zeta_p\right)\right| \tag{3.88}$$

Si particularizamos para cada submatriz K_{ij} de la matriz K^e tendremos que:

$$K_{ij} = \sum_{p=1}^{8} B_i\left(\xi_p, \eta_p, \zeta_p\right)^T DB_j\left(\xi_p, \eta_p, \zeta_p\right) \left|J\left(\xi_p, \eta_p, \zeta_k\right)\right|,\ i, j = 1, 2, ..., 8 \tag{3.89}$$

Identificada la matriz de rigidez, la relación entre fuerzas y desplazamientos en los nodos del hexaedro será:

$$f^e = K^e d^e \tag{3.90}$$

y únicamente nos quedará pendiente realizar el ensamblaje de la matriz de rigidez y el vector de fuerzas en las posiciones correspondientes a los nudos del hexaedro.

3.3. Generación del vector de cargas

3.3.1. Generalidades

Hasta el momento, las estructuras analizadas sólo podrán presentar cargas externas aplicadas directamente sobre los nudos, de manera que todas aquellas cargas exteriores que actúan sobre un nudo se suman en su componente correspondiente del vector de fuerzas. Sin embargo, lo habitual en estructuras de edificación es que aparezcan no sólo cargas aplicadas sobre los nudos, sino también sobre las barras y sobre los elementos finitos que la componen.

3.3.2. Cargas aplicadas sobre las barras

Las cargas aplicadas sobre las barras de la estructura podrán ser puntuales o repartidas. Una carga puntual se caracteriza por su punto de aplicación sobre la barra, además de su dirección y su magnitud. En cambio, una carga repartida queda definida por su punto inicial y final de aplicación dentro de la barra, su dirección y sus valores inicial y final. Del mismo modo, las cargas podrán ser de fuerzas, cuando se aplican longitudinalmente sobre los ejes locales de la barra, o de momentos, cuando se aplican en forma de giros sobre dichos ejes.

Para obtener el comportamiento de la estructura ante cargas actuando sobre las barras debemos llevar a cabo dos pasos, recurriendo al principio de superposición. En primer lugar, llevaremos a la estructura a la condición de empotramiento perfecto, entendiendo por tanto que ninguno de sus nudos puede moverse. Las cargas aplicadas sobre dichas barras son transmitidas a sus nudos, desarrollando en los mismos unas reacciones, denominadas esfuerzos de empotramiento perfecto. Si suponemos una barra b, con nudos i, j, en la cual actúa una carga y bajo el supuesto del impedimento de movimiento de los nudos, los esfuerzos en los extremos de la barra serán exactamente los de empotramiento perfecto:

$$\begin{bmatrix} f_i'^b \\ f_j'^b \end{bmatrix} = \begin{bmatrix} p_i'^b \\ p_j'^b \end{bmatrix} \tag{3.91}$$

Si a estos esfuerzos de empotramiento les cambiamos el signo, tendremos las denominadas fuerzas nodales equivalentes a las cargas que actúan sobre las barras. Estas acciones o fuerzas nodales equivalentes en un nudo estarán por tanto formadas por la suma de los esfuerzos de empotramiento de todas las nb barras que a él concurren, como a continuación se indica, cambiados de signo:

$$p_i = \sum_b^{nb} p_i^b = \sum_b^{nb} R^b p_i'^b \tag{3.92}$$

En segundo lugar, supondremos que no actúan cargas sobre ninguna barra de la estructura. Bajo esta premisa, los únicos esfuerzos que aparecerán en los extremos de las barras serán debidos a los movimientos de los nudos libres de la estructura:

$$\begin{bmatrix} f_i'^b \\ f_j'^b \end{bmatrix} = K'^b \begin{bmatrix} d_i'^b \\ d_j'^b \end{bmatrix} = \begin{bmatrix} K'_{ii} & K'_{ij} \\ K'_{ji} & K'_{jj} \end{bmatrix} \begin{bmatrix} d_i'^b \\ d_j'^b \end{bmatrix} \tag{3.93}$$

Considerando que un nudo cualquiera de la estructura recibirá una carga equivalente a la suma de todas las acciones particulares que posea, los desplazamientos en los nudos y los esfuerzos en los extremos de las barras serán por tanto la superposición de ambos estados.

Esto supone, por tanto, que debemos analizar la estructura con las cargas externas aplicadas directamente sobre los nudos junto con las acciones equivalentes sobre los mismos, lo cual da lugar a que la relación entre los esfuerzos en los extremos de una barra y los desplazamientos en sus extremos sea en realidad:

$$\begin{bmatrix} f_i'^b \\ f_j'^b \end{bmatrix} = \begin{bmatrix} p_i'^b \\ p_j'^b \end{bmatrix} + K'^b \begin{bmatrix} d_i'^b \\ d_j'^b \end{bmatrix} = \begin{bmatrix} p_i'^b \\ p_j'^b \end{bmatrix} + \begin{bmatrix} K'_{ii} & K'_{ij} \\ K'_{ji} & K'_{jj} \end{bmatrix} \begin{bmatrix} d_i'^b \\ d_j'^b \end{bmatrix} \tag{3.94}$$

donde $p_i'^b$ y $p_j'^b$ representan los esfuerzos de empotramiento perfecto de la barra expresados en ejes locales, o lo que es lo mismo en ejes globales:

$$\begin{bmatrix} f_i^b \\ f_j^b \end{bmatrix} = \begin{bmatrix} R^b & 0 \\ 0 & R^b \end{bmatrix} \begin{bmatrix} p_i'^b \\ p_j'^b \end{bmatrix} + K_b \begin{bmatrix} d_i^b \\ d_j^b \end{bmatrix} = \begin{bmatrix} p_i^b \\ p_j^b \end{bmatrix} + \begin{bmatrix} K_{ii} & K_{ij} \\ K_{ji} & K_{jj} \end{bmatrix} \begin{bmatrix} d_i^b \\ d_j^b \end{bmatrix} \tag{3.95}$$

dando lugar al correspondiente ensamblaje de la barra b en la ecuación de equilibrio de la estructura:

$$
\begin{matrix} \\ i\cdots \\ \\ j\cdots \\ \\ \end{matrix}
\begin{bmatrix} 0 \\ \vdots \\ f_i^b \\ \vdots \\ f_j^b \\ \vdots \\ 0 \end{bmatrix}
=
\begin{bmatrix} 0 \\ \vdots \\ p_i^b \\ \vdots \\ p_j^b \\ \vdots \\ 0 \end{bmatrix}
+
\begin{bmatrix}
0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots \\
0 & \cdots & K_{ii} & \cdots & K_{ij} & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots \\
0 & \cdots & K_{ji} & \cdots & K_{jj} & \cdots & 0 \\
\vdots & & \vdots & & \vdots & & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
 & & \vdots & & \vdots & & \\
 & & i & & j & &
\end{bmatrix}
\begin{bmatrix} d_1 \\ \vdots \\ d_i \\ \vdots \\ d_j \\ \vdots \\ d_N \end{bmatrix}
\tag{3.96}
$$

En consecuencia, dado un nudo i cualquiera, la componente i del vector F global de fuerzas estará compuesta por la suma de todas aquellas cargas externas que actúan directamente sobre el nudo menos la suma de todos aquellos esfuerzos de empotramiento que sobre él transmiten el conjunto de barras que lo unen con el resto de la estructura. De esta forma, la matriz F de fuerzas sobre los nudos generada en ejes globales se obtendrá del siguiente modo:

$$F = F_{_N} - P \tag{3.97}$$

donde $F_N \in \mathbb{R}^{nxm}$ se corresponde con la matriz compuesta por las cargas actuando directamente sobre los nudos de la estructura en ejes globales y $P \in \mathbb{R}^{nxm}$ es la llamada matriz de esfuerzos de empotramiento perfecto también en ejes globales, generada, como hemos comentado, a partir de las cargas aplicadas sobre las barras de la estructura.

En los apartados siguientes, los esfuerzos de empotramiento en el extremo i de una barra b vendrán dados por el siguiente vector, expresados siempre en ejes

locales, los cuales se calcularán de acuerdo a la formulación proporcionada en [32]:

$$p_i'^b = \begin{bmatrix} F'_{ix} \\ F'_{iy} \\ F'_{iz} \\ M'_{ix} \\ M'_{iy} \\ M'_{iz} \end{bmatrix} \tag{3.98}$$

Comenzando del primero al último, a estos esfuerzos se les denominan *axiles*, *cortantes* en el eje Y o en el eje Z, *momento torsor* y *momentos flectores* en Y o en Z. El esfuerzo axil en una barra es aquel que ésta experimenta cuando actúa sobre ella una fuerza paralela a su directriz. Es por tanto un esfuerzo normal o perpendicular a la sección de la barra, que da lugar a un alargamiento (tracción) o acortamiento (compresión) de la longitud. Los esfuerzos cortantes aparecen al aplicar fuerzas en el plano de la sección de la barra, siendo por tanto tangenciales a la misma. El momento torsor es debido al giro de la barra sobre sí misma, dando lugar a un efecto de torsión. Por último, los momentos flectores, o de flexión, son debidos al giro lateral de la barra, la cual pasa a curvarse en los planos XY o XZ.

A modo de resumen, diremos que:

- Las cargas de fuerza aplicadas en el eje X generan esfuerzos axiles, aplicadas en el eje Y dan lugar a esfuerzos cortantes en Y y momentos flectores en Z y aplicadas en el eje Z producen esfuerzos cortantes en Z y momentos flectores en Y.
- Las cargas de momento torsor dan lugar momentos torsores.
- Las cargas de momento flector en Y producen esfuerzos cortantes en Z y momentos flectores en Y, mientas que las cargas de momento flector en Z dan lugar a esfuerzos cortantes en Y y momentos flectores en Z.

La figura 3.12 recoge el convenio de signos seguidos en las cargas de fuerza y momento aplicadas, así como los esfuerzos y momentos de empotramiento perfecto a los que dan lugar. Como puede observarse, las fuerzas y los esfuerzos de

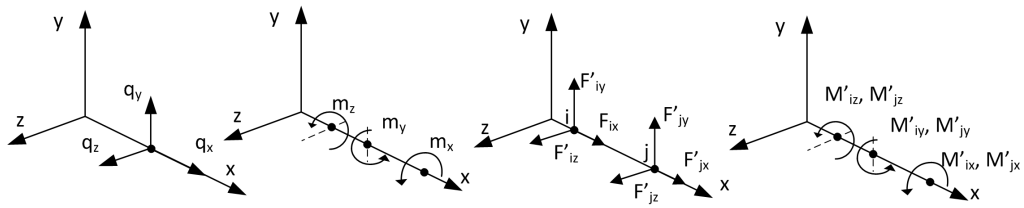


Figura 3.12: Convención de signos en cargas y esfuerzos de empotramiento.

empotramiento son positivos cuando se aplican o se producen en el mismo sentido que el de los ejes de la barra. Por otro lado, los momentos de carga y los momentos se esfuerzos producidos son positivos cuando son contrarios a las agujas del reloj.

Como anteriormente decíamos, habrá que transformar a globales dichos esfuerzos de empotramiento expresados en los ejes locales de la barra antes de ensamblar su aportación al vector P, de acuerdo a la siguiente expresión:

$$\begin{bmatrix} p_i^b \\ p_j^b \end{bmatrix} = \begin{bmatrix} R^b & 0 \\ 0 & R^b \end{bmatrix} \begin{bmatrix} p_i'^b \\ p_j'^b \end{bmatrix} \tag{3.99}$$

3.3.2.1. Cargas de fuerzas puntuales en barras

Supongamos una barra biapoyada en sus nudos extremos (lo que significa que tendrá impedidos sus desplazamientos, pero no sus giros) a la cual se le aplican cargas de fuerza puntuales con valores q_x, q_y y q_z sobre cada uno de sus ejes locales X, Y y Z, respectivamente, como muestra la figura 3.13.

Inicialmente, la carga q_x generará los siguientes esfuerzos axiles en los extremos de la barra:

$$F'_{ix} = \frac{-q_x b}{L}, \quad F'_{jx} = \frac{-q_x a}{L} \tag{3.100}$$

Puesto que los giros son libres, calculamos los ángulos de dichos giros, en los ejes Z e Y, en los extremos de la barra, generados por las cargas q_y y q_z multiplicados por EI_z o EI_y:

$$\varphi'_{iz} = \frac{q_y ab(L+b)}{6L}, \; \varphi'_{jz} = \frac{-q_y ab(L+a)}{6L} \tag{3.101}$$

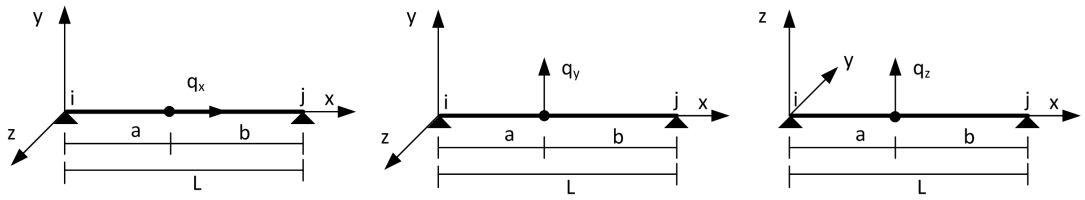


Figura 3.13: Cargas puntuales actuando sobre los ejes locales de una barra.

$$\varphi'_{iy} = \frac{-q_z ab(L+b)}{6L},\ \varphi'_{jy} = \frac{q_z ab(L+a)}{6L} \tag{3.102}$$

En función de estos ángulos y de los posibles grados de rigidez a giro en Y y en Z que puedan existir entre el extremo de la barra y el nudo, determinaremos los momentos flectores:

$$M'_{iy} = -\frac{6\varphi_{jy}Gr_{iy}Gr_{jy} + 12\varphi_{iy}Gr_{iy}}{L*(4-Gr_{iy}Gr_{jy})}$$

$$M'_{jy} = \frac{6\varphi_{iy}}{L} - \frac{12*\varphi_{jy}Gr_{jy} + 24\varphi_{iy}}{L*(4-Gr_{iy}Gr_{jy})} \tag{3.103}$$

$$M'_{iz} = -\frac{6\varphi'_{jz}Gr_{iz}Gr^z_j + 12\varphi'_{iz}Gr_{iz}}{L(4-Gr_{iz}Gr_{jz})}$$

$$M'_{jz} = \frac{6\varphi'_{iz}}{L} - \frac{12\varphi'_{jz}Gr_{jz} + 24\varphi'_{iz}}{L(4-Gr_{iz}Gr_{jz})} \tag{3.104}$$

Finalmente, y por equilibrio, obtenemos los esfuerzos cortantes:

$$R'_{iy} = \frac{-q_y b}{L},\ \ R'_{jy} = \frac{-q_y a}{L}$$

$$F'_{iy} = R'_{iy} + \frac{M'_{iz}+M'_{jz}}{2},\ \ F'_{jy} = R'_{jy} - \frac{M'_{iz}+M'_{jz}}{2} \tag{3.105}$$

$$R'_{iz} = \frac{-q_z b}{L},\ \ R'_{jz} = \frac{-q_z a}{L}$$

$$F'_{iz} = R'_{iz} - \frac{M'_{iy}+M'_{jy}}{2},\ \ F'_{jz} = R'_{jz} + \frac{M'_{iy}+M'_{jy}}{2} \tag{3.106}$$

3.3.2.2. Cargas de fuerzas constantes en barras

Sea una barra biapoyada a la cual le aplicamos cargas de fuerza constantes con valores q_x, q_y y q_z sobre cada uno de sus ejes locales X, Y y Z, respectivamente, como ocurre en la figura 3.14. Debido a la acción de la carga q_x, obtendremos los siguientes esfuerzos axiles:

$$F'_{ix} = \frac{-q_x bc}{L}, \;\; F'_{jx} = \frac{-q_x ac}{L} \tag{3.107}$$

Si calculamos los ángulos de los giros en los extremos de la barra, debido a la actuación de las cargas q_y y q_z, multiplicados por EI_z o EI_y, tendremos que:

$$\varphi'_{iz} = \frac{q_y abc(L + b - \frac{c^2}{4a})}{6L}, \; \varphi'_{jz} = \frac{-q_y abc(L + a - \frac{c^2}{4b})}{6L} \tag{3.108}$$

$$\varphi_{iy} = \frac{-q_z abc(L + b - \frac{c^2}{4a})}{6L}, \; \varphi_{jy} = \frac{q_z abc(L + a - \frac{c^2}{4b})}{6L} \tag{3.109}$$

Considerando los grados de rigidez a giro en Y y en Z entre los extremos de las barras y sus nudos y los ángulos de giro calculados, obtendremos los siguientes momentos flectores:

$$M'_{iy} = -\frac{6\varphi_{jy} Gr_{iy} Gr_{jy} + 12\varphi_{iy} Gr_{iy}}{L(4 - Gr_{iy} Gr_{jy})}$$

$$M'_{jy} = \frac{6\varphi_{iy}}{L} - \frac{12\varphi_{jy} Gr_{jy} + 24\varphi_{iy}}{L(4 - Gr_{iy} Gr_{jy})} \tag{3.110}$$

$$M'_{iz} = -\frac{6\varphi'_{jz} Gr_{iz} Gr^z_j + 12\varphi'_{iz} Gr_{iz}}{L(4 - Gr_{iz} Gr_{jz})}$$

$$M'_{jz} = \frac{6\varphi'_{iz}}{L} - \frac{12\varphi'_{jz} Gr_{jz} + 24\varphi'_{iz}}{L(4 - Gr_{iz} Gr_{jz})} \tag{3.111}$$

Por último, a partir de dichos momentos flectores obtenemos el valor de los esfuerzos cortantes:

$$R'_{iy} = \frac{-q_y bc}{L}, \;\; R'_{jy} = \frac{-q_y ac}{L}$$

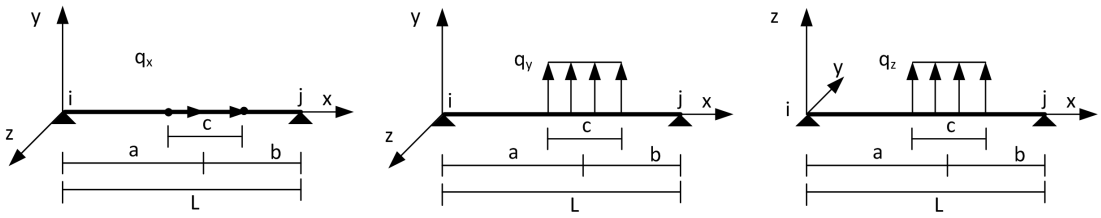


Figura 3.14: Cargas de fuerza constante actuando sobre los ejes locales de una barra.

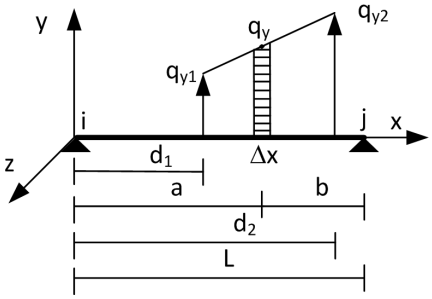


Figura 3.15: Cargas de fuerza variable actuando sobre el eje Y local de una barra.

$$F'_{iy} = R'_{iy} + \frac{M'_{iz} + M'_{jz}}{2},\ F'_{jy} = R'_{jy} - \frac{M'_{iz} + M'_{jz}}{2} \tag{3.112}$$

$$R'_{iz} = \frac{-q_z bc}{L},\ R'_{jz} = \frac{-q_z ac}{L}$$

$$F'_{iz} = R'_{iz} - \frac{M'_{iy} + M'_{jy}}{2},\ F'_{jz} = R_{jz} + \frac{M'_{iy} + M'_{jy}}{2} \tag{3.113}$$

3.3.2.3. Cargas de fuerzas variables en barras

Consideramos ahora en caso en el cual las cargas de fuerza tienen un valor inicial y final diferente (ver figura 3.15), siguiendo una variación lineal entre dichos valores. Aplicamos por tanto a los ejes locales X, Y y Z de una barra biapoyada las 3 siguientes cargas $[q_{x1},\ q_{x2}]$, $[q_{y1},\ q_{y2}]$, $[q_{z1},\ q_{z2}]$, definidas por sus valores en su punto inicial y final de aplicación.

Para obtener los esfuerzos de empotramiento y los ángulos de giro generados en los extremos de la barra, dividiremos el intervalo de aplicación de la carga en un conjunto de n subintervalos, de un tamaño $\Delta x = 1$ centímetro, por ejemplo, en los cuales aplicaremos una carga puntual de valor q, sumando el efecto de

todas ellas.

Comenzamos obteniendo el número de subintervalos y el valor de la pendiente de la carga lineal aplicada sobre cada eje. Para simplificar, entenderemos que las tres cargas tienen un mismo punto inicial d_1 y final d_2 de aplicación en la barra:

$$n = \frac{d_2 - d_1}{\Delta x}, \quad \Delta q_x = \frac{q_{x2} - q_{x1}}{d_2 - d_1}, \quad \Delta q_y = \frac{q_{y2} - q_{y1}}{d_2 - d_1}, \quad \Delta q_z = \frac{q_{z2} - q_{z1}}{d_2 - d_1} \tag{3.114}$$

A continuación, y para subintervalo k, calculamos la distancia a desde el extremo inicial de la barra al punto de aplicación de la carga, la distancia b entre dicho punto de aplicación y el extremo final de la barra, así como los valores de las cargas puntuales q_x, q_y y q_z aplicadas sobre el eje local correspondiente:

$$\begin{cases} a(k) = d_1 + (k-1)\Delta x + \frac{\Delta x}{2}, \quad b(k) = L - a(k), \quad \forall k = 1, ..., n \\ q_x(k) = [q_{x1} + \Delta q_x(a(k) - d_1)]\,\Delta x \\ q_y(k) = [q_{y1} + \Delta q_y(a(k) - d_1)]\,\Delta x \\ q_z(k) = [q_{z1} + \Delta q_z(a(k) - d_1)]\,\Delta x \end{cases} \tag{3.115}$$

A partir de dichos valores, los esfuerzos de empotramiento perfecto y los ángulos de giro serán los siguientes, como resultado de la aportación de la carga puntual aplicada en cada subintervalo.

Si comenzamos por la carga variable aplicada en el eje X, tendremos que el esfuerzo axil al que da lugar vale:

$$F'_{ix} = -\sum_{k=1}^{n} \frac{q_x(k)b(k)}{L}, \quad F'_{jx} = -\sum_{k=1}^{n} \frac{q_x(k)a(k)}{L} \tag{3.116}$$

Debido a la libertad a giro de los nudos, es posible calcular los ángulos del giro en los extremos de las barras, multiplicados por EI_z o EI_y, teniendo en cuenta el grado de desconexión entre los nudos y los extremos de las mismas, debido a la actuación de las cargas en Y y en Z:

$$\varphi'_{iz} = \sum_{k=1}^{n} \frac{q_y(k)a(k)b(k)(L + b(k))}{6L}$$

$$\varphi'_{jz} = -\sum_{k=1}^{n} \frac{q_y(k)a(k)b(k)(L+a(k))}{6L} \tag{3.117}$$

$$\varphi'_{iy} = -\sum_{k=1}^{n} \frac{q_z(k)a(k)b(k)(L+b(k))}{6L}$$

$$\varphi'_{jy} = \sum_{k=1}^{n} \frac{q_z(k)a(k)b(k)(L+a(k))}{6*L} \tag{3.118}$$

Además, es posible expresar los momentos flectores en función de los ángulos de giro de los extremos y del grado de rigidez a giro, en los ejes X e Y, entre el extremo de la barra y el nudo al que confluye:

$$M'_{iy} = -\frac{6\varphi_{jy}Gr_{iy}Gr_{jy} + 12\varphi_{iy}Gr_{iy}}{L(4 - Gr_{iy}Gr_{jy})}$$

$$M'_{jy} = \frac{6\varphi_{iy}}{L} - \frac{12\varphi_{jy}Gr_{jy} + 24\varphi_{iy}}{L(4 - Gr_{iy}Gr_{jy})} \tag{3.119}$$

$$M'_{iz} = -\frac{6\varphi'_{jz}Gr_{iz}Gr^z_j + 12\varphi'_{iz}Gr_{iz}}{L(4 - Gr_{iz}Gr_{jz})}$$

$$M'_{jz} = \frac{6\varphi'_{iz}}{L} - \frac{12\varphi'_{jz}Gr_{jz} + 24\varphi'_{iz}}{L(4 - Gr_{iz}Gr_{jz})} \tag{3.120}$$

Para finalizar el cálculo de los esfuerzos de empotramiento resultantes, obtenemos los esfuerzos a cortante:

$$R'_{iy} = -\sum_{k=1}^{n} \frac{q_y(k)b(k)}{L}, \;\; R'_{jy} = -\sum_{k=1}^{n} \frac{q_y(k)a(k)}{L}$$

$$F'_{iy} = R'_{iy} + \frac{M'_{iz} + M'_{jz}}{2}, \;\; F'_{jy} = R'_{jy} - \frac{M'_{iz} + M'_{jz}}{2} \tag{3.121}$$

$$R'_{iz} = -\sum_{k=1}^{n} \frac{q_z(k)b(k)}{L}, \;\; R'_{jz} = -\sum_{k=1}^{n} \frac{q_z(k)a(k)}{L}$$

$$F'_{iz} = R'_{iz} - \frac{M'_{iy} + M'_{jy}}{2}, \;\; F'_{jz} = R'_{jz} + \frac{M'_{iy} + M'_{jy}}{2} \tag{3.122}$$

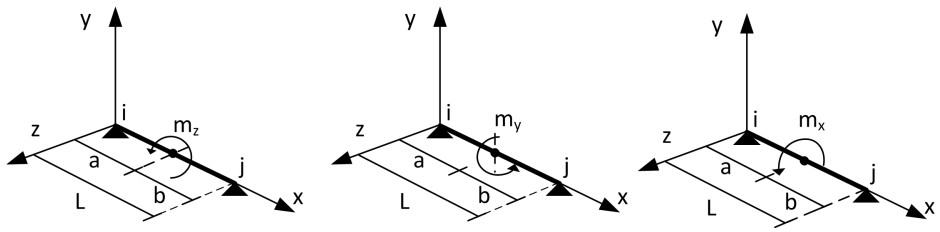


Figura 3.16: Cargas de momento puntual actuando sobre los ejes locales de una barra.

3.3.2.4. Cargas de momentos puntuales en barras

La aplicación de cargas de momento puntual m_x, m_y y m_z sobre los ejes X, Y y Z de una barra biapoyada, como las aplicadas en la figura 3.16, genera como resultado esfuerzos de empotramiento correspondientes a momentos torsores, esfuerzos cortantes y momentos flectores, además de los diferentes giros en los extremos, de los que calculamos sus ángulos. Para comenzar, los momentos torsores que aparecen en los extremos debido a la actuación del momento puntual en X son:

$$M'_{ix} = \frac{-m_x b}{L}, \;\; M'_{jx} = \frac{-m_x a}{L} \tag{3.123}$$

El hecho de que los nudos estén libres a giros, nos habilita a calcular el ángulo del giro en Y y en Z de los extremos de las barras, multiplicados por EI_y o EI_z, debido a la actuación de las cargas m_y y m_z:

$$\varphi'_{iy} = \frac{m_y L}{6}\left(\frac{3b^2}{L^2} - 1\right), \; \varphi'_{jy} = \frac{m_y L}{6}\left(\frac{3a^2}{L^2} - 1\right) \tag{3.124}$$

$$\varphi'_{iz} = \frac{m_z L}{6}\left(\frac{3b^2}{L^2} - 1\right), \; \varphi'_{jz} = \frac{m_z L}{6}\left(\frac{3a^2}{L^2} - 1\right) \tag{3.125}$$

Considerando dichos ángulos de giro y el grado de rigidez rotacional en Y y en Z entre los extremos de las barras y los nudos, obtenemos el valor de los momentos flectores:

$$M'_{iy} = -\frac{6\varphi_{jy} Gr_{iy} Gr_{jy} + 12\varphi_{iy} Gr_{iy}}{L(4 - Gr_{iy} Gr_{jy})}$$

$$M'_{jy} = \frac{6\varphi_{iy}}{L} - \frac{12\varphi_{jy} Gr_{jy} + 24\varphi_{iy}}{L(4 - Gr_{iy} Gr_{jy})} \tag{3.126}$$

$$M'_{iz} = -\frac{6\varphi'_{jz}Gr_{iz}Gr_j^z + 12\varphi'_{iz}Gr_{iz}}{L(4 - Gr_{iz}Gr_{jz})}$$
$$M'_{jz} = \frac{6\varphi'_{iz}}{L} - \frac{12\varphi'_{jz}Gr_{jz} + 24\varphi'_{iz}}{L(4 - Gr_{iz}Gr_{jz})} \tag{3.127}$$

Finalmente, y ya para concluir, calculamos los esfuerzos cortantes que dan lugar al equilibrio de la barra:

$$R'_{iz} = \frac{-m_y}{L},\ \ R'_{jz} = \frac{m_y}{L}$$
$$F'_{iz} = R'_{iz} - \frac{M'_{iy} + M'_{jy}}{2},\ \ F'_{jz} = R'_{jz} + \frac{M'_{iy} + M'_{jy}}{2} \tag{3.128}$$
$$R'_{iy} = \frac{m_z}{L},\ \ R'_{jy} = \frac{-m_z}{L}$$
$$F'_{iy} = R'_{iy} + \frac{M'_{iz} + M'_{jz}}{2},\ \ F'_{jy} = R'_{jy} - \frac{M'_{iz} + M'_{jz}}{2} \tag{3.129}$$

3.3.2.5. Cargas de momentos constantes en barras

A pesar de que la carga de momento constante puede aplicarse sobre cualquier eje, consideraremos únicamente cargas de momentos torsores constantes, como los que podemos observar en la figura 3.17, aplicados por tanto sobre el eje X de la barra. Manteniendo las consideraciones de barra biapoyada y sin desconexiones a giro en el eje X, la aplicación de dicha carga genera, a efectos de momentos de empotramiento, los siguientes momentos torsores:

$$M'_{ix} = \frac{-m_x bc}{L},\ \ M'_{jx} = \frac{-m_x ac}{L} \tag{3.130}$$

3.3.2.6. Cargas de momentos variables en barras

Al igual que comentábamos en el punto anterior, en este documento solo vamos a tener en cuenta la aplicación de cargas de momento torsor variable, con diferente valor inicial m_{x1} y final m_{x2}, sobre el eje X de la barra (biapoyada y

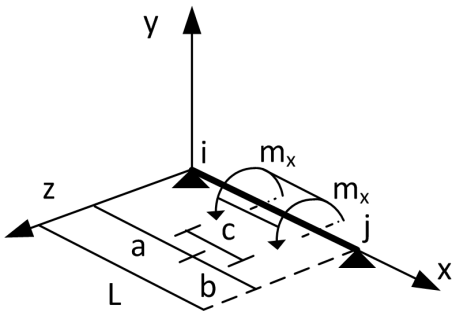


Figura 3.17: Carga de momento torsor constante.

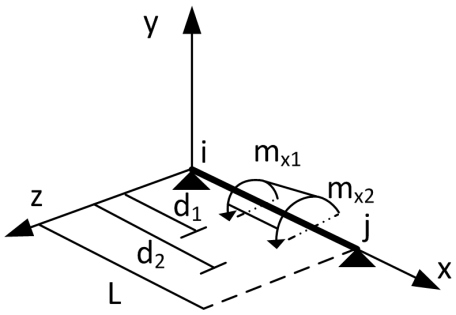


Figura 3.18: Carga de momento torsor variable.

sin relajaciones a giro en X), siendo lineal y constante dicha variación de valores (ver figura 3.18). La forma de proceder es similar a la ya llevada a cabo en el caso de las cargas de fuerzas variables, dividiendo el intervalo de aplicación de la carga en un conjunto de subintervalos de longitud Δx=1 cm, en los cuales aplicamos un momento torsor puntual. Para comenzar, calculamos el número de subintervalos n y el valor de la pendiente Δm_x del momento lineal aplicado:

$$n = \frac{d_2 - d_1}{\Delta x}, \quad \Delta m_x = \frac{m_{x2} - m_{x1}}{d_2 - d_1} \tag{3.131}$$

Para subintervalo k, obtenemos la distancia a desde el extremo inicial de la barra al punto de aplicación de la carga, la distancia b entre dicho punto de aplicación y el extremo final de la barra y el valor del momento torsor puntual m_x aplicado:

$$\begin{cases} a(k) = d_1 + (k-1)\Delta x + \frac{\Delta x}{2}, \quad b(k) = L - a(k), \quad \forall k = 1, ..., n \\ m_x(k) = [m_{x1} + \Delta m_x(a(k) - d_1)] * \Delta x \end{cases} \tag{3.132}$$

Finalmente, obtenemos el esfuerzo de empotramiento como suma del efecto producido tras la aplicación de cada carga de momento:

$$M'_{ix} = -\sum_{k=1}^{n} \frac{m_x(k)b(k)}{L}, \quad M'_{jx} = -\sum_{k=1}^{n} \frac{m_x(k)a(k)}{L} \tag{3.133}$$

3.3.2.7. Cargas de temperatura en barras

Como es lógico, las estructuras se ven sometidas a cambios de temperatura, con respecto a una situación inicial de equilibrio. La forma de calcular los efectos de dicha variación térmica consistirá en sumar el resultado de considerar un incremento o decremento de temperatura sobre cada una de las barras.

Si consideráramos cada barra a título individual, observaríamos que dicho incremento de temperatura ΔT equivale la aplicación de una carga puntual $-q_x$ en su extremo inicial más otra carga puntual q_x en su extremo final, con el siguiente valor:

$$q_x = \Delta T \alpha E A \tag{3.134}$$

En dicha expresión, α representa el coeficiente de dilatación del material de la barra, E es su módulo de deformación longitudinal y A se corresponde con el área de su sección. Aplicando ambas cargas puntuales de acuerdo a lo detallado en la sección 3.3.2.1, obtendríamos los valores de los esfuerzos de empotramiento generados.

3.3.2.8. Esfuerzos de empotramiento en barras con desconexiones en extremos

En las secciones anteriores, hemos obtenido los esfuerzos de empotramiento en los extremos de las barras considerando, únicamente, una posible desconexión a giro en Y o en Z entre los extremos de la barra y el nudo al que confluyen. En consecuencia, proporcionamos ahora los valores de los esfuerzos de empotramiento cuando se produzcan desconexiones longitudinales en cualquiera de los 3 ejes o desconexiones a giro en el eje X.

Desconexión longitudinal en el eje X

Sean F'_{ix} y F'_{iy} los esfuerzos axiles de empotramiento en el extremo inicial y final de la barra, calculados de acuerdo a las expresiones proporcionadas en las secciones anteriores. Si la barra presenta una desconexión longitudinal en el eje X, el valor efectivo de los esfuerzos axiles de empotramiento los calculamos del siguiente modo:

$$\begin{aligned} F^{*\prime}_{ix} &= F'_{ix} - F'_{ix}\,(1 - Gt_{ix}) + F'_{jx}\,(1 - Gt_{jx}) \\ F^{*\prime}_{jx} &= F'_{jx} - F'_{jx}\,(1 - Gt_{jx}) + F'_{ix}\,(1 - Gt_{ix}) \end{aligned} \tag{3.135}$$

Desconexión longitudinal en el eje Y

En los apartados anteriores, hemos calculado los cortantes y los momentos flectores de empotramiento perfecto considerando que la barra no presenta desconexiones longitudinales en el eje Y entre el nudo y sus extremos. Si dichas desconexiones sí que estuvieran presentes, y siempre a partir de los valores anteriores obtenidos, los esfuerzos efectivos a cortante pasarían a ser estos:

$$\begin{aligned} F^{*\prime}_{iy} &= F'_{iy} - F'_{iy}\,(1 - Gt_{iy}) + F'_{jy}\,(1 - Gt_{jy}) \\ F^{*\prime}_{jy} &= F'_{jy} - F'_{jy}\,(1 - Gt_{jy}) + F'_{iy}\,(1 - Gt_{iy}) \end{aligned} \tag{3.136}$$

Esta nueva configuración de la conexión de la barra con sus nudos inicial y final obliga a calcular de nuevo los momentos flectores, siempre a partir de los valores en los cuales no se han considerado las desconexiones longitudinales, y de acuerdo a las dos siguientes escenarios posibles:

1. Desconexión en el extremo inicial de la barra ($Gt_{iy} = 0$ y $Gt_{jy} = 1$):

$$\begin{aligned} M^{*\prime}_{iz} &= M'_{iz} - F'_{iy}L\left(\frac{|M'_{iz}|}{|M'_{iz}| + |M'_{jz}|}\right) \\ M^{*\prime}_{jz} &= M'_{jz} - F'_{iy}L\left(1 - \frac{|M'_{iz}|}{|M'_{iz}| + |M'_{jz}|}\right) \end{aligned} \tag{3.137}$$

2. Desconexión en el extremo final de la barra ($Gt_{iy} = 1$ y $Gt_{jy} = 0$):

$$M^{*\prime}_{iz} = M'_{iz} + F'_{jy}L\left(\frac{|M'_{iz}|}{|M'_{iz}| + |M'_{jz}|}\right)$$

$$M^{*\prime}_{jz} = M'_{jz} + F'_{jy}L\left(1 - \frac{|M'_{iz}|}{|M'_{iz}| + |M'_{jz}|}\right) \tag{3.138}$$

Desconexión longitudinal en el eje Z

De modo similar al apartado anterior, tenemos ahora en cuenta la posible desconexión longitudinal en el eje Z de la barra. Eso supone que los esfuerzos efectivos a cortante serían:

$$F^{*\prime}_{iz} = F'_{iz} - F'_{iz}\,(1 - Gt_{iz}) + F'_{jz}\,(1 - Gt_{jz})$$

$$F^{*\prime}_{jz} = F'_{jz} - F'_{jz}\,(1 - Gt_{jz}) + F'_{iz}\,(1 - Gt_{iz}) \tag{3.139}$$

Del mismo modo, proporcionamos los nuevos momentos flectores considerando los dos escenarios posibles, partiendo de valores en los cuales no hemos contemplado la desconexión longitudinal:

1. Desconexión en el extremo inicial de la barra ($Gr_{iz} = 0$ y $Gr_{jz} = 1$):

$$M^{*\prime}_{iy} = M'_{iy} + F'_{iz}L\left(\frac{|M'_{iy}|}{|M'_{iy}| + |M'_{jy}|}\right)$$

$$M^{*\prime}_{jy} = M'_{jy} + F'_{iz}L\left(1 - \frac{|M'_{iy}|}{|M'_{iy}| + |M'_{jy}|}\right) \tag{3.140}$$

2. Desconexión en el extremo final de la barra ($Gr_{iz} = 1$ y $Gr_{jz} = 0$):

$$M^{*\prime}_{iy} = M'_{iy} - F'_{jz}L\left(\frac{|M'_{iy}|}{|M'_{iy}| + |M'_{jy}|}\right)$$

$$M^{*\prime}_{jy} = M'_{jy} - F'_{jz}L\left(1 - \frac{|M'_{iy}|}{|M'_{iy}| + |M'_{jy}|}\right) \tag{3.141}$$

Desconexión a giro en el eje X

La desconexión a giro en el eje X de la barra tiene repercusión sobre el momento torsor de empotramiento proporcionado en las secciones previas, el cual era debido a la actuación de una carga de momento torsor puntual o repartida. Partiendo por tanto de los valores anteriores, los momentos torsores efectivos de empotramiento son los siguientes:

$$\begin{aligned} M^{*\prime}_{ix} &= M'_{ix} - M'_{ix}\,(1 - Gr_{ix}) + M'_{jx}\,(1 - Gr_{jx}) \\ M^{*\prime}_{jx} &= M'_{jx} - M'_{jx}\,(1 - Gr_{jx}) + M'_{ix}\,(1 - Gr_{ix}) \end{aligned} \tag{3.142}$$

3.3.2.9. Esfuerzos de empotramiento en barras con excentricidades

Supongamos una barra excéntrica en alguno de sus extremos, como es el caso de la barra b mostrada en la figura 3.3, la cual presenta una excentricidad en su extremo i. Para obtener los esfuerzos de empotramiento en ejes globales en el nudo A al que confluye la barra en dicho extremo, emplearemos la matriz H_{iA} recogida en la expresión (3.17):

$$p_A = H_{iA} R^b p'^b_i \tag{3.143}$$

siendo p'^b_i los esfuerzos de empotramiento, en ejes locales, en el extremo i. Teniendo en cuenta la que sería la matriz H_{jB} para el extremo j de la barra b, se obtiene que:

$$\begin{bmatrix} p_A \\ p_B \end{bmatrix} = \begin{bmatrix} H_{iA} & 0 \\ 0 & H_{jB} \end{bmatrix} \begin{bmatrix} R^b & 0 \\ 0 & R^b \end{bmatrix} \begin{bmatrix} p'^b_i \\ p'^b_j \end{bmatrix} = \begin{bmatrix} H_{iA} & 0 \\ 0 & H_{jB} \end{bmatrix} \begin{bmatrix} p^b_i \\ p^b_j \end{bmatrix} = \begin{bmatrix} H_{iA} p^b_i \\ H_{jB} p^b_j \end{bmatrix} \tag{3.144}$$

3.3.3. Cargas aplicadas sobre los triángulos

Además de las cargas que pueden estar actuando directamente sobre los nudos de los triángulos, hay que contemplar además todas aquellas aplicadas sobre el elemento como tal. Dentro de este segundo tipo, destacan las cargas superficiales (entre las que se incluyen, por ejemplo, las cargas de peso propio del elemento) o

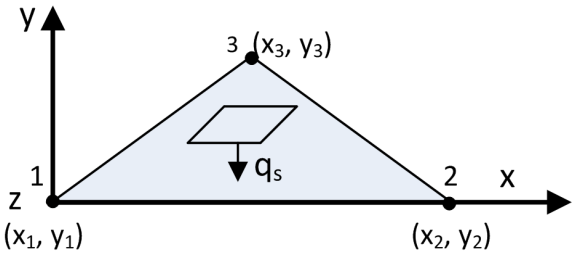


Figura 3.19: Carga superficial aplicada sobre un triángulo.

las cargas debidas a la variación de la temperatura.

Como ya hemos comentado con anterioridad, el valor de todas aquellas acciones que se apliquen directamente sobre los nudos de los triángulos se sumarán en las componentes correspondientes a dichos nudos en el vector de fuerzas nodales F_N, al estar expresadas en ejes globales de la estructura.

3.3.3.1. Cargas superficiales sobre los triángulos

Supongamos una carga superficial $q_s = [q_x, q_y, q_z]^T$, expresada como fuerza por unidad de superficie y en los ejes globales de la estructura, que está actuando sobre el triángulo e, como la mostrada en la figura 3.19. A partir de dicha carga, debemos generar el vector, de tamaño 9x1, con las fuerzas equivalentes sobre los nodos que lo componen:

$$f^e = \int_{S^e} \mathrm{N}^{\mathrm{T}} q_s \, dx \, dy \tag{3.145}$$

siendo N la matriz que recoge las funciones de forma del elemento:

$$\mathrm{N} = \begin{bmatrix} \mathrm{N}_1 & \mathrm{N}_2 & \mathrm{N}_3 \end{bmatrix}, \ \mathrm{N_i} = \begin{bmatrix} N_i(\xi,\eta,\zeta) & 0 & 0 \\ 0 & N_i(\xi,\eta,\zeta) & 0 \\ 0 & 0 & N_i(\xi,\eta,\zeta) \end{bmatrix} \tag{3.146}$$

Resolviendo la integral anterior, se concluye que la carga aplicada sobre cualquiera de los tres nudos se calcula como:

$$f_i^e = \begin{bmatrix} f_{ix} \\ f_{iy} \\ f_{iz} \end{bmatrix} = \frac{A}{3} \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} i = 1, 2, 3 \tag{3.147}$$

siendo A el área del triángulo, lo cual significa que la carga superficial aplicada ha quedado distribuida por igual entre los 3 nudos del triángulo. Finalmente, habrá que ensamblar dicha carga distribuida en el vector global de fuerzas, en aquellas posiciones correspondientes a la numeración global de los nodos dentro de la estructura.

3.3.3.2. Cargas por variación de la temperatura en triángulos

Las variaciones térmicas generan un cambio en la superficie del elemento, dependiendo de las propiedades del material que lo compone. Así, en un material isótropo, el aumento de temperatura ΔT vendrá dado por la siguiente deformación unitaria uniforme inicial, dependiendo del coeficiente α de dilatación del material:

$$\varepsilon_0 = \begin{bmatrix} \alpha\Delta T & \alpha\Delta T & 0 \end{bmatrix}^T \tag{3.148}$$

Recurriendo a la expresión (2.21), el vector de fuerzas, de tamaño 6x1, aplicado sobre los nudos del elemento se calcula como:

$$f'^e = \int_{V^e} B^T D\,\varepsilon_0\,d\nu = \int_{S^e} B^T D\,\varepsilon_0 h\,ds \tag{3.149}$$

siendo B la matriz de deformación del elemento, según la expresión (3.29), D la matriz constitutiva del material y h el espesor del elemento.

Teniendo en cuenta que dicha variación térmica ocasionará deformaciones longitudinales en los ejes X e Y del elemento, tendremos en consideración su comportamiento de membrana, motivo por el cual dicha matriz D vendrá dada por la matriz E_m, recogida en la expresión (3.39). De este modo, y gracias a la cuadratura de Gauss-Legendre para resolver numéricamente las integrales resultantes, podemos decir que las componentes longitudinales X e Y del vector de fuerzas aplicadas sobre cada nodo del triángulo se calcularían como:

$$f_i'^e = \begin{bmatrix} f'_{ix} \\ f'_{iy} \end{bmatrix} = \frac{1}{2}\,h\,|J|\,B_i^T E_m\,\varepsilon_0,\ i = 1,2,3 \tag{3.150}$$

donde J representa la matriz Jacobiana del elemento, siendo iguales a 0 el resto de componentes del vector de fuerzas. Como ya es bien sabido, habría que

transformar a ejes globales las fuerzas obtenidas, de acuerdo a lo descrito en la expresión (3.56), además de ensamblarlas, en sus posiciones apropiadas, en el vector de fuerzas globales de la estructura.

3.3.4. Cargas aplicadas sobre los cuadriláteros

Como hemos dicho con anterioridad, únicamente consideraremos a los cuadriláteros como elementos de mallado que darán lugar, a efectos de cálculo, a 4 triángulos a procesar y a promediar. Por tanto, multiplicaremos por 0.5 el valor de toda carga (superficial o de incremento temperatura) aplicada sobre un triángulo que provenga de un cuadrilátero.

3.3.5. Cargas aplicadas sobre los tetraedros

Las acciones que tendremos en consideración aplicadas sobre este tipo de elementos son las que actúan directamente sobre sus nodos, las cargas de peso propio de los elementos y las debidas a las variaciones de la temperatura. En el caso de las cargas aplicadas sobre los nodos, las sumaremos en las posiciones correspondientes a su numeración en el vector de fuerzas. A continuación abordamos el resto de cargas contempladas.

3.3.5.1. Cargas volumétricas sobre los tetraedros

El vector que recoge las fuerzas equivalentes distribuidas en los nodos de un tetraedro e a partir de una carga $q_v = [q_x, q_y, q_z]^T$ que esté actuando sobre el mismo, expresada como fuerza por unidad de volumen, se obtiene a partir de la siguiente expresión:

$$f^e = \int_{V^e} \mathrm{N}^{\mathrm{T}} q_v \, d\nu \tag{3.151}$$

donde N es la matriz con las funciones de forma recogida en (3.63). Entre las acciones expresadas por unidad de volumen que puedan actuar sobre un elemento tetraédrico encontramos a las cargas del peso propio del elemento, definidas por medio del vector de carga $q_v = [q_x, q_y, q_z]^T = [0, 0, -\gamma]^T$, donde γ representa el

peso específico del material del que se compone el elemento.

Resolviendo la citada integral tras emplear 4 puntos de integración, ocurre que la carga queda repartida por igual en cada nodo i del tetraedro:

$$f_i^e = \begin{bmatrix} f_{ix} \\ f_{iy} \\ f_{iz} \end{bmatrix} = \frac{V}{4} \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix}, i = 1, 2, 3, 4 \tag{3.152}$$

siendo V, como ya hemos comentado en numerosas ocasiones, el volumen del elemento. Dichos valores de carga ya están expresados en ejes globales, debiendo ensamblarlos, en las posiciones correspondientes a cada nudo, en el vector de fuerzas de la estructura.

3.3.5.2. Cargas por variación de la temperatura en los tetraedros

La variación de la temperatura ΔT, en un tetraedro, viene recogida en el vector de deformación unitaria inicial. Si el elemento está compuesto por un material isótropo, el vector tendrá el valor siguiente, el cual depende del coeficiente α de dilatación del material:

$$\varepsilon_0 = \begin{bmatrix} \alpha\Delta T & \alpha\Delta T & \alpha\Delta T & 0 & 0 & 0 \end{bmatrix}^T \tag{3.153}$$

De acuerdo a lo recogido en la expresión (2.21), el vector con las fuerzas que actúan sobre los cuatro nodos de un elemento debido a la deformación unitaria inicial se calcula como:

$$f^e = \int_V B^T D\, \varepsilon_0\, d\nu \tag{3.154}$$

donde B es la matriz de deformación del elemento y D la matriz constitutiva del material. Empleando 4 puntos de integración para resolver la integral, el vector de fuerzas equivalentes en los nudos de un tetraedro de volumen V valdrá:

$$f^e = V B^T D \varepsilon_0 \tag{3.155}$$

Dicho vector ya estará expresado en ejes globales. No obstante, si queremos calcular el valor de las fuerzas aplicadas sobre el nudo i del tetraedro, a título

individual, la expresión a emplear será esta otra, debiendo ensamblar el vector obtenido, con posterioridad:

$$f_i^e = \begin{bmatrix} f_{ix} \\ f_{iy} \\ f_{iz} \end{bmatrix} = VB_i^T D\varepsilon_0,\ i = 1, 2, 3, 4 \tag{3.156}$$

3.3.6. Cargas aplicadas sobre los prismas triangulares

Teniendo en cuenta que los prismas triangulares sólo se consideran a efectos de mallado y que a nivel de cálculo se dividen en 3 tetraedros, no solapados, las cargas aplicadas sobre cada uno de ellos se tratarán de manera idéntica a lo que hemos visto en el punto anterior.

3.3.7. Cargas aplicadas sobre los hexaedros

Como ocurría con los tetraedros, consideraremos las acciones aplicadas sobre los nudos del hexaedro, así como las cargas debidas al peso propio del mismo y a los cambios de temperatura. Como es bien sabido, las cargas aplicadas sobre los nudos se sumarán en el vector de fuerzas, de acuerdo al valor de sus índices.

3.3.7.1. Cargas volumétricas sobre los hexaedros

Sea $q_v = [q_x, q_y, q_z]^T$ el vector de cargas, expresado como fuerza por unidad de volumen, aplicadas sobre un elemento hexaédrico e. El vector con las fuerzas nodales equivalentes se obtiene del siguiente modo:

$$f^e = \int_V \mathrm{N}^T q_v \, d\nu =$$

$$= \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \mathrm{N}\left(\xi, \eta, \zeta\right)^T q_v |J\left(\xi, \eta, \zeta\right)| \, d\xi \, d\eta \, d\zeta \tag{3.157}$$

donde N, de tamaño 3x24, es la matriz con las funciones de forma de la expresión (3.80) y J es la matriz Jacobiana (3.83), cuyo determinante hay que calcular.

Un ejemplo de fuerza de volumen es precisamente la carga de peso propio del elemento, expresada como $q_v = [q_x, q_y, q_z]^T = [0, 0, -\gamma]^T$, donde γ se corresponde con el peso específico del material que forma el elemento.

Resolviendo numéricamente dicha integral de manera similar a como ya vimos al obtener la matriz de rigidez del elemento, empleando por tanto los puntos de integración mostrados en la tabla 3.2, concluimos que el vector de fuerzas equivalentes sobre cada nodo i se obtiene como:

$$f_i^e = \sum_{p=1}^{8} \mathrm{N}_i q_v |J(\xi_p, \eta_p, \zeta_p)| \tag{3.158}$$

donde:

$$\mathrm{N_i} = \begin{bmatrix} N_i(\xi_p, \eta_p, \zeta_p) & 0 & 0 \\ 0 & N_i(\xi_p, \eta_p, \zeta_p) & 0 \\ 0 & 0 & N_i(\xi_p, \eta_p, \zeta_p) \end{bmatrix} \tag{3.159}$$

Estos valores de carga ya están expresados en ejes globales y debemos sumarlos a las posiciones correspondientes de cada nudo en el vector de fuerzas aplicadas sobre la estructura.

3.3.7.2. Cargas por variación de la temperatura en los hexaedros

Si pretendemos obtener los efectos de la variación de la temperatura ΔT en un hexaedro, la deformación unitaria inicial correspondiente a dicha variación vendrá dada por la siguiente expresión. Si el elemento está compuesto por un material isótropo, con un coeficiente lineal de dilatación térmica α tendremos que:

$$\varepsilon_0 = \begin{bmatrix} \alpha\Delta T & \alpha\Delta T & \alpha\Delta T & 0 & 0 & 0 \end{bmatrix}^T \tag{3.160}$$

Para obtener el vector de fuerzas equivalentes f^e en los ocho nudos del hexaedro, teniendo en cuenta la expresión (2.21), tendremos que resolver la siguiente integral, donde B es la matriz de deformación del elemento y D la matriz constitutiva del material:

$$f^e = \int_V B^T D\, \varepsilon_0\, d\nu =$$

$$= \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} B\left(\xi,\eta,\zeta\right)^{T} D\varepsilon_0 |J\left(\xi,\eta,\zeta\right)|\, d\xi\; d\eta\; d\zeta \tag{3.161}$$

Resolviendo numéricamente dicha integral obtendremos el siguiente vector de fuerza:

$$f^e = \sum_{p=1}^{8} B\left(\xi_p,\eta_p,\zeta_p\right)^{T} D\varepsilon_0 |J\left(\xi_p,\eta_p,\zeta_p\right)| \tag{3.162}$$

el cual está ya expresado en ejes globales de la estructura. Con todo ello, el vector de fuerzas equivalentes aplicado sobre el nudo i del hexaedro valdrá:

$$f_i^e = \begin{bmatrix} f_{ix} \\ f_{iy} \\ f_{iz} \end{bmatrix} = \sum_{p=1}^{8} B_i\left(\xi_p,\eta_p,\zeta_p\right)^{T} D\varepsilon_0 |J\left(\xi_p,\eta_p,\zeta_p\right)|, i = 1,2,...,8 \tag{3.163}$$

3.4. Imposición de las condiciones de contorno

Todas las consideraciones realizadas hasta el momento relativas a la deducción de la ecuación de equilibrio de la estructura se han llevado a cabo al margen de las condiciones de contorno, motivo por el cual el sistema presentará infinitas posiciones de equilibrio, una para cada posición en el espacio del sólido rígido que representa. Matemáticamente esto supone que la matriz K de rigidez podrá ser singular, así que debemos realizar las operaciones oportunas para que se convierta en una matriz definida positiva.

En realidad, la ecuación de equilibrio estático de la estructura está compuesta por un conjunto de ecuaciones que contienen valores conocidos e incógnitas en ambos miembros de la igualdad. Así, la matriz de fuerzas contiene no solamente las cargas externas aplicadas sobre la estructura, sino también las reacciones en los apoyos (nudos que normalmente van unidos a la cimentación) que son desconocidas. Por su parte, la matriz de desplazamientos D incluye también los desplazamientos conocidos en los apoyos, junto con los desplazamientos incógnita del resto de nudos.

Dependiendo del tipo de apoyo, estos desplazamientos conocidos podrán ser cero, como es el caso de un apoyo fijo, iguales a un valor determinado, como son los asentamientos de los apoyos (movimientos impuestos), o iguales a funciones

como es el caso de los apoyos elásticos. Se entiende por supuesto que el resto de nudos de la estructura serán libres, con lo cual no tendrán ningún impedimento en los movimientos de cualquiera de sus grados de libertad. Un ejemplo habitual es el llamado apoyo de empotramiento perfecto, en el cual el nudo no puede moverse en ninguno de sus grados de libertad.

Realmente, imponer las condiciones de contorno en los nudos consiste en modificar las ecuaciones del sistema, al conocer de antemano sus desplazamientos y en consecuencia los elementos correspondientes del vector solución.

A modo de ejemplo, para expresar que el grado de libertad j de la estructura tiene impedidos sus movimientos ($d_j = 0$) bastará con llevar a cabo alguna de las estrategias siguientes:

1. Eliminar la ecuación j del sistema de ecuaciones, lo cual supone eliminar la fila y columna j de la matriz K de rigidez y la fila j de las matrices de fuerzas y desplazamientos.

2. Sustituir la ecuación j por la ecuación $d_j = 0$ o, lo que es lo mismo, por $K_{jj}d_j = 0$, ya que $K_{jj} \neq 0$. Esto implica:

   - Anular todos los elementos de la fila j de la matriz K, a excepción del elemento de la diagonal, y sustituir por *0* los elementos de las filas j en la matriz F de fuerzas.

   - Forzar a que $d_j = 0$ en el resto de ecuaciones, lo cual equivale a anular todos los elementos de la columna j en la matriz K, con excepción del elemento de la diagonal principal.

$$
j \cdots \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ 0 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} & & & & 0 & & \\ & & & & 0 & & \\ & & & & 0 & & \\ & & & & \vdots & & \\ 0 & 0 & 0 & \cdots & K_{jj} & \cdots & 0 \\ & & & & \vdots & & \\ & & & & 0 & & \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_j \\ \vdots \\ d_n \end{bmatrix} \tag{3.164}
$$

$$\begin{matrix} \vdots \\ j \end{matrix}$$

3. Sumar un valor elevado C en el elemento K_{jj}, por ejemplo $C = 10^{30}$, y resolver el sistema de ecuaciones correspondiente, lo cual equivale a decir que la rigidez en dicho punto es infinita. Esto supondrá que al despejar la componente d_j en la ecuación j se divida por un valor muy elevado, con lo cual, independientemente del valor del numerador, el resultado será 0:

$$
d_j = \frac{f_j - K_{j1}d_1 - \cdots - K_{jj-1}d_{j-1} - K_{jj+1}d_{j+1} - \cdots - K_{jn}d_n}{K_{jj} + C} \approx 0 \tag{3.165}
$$

Sin embargo, como hemos citado, son también otros los tipos de apoyos que pueden presentarse en la estructura, como son los apoyos elásticos o los asentamientos de los apoyos (movimientos impuestos). Esta tercera metodología abordada es válida en el caso en el que la estructura presente condiciones elásticas en sus apoyos, incorporando al elemento de la diagonal principal el valor C de rigidez que equivalga a la elasticidad del soporte y obteniendo como desplazamiento el valor correspondiente. Por otro lado, si al grado de libertad j de la estructura se le impone un movimiento o giro $A \neq 0$ determinado, entonces modificaremos la ecuación j correspondiente, de la forma:

$$
K_{j1}d_1 + \cdots + K_{jj-1}d_{j-1} + (K_{jj} + C)d_j + K_{jj+1}d_{j+1} + \cdots + K_{jn}d_n = f_j + A * C \tag{3.166}
$$

con lo cual, despejando la componente d_j nos queda:

$$d_j = \frac{f_j + A * C - K_{j1}d_1 - \cdots - K_{jj-1}d_{j-1} - K_{jj+1}d_{j+1} - \cdots - K_{jn}d_n}{K_{jj} + C} \approx$$
$$\approx A \tag{3.167}$$

3.5. Cálculo de los desplazamientos en los nudos de la estructura

El cálculo de los desplazamientos en los nudos de la estructura en ejes globales supone la resolución de la ecuación de equilibrio estático (3.1) o, lo que es lo mismo, la resolución de un sistema de ecuaciones lineales con múltiples partes derecha, teniendo una ecuación para cada grado de libertad de un nudo (6 en nuestro caso al trabajar en 3D):

$$KD = F.$$

Debe observarse que la matriz $F \in \mathbb{R}^{nxm}$ contiene m hipótesis básicas con las diferentes cargas externas conocidas que actúan, directa o indirectamente, sobre los nudos de la estructura, así como cualquier combinación de las mismas. Por su parte, la matriz $D \in \mathbb{R}^{nxm}$ contiene los desplazamientos incógnita de los nudos de la estructura, para las diferentes hipótesis de carga aplicadas.

Es precisamente la resolución de este sistema de ecuaciones lineales la que supone la parte más costosa en tiempo del análisis estático de estructuras, consumiendo la mayor parte del tiempo de CPU y el mayor espacio de memoria, motivo por el cual es muy importante disponer de métodos numéricos que resuelvan el sistema en el menor tiempo posible y que gestionen adecuadamente el consumo de memoria, lo que permitirá abordar problemas de mayor dimensión.

Esta matriz de rigidez $K \in \mathbb{R}^{nxn}$ es una matriz simétrica (puesto que se ha formado ensamblando matrices de barra y de elementos finitos que también eran simétricas), banda (si la numeración de los nudos no es aleatoria), dispersa (con gran parte de los elementos iguales a cero) y definida positiva (una vez que se

han impuesto las condiciones de contorno).

El ancho de banda de la matriz (ab) viene determinado por la numeración de los nudos y puede calcularse aplicando la siguiente expresión:

$$ab = (maxdif + 1) * 6 \tag{3.168}$$

siendo *maxdif* la máxima diferencia existente, en valor absoluto, entre la numeración de los nudos que forman parte de elemento, aplicando dicha diferencia a todas las barras y elementos finitos de la estructura.

El hecho de emplear una buena ordenación de los nudos, o una buena técnica de ordenación de la matriz K de coeficientes como paso previo a la resolución del sistema de ecuaciones, tiene una repercusión directa sobre la memoria demandada y sobre el tiempo de cómputo empleado, pudiendo reducir considerablemente el número de operaciones aritméticas efectuadas y en consecuencia los errores de redondeo en la solución del mismo.

3.6. Cálculo de solicitaciones en los extremos de las barras

Una vez calculados los movimientos en coordenadas globales de todos los nudos de la estructura, estamos en condiciones de calcular las solicitaciones o esfuerzos en ejes locales en los extremos de todas sus barras, para cada una de las combinaciones de cargas. Dichos esfuerzos no son más que las fuerzas generadas dentro de la estructura en respuesta a las cargas externas aplicadas, y se obtienen, para una barra b conectada a los nudos i y j, del siguiente modo:

$$\begin{bmatrix} f_i'^b \\ f_j'^b \end{bmatrix} = \begin{bmatrix} p_i'^b \\ p_j'^b \end{bmatrix} + \begin{bmatrix} K'_{ii} & K'_{ij} \\ K'_{ji} & K'_{jj} \end{bmatrix} \begin{bmatrix} R^{b^T} & 0 \\ 0 & R^{b^T} \end{bmatrix} \begin{bmatrix} d_i^b \\ d_j^b \end{bmatrix} \tag{3.169}$$

donde $f_i'^b$ y $f_j'^b$ son las solicitaciones, o esfuerzos, en ejes locales en los extremos de la barra y $p_i'^b$ y $p_j'^b$ representan los esfuerzos de empotramiento perfecto de la

barra expresados en ejes locales, siendo ceros en caso de que la barra no esté cargada. K'_{ii}, K'_{ij}, K'_{ji} y K'_{jj} componen la matriz de rigidez de la barra en ejes locales, R^b es su matriz de rotación y $d_i^b y d_j^b$ son los movimientos de los nudos i y j expresados en ejes globales.

Para cada barra calcularemos 12 esfuerzos, 6 correspondientes al extremo inicial y otros 6 correspondientes al nudo final. Como puede verse en la expresión anterior, estos esfuerzos se corresponden con la suma de las fuerzas que ejercen los nudos sobre los extremos de las barras más los esfuerzos de empotramiento perfecto, debido a las cargas que actúan en las mismas.

No obstante, no deberíamos olvidar que la barra puede presentar alguna excentricidad en sus extremos. Si así fuera el caso, como el mostrado en la figura 3.3, tendríamos que transformar previamente los desplazamientos en globales de los nudos a los desplazamientos locales en los extremos de las barras, tal y como indicábamos en la expresión (3.19):

$$\begin{bmatrix} f_i'^b \\ f_j'^b \end{bmatrix} = \begin{bmatrix} p_i'^b \\ p_j'^b \end{bmatrix} + \begin{bmatrix} K'_{ii} & K'_{ij} \\ K'_{ji} & K'_{jj} \end{bmatrix} \begin{bmatrix} R^{bT} & 0 \\ 0 & R^{bT} \end{bmatrix} \begin{bmatrix} H_{iA}^T & 0 \\ 0 & H_{jB}^T \end{bmatrix} \begin{bmatrix} d_A \\ d_B \end{bmatrix} \tag{3.170}$$

3.7. Cálculo de reacciones en apoyos

Las reacciones en apoyos son los esfuerzos, expresados en ejes globales, en aquellos nudos de la estructura que tienen limitado o impuesto su movimiento, a fin de equilibrar el efecto de las cargas aplicadas. La reacción en un nudo i que sea un apoyo viene dada por suma de la fuerza que le transmiten todos los elementos (nb barras y ne elementos finitos) que confluyan al mismo menos las fuerzas que actúan directamente sobre el nudo:

$$R_i = \sum_b^{nb} R_i^b + \sum_e^{ne} R_i^e - F_{Ni} \tag{3.171}$$

3.7.1. Reacciones en apoyos debidas a las barras

Supongamos que el extremo i de una barra b es un apoyo. Si transformamos a ejes globales los esfuerzos obtenidos en el extremo i de la barra, obtendremos la reacción en dicho nudo debido a la fuerza que la barra le transmite:

$$R_i^b = R^b f_i'^b \tag{3.172}$$

siendo R^b la matriz de rotación de la barra y $f_i'^b$ sus solicitaciones en el extremo i expresadas en ejes locales. Habría además que tener en cuenta que alguna de las barras pudiera ser excéntrica en el extremo que alcanza al apoyo. Si ese fuera el caso, su aportación a la reacción en el apoyo sería:

$$R_i^b = H_{iA} R^b f_i'^b \tag{3.173}$$

3.7.2. Reacciones en apoyos debidas a los triángulos

Supongamos un elemento triangular e compuesto por los nodos i, j y k, del cual resulta que el nodo i es un apoyo. La reacción en dicho nodo i debida a la fuerza que triángulo e le transmite es la siguiente:

$$R_i^e = K_{ii} d_i + K_{ij} d_j + K_{ik} d_k - f_i^e \tag{3.174}$$

siendo K_{ii}, K_{ij} y K_{ik} submatrices de la matriz de rigidez en ejes globales del elemento, d_i, d_j y d_k los desplazamientos en los nodos del triángulo y f_i^e el vector de fuerzas equivalentes en el nodo i debido a las cargas que actúan sobre el elemento.

Si el triángulo proviene de un rectángulo, habrá que multiplicar por 0.5 la expresión anterior que recoge la fuerza que el triángulo le transmite al apoyo.

3.7.3. Reacciones en apoyos debidas a los tetraedros

Si un nodo i perteneciente a un tetraedro e, de vértices i, j, k y l, se comporta como un apoyo, su reacción debida a las fuerzas que le transmite el elemento

vendrá dada por la siguiente expresión:

$$R_i^e = K_{ii}d_i + K_{ij}d_j + K_{ik}d_k + K_{il}d_l - f_i^e \tag{3.175}$$

la cual depende de las submatrices K_{ii}, K_{ij}, K_{ik} y K_{il} de la matriz de rigidez del elemento, de los desplazamientos en sus nodos y del vector de fuerzas equivalentes en el nodo i debido a las cargas aplicadas sobre el tetraedro.

3.7.4. Reacciones en apoyos debidas a los hexaedros

Sea un hexaedro e de nodos numerados del i al p del cual uno de sus nodos, por ejemplo el i, es un apoyo de la estructura. La reacción en dicho apoyo debido a las fuerzas trasmitidas por el hexaedro dependerá, como en los elementos anteriores, del siguiente conjunto de submatrices de la matriz de rigidez en ejes globales del elemento, de los desplazamientos en todos sus nodos y de la fuerzas equivalentes en el nodo que representa el apoyo:

$$R_i^e = K_{ii}d_i + K_{ij}d_j + K_{ik}d_k + ... + K_{ip}d_p - f_i^e \tag{3.176}$$

la cual depende de las submatrices K_{ii}, K_{ij}, K_{ik} y K_{il} de la matriz de rigidez del elemento, de los desplazamientos en sus nodos y del vector de fuerzas equivalentes en el nodo i debido a las cargas aplicadas sobre el hexaedro.

3.7.5. Reacciones en apoyos debidas a cargas aplicadas sobre los nudos

Si un nudo de la estructura es un apoyo y posee cargas aplicas sobre él mismo, la reacción en el nodo equivaldrá a las fuerzas aplicadas cambias de signo. Eso significa que debemos restar el valor de las cargas que actúan directamente sobre el apoyo al resto de fuerzas transmitidas por los elementos conectados al nudo:

$$R_i = -F_{Ni} \tag{3.177}$$

3.8. Cálculo de esfuerzos en puntos intermedios de las barras

Una vez calculados los movimientos en los nudos y los esfuerzos en los extremos de las barras, podrá calcularse para cada barra, y con independencia ya del resto de barras y de su posición en el espacio, los valores correspondientes a los esfuerzos (axiles, cortantes, flectores y torsores) en cada uno de los múltiples puntos intermedios en los que dicha barra se divida y para cada una de la combinaciones de hipótesis de carga. El número de puntos intermedios de cada elemento dependerá de su longitud, añadiendo nuevos valores en los puntos de aplicación inicial y final de cada carga que actúe sobre la barra.

Dichos esfuerzos en puntos intermedios los calcularemos a partir de las cargas aplicadas en las barras y por medio de las denominadas *funciones de discontinuidad* [35, 36], las cuales se utilizan en una amplia variedad de aplicaciones de ingeniería. Se distinguen dos tipos: las *funciones de Macaulay* y las *funciones de singularidad*.

Las funciones de Macaulay se definen por las siguientes expresiones, siendo $n \geq 0$:

$$F_n(x) =< x-a>^n = \begin{cases} 0 & \text{si } x < a \\ (x-a)^n & \text{si } x \geq a \end{cases} \tag{3.178}$$

A modo de ejemplo, la figura 3.20 recoge las funciones de Macaulay denominadas función escalón unitario, $F_0(x) =< x-a >^0$, y función rampa unitaria, $F_1(x) =< x-a >^1$. En el caso de la función escalón unitario, se observa la discontinuidad cuando $x = a$, de forma que se le podría asignar el valor 0 o 1. En nuestro caso, optaremos por asignarle 1 como valor, de acuerdo a esta definición de la función:

$$F_0(x) =< x-a>^0 = \begin{cases} 0 & \text{si } x < a \\ 1 & \text{si } x \geq a \end{cases} \tag{3.179}$$

Aplicadas al cálculo de esfuerzos, estas funciones nos permiten definir cargas distribuidas aplicadas sobre una barra, así como los esfuerzos correspondientes, donde x representará la coordenada de un punto intermedio a lo largo de la barra

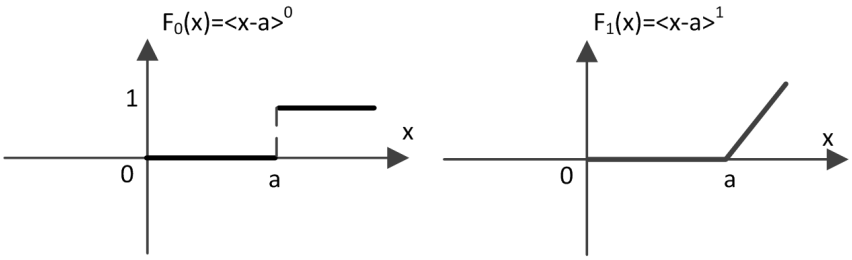


Figura 3.20: Ejemplos de funciones de Macaulay: escalón unitario y rampa unitaria.

y a será el punto donde se produce la discontinuidad en el valor de la carga distribuida o en el valor del esfuerzo producido.

Es también posible definir operaciones básicas sobre las funciones de Macaulay, como el producto por una constante, del siguiente modo:

$$Q* < x-a>^n = \begin{cases} 0 & \text{si } x < a \\ Q*(x-a)^n & \text{si } x \geq a \end{cases} \tag{3.180}$$

Como decíamos anteriormente, el segundo tipo de funciones de discontinuidad son las funciones de singularidad, definidas de acuerdo a esta otra formulación, con valores de $n < 0$:

$$F_n(x) =< x-a>^n = \begin{cases} 0 & \text{si } x \neq a \\ \pm\infty & \text{si } x = a \end{cases} \tag{3.181}$$

Un ejemplo de función de singularidad es la función impulso unitaria, mostrada en la figura 3.21 y definida así:

$$F_{-1}(x) =< x-a>^{-1} = \begin{cases} 0 & \text{si } x \neq a \\ +\infty & \text{si } x = a \end{cases} \tag{3.182}$$

Al igual que en el caso de las funciones de Macaulay, podemos aplicar operaciones elementales sobre las funciones de singularidad, como es el caso de la multiplicación de la función por un valor. Usaremos así por ejemplo estas funcio-

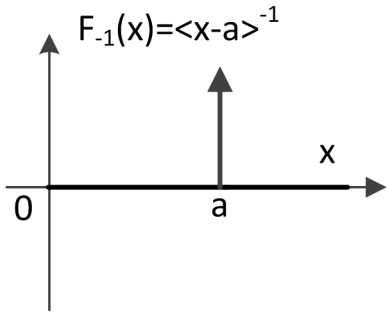


Figura 3.21: Ejemplo de función de singularidad: impulso unitaria.

nes para representar la aplicación de una carga puntual de valor Q, y el esfuerzo que produce, en un punto situado a una distancia a del origen de la barra, recordando que $n < 0$:

$$Q* < x - a >^n = \begin{cases} 0 & \text{si } x \neq a \\ Q & \text{si } x = a \end{cases} \tag{3.183}$$

En los siguientes apartados se proporcionarán los valores de los diferentes esfuerzos y momentos en función de un conjunto de cargas tipo aplicadas, así como los distintos parámetros que las definen. Entre dichas cargas tipo encontraremos acciones puntuales y repartidas. A su vez, dentro de las cargas repartidas, consideraremos cargas uniformes y triangulares, en las cuales ocurrirá que el punto final de aplicación de las mismas coincidirá con el extremo final de la barra. El efecto, sobre las leyes de esfuerzos, del resto de cargas no recogidas se puede calcular por superposición de la acción conjunta de varias de estas cargas tipo, con valores de carga positivos o negativos. Así por ejemplo, el resultado de una carga trapezoidal, que alcance el punto final de la barra, se corresponde con la aplicación de una carga uniforme y de una carga triangular.

Al resultado de la aplicación del conjunto de cargas que actúan sobre una barra en su ley de distribución de esfuerzos, recogido en las secciones siguientes, habrá que añadir el resultado de las solicitaciones $s_i'^b$ calculadas en su extremo inicial, las cuales incorporan los esfuerzos calculados sobre los nudos. Para ello, aplicaremos una carga puntual de fuerza o de momento, según corresponda para cada uno de los 6 grados de libertad y siempre en el punto inicial de la barra, equivalente al valor de la solicitación en dicho extremo.

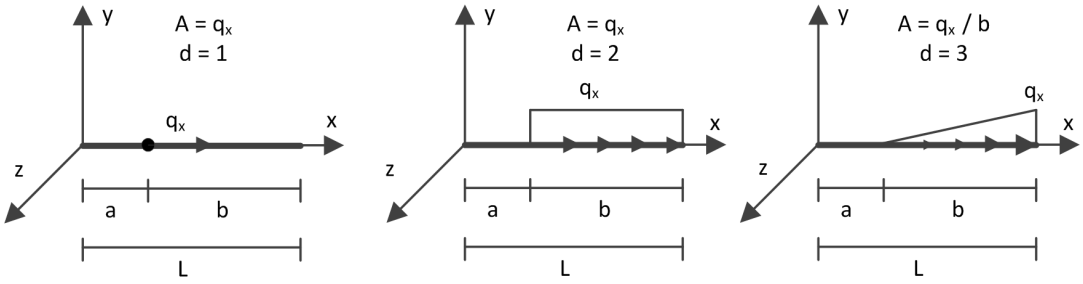


Figura 3.22: Cargas tipo y parámetros a emplear en el cálculo de esfuerzos axiles.

3.8.1. Cálculo de esfuerzos axiles

Para calcular el esfuerzo axil *N(x)* en un punto x de una barra de longitud L, comprendido entre su posición inicial ($x = 0$) y su posición final ($x = L$), tendremos que tener en cuenta la aportación de las diferentes cargas de fuerza puntuales o repartidas que actúan sobre su eje X, siendo C el número total de cargas aplicadas de este tipo, a_c el punto de aplicación inicial de la carga c y A_c y d_c valores dependientes de dicha carga c aplicada, incluidos en la figura 3.22:

$$N(x) = -\sum_{c=1}^{C} \frac{d_c * A_c* < x - a_c >^{d_c-1}}{d_c!},\ x \in [0, L] \tag{3.184}$$

3.8.2. Cálculo de esfuerzos cortantes en el eje Y y momentos flectores en el eje Z

La figura 3.23 detalla los parámetros de diferentes cargas tipo necesarios para calcular los esfuerzos cortantes $V_y(x)$ en el eje Y de la barra, así como los esfuerzos flectores $M_z(x)$ en el eje Z, tal y como se indica a continuación:

$$V_y(x) = -\sum_{c=1}^{C} \frac{d_c * A_c* < x - a_c >^{d_c-1}}{d_c!},\ x \in [0, L] \tag{3.185}$$

$$M_z(x) = \sum_{c=1}^{C} \frac{A_c* < x - a_c >^{d_c}}{d_c!},\ x \in [0, L] \tag{3.186}$$

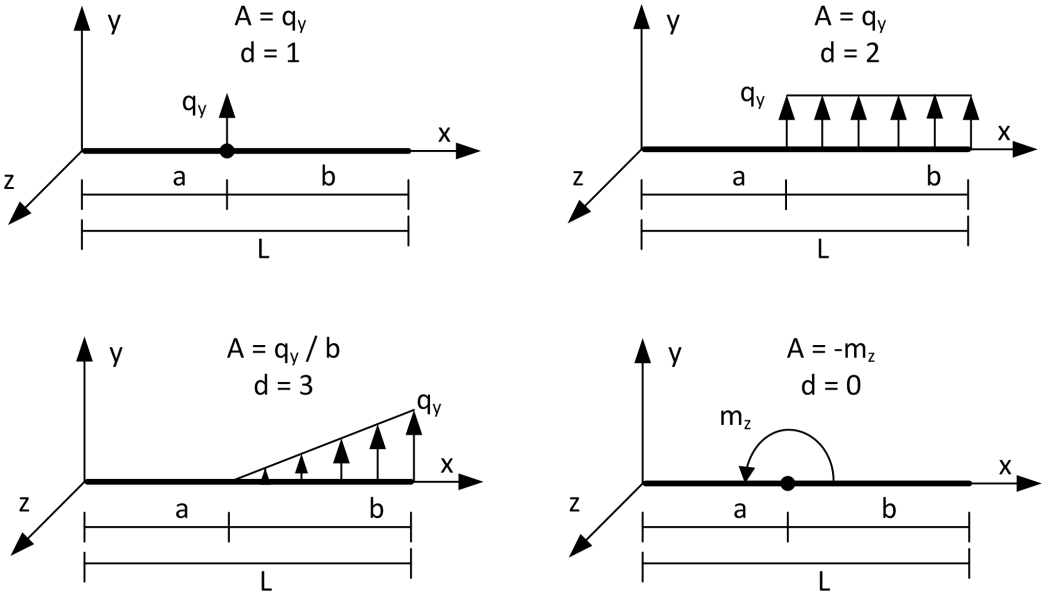


Figura 3.23: Cargas tipo y parámetros a emplear en el cálculo de esfuerzos cortantes en Y y momentos flectores en Z.

3.8.3. Cálculo de esfuerzos cortantes en el eje Z y momentos flectores en el eje Y

Para calcular este tipo de esfuerzos tenemos que seguir unos pasos muy similares a los de la sección anterior. Así, la figura 3.24 incluye los parámetros de distintas cargas tipo que dan lugar a esfuerzos cortantes $V_z(x)$ en el eje Z y flectores $M_y(x)$ en el eje Y, siguiendo para ello las siguientes expresiones:

$$V_z(x) = -\sum_{c=1}^{C} \frac{d_c * A_c * < x - a_c >^{d_c - 1}}{d_c!},\ x \in [0, L] \tag{3.187}$$

$$M_y(x) = -\sum_{c=1}^{C} \frac{A_c * < x - a_c >^{d_c}}{d_c!},\ x \in [0, L] \tag{3.188}$$

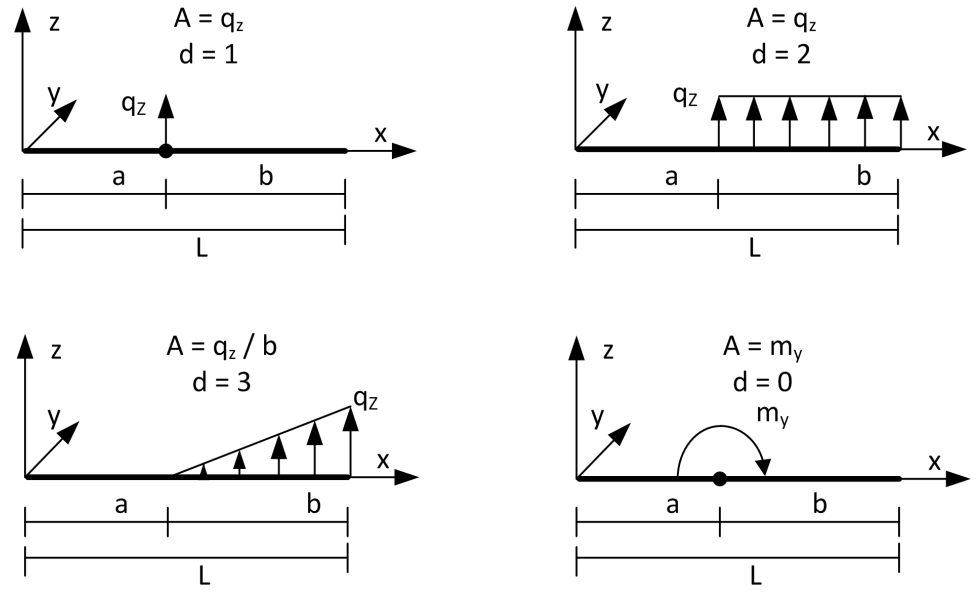


Figura 3.24: Cargas tipo y constantes a emplear en el cálculo de esfuerzos cortantes en Z y momentos flectores en Y.

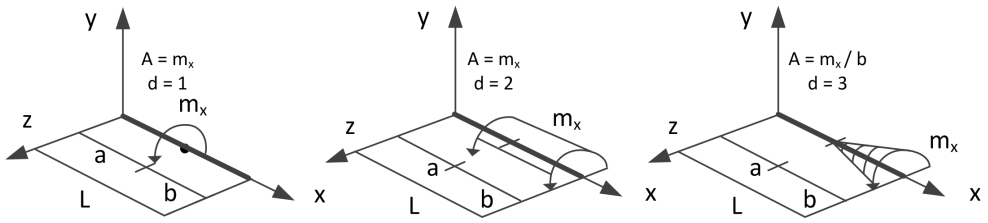


Figura 3.25: Cargas tipo y parámetros a emplear en el cálculo de momentos torsores.

3.8.4. Cálculo de momentos torsores en X

Incorporamos finalmente el modo de determinar los momentos torsores en el eje X de una barra, obtenidos a partir de un número C de cargas de momento torsor puntual o distribuido. De acuerdo a los parámetros de cada tipo de carga indicados en la figura 3.25, el valor del momento torsor en un punto x cualquiera de la barra vale:

$$M_x(x) = -\sum_{c=1}^{C} \frac{d_c * A_c * < x - a_c >^{d_c - 1}}{d_c!},\ x \in [0, L] \tag{3.189}$$

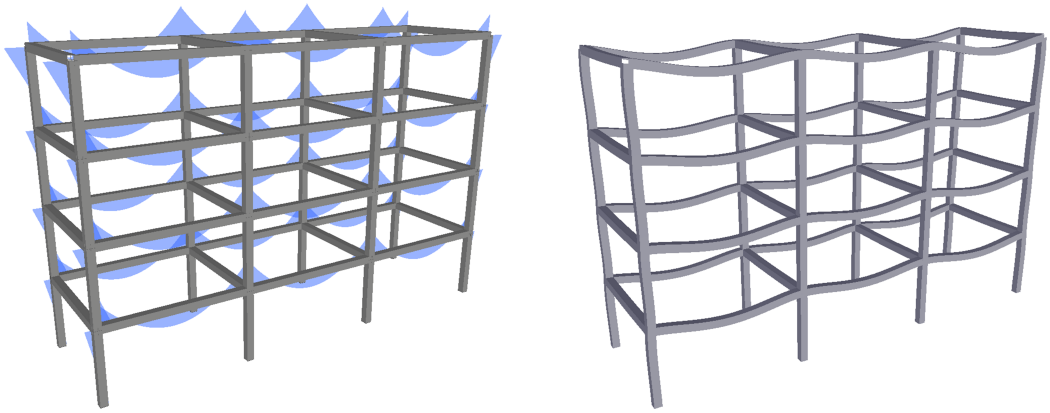
Figura 3.26: Deformación de una barra y desplazamientos de uno de sus puntos intermedios con respecto a su posición inicial indeformada.

3.9. Cálculo de deformaciones en puntos intermedios de las barras

Además de los esfuerzos en los puntos intermedios de una barra, debemos obtener su deformación, entendiéndose como tal los desplazamientos absolutos con respecto a la posición inicial indeformada de cada punto intermedio. Dichos desplazamientos, expresados en ejes locales de la barra, vendrán expresados en forma de elongaciones (desplazamientos a lo largo de la directriz de la barra), flechas (valores de desplazamiento longitudinal en los ejes Y y Z), distorsiones (ángulo de giro con respecto al eje X) y giros (ángulos de giro con respecto a los ejes Y y Z). En la figura 3.26 podemos apreciar la deformación de una barra, así como los desplazamientos en X e Y y el giro en Z de un punto intermedio de la misma.

Estos esfuerzos y deformaciones en los puntos intermedios de las barras que componen la estructura se calculan con un doble objetivo. Por un lado, para comprobar que no exceden de unas limitaciones máximas recogidas en las normativas vigentes de diseño, aunque hay que considerar que las normativas emplean las deformadas o flechas relativas de la barra, en lugar de la deformación o flecha absoluta. Por otro, nos permiten representar gráficamente el diagrama de esfuerzos de las barras y la posición deformada de la estructura, bajo una modelización alámbrica o sólida de la misma. A modo de ejemplo, la figura 3.27 nos muestra el diagrama de flectores en Z y la deformación de una sencilla estructura de barras bajo la acción de su carga de peso propio.

A partir de los esfuerzos en puntos intermedios, y por medio de la siguiente



(a) Momentos flectores en Z. (b) Deformada de la estructura.

Figura 3.27: Esfuerzos y deformadas de una estructura de barras bajo su carga de peso propio.

ecuación diferencial conocida como *ecuación de la elástica*, cabe la posición de obtener los desplazamientos que sufre el eje de la barra, desde su posición recta original a la forma curvada final en los citados puntos. Dicha ecuación debe resolverse para los planos de carga XY y XZ de la barra:

$$\frac{\partial^2 y(x)}{\partial x^2} = \frac{M(x)}{EI}, \; x \in [0, L] \tag{3.190}$$

donde *M(x)* se corresponde con los momentos flectores (en Y o en Z) en el punto x de la barra, *y(x)* es la función de la deformada de la barra (en Z o en Y) y E e I representan, respectivamente, su módulo de elasticidad longitudinal y su momento de inercia (en Y o en Z).

Sin embargo, también es posible obtener las flechas y los giros de una barra debido conjuntamente a los movimientos que sufren sus extremos y al efecto de las cargas a las que se ve sometida. Dicha deformación y giros de la barra, a causa de los desplazamientos y giros de sus extremos, puede calcularse mediante los polinomios de Hermite. Por otro lado, la deformación y los giros debido a las cargas aplicadas puede determinarse a partir de las expresiones analíticas conocidas que proporcionan la deformada de una barra biapoyada.

3.9.1. Cálculo de flechas y giros debido a los movimientos de sus extremos

Para calcular la flexión en el eje Y a los que se ven sometidos los puntos intermedios de una barra (de extremo inicial i y extremo final j), debido exclusivamente a los movimientos de los dos nudos que une, supondremos que dicha barra es indeformable horizontalmente, de modo que sus únicos 4 grados de libertad en sus extremos se corresponderán con desplazamientos verticales y giros. De este modo, la flexión de la barra vendrá dada por el siguiente polinomio cúbico:

$$Y(x) = \Psi_1(x)\delta'_{iy} + \Psi_2(x)\beta'_{iz} + \Psi_3(x)\delta'_{jy} + \Psi_4(x)\beta'_{jz},\ x \in [0,\, L] \tag{3.191}$$

donde las funciones de forma empleadas son los denominados polinomios de Hermite:

$$\begin{aligned}
\Psi_1(x) &= 1 - 3\left(\tfrac{x}{L}\right)^2 + 2\left(\tfrac{x}{L}\right)^3 \\
\Psi_2(x) &= x\left(1 - \tfrac{x}{L}\right)^2 \\
\Psi_3(x) &= 3\left(\tfrac{x}{L}\right)^2 - 2\left(\tfrac{x}{L}\right)^3 \\
\Psi_4(x) &= \tfrac{x^2}{L}\left(\tfrac{x}{L} - 1\right)
\end{aligned} \tag{3.192}$$

Entre los parámetros que aparecen en la expresión (3.191) encontramos a δ'_{iy} y δ'_{jy}, los cuales representan los desplazamientos locales en el eje Y en los extremos inicial y final de la barra, junto con β'_{iz} y β'_{jz}, que se corresponden con los giros locales en Z en los extremos de la barra:

$$\begin{aligned}
\beta'_{iz} &= \left(\theta'_{iz} - \tfrac{\varphi'_{iz}}{EI_z}\right) Gr_{iz} - \tfrac{1}{2}\left(\theta'_{jz} - \tfrac{\varphi'_{jz}}{EI_z}\right) Gr_{jz}\,(1 - Gr_{iz}) - \\
&\quad -\tfrac{\delta'_{iy}}{L}\,(1 - Gr_{iz}) + \tfrac{\delta'_{jy}}{L}\,(1 - Gr_{iz}) \\
\beta'_{jz} &= \left(\theta'_{jz} - \tfrac{\varphi'_{jz}}{EI_z}\right) Gr_{jz} - \tfrac{1}{2}\left(\theta'_{iz} - \tfrac{\varphi'_{iz}}{EI_z}\right) Gr_{iz}\left(1 - Gr_j^z\right) - \\
&\quad -\tfrac{\delta'_{iy}}{L}\,(1 - Gr_{jz}) + \tfrac{\delta'_{jy}}{L}\,(1 - Gr_{jz})
\end{aligned} \tag{3.193}$$

A su vez, θ'_{iz} y θ'_{jz} almacenan los giros de los nudos a los que confluye la barra pasados a ejes locales, mientras que φ'_{iz} y φ'_{jz} representan los giros de empotramiento perfecto en los extremos, de acuerdo a la aportación de las diferentes cargas aplicadas sobre la barra, tal y como se indicó en la sección 3.3. Como es lógico, los giros en los extremos de la barra dependen también de Gr_{iz} y Gr_{jz}, denominados grados de rigidez a giro en el eje Z de sus extremos.

Por último, conviene destacar que, antes de aplicar la expresión (3.191), hay que tener en cuenta la siguiente consideración. Si una barra está relajada longitudinalmente en uno de sus extremos, para un grado de libertad concreto, y bajo la premisa de ausencia de cargas aplicadas, ocurrirá que el desplazamiento en dicho grado de libertad y en el extremo relajado coincidirá con el del extremo sin desconectar. Eso supone por tanto que, si una barra está relajada longitudinalmente en Y en su extremo inicial, entonces δ'_{iy} será igual a δ'_{jy}, y viceversa.

De manera idéntica al cálculo de la flecha en el eje Y, calcularemos la flexión de la barra en el eje Z, conservando las funciones de forma empleadas:

$$Z(x) = \Psi_1(x)\delta'_{iz} + \Psi_2(x)\beta'_{iy} + \Psi_3(x)\delta'_{jz} + \Psi_4(x)\beta'_{jy},\ x \in [0,\, L] \tag{3.194}$$

El significado de los diferentes parámetros y el modo de obtenerlos es similar al ya mostrado para el caso de la flecha en Y, con la salvedad de estar trabajando ahora en el plano XZ en lugar de en el plano XY:

$$\begin{aligned}
\beta'_{iy} &= \left(-\theta'_{iy} + \tfrac{\varphi'_{iy}}{EI_y}\right) Gr_{iy} + \tfrac{1}{2}\left(\theta'_{jy} - \tfrac{\varphi'_{jy}}{EI_y}\right) Gr_{jy}\,(1 - Gr_{iy}) + \\
&\quad + \tfrac{\delta'_{iz}}{L}\,(1 - Gr_{iy}) - \tfrac{\delta'_{jz}}{L}\,(1 - Gr_{iy}) \\
\beta'_{jy} &= \left(-\theta'_{jy} + \tfrac{\varphi'_{jy}}{EI_y}\right) Gr_{jy} + \tfrac{1}{2}\left(\theta'_{iy} - \tfrac{\varphi'_{iy}}{EI_y}\right) Gr_{iy}\,(1 - Gr_{jy}) + \\
&\quad + \tfrac{\delta'_{iz}}{L}\,(1 - Gr_{jy}) - \tfrac{\delta'_{jz}}{L}\,(1 - Gr_{jy})
\end{aligned} \tag{3.195}$$

Para calcular los giros de la sección de la barra en cada punto intermedio, tendremos que derivar las expresiones anteriores encargadas de obtener las flechas.

Así, derivando la expresión (3.191) obtendremos los giros en el eje Z:

$$\theta_z(x) = \Psi'_1(x)\delta'_{iy} + \Psi'_2(x)\beta'_{iz} + \Psi'_3(x)\delta'_{jy} + \Psi'_4(x)\beta'_{jz},\ x \in [0,\, L] \tag{3.196}$$

donde la única diferencia reside en las derivadas de las funciones de Hermite:

$$\begin{aligned} \Psi'_1(x) &= -\tfrac{6x}{L^2} + \tfrac{6x^2}{L^3} \\ \Psi'_2(x) &= 1 + \tfrac{3x^2}{L^2} - \tfrac{4x}{L} \\ \Psi'_3(x) &= \tfrac{6x}{L^2} - \tfrac{6x^2}{L^3} \\ \Psi'_4(x) &= \tfrac{3x^2}{L^2} - \tfrac{2x}{L} \end{aligned} \tag{3.197}$$

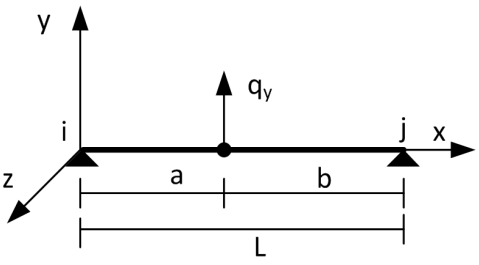
Del mismo modo, tras derivar la expresión del cálculo de la fecha en Z (3.194) dispondremos de los giros en Y, teniendo en consideración el criterio de signos establecido:

$$\theta_z(x) = -\Psi'_1(x)\delta'_{iz} - \Psi'_2(x)\beta'_{iy} - \Psi'_3(x)\delta'_{jz} - \Psi'_4(x)\beta'_{jy},\ x \in [0,\, L] \tag{3.198}$$

3.9.2. Cálculo de flechas y giros debido a las cargas aplicadas

A la deformación de flexión y giro de la barra debida a los movimientos en sus extremos, debemos sumar el efecto que sobre dicha deformación producen las cargas aplicadas directamente sobre la barra. En este apartado se incluirán las expresiones analíticas correspondientes a la flexión y al giro, en los planos XY y XZ, de una barra biapoyada, para un conjunto de cargas tipo, recogidas desde la figura 3.28 a la 3.37. Las expresiones correspondientes a las flechas se encuentran con facilidad en multitud de prontuarios, no siendo así todas aquellas relacionadas con los giros, aunque se obtienen simplemente calculando su derivada.

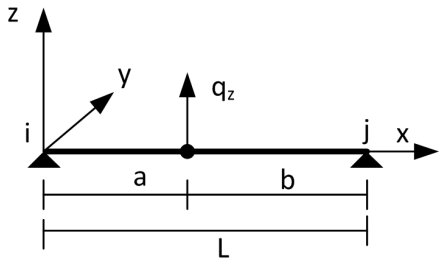
Si la carga aplicada no coincide con alguna de las que acabamos de citar, habría que descomponerla como la suma de cargas que den lugar a flechas y giros conocidos, aplicando el principio de superposición. A modo de ejemplo, la figura



$$Y(x)=\begin{cases}\dfrac{q_y b x}{6EI_z L}\left(L^2-b^2-x^2\right), & 0\le x\le a\\ \dfrac{q_y b}{6EI_z L}\left(\dfrac{L}{b}(x-a)^3+\left(L^2-b^2\right)x-x^3\right), & a\le x\le L\end{cases}$$

$$\theta_z(x)=\begin{cases}\dfrac{q_y b}{6EI_z L}\left(L^2-b^2-3x^2\right), & 0\le x\le a\\ \dfrac{q_y b}{6EI_z L}\left(\dfrac{3(x-a)^2 L}{b}+L^2-b^2-3x^2\right), & a\le x\le L\end{cases}$$

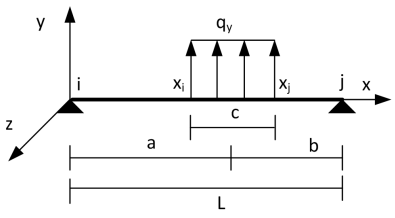
Figura 3.28: Flechas en Y y giros en Z ante una carga de fuerza puntual en Y.



$$Z(x)=\begin{cases}\dfrac{q_z b x}{6EI_y L}\left(L^2-b^2-x^2\right), & 0\le x\le a\\ \dfrac{q_z b}{6EI_y L}\left(\dfrac{L}{b}(x-a)^3+\left(L^2-b^2\right)x-x^3\right), & a\le x\le L\end{cases}$$

$$\theta_y(x)=\begin{cases}\dfrac{q_z b}{6EI_y L}\left(L^2-b^2-3x^2\right), & 0\le x\le a\\ \dfrac{q_z b}{6EI_y L}\left(\dfrac{3(x-a)^2 L}{b}+L^2-b^2-3x^2\right), & a\le x\le L\end{cases}$$

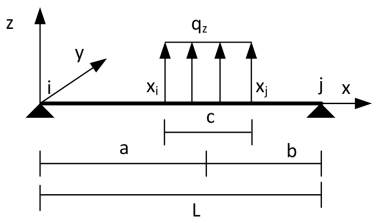
Figura 3.29: Flechas en Z y giros en Y ante una carga de fuerza puntual en Z.



$$Y(x)=\begin{cases}\dfrac{q_y bcx}{6EI_z L}\left(-x^2+a\left(L+b-\dfrac{c^2}{4a}\right)\right), & 0\le x\le x_i\\ \dfrac{q_y}{24EI_z L}\left[L\left(x-\left(a-\dfrac{c}{2}\right)\right)^4-4bcx^3+4abc\left(L+b-\dfrac{c^2}{4a}\right)x\right], & x_i\le x\le x_j\\ \dfrac{q_y ac(L-x)}{6EI_z L}\left(-(L-x)^2+b\left(L+a-\dfrac{c^2}{4b}\right)\right), & x_j\le x\le L\end{cases}$$

$$\theta_z(x)=\begin{cases}\dfrac{q_y bc}{6EI_z L}\left(aL+ab-\dfrac{c^2}{4}-3x^2\right), & 0\le x\le x_i\\ \dfrac{q_y}{48EI_z L}\left[(8x^3+(c-2a)12x^2+(6c^2-24ac+24a^2)x+(c^2-6ac+8ab+12a^2)c-8a^3)L-2bc(12x^2+c^2-4ab)\right], & x_i\le x\le x_j\\ \dfrac{q_y ac}{6EI_z L}\left((3L-6x-b)L+3x^2+\dfrac{c^2}{4}-ab\right), & x_j\le x\le L\end{cases}$$

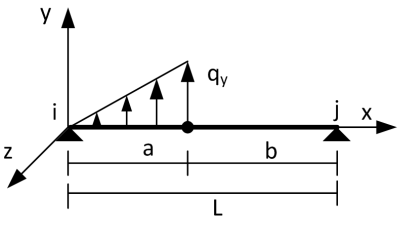
Figura 3.30: Flechas en Y y giros en Z ante una carga de fuerza distribuida constante en Y.



$$Z(x)=\begin{cases}\dfrac{q_z bcx}{6EI_y L}\left(-x^2+a\left(L+b-\dfrac{c^2}{4a}\right)\right), & 0\le x\le x_i\\ \dfrac{q_z}{24EI_y L}\left[L\left(x-\left(a-\dfrac{c}{2}\right)\right)^4-4bcx^3+4abc\left(L+b-\dfrac{c^2}{4a}\right)x\right], & x_i\le x\le x_j\\ \dfrac{q_z ac(L-x)}{6EI_y L}\left(-(L-x)^2+b\left(L+a-\dfrac{c^2}{4b}\right)\right), & x_j\le x\le L\end{cases}$$

$$\theta_y(x)=\begin{cases}\dfrac{q_z bc}{6EI_y L}\left(aL+ab-\dfrac{c^2}{4}-3x^2\right), & 0\le x\le x_i\\ \dfrac{q_z}{48EI_y L}\left[\left(8x^3+(c-2a)12x^2+6\left(c^2-4ac+4a^2\right)x+\left(c^2-6ac+8ab+12a^2\right)c-8a^3\right)L-2bc\left(12x^2+c^2-4ab\right)\right], & x_i\le x\le x_j\\ \dfrac{q_z ac}{6EI_y L}\left((3L-6x-b)L+3x^2+\dfrac{c^2}{4}-ab\right), & x_j\le x\le L\end{cases}$$

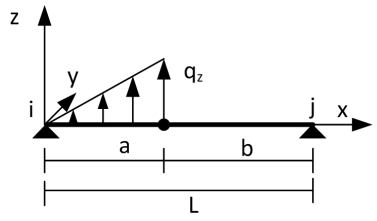
Figura 3.31: Flechas en Z y giros en Y ante una carga de fuerza distribuida constante en Z.



$$Y(x)=\begin{cases}\dfrac{q_y x}{360aEI_z L}\left(3Lx^4-10(3L-2a)a^2x^2+a^3\left(40L^2-45aL+12a^2\right)\right), & 0\le x\le a\\ \dfrac{-q_y a^2}{18EI_z L}(L-x)\left((L-x)^2-L^2\left(1-\dfrac{3a^2}{5L^2}\right)\right), & a\le x\le L\end{cases}$$

$$\theta_z(x)=\begin{cases}\dfrac{q_y}{360aEI_z L}\left(15x^4L+a^2(60a-90L)x^2+a^3\left(40L^2-45aL+12a^2\right)\right), & 0\le x\le a\\ \dfrac{q_y a^2}{90EI_z L}\left(15x^2-30xL+3a^2+10L^2\right), & a\le x\le L\end{cases}$$

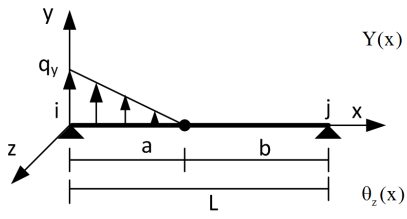
Figura 3.32: Flechas en Y y giros en Z ante una carga de fuerza triangular distribuida ascendente en Y.



$$Z(x)=\begin{cases}\dfrac{q_z x}{360aEI_y L}\left(3Lx^4-10(3L-2a)a^2x^2+a^3\left(40L^2-45aL+12a^2\right)\right), & 0\le x\le a\\ \dfrac{-q_z a^2}{18EI_y L}(L-x)\left((L-x)^2-L^2\left(1-\dfrac{3a^2}{5L^2}\right)\right), & a\le x\le L\end{cases}$$

$$\theta_y(x)=\begin{cases}\dfrac{q_z}{360aEI_y L}\left(15x^4L+a^2(60a-90L)x^2+a^3\left(40L^2-45aL+12a^2\right)\right), & 0\le x\le a\\ \dfrac{q_z a^2}{90EI_y L}\left(15x^2-30xL+3a^2+10L^2\right), & a\le x\le L\end{cases}$$

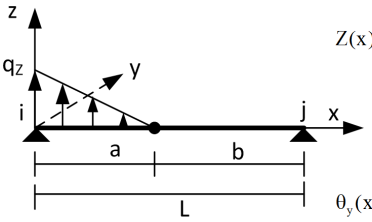
Figura 3.33: Flechas en Z y giros en Y ante una carga de fuerza triangular distribuida ascendente en Z.



$$Y(x)=\begin{cases}\dfrac{q_y x}{360aEI_z L}\left(3Lx^4-15aLx^3+10a^2(3L-a)x^2-a^3\left(20L^2-15aL+3a^2\right)\right), & 0\le x\le a\\ \dfrac{-q_y a^2}{360EI_z L}(L-x)\left(10(L-x)^2-10L^2+3a^2\right), & a\le x\le L\end{cases}$$

$$\theta_z(x)=\begin{cases}\dfrac{q_y}{360aEI_z L}\left[\left(15x^4-60ax^3+90a^2x^2+15a^4\right)L-30a^3x^2-3a^5-20a^3L^2\right], & 0\le x\le a\\ \dfrac{q_y}{360EI_z L}\left[30a^2x^2-60a^2Lx+3a^4+20a^2L^2\right], & a\le x\le L\end{cases}$$

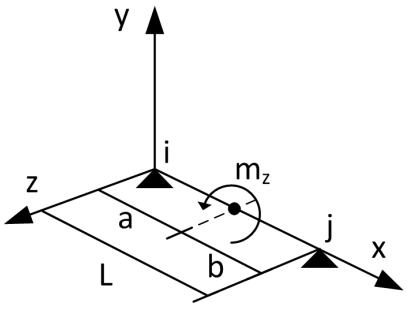
Figura 3.34: Flechas en Y y giros en Z ante una carga de fuerza triangular distribuida descendente en Y.



$$Z(x)=\begin{cases}\dfrac{q_z x}{360aEI_y L}\left(3Lx^4-15aLx^3+10a^2(3L-a)x^2-a^3\left(20L^2-15aL+3a^2\right)\right), & 0\le x\le a\\ \dfrac{-q_z a^2}{360EI_y L}(L-x)\left(10(L-x)^2-10L^2+3a^2\right), & a\le x\le L\end{cases}$$

$$\theta_y(x)=\begin{cases}\dfrac{q_z}{360aEI_y L}\left[\left(15x^4-60ax^3+90a^2x^2+15a^4\right)L-30a^3x^2-3a^5-20a^3L^2\right], & 0\le x\le a\\ \dfrac{q_z}{360EI_y L}\left[30a^2x^2-60a^2Lx+3a^4+20a^2L^2\right], & a\le x\le L\end{cases}$$

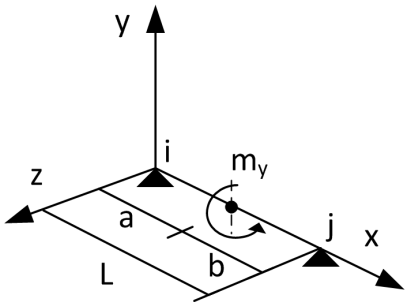
Figura 3.35: Flechas en Z y giros en Y ante una carga de fuerza triangular distribuida descendente en Z.



$$Y(x)=\begin{cases}\dfrac{-m_z Lx}{6EI_z}\left(1-\dfrac{3b^2}{L^2}-\dfrac{x^2}{L^2}\right), & 0\le x\le a\\ \dfrac{m_z L(L-x)}{6EI_z}\left(1-\dfrac{3a^2}{L^2}-\left(\dfrac{L-x}{L}\right)^2\right), & a\le x\le L\end{cases}$$

$$\theta_z(x)=\begin{cases}\dfrac{-m_z}{6EI_z L}\left(L^2-3x^2-3b^2\right), & 0\le x\le a\\ \dfrac{m_z}{6EI_z L}\left(2L^2-6xL+3x^2+3a^2\right), & a\le x\le L\end{cases}$$

Figura 3.36: Flechas en Y y giros en Z ante una carga de momento puntual en Z.



$$Z(x)=\begin{cases}\dfrac{m_y L x}{6EI_y}\left(1-\dfrac{3b^2}{L^2}-\dfrac{x^2}{L^2}\right), & 0\le x\le a\\ \dfrac{-m_y L(L-x)}{6EI_y}\left(1-\dfrac{3a^2}{L^2}-\left(\dfrac{L-x}{L}\right)^2\right), & a\le x\le L\end{cases}$$

$$\theta_y(x)=\begin{cases}\dfrac{m_y}{6EI_y L}\left(L^2-3x^2-3b^2\right), & 0\le x\le a\\ \dfrac{-m_y}{6EI_y L}\left(2L^2-6xL+3x^2+3a^2\right), & a\le x\le L\end{cases}$$

Figura 3.37: Flechas en Z y giros en Y ante una carga de momento puntual en Y.

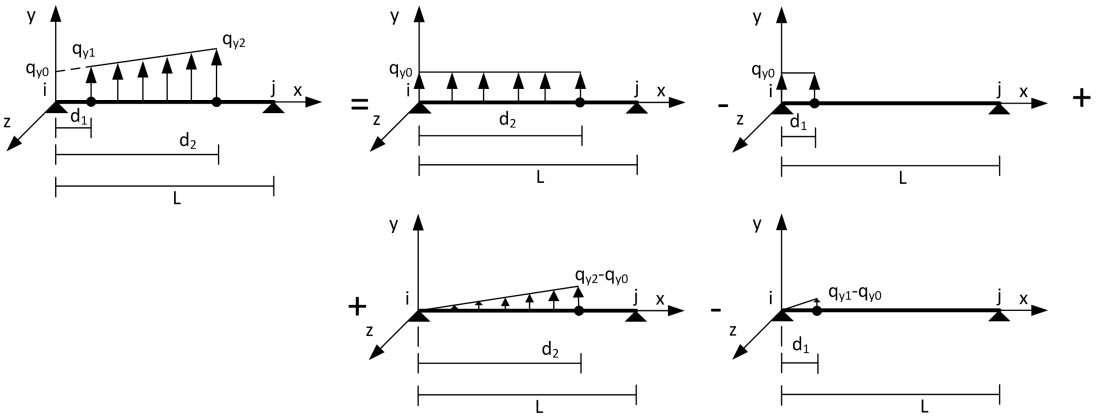


Figura 3.38: Descomposición de una carga trapezoidal distribuida en 4 cargas tipo.

3.38 muestra el caso de una carga trapezoidal distribuida y su descomposición en 4 cargas predefinidas.

3.9.3. Cálculo de elongaciones y distorsiones

La elongación es la magnitud encargada de medir el incremento de longitud de una barra cuando se le somete a un esfuerzo de tracción. Conocido por tanto el esfuerzo axil $N(x)$ en los puntos intermedios de la barra, así como el área A de su sección y el módulo de elasticidad E del material que la compone, la elongación en un punto situado a una distancia x del origen de la barra la calculamos del

siguiente modo:

$$U(x) = U(0) + \int_0^x \frac{N(x)}{EA} dx, \quad x \in [0, L] \tag{3.199}$$

siendo U(0) el valor del desplazamiento en el eje X del nudo al que confluye el extremo inicial de la barra, pasado a ejes locales de acuerdo a lo mostrado en las expresiones (3.13) y (3.19), donde se contempla la posibilidad de que la barra sea excéntrica. Para calcular dicha integral, habrá que emplear alguna de las múltiples técnicas de cuadratura numérica existentes, como por ejemplo la técnica de Gauss-Legendre.

Como es bien sabido, la barra podrá presentar alguna discontinuidad en sus extremos, siendo la relajación longitudinal en X la única que afectará al cálculo de la elongación. Puesto que dicha relajación no se producirá conjuntamente en ambos extremos de la barra, procederemos del modo siguiente. Si la barra está desconectada en su extremo derecho, iremos calculando la elongación desde el extremo inicial hasta el final, conocida la elongación en ese extremo inicial. Por el contrario, si la barra está desconectada en su extremo inicial, calcularemos la elongación del extremo final al inicial, conociendo la elongación en el extremo final de la barra.

Por otro lado, la distorsión es la magnitud que expresa el ángulo de giro de la directriz de la barra cuando se le somete a un momento torsor. Para determinar su valor en un punto situado a una distancia x del origen de la barra es necesario conocer, a priori, el momento torsor $M_x(x)$ en dicho punto, junto con el módulo de elasticidad transversal del material de la barra (G) y su módulo de inercia a torsión (J). Además, previamente habremos calculado $\theta_x(0)$ o, lo que es lo mismo, el ángulo de giro en el X en el extremo inicial de la barra, expresado en ejes locales. De esta forma, resulta que:

$$\theta_x(x) = \theta_x(0) + \int_0^x \frac{M(x)}{GJ} dx, \quad x \in [0, L] \tag{3.200}$$

Al igual que comentábamos anteriormente, debemos emplear alguna técnica como la de Gauss-Legendre para calcular numéricamente la integral. Sin embargo, el diagrama del momento torsor puede presentar discontinuidades en aquellos

puntos intermedios de la barra donde se haya aplicado una carga de momento torsor. Para solventar dicho inconveniente, convendrá calcular la integral por partes. A modo de ejemplo, si pretendemos calcular la distorsión en un punto x y ocurre que un punto $x_i < x$ aparece la citada discontinuidad, tras aplicar una carga del tipo mencionado, diremos que:

$$\theta_x(x) = \theta_x(0) + \int_0^{x_i} \frac{M(x)}{GJ} dx + \int_{x_i}^{x} \frac{M(x)}{GJ} dx, \;\; x \in [0,\, L] \tag{3.201}$$

Si la barra presentara una desconexión total o parcial a giro en el eje X en uno de sus extremos (conjuntamente en ambos no puede presentarla), calcularíamos la distorsión de modo similar al procedimiento indicado en el caso de las elongaciones, es decir, comenzaríamos desde aquel extremo que no esté desconectado, donde la distorsión es conocida inicialmente, e iríamos avanzando hacia el extremo contrario, calculando la distorsión en cada uno de los puntos intermedios.

3.10. Cálculo de deformaciones, esfuerzos y tensiones en elementos finitos

Una vez que se han calculado los desplazamientos en los nodos de un elemento finito, lo que procede es hallar otras magnitudes de interés. En consecuencia, en este apartado mostraremos la forma de calcular la deformación unitaria y la tensión en un nodo de acuerdo al tipo de elemento finito al que pertenece (triángulo, tetraedro o hexaedro), además de los esfuerzos (cortantes y momentos flectores) en los nodos de los elementos finitos bidimensionales. De aquí en adelante, el vector con las deformaciones unitarias en un nodo i de un elemento e será de la forma:

$$\varepsilon_i^e = \left[\begin{array}{cccccc} \varepsilon_{ix} & \varepsilon_{iy} & \varepsilon_{iz} & \gamma_{ixy} & \gamma_{ixz} & \gamma_{iyz} \end{array} \right]^T \tag{3.202}$$

Sin embargo, ya que un nodo puede formar parte de más de un elemento finito, su deformación unitaria será el promedio de la deformación unitaria debida a cada uno de los elementos finitos a los que pertenece. Así, la deformación unitaria del

nodo i de la estructura al que confluyen ne elementos finitos se obtendrá como:

$$\varepsilon_i = \frac{\sum_{e}^{ne} \varepsilon_i^e}{ne} \tag{3.203}$$

Adicionalmente, debemos obtener las tensiones en cada nodo, a partir de sus deformaciones unitarias. Aunque es habitual almacenar la tensión del nodo i de un elemento e en forma de una matriz simétrica de tamaño 3x3, denominada tensor de tensiones, por nuestra parte la guardaremos como un vector σ_i^e de 6 elementos:

$$\sigma_i^e = \left[\begin{array}{cccccc} \sigma_{ix} & \sigma_{iy} & \sigma_{iz} & \tau_{ixy} & \tau_{ixz} & \tau_{iyz} \end{array} \right]^T \tag{3.204}$$

Las componentes σ del vector expresan tensiones normales en una dirección, mientras que las componentes τ expresan tensiones tangenciales en un plano. De igual manera a como ocurre con las deformaciones unitarias, las tensiones en un nodo i se obtienen como el promedio de la tensión debida a cada elemento finito conectado al nodo:

$$\sigma_i = \frac{\sum_{e}^{ne} \sigma_i^e}{ne} \tag{3.205}$$

Finalmente, obtendremos los esfuerzos cortantes T_i^e y los momentos flectores m_i^e en un nodo i de un triángulo e:

$$T_i^e = \left[\begin{array}{ccc} T_{ix} & T_{iy} & T_{iz} \end{array} \right]^T \tag{3.206}$$

$$m_i^e = \left[\begin{array}{cccccc} m_{ix} & m_{iy} & m_{iz} & m_{ixy} & m_{ixz} & m_{iyz} \end{array} \right]^T \tag{3.207}$$

calculando los esfuerzos cortantes y los momentos flectores en el nodo i como la media de los esfuerzos ocasionados por todos los triángulos (nt) que confluyan al nodo:

$$T_i = \frac{\sum_{e}^{nt} T_i^e}{nt}, \qquad m_i = \frac{\sum_{e}^{nt} m_i^e}{nt} \tag{3.208}$$

Nodo	ζ_1	ζ_2	ζ_3
1	1	0	0
2	0	1	0
3	0	0	1

Tabla 3.3: Valor de las coordenadas de área en los nodos del triángulo.

3.10.1. Deformaciones y tensiones en triángulos

Para obtener la deformación unitaria en los nodos de un triángulo e de área A y espesor h, tendremos en cuenta su comportamiento como membrana [34]. Así, la deformación unitaria en el nodo i se calcula del siguiente modo, expresada en ejes locales del elemento:

$$\varepsilon_i'^e = \begin{bmatrix} \varepsilon_x' \\ \varepsilon_y' \\ \varepsilon_{xy}' \end{bmatrix} = \frac{1}{Ah} L^T d'^e + Te\beta_0^e \left(Q_1\zeta_1 + Q_2\zeta_2 + Q_3\zeta_3\right) \tilde{T}_{\theta u} d'^e,\ i = 1,2,3 \quad (3.209)$$

Conviene aclarar que el vector d'^e estará formado únicamente por los desplazamientos, en ejes locales, de aquellos grados de libertad de los nodos i, j y k del elemento relacionados con el comportamiento de membrana, es decir:

$$d'^e = \begin{bmatrix} \delta_{ix}' & \delta_{iy}' & \theta_{iz}' & \delta_{jx}' & \delta_{jy}' & \theta_{jz}' & \delta_{kx}' & \delta_{ky}' & \theta_{kz}' \end{bmatrix}^T \quad (3.210)$$

El significado así como los elementos que componen las matrices L, T_e, $\tilde{T}_{\theta u}$, Q_1, Q_2 y Q_3 quedó recogido en el apartado 3.2.3.2. En el caso de un material isótropo, el valor recomendado para β_0^e es de 3/2. Por otro lado, ζ_1, ζ_2 y ζ_3 representan las denominadas coordenadas de área en un nodo del triángulo (ver tabla 3.3), expresadas en función de sus coordenadas naturales:

$$\begin{aligned} \zeta_1 &= 1 - \xi - \eta \\ \zeta_2 &= \xi \\ \zeta_3 &= \eta \end{aligned} \quad (3.211)$$

Procederá, a continuación, pasar el citado vector de deformaciones unitarias a ejes globales, empleando para ello la matriz de rotación del triángulo y generando

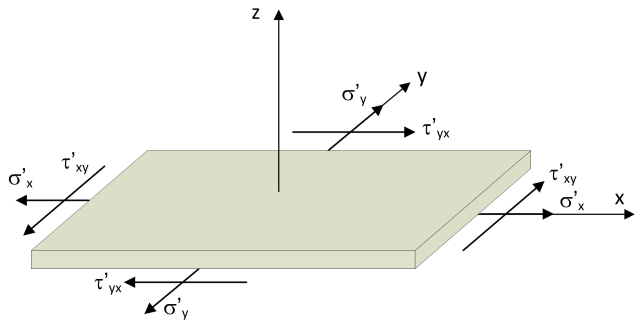


Figura 3.39: Criterio de signo para las tensiones, en ejes locales, en un elemento bidimensional.

el tensor de deformaciones del nodo:

$$E_i^e = \begin{bmatrix} E_{ix} & E_{ixy} & E_{ixz} \\ E_{iyx} & E_{iy} & E_{iyz} \\ E_{izx} & E_{izy} & E_{iz} \end{bmatrix} = R^e \begin{bmatrix} \varepsilon'_{ix} & \frac{1}{2}\varepsilon'_{ixy} & 0 \\ \frac{1}{2}\varepsilon'_{ixy} & \varepsilon'_{iy} & 0 \\ 0 & 0 & 0 \end{bmatrix} R^{eT}, \quad i = 1, 2, 3 \quad (3.212)$$

Como la matriz obtenida es simétrica, formaremos un vector de deformaciones unitarias en ejes globales a partir de su parte triangular superior o inferior, modificando oportunamente los elementos que no pertenezcan a la diagonal, el cual emplearemos para promediar con la deformación unitaria del resto de elementos finitos que converjan al nodo:

$$\varepsilon_i^e = \begin{bmatrix} E_{ix} & E_{iy} & E_{iz} & 2E_{ixy} & 2E_{ixz} & 2E_{iyz} \end{bmatrix}^T \quad (3.213)$$

En lo que respecta a las tensiones de membrana en los nodos del triángulo, comenzaremos generando dichos valores a partir de las deformaciones unitarias y a partir de la matriz de elasticidad del material E_m, cuyos valores quedaron recogidos en la expresión (3.39):

$$\sigma_i'^e = \begin{bmatrix} \sigma'_{ix} \\ \sigma'_{iy} \\ \tau'_{ixy} \end{bmatrix} = E_m \varepsilon_i'^e, \quad i = 1, 2, 3 \quad (3.214)$$

Puesto que dichas tensiones estarán expresadas en ejes locales del elemento (ver figura 3.39), procederá transformarlas a ejes globales de la estructura, empleando para ello la matriz de rotación del triángulo y dando lugar a la matriz tensor de

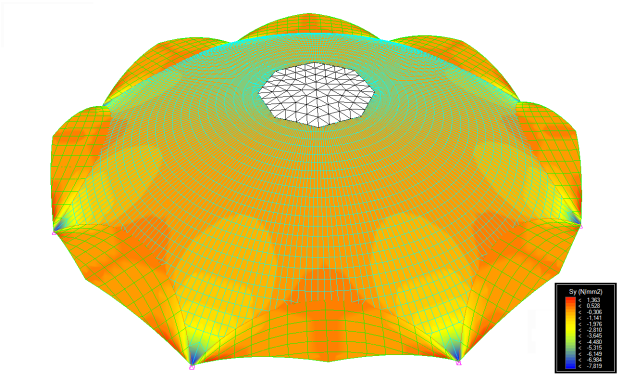


Figura 3.40: Representación gráfica de la tensión de membrana σ_y debida al peso propio de la estructura.

tensiones:

$$T_i^e = \begin{bmatrix} \sigma_{ix} & \tau_{ixy} & \tau_{ixz} \\ \tau_{iyx} & \sigma_{ix} & \tau_{iyz} \\ \tau_{izx} & \tau_{izy} & \sigma_{ix} \end{bmatrix} = R^e \begin{bmatrix} \sigma'_{ix} & \tau'_{ixy} & 0 \\ \tau'_{ixy} & \sigma'_{iy} & 0 \\ 0 & 0 & 0 \end{bmatrix} R^{eT}, \quad i = 1, 2, 3 \tag{3.215}$$

Teniendo en cuenta la simetría de la matriz, nos quedaremos con su parte triangular inferior o superior, formando el vector de tensiones que emplearemos para promediar con el valor que proporcionan los demás elementos finitos a los que el nodo pertenece:

$$\sigma_i^e = \begin{bmatrix} \sigma_{ix} & \sigma_{iy} & \sigma_{iz} & \tau_{ixy} & \tau_{ixz} & \tau_{iyz} \end{bmatrix}^T \tag{3.216}$$

A modo de ejemplo, la figura 3.40 nos muestra, de manera gráfica, las tensiones de membrana σ_y de una estructura debidas a su carga de peso propio. El hecho de que la zona central no aparezca coloreada se debe a que dicha parte de la estructura se ha modelizado con barras y no con elementos triangulares, como pudiera parecer.

3.10.2. Esfuerzos cortantes y momentos flectores en triángulos

Si bien en el apartado anterior hemos tenido en cuenta el comportamiento de la placa como membrana, en este apartado consideraremos su respuesta a

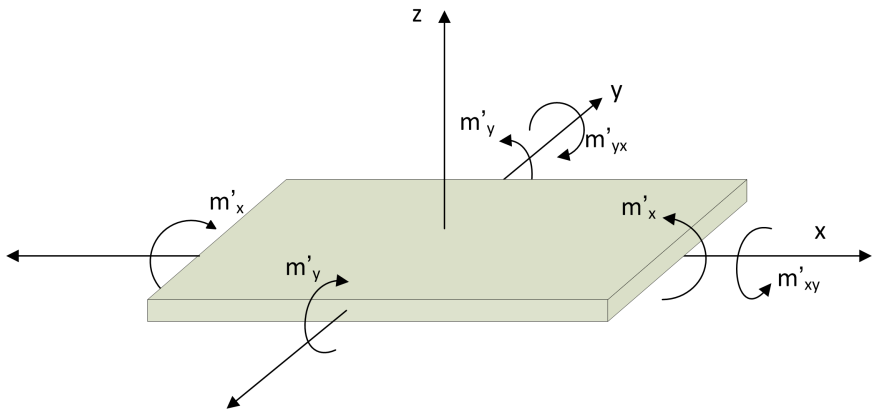


Figura 3.41: Criterio de signos de los momentos flectores en un elemento bidimensional.

flexión. Para ello, comenzaremos proporcionando las que podríamos denominar como curvaturas unitarias en un nodo i del triángulo e, siendo B la matriz de deformación del elemento detallada en la expresión (3.33):

$$\varepsilon_i'^e = B\left(\xi_i, \eta_i\right) d'^e, \;\; i = 1, 2, 3 \tag{3.217}$$

El vector d'^e incluirá ahora únicamente los movimientos longitudinales y los giros, en ejes locales, de aquellos grados de libertad relacionados con el comportamiento a flexión, para los 3 nodos i, j y k del triángulo:

$$d'^e = \left[\begin{array}{cccccccccc} \delta'_{iz} & \theta'_{ix} & \theta'_{iy} & \delta'_{jz} & \theta'_{jx} & \theta'_{jy} & \delta'_{kz} & \theta'_{kx} & \theta'_{ky} \end{array} \right]^T \tag{3.218}$$

Si premultiplicamos dichas curvaturas unitarias por la matriz constitutiva del material D_b, incluida en la expresión (3.32), obtendremos los esfuerzos a flexión, o momentos flectores, de cada nodo del triángulo [33]:

$$m_i'^e = \left[\begin{array}{c} m'_x \\ m'_y \\ m'_{xy} \end{array} \right] = D_b \varepsilon_i'^e, \;\; i = 1, 2, 3 \tag{3.219}$$

La figura 3.41 recoge los signos positivos de dichos momentos flectores, expresados en ejes locales del elemento. Como es lógico, convertiremos dichos esfuerzos a ejes

globales, dando lugar a la matriz llamada tensor de momentos:

$$M_i^e = \begin{bmatrix} m_{ix} & m_{ixy} & m_{ixz} \\ m_{iyx} & m_{iy} & m_{iyz} \\ m_{izx} & m_{izy} & m_{iz} \end{bmatrix} = R^e \begin{bmatrix} m'_{ix} & m'_{ixy} & 0 \\ m'_{ixy} & m'_{iy} & 0 \\ 0 & 0 & 0 \end{bmatrix} R^{e^T},\ i = 1, 2, 3 \quad (3.220)$$

Dada la simetría que presenta dicha matriz, obtendremos un vector, denominado tensor de momentos, con los elementos de la parte triangular inferior o superior:

$$m_i^e = \begin{bmatrix} m_{ix} & m_{iy} & m_{iz} & m_{ixy} & m_{ixz} & m_{iyz} \end{bmatrix}^T \quad (3.221)$$

el cual se empleará para promediar con los valores proporcionados en el mismo nodo por los nt triángulos de los que el nodo es vértice.

Para finalizar, determinaremos los componentes del vector con los esfuerzos cortantes de placa en los ejes locales del elemento, los cuales presentan un valor idéntico en cada nodo del triángulo:

$$T_i'^e = \begin{bmatrix} T'_{ix} \\ T'_{iy} \end{bmatrix} = B^T m'^e,\ i = 1, 2, 3 \quad (3.222)$$

siendo B la matriz de la expresión (3.29) y m'^e el vector con los esfuerzos a flexión, en ejes locales, en los 3 nodos del triángulo:

$$m'^e = \begin{bmatrix} m_i'^e \\ m_j'^e \\ m_k'^e \end{bmatrix} =$$

$$= \begin{bmatrix} m'_{ix} & m'_{iy} & m'_{ixy} & m'_{jx} & m'_{jy} & m'_{jxy} & m'_{kx} & m'_{ky} & m'_{kxy} \end{bmatrix}^T \quad (3.223)$$

Por convenio, le cambiaremos el signo a los cortantes de placa obtenidos, los pasaremos a ejes globales y los promediaremos con el resto de triángulos que confluyen al nodo:

$$T_i^e = \begin{bmatrix} T_{ix} \\ T_{iy} \\ T_{iz} \end{bmatrix} = R^e \begin{bmatrix} -T'_{ix} \\ -T'_{iy} \\ 0 \end{bmatrix},\ i = 1, 2, 3 \quad (3.224)$$

3.10.3. Deformaciones unitarias y tensiones en tetraedros

El modo de obtener las deformaciones unitarias en un nodo i de un tetraedro e es el siguiente, siendo d^e el vector con los desplazamientos en los 4 nodos del elemento:

$$\varepsilon_i^e = Bd^e = \sum_{n=1}^{4} B_n d_n^e, \quad i = 1, 2, 3, 4 \tag{3.225}$$

lo cual significa que las deformaciones unitarias son idénticas en cada nodo. Por último, las tensiones del nodo i del tetraedro e las obtendremos a partir de la matriz D constitutiva del material (ver expresión (3.70)):

$$\sigma_i^e = D\varepsilon_i^e, \quad i = 1, 2, 3, 4 \tag{3.226}$$

Dichas deformaciones y tensiones estarán expresadas en ejes globales y se promediarán con las deformaciones unitarias y las tensiones aportadas por el resto de elementos finitos que alcancen al nodo.

3.10.4. Deformaciones unitarias y tensor de tensiones en hexaedros

Inicialmente, comenzaremos calculando las deformaciones unitarias en los 8 puntos de integración utilizados, a partir de los desplazamientos d^e en los nodos del elemento:

$$\varepsilon_p^e = B\left(\xi_p, \eta_p, \zeta_p\right) d^e = \sum_{i=1}^{8} B_i\left(\xi_p, \eta_p, \zeta_p\right) d_i^e, \quad p = 1, 2, ..., 8 \tag{3.227}$$

A continuación, obtendremos las deformaciones unitarias en cada nodo del hexaedro, interpolando los valores calculados previamente en los puntos de integración mediante las funciones de forma de cada nodo:

$$\varepsilon_i^e = \sum_{p=1}^{8} \mathrm{N}_i \varepsilon_p^e, \quad i = 1, 2, ..., 8 \tag{3.228}$$

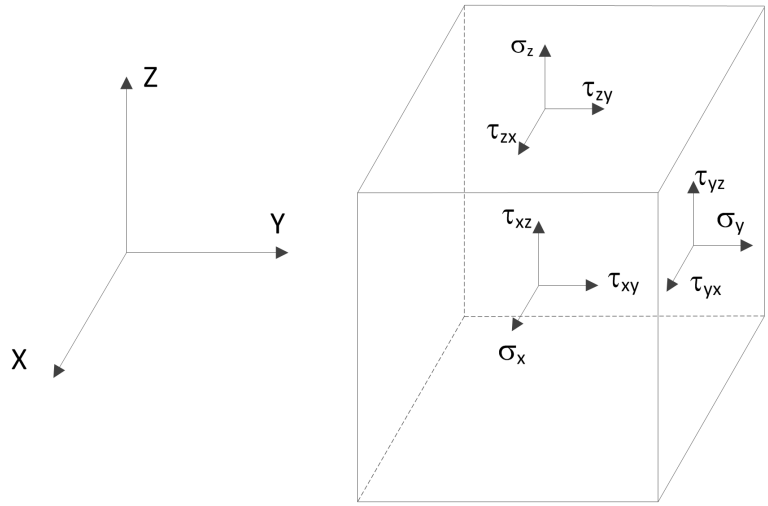


Figura 3.42: Criterio de signos de las tensiones en los laterales de un cubo.

siendo:

$$\mathrm{N_i} = \begin{bmatrix} N_i(\xi_p,\eta_p,\zeta_p) & 0 & 0 & 0 & 0 & 0 \\ 0 & N_i(\xi_p,\eta_p,\zeta_p) & 0 & 0 & 0 & 0 \\ 0 & 0 & N_i(\xi_p,\eta_p,\zeta_p) & 0 & 0 & 0 \\ 0 & 0 & 0 & N_i(\xi_p,\eta_p,\zeta_p) & 0 & 0 \\ 0 & 0 & 0 & 0 & N_i(\xi_p,\eta_p,\zeta_p) & 0 \\ 0 & 0 & 0 & 0 & 0 & N_i(\xi_p,\eta_p,\zeta_p) \end{bmatrix} \tag{3.229}$$

Finalmente, procederemos de manera idéntica a lo mostrado para los tetraedros en lo que respecta al cálculo de la tensión en el nodo i de un elemento hexáedrico e, siempre considerando la matriz D constitutiva del material recogida en (3.70):

$$\sigma_i^e = D\varepsilon_i^e, \quad i = 1, 2, ..., 8 \tag{3.230}$$

Las deformaciones unitarias y las tensiones calculadas están ya expresadas en ejes globales, con lo cual únicamente nos resta promediarlas con los valores aportados por los elementos finitos que confluyen al nodo. La figura 3.42 muestra las componentes del tensor de tensiones en las caras de un elemento tridimensional.

Capítulo 4

El Análisis Dinámico Lineal de Estructuras

Este capítulo está dedicado al análisis dinámico de estructuras, en régimen lineal. En él se describen las principales diferencias con respecto al análisis estático, y los diferentes tipos de análisis que permiten analizar dinámicamente una estructura, como pueden ser el análisis basado en métodos de integración directa, el análisis modal por superposición directa o el análisis modal espectral. Mientras que los dos primeros obtienen la respuesta a lo largo del tiempo, el tercero obtiene la respuesta máxima del sistema. Además de incluir la generación de las matrices de rigidez y masa de cada tipo de elemento y su posterior ensamblaje al sistema global, se recogen también los diferentes métodos implementados en esta tesis doctoral para cada tipo de análisis, incluyendo las características y la formulación matemática de cada uno de ellos. Más concretamente, se detalla como obtener la matriz de rigidez, masa y amortiguamiento de la estructura, junto con los desplazamientos, velocidades y aceleraciones en los nudos. El resto de cálculos a proporcionar (esfuerzos en extremos de las barras, reacciones en apoyos, esfuerzos y deformaciones en puntos intermedios o tensiones y esfuerzos en elementos finitos) se obtendrán de manera idéntica a lo recogido en el capítulo anterior.

4.1. Introducción

El comportamiento dinámico de una estructura difiere del estático en dos aspectos fundamentales. El primero de ellos se refiere a la posible dependencia con el tiempo del valor de la carga, lo que lleva consigo la determinación de la respuesta en cada uno de los intervalos de tiempo de actuación de la misma. El segundo y más significativo radica en la aparición, durante el movimiento, de fuerzas de inercia relacionadas con la masa del sistema.

Determinadas acciones tienen una intervención claramente dinámica y en este sentido cabe citar: terremotos, acciones de viento, fuerzas aplicadas rápidamente, ondas explosivas, fuerzas o masas móviles (paso de vehículos o trenes sobre puentes), fuerzas periódicas (máquinas rotativas, turbinas, etc.), impactos, tránsito de personas y oleaje. Todas estas acciones, definidas mediante su intensidad, dirección y sentido para el conjunto de instantes de tiempo en los que actúan, provocan una respuesta dinámica de la estructura, de modo que sus diferentes puntos quedan sometidos a velocidades y aceleraciones que generan fuerzas a tener en cuenta en el equilibrio que, en cada instante, debe existir en todas los nudos de la estructura.

Las características físicas a considerar en la definición del modelo matemático de la estructura, que expresa la relación entre la acción dinámica y la respuesta, son la rigidez (al igual que en el análisis estático) junto con la masa y el amortiguamiento. Así, el análisis dinámico de una estructura a lo largo del tiempo obtiene la respuesta dinámica de la misma, la cual se caracteriza por determinar tanto la carga dinámica actuante como los desplazamientos, velocidades y aceleraciones en los nudos, así como los esfuerzos, las deformaciones o las tensiones en cualquier punto de la estructura para cada instante de tiempo, dando lugar a la historia temporal de la respuesta. Es importante resaltar que una vez conocidos los desplazamientos en los nudos, los esfuerzos en los extremos de las barras, las reacciones en apoyos, los valores de esfuerzos y deformaciones en los puntos intermedios de las barras y las tensiones en los vértices de los elementos finitos pueden determinarse, para cada instante de tiempo, utilizando los conceptos ya citados en el análisis estático.

A diferencia por tanto del análisis estático en el cual se producía una discreti-

zación espacial del edificio, dando lugar a un modelo matemático empleado para obtener la respuesta estructural en los nudos, en un análisis dinámico es también necesario realizar una discretización espacial, a fin de conocer el comportamiento de la estructura en un número finito de instantes de tiempo. Que duda cabe que el modelo empleado y la cantidad de instantes de tiempo en los que se conoce el comportamiento de la estructura determinan la fiabilidad de la respuesta calculada.

En algunos casos, la sincronización entre el periodo propio de la estructura y el de aplicación de las cargas puede provocar un fenómeno de resonancia, de modo que la estructura, por muy resistente que sea, acabará quedando fuera de servicio. Como ejemplo significativo y muy conocido podemos citar el puente de Tacoma, ya que se produjo su rotura al coincidir su frecuencia de torsión con la de los remolinos del viento actuantes sobre el mismo.

Dentro del campo general de las estructuras de edificación, los movimientos del terreno o terremotos representan en muchos países las acciones de mayor riesgo en cuanto a las pérdidas humanas y los daños materiales que ocasionan, donde sólo un adecuado y preciso conocimiento de la respuesta dinámica estructural puede dar lugar a un estudio suficientemente exacto del problema.

Sin embargo, en países con una actividad sísmica no muy acusada, como es España, existe en arquitectura y en ingeniería civil un interés creciente por el análisis dinámico, debido a los siguientes aspectos:

- Se tiende a proyectar estructuras esbeltas y ligeras, las cuales son bastante susceptibles de experimentar fenómenos vibratorios.
- Algunas excitaciones son cada vez más intensas debido a las mayores prestaciones de los productos de la tecnología moderna tales como ascensores, metros, trenes de alta velocidad, etc.
- Las cargas dinámicas presentan características propias que requieren un tratamiento específico y distinto al del resto de acciones.
- Un análisis dinámico puede poner al descubierto un posible fallo de servicio imposible de predecir con un análisis puramente estático.

- La existencia de la Norma de Construcción Sismorresistente (NCSE-02), la cual es de vigente cumplimiento y aplicación al proyecto, construcción y conservación de edificaciones de nueva planta.

En el caso particular de los edificios, la modelización más sencilla y a la vez ampliamente utilizada durante mucho tiempo tenía en cuenta que la masa está generalmente agrupada en unas zonas de la estructura fácilmente identificables, bajo el denominado modelo de masas concentradas. Esto da lugar a modelos dinámicos de fácil aplicación y resolución, con un número de grados de libertad muy reducido y que proporcionan resultados válidos sólo bajo determinadas circunstancias.

En estos modelos, basados en edificios simétricos, la masa total del edificio se concentra en puntos predefinidos del mismo, los cuales se encuentran concretamente a nivel de los forjados, simulándose de esta manera el efecto de las fuerzas de inercia reales que aparecen en la estructura durante su vibración. Se lleva también a cabo la suposición de que el resto de la estructura tiene solamente rigidez, pero no masa, y su comportamiento se describe mediante barras elásticas sin masa, que conectan entre sí las masas concentradas.

Si además se introducen las simplificaciones de despreciar la deformación por esfuerzo axil de los pilares y de considerar que los forjados son perfectamente rígidos tanto a flexión como a axil, el edificio se modeliza mediante el sistema de masas concentradas denominado modelo de edificio simple o de edificio de cortante, el cual constituye el modelo más sencillo con varios grados de libertad que se utilizó habitualmente en el cálculo dinámico para describir el comportamiento de una estructura. Debido en consecuencia a que los únicos movimientos de los nudos son los horizontales, esta simplificación excesiva conduce a que el número de grados de libertad de un edificio quedará reducido únicamente al número de plantas que lo configuran (1 grado de libertad por cada nudo representando a cada planta).

Tanto el Eurocódigo E8 como la normativa de diseño española aceptan el uso del citado método de cálculo simplificado, de manera que todas aquellas construcciones que cumplan ciertos requisitos de regularidad en altura y planta se podrán asimilar a un modelo unidimensional constituido por un oscilador simple

y con un solo grado de libertad por planta, realizando su análisis a través de fuerzas horizontales equivalentes a las de los terremotos. Dichas normativas dan además los procedimientos muy sencillos de obtención de las fuerzas estáticas equivalentes a las fuerzas sísmicas. Ejemplos de dichos requisitos en el caso de la normativa española son:

- Número de plantas sobre rasante inferior a veinte.
- Altura del edificio inferior a sesenta metros, con regularidad geométrica en planta y en alzado, sin entrantes ni salientes.
- Soportes continuos hasta la cimentación, uniformemente distribuidos en planta y sin cambios bruscos de rigidez entre plantas.
- Regularidad mecánica en la distribución de rigideces, resistencias y masas con los centros de gravedad y de torsión de todas las plantas sobre la misma vertical.
- La excentricidad del centro de las masas respecto al de la torsión será inferior al 10 % de la dimensión en planta del edificio en cada una de las direcciones principales.

Si queremos someter al edificio a dos componentes horizontales ortogonales de aceleración sísmica, se tendrán que realizar dos cálculos diferentes, uno para cada componente de aceleración, utilizando en cada uno de ellos el modelo de edificio de cortante correspondiente y considerando la traslación horizontal como único grado de libertad por planta, combinando posteriormente los resultados.

En el supuesto en el que la dirección de la carga dinámica, como puede ser un terremoto, no esté contenida en el plano de simetría del edificio, será necesario considerar en el modelo grados de libertad adicionales, como es la posibilidad de giro de los pisos en su propio plano o, lo que es lo mismo, el fenómeno de torsión global de la estructura. Aun así, dicha torsión se trata con total asiduidad de una manera simplificada (por utilizar las hipótesis de plantas rígidas y de deformación por axil nula en los pilares) con lo cual la masa del forjado se quedaría de nuevo concentrada en un único nudo, para el cual se consideran tres grados de libertad, con dos desplazamientos horizontales y un giro.

Una modelización más realista recogida en la normativa, y de aplicación general a cualquier tipo de estructura, sería la implementada mediante su discretización en barras y elementos finitos, interconectados entre sí en los nudos y considerando seis grados de libertad en cada uno de ellos. Debemos decir que la necesidad de emplear modelos espaciales está decidida básicamente por el fenómeno de la torsión. En general, tanto las cargas verticales, como las horizontales en mayor medida, pueden causar la torsión del edificio, siendo ésta la fuente de colapso de muchas estructuras bajo la acción de terremotos. Así, si un pórtico sufre un desplazamiento lateral por cargas verticales, arrastra lateralmente a otros que, bajo las simplificaciones de un análisis plano, aparentemente no se desplazan, lo cual ilustra la deficiencia del análisis bidimensional aún bajo carga vertical.

En esta modelización tridimensional de la estructura, la normativa contempla el análisis de la estructura mediante espectros de respuesta, obteniendo la respuesta máxima de la misma, o mediante métodos de integración, empleando acelerogramas representativos de acuerdo a la sismicidad de la ubicación de la estructura y determinando su respuesta a lo largo del tiempo.

4.2. La ecuación del movimiento y su resolución

Mientras que en el cálculo estático, debido a la lentitud del movimiento de los elementos estructurales, se establece la ecuación del equilibrio despreciando las fuerzas de inercia, no sucede lo mismo cuando dicho elemento se mueve con cierta velocidad y aceleración. De este modo, y de acuerdo a lo que ya vimos en la sección 2.5.3.5, la ecuación de movimiento de una estructura puede ser expresada mediante el equilibrio de las fuerzas que actúan sobre cada uno de los grados de libertad de la misma en cada instante de tiempo [7, 8, 37]:

$$f_I(t) + f_D(t) + f_S(t) = f(t) \tag{4.1}$$

siendo $f_I(t)$ la fuerza de inercia, $f_D(t)$ la fuerza de amortiguamiento, $f_S(t)$ la fuerza elástica y $f(t)$ la fuerza externa aplicada sobre la estructura. Estas fuerzas pueden expresarse en términos de las matrices de masa, amortiguamiento y rigidez de la estructura, así como de los desplazamientos, velocidades y aceleraciones en

función del tiempo en los nudos de la misma:

$$f_I(t) = Ma(t);\ f_D(t) = Cv(t);\ f_S(t) = Kd(t) \tag{4.2}$$

Sustituyendo dichas fuerzas por sus valores correspondientes, obtenemos la siguiente ecuación diferencial de segundo orden, conocida como *ecuación del movimiento*, o ecuación del equilibrio dinámico, la cual representa el equilibrio estático de fuerzas internas y externas para cualquier instante de tiempo t:

$$Ma(t) + Cv(t) + Kd(t) = f(t) \tag{4.3}$$

donde las condiciones iniciales en el instante $t = 0$ vienen dadas por $d(0) = d_0$ y $v(0) = v_0$, calculando a_0 al resolver dicha ecuación en el instante t=0. Los diferentes parámetros que aparecen en la ecuación son los siguientes:

- $K, M, C \in \mathbb{R}^{nxn}$ son las matrices de rigidez, masa y amortiguamiento de la estructura, respectivamente, siendo n es el número total de grados de libertad del sistema. En nuestro caso, al igual que en el caso estático, contemplaremos seis grados de libertad por nudo, siendo $n = 6N$, donde N representa el número total de nudos de la estructura. Tanto la matriz de rigidez como la de masa se generarán tras el ensamblaje de las matrices de rigidez y masa individuales de los elementos estructurales. Por su parte, la matriz de amortiguamiento se obtendrá mediante el método de Rayleigh, como combinación de las matrices de rigidez y masa de la estructura.
- $f(t) \in \mathbb{R}^{nxm}$ es la matriz de fuerzas externas generada a partir de las cargas aplicadas directamente sobre los nudos como de las fuerzas nodales equivalentes debidas a las cargas que actúan sobre las barras y los elementos finitos, siendo m el número total de hipótesis de carga básica diferente aplicadas, junto con cualquier combinación de todas ellas que se desee.
- $d(t), v(t), a(t) \in \mathbb{R}^{nxm}$ son matrices con los desplazamientos, velocidades y aceleraciones de los nudos de la estructura, para las distintas combinaciones de fuerzas externas aplicadas y para cada instante de tiempo.

En el caso de un análisis dinámico lineal, las matrices de coeficientes citadas

son constantes y no varían a lo largo del tiempo. En ese caso, la solución de la ecuación del movimiento puede llevarse a cabo mediante técnicas de propósito general de resolución de sistemas de ecuaciones diferenciales. Sin embargo, dichas técnicas no aprovechan las características particulares de las matrices del problema, motivo por el cual la resolución del tiempo conllevará un tiempo de ejecución excesivamente elevado, particularmente en el caso de problemas de gran dimensión.

No obstante, se han desarrollado todo un conjunto de técnicas eficientes orientadas a resolver dicha ecuación, en consideración a los requerimientos computacionales demandados por un análisis dinámico basado en barras y en elementos finitos. Ejemplos de dichas técnicas son los métodos de integración directa, el análisis modal o el análisis en el dominio de la frecuencia. Mientras que la primera de las técnicas es aplicable a problemas lineales y no lineales, las otras dos son únicamente válidas en problemas lineales. De manera genérica, cada una de estas técnicas consiste en lo siguiente:

- **Métodos de integración directa**: Gracias a la integración de la ecuación del movimiento paso a paso, bien sea mediante procedimientos numéricos implícitos o explícitos, estos métodos proporcionan la respuesta dinámica de la estructura en un conjunto de instantes de tiempos predeterminados. En todos los métodos de integración, las velocidades y aceleraciones para un instante de tiempo determinado se expresan en función únicamente del desplazamiento correspondiente al instante de tiempo en que se quiere hallar la solución y de los desplazamientos, velocidades y aceleraciones ya conocidos, correspondientes a instantes de tiempo anteriores.

  En el caso de los métodos implícitos, los desplazamientos en el instante de tiempo t_i se obtienen a partir de la ecuación diferencial del movimiento planteada para el mismo instante de tiempo, debiendo resolver en consecuencia un sistema de ecuaciones. En cambio, los métodos explícitos obtienen los desplazamientos en el instante de tiempo t_i a partir de la ecuación de movimiento expresada el instante anterior t_{i-1}, motivo por el cual es posible evitar la resolución del sistema de ecuaciones si las matrices de masa y amortiguamiento cumplen ciertas propiedades. Por otro lado, mientras que los métodos implícitos son incondicionalmente estables, independientemen-

te del paso de tiempo Δt elegido para resolver el problema, los métodos explícitos son condicionalmente estables, amplificando de manera artificial el comportamiento de la estructura si el paso de tiempo Δt empleado supera al llamado incremento crítico de integración.

- **Análisis modal**: El método de análisis modal está basado en una superposición apropiada de los modos propios de vibración del modelo estructural. Su punto de arranque viene dado por el análisis del problema de vibración libre, en el cual no están presentes ni la fuerza dinámica que actúa sobre la estructura ni el amortiguamiento de la misma. Esto conduce a la resolución de un problema de valores propios generalizado, donde deben calcularse las frecuencias naturales de vibración (valores propios) y los modos propios o formas modales de vibración (vectores propios) a fin de definir las características de vibración naturales de la estructura y evitar la temida resonancia.

  Gracias a métodos como el de Superposición Modal, es posible desacoplar un sistema inicial de n ecuaciones diferenciales en un conjunto de n ecuaciones independientes, pudiendo resolver cada una de ellas de manera aislada para cada instante, obteniendo el comportamiento de la estructura a medida que la fuerza dinámica va siendo aplicada, al igual que en los métodos anteriores. De manera alternativa, existen otros métodos como el Modal Espectral que nos permiten determinar, por medio de la respuesta espectral, el comportamiento estructural más desfavorable, reduciendo el gran medida el tiempo de simulación invertido en la resolución del problema y la cantidad de resultados generados.

- **Análisis en el dominio de la frecuencia**: A partir de la transformadas de Fourier de la fuerza dinámica externa aplicada sobre la estructura y de sus vectores de desplazamiento, velocidad y aceleración, la ecuación del movimiento se transforma en un sistema de ecuaciones algebraicas de coeficientes complejos. La respuesta del sistema en el dominio del tiempo se obtiene por medio de transformadas inversas de Fourier de la respuesta en frecuencias. Numéricamente, las transformadas directas e inversas de Fourier se calculan mediante la transformada discreta, siempre gracias al algoritmo que lleva el mismo nombre.

En esta tesis doctoral se han implementando los algoritmos secuenciales y paralelos asociados a la resolución de la ecuación del movimiento mediante métodos de integración directa y mediante las técnicas de análisis modal espectral y de superposición modal. Por ese motivo, todos estos métodos serán descritos con más detalle en secciones posteriores.

4.3. Generación de la matriz de rigidez de la estructura

Puesto que en nuestro caso abordaremos el análisis de estructuras tridimensionales compuestas por barras y elementos finitos con seis grados de libertad por nudo, la matriz K de rigidez de la estructura será exactamente la misma que la generada en el caso estático, con lo cual el modo de obtener la matriz de rigidez de cada elemento estructural y su proceso de ensamblaje será idéntico al ya citado en el capítulo anterior.

4.4. Generación de la matriz de masa de la estructura

El procedimiento más simple para obtener dicha matriz consiste en asumir que la masa de la estructura se concentra en los nudos de la misma, los cuales recogen las contribuciones másicas de las barras y de los elementos finitos que en ellos confluyen. Así, la composición de la matriz de masa de la estructura será simplemente la suma de las contribuciones de las masas concentradas en cada en los nudos. Este procedimiento da lugar a la matriz de *masas concentradas*, la cual consiste, tanto a nivel de un elemento individual como a nivel de la estructura conjunta, en una matriz diagonal.

Sin embargo, es posible obtener la aportación de la masa de cada elemento de la estructura siguiendo un procedimiento similar al de la generación de la matriz de rigidez, denominado método de las *masas consistentes*. Éste da lugar

a una matriz de masa simétrica, no diagonal, y que poseerá el mismo patrón de dispersidad que el de la matriz de rigidez.

La utilización de la matriz de masas concentradas conlleva numerosas ventajas, por el hecho de ser diagonal, tanto en su generación como en la resolución de los problemas numéricos en los que aparece (sistemas de ecuaciones lineales y problemas de valores propios generalizados), reduciendo considerablemente la memoria utilizada y los tiempos de cálculo frente a la aproximación de emplear la matriz de masas consistentes.

Sin embargo, el análisis dinámico que emplea matrices de masa consistente proporciona, a priori, resultados más precisos comparados con los obtenidos al usar matrices de masa concentrada, bajo una misma discretización de la estructura [38]. Por ese motivo, la aproximación implementada en esta tesis doctoral es la de las masas consistentes.

Conviene aclarar que la masa de cada elemento estructural no será únicamente la debida a su peso propio, sino además la que soporta de otros elementos no estructurales y la debida a otras acciones gravitatorias (permanentes y una fracción de las variables), como pueden ser las sobrecargas de uso. En consecuencia, ello supone que la masa de cada elemento se obtendrá a partir de una combinación de acciones gravitatorias aplicadas sobre la estructura, cada una de ellas multiplicada por un coeficiente apropiado, a la que denominaremos *combinación de masa*.

4.4.1. Matriz de masa de barras

Dada una modelización tridimensional de la estructura compuesta por barras, la matriz de masa concentrada M'^b de una barra b consiste en una matriz diagonal cuyos elementos quedan recogidos en el siguiente vector:

$$m^b = \left[\begin{array}{cccccccccccc} m_i & m_i & m_i & \frac{m_i I_0}{A} & 0 & 0 & m_j & m_j & m_j & \frac{m_j I_0}{A} & 0 & 0 \end{array} \right] \tag{4.4}$$

donde L es la longitud de la barra, I_o representa el momento polar de inercia, A es el área de la sección transversal de la barra y m_i y m_j son las masas concentradas en los extremos de la barra. En esta matriz diagonal, los coeficientes relativos a

las rotaciones de flexión se suponen iguales a cero.

Sin embargo, la matriz de masa consistente de una barra b en ejes locales es ésta otra, la cual recoge la combinación apropiada de los efectos axiles, de torsión y de flexión, requiriendo un esfuerzo computacional adicional [38]:

$$M'^{b} = \begin{bmatrix} M'_{ii} & M'_{ij} \\ M'_{ji} & M'_{jj} \end{bmatrix} =$$

$$= \frac{\bar{m}L}{420} \begin{bmatrix} 140 & 0 & 0 & 0 & 0 & 0 & 70 & 0 & 0 & 0 & 0 & 0 \\ 0 & 156 & 0 & 0 & 0 & 22L & 0 & 54 & 0 & 0 & 0 & -13L \\ 0 & 0 & 156 & 0 & -22L & 0 & 0 & 0 & 54 & 0 & 13L & 0 \\ 0 & 0 & 0 & \frac{140Io}{A} & 0 & 0 & 0 & 0 & 0 & \frac{70Io}{A} & 0 & 0 \\ 0 & 0 & -22L & 0 & 4L^2 & 0 & 0 & 0 & -13L & 0 & -3L^2 & 0 \\ 0 & 22L & 0 & 0 & 0 & 4L^2 & 0 & 13L & 0 & 0 & 0 & -3L^2 \\ 70 & 0 & 0 & 0 & 0 & 0 & 140 & 0 & 0 & 0 & 0 & 0 \\ 0 & 54 & 0 & 0 & 0 & 13L & 0 & 156 & 0 & 0 & 0 & -22L \\ 0 & 0 & 54 & 0 & -13L & 0 & 0 & 0 & 156 & 0 & 22L & 0 \\ 0 & 0 & 0 & \frac{70Io}{A} & 0 & 0 & 0 & 0 & 0 & \frac{140Io}{A} & 0 & 0 \\ 0 & 0 & 13L & 0 & -3L^2 & 0 & 0 & 0 & 22L & 0 & 4L^2 & 0 \\ 0 & -13L & 0 & 0 & 0 & -3L^2 & 0 & -22L & 0 & 0 & 0 & 4L^2 \end{bmatrix} \tag{4.5}$$

En dicha matriz, $\bar{m}$ representa la masa por unidad de longitud, la cual determinamos del siguiente modo:

$$\bar{m} = \frac{F_{iz} + F_{jz}}{gL} \tag{4.6}$$

siendo g el valor de la gravedad y F_{iz} y F_{jz} los esfuerzos cortantes de empotramiento en el eje Z en los extremos de la barra, expresados en ejes globales, aportados por todas las cargas que actúan sobre ella y que forman parte de la combinación de masa que mencionábamos con anterioridad.

En el caso de la matriz concentrada, m_i y m_j valdrían:

$$m_i = \frac{F_{iz}}{g},\ m_j = \frac{F_{jz}}{g} \tag{4.7}$$

Al estar dichos valores expresados en ejes globales, también lo estará la matriz concentrada.

De igual manera a como habíamos comentado para la matriz de rigidez, es necesario transformar dicha matriz de masa a ejes globales, antes de ensamblarla en la matriz de masa global de la estructura, recurriendo de nuevo para ello a la

matriz de rotación de la barra R^b. Así, la matriz de masa de una barra en ejes globales M^b es:

$$M^b = \begin{bmatrix} M_{ii} & M_{ij} \\ M_{ji} & M_{jj} \end{bmatrix} = \begin{bmatrix} R^b & 0 \\ 0 & R^b \end{bmatrix} \begin{bmatrix} M'_{ii} & M'_{ij} \\ M'_{ji} & M'_{jj} \end{bmatrix} \begin{bmatrix} R^{bT} & 0 \\ 0 & R^{bT} \end{bmatrix} =$$

$$= \begin{bmatrix} R_b M'_{ii} R^{bT} & R_b M'_{ij} R^{bT} \\ R_b M'_{ji} R^{bT} & R_b M'_{jj} R^{bT} \end{bmatrix} \tag{4.8}$$

Siguiendo el proceso de ensamblaje ya conocido, la aportación de la matriz de masa en ejes globales de una barra, delimitada por los nudos i, j, a la matriz de masa completa M de la estructura se obtendrá sumando cada una de sus cuatro submatrices en las posiciones correspondientes a los nudos i y j.

En caso de que la barra presente algún tipo de desconexión con los nudos a los que confluye, la expresión (4.5) no será válida, debiendo ser modificada oportunamente de acuerdo a lo que vimos en la expresión (3.7) en el caso de la matriz de rigidez, teniendo en cuenta las posibles relajaciones en sus extremos y el valor de las variables recogidas en las expresiones (3.6).

Finalmente, en el caso de que la barra fuera excéntrica, obtendríamos las matrices H_{iA} y H_{jB} de acuerdo a la expresión (3.17), a fin de obtener la matriz de masa de la barra del siguiente modo:

$$M^b = \begin{bmatrix} H_{iA} & 0 \\ 0 & H_{jB} \end{bmatrix} \begin{bmatrix} R^b & 0 \\ 0 & R^b \end{bmatrix} \begin{bmatrix} M'_{ii} & M'_{ij} \\ M'_{ji} & M'_{jj} \end{bmatrix} \begin{bmatrix} R^{bT} & 0 \\ 0 & R^{bT} \end{bmatrix} \begin{bmatrix} H^T_{iA} & 0 \\ 0 & H^T_{jB} \end{bmatrix} =$$

$$= \begin{bmatrix} H_{iA} & 0 \\ 0 & H_{jB} \end{bmatrix} \begin{bmatrix} M_{ii} & M_{ij} \\ M_{ji} & M_{jj} \end{bmatrix} \begin{bmatrix} H^T_{iA} & 0 \\ 0 & H^T_{jB} \end{bmatrix} = \begin{bmatrix} H_{iA} M_{ii} H^T_{iA} & H_{iA} M_{ij} H^T_{jB} \\ H_{jB} M_{ji} H^T_{iA} & H_{jB} M_{jj} H^T_{jB} \end{bmatrix} \tag{4.9}$$

4.4.2. Matriz de masa de elementos triangulares

A la hora de trabajar con los elementos superficiales triangulares, debemos diferenciar de nuevo entre su comportamiento a flexión y su comportamiento a

membrana, combinando ambos posteriormente para dar lugar a la matriz de masa del elemento triangular de lámina.

Como ya hemos comentado, la masa del elemento la obtendremos a partir de la combinación de cargas gravitatorias aplicadas sobre el mismo. Para ello, sumaremos las componentes q_z de todas las cargas superficiales $q_s = [q_x, q_y, q_z]^T$ que actúen sobre el elemento, expresadas en ejes globales como fuerza por unidad de superficie, dividiendo finalmente dicha suma por el valor de la gravedad. El resultado obtenido será equivalente al producto de la densidad del material ρ por el espesor h del elemento que aparece en las siguientes secciones.

4.4.2.1. Formulación del elemento triangular a flexión

Como ya vimos en la expresión (2.28), la matriz de masa de un elemento finito e compuesto por un material de densidad ρ constante se calcula del siguiente modo, siendo N la matriz que agrupa las funciones de forma:

$$M^e = \rho \int_{V^e} \mathrm{N^T N}\, d\nu \tag{4.10}$$

Si aplicamos dicha expresión a un elemento bidimensional triangular de espesor h, la integral anterior es equivalente a esta otra, siendo A el área del elemento:

$$M^e = \rho h \int_{A^e} \mathrm{N^T N}\, dA \tag{4.11}$$

Según [28], la resolución numérica de dicha integral conduce a la siguiente matriz de masa en ejes locales, la cual considera únicamente aquellos grados de libertad

del elemento relacionados con su comportamiento a flexión:

$$M_p'^e = \frac{\rho h A}{12} \begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 \end{bmatrix} \tag{4.12}$$

4.4.2.2. Formulación del elemento triangular a membrana

De acuerdo a lo descrito por Felippa en [34], la matriz de masa consistente en ejes locales de un triángulo e con comportamiento de membrana, compuesto por un material de densidad ρ, se obtiene del siguiente modo:

$$M_m'^e = T_{CR}^T M_C T_{CR} \tag{4.13}$$

siendo M_C la siguiente matriz, la cual depende del espesor h del elemento, que se entiende uniforme, y de su área A:

$$M_C = \frac{\rho h A}{180} \begin{bmatrix} 6 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & -4 & 0 & 0 & 0 \\ 0 & 6 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & -4 & 0 & 0 \\ -1 & 0 & 6 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -4 & 0 \\ 0 & -1 & 0 & 6 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -4 \\ -1 & 0 & -1 & 0 & 6 & 0 & -4 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 6 & 0 & -4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -4 & 0 & 32 & 0 & 16 & 0 & 16 & 0 \\ 0 & 0 & 0 & 0 & 0 & -4 & 0 & 32 & 0 & 16 & 0 & 16 \\ -4 & 0 & 0 & 0 & 0 & 0 & 16 & 0 & 32 & 0 & 16 & 0 \\ 0 & -4 & 0 & 0 & 0 & 0 & 0 & 16 & 0 & 32 & 0 & 16 \\ 0 & 0 & -4 & 0 & 0 & 0 & 16 & 0 & 16 & 0 & 32 & 0 \\ 0 & 0 & 0 & -4 & 0 & 0 & 0 & 16 & 0 & 16 & 0 & 32 \end{bmatrix} \tag{4.14}$$

y donde T_{CR} estará formada por los siguientes elementos, siendo $\frac{3}{2}$ el valor recomendado para α_b:

$$T_{CR} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{8}\alpha_b y_{12} & 0 & 0 & \frac{1}{8}\alpha_b y_{21} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{8}\alpha_b x_{21} & 0 & 0 & \frac{1}{8}\alpha_b x_{12} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{8}\alpha_b y_{23} & \frac{1}{2} & 0 & \frac{1}{8}\alpha_b y_{32} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{8}\alpha_b x_{32} & 0 & \frac{1}{2} & \frac{1}{8}\alpha_b x_{23} \\ \frac{1}{2} & 0 & \frac{1}{8}\alpha_b y_{13} & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{8}\alpha_b y_{31} \\ 0 & \frac{1}{2} & \frac{1}{8}\alpha_b x_{31} & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{8}\alpha_b x_{13} \end{bmatrix} \tag{4.15}$$

4.4.2.3. Formulación del elemento de lámina

A fin de obtener una matriz de masa relativa al comportamiento de lámina del triángulo, combinaremos las matrices de masa a flexión y a membrana. El modo de reagrupar ambas matrices en una sola será idéntico a lo que ya describimos en el caso de la matriz de rigidez.

Si el triángulo en cuestión tiene a los nudos i, j y k como vértices, sus matrices de masa a flexión y a membrana, de tamaño 9x9, serán las siguientes, compuestas por un subconjunto de submatrices de tamaño 3x3:

$$M_p'^e = \begin{bmatrix} M_{ii}'^p & M_{ij}'^p & M_{ik}'^p \\ M_{ji}'^p & M_{jj}'^p & M_{jk}'^p \\ M_{ki}'^p & M_{kj}'^p & M_{kk}'^p \end{bmatrix}, \quad M_m'^e = \begin{bmatrix} M_{ii}'^m & M_{ij}'^m & M_{ik}'^m \\ M_{ji}'^m & M_{jj}'^m & M_{jk}'^m \\ M_{ki}'^m & M_{kj}'^m & M_{kk}'^m \end{bmatrix} \tag{4.16}$$

Si la matriz de masa de la lámina combina ambas matrices, su tamaño será de 18 filas y 18 columnas, formada por submatrices de tamaño 6x6:

$$M'^e = \begin{bmatrix} M_{ii}' & M_{ij}' & M_{ik}' \\ M_{ji}' & M_{jj}' & M_{jk}' \\ M_{ki}' & M_{kj}' & M_{kk}' \end{bmatrix} \tag{4.17}$$

Como ya comentamos, cada submatriz de la matriz de lámina, como por ejemplo la M'_{ij}, se obtendrá mediante la suma de las submatrices con los mismos subíndices de las matrices a flexión y a membrana, tras ubicar correctamente los elementos según los grados de libertad que cada una considera:

$$M'_{ij} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & M'^p & M'^p & M'^p & 0 \\ 0 & 0 & M'^p & M'^p & M'^p & 0 \\ 0 & 0 & M'^p & M'^p & M'^p & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} M'^m & M'^m & 0 & 0 & 0 & M'^m \\ M'^m & M'^m & 0 & 0 & 0 & M'^m \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ M'^m & M'^m & 0 & 0 & 0 & M'^m \end{bmatrix} =$$

$$= \begin{bmatrix} M'^m & M'^m & 0 & 0 & 0 & M'^m \\ M'^m & M'^m & 0 & 0 & 0 & M'^m \\ 0 & 0 & M'^p & M'^p & M'^p & 0 \\ 0 & 0 & M'^p & M'^p & M'^p & 0 \\ 0 & 0 & M'^p & M'^p & M'^p & 0 \\ M'^m & M'^m & 0 & 0 & 0 & M'^m \end{bmatrix} \tag{4.18}$$

Finalmente, debemos transformar a ejes globales la matriz obtenida empleando la matriz de rotación R^e del elemento, para ensamblarla después en la matriz de masa global de la estructura, en las posiciones correspondientes a sus nodos:

$$M^e = \begin{bmatrix} R^e & 0 & 0 \\ 0 & R^e & 0 \\ 0 & 0 & R^e \end{bmatrix} \begin{bmatrix} M'_{ii} & M'_{ij} & M'_{ik} \\ M'_{ji} & M'_{jj} & M'_{jk} \\ M'_{ki} & M'_{kj} & M'_{kk} \end{bmatrix} \begin{bmatrix} R^{eT} & 0 & 0 \\ 0 & R^{eT} & 0 \\ 0 & 0 & R^{eT} \end{bmatrix} =$$

$$= \begin{bmatrix} R^e M'_{ii} R^{eT} & R^e M'_{ij} R^{eT} & R^e M'_{ik} R^{eT} \\ R^e M'_{ji} R^{eT} & R^e M'_{jj} R^{eT} & R^e M'_{jk} R^{eT} \\ R^e M'_{ki} R^{eT} & R^e M'_{kj} R^{eT} & R^e M'_{kk} R^{eT} \end{bmatrix} = \begin{bmatrix} M_{ii} & M_{ij} & M_{ik} \\ M_{ji} & M_{jj} & M_{jk} \\ M_{ki} & M_{kj} & M_{kk} \end{bmatrix} \tag{4.19}$$

4.4.3. Matriz de masa de elementos cuadriláteros

Cuando un triángulo provenga de la división de un cuadrilátero en cuatro triángulos, su matriz de masa M^e generada de acuerdo a la sección anterior se

deberá multiplicar por $\frac{1}{2}$, con el objetivo de promediar las dos particiones superpuestas del rectángulo en dos triángulos. Previamente, la suma de las componentes q_z de las diferentes cargas aplicadas sobre el cuadrilátero, considerada para calcular su masa, también se habrá multiplicado por $\frac{1}{2}$.

4.4.4. Matriz de masa de elementos tetraédricos

Supongamos un elemento tetraédrico e compuesto por un material de densidad ρ homogénea en todo el elemento. Si resolvemos numéricamente la integral de la expresión (2.28), la matriz de masa de dicho elemento, en ejes globales, la calcularemos del siguiente modo [28], donde V equivale a su volumen:

$$M^e = \frac{\rho V}{20} \begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 \end{bmatrix} \tag{4.20}$$

Si sumamos las componentes q_z de las cargas volumétricas $q_v = [q_x, q_y, q_z]^T$ aplicadas sobre el elemento pertenecientes a la combinación de masa (expresadas en ejes globales y como fuerzas por unidad de volumen) y dividimos dicha suma por la gravedad, ocurrirá que el resultado obtenido sustituirá a la densidad ρ que aparece en la matriz anterior.

4.4.5. Matriz de masa de elementos hexaédricos

La integración de la expresión (2.28), a fin de obtener la matriz de masa de un hexaedro e compuesto por un material de densidad ρ, equivaldría a esta otra,

si las variables independientes del problema pasan ahora a ser las coordenadas naturales del elemento:

$$M^e = \rho \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \mathrm{N}\left(\xi,\eta,\zeta\right)^T \mathrm{N}\left(\xi,\eta,\zeta\right) \left|J\left(\xi,\eta,\zeta\right)\right| d\xi d\eta d\zeta \tag{4.21}$$

En dicha integral aparece la matriz N compuesta por las funciones de forma, documentada en la expresión (3.80), y el determinante de la matriz Jacobiana recogida en (3.83). Resolviendo la integral por la técnica de Gauss-Legendre como vimos en el caso de la matriz de rigidez, y empleando para ello los 8 puntos de integración de la tabla 3.2, diríamos que la matriz de masa en ejes globales del hexaedro es:

$$M^e = \rho \sum_{p=1}^{8} \mathrm{N}(\xi,\eta,\zeta)^T \mathrm{N}\left(\xi,\eta,\zeta\right) \left|J\left(\xi,\eta,\zeta\right)\right| \tag{4.22}$$

Al igual que en el caso del tetraedro, en esta expresión sustituiríamos la densidad del material ρ por el cociente entre la suma de la componente q_z de todas las cargas volumétricas que están actuando sobre el elemento y forman parte de la combinación de masa, expresadas en ejes globales, y la gravedad.

4.4.6. Aportación a la matriz de masa de las cargas aplicadas sobre los nudos

Supongamos un nudo que presenta una carga que está actuando directamente sobre él, la cual forma parte del conjunto de cargas de la combinación de masas. A efectos de generar la matriz de masa global de la estructura, diremos que la componente Z en ejes globales de dicha carga se sumará, en la matriz de masa global de la estructura, a los 3 elementos de la diagonal correspondientes a los grados longitudinales del nudo.

4.5. Generación de la matriz de amortiguamiento

Las fuerzas de amortiguamiento en las estructuras se deben a diversas razones, tales como un rozamiento entre superficies de deslizamiento, por vibraciones de la estructura situada en un medio exterior, o por fricción interna del material propio de la estructura.

En el caso de las estructuras de edificación es necesario aplicar diferentes hipótesis simplificadoras a fin de obtener una representación numérica razonable de las propiedades de amortiguamiento de la estructura. Una de estas hipótesis consiste en suponer que existe un mecanismo de pérdida de energía homogéneo en toda la estructura. En tal caso, puede obtenerse una matriz de amortiguamiento que sea ortogonal con respecto a la matriz modal, compuesta por los modos de vibración de la estructura. Esta condición permite definir la matriz de amortiguamiento como una combinación lineal de las matrices de masa y rigidez, mediante el llamado *amortiguamiento de Rayleigh*, donde α_0 y α_1 son las constantes de proporcionalidad:

$$C = \alpha_0 M + \alpha_1 K \tag{4.23}$$

Conocidas dos frecuencias naturales de vibración de la estructura ω_m y ω_n y sus dos factores de amortiguamiento crítico asociados ξ_m y ξ_n, las constantes de proporcionalidad pueden determinarse resolviendo el siguiente sistema de ecuaciones lineales:

$$\frac{1}{2}\begin{bmatrix} 1/\omega_m & \omega_m \\ 1/\omega_n & \omega_n \end{bmatrix}\begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} \xi_m \\ \xi_n \end{bmatrix} \tag{4.24}$$

Una manera alternativa de obtener la matriz de amortiguamiento es mediante el llamado amortiguamiento de Rayleigh extendido o generalizado, en el cual la matriz C se obtiene de la forma:

$$C = M \sum_{q=0}^{n-1} \alpha_q (M^{-1}K)^q \tag{4.25}$$

Puede observarse que el amortiguamiento de Rayleigh es un caso particular de este otro para n=2. Los valores de α_q se pueden determinar resolviendo el sistema de ecuaciones lineales que resulta de aplicar la siguiente expresión a aquellas n frecuencias naturales conocidas y sus correspondientes factores de amortiguamiento crítico:

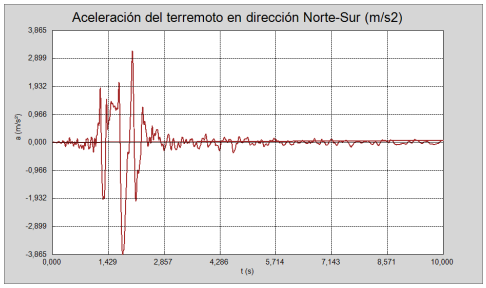
$$\xi_i = \frac{1}{2\omega_i} \sum_{q=0}^{n-1} \alpha_q \omega_i^{2q}; \;\; i = 1, 2, \ldots, n \tag{4.26}$$

La aproximación seguida en esta tesis doctoral para generar la matriz de amortiguamiento se corresponde con el amortiguamiento de Rayleigh.

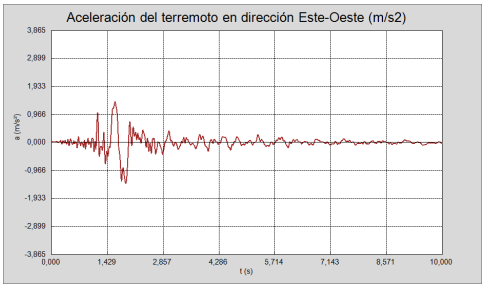
4.6. Generación del vector de cargas dinámicas

Para cada instante de tiempo, la matriz $f(t) \in \mathbb{R}^{nxm}$ estará compuesta por las m hipótesis de cargas dinámicas aplicadas sobre la estructura, o cualquier combinación de las mismas, todas ellas en ejes globales. A su vez, cada hipótesis estará compuesta por la suma de aquellas cargas aplicadas directamente sobre los nudos más las fuerzas nodales equivalentes, cuyos valores se obtienen de las cargas aplicadas sobre las barras o sobre los elementos finitos.

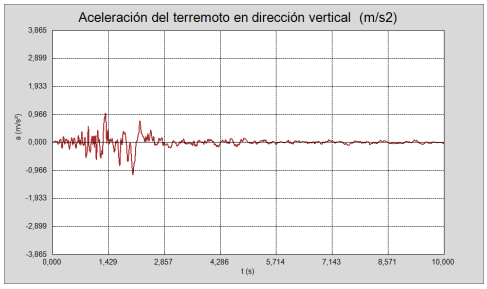
Hay que decir que las cargas dinámicas podrán estar expresadas como fuerzas o como aceleraciones (ver figura 4.1). En este segundo caso, habrá que premultiplicarlas por la matriz de masa, cambiada de signo. A modo de ejemplo, supongamos una carga sísmica que está actuando sobre la estructura modelizada a modo de acelerograma, el cual habitualmente incluye valores de las aceleraciones del terreno para cada instante de tiempo según dos componentes longitudinales (Norte-Sur y Este-Oeste) y una componente vertical. Si las aceleraciones horizontales $a_x(t)$ y $a_y(t)$ coinciden con los ejes globales horizontales de la estructura y la aceleración vertical $a_z(t)$ coincide con el eje vertical de la misma, el vector de fuerzas $f(t)$ debido a la actuación del terremoto podrá expresarse del siguiente modo, el cual poseerá elementos no nulos en los grados de libertad de traslación de cada nudo



(a) Componente Norte-Sur.



(b) Componente Este-Oeste.



(c) Componente vertical.

Figura 4.1: Acelerograma del terremoto ocurrido en Lorca en el año 2011.

de la estructura:

$$f(t) = -M(J_x a_x(t) + J_y a_y(t) + J_z a_z(t)) \tag{4.27}$$

donde J_x, J_y o $J_z \in \mathbb{R}^{nx1}$ son de la forma:

$$\begin{aligned} J_x &= [1\,0\,0\,0\,0\,0 \ldots 1\,0\,0\,0\,0\,0 \ldots 1\,0\,0\,0\,0\,0]^T \\ J_y &= [0\,1\,0\,0\,0\,0 \ldots 0\,1\,0\,0\,0\,0 \ldots 0\,1\,0\,0\,0\,0]^T \\ J_z &= [0\,0\,1\,0\,0\,0 \ldots 0\,0\,1\,0\,0\,0 \ldots 0\,0\,1\,0\,0\,0]^T \end{aligned} \tag{4.28}$$

En caso de que la dirección del sismo, para cualquiera de sus aceleraciones con las que se define, no coincida con los ejes principales de la estructura, los vectores J_x, J_y o J_z estarán compuestos por cosenos directores, a fin de expresar la carga en los ejes principales. Además de las cargas sísmicas expresadas en forma de acelerograma, otras cargas que varíen con el tiempo incorporadas en este trabajo son las cargas tipo (en forma triangular, rectangular, trapezoidal, senoidal y exponencial) y las cargas dinámicas genéricas, todas ellas expresadas como fuerza o como aceleración y aplicadas siempre sobre los nudos de la estructura.

En las cargas tipo, se deberá indicar los nudos, y a su vez los grados de libertad, sobre los que se aplican, junto con los parámetros de definición de las mismas. En las cargas dinámicas genéricas, además de los nudos y sus grados de libertad a los que se asignan, se detallarán todos aquellos instantes de tiempo de actuación y el valor de la carga para cada uno de ellos. En este tipo de acciones, el vector $f(t)$ será del siguiente modo:

$$f(t) = -MJ_a(t) + f_f(t) \tag{4.29}$$

Si las cargas son de fuerza, su intensidad se sumará en el vector $f_f(t)$ de fuerzas, siempre dependiendo del instante de tiempo considerado, en las posiciones correspondientes a los nudos y a los grados de libertad sobre los que actúa. Si por el contrario la carga es de aceleración, sumaremos su valor en las posiciones apropiadas de un vector $J_a(t)$ de aceleraciones, el cual premultiplicaremos por la matriz de masa y le cambiaremos el signo, antes de sumárselo al vector $f_f(t)$ para componer, finalmente, el vector $f(t)$ con la fuerza dinámica aplicada en el instante de tiempo actual.

4.7. Análisis dinámico lineal mediante métodos de integración directa

4.7.1. Generalidades

Los métodos de integración directa en el tiempo integran la ecuación del movimiento (4.3) usando procedimientos numéricos paso a paso, obteniendo así la historia de la respuesta dinámica de la estructura en una serie de instantes de tiempo determinados a priori, de la forma $t = 0$, $t = \Delta t$, $t = 2\Delta t$, $t = 3\Delta t$, ..., $t = T$.

Son muy numerosos los diferentes métodos de integración directa en el tiempo desarrollados [39, 40, 41, 42, 43], bien sean explícitos, implícitos, mixtos explícitos-implícitos, variables, adaptativos, etc. La variación de los desplazamientos, velocidades y aceleraciones entre un instante de tiempo y el siguiente son siempre

dependientes del método de integración empleado, lo cual determina la precisión, la estabilidad y el coste computacional del algoritmo elegido.

Tradicionalmente, los métodos desarrollados asumen que los cambios en los desplazamientos, velocidades y aceleraciones de un paso al siguiente están linealmente correlacionados mediante ecuaciones en diferencias. Con estas premisas, la mayor parte de los algoritmos intentan satisfacer la ecuación del equilibrio dinámico en un instante de tiempo determinado.

Los llamados métodos *implícitos* permiten expresar los desplazamientos en el instante de tiempo actual usando las condiciones de equilibrio en dicho instante de tiempo. Por ello, estos métodos necesitan resolver un sistema de ecuaciones para cada paso de tiempo, a fin de obtener los desplazamientos en los nudos de la estructura. Sin embargo, las matrices K, M y C permanecerán invariables en caso de implementar un análisis lineal. Esto supone la ventaja de que, en caso de aplicar un método directo en la resolución del sistema de ecuaciones lineales, la llamada matriz de rigidez efectiva o matriz de coeficientes del sistema, obtenida a partir de las tres matrices iniciales del problema, podrá ser factorizada al comienzo, resolviendo únicamente sistemas de ecuaciones triangulares para cada uno de los instantes de tiempo. Una vez calculados los desplazamientos en los nudos, las velocidades y aceleraciones en dichos puntos pueden ser obtenidas mediante sencillas operaciones vectoriales.

Como clara ventaja de estos métodos frente a los explícitos, que describiremos a continuación, ocurre que la mayoría de ellos suelen ser estables incondicionalmente, con lo cual el tamaño del paso de tiempo a emplear únicamente depende de la precisión requerida en los resultados, sin necesidad de ser tan pequeño como en los métodos explícitos. Ejemplos de métodos implícitos clásicos de integración directa son los métodos de Newmark, Wilson-θ o Generalizado-α.

Los métodos *explícitos* calculan los desplazamientos en el instante de tiempo actual usando las condiciones de equilibrio en el instante de tiempo anterior. Estos métodos surgen con el objetivo de no tener que resolver un sistema de ecuaciones para cada paso de tiempo, siendo fáciles de programar y menos costosos, a priori, en tiempo de ejecución. Sin embargo, estos métodos tienden a ser condicionalmente estables, motivo por el cual la estabilidad de la solución obtenida depende

exclusivamente del tamaño del paso de tiempo escogido, el cual muchas veces puede ser excesivamente pequeño. Esta limitación en el tamaño del paso de tiempo a emplear los convierte a menudo en inadecuados a la hora de resolver problemas de dinámica estructural. Ejemplos de de métodos explícitos de integración directa son el método de las Diferencias Centradas o el de Runge-Kutta.

Existen también los llamados algoritmos *explícitos-implícitos*, los cuales intentan combinar las principales ventajas de ambas metodologías. En estos métodos híbridos, el sistema a estudiar se divide en diferentes subdominios, en los cuales se aplica o bien un método implícito o bien uno explícito, empleando para ello diferentes pasos de tiempo.

Aunque no existe actualmente un consenso universal que proporcione una lista ordenada que muestre claramente las ventajas de la aplicación de un método con respecto a sus competidores, Hughes [40] propuso un siguiente conjunto de atributos a satisfacer por parte de cada método con vistas a ser competitivo:

- Estabilidad incondicional al aplicarse a problemas lineales. Los métodos condicionalmente estables son aquellos que requieren emplear un paso de tiempo inferior a una cierta fracción del periodo de vibración más pequeño de la estructura. Esta restricción, debida a los requerimientos de estabilidad, los convierte en claramente ineficientes, ya que el paso de tiempo llega a ser mucho más pequeño que el necesario para proporcionar unos resultados con una precisión aceptable. Son en consecuencia necesarios métodos cuya estabilidad no dependa del paso de tiempo empleado, sino que éste únicamente determine la precisión de los resultados.

- Resolución, a lo sumo, de un sistema de ecuaciones lineales por cada paso de tiempo, ya que dicha resolución supondrá el coste computacional más elevado del proceso de simulación.

- Uso de métodos de integración monopaso en los cuales los resultados en un instante de tiempo no dependan más allá de los resultados obtenidos en el instante anterior.

- Precisión de segundo orden en los resultados, es decir, que los errores cometidos sean del orden Δt^2, siendo Δt el incremento de tiempo empleado

en los algoritmos.

- Disipación algorítmica alta, o controlable, de los modos de vibración más altos. Es bien conocido que las frecuencias de vibración más altas presentes en la ecuación del equilibrio dinámico de una estructura son debidas al proceso de discretización espacial de la misma y no se corresponden con la propia estructura. Es por tanto recomendable eliminar la participación de dichos modos en el proceso de resolución.
- Capacidad autónoma para ponerse en marcha sin necesitar de realizar un esfuerzo computacional adicional ni de requerir un almacenamiento en memoria añadido.

La disipación de un algoritmo puede ser medida por el *radio espectral* o, lo que es lo mismo, por la frecuencia de vibración más grande de la llamada *matriz de amplificación numérica*. Wood [39] sugirió que la curva del radio espectral frente a $\Delta t/T$, donde T es el periodo natural del sistema no amortiguado y Δt es el paso de tiempo empleado, debería estar cercana a 1 durante el mayor tiempo posible, para luego decrecer hasta alcanzar un valor comprendido entre 0.5 y 0.8 cuando $\Delta t/T$ tiene hacia infinito. El radio espectral alcanzado cuando $\Delta t/T$ tiende a infinito es el llamado radio espectral último, denotado por μ. En un algoritmo incondicionalmente estable, el radio espectral no excederá de 1 para cualquier valor de $\Delta t/T$. Aquellos métodos en los cuales $\mu = 0$ se denominan, de acuerdo a la terminología inglesa, *asymptotically annihilating*, por la cual la aportación de las frecuencias altas se elimina completamente tras un único paso de tiempo.

Por otro lado, según Dahlquist [44], los métodos multipaso implícitos sólo podrán alcanzar, a lo sumo, una precisión de segundo orden si son incondicionalmente estables. En otras palabras, los algoritmos multipaso con una precisión de tercer o mayor orden serán en consecuencia condicionalmente estables.

A modo de resumen, los pasos a seguir para calcular la historia de la respuesta completa de una estructura a lo largo del tiempo mediante un método cualquiera de integración directa son los siguientes:

1. Generar las matrices de rigidez K, masa M y amortiguamiento C de la estructura.

2. Calcular los desplazamientos, velocidades y aceleraciones iniciales.

3. Determinar la matriz de rigidez efectiva.

4. Para cada paso de tiempo t=Δt, $2\Delta t$, ..., T:

   a) Evaluar el vector de cargas dinámicas y de cargas dinámicas efectivas.

   b) Calcular los desplazamientos *d(t)*, velocidades *v(t)* y aceleraciones *a(t)* en los nudos.

   c) Determinar las solicitaciones en los extremos de las barras.

   d) Obtener las reacciones en los apoyos.

   e) Calcular los esfuerzos y las deformaciones en los puntos intermedios de las barras.

   f) Proporcionar las deformaciones, las tensiones y los esfuerzos en los nudos de los elementos finitos.

   g) Volver al punto 4, con $t = t + \Delta t$.

4.7.2. Estado del arte

Son múltiples los métodos de integración que pueden encontrarse en la literatura. Si nos centramos en los métodos implícitos, el método de Houbolt [45], desarrollado en 1950, fue uno de los primeros métodos empleados en calcular la respuesta estructural de un avión sometido a cargas dinámicas. Este método aproxima la velocidad y la aceleración a partir de diferencias en el desplazamiento en instantes diferentes, estableciendo una relación de recurrencia usada para obtener la respuesta paso a paso de la estructura. Desde el punto de vista de la estabilidad y la precisión, este método multipaso es incondicionalmente estable, ofrece una precisión de segundo orden y es *asymptotically annihilating*. Esto lo convierte en un método no apropiado para problemas dinámicos donde aparezcan altas frecuencias. En 1994, este método fue convertido a un método monopaso, conservando sus mismas características, gracias a Chung y a Hulbert [46].

En 1959, Newmark [9] introdujo un método implícito, bajo el mismo nombre, para resolver problemas dinámicos estructurales. Dicho algoritmo asume que

la aceleración media es constante entre dos instantes de tiempo consecutivos. Después, Belytschko y Hughes [47] desarrollaron la familia de métodos llamada Newmark-β, los cuales son implícitos o explícitos, dependiendo del valor de dos parámetros que controlan la estabilidad y la precisión de los algoritmos. Como esquema implícito, el método de Newmark es incondicionalmente estable dependiendo del valor de los parámetros que lo configuran y ofrece resultados con una precisión de segundo orden, aunque dicha precisión y la disipación de altas frecuencias no pueden ocurrir conjuntamente.

En 1968 Wilson desarrolló el llamado método de Wilson-θ [10], el cual es esencialmente una extensión del método de Newmark, asumiendo que la aceleración es lineal entre el instante t y el instante $t + \theta\Delta t$, con $\theta \geq 1$. Posteriormente, Bathe y Wilson [48] evaluaron y compararon entre sí la estabilidad y la precisión de los métodos de Houbolt, Newmark, y Wilson-θ y describieron la relación entre los métodos de integración directa y las técnicas de análisis modal. Park [49], en 1975, diseño un método multipaso, con precisión de segundo orden e incondicionalmente estable. El método es también *asymptotically annihilating* y ofrece unas propiedades disipativas en las bajas frecuencias mejores que el de Houbolt.

Zienkiewicz [50] desarrolló en 1977 una clase diferente de relaciones de recurrencia para expresar la ecuación del movimiento usando la Aproximación Residual Ponderada. Posteriormente Zienkiewicz, Wood, Hine y Taylor [51] propusieron en 1984 una familia de algoritmos, conocida como el método SSpj, relacionados con la metodología anterior. Ajustando apropiadamente los parámetros del método resulta que muchos de los algoritmos descritos hasta el momento, como Newmark o Houbolt, fueron identificados como casos particulares del mismo. Sin embargo, Wood [52] mostró que para ciertos valores de estos parámetros ocurría que el método SSpj no coincidía con ninguno de los métodos conocidos. Este método puede dar lugar a otros con una precisión superior a la de segundo orden introduciendo polinomios de interpolación de mayor orden, aunque como resultado los nuevos algoritmos son a menudo condicionalmente estables y necesitan recurrir a técnicas auxiliares para comenzar.

Bazzi y de Anderheggen desarrollaron en 1982 el método ρ [53], incondicionalmente estable pero con una precisión de segundo orden, a cambio de no introducir amortiguamiento numérico (si presenta amortiguamiento, la precisión es de

primer orden). Katona y Zienkiewicz [54] mostraron, en 1985, que las aproximaciones de diferencias finitas empleadas en la derivación del método de Newmark eran, en realidad, series truncadas de Taylor aplicadas al desplazamiento y a la velocidad. En consecuencia, dichos autores demostraron que se podían emplear aproximaciones basadas en series de Taylor de mayor orden, dando lugar al llamado método β_m. Este método presenta m términos en la aproximación de la aceleración, $m+1$ en la de la velocidad y $m+2$ en los desplazamientos, siendo $m \geq 2$. Puede observarse que el método β_2 equivale al método de Newmark.

Los métodos empleados hasta el momento tenían la peculiaridad de satisfacer la condición del equilibrio dinámico en un instante de tiempo determinado, a fin de obtener los desplazamientos, velocidades y aceleraciones en dicho instante. Sin embargo, los métodos de colocación, tales como los analizados por Hilber y Hughes [55] en 1978, evitan dicha restricción. Estos métodos combinan los métodos de Newmark y de Wilson-θ, pudiendo ser ajustados para reducirse a uno o a otro, dando lugar a algoritmos incondicionalmente estables, con precisión de segundo orden y con disipación numérica.

Con el objetivo de controlar el amortiguamiento algorítmico de las frecuencias altas del sistema, se desarrollaron extensiones y modificaciones intentando mejorar, como punto de partida, los métodos de la familia de Newmark. A modo de ejemplo, podemos citar los métodos HHT-α, WBZ-α, SSpj, θ_1, η, Generalizado-α, SDIRK o Generalizado, que a continuación describimos. El método HHT-α [56], desarrollado por Hilber, Hughes y Taylor en 1977, es incondicionalmente estable y ofrece precisión de segundo orden si los parámetros del método se ajustan apropiadamente. El método WBZ-α [57], diseñado en 1980 por Wood, Bossak y Zienkiewicz, presenta características muy similares al anterior. Ambos métodos fueron comparados por Adams y Wood en 1982. El método θ_1 de Hoff and Pahl, descrito en 1988 [58, 59, 60], y el método η [61], descrito por Valliappan y Ang en 1989, son al igual que los anteriores incondicionalmente estables y presentan una precisión de segundo orden, amortiguando en mayor cantidad las altas frecuencias y en menor cantidad las bajas frecuencias.

El método Generalizado-α [62], creado por Chung y Hulbert en 1993 a partir de los métodos HHT-α y WBZ-α, presenta un número de parámetros superior a los anteriores, los cuales pueden ser ajustados para obtener las propiedades

algorítmicas disipativas deseables. Esto lo convierte en un método que presenta unas características de disipación numérica superiores al resto. Además el error en los resultados obtenidos con este método es inferior al proporcionado por el resto de métodos con características disipativas. En 2003, Chung, Cho y Choi [63] diseñaron un estimador del error *a priori* del método Generalizado-α, donde el error producido se calcula a partir de la información obtenida en dos pasos consecutivos. Esto da lugar a un paso de tiempo con un tamaño variable y adaptativo, lo cual repercute en un método de integración con unos tiempos de computación más reducidos. En caso de que la carga aplicada cambie súbitamente, el estimador del error a priori puede ser grande. En ese caso conviene recurrir a un esquema híbrido, en el cual el error a priori se usa en la mayoría de los pasos de tiempo, pero en aquellos en los cuales supera un valor umbral determinado convendrá emplear el error a posteriori.

Owren y Simonsen presentaron en 1995 la aplicación de los métodos SDIRK (Singly-Diagonally-Implicit Runge-Kutta) para resolver la ecuación del movimiento en dinámica estructural [64]. Los métodos desarrollados son incondicionalmente estables, *asymptotically annihilating* y presentan un parámetro que controla la disipación numérica de las altas frecuencias. El método permite escoger el número de etapas empleadas, de manera que el método de cuatro etapas proporciona una precisión de tercer orden. El inconveniente principal de estos métodos es que necesitan resolver dos sistemas de ecuaciones, uno de ellos con múltiples partes derecha, motivo por el cual serán más costosos computacionalmente y requerirán unos mayores requerimientos de memoria que otros métodos tradicionales, tales como Newmark, HHT-α, Generalizado-α, etc.

Tamma y Namburu [65, 66, 67] describieron entre 1988 y 1992 la familia de algoritmos γ_s implícitos y explícitos, basándose en la formulación de elementos finitos de Lax-Wendroff [68], los cuales eliminan la necesidad de calcular el valor de la aceleración.

En 2002, Modak y Sotelino crearon el método Generalizado [69], el cual emplea aproximaciones de series de Taylor para describir el desplazamiento, la velocidad y la aceleración entre dos instantes de tiempo consecutivos y donde la ecuación del equilibrio dinámico se satisface según la Aproximación Media Ponderada. El método presenta 9 parámetros y pretende que cualquier otro algoritmo monopaso

existente sea obtenido como un paso particular de éste.

Es posible construir algoritmos de una precisión mayor a partir de un algoritmo con una precisión de segundo orden. A modo de ejemplo, Austin [70] usó en 1993 la técnica de extrapolación de Romberg para mejorar el método de Newmark, evaluando los resultados al final de cada paso de tiempo un número determinado de veces, empleando diferentes tamaños de subpasos de tiempo. El resultado son algoritmos con precisión de cuarto, sexto u octavo orden empleando tres, siete o quince evaluaciones intermedias para cada paso de tiempo. Ciertamente los resultados numéricos son muy precisos, pero los métodos son incondicionalmente inestables, debiendo emplear pasos de tiempo muy pequeños a fin de mantener la estabilidad numérica sólo durante reducidos periodos de tiempo.

Tarnow y Simo [71] presentaron en 1994 un procedimiento para construir algoritmos con una precisión de cuarto orden a partir de algoritmos de precisión de segundo orden, conservando sus propiedades conservativas y de estabilidad. Para ello, el procedimiento requiere emplear las tres siguientes evaluaciones para cada paso de tiempo: progresiva, en el instante $\alpha\Delta t$, regresiva, en el instante $(1-2\alpha)\Delta t$, y de nuevo progresiva en $\alpha\Delta t$, siendo $\alpha = 1.3512$. Cuando este procedimiento se aplica al método de Newmark, el resultado es el de un método no disipativo computacionalmente más costoso que el inicial.

Fung propuso en 1997 un procedimiento basado en *subpasos* para construir algoritmos de orden superior (tercera, cuarta, quinta, sexta, etc.) a partir del método de Newmark [72]. Para cada paso de tiempo, el método genera de manera independiente los resultados en diferentes instantes de tiempo intermedios, o subpasos. Posteriormente, todos esos resultados son combinados linealmente, empleando diferentes factores de peso, para así obtener una mayor precisión al final de dicho paso de tiempo. El procedimiento ofrece una disipación numérica controlable de altas frecuencias, ajustando apropiadamente los parámetros algorítmicos. A su vez, los algoritmos desarrollados pueden ser incondicionalmente estables, mejorando la precisión de los resultados simplemente empleando un número mayor de instantes de tiempo intermedios. Es evidente que el esfuerzo computacional es mayor con respecto al método de Newmark a medida que aumenta el número de subpasos empleados.

Si los subpasos elegidos y los factores de peso son números complejos, el método resultante es conocido como el método de Pasos de Tiempo Complejos [73], desarrollado en 1998 por Fung. En general, si hablamos de algoritmos con un orden de precisión de $(2n-1)$ y donde el radio espectral último μ es un parámetro algorítmico controlable, n deberá ser el número de subpasos de tiempo a emplear, los cuales se corresponden con las raíces, que pueden ser complejas, de un polinomio de grado n conocido expresado en términos de μ. A su vez, los factores de peso se obtienen resolviendo un sistema de n ecuaciones. Si $-1 < \mu \leq 1$, los algoritmos son incondicionalmente estables para todo valor de n. Si el algoritmo no es disipativo, $\mu = 1$, entonces el orden de precisión pasa de $2n-1$ a $2n$. Es lógico pensar que el esfuerzo computacional es mayor con respecto al método de Newmark, a medida que aumenta el orden de precisión de los resultados. A modo de ejemplo, un algoritmo de tercer orden requiere, para cada paso de tiempo, un esfuerzo computacional que es cuatro veces superior al método de Newmark. Fung compara también los resultados de este método con aquellos obtenidos de las aproximaciones de Padé. La aproximación diagonal de Padé, la cual da lugar a algoritmos no disipativos incondicionalmente estables y de un orden mayor que dos, puede derivarse de los métodos de Runge-Kutta [74], del método de Galerkin bidiscontinuo en el tiempo [75], del método de Petrov-Galerkin [76] o del método Residual Ponderado [77, 78]. Por otro lado, la aproximación de Padé subdiagonal, la cual da lugar a algoritmos de alto orden, incondicionalmente estables pero *asymptotic annihilating*, puede ser obtenida a partir del método de Galerkin discontinuo en el tiempo [79, 80, 75] del método de Petrov-Galerkin [76] o de la Aproximación Residual Ponderada [78]. Sin embargo dichas aproximaciones suponen incrementar sustancialmente el tamaño así como el ancho de banda en la matriz de coeficientes a emplear en el problema, lo cual aumentará notablemente el tiempo invertido en la resolución de los sistemas de ecuaciones subyacentes, comparándolas por ejemplo con el método tradicional de Newmark.

Con posterioridad, Golley y Amer [81] presentaron en 1999 un algoritmo incondicionalmente estable con amortiguamiento numérico basado en integrar dos ecuaciones ponderadas generales. Tamma, Zhou y Sha [82] propusieron en el año 2001 una teoría general para clasificar y diseñar algoritmos de integración orientados a resolver problemas de dinámica estructural.

En el año 2013, Gholampour, Ghassemieh y Karimi-Rad [83] publicaron un nuevo método implícito en el cual se asume que la aceleración varía cuadráticamente en un paso de tiempo, siendo de esperar que proporcione resultados más precisos que los métodos clásicos. Presenta una precisión de segundo orden y un amortiguamiento numérico controlable de los modos más altos, a la vez que presenta menor disipación numérica en los modos más bajos, en comparación con los métodos de Newmark con aceleración media, Wilson o Generalizado-α. Proporciona además un tiempo de ejecución más rápido que el resto de métodos clásicos.

En lo que respecta a los métodos explícitos, el método de las Diferencias Centradas es el más conocido y ampliamente empleado [84], donde el desplazamiento se expresa en términos de una serie de Taylor. El método ofrece una precisión de segundo orden pero no posee amortiguamiento numérico y es condicionalmente estable. Fu [85] describió en 1970 otro método también explícito y condicionalmente estable. Krieg [86] presentó en 1973 diferentes aproximaciones explícitas aplicables a problemas lineales no amortiguados, pero con precisión de primer orden. Trujillo [87] desarrolló en 1977 un método explícito con precisión de segundo orden e incondicionalmente estable. Kujawski y Gallagher [88] presentaron en 1984 una familia de algoritmos explícitos de mayor orden. Hahn [89] describió en 1991 el Método de Euler Modificado (MEM), el cual fue aplicado para obtener la respuesta dinámica de un edificio sometido a una excitación sísmica así como al impacto de las olas del mar sobre una estructura. Otro tipo de algoritmos explícitos son por ejemplo los que surgen de emplear variaciones polinomiales en el tiempo y la Aproximación Residual Ponderada [39, 50, 51, 52], la cual dio lugar por ejemplo al método SS22, dentro de la familia SSpj desarrollada en 1984 por Zienkiewicz, Wood, Hine y Taylor [51].

Partiendo de líneas similares Hoff y Taylor desarrollaron en 1990 nuevos algoritmos explícitos [90, 91]. Pezeshk y Camp [92] describieron también en 1995 otro método explícito. Algoritmos explícitos más avanzados son los desarrollados por Hulbert y Chung en 1996 [93] que introducían disipación numérica, o el de Hulbert y Jang [94] de 1995 que presenta un control automático del tamaño del paso de tiempo a emplear.

Alternativamente, Tamma y Namburu [65, 95] y Li, Tamma y Namburu [96]

mostraron en 1990 y 1991 los desarrollos iniciales hacia una formulación explícita más efectiva, condicionalmente estable y con precisión de segundo orden con carácter genérico y orientada a la dinámica computacional no lineal. Dicha formulación parte de la filosofía de elementos finitos de tipo Lax-Wendroff [68] y de Taylor-Galerkin. Estos métodos poseen una capacidad autónoma para ponerse en marcha por sí solos y ofrecen mayor precisión que el método de diferencias centradas. Además, no es necesario calcular las aceleraciones en el proceso de análisis y parecen destacar por su simplicidad y facilidad a la hora de ser implementados en un computador.

Es posible también aplicar otro tipo de métodos explícitos clásicos y bien conocidos, como es el método de Runge-Kutta de cuarto orden [97, 98, 99]. Sin embargo el método requiere obtener el valor de la aceleración cuatro veces para cada paso de tiempo, a partir de la ecuación dinámica del movimiento, debiendo resolver cuatro sistemas de ecuaciones lineales que comparten a la matriz de masa como matriz de coeficientes. Existen alternativas a dicho método clásico a fin de determinar el error cometido en cada paso de tiempo, como son el método de Runge-Kutta-Fehlberg de orden uno a tres o el método de Runge-Kutta adaptativo [100], donde el tamaño del paso de tiempo depende del error local estimado [101]. Sin embargo, Braekhus y Aasen [102] compararon en 1981 las prestaciones del método de las Diferencias Centras frente a los métodos de Runge-Kutta, mostrando la superioridad del método de las Diferencias Centradas en términos del tiempo computacional empleado. Humar y Wright [103] aplicaron los métodos predictor-corrector [97, 104] al análisis dinámico de estructuras en 1974. En 1975, Armen, Pifko y Levine [105] y Park, Felippa y DeRuntz [106] aplicaron un método de segundo orden Predictor-Corrector basado en Adams-Molton al análisis dinámico estructural no lineal, con tiempos de computación también superiores a las Diferencias Centradas.

Como ya hemos comentado, el inconveniente de los métodos explícitos es que habitualmente no son incondicionalmente estables, debiendo emplear pequeños pasos de tiempo para mantener la estabilidad. Esa dificultad es aún mayor en estructuras en las cuales existan grandes diferencias entre diferentes partes de la misma, bien sea por presentar materiales distintos, porque los tamaños en los elementos estructurales sean muy dispares o también porque las propiedades

dinámicas son muy diferentes. Teniendo en cuenta que la limitación en el paso de tiempo empleado viene dada por la frecuencia más alta de la estructura ($\Delta t \leqslant \frac{2}{\omega_{max}}$), aquellos elementos estructurales que presenten una frecuencia natural elevada obligarán a emplear pasos de tiempo pequeños al analizar la estructura conjuntamente, incrementando el coste computacional del análisis.

Con el objetivo de limitar esta desventaja, se han desarrollado otro tipo de aproximaciones de integración en las cuales la estructura se divide en diferentes subdominios. Surgen de este modo los esquemas mixtos del tipo explícito-implícito, donde un subdominio se integra empleando un método explícito y el otro empleando un método implícito, o mediante esquemas explícito-explícito donde cada subdominio se integra empleando un paso de tiempo de tiempo diferente, dando lugar a lo que podríamos denominar como algoritmos de integración de paso de tiempo múltiples.

Belytschko y Mullen [107] and Bathe [108] presentaron en 1976 uno de los primeros métodos de integración basados en un esquema de integración mixto, el cual consiste en emplear un algoritmo implícito-explícito para resolver la ecuación de equilibrio. En 1978, Hughes y Liu [109, 110] describieron un método que combina, también en un esquema implícito-explícito, versiones provenientes implícitas y explícitas del método de Newmark. Miranda, Ferencz y Hughes [111] desarrollaron en 1989 un método basado en el método HHT-α que poseía disipación numérica. La combinación de las versiones implícitas y explícitas del método HHT-α en un esquema implícito-explícito único ofrecía mejores propiedades que el método desarrollado previamente por Hughes y Liu. En 1996, Hulbert y Chung [93] presentaron una versión explícita del método Generalizado-α, además de un un algoritmo predictor-corrector explícito basado en dicho método, ofreciendo por tanto una disipación numérica controlada. Posteriormente, Daniel [112] reescribió y mejoró en 2003 la aproximación explícita del método Generalizado-α, de manera que cualquiera de los cuatro parámetros algorítmicos pudiera tomar un valor común tanto en la versión explícita del algoritmo como en la implícita, siendo así totalmente compatibles ambas aproximaciones.

En lo que a los algoritmos explícitos de integración de paso de tiempo múltiples se refiere, en la literatura podemos encontrar a modo de ejemplo el presentado por Belytschko y Mullen en 1977 [84], basado en una partición nodal de la estructura,

o el propuesto por Neal y Belytschko en 1989 [113], donde la fracción entre los diferentes pasos de tiempo empleados no tiene por qué ser un número entero.

Con posterioridad, Smolinski, Sleith y Belytschko [114] propusieron en 1996 un algoritmo derivado del método explícito de Newmark-β, el cual sacrificaba la eficiencia computacional en beneficio de la estabilidad. Sin embargo, los pasos de tiempo empleados debían ser múltiples enteros entre sí y no se podía aplicar a más de dos subdominios. Smolinski [115] mejoró dicho algoritmo también en 1996, de manera que pudieran emplearse múltiples subdominios y que los pasos de tiempo no tuvieran que ser múltiples enteros. Daniel [116, 117] también mejoró el algoritmo de Smolinski, Sleith y Belytschko eliminando ciertas restricciones, pero el nuevo método desarrollado tampoco es aplicable a particiones de más de dos subestructuras.

Además, Wu y Smolinski [118] presentaron en el año 2000 un nuevo algoritmo de integración explícito y de paso de tiempo múltiple desarrollado a partir del esquema de integración trapezoidal modificado, previamente implementado por Pezeshk y Camp [92] en el cual se podían emplear más de dos subgrupos pero en cambio los pasos de tiempo debían ser múltiplos enteros. Chen y Ricles [119] desarrollaron en 2008 un nuevo método de integración explícito el que llamaron el método CR, basado en la teoría de control. El método es incondicionalmente estable y ofrece una precisión similar al método de Newmark con aceleración constante.

Con el objetivo de unificar las ventajas de los métodos de integración directa y de las técnicas de análisis modal y de proporcionar una base teórica para el desarrollo de operadores integrales en el tiempo, Sha, Chen y Tamma [120, 121, 122] describieron en 1995 una metodología explícita, incondicionalmente estable, con capacidad autónoma para ponerse en marcha por sí solos y con precisión de segundo orden, conocida como integral en el tiempo de Pulso VIrtual (VIP), sin la necesidad de que la matriz de masa sea diagonal. Entre otras, el método ha sido aplicado con éxito en el análisis dinámico lineal de estructuras en 3D. Si se emplea un número reducido de modos, el método VIP ofrece claras ventajas en precisión y ahorro en tiempo de computación con respecto a otros métodos de integración tradicionales. Para el caso de un análisis no lineal, el problema de valores propios se resuelve una única vez, lo que representa una importante

ventaja computacional.

A continuación se describen con detalle los ocho métodos de integración implementados y paralelizados en esta tesis, prestando especial atención a las características de cada uno de ellos y a las expresiones numéricas necesarias para ser programados en un computador.

4.7.3. El Método de Newmark

Newmark [9] propuso en 1959 el que ha llegado a ser uno de los métodos más populares y ampliamente utilizados para resolver la ecuación del movimiento. Se basa en la suposición de que la aceleración varía linealmente entre dos instantes de tiempo consecutivos. Este método asume que:

$$d(t+\Delta t) = d(t) + \Delta t v(t) + (\Delta t)^2[(\frac{1}{2} - \beta)a(t) + \beta a(t+\Delta t)] \tag{4.30}$$

$$v(t+\Delta t) = v(t) + \Delta t[(1-\gamma)a(t) + \gamma a(t+\Delta t)] \tag{4.31}$$

donde los parámetros β y γ determinan la estabilidad del método y la precisión de los resultados. Mientras que β permite variar el valor de la aceleración para cada paso de tiempo, γ introduce el amortiguamiento numérico. Sean las siguientes constantes de integración:

$$b_1 = \frac{1}{\beta \Delta t^2}, \quad b_2 = \frac{1}{\beta \Delta t}, \quad b_3 = \frac{1}{2\beta} - 1, \quad b_4 = \frac{\gamma}{\beta \Delta t},$$
$$b_5 = \frac{\gamma}{\beta} - 1, \quad b_6 = \Delta t(\frac{\gamma}{2\beta} - 1) \tag{4.32}$$

Si despejamos $a(t+\Delta t)$ en la ecuación (4.30), nos queda que:

$$a(t+\Delta t) = b_1[d(t+\Delta t) - d(t)] - b_2 v(t) - b_3 a(t) \tag{4.33}$$

Sustituyendo $a(t+\Delta t)$ en la ecuación (4.31), obtenemos $v(t+\Delta t)$ en términos de los desplazamientos desconocidos $d(t+\Delta t)$:

$$v(t+\Delta t) = b_4[d(t+\Delta t) - d(t)] - b_5 v(t) - b_6 a(t) \tag{4.34}$$

Tras emplear las expresiones (4.33) y (4.34) en la ecuación del equilibrio dinámico (4.3), y considerando dicha ecuación en el instante de tiempo $t + \Delta t$, llegamos al sistema de ecuaciones lineales a resolver, obteniendo así $d(t + \Delta t)$:

$$\begin{aligned}(K + b_1 M + b_4 C)\, d(t + \Delta t) = f(t + \Delta t) + \\ + M\left[b_1 d(t) + b_2 v(t) + b_3 a(t)\right] + C[b_4 d(t) + b_5 v(t) + b_6 a(t)]\end{aligned} \tag{4.35}$$

La matriz de coeficientes de este sistema es conocida como la *matriz de rigidez efectiva* y a la parte derecha se le denomina *vector de cargas efectivas*. Los desplazamientos de la estructura se obtienen, para cada paso de tiempo, resolviendo el sistema de ecuaciones presente en (4.35). A su vez, las velocidades y aceleraciones se calcularán mediante las ecuaciones (4.34) y (4.33), respectivamente.

Habitualmente los desplazamientos y las velocidades de la estructura en el instante $t = 0$ son conocidos. Sustituyendo dichos valores en la ecuación (4.3), la aceleración en el instante inicial se calcularía, por tanto, resolviendo el siguiente sistema de ecuaciones, donde la matriz de masa adquiere el papel de matriz de coeficientes:

$$Ma(0) = f(0) - Kd(0) - Cv(0) \tag{4.36}$$

Resulta evidente que en caso de que los desplazamientos y velocidades sean nulos inicialmente, ocurre que:

$$Ma(0) = f(0) \tag{4.37}$$

El método de Newmark puede ser visto, en realidad, como una familia de métodos, dependiendo de los valores de los parámetros β y γ. Se obtienen así las siguientes variaciones, algunas de las cuales han recibido un nombre concreto: *Aceleración Media* o *Regla Trapezoidal* ($\gamma = 1/2$, $\beta = 1/4$), *Aceleración Lineal* ($\gamma = 1/2$, $\beta = 1/6$), *Fow-Goodwin* ($\gamma = 1/2$, $\beta = 1/12$) o *Diferencias Centradas* ($\gamma = 1/2$, $\beta = 0$).

En general, la familia de métodos de Newmark es incondicionalmente estable en problemas lineales si $\gamma \geq \frac{1}{2}$ y si $\beta \geq \frac{1}{4}\left(\frac{1}{2} + \gamma\right)^2$. Sin embargo, la precisión de segundo orden y el amortiguamiento numérico de las altas frecuencias no pueden ocurrir simultáneamente. Si bien este método da lugar a resultados con una precisión de segundo orden si $\gamma = \frac{1}{2}$, la disipación de las altas frecuencias sólo tiene

lugar si $\gamma \neq \frac{1}{2}$, introduciendo un amortiguamiento en los resultados a medida que se incrementa este parámetro.

Así por ejemplo, el algoritmo es incondicionalmente estable y presenta amortiguamiento de las altas frecuencias si se satisface la relación $2\beta \geq \gamma > \frac{1}{2}$, aunque ofrece resultados de primer orden. Por el contrario, el algoritmo es incondicionalmente estable y presenta una precisión de segundo orden si $\gamma = \frac{1}{2}$ y $\beta \geq \frac{1}{4}$, aunque no introduce disipación. Hay que destacar que las propiedades de disipación de esta familia de métodos frente a otros, como Wilson-θ o Houbolt, son inferiores, y los modos inferiores podrán verse seriamente afectados.

4.7.4. El Método de Wilson-θ

Con el objetivo de que el método de Aceleración Lineal fuera incondicionalmente estable, Wilson [10] propuso la idea de satisfacer las ecuaciones del movimiento (4.3) no en el instante de tiempo $t + \Delta t$ sino en el instante $t + \theta\Delta t$. El resultado fue un método que es incondicionalmente estable y que ofrece una precisión de segundo orden y disipación numérica de altas frecuencias cuando $\theta > 1.37$, obteniendo la solución más precisa y estable si $\theta = 1.4$ según [123]. Para valores inferiores a 1.37, el método es condicionalmente estable. Hilber y Hughes [55] determinaron que el valor $\theta = 1.420815$ mejoraba las características espectrales del método y debería ser el valor óptimo. De hecho, a medida que el parámetro θ aumenta se reduce la precisión y se incrementa la disipación.

En realidad, el método de Wilson es un método de colocación, asumiendo que la aceleración varía linealmente entre los instantes de tiempo t y $t+\theta\Delta t$, con $\theta \geq 1$. Si el incremento de tiempo entre el intervalo t a $t+\theta\Delta t$ se denota mediante τ, con $0 \leq \tau \leq \theta\Delta t$, la siguiente expresión se corresponde con una variación lineal de la aceleración entre t y $t + \theta\Delta t$:

$$a(t+\tau) = a(t) + \frac{\tau}{\theta\Delta t}[a(t+\theta\Delta t) - a(t)] \tag{4.38}$$

Integrando la ecuación (4.38) dos veces entre t y τ obtenemos los valores

correspondientes a las velocidades y a los desplazamientos:

$$v(t+\tau) = v(t) + \tau a(t) + \frac{\tau^2}{2\theta\Delta t}[a(t+\theta\Delta t) - a(t)] \tag{4.39}$$

$$d(t+\tau) = d(t) + \tau v(t) + \frac{\tau^2}{2}a(t) + \frac{\tau^3}{6\theta\Delta t}[a(t+\theta\Delta t) - a(t)] \tag{4.40}$$

Considerando $\tau = \theta\Delta t$ en (4.40) y en (4.39), respectivamente, los desplazamientos y las velocidades pueden ser expresados como:

$$d(t+\theta\Delta t) = d(t) + \theta\Delta t v(t) + \frac{(\theta\Delta t)^2}{6}[a(t+\theta\Delta t) + 2a(t)] \tag{4.41}$$

$$v(t+\theta\Delta t) = v(t) + \frac{\theta\Delta t}{2}[a(t+\theta\Delta t) + a(t)] \tag{4.42}$$

Sean las siguientes constantes de integración:

$$\begin{gathered} b_1 = \frac{3}{\theta\Delta t}, b_2 = \frac{\theta\Delta t}{2}, b_3 = \frac{6}{(\theta\Delta t)^2}, b_4 = \frac{6}{\theta\Delta t}, b_5 = \frac{\Delta t^2}{2}(1 - \frac{1}{3\theta}), \\ b_6 = \frac{\Delta t^2}{6\theta}, b_7 = \Delta t(1 - \frac{1}{2\theta}), b_8 = \frac{\Delta t}{2\theta}, b_9 = \frac{1}{\theta}, b_{10} = \frac{\theta - 1}{\theta} \end{gathered} \tag{4.43}$$

Si despejamos $a(t+\theta\Delta t)$ en la ecuación (4.41) y la sustituimos en (4.42), ocurre que $v(t+\theta\Delta t)$ y $a(t+\theta\Delta t)$ pueden expresarse en términos de los desplazamientos $d(t+\theta\Delta t)$ del siguiente modo:

$$v(t+\theta\Delta t) = b_1[d(t+\theta\Delta t) - d(t)] - 2v(t) - b_2 a(t) \tag{4.44}$$

$$a(t+\theta\Delta t) = b_3[d(t+\theta\Delta t) - d(t)] - b_4 v(t) - 2a(t) \tag{4.45}$$

Por último, si sustituimos estas últimas ecuaciones (4.44) y (4.45) en la ecuación de equilibrio (4.3) expresada en el intervalo de tiempo $t+\theta\Delta t$, damos lugar a un sistema de ecuaciones lineales cuya resolución obtiene los desplazamientos en el instante $t+\theta\Delta t$:

$$\begin{gathered} (K + b_3 M + b_1 C)d(t+\theta\Delta t) = f(t+\theta\Delta t) + \\ + M[b_3 d(t) + b_4 v(t) + 2a(t)] + C[b_1 d(t) + 2v(t) + b_2 a(t)] \end{gathered} \tag{4.46}$$

donde el vector de cargas puede extrapolarse linealmente de la forma:

$$f(t+\theta\Delta t) = f(t) + \theta[f(t+\Delta t) - f(t)] \tag{4.47}$$

Una vez que los desplazamientos en el instante $t+\theta\Delta t$ son conocidos, tras resolver el sistema de ecuaciones lineales expresado en (4.46), podemos sustituir dichos valores en la ecuación (4.45), obteniendo así las aceleraciones en el mismo instante de tiempo. Posteriormente, ambos valores pueden ser sustituidos en las expresiones (4.40), (4.39) y (4.38), usando $\tau = \Delta t$, con el objetivo de obtener, respectivamente, los desplazamientos, velocidades y aceleraciones en el instante $t+\Delta t$:

$$d(t+\Delta t) = d(t) + \Delta t v(t) + b_5 a(t) + b_6 a(t+\theta\Delta t) \tag{4.48}$$

$$v(t+\Delta t) = v(t) + b_7 a(t) + b_8 a(t+\theta\Delta t) \tag{4.49}$$

$$a(t+\Delta t) = b_9 a(t+\theta\Delta t) + b_{10} a(t) \tag{4.50}$$

En resumen, y para cada paso de tiempo, el método de Wilson calcula en primer lugar los desplazamientos en el instante $t+\theta\Delta t$ (4.46) y a continuación las aceleraciones en ese mismo instante (4.45). Finalmente, el método obtiene los desplazamientos, velocidades y aceleraciones en el instante $t+\Delta t$ por medio de las expresiones (4.48), (4.49) y (4.50), respectivamente.

Este método está considerado como uno de los más apropiados en referencia al amortiguamiento numérico. Sin embargo, el método presenta el inconveniente de presentar una sobreestimación de segundo orden en los desplazamientos y de primer orden en las velocidades en los primeros instantes, si se emplean pasos de tiempo elevados.

4.7.5. El Método de las Diferencias Centradas

El método de las Diferencias Centradas [84] es el método de integración explícito más empleado. En él, los vectores de velocidades y aceleraciones se escriben en la forma:

$$v(t) = \frac{1}{2\Delta t}[d(t+\Delta t) - d(t-\Delta t)] \tag{4.51}$$

$$a(t) = \frac{1}{\Delta t^2}[d(t+\Delta t) - 2d(t) + d(t-\Delta t)] \tag{4.52}$$

Consideremos las siguientes constantes de integración:

$$b_1 = \frac{1}{\Delta t^2}, \quad b_2 = \frac{1}{2\Delta t}, \quad b_3 = \frac{2}{\Delta t^2} \tag{4.53}$$

Si sustituimos las expresiones (4.51) y (4.52) en la ecuación del movimiento (4.3), expresada en el instante de tiempo t, obtenemos la siguiente ecuación, la cual nos permite calcular los desplazamientos en el instante de tiempo $t + \Delta t$:

$$\begin{aligned}(b_1 M + b_2 C)d(t+\Delta t) = f(t) + \\ + M[b_3 d(t) - b_1 d(t-\Delta t)] + b_2 C d(t-\Delta t) - K d(t)\end{aligned} \tag{4.54}$$

En realidad, dicha ecuación (4.54) representa un sistema de ecuaciones lineales, cuya resolución es intrascendente en tiempos de ejecución en el caso de que las matrices M y C sean diagonales. Si ahora sustituimos estos desplazamientos en las expresiones (4.51) y (4.52), obtendremos las velocidades y aceleraciones en el instante de tiempo t, respectivamente.

Resulta sencillo observar que este método requiere de un procedimiento especial para comenzar, puesto que para calcular los desplazamientos en el instante de tiempo Δt son necesarios dichos valores en los instantes $-\Delta t$ y 0. Una alternativa posible puede ser ésta:

$$d(-\Delta t) = d(0) - \Delta t v(0) + \frac{\Delta t^2}{2} a(0) \tag{4.55}$$

Este método ofrece una precisión de segundo orden, pero es condicionalmente estable y no ofrece amortiguamiento numérico.

4.7.6. El Método de Houbolt Monopaso

En 1950, Houbolt diseñó un método de integración multipaso implícito, con una precisión de segundo orden e incondicionalmente estable [45]. El método presenta amortiguamiento numérico, aunque es conocido por amortiguar en exceso

las frecuencias altas. De hecho, es más disipativo que el método de Wilson, y su principal desventaja es la de no disponer de un parámetro que controle la cantidad de amortiguamiento numérico presente.

En un esfuerzo por eliminar el carácter multipaso, Chung y Hulbert introdujeron la familia de algoritmos de Houbolt Monopaso [46], con las mismas características que el método original pero más apropiados desde el punto de vista computacional.

La ecuación genérica del movimiento de esta familia de métodos está compuesta por nueve parámetros. Sin embargo, cuando se imponen las condiciones de estabilidad incondicional, precisión de segundo orden y amortiguamiento numérico, y se intenta evitar una sobreestimación inicial de ciertos valores, el número de parámetros independientes entre sí se reduce a cuatro. Más aún, esos cuatro parámetros pueden ser expresados en términos del parámetro γ_1, el cual, dependiendo del problema, debería tomar el valor $\gamma_1 = \frac{3}{2}$ a fin de minimizar el error en la velocidad obtenida o $\gamma_1 = \frac{1}{2}$ para evitar la oscilación elevada en la velocidad en los primeros pasos de tiempo, fenómeno este último denominado *overshoot* en la bibliografía. De esta forma, la ecuación del movimiento de la familia de métodos de Houbolt Monopaso puede expresarse como:

$$\begin{aligned} &Ma(t+\Delta t) - \frac{1}{2}Ma(t) + \alpha_{c1}Cv(t+\Delta t) + \\ &\alpha_c Cv(t) + \alpha_{k1}Kd(t+\Delta t) = \alpha_{k1}f(t+\Delta t) \end{aligned} \tag{4.56}$$

Además, las expresiones empleadas para obtener los vectores de desplazamientos y velocidades son las siguientes:

$$d(t+\Delta t) = d(t) + \Delta t v(t) + (\frac{1}{2} - \beta_1)\Delta t^2 a(t) + \beta_1 \Delta t^2 a(t+\Delta t) \tag{4.57}$$

$$v(t+\Delta t) = v(t) + \frac{1}{2}(\frac{1}{2} - \gamma_1)\Delta t a(t) + \gamma_1 \Delta t a(t+\Delta t) \tag{4.58}$$

siendo:

$$\beta_1 = \frac{1}{2}(\frac{1}{2} + \gamma_1),\ \alpha_{k1} = \frac{1}{2\beta_1},\ \alpha_{c1} = \frac{1+\beta 1}{(2\beta_1)^2},\ \alpha_c = \frac{\beta_1 - 1}{(2\beta_1)^2} \tag{4.59}$$

Consideremos las siguientes constantes de integración:

$$b_1 = \frac{1}{\beta_1 \Delta t^2},\ b_2 = \frac{1}{\beta_1 \Delta t}\ ,\ b_3 = \frac{1}{2\beta_1} - 1,\ b_4 = \frac{\gamma_1}{\beta_1 \Delta t},\ b_5 = \frac{\gamma_1}{\beta_1} - 1,$$
$$b_6 = \frac{-\Delta t}{2}(\frac{1}{2} - \frac{\gamma_1}{\beta_1} + \gamma_1),\quad b_7 = b_4 \alpha_{c1},\quad b_8 = \alpha_{k1},$$
$$b_9 = b_3 + \frac{1}{2},\quad b_{10} = b_5 \alpha_{c1} - \alpha_c,\quad b_{11} = b_6 \alpha_{c1} \tag{4.60}$$

Si despejamos $a(t + \Delta t)$ en (4.57) en términos de $d(t + \Delta t)$, obtenemos que:

$$a(t + \Delta t) = b_1[d(t + \Delta t) - d(t)] - b_2 v(t) - b_3 a(t) \tag{4.61}$$

Si ahora sustituimos dicha expresión (4.61) en (4.58), la velocidad en el instante $t + \Delta t$ se expresa de la forma:

$$v(t + \Delta t) = b_4[d(t + \Delta t) - d(t)] - b_5 v(t) - b_6 a(t) \tag{4.62}$$

Finalmente, la sustitución de (4.61) y de (4.62) en (4.56) da lugar al siguiente sistema de ecuaciones lineales, el cual debe ser resuelto para obtener los desplazamientos en el instante $d(t + \Delta t)$:

$$(b_8 K + b_1 M + b_7 C) d(t + \Delta t) = b_8 f(t + \Delta t) +$$
$$+ M[b_1 d(t) + b_2 v(t) + b_9 a(t)] + C[b_7 d(t) + b_{10} v(t) + b_{11} a(t)] \tag{4.63}$$

En resumen, y para cada paso de tiempo, el algoritmo calcula en primer lugar los desplazamientos en dicho instante de tiempo resolviendo la ecuación (4.63), obteniendo posteriormente las velocidades y aceleraciones gracias a (4.62) y (4.61), respectivamente.

4.7.7. El Método HHT-α

El método de Hilber, Hughes y Taylor (HHT-α) [56] es una variante del método de Newmark con el objetivo de introducir un amortiguamiento numérico y controlable de las altas frecuencias sin degradar la precisión de los resultados. Hilber, Hughes y Taylor mostraron que la respuesta del sistema podía ser calculada

con mayor precisión si, en la ecuación (4.3) del equilibrio dinámico, los valores de la velocidad y la aceleración se toman en el instante de tiempo $t + \Delta t$ y los desplazamientos en el instante $t+\Delta t-\alpha$. De este modo, la ecuación del equilibrio dinámico se modifica de acuerdo a:

$$\begin{gathered} Ma(t+\Delta t) + (1+\alpha)Cv(t+\Delta t) - \alpha Cv(t) + \\ (1+\alpha)Kd(t+\Delta t) - \alpha Kd(t) = (1+\alpha)f(t+\Delta t) - \alpha f(t) \end{gathered} \tag{4.64}$$

donde α es el parámetro de disipación numérica, de manera que si α decrece se incrementa el amortiguamiento numérico. El algoritmo se basa en las mismas premisas iniciales en los desplazamientos y velocidades que las asumidas en el método de Newmark, expresadas en (4.30) y en (4.31). Por tanto, los valores de velocidades y aceleraciones en el instante $t + \Delta t$ son los correspondientes a las ecuaciones (4.34) y (4.33). Consideremos las siguientes constantes de integración:

$$\begin{gathered} b_1 = \frac{1}{\beta \Delta t^2},\ b_2 = \frac{1}{\beta \Delta t},\ b_3 = \frac{1}{2\beta} - 1,\ b_4 = \frac{\gamma}{\beta \Delta t},\ b_5 = \frac{\gamma}{\beta} - 1, \\ b_6 = \Delta t(\frac{\gamma}{2\beta} - 1),\ b_7 = 1 + \alpha,\ b_8 = b_4 b_7,\ b_9 = b_5 + \alpha\frac{\gamma}{\beta},\ b_{10} = b_6 b_7 \end{gathered} \tag{4.65}$$

Reemplazando las expresiones (4.34) y (4.33) en (4.64), obtenemos un sistema de ecuaciones lineales que debe ser resuelto para calcular los desplazamientos en el instante de tiempo $t + \Delta t$:

$$\begin{gathered} (b_7 K + b_1 M + b_8 C)d(t+\Delta t) = b_7 f(t+\Delta t) - \alpha f(t) + \\ + M[b_1 d(t) + b_2 v(t) + b_3 a(t)] + C[b_8 d(t) + b_9 v(t) + b_{10} a(t)] + K\alpha d(t) \end{gathered} \tag{4.66}$$

En resumen, por tanto, las ecuaciones (4.66), (4.34) y (4.33) deben ser resueltas para cada paso de tiempo obteniendo así, respectivamente, los vectores de desplazamientos, velocidades y aceleraciones correspondientes.

Como puede observarse, el método HHT-α equivale al método de Newmark si $\alpha = 0$. Si $-\frac{1}{3} \leq \alpha \leq 0$, $\gamma = \frac{1}{2} - \alpha$ y $\beta = \frac{(1-\alpha)^2}{4}$, el algoritmo es incondicionalmente estable y presenta una precisión de segundo orden. Por otro lado, el parámetro α se obtiene de la forma $\alpha = (\mu - 1)/(\mu + 1)$, siendo $\frac{1}{2} \leq \mu \leq 1$ el llamado *radio espectral último*, parámetro que permite controlar el amortiguamiento numérico. Cuando más negativo sea α, y más cercano esté por tanto el radio espectral a 0.5,

mayor será el factor de disipación numérica.

4.7.8. El Método WBZ-α

El método de Wood, Bossak y Zienkiewicz (WBZ-α) [57] es muy similar al método HHT-α. Al igual que este último, se diseño como una variante del método de Newmark, con la idea de introducir un amortiguamiento artificial controlable en las frecuencias más altas. En la ecuación del equilibrio dinámico (4.3), los vectores de desplazamientos y velocidades se expresan en el instante de tiempo $t+\Delta t$ y la aceleración en el instante $t+\Delta t-\alpha$, con el correspondiente incremento de precisión que ello supone.

En este caso, el parámetro α está relacionado con la matriz de masa, como se muestra en la siguiente ecuación del equilibrio dinámico, en lugar de relacionarse con las matrices de rigidez y amortiguamiento como ocurre con el método HHT-α:

$$(1-\alpha)Ma(t+\Delta t)+\alpha Ma(t)+Cv(t+\Delta t)+Kd(t+\Delta t)=f(t+\Delta t) \quad (4.67)$$

Al igual que el anterior, este método también parte de las expresiones de velocidades (4.30) y desplazamientos (4.31) indicadas en el método de Newmark. Sean las siguientes constantes de integración:

$$\begin{gathered} b_1=\frac{1}{\beta\Delta t^2},\ b_2=\frac{1}{\beta\Delta t},\ b_3=\frac{1}{2\beta}-1,\ b_4=\frac{\gamma}{\beta\Delta t},\ b_5=\frac{\gamma}{\beta}-1, \\ b_6=\Delta t(\frac{\gamma}{2\beta}-1),\ b_7=\frac{1-\alpha}{\beta\Delta t^2},\ b_8=\frac{1-\alpha}{\beta\Delta t},\ b_9=\frac{1-\alpha}{2\beta}-1 \end{gathered} \quad (4.68)$$

Después de la sustitución de las expresiones (4.34) y (4.33) en (4.67), los desplazamientos en el instante de tiempo $t+\Delta t$ se calculan resolviendo el siguiente sistema de ecuaciones lineales:

$$\begin{gathered} (K+b_7M+b_4C)d(t+\Delta t)=f(t+\Delta t)+ \\ +\ M[b_7d(t)+b_8v(t)+b_9a(t)]+C[b_4d(t)+b_5v(t)+b_6a(t)] \end{gathered} \quad (4.69)$$

Así, para cada paso de tiempo, el método obtiene los desplazamientos resolviendo la ecuación (4.69), calculando a continuación los vectores de velocidades y

aceleraciones gracias a las expresiones (4.34) y (4.33), respectivamente.

Con el objetivo de garantizar una estabilidad incondicional y una precisión de segundo orden, se recomienda asignar los siguientes valores a los parámetros del método: $\alpha < 0$, $\gamma = \frac{1}{2} - \alpha$ y $\beta = \frac{1}{4}(1-\alpha)^2$. Como puede observarse, este método equivale al de Newmark si $\alpha = 0$ y no introduce por tanto disipación numérica. El parámetro α puede obtenerse a partir del *radio espectral último* μ, de forma que $\alpha = (\mu-1)/(\mu+1)$ y siendo $0 \leq \mu \leq 1$. A medida que μ se acerca a 0, y α es cada vez más negativo, el amortiguamiento numérico es mayor, siendo completo cuando μ vale 0 y α vale -1.

4.7.9. El Método Generalizado-α

El método Generalizado-α [62] es una combinación de los métodos HHT-α y WBZ-α. En este método, la ecuación del equilibrio (4.3), los desplazamientos, las velocidades y el vector de cargas dinámicas se expresan en el instante $t + \Delta t - \alpha_f$ y las aceleraciones en el instante $t + \Delta t - \alpha_m$. Las expresiones iniciales correspondientes a los desplazamientos y las velocidades son las dadas en (4.30) y en (4.31) y la ecuación del movimiento se escribe de la forma:

$$\begin{aligned} &M[(1-\alpha_m)a(t+\Delta t) + \alpha_m a(t)] + C[(1-\alpha_f)v(t+\Delta t) + \alpha_f v(t)] + \\ &+ K[(1-\alpha_f)d(t+\Delta t) + \alpha_f d(t)] = (1-\alpha_f)f(t+\Delta t) + \alpha_f f(t) \end{aligned} \tag{4.70}$$

El método presenta más parámetros que los dos anteriores y puede ser ajustado y optimizado a las diferentes características algorítmicas deseables. Si $\alpha_m = \alpha_f = 0$, el método se reduce al método de Newmark. Por otro lado, si $\alpha_m = 0$ obtenemos el método HHT-α, o el WBZ-α si $\alpha_f = 0$.

Consideremos las siguientes constantes de integración:

$$\begin{aligned} b_1 = \frac{1}{\beta\Delta t^2},\ b_2 = \frac{1}{\beta\Delta t},\ b_3 = \frac{1}{2\beta} - 1,\ b_4 = \frac{\gamma}{\beta\Delta t},\ b_5 = \frac{\gamma}{\beta} - 1, \\ b_6 = \Delta t(\frac{\gamma}{2\beta} - 1),\ b_7 = 1-\alpha_m,\ b_8 = 1-\alpha_f,\ b_9 = b_1 b_7,\ \ b_{10} = b_4 b_8, \\ b_{11} = b_2 b_7,\ b_{12} = \frac{1-\alpha_m}{2\beta} - 1,\ b_{13} = b_5 - \alpha_f\frac{\gamma}{\beta},\ b_{14} = b_6 b_8 \end{aligned} \tag{4.71}$$

Si los términos de la velocidad y la aceleración, expresados en el instante $t + \Delta t$ en (4.34) y en (4.33) para el método de Newmark, son reemplazados en la ecuación (4.70) obtenemos el siguiente sistema de ecuaciones lineales, cuya resolución conduce a obtener los desplazamientos en el instante $t + \Delta t$:

$$(b_8K + b_9M + b_{10}C)d(t + \Delta t) = b_8 f(t + \Delta t) + \alpha_f f(t) + M[b_9 d(t) + b_{11}v(t) + b_{12}a(t)] + C[b_{10}d(t) + b_{13}v(t) + b_{14}a(t)] - K\alpha_f d(t) \quad (4.72)$$

Por tanto, para cada paso de tiempo, los desplazamientos, las velocidades y las aceleraciones se calculan, respectivamente, por medio de las ecuaciones (4.72), (4.34) y (4.33).

El método es incondicionalmente estable, ofrece una precisión de segundo orden y presenta una disipación controlable de las altas frecuencias si $\alpha_f = \frac{\mu}{\mu+1}$, $\alpha_m = \frac{2\mu-1}{\mu+1}$, $\beta = \frac{(1-\alpha_m+\alpha_f)^2}{4}$ y $\gamma = \frac{1}{2} - \alpha_m + \alpha_f$, siendo $0 \leq \mu \leq 1$ el radio espectral último que permite controlar el amortiguamiento numérico del algoritmo. No existe disipación numérica si $\mu = 1$, pero decrementar el parámetro μ supone incrementar el amortiguamiento numérico.

Si así se desea, es posible determinar α_f y μ en función del parámetro α_m, el cual tomará valores entre -1 y 0.5:

$$\alpha_f = \frac{\alpha_m + 1}{3}, \qquad \mu = \frac{1 + \alpha_m - \alpha_f}{1 - \alpha_m + \alpha_f} \quad (4.73)$$

Alternativamente, también es posible obtener α_m y μ a partir de α_f, cuyos valores oscilarán entre 0 y 0.5:

$$\alpha_m = 3\alpha_f - 1, \qquad \mu = \frac{1 + \alpha_m - \alpha_f}{1 - \alpha_m + \alpha_f} \quad (4.74)$$

Una característica común a los métodos HHT-α, WBZ-α y Generalizado-α es que presentan una sobreestimación de primer orden de la velocidad.

4.7.10. El Método SDIRK

La popularidad alcanzada por los métodos SDIRK (Singly-Diagonal-Implicit Runge-Kutta Methods) [97] puede explicarse por su capacidad para combinar unas buenas propiedades de estabilidad junto con una implementación relativamente sencilla. La llamada *Butcher tableau* se emplea para representar a dichos métodos, donde el parámetro s denota el número de *etapas* del mismo:

$$\begin{array}{c|cccc} \gamma & \gamma & & & \\ c_2 & a_{21} & \gamma & & \\ \vdots & \vdots & & \ddots & \\ c_s & a_{s1} & a_{s2} & \dots & \gamma \\ \hline & b_1 & b_2 & \dots & b_s \end{array}$$

Sea $A \in \mathbb{R}^{sxs}$ una matriz triangular inferior y b, $c \in \mathbb{R}^{sx1}$ vectores, donde la disipación numérica se controla por el parámetro γ y donde se imponen las siguientes condiciones adicionales: $\sum_{j=1}^{r} a_{rj} = c_r$ (lo que implica que $c_1 = \gamma$) junto con $a_{sj} = b_j, j = 1, \dots, s$.

Si este método se aplica a la ecuación del movimiento expresada en (4.3), obtenemos las siguientes expresiones para cada paso de tiempo [64]. En primer lugar, la matriz de coeficientes $T \in \mathbb{R}^{nxn}$, siendo n es el número total de grados de libertad del sistema, es de la forma:

$$T = M + \Delta t \gamma C + (\Delta t \gamma)^2 K \tag{4.75}$$

Después, debemos resolver el siguiente sistema de ecuaciones lineales, compuesto por s partes derecha, obteniendo como solución la matriz $\bar{k}_r \in \mathbb{R}^{nxs}$:

$$\begin{aligned} T\bar{k}_r = f(t + c_r \Delta t) - K \left[d(t) + c_r \Delta t v(t) + \Delta t^2 \sum_{j=1}^{r-1} \bar{a}_{rj} \bar{k}_j \right] - \\ -C \left[v(t) + \Delta t \sum_{j=1}^{r-1} a_{rj} \bar{k}_j \right], \; r = 1, \dots, s \end{aligned} \tag{4.76}$$

donde $\bar{a}_{rj} = \sum_{k=1}^{r} a_{rk} a_{kj}$. A continuación, los desplazamientos y velocidades en el instante $(t + \Delta t)$ se obtienen, respectivamente, a partir de las siguientes expre-

siones:

$$d(t+\Delta t) = d(t) + \Delta t v(t) + \Delta t^2 \sum_{r=1}^{s} \bar{b}_r \bar{k}_r \tag{4.77}$$

$$v(t+\Delta t) = v(t) + \Delta t \sum_{r=1}^{s} b_r \bar{k}_r \tag{4.78}$$

donde $\bar{b}_r = \sum_{k=r}^{s} a_{kr} b_k$. Una vez que se han calculado los desplazamientos y las velocidades, las aceleraciones en el mismo instante de tiempo $t + \Delta t$ se obtienen usando la ecuación del equilibrio (4.3), resolviendo por tanto el siguiente sistema de ecuaciones lineales, siendo la matriz de masa la matriz de coeficientes del sistema:

$$Ma(t+\Delta t) = f(t+\Delta t) - Ku(t+\Delta t) - Cv(t+\Delta t) \tag{4.79}$$

Como puede observarse, son varios los sistemas de ecuaciones que deben resolverse para cada paso de tiempo, lo cual supone una clara contradicción a las características que debía cumplir un método de integración, expresadas en [40], donde se argumentaba que no debía resolverse más de un sistema de ecuaciones lineales por cada paso de tiempo.

Sin embargo, las matrices T y M de coeficientes no cambian a lo largo del tiempo. Eso supone que si se aplica un método directo para resolver dichos sistemas de ecuaciones, basado por ejemplo en la descomposición de Cholesky, las matrices T y M serán factorizadas una sola vez, resolviendo para cada paso de tiempo varios sistemas triangulares.

Los métodos SDIRK consideran 2, 3 ó 4 etapas. Los valores para la matriz A y los vectores b y c, dependiendo del número de etapas elegidas, se detallan en [64].

Independientemente del número de etapas, los métodos son *asymptotically annihilating*. Los métodos de 2 etapas son incondicionalmente estables y presentan una precisión de segundo orden si $\gamma = 1 - \frac{\sqrt{2}}{2}$, pero no poseen parámetros que permitan controlar la disipación numérica. Los métodos de 3 etapas son incondicionalmente estables, con una precisión de segundo orden y con un amortiguamiento controlable si el parámetro γ toma valores comprendidos entre 0.1804 y 2.1856. Finalmente, los métodos de 4 etapas son también incondicionalmente estables, con una precisión de tercer orden y con una disipación algorítmica controlable de

las altas frecuencias si el parámetro γ varía desde 0.2236 hasta 0.5728. El error cometido en todos estos métodos es más pequeño o comparable a métodos como el HHT-α, y no presentan problemas de sobreestimación en los desplazamientos o en la velocidad.

4.7.11. La Integral de Duhamel

El movimiento de un sistema de un solo grado de libertad viene gobernado por la siguiente ecuación diferencial:

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = f(t) \tag{4.80}$$

Si dividimos los términos de dicha ecuación por la masa m del sistema resulta:

$$\ddot{x}(t) + 2\xi\omega\dot{x}(t) + \omega^2 x(t) = \frac{f(t)}{m} \tag{4.81}$$

siendo ω la frecuencia natural del sistema no amortiguado y ξ el factor de amortiguamiento crítico:

$$\omega = \sqrt{\frac{k}{m}}, \quad \xi = \frac{c}{2\omega m} \tag{4.82}$$

Considerando un sistema elástico y lineal, la respuesta $x(t)$ en un instante t puede obtenerse como la suma de las respuestas ante la actuación de un conjunto de impulsos elementales sucesivos, cada uno de ellos aplicado en un instante τ, con $0 \leq \tau \leq t$, con una duración $d\tau$ y con una intensidad $f(\tau)$:

$$x(t) = \frac{1}{m\omega_\xi}\int_0^t f(\tau)h(t)d\tau = \frac{1}{m\omega_\xi}\int_0^t f(\tau)e^{-\xi\omega(t-\tau)}sen\left(\omega_\xi(t-\tau)\right)d\tau \tag{4.83}$$

A dicha integral se le denomina *Integral de Duhamel* [5, 37] y puede emplearse para calcular la respuesta a lo largo del tiempo de un sistema de un grado de libertad bajo la actuación de una carga dinámica cualquiera. El integrando $h(t)$ representa la respuesta en el instante t a un impulso unidad (delta de Dirac) aplicado en el instante τ.

A ω_ξ se le denomina la frecuencia natural del sistema amortiguado y se obtiene

de la forma $\omega_\xi = \omega\sqrt{1-\xi^2}$. La ecuación (4.83) no considera el efecto de las condiciones iniciales, motivo por el cual sólo es válida cuando dichas condiciones, a nivel de desplazamientos y velocidades, sean nulas.

La Integral de Duhamel tiene solución analítica solamente para determinadas funciones que describen la carga dinámica aplicada, por lo que debe resolverse, en general, numéricamente. Si dicha carga $f(t)$ se define a trozos y es lineal en cada uno de los intervalos de tiempo, la Integral de Duhamel tiene solución conocida, motivo por el cual puede obtenerse la solución analítica de la ecuación del movimiento. De manera alternativa, la ecuación (4.83), y por tanto la solución del sistema de un grado de libertad, puede expresarse del siguiente modo:

$$x(t) = \frac{e^{-\xi\omega t}}{m\omega_\xi}[A(t)sen\,(\omega_\xi t) - B(t)cos\,(\omega_\xi t)] \tag{4.84}$$

siendo:

$$A(t) = \int_0^t f(\tau)e^{\xi\omega\tau}cos\,(\omega_\xi\tau)\,d\tau \tag{4.85a}$$

$$B(t) = \int_0^t f(\tau)e^{\xi\omega\tau}sin\,(\omega_\xi\tau)\,d\tau \tag{4.85b}$$

Al ser $f(t)$ lineal en el intervalo comprendido entre $t-\Delta t$ y t, y teniendo en cuenta que $t-\Delta t \leq \tau \leq t$, es posible expresar $f(\tau)$ como:

$$f(\tau) = f(t-\Delta t) + s\,(\tau - (t-\Delta t)) \tag{4.86}$$

donde s representa la pendiente de la recta:

$$s = \frac{f(t) - f(t-\Delta t)}{\Delta t} \tag{4.87}$$

Utilizando estas formulaciones, las expresiones (4.85) se reescriben de la forma:

$$A(t) = A(t-\Delta t) + \frac{e^{\xi\omega\tau}}{\omega^2}\left[\left(\xi\omega f(\tau) + s(1-2\xi^2)\right)cos\,(\omega_\xi\tau) + \right.$$

$$\left. + \frac{\omega_\xi}{\omega}\left(\omega f(\tau) - 2\xi s\right)sin\,(\omega_\xi\tau)\right]\Big|_{t-\Delta t}^{t} \tag{4.88a}$$

$$B(t) = B(t-\Delta t) + \frac{e^{\xi\omega\tau}}{\omega^2}\left[\left(\xi\omega f(\tau) + s(1-2\xi^2)\right)sin\,(\omega_\xi\tau) - \right.$$

$$-\frac{\omega_\xi}{\omega}(\omega f(\tau) - 2\xi s)cos\left(\omega_\xi \tau\right)] \Big|_{t-\Delta t}^{t} \tag{4.88b}$$

Derivando la expresión (4.84) con la solución $x(t)$ del sistema, es posible obtener los valores de la velocidad y la aceleración a lo largo del tiempo, de manera que:

$$\dot{x}(t) = \frac{e^{-\xi\omega t}}{m}\left[A(t)cos\left(\omega_\xi t\right) + B(t)sin\left(\omega_\xi t\right)\right] - \xi w x(t) \tag{4.89}$$

$$\ddot{x}(t) = -\omega^2 x(t) - 2\xi\omega\dot{x}(t) + \frac{f(t)}{m} \tag{4.90}$$

siendo $A(t)$ y $B(t)$ los valores obtenidos en (4.85).

Es necesario destacar que la solución deberá emplear incrementos de tiempo Δt suficientemente pequeños, para asegurar tanto la estabilidad como la precisión de la respuesta obtenida, así como la correcta discretización de la carga dinámica.

4.7.12. Imposición de las condiciones de contorno en los métodos de integración

Los sistemas de ecuaciones lineales planteados en cualquiera de los métodos de integración descritos con anterioridad tienen como incógnitas, según sea el caso, los desplazamientos o las aceleraciones en los nudos de la estructura. Puesto que habrá nudos en los cuales dichos valores serán conocidos a priori (como ocurre por ejemplo en el caso de los apoyos empotrados), tendremos que modificar las ecuaciones del sistema para obtener a los citados valores como solución. Para ilustrarlo, si el grado de libertad j tiene impedidos sus movimientos, seguiremos las distintas alternativas recogidas en la sección 3.4:

1. Eliminar la ecuación j del sistema, lo cual equivale a no ensamblar la fila y la columna j de las matrices de rigidez y masa ni el elemento j del vector de carga dinámica. En consecuencia, tampoco se generarán las filas y las columnas j ni de la matriz de amortiguamiento ni de la matriz efectiva del sistema, ni tampoco la posición j del vector de carga dinámica efectiva.

2. Penalizar la ecuación j del sistema sumando una cantidad elevada, como por ejemplo 10^{30}, al elemento diagonal de las matrices de coeficientes del

sistema de ecuaciones, es decir, de la matriz de rigidez efectiva, si calculamos desplazamientos, o de la matriz de masa, si calculamos aceleraciones. En cualquiera de los dos casos, bien sea al obtener el elemento solución d_j o el a_j, el resultado de dividir por un valor muy elevado valdrá 0.

3. Penalizar la matriz de rigidez, sumando una cantidad elevada como 10^{30} al elemento de la diagonal situado en la fila j y columna j, obteniendo a partir de ella la matriz de amortiguamiento, la matriz de rigidez efectiva y la carga dinámica efectiva, para cada instante de tiempo. El resultado de dividir entre un número muy elevado nos dará 0 como valor de d_j.

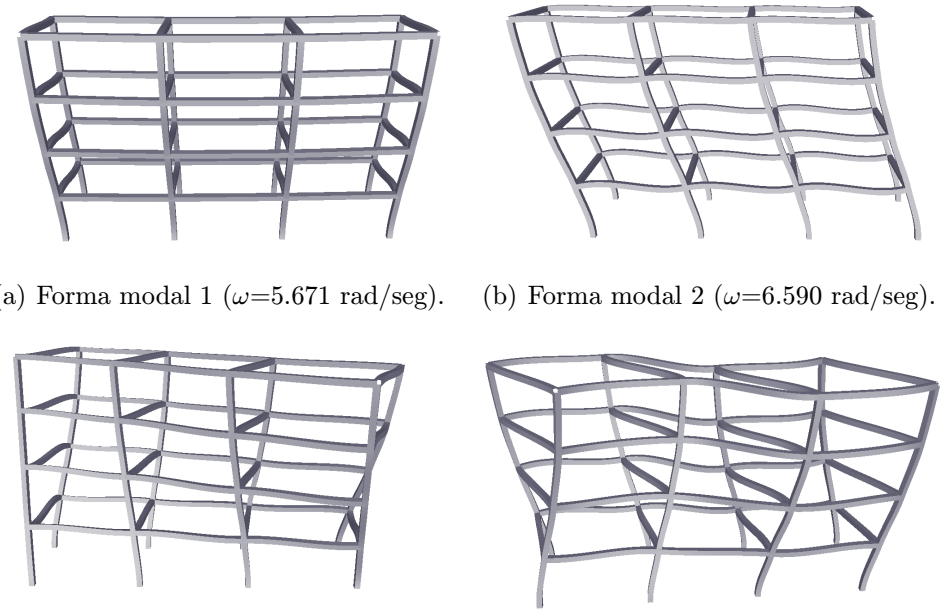
En caso de que existan otro tipo de apoyos elásticos, sumaremos la cantidad equivalente a la elasticidad del soporte al elemento diagonal K_{jj} de la matriz de rigidez. Dicha matriz la emplearemos posteriormente para generar la matriz de amortiguamiento, la matriz efectiva del sistema y la carga dinámica efectiva.

4.8. Análisis dinámico lineal mediante técnicas de análisis modal

4.8.1. Vibraciones libres en sistemas no amortiguados

Definimos como *movimiento vibratorio libre* a aquel en el que la estructura oscila sin que actúe sobre ella ninguna fuerza exterior, quedando definido por las condiciones iniciales de desplazamiento y velocidad. Por otro lado, definimos el *movimiento vibratorio forzado*, como aquel en el cual la estructura oscila bajo la aplicación de una fuerza exterior $F(t)$, dependiendo su movimiento de dicha fuerza y de las condiciones iniciales.

En un sistema con n grados de libertad, cada una de las masas puede moverse independientemente del resto y sólo bajo ciertas condiciones todas ellas tendrán un movimiento armónico con la misma frecuencia. Dicha frecuencia recibe el nombre de *frecuencia natural* y a la configuración deformada del sistema para esta frecuencia se le conoce como *forma modal* o *modo natural de vibración*, siendo el número de frecuencias naturales y modos de vibración de un sistema igual al



(a) Forma modal 1 (ω=5.671 rad/seg). (b) Forma modal 2 (ω=6.590 rad/seg).

(c) Forma modal 3 (ω=7.399 rad/seg). (d) Forma modal 4 (ω=16.069 rad/seg).

Figura 4.2: Primeras frecuencias y modos de vibración de una estructura.

número de grados de libertad que lo forman. A modo de ejemplo, la figura 4.2 nos muestra los cuatro primeros modos de vibración, y sus frecuencias naturales, de una sencilla estructura de barras.

Es conveniente, por tanto, pensar en el modo de vibración como una configuración deformada en la que el movimiento de cada masa es una vibración armónica, alrededor de la posición de equilibrio, con una frecuencia específica asociada a este modo de vibración. Hay que decir además que estas características son intrínsecas al sistema vibrante, estando asociadas al movimiento vibratorio libre, ya que dependen, únicamente, de sus propiedades de masa, amortiguamiento y rigidez y no de las cargas exteriores.

Si bien la resolución de este movimiento vibratorio libre podría parecer intrascendente, no lo es así en absoluto, ya que proporciona las propiedades dinámicas más importantes de la estructura, como son el conocimiento de las frecuencias naturales de cara a la temida resonancia, esto es, la coincidencia de la frecuencia de la excitación externa con una de las frecuencias naturales del sistema. En estas circunstancias, la amplitud de la respuesta aumenta gradualmente hasta el

infinito, pudiendo producir el colapso del sistema. Sin embargo, los materiales comúnmente usados en la práctica están sujetos a límites de resistencia y los fallos estructurales ocurrirán mucho antes de que las amplitudes puedan alcanzar valores extremadamente altos.

Tras omitir los términos de la carga dinámica aplicada y la matriz de amortiguamiento en la ecuación del movimiento (4.3), la ecuación diferencial de movimiento de un sistema lineal no amortiguado con vibración libre es la siguiente:

$$Ma(t) + Kd(t) = 0 \tag{4.91}$$

con las condiciones iniciales en $t = 0$ dadas por:

$$d(0) = d_0,\ v(0) = v_0 \tag{4.92}$$

En el caso de un sistema lineal, su movimiento vibratorio libre será armónico simple, pudiendo ser descrito de la forma:

$$d(t) = \phi \sin(wt + \theta) \tag{4.93}$$

donde ϕ representa la forma del sistema, ω es la frecuencia circular de vibración y θ es un ángulo de fase que depende de los desplazamientos y velocidades de los nudos en el instante inicial $t = 0$. La sustitución de la ecuación (4.93) en la ecuación (4.91) da lugar a la ecuación algebraica que gobierna la vibración libre no amortiguada de un sistema compuesto por múltiples grados de libertad:

$$K\phi = \omega^2 M\phi \tag{4.94}$$

Este conjunto de n ecuaciones, donde n representa el número de grados de libertad considerados en el sistema, equivale en realidad a un problema de valores propios generalizado, siendo $K, M \in \mathbb{R}^{nxn}$, respectivamente, las matrices de rigidez y masa de la estructura. Si dichas matrices son reales, simétricas y definidas positivas, hasta n valores propios ω_i^2 reales, diferentes y mayores que cero, y sus correspondientes vectores propios, $\phi_i \in \mathbb{R}^{nx1}$, satisfacen dicha ecuación (4.94), es decir:

$$K\phi_i = \omega_i^2 M\phi_i, \quad i = 1, 2, ..., n \tag{4.95}$$

Las raíces cuadradas de los n valores propios ω_i^2, ordenadas de menor a mayor ($\omega_1 < \omega_2 < \cdots < \omega_n$), constituyen las frecuencias propias o naturales del modelo estructural, expresadas en radianes por segundo. A su vez, cada vector ϕ_i es conocido como un modo natural de vibración, o patrón particular de deformación de la estructura asociada a dicha frecuencia natural ω_i. A la frecuencia más baja ω_1 y a su modo correspondiente ϕ_1 se le denominan *frecuencia* y *modo principal* del sistema. Por otro lado, el resto de frecuencias son conocidas como primer, segundo, etc., armónicos. Dicho primer modo es el que contribuye en mayor medida a la respuesta de la estructura, mientras que los modos superiores, asociados a las frecuencias más altas, presentan una aportación cada vez menor. Obviamente, los periodos propios del modelo, medidos en segundos, y las frecuencias cíclicas, medidas en ciclos por segundo o hercios, se definen por:

$$T_i = \frac{1}{f_i} = \frac{2\Pi}{\omega_i}, \quad i = 1, 2, ..., n \tag{4.96}$$

siendo T_1 es el periodo principal.

Es importante tener en cuenta que, a la hora de generar la matriz K y la matriz M, no se ensamblarán todas aquellas filas y columnas correspondientes a los grados de libertad que tengan su movimiento impedido. Por el contrario, si el grado de libertad se corresponde con un apoyo elástico, sí que se tendrá en consideración, sumando su valor al elemento diagonal correspondiente de la matriz de rigidez.

Puesto que los modos de vibración del sistema no expresan el valor de las amplitudes sino la forma de la deformada que adquiere el sistema vibrando con la correspondiente frecuencia natural, es habitual normalizar los elementos de dichos vectores de manera, por ejemplo, que el valor máximo de cada vector valga la unidad. Otra posibilidad de normalizar los vectores de formas modales, especialmente conveniente, es la siguiente:

$$\phi_i = \frac{\phi_i}{\sqrt{\phi_i^T M \phi_i}}, \quad i = 1, 2, ..., n \tag{4.97}$$

la cual garantiza que $\phi_i^T M \phi_i = 1$.

Por otro lado, los vectores propios ϕ_i, linealmente independientes entre sí,

forman la denominada *matriz Modal* $\Phi \in \mathbb{R}^{nxn}$:

$$\Phi = \begin{bmatrix} \phi_1 & \phi_2 & \cdots & \phi_n \end{bmatrix} \tag{4.98}$$

En la práctica, las frecuencias de vibración más bajas contienen menor energía elástica de deformación, contribuyendo en mayor medida a la respuesta de la estructura. A su vez, las frecuencias más altas contienen un mayor error numérico en su obtención, debido a aspectos tales como la propia discretización de la estructura. Consecuentemente, la solución del problema puede obtenerse con buena precisión empleando únicamente las q frecuencias naturales más pequeñas, con lo que la matriz Modal $\Phi \in \mathbb{R}^{nxq}$ será de la forma:

$$\Phi = \begin{bmatrix} \phi_1 & \phi_2 & \cdots & \phi_q \end{bmatrix} \tag{4.99}$$

No obstante, conviene tener en cuenta el elevado coste computacional que puede conllevar el calcular unos pocos modos de vibración de una estructura de gran dimensión.

Habitualmente, las normativas sismorresistentes incluyen un criterio para decidir el número q de modos de vibración a emplear, entendiéndose como tal todos aquellos que tengan repercusión en los resultados. Así, la NCSE-02 [2] determina que deben considerarse aquellos modos para los que la suma de sus masas efectivas sea superior al 90 % de la masa movilizada por el movimiento sísmico. El Eurocódigo 8 [124] también incluye dicho requerimiento y adicionalmente indica que se deben tener en cuenta todos los modos cuya masa efectiva supere al 5 % de la masa total. Es necesario por tanto determinar la *masa efectiva* de cada modo de vibración del sistema, la cual obtendremos como:

$$\text{Mef}_i = \frac{\phi_i^T M J}{\phi_i^T M \phi_i} \phi_i^T M J, \quad i = 1, 2, ..., q \tag{4.100}$$

donde $J \in \mathbb{R}^{nx1}$ será un vector de influencia en el cual sus elementos correspondientes a los grados de libertad longitudinales valdrán 1 y los relativos a los grados de libertad rotacionales valdrán 0:

$$J = [1\,1\,1\,0\,0\,0\,1\,1\,1\,0\,0\,0 \ldots 1\,1\,1\,0\,0\,0]^T \tag{4.101}$$

Teniendo en cuenta la normalización de los modos según (4.97), las masas efectivas de cada modo se calcularán como:

$$\mathrm{Mef}_i = \left(\phi_i^T M J\right)^2, \quad i = 1, 2, ..., q \tag{4.102}$$

No obstante, podemos determinar, si resulta de interés, la masa efectiva de cada modo y para cada eje del sistema. Tras tener en cuenta la citada normalización, tendremos que:

$$\mathrm{Mef}_{ix} = \left(\phi_i^T M J_x\right)^2, \quad \mathrm{Mef}_{iy}\left(\phi_i^T M J_y\right)^2, \quad \mathrm{Mef}_{iz} = \left(\phi_i^T M J_z\right)^2, \quad i = 1, 2, ..., q \tag{4.103}$$

con:

$$\begin{aligned} J_x &= [1\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0 \ldots 1\,0\,0\,0\,0\,0]^T \\ J_y &= [0\,1\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0 \ldots 0\,1\,0\,0\,0\,0]^T \\ J_z &= [0\,0\,1\,0\,0\,0\,0\,0\,1\,0\,0\,0 \ldots 0\,0\,1\,0\,0\,0]^T \end{aligned} \tag{4.104}$$

Como era de esperar, la suma acumulada de las masas efectivas de todos los modos de vibración será igual a la masa total de la estructura movilizada en el sismo. Por tanto, si calculamos dicha masa total como $\mathrm{Mt} = J^T M J$, la normativa exige que:

$$\sum_{i=1}^{q} \mathrm{Mef}_i \geq 0,9\mathrm{Mt} \tag{4.105}$$

Alternativamente, es también habitual expresar dicha masa efectiva de un modo i como el porcentaje o fracción de la masa efectiva total, el cual se obtiene como:

$$\%\mathrm{Meft}_{\mathrm{i}} = 100\ \frac{\mathrm{Mef_i}}{\mathrm{Mt}},\ \mathrm{i} = 1, 2, ..., \mathrm{q} \tag{4.106}$$

4.8.2. Propiedades de los modos de vibración

Los modos de vibración satisfacen las siguientes condiciones de ortogonalidad, respecto a las matrices de rigidez y de masa:

$$\phi_i^T K \phi_j = 0, \quad \phi_i^T M \phi_j = 0, \quad i, j = 1, 2, ..., n, \quad i \neq j \tag{4.107}$$

Esta ortogonalidad de los modos de vibración implica que las siguientes matrices K^* y M^* sean diagonales:

$$K^* = \Phi^T K \Phi, \quad M^* = \Phi^T M \Phi \tag{4.108}$$

donde cada término de la diagonal toma los siguientes valores:

$$K^*_{ii} = \phi_i^T K \phi_i, \quad M^*_{ii} = \phi_i^T M \phi_i, \quad i = 1, 2, \ldots, n \tag{4.109}$$

motivo por el cual si los modos de vibración se han normalizado de acuerdo a la expresión (4.97) ocurre que:

$$M^*_{ii} = \phi_i^T M \phi_i = 1, \quad K^*_{ii} = \phi_i^T M \phi_i = \omega_i^2, \quad i = 1, 2, \ldots, n \tag{4.110}$$

lo que permite expresar, en combinación con la condición de ortogonalidad de (4.107), la condición de ortonormalidad con respecto a la matriz de masa:

$$\Phi^T M \Phi = I \tag{4.111}$$

siendo I la matriz identidad.

Considerando la definición del amortiguamiento de Rayleigh en la expresión (4.23), según la cual la matriz de amortiguamiento C se obtiene como combinación lineal de las matrices de rigidez y masa, siendo α_0 y α_1 coeficientes a determinar, obtenemos la matriz C^* del siguiente modo:

$$C^* = \alpha_0 M^* + \alpha_1 K^* \tag{4.112}$$

Si esto es así y si se han normalizado las formas modales según (4.97), C^* cumple además la condición de ortogonalidad respecto a la matriz modal, siendo por tanto una matriz diagonal con los siguientes valores:

$$C^* = \Phi^T C \Phi, \; C^*_{ii} = \phi_i^T C \phi_i = 2\xi_i \omega_i, \; i = 1, 2, \ldots, n \tag{4.113}$$

donde ξ_i es denominado *el factor de amortiguamiento crítico en el modo* i y se entiende que se está basado en evidencias experimentales.

4.9. Análisis dinámico lineal mediante el método de superposición modal

Una vez que el problema de valores propios ha sido resuelto, calculando las frecuencias naturales y los modos de vibración del sistema, podemos emplear el llamado *Método de Superposición Modal* para desacoplar y resolver la ecuación del movimiento (4.3), calculando los valores de los vectores $d(t)$, $v(t)$ y $a(t)$ a lo largo del tiempo.

Puesto que los modos de vibración de un sistema son linealmente independientes y ortogonales entre sí, forman mediante la matriz modal Φ una base completa, y cualquier movimiento del sistema puede expresarse como combinación lineal de ellos. De este modo, es posible escribir las siguientes expresiones, en las cuales los vectores de desplazamientos, velocidades y aceleraciones en los nudos de la estructura se obtienen mediante una superposición de la respuesta de cada modo de vibración individual. Teniendo en cuenta que solamente son necesarias las q frecuencias naturales más pequeñas para determinar la respuesta del sistema, nos quedaría que:

$$d(t) = \sum_{i=1}^{q} \phi_i x_i(t) = \Phi x(t), \tag{4.114a}$$

$$v(t) = \sum_{i=1}^{q} \phi_i \dot{x}_i(t) = \Phi \dot{x}(t), \tag{4.114b}$$

$$a(t) = \sum_{i=1}^{q} \phi_i \ddot{x}_i(t) = \Phi \ddot{x}(t), \tag{4.114c}$$

siendo $x(t)$, $\dot{x}(t)$ y $\ddot{x}(t) \in \mathbb{R}^{nx1}$ vectores que deben ser calculados. En dichas expresiones, el vector propio ϕ_i describe la forma de vibración de la estructura en el modo i, mientras que $x_i(t) \in \mathbb{R}$ representa su amplitud en el tiempo, también denominada *respuesta generalizada* o *coordenada normal del modo i*. Por otra parte, al vector $x(t)$ también se le denomina *vector de amplitud modal* o *vector de coordenadas modales o normales* del sistema.

La premultiplicación de la ecuación (4.3) por Φ^T, junto con la sustitución en la misma de $d(t)$, $v(t)$ y $a(t)$ por sus formulaciones correspondientes en la

expresión (4.114), además de la aplicación de las condiciones de normalización de los modos de vibración según (4.97) y las de ortogonalidad de las matrices K, M y C expresadas en (4.108) y (4.113) dan lugar a:

$$M^*\ddot{x}(t) + C^*\dot{x}(t) + K^*x(t) = \Phi f(t) \tag{4.115}$$

Si dividimos la anterior expresión por M^* y tenemos en cuenta las condiciones expresadas en (4.109), (4.110) y (4.113) resulta que la ecuación (4.3) queda finalmente desacoplada en el siguiente sistema de q ecuaciones diferenciales ordinarias de segundo orden, donde ξ_i representa el *factor de amortiguamiento crítico* en el modo i, expresado en tanto por uno:

$$\ddot{x}_i(t) + 2\omega_i\xi_i\dot{x}_i(t) + \omega_i^2 x_i(t) = \phi_i^T f(t), \quad i = 1, ..., q \tag{4.116}$$

Por tanto, es ahora necesario resolver un conjunto de q ecuaciones desacopladas, cada una de ellas correspondiente a un modelo dinámico compuesto por un solo grado de libertad. Dichas ecuaciones pueden resolverse de manera totalmente independiente entre sí, empleando para ello cualquiera de los métodos de integración que hemos citado en la sección anterior, desde Newmark hasta Duhamel, implementados específicamente para resolver ecuaciones diferenciales de segundo orden de un único grado de libertad.

De este modo, la solución de las ecuaciones (4.116) y (4.114) para cada paso de tiempo t=Δt, $2\Delta t$, ..., T proporciona la historia de la respuesta de la estructura a lo largo del tiempo, calculando los desplazamientos *d(t)*, velocidades *v(t)* y aceleraciones *a(t)* de los nudos que la componen. Conociendo los desplazamientos *d(t)* para cada paso de tiempo es posible obtener, al igual que un análisis estático o en un análisis dinámico mediante métodos de integración directa, los esfuerzos en los extremos de las barras, las reacciones en los apoyos, los esfuerzos y las deformaciones en los múltiples puntos intermedios en los que cada barra se divide y las tensiones y los esfuerzos en los nudos vértices de los elementos finitos, siguiendo los procedimientos descritos en el capítulo anterior.

Adicionalmente, es de especial interés conocer el *factor de participación* de cada modo de vibración en la respuesta de la estructura, para una carga dinámica aplicada. Dicho factor de participación $\Gamma_i \in \mathbb{R}$ de un modo i recoge el cociente

entre la masa que moviliza el sismo asociada a dicho modo y la masa total asociada al modo i:

$$\Gamma_i = \frac{\phi_i^T M J}{\phi_i^T M \phi_i}, \quad i = 1, 2, ..., q \tag{4.117}$$

donde $J \in \mathbb{R}^{nx1}$ será un vector en el cual cada componente valdrá 1 o 0 dependiendo si la carga dinámica está aplicada sobre ese grado de libertad o no. En el caso de un terremoto que esté actuando sobre los tres ejes principales de la estructura, y a modo de ejemplo, los elementos del vector J que valdrán un 1 serán aquellos correspondientes a los tres grados de libertad longitudinales de cada nudo, mientras que los grados de libertad rotacionales valdrán 0:

$$J = [1\,1\,1\,0\,0\,0\,1\,1\,1\,0\,0\,0 \ldots 1\,1\,1\,0\,0\,0]^T \tag{4.118}$$

Si las direcciones de las componentes del sismo no coincidieran directamente con los ejes principales de la estructura, el vector J estaría formado por cosenos directores.

Es lógico observar que si los modos de vibración están normalizados de acuerdo a (4.97), el denominador del cálculo del factor de participación de cada modo valdrá 1, con lo cual:

$$\Gamma_i = \phi_i^T M J, \quad i = 1, 2, ..., q \tag{4.119}$$

Si deseamos obtener, para cada modo de vibración, su factor de participación con respecto a cada eje, lo haremos así:

$$\Gamma_{ix} = \phi_i^T M J_x, \quad \Gamma_{iy} = \phi_i^T M J_y, \quad \Gamma_{iz} = \phi_i^T M J_z, \quad i = 1, 2, ..., q \tag{4.120}$$

donde los vectores J_x, J_y o J_z son idénticos a los de la expresión (4.29), si las componentes del sismo coinciden con los ejes globales de la estructura.

Al igual como ocurre en el ensamblaje de la matrices K y M, no consideraremos la aportación de todos aquellos grados de libertad empotrados al vector $f(t)$ con la carga dinámica aplicada en cada instante.

De forma resumida, los pasos necesarios para obtener la respuesta de la estructura a lo largo del tiempo mediante las técnicas de análisis por superposición modal son los siguientes:

1. Generar las matrices de rigidez K y de masa M de la estructura.

2. Obtener las frecuencias naturales y las formas modales resolviendo el problema de valores propios generalizado.

3. Determinar las masas efectivas y los factores de participación de cada modo.

4. Para cada paso de tiempo t=Δt, $2\Delta t$, ..., T:

   a) Evaluar el vector de cargas dinámicas y de cargas dinámicas efectivas.

   b) Obtener las amplitudes y sus derivadas al resolver un conjunto de ecuaciones diferenciales independientes de un solo grado de libertad.

   c) Calcular los desplazamientos *d(t)*, velocidades *v(t)* y aceleraciones *a(t)* en los nudos.

   d) Determinar las solicitaciones en los extremos de las barras.

   e) Obtener las reacciones en los apoyos.

   f) Calcular los esfuerzos y las deformaciones en los puntos intermedios de las barras.

   g) Proporcionar las deformaciones unitarias, las tensiones y los esfuerzos en los nudos de los elementos finitos.

   h) Volver al punto 4, con $t = t + \Delta t$.

Como ya hemos comentado, si pretendemos obtener la respuesta de la estructura a lo largo del tiempo tras la actuación de un terremoto, será necesario disponer del acelerograma correspondiente a dicho movimiento sísmico. Sin embargo, la mayoría de las normativas de diseño sismorresistentes definen habitualmente la acción sísmica mediante los llamados *espectros sísmicos de respuesta*. El método de análisis modal espectral, que describiremos a continuación, obtiene la respuesta máxima de la estructura a partir de la definición del sismo en dicho espectro de respuesta.

4.10. Análisis dinámico lineal mediante el método del análisis modal espectral

4.10.1. Descripción

Como alternativa al procedimiento que acabamos de describir, es posible determinar los valores de respuesta máxima de una estructura bajo un terremoto empleando los llamados *espectros sísmicos de respuesta.*

A partir de un cierto acelerograma definido por una aceleración $a(t)$ correspondiente a una de las tres posibles componentes de un terremoto, el *espectro sísmico de respuesta* de desplazamientos, velocidades o aceleraciones consiste en un diagrama que proporciona las respuestas máximas Sd, Sv y Sa de sistemas de un grado de libertad expresadas en función de la frecuencia ω o periodo T de la estructura, para un factor de amortiguamiento crítico ξ dado.

Para obtener dicho espectro sísmico de respuesta es necesario aplicar un procedimiento de integración numérica, como los vistos en la sección 4.7, que nos resuelva una ecuación diferencial de segundo orden correspondiente a un sistema de un grado de libertad, como la mostrada en la ecuación (4.116). En dicho procedimiento de integración, se emplearán incrementos de tiempo reducidos, hasta alcanzar el registro total del terremoto, y donde los valores máximos obtenidos de desplazamientos, velocidades y aceleraciones se registrarán como puntos de la respuesta espectral del sistema caracterizado por una frecuencia natural y un factor de amortiguamiento determinado.

Dicho proceso debe repetirse para un amplio abanico de frecuencias naturales y factores de amortiguamiento diferentes, obteniendo así, en un único diagrama, las respuestas máximas del sistema en función de sus frecuencias o periodos. A dicho diagrama, bien sea de desplazamientos, velocidades o aceleraciones, se le denomina *espectro de respuesta.*

A partir de los espectros de respuesta correspondientes a un número suficiente de terremotos característicos de una determinada región, se determinan los *espectros de diseño*, formados por la media de los valores de desplazamientos,

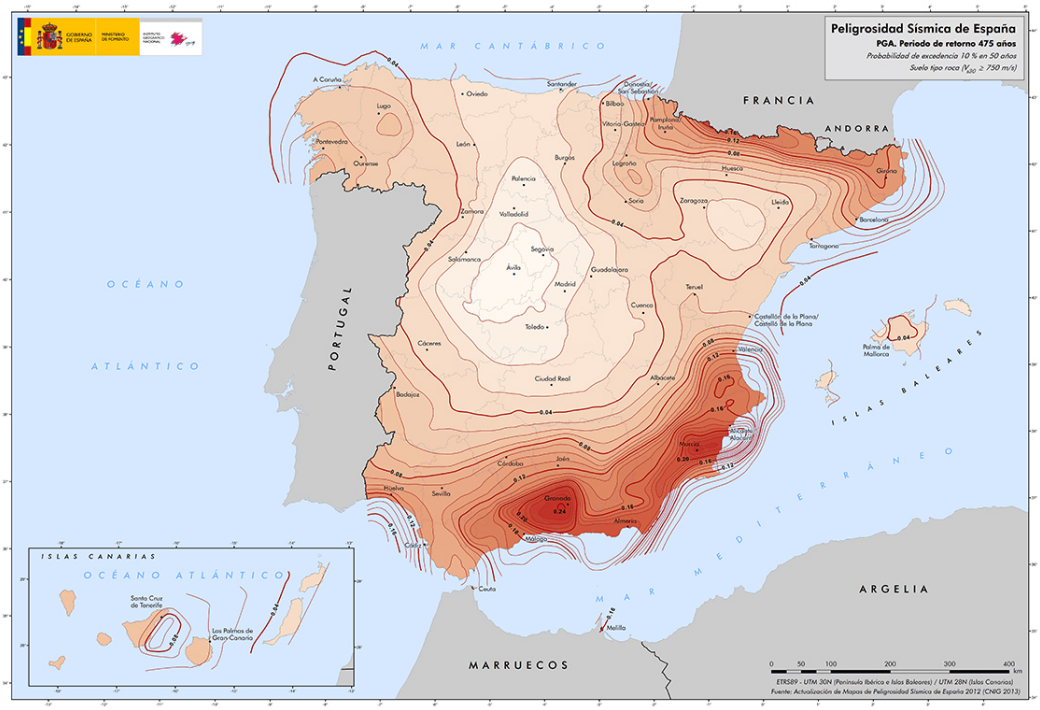


Figura 4.3: Mapa de peligrosidad sísmica de España en valores de aceleración (INE).

velocidades o aceleraciones de los espectros de respuesta de partida.

Así, las normativas sismorresistentes incluyen, para diferentes ubicaciones geográficas, los parámetros necesarios para determinar los espectros sísmicos de diseño, válidos para llevar a cabo un análisis sísmico sobre las estructuras a construir en dicha zona, en forma de tabla o mediante un mapa como el que se muestra en la figura 4.3 para la normativa española.

El llamado *análisis modal espectral* calcula la respuesta máxima del sistema combinando las respuestas máximas calculadas para cada uno de sus modos, de acuerdo a la acción sísmica caracterizada por el espectro de respuesta elegido. Para ello, y de manera similar al análisis por superposición modal, será necesario calcular previamente las frecuencias propias y los modos de vibración de la estructura.

Una vez obtenidas las frecuencias naturales y los modos de vibración que caracterizan al sistema, tendremos que calcular, dependiendo del espectro de diseño

empleado, los valores máximos de la aceleración $Sa_i \in \mathbb{R}$ para cada modo i de vibración, a partir de su periodo T_i. Conocidos dichos valores de aceleración máxima del modo, obtendremos posteriormente sus velocidades y desplazamientos máximos, Sv_i y $Sd_i \in \mathbb{R}$, como:

$$Sv_i = \frac{Sa_i}{\omega_i} \tag{4.121a}$$

$$Sd_i = \frac{Sa_i}{\omega_i^2}, \qquad i = 1, 2, ..., q \tag{4.121b}$$

A partir de ellos, los desplazamientos, velocidades y aceleraciones modales máximos d_i, v_i, $a_i \in \mathbb{R}^{nx1}$ en los nudos de la estructura, para cada modo de vibración i, pueden calcularse de la siguiente forma, teniendo en cuenta la normalización de los modos de vibración presentada en (4.97):

$$d_i = \phi_i \left|x_i(t)\right|_{max} = \phi_i \Gamma_i Sd_i \tag{4.122a}$$

$$v_i = \phi_i \left|\dot{x}_i(t)\right|_{max} = \phi_i \Gamma_i Sv_i \tag{4.122b}$$

$$a_i = \phi_i \left|\ddot{x}_i(t)\right|_{max} = \phi_i \Gamma_i Sa_i, \ i = 1, \ldots, q \tag{4.122c}$$

Al término $\left|x_i(t)\right|_{max} \in \mathbb{R}$ se le denomina *factor de amplificación* del modo i y se obtiene multiplicando el factor de participación del modo Γ_i por su desplazamiento máximo Sd_i proporcionado por el espectro de diseño. Conviene aclarar que, en el caso del análisis modal espectral, el factor de participación de cada modo lo obtendremos según (4.119), pero empleando el vector J de influencia definido en (4.136).

Estos desplazamientos máximos d_i en los nudos de la estructura, como aportación del modo i, son el punto de partida para calcular el resto de resultados a determinar para dicho modo, tales como los esfuerzos en los extremos de las barras, las reacciones en los apoyos, los esfuerzos y deformaciones en los puntos intermedios de las barras y las tensiones y los esfuerzos en los nudos de los elementos finitos.

En principio y como primera posibilidad, podríamos pensar en obtener los

desplazamientos máximos $d \in \mathbb{R}^{nx1}$ en los nudos de una estructura como la suma de los valores máximos correspondientes a cada modo, y obtener a partir de ellos el resto de resultados. Sin embargo, puesto que en cada grado de libertad el máximo de cada modo no se produce en el mismo instante de tiempo, la respuesta máxima de la estructura superaría la respuesta máxima real. Es decir:

$$d \neq \sum_{i=1}^{q} d_i \tag{4.123}$$

Son por tanto múltiples las técnicas desarrolladas para aproximar apropiadamente los valores máximos de las variables de respuesta, tal y como describimos en la sección 4.10.4, a partir de los valores máximos de dichas variables para cada uno de los modos. Incluso, veremos que muchos de los resultados a proporcionar gracias al cálculo modal espectral de la estructura no deben obtenerse a partir de los valores máximos obtenidos en los nudos, sino como una combinación apropiada de los valores máximos de dichos resultados para cada modo.

Por último, conviene aclarar lo siguiente. Tal y como hemos comentado, las normativas de diseño definen habitualmente la acción de un terremoto mediante un espectro de respuesta, motivo por el cual lo más inmediato consiste en calcular la respuesta máxima de la estructura. Sin embargo, si se desea obtener la respuesta a lo largo del tiempo, bien sea mediante los métodos de integración directa o mediante las técnicas de superposición modal, es posible generar a partir de dichos espectros familias de acelerogramas compatibles [125].

De forma resumida, a continuación se recoge el proceso encargado de obtener la respuesta máxima de una estructura mediante el método de análisis modal espectral:

1. Generar las matriz de rigidez K y de masa M.

2. Obtener las frecuencias naturales y las formas modales resolviendo el problema de valores propios generalizado.

3. Determinar las masas efectivas y los factores de participación de cada modo.

4. Calcular la aceleración, la velocidad y el desplazamiento espectral máximo

para cada modo de vibración, empleando técnicas de combinación direccional.

5. Obtener los desplazamientos, velocidades y aceleraciones máximos en los nudos, para cada modo de vibración.

6. Calcular las solicitaciones en los extremos de las barras, para cada modo.

7. Determinar las reacciones en los apoyos, para cada modo de vibración.

8. Obtener los esfuerzos y las deformaciones en los puntos intermedios de las barras, para cada modo.

9. Calcular las deformaciones unitarias, las tensiones y los esfuerzos en los nudos de los elementos finitos, para cada modo de vibración.

10. Combinar los resultados de los diferentes modos de vibración mediante técnicas de combinación modal.

11. Determinar el signo de los resultados.

4.10.2. Cálculo de la aceleración espectral máxima

El cálculo de la aceleración espectral máxima S_a es totalmente dependiente del espectro de respuesta que forma parte de la normativa de diseño empleada. En este trabajo se han considerado los 4 siguientes tipos diferentes de espectros: presente en la normativa española (NCSE-02), presente en el Eurocódigo 8 (E8), definido por el usuario y proveniente de un acelerograma. Pasamos a continuación a describir la forma de obtener dicha aceleración espectral en cada uno de ellos, aludiendo a los parámetros de los que depende.

4.10.2.1. Espectro de respuesta de la NCSE-02

Para aceleraciones del terreno horizontales, la norma [2] establece un espectro normalizado de respuesta elástica en la superficie libre del terreno correspondiente

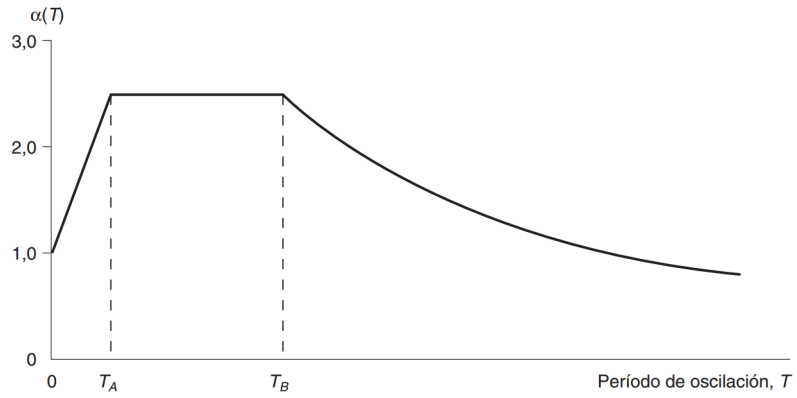


Figura 4.4: Espectro normalizado de respuesta elástica según la NCSE-02.

a un oscilador lineal simple con un amortiguamiento de referencia del 5 % respecto al crítico, tal y como se muestra en la figura 4.4, definido por los siguientes valores:

$$\alpha(T) = \begin{cases} 1 + 1,5\frac{T}{T_A} & \text{si } T < T_A \\ 2,5 & \text{si } T_A \leq T \leq T_B \\ K\frac{C}{T} & \text{si } T > T_B \end{cases} \tag{4.124}$$

siendo $\alpha(T)$ el valor del espectro normalizado de respuesta elástica, T el periodo propio en segundos del modo a evaluar, K el coeficiente de contribución y C el coeficiente del terreno. Los periodos característicos T_A y T_B del espectro se calculan como:

$$T_A = K\frac{C}{10}, \quad T_B = K\frac{C}{2,5} \tag{4.125}$$

Cuando se considere la aceleración vertical del terreno, se adoptará un espectro de respuesta elástica cuyo valor será un 70 % de los valores $\alpha(T)$ obtenidos según (4.124) para una aceleración horizontal del terreno. Este valor del espectro se empleará ahora para obtener la aceleración máxima de cada modo de acuerdo a su periodo:

$$Sa = \begin{cases} \left(1 + \left(2,5\frac{\nu}{\mu} - 1\right)\frac{T}{TA}\right) a_c & \text{si } T < T_A \\ \alpha(T)\,\frac{\nu}{\mu} a_c & \text{si } T \geq T_A \end{cases} \tag{4.126}$$

siendo a_c la aceleración sísmica de cálculo, μ el coeficiente de comportamiento por ductilidad y ν el siguiente valor relacionado con el factor de amortiguamiento

crítico ξ (expresado en tanto por cien) de la estructura:

$$\nu = \left(\frac{5}{\xi}\right)^{0,4} \tag{4.127}$$

La aceleración sísmica de cálculo a_c se obtiene a partir de la aceleración sísmica básica a_b, definida en la figura 4.3, el coeficiente adimensional de riesgo ρ, el cual es función de la probabilidad de que a_c se exceda en el periodo de vida de la estructura, y el coeficiente de amplificación del terreno S:

$$a_c = S\rho a_b \tag{4.128}$$

El valor de la aceleración máxima Sa obtenido habrá que multiplicarlo por el valor de la gravedad, para expresarla en m/s^2.

4.10.2.2. Espectro de respuesta del Eurocódigo 8

El Eurocódigo 8 [124] establece al análisis modal espectral como una de las alternativas posibles para calcular el comportamiento sísmico de una estructura bajo un comportamiento elástico-lineal. Dicho análisis debe ir acompañado del espectro de cálculo que se indica a continuación, para las componentes horizontales de la acción sísmica, siendo Sa el valor de la aceleración espectral para un periodo de vibración T. El espectro de respuesta elástica de esta normativa se muestra en la figura 4.5:

$$Sa = \begin{cases} a_g S \left[\frac{2}{3} + \frac{T}{T_B}\left(\frac{2,5}{q} - \frac{2}{3}\right)\right] & \text{si } 0 \leq T \leq T_B \\ a_g S \frac{2,5}{q} & \text{si } T_B \leq T \leq T_C \\ \text{máx}\left(a_g S \frac{2,5}{q}\frac{T_C}{T}, \beta a_g\right) & \text{si } T_C \leq T \leq T_D \\ \text{máx}\left(a_g S \frac{2,5}{q}\frac{T_C T_D}{T^2}, \beta a_g\right) & \text{si } T \geq T_D \end{cases} \tag{4.129}$$

donde a_g es el valor de la aceleración del suelo en un terreno tipo A, T_B es el límite inferior del periodo del tramo de aceleración espectral constante, T_C representa el límite superior del periodo del tramo de aceleración espectral constante, T_D equivale al valor que define el comienzo del tramo de respuesta de desplazamiento

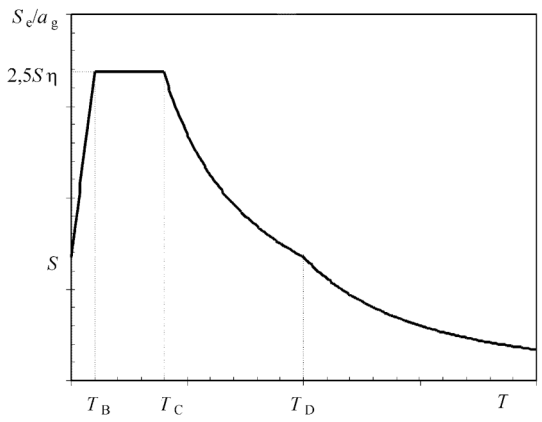


Figura 4.5: Espectro de respuesta elástica, para las componentes horizontales de la acción sísmica, según el Eurocódigo 8.

constante, q es el coeficiente de comportamiento, S representa al coeficiente del suelo y β es el coeficiente correspondiente al umbral inferior del espectro de cálculo horizontal. El valor de todos los parámetros mencionados debe consultarse en la normativa.

Para la componente vertical de la acción sísmica, la expresión (4.129) es válida siempre que se reemplace a_g por a_{vg} y S valga 1. Adicionalmente, se establecen valores diferentes para el resto de parámetros, que deben consultarse en la normativa.

4.10.2.3. Espectro de respuesta definido por el usuario

Bajo la posibilidad de que se desee trabajar con una normativa diferente a las mencionadas con anterioridad o que dichos espectros de respuesta no se ajusten a nuestras necesidades, se ha considerado el hecho de definir un espectro de respuesta normalizado compuesto por un conjunto de parámetros que tratan de aportar flexibilidad en la definición del mismo. Con todo ello, el valor del espectro

$\alpha(T)$, en función del periodo T, sería éste:

$$\alpha(T) = \begin{cases} q\left[f0 + \frac{T}{T_B}\left(\frac{\alpha 0}{q} - f0\right)\right] & \text{si } 0 \leq T \leq T_B \\ \alpha 0 & \text{si } T_B \leq T \leq T_C \\ \alpha 0 \frac{T_C}{T} & \text{si } T_C \leq T \leq T_D \\ \alpha 0 \frac{T_C T_D}{T^2} & \text{si } T \geq T_D \end{cases} \tag{4.130}$$

siendo T_B, T_C y T_D parámetros con un significado idéntico a los del apartado anterior, $f0$ la ponderación de la aceleración de cálculo, α_0 la amplificación de meseta y q un factor de estructura. A partir de dicho valor normalizado del espectro, la aceleración máxima proporcionada correspondiente a un periodo T sería:

$$Sa = \frac{a_c}{q}\alpha(T) \tag{4.131}$$

donde a_c representa la aceleración sísmica de cálculo.

4.10.2.4. Espectro de respuesta proveniente de un acelerograma

En ocasiones, puede ocurrir que el usuario disponga de una acelerograma representativo de una ubicación geográfica concreta y desee realizar un análisis modal espectral a partir del mismo. Para ello, dicho acelerograma deberá ser previamente transformado a un espectro de respuesta, mediante algunas de las diversas técnicas disponibles, proporcionando como salida un conjunto de valores, a pares, que almacenan periodos y sus valores de aceleración máxima. En nuestro caso, hemos contemplado la definición del espectro de este modo, aplicando interpolación lineal para obtener aquellos valores de la aceleración máxima correspondientes a periodos no proporcionados como datos de entrada.

4.10.3. Técnicas de combinación de los efectos de las componentes de la acción sísmica

Si alguno de los parámetros de definición del espectro, tales como la ductilidad, el factor de amortiguamiento crítico, etc., son distintos en las diferentes

direcciones horizontales o verticales de la estructura, los valores de la aceleración máxima S_a proporcionados para un mismo periodo T serán diferentes, debido obviamente a que los espectros de respuesta considerados para cada dirección no serán coincidentes. Un caso claro de lo que estamos comentado viene dado por el Eurocódigo 8, donde los parámetros empleados para el espectro vertical son distintos a los del espectro horizontal, o la propia NCSE-02, que establece que el espectro de respuesta elástica relativo a la aceleración vertical sea un 70 % del valor obtenido para una aceleración horizontal. En ese sentido, las normativas de diseño recomiendan que se lleve a cabo un análisis modal espectral independiente para cada dirección de la aceleración sísmica y se combinen posteriormente los resultados.

Sean R_x, R_y y R_z los resultados máximos de una magnitud, como podrían ser los desplazamientos en los nudos de la estructura o la tensión en los nodos de los elementos finitos, obtenidos tras realizar, respectivamente, un análisis modal espectral independiente considerando única y exclusivamente la actuación de la acción sísmica en una de sus tres componentes, bien sea horizontal (sobre el eje X o Y) o vertical (sobre el eje Z) y tras aplicar las técnicas de combinación de resultados modales de la sección 4.10.4. La respuesta máxima de la estructura R, para dicha magnitud, podría obtenerse mediante la aplicación de alguna de las técnicas que a continuación se enumeran:

- El método de la Raíz Cuadrada de la Suma de los Cuadrados (SRSS) [126] donde, como su propio nombre indica, se suma la respuesta de cada componente, elevada al cuadrado, y se obtiene la raíz cuadrada de dicha suma. Este método asume que no hay correlación entre las componentes sísmicas:

$$R = \sqrt{R_x^2 + R_y^2 + R_z^2} \tag{4.132}$$

- La Regla del 30 %, propuesta por Rosenblueth y Contreras [127], la cual establece diferentes combinaciones que ponderan la respuesta en uno u otro eje, eligiendo después la que proporciona los resultados más desfavorables:

$$R = R_x + 0,3R_y + 0,3R_z$$

$$R = 0,3R_x + R_y + 0,3R_z$$

$$R = 0,3R_x + 0,3R_y + R_z \tag{4.133}$$

- El método de Combinación Cuadrática Completa con 3 Componentes sísmicas (CQC3). Este método fue ideado por Smeby y Kiureghian [128] como una extensión al método CQC (que más adelante describiremos) con la intención de obtener la respuesta de la estructura tras la aplicación de un sismo compuesto por tres componentes (dos horizontales y una vertical). En [129], se muestran las ventajas que proporciona con respecto a los métodos que acabamos de mencionar.

  Entendiéndose que las componentes horizontales del terremoto pueden no estar alineadas con los ejes principales de la estructura (sí que lo estará la componente vertical), el método trata de obtener la respuesta crítica, entendida como tal la respuesta estructural máxima considerando cualquier ángulo de incidencia de las citadas componentes horizontales. Para que el método sea aplicable, los espectros de respuesta en X y en Y deben tener la misma forma, lo cual supone que el cociente de la aceleración obtenida en ambos espectros, para un mismo valor de T, permanece constante para todo valor de T [130]. A dicho cociente γ se le denomina *ratio de intensidades espectrales*, siendo $0 \leq \gamma \leq 1$. Partiendo de las consideraciones citadas, la respuesta crítica se obtiene como:

$$R = \sqrt{(1+\gamma^2)\,\frac{R_x^2 + R_y^2}{2} + (1-\gamma^2)\sqrt{\left(\frac{R_x^2 - R_y^2}{2}\right)^2 + R_{xy}^2} + R_z^2} \tag{4.134}$$

  siendo R_{xy} la correlación de las respuestas modales en R_x y en R_y:

$$R_{xy} = \sum_{i=1}^{q}\sum_{j=1}^{q} \varepsilon_{ij} R_{xi} R_{yj} \tag{4.135}$$

  donde R_{xi} es la aportación del modo i a la respuesta máxima R_x, R_{yj} es la aportación del modo j a la respuesta máxima R_y y donde ε_{ij} representa el coeficiente de correlación modal entre los modos de vibración i y j. Hernández y López ampliaron los trabajos anteriores, considerando la incidencia arbitraria de las tres componentes sísmicas y dando lugar al método

llamado GCGC3 (Combinación Cuadrática Completa de 3 Componentes Generalizada) [131] o, en colaboración con Chopra, adaptando el método CQC3 al caso en el cual el ratio γ de intensidad espectral entre los espectros horizontales no sea constante para cualquier valor de T [132].

De manera alternativa, en esta tesis doctoral hemos optado porque el análisis modal espectral bajo la acción de un sismo con tres componentes direccionales no suponga realizar tres análisis independientes, cuyos resultados habrá que combinar con alguno de los métodos citados, sino que se realice un único análisis espectral considerando la aportación conjunta de las tres componentes sísmicas. Para ello, se empleará un vector unitario $a = [a_x, a_y, a_z]^T$, al que llamaremos *factor de aceleración espectral*, obtenido a partir del vector director de 3 elementos que define la dirección de actuación del sismo sobre los ejes de la estructura.

Los elementos de este vector unitario serán además los que formarán parte del vector J de influencia, en lo que respecta a sus tres grados de libertad longitudinales, siendo el resto iguales a 0:

$$J = [a_x\, a_y\, a_z\, 0\, 0\, 0\, a_x\, a_y\, a_z\, 0\, 0\, 0 \ldots a_x\, a_y\, a_z\, 0\, 0\, 0\,]^T \tag{4.136}$$

el cual se empleará a su vez para obtener los factores de participación (4.119) de cada modo de vibración.

Se propone por tanto que los métodos de combinación direccional que acabamos de describir no se apliquen en la última fase del proceso, combinando los resultados máximos proporcionados por cada componente sísmica, sino a la hora de obtener un único valor de aceleración máxima obtenido como combinación de las aceleraciones proporcionadas por los espectros de aceleración relativos a cada eje. Dicho valor combinado de aceleración espectral máxima Sa_i será el que utilicemos en la expresión (4.122) para obtener los desplazamientos d_i, las velocidades v_i y las aceleraciones máximas a_i en los nudos de la estructura, para cada modo de vibración i.

En ese sentido, sea T_i el periodo correspondiente al modo de vibración i de la estructura y sean Sa_{ix}, Sa_{iy} y Sa_{iz} las aceleraciones espectrales máximas proporcionadas por el espectro de respuesta de cada eje, obtenidas cada una de ellas

tal y como vimos en la sección 4.10.2. Si dichas tres aceleraciones son iguales, el valor de la aceleración máxima Sa_i a emplear coincidirá con el de cualquiera de ellas. Si no es así, el valor de la aceleración espectral Sa_i se obtendrá empleando alguna de las técnicas de combinación que se describen a continuación:

- Raíz Cuadrada de la Suma de los Cuadrados (SRSS):

$$Sa_i = \sqrt{Sa_{ix}^2 + Sa_{iy}^2 + Sa_{iz}^2} \tag{4.137}$$

- Combinación Cuadrática Completa de 3 Componentes (CQC3):

$$Sa_i = \sqrt{(1+\gamma^2)\,\frac{Sa_{ix}^2 + Sa_{iy}^2}{2} + (1-\gamma^2)\sqrt{\left(\frac{Sa_{ix}^2 - Sa_{iy}^2}{2}\right)^2 + Sa_{ixy}^2} + Sa_{iz}^2} \tag{4.138}$$

  siendo γ el coeficiente entre la menor y la mayor de las aceleraciones Sa_{ix} y Sa_{iy}, tal que $0 \leq \gamma \leq 1$, y siendo Sa_{ixy} la correlación entre Sa_{ix} y en Sa_{iy}, entendida tras una interpretación propia como:

$$Sa_{ixy} = Sa_{ix} Sa_{iy} \tag{4.139}$$

- Combinación Ponderada de las Aceleraciones Espectrales (CPAE), empleando las componentes a_x, a_y y a_z del vector unitario de factores de aceleración espectral como términos de ponderación:

$$Sa_i = a_x^2 Sa_{ix} + a_y^2 Sa_{iy} + a_z^2 Sa_{iz} \tag{4.140}$$

4.10.4. Técnicas de combinación de los resultados modales

Sea R la variable cuyo valor máximo se quiere determinar y sea R_i su valor máximo en el modo i. Dado que, para cada grado de libertad, el máximo de cada modo no se produce simultáneamente, R no se podrá calcular como suma de los máximos de cada modo. Es por ello que se han propuesto distintos métodos para calcular la respuesta máxima R a partir de los valores de los q modos de vibración considerados en el sistema [133].

Los resultados que se calcularán para cada modo de vibración y que se combinarán mediante los métodos de combinación modal que aquí se describen, para dar lugar a la respuesta del análisis espectral son: desplazamientos, velocidades y aceleraciones en los nudos, esfuerzos en los extremos de las barras, reacciones en apoyos y, por último, deformaciones unitarias, tensiones y esfuerzos en nudos de elementos finitos. No se combinarán en cambio los esfuerzos y de las deformaciones en los puntos intermedios de las barras, sino que dichos valores se calcularán, posteriormente, a partir de los resultados ya combinados de desplazamientos en los nudos y los esfuerzos en los extremos de las barras. A continuación se detallan los diferentes métodos de combinación modal implementados en este trabajo:

- Suma Absoluta: Proporciona el límite superior de la respuesta combinada, obtenida como la suma, en valor absoluto, de los valores modales:

$$R = \sum_{i=1}^{q} |R_i| \tag{4.141}$$

  El método supone que las respuestas modales máximas se producen al mismo tiempo para todos los modos.

- Raíz Cuadrada de la Suma de los Cuadrados (SRSS): Publicada por Goodman, Rosenblueth y Newmark [126], se trata de uno de los métodos más simples y a la vez más usados. Estima que la respuesta máxima se obtiene como la raíz cuadrada de la suma de los valores modales al cuadrado:

$$R = \sqrt{\sum_{i=1}^{q} R_i^2} \tag{4.142}$$

  Esta técnica proporciona resultados razonablemente válidos siempre y cuando no se produzca un acoplamiento entre los q modos de vibración considerados, lo cual equivale a decir que sus frecuencias estén suficientemente distanciadas. Por el contrario, cuando la estructura presenta frecuencias propias relativamente cercanas, de manera que la diferencia entre dos frecuencias consecutivas sea inferior al 10 % de la menor de las dos, el método SRSS puede proporcionar errores considerables. Esta técnica de combinación puede también dar lugar a errores sustanciales cuando el análisis sea

tridimensional y el efecto de la torsión sea significativo.

Hay que decir que cuando dos modos están en fase (lo que equivale a decir que están correlacionados o que puede existir un acoplamiento entre ellos), los valores máximos de cada modo individual ocurren simultáneamente, dando lugar a un efecto combinado que se obtiene sumando algebraicamente sus respuestas modales individuales.

Conviene en esos casos emplear otros métodos alternativos que tengan en cuenta la cercanía de las frecuencias naturales, como son los métodos de Rosenblueth, Suma Doble, Combinación Cuadrática Completa o Gupta y Cordero que veremos a continuación.

- Media de los métodos SUM y SRSS: Publicada por primera vez en [134] y propuesto su uso por la Comisión Federal de Electricidad (CFE) de México en [135], ofrece como resultado el valor medio de aplicar los métodos de la Suma Absoluta y SRSS, tendiendo por tanto del lado de la seguridad en sus resultados:

$$R = \frac{\sum_{i=1}^{q} |R_i| + \sqrt{\sum_{i=1}^{q} R_i^2}}{2} \tag{4.143}$$

- Suma del Laboratorio de Investigación Naval (NRL): Esta combinación suma la respuesta modal máxima en valor absoluto al resultado de aplicar el método SRSS al resto de respuestas modales. Esta regla se atribuye al Laboratorio de Investigación Naval de EEUU [136] y se ha utilizado en estudios de respuesta de estructuras submarinas bajo acciones sísmicas:

$$R = |R_j| + \sqrt{\sum_{\substack{i=1 \\ i \neq j}}^{q} R_i^2} \tag{4.144}$$

siendo $|R_j|$ el valor absoluto de la respuesta modal más desfavorable:

$$|R_j| = max\,|R_i|\,, \quad i = 1, 2, ..., q \tag{4.145}$$

- Media de los métodos SRSS y NRL: Calcula el valor medio de los resultados

que se obtendrían tras aplicar las técnicas SRSS y NRL [137]:

$$|R_j| = max\,|R_i|\,, \quad i = 1, 2, ..., q$$

$$R = \frac{\sqrt{\sum_{i=1}^{q} R_i^2} + |R_j| + \sqrt{\sum_{\substack{i=1 \\ i \neq j}}^{q} R_i^2}}{2} \tag{4.146}$$

- Rosenblueth: Debido a los inconvenientes que presenta el método SRSS si las frecuencias naturales no están suficientemente distanciadas, Rosenblueth y Elorduy [138] propusieron este método, el cual calcula la respuesta máxima de la estructura de manera más precisa que la técnica SRSS cuando las frecuencias de vibración están cercanas. Los autores asemejaron el movimiento del terreno a un ruido blanco de duración finita, asumiendo una respuesta periódica y amortiguada al mismo. Con ello, la respuesta máxima del sistema se obtiene del siguiente modo:

$$R = \sqrt{\sum_{i=1}^{q} \sum_{j=1}^{q} \varepsilon_{ij} R_i R_j} \tag{4.147}$$

siendo ε_{ij} el factor de correlación entre los modos i y j:

$$\varepsilon_{ij} = \frac{1}{1 + \left(\frac{\omega_i' - \omega_j'}{\xi_i' \omega_i + \xi_j' \omega_j}\right)^2} = \frac{1}{1 + \left(\frac{\omega_i' - \omega_j'}{\xi_i \omega_i + \xi_j \omega_j + \frac{4}{s}}\right)^2}, \quad i, j = 1, 2, ..., q \tag{4.148}$$

donde ω_i' y ω_j' son las frecuencias de vibración amortiguadas, obtenidas a partir de las frecuencias de vibración ω_i y ω_j de los modos i y j, y de sus factores de amortiguamiento crítico ξ_i y ξ_j, expresados en tanto por uno:

$$\omega_k' = \omega_k \sqrt{1 - \xi_k^2}, \quad k = 1, 2, ..., q \tag{4.149}$$

y donde ξ_i' y ξ_j' son los factores de amortiguamiento crítico equivalentes teniendo en cuenta la reducción en la respuesta debido a la duración finita del ruido blanco, dependientes del parámetro s que representa la duración

efectiva de dicho ruido (empleando por defecto un valor igual a 10 segundos):

$$\xi'_k = \xi_k + \frac{2}{\omega_k s}, \quad k = 1, 2, ..., q \tag{4.150}$$

Dicho coeficiente de correlación ε_{ij} es simétrico, es decir $\varepsilon_{ij} \equiv \varepsilon_{ji}$, cumpliéndose además que $\varepsilon_{ij} = 1$ si $i = j$. Teniendo en cuenta dichas propiedades, podemos reducir el número de sumandos y, por tanto, el de operaciones aritméticas al calcular la respuesta máxima del sistema:

$$R = \sqrt{\sum_{i=1}^{q} R_i^2 + 2 \sum_{i=1}^{q} \sum_{j=1}^{i-1} \varepsilon_{ij} R_i R_j} \tag{4.151}$$

- Suma Doble: Se ideó por parte de la Comisión Nuclear Regulatoria (CNR) de EEUU con el objetivo de mejorar al método SRSS cuando los modos de vibración están cercanos [139]. Es una versión más conservadora que el método de Rosenblueth, al emplear el valor absoluto de la respuesta máxima de cada modo:

$$R = \sqrt{\sum_{i=1}^{q} R_i^2 + 2 \sum_{i=1}^{q} \sum_{j=1}^{i-1} \varepsilon_{ij} |R_i| |R_j|} \tag{4.152}$$

El coeficiente de correlación ε_{ij} y su resto de parámetros asociados se obtienen de manera idéntica a los del método de Rosenblueth. Con posterioridad, Gupta [133] propuso la siguiente modificación en el cálculo de los coeficientes de correlación, teniendo en cuenta la posibilidad de que proporcionaran un valor sobreestimado cuando los factores de amortiguamiento de los modos fueran muy diferentes:

$$\varepsilon_{ij} = \frac{2\sqrt{\xi_i \xi_j}}{\xi_i + \xi_j} \frac{1}{1 + \left(\frac{\omega'_i - \omega'_j}{\xi_i \omega_i + \xi_j \omega_j + \frac{4}{td}} \right)^2}, \quad i, j = 1, 2, ..., q \tag{4.153}$$

- Gupta y Cordero: Con el objetivo de evitar la estimación del valor del parámetro s en el método de Rosenblueth, equivalente a la duración efectiva del segmento de ruido blanco, Gupta y Cordero [140] modificaron del siguiente

modo el cálculo del coeficiente de correlación:

$$\varepsilon_{ij} = \frac{1}{1 + \left(\frac{\omega_i' - \omega_j'}{\xi_i\omega_i + \xi_j\omega_j + c_{ij}}\right)^2}, \quad i,j = 1,2,...,q \tag{4.154}$$

sugiriendo que el término c_{ij} se calcule como:

$$c_{ij} = (0,16 - 0,5\xi_{ij})\left(1,4 - \left|\omega_i^2 - \omega_j^2\right|\right), \quad \xi_{ij} = \frac{\xi_i + \xi_j}{2}, \quad i,j = 1,2,...,q \tag{4.155}$$

El modo de calcular la respuesta máxima del sistema R permanece inalterado frente a lo que hemos visto en el método de Rosenblueth. En este caso, Gupta [133] también propuso un cambio en el cálculo de los coeficientes de correlación a fin de mejorar sus resultados cuando los factores de amortiguamiento de los modos de vibración sean muy dispares:

$$\varepsilon_{ij} = \frac{2\sqrt{\xi_i\xi_j}}{\xi_i + \xi_j} \frac{1}{1 + \left(\frac{\omega_i' - \omega_j'}{\xi_i\omega_i + \xi_j\omega_j + c_{ij}}\right)^2}, \quad i,j = 1,2,...,q \tag{4.156}$$

- Combinación Cuadrática Completa (CQC): Wilson, Der Kiurehian y Bayo propusieron el que, a día de hoy, se ha convertido en uno de los métodos de combinación más utilizados [141, 142]. Continua en la línea de los métodos de Rosenblueth, Suma Doble, y Gupta y Cordero, basado en la aplicación de la teoría de vibración aleatoria, pero utiliza un ruido blanco de duración infinita para representar la carga sísmica. La respuesta máxima del sistema se obtiene como:

$$R = \sqrt{\sum_{i=1}^{q} R_i^2 + 2\sum_{i=1}^{q}\sum_{j=1}^{i-1} \varepsilon_{ij} R_i R_j} \tag{4.157}$$

donde el coeficiente modal de correlación ε_{ij} de los modos i y j se calcula en este caso mediante la siguiente expresión, lo cual mejora los resultados de los métodos citados:

$$\varepsilon_{ij} = \frac{8\sqrt{\xi_i\xi_j}(\xi_i + \lambda_{ij}\xi_j)\sqrt{\lambda_{ij}^3}}{(1 - \lambda_{ij}^2)^2 + 4\xi_i\xi_j\lambda_{ij}(1 + \lambda_{ij}^2) + 4(\xi_i^2 + \xi_j^2)\lambda_{ij}^2}, \quad i,j = 1,2,...,q \tag{4.158}$$

siendo $\lambda_{ij} = \omega_j/\omega_i$ la relación entre las frecuencias naturales de los modos j e i, y siendo ξ_i y ξ_j los factores de amortiguamiento crítico correspondientes a dichos modos, expresados en tanto por uno. Si el factor de amortiguamiento crítico ξ es idéntico para los distintos modos de vibración, ocurre que:

$$\varepsilon_{ij} = \frac{8\xi^2(1+\lambda_{ij})\sqrt{\lambda_{ij}^3}}{(1-\lambda_{ij}^2)^2 + 4\xi^2\lambda_{ij}(1+\lambda_{ij})^2}, \qquad i,j = 1,2,...,q \tag{4.159}$$

En la práctica, el método CQC proporciona resultados similares al método SRSS siempre y cuando las frecuencias naturales estén bien distanciadas.

- Agrupamiento (Grouping): Este método [139], muy usado en la industria de la energía nuclear y desarrollado por la NRC (U.S. Nuclear Regulatory Commission), divide las frecuencias de vibración, ordenadas de menor a mayor, en G grupos, donde la diferencia entre la frecuencia menor de un grupo y la mayor no supere el 10 %. Cada frecuencia de vibración sólo puede pertenecer a un grupo. Dentro de cada uno de ellos, sus respuestas se suman en valor absoluto (lo que supone la principal crítica por parte de sus detractores), las cuales se combinan posteriormente con las del resto de grupos de acuerdo al método SRSS. Matemáticamente, la expresión que recoge la respuesta máxima del sistema es la siguiente:

$$R = \sqrt{\sum_{i=1}^{q} R_i^2 + \sum_{g=1}^{G} \sum_{i=ni(g)}^{nf(g)} \sum_{j=ni(g)}^{nf(g)} |R_{ig}R_{jg}|}, \ i \neq j \tag{4.160}$$

siendo *ni(g)* y *nf(g)* el índice de la frecuencia inicial y final de cada grupo, y donde R_{ig} o R_{jg} representan la respuesta máxima del modo i o j del grupo g.

- Método del 10 %: Desarrollado por la NRC [139], considera que dos modos de vibración están correlacionados si su frecuencia natural de vibración no difiere en más de un 10 %. Mediante este método, la respuesta máxima se obtiene como:

$$R = \sqrt{\sum_{i=1}^{q} R_i^2 + 2\sum_{i=1}^{q}\sum_{j=1}^{i-1} \varepsilon_{ij}\,|R_iR_j|} \tag{4.161}$$

En este caso, el coeficiente de correlación valdrá 1 o 0 dependiendo de las siguientes condiciones que determinan si dos frecuencias están o no suficientemente próximas:

$$\varepsilon_{ij} = \begin{cases} 1 & \text{si } \frac{\omega_j - \omega_i}{\min(\omega_i,\ \omega_j)} \leq 0,1 \\ 0 & \text{si } \frac{\omega_j - \omega_i}{\min(\omega_i,\ \omega_j)} > 0,1 \end{cases} \qquad i,j = 1,2,...,q \tag{4.162}$$

Este método proporciona unos resultados superiores a los del método anterior.

- Combinación Modal de Gupta (GMC): Todos los métodos utilizados hasta el momento para combinar la respuesta individual de los modos proporcionan sólo la respuesta máxima aproximada. Si en el análisis de cualquier estructura debe proporcionase la respuesta más fiable posible, en el caso de estructuras de especial interés, como pueden ser las centrales nucleares, dicha exigencia es máxima [143]. La respuesta de este tipo de sistemas está compuesta por dos partes: la respuesta periódica o amortiguada y la respuesta rígida [144]. En la teoría de vibraciones, a dichas partes se les denomina fase transitoria y fase estable, respectivamente. Se entiende que todos los métodos vistos hasta el momento han obtenido, únicamente, la respuesta modal periódica o amortiguada de la estructura.

  Es bien conocido que a altas frecuencias, la aceleración espectral llega a ser igual a la aceleración máxima del terreno. Teóricamente, la frecuencia más alta valdría infinito, a la cual le correspondería un periodo igual a 0. De este modo, la aceleración espectral de un oscilador de periodo 0 es igual a la aceleración máxima del terreno, a la cual se le denomina *aceleración de periodo cero* (ZPA). La frecuencia mínima a la cual la aceleración espectral llega a ser aproximadamente igual a la ZPA y permanece igual a ella se le denomina frecuencia ZPA o frecuencia rígida (f_r), expresada en hercios.

  Sean Sa^{max} y Sv^{max} la aceleración y la velocidad espectral máxima, respectivamente del espectro utilizado. A partir de dichos valores, y de la frecuencia rígida f_r, obtenemos las siguientes frecuencias f_1 y f_2, ambas expresadas en hercios:

$$f_1 = \frac{Sa^{max}}{2\pi Sv^{max}}, \quad f_2 = \frac{f_1 + 2f_r}{3} \tag{4.163}$$

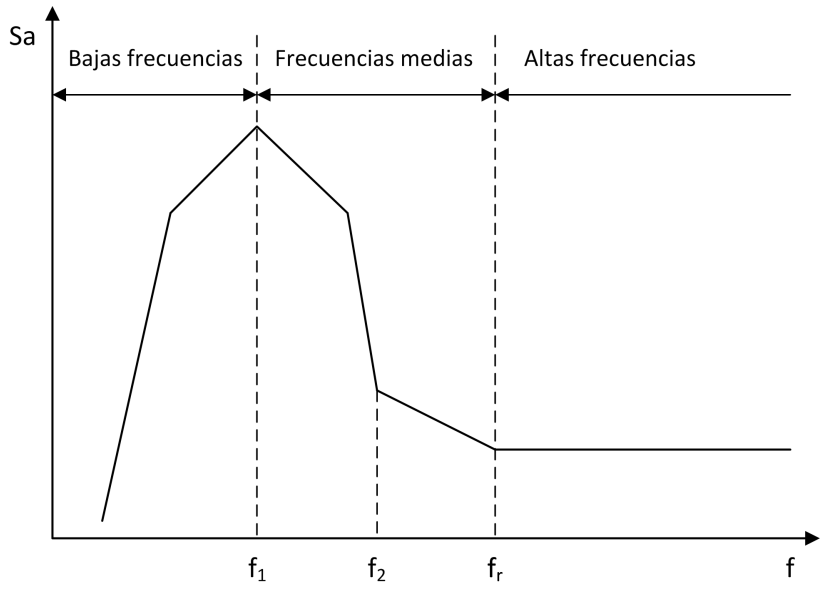


Figura 4.6: Rango de frecuencias bajas, medias y altas en un espectro de respuesta.

Conocidas dichas frecuencias, y tal y como se muestra en la figura 4.6, podemos dividir en tres al conjunto de frecuencias del espectro de respuesta utilizado. Por debajo de la frecuencia f_1, la respuesta será amortiguada y los modos no están en fase, entre sí. Por encima de f_2, la respuesta será rígida y las respuestas modales máximas están en fase, o perfectamente acopladas, entre sí, además de estar en fase con la acción sísmica de entrada. Entre f_1 y f_2, la respuesta será amortiguada y rígida, tratándose de un intervalo de transición.

Adicionalmente, calculamos para cada modo su *coeficiente de respuesta rígida* α_i, el cual expresa la correlación entre la respuesta modal y la aceleración sísmica de entrada. Recordemos que $f_i = \frac{\omega_i}{2\pi}$:

$$\alpha_i = \begin{cases} 0 & \text{si } f_i < f_1 \\ \frac{\ln(f_i/f_1)}{\ln(f_2/f_1)} & \text{si } f_1 \leq f_i \leq f_2 \\ 1 & \text{si } f_i \geq f_2 \end{cases} \qquad i = 1, 2, ..., q \tag{4.164}$$

Basándonos en lo que acabamos de mencionar, podemos decir que toda respuesta modal R_i de un modo de vibración i se divide en su parte periódica amortiguada R_i^p y en su parte rígida R_i^r. En lo que respecta a esta última,

su valor dependerá del coeficiente de respuesta rígida del modo:

$$R_i^r = \alpha_i R_i \tag{4.165}$$

Si asumimos que la parte rígida y la parte periódica amortiguada son estadísticamente independientes tendremos que:

$$R_i^p = \sqrt{\left(1 - \alpha_i^2\right) R_i} \tag{4.166}$$

$$R^i = \sqrt{\left(R_i^p\right)^2 + \left(R_i^r\right)^2} \tag{4.167}$$

Puesto que las partes rígidas están perfectamente correlacionadas entre sí, ocurre que la suma algebraica de todas ellas da lugar a la respuesta rígida del sistema:

$$R^r = \sum_{i=1}^{q} R_i^r \tag{4.168}$$

En lo que respecta a las partes periódicas de cada modo de vibración, todas ellas se deben combinar para obtener la respuesta periódica amortiguada del sistema R^p mediante alguno de los métodos que hemos visto con anterioridad (Rosenblueth, CQC o Gupta y Cordero, por ejemplo).

Con todo ello, finalmente, y aplicando la técnica SRSS, la respuesta del sistema es:

$$R = \sqrt{\left(R^p\right)^2 + \left(R^r\right)^2} \tag{4.169}$$

De manera alternativa, la formulación anterior puede agruparse en la siguiente expresión, a fin de calcular la respuesta máxima de la estructura [133]:

$$R = \sqrt{\sum_{i=1}^{q} R_i^2 + 2 \sum_{i=1}^{q} \sum_{j=1}^{i-1} \bar{\varepsilon}_{ij} R_i R_j} \tag{4.170}$$

en la cual, el coeficiente de correlación modificado $\bar{\varepsilon}_{ij}$ recoge el efecto de la respuesta rígida, obtenido como:

$$\bar{\varepsilon}_{ij} = \alpha_i \alpha_j + \varepsilon_{ij} \sqrt{\left(1 - \alpha_i^2\right)\left(1 - \alpha_j^2\right)} \tag{4.171}$$

siendo ε_{ij} el coeficiente de correlación calculado mediante los métodos de

Rosenblueth, CQC o Gupta y Cordero.

- Respuesta Rígida Residual (RRR): En ocasiones, las frecuencias altas de vibración tienen una importante contribución a la respuesta del sistema. Es el caso por ejemplo de una central nuclear donde, a diferencia de lo que ocurre en un edificio, los modos de vibración con frecuencias mayores que f_r resultan ser de vital importancia. Si estos modos se ignoraran, en dichas circunstancias, los errores cometidos en la respuesta calculada serían inaceptablemente elevados. Ello supone la necesidad de desarrollar métodos de combinación que incluyan el efecto de los modos más altos, pero sin incurrir en el considerable coste computational que requeriría calcularlos.

  Teniendo todo ello en cuenta, y de manera alternativa a (4.165), la respuesta rígida del sistema la obtendremos de acuerdo a la siguiente expresión, donde q representa ahora el número de modos cuya frecuencia de vibración f_i es inferior a la frecuencia rígida f_r, descrita en el método GMC:

$$R^r = R_0 + \sum_{i=1}^{q} R_i^r \tag{4.172}$$

  Eso supone, en consecuencia, que el término R_0 es el encargado de recoger la respuesta conjunta de los modos de vibración con una frecuencia f_i mayor o igual a f_r. Son distintos los procedimientos disponibles para obtener R_0, sin tener que calcular previamente el valor de dichas frecuencias. Uno de ellas es el llamado *Método de las Masas Ausentes* [133, 143], el cual obtiene el efecto de las masas del sistema (*masas ausentes*) que no están incluidas en los modos con frecuencias inferiores a f_r. La respuesta del sistema ante dichas masas ausentes se obtiene mediante un análisis estático, en el cual se aplica una carga equivalente al producto de las masas ausentes por la aceleración espectral correspondiente a la frecuencia f_r.

  Sea $d \in \mathbb{R}^{nx1}$ el vector con la fracción de masa incluida en la respuesta del sistema obtenida como aportación conjunta de los q modos con una frecuencia natural $f_i < f_r$, a partir de sus factores de participación Γ_i y sus modos de vibración ϕ_i:

$$d = \sum_{i=1}^{q} \Gamma_i \phi_i \tag{4.173}$$

Sea J el vector de influencia con los factores de aceleración espectral recogidos en (4.136). A partir de dichos vectores d y J, podremos calcular el vector e referente a la fracción de masa no incluida en la respuesta modal:

$$e = d - J \tag{4.174}$$

Para obtener la respuesta conjunta de todos los modos con altas frecuencias, tendremos que resolver el siguiente sistema de ecuaciones:

$$KR_0 = P \tag{4.175}$$

donde el vector P parte derecha se obtiene como:

$$P = Sa_r Me \tag{4.176}$$

siendo K y M las matrices de rigidez y masa de la estructura y Sa_r la aceleración espectral correspondiente a la frecuencia f_r.

El inconveniente que se nos presenta es que, si los espectros de respuesta considerados para cada dirección no son coincidentes, podremos tener valores de aceleración espectral Sa_{rx}, Sa_{ry} y Sa_{rz} diferentes, uno de ellos por cada eje. Para solucionarlo, dispondremos de dos alternativas. En la primera de ellas, calcularemos la respuesta individual en cada dirección y las combinaremos posteriormente con alguna técnica de combinación direccional (SRSS, Regla del 30 % o CQC3) descritas en la sección 4.10.3, lo cual supone que el sistema de ecuaciones (4.175) tendrá tres vectores partes derecha, esto es, $P \in \mathbb{R}^{nx3}$. Como segunda alternativa, podemos emplear una única aceleración espectral Sa_r, obtenida como combinación de las aceleraciones espectrales Sa_{rx}, Sa_{ry} y Sa_{rz} mediante los métodos SRSS, CQC3 o CPAE vistos también en la sección 4.10.3. Esta segunda variante conlleva que el sistema (4.175) tendrá un único vector parte derecha, es decir, $P \in \mathbb{R}^{nx1}$. Conviene aclarar que la opción elegida por nuestra parte ha sido esta segunda, siendo consecuentes de acuerdo a lo descrito en el apartado 4.10.3.

Una vez obtenido el valor de R_0, la respuesta rígida del sistema la obtendremos aplicando las expresiones de la (4.163) a la (4.169) recogidas en el

método GMC, sustituyendo la expresión (4.168) por la (4.172). Alternativamente, y de acuerdo a (4.170), podemos expresar la respuesta del sistema como:

$$R = \sqrt{\left(\sum_{i=1}^{q} R_i + R_0\right)^2 + 2\sum_{i=1}^{q}\sum_{j=1}^{i-1} \bar{\varepsilon}_{ij} R_i R_j} \tag{4.177}$$

siendo $\bar{\varepsilon}_{ij}$ el coeficiente calculado según (4.171).

4.10.5. Técnicas de obtención del signo de los resultados

Llamemos R a la variable (desplazamientos en los nudos, reacciones en apoyos, tensiones, etc.) cuyo máximo, en valor absoluto, hemos determinado mediante cualquiera de los métodos que hemos descrito en la sección 4.10.4, a partir de las respuestas máximas R_i obtenidas para cada modo i, de los q modos considerados. Si bien hay casos en los cuales dicha información, en forma de valor absoluto, pueda ser suficiente, puesto que únicamente se pretenda conocer el valor de la magnitud de la variable y su signo no sea relevante, en otros casos no será así, como puede ocurrir a la hora de calcular el esfuerzo axil de un pilar, cuyo signo determina si el mismo está trabajando a tracción o a compresión y su desconocimiento genera problemas en el dimensionamiento y en el diseño de la estructura.

Algo similar ocurre si queremos mostrar por pantalla la deformada de una estructura de barras, como observamos en la figura 3.27, correspondiente a un análisis modal espectral, para lo cual resulta imprescindible calcular satisfactoriamente el signo de los movimientos máximos en los nudos extremos de las barras, antes de obtener la deformación de los distintos puntos intermedios de las mismas.

En un intento por determinar el signo de los resultados de una variable R obtenida tras un análisis modal espectral, se han implementado, en esta tesis doctoral, las siguientes alternativas:

- Dado un grado de libertad concreto de la respuesta R obtenida, elevamos al cuadrado y sumamos los valores de las respuestas de cada modo de vibración que sean positivas, para dicho grado de libertad. Del mismo modo, elevamos

al cuadrado y sumamos los valores de las respuestas modales que, para el citado grado de libertad, sean negativas. A continuación, comparamos los resultados. Si la suma de los valores positivos al cuadrado es mayor que la suma de los cuadrados de los valores negativos, el signo del resultado será positivo. En caso contrario, será negativo.

- Que el signo de la respuesta combinada R, para cada grado de libertad concreto, sea el correspondiente al de la respuesta máxima del modo de vibración principal p de la estructura. Dicho modo principal vendrá dado por aquel que pueda decidir el usuario o se deberá obtener automáticamente en base a un criterio determinado. Por nuestra parte, el criterio escogido supone obtener, para cada modo i de vibración, su porcentaje o fracción de masa efectiva total, calculado en función de su factor de participación Γ_i o, lo que es lo mismo, en función de la matriz de masa M de la estructura, del propio modo de vibración ϕ_i y del vector J, formado por los factores de aceleración espectral, de acuerdo a la expresión (4.136):

$$\%\mathrm{Meft_i} = 100\ \frac{\left(\phi_\mathrm{i}^\mathrm{T}\mathrm{MJ}\right)^2}{\mathrm{J^T MJ}} = 100\frac{\Gamma_\mathrm{i}^2}{\mathrm{J^T MJ}},\quad \mathrm{i} = 1, 2, ..., \mathrm{q} \tag{4.178}$$

  Aquel modo que posea el mayor porcentaje de masa efectiva total será el que se convierta en el modo de vibración principal p de la estructura.

  En caso de que el grado de libertad del modo principal valga 0, aplicaremos la alternativa anterior para decidir el signo del resultado.

- Método de Cominetti y Jorquera: En el año 2005, Cominetti y Jorquera [145] propusieron la siguiente metodología para obtener el signo de los resultados. Basándose en principio en el método CQC, utilizan los porcentajes de participación de cada modo con el objetivo de asignar un peso al signo de cada una de las respuestas modales, en forma proporcional al porcentaje de participación de dicho modo. Se incluye a continuación dicha propuesta de acuerdo a una interpretación por nuestra parte de determinados aspectos de la misma:

  1. Calculamos el cuadrado de la respuesta máxima, de igual modo a como vimos en el método de la Suma Doble, con lo cual sumamos las

respuestas aportantes al signo en valor absoluto:

$$R^2 = \sum_{i=1}^{q} R_i^2 + 2\sum_{i=1}^{q}\sum_{j=1}^{i-1} \varepsilon_{ij}|R_i||R_j| \tag{4.179}$$

donde el coeficiente de correlación ε_{ij} podrá determinarse mediante los métodos de Rosenblueth, CQC o Gupta y Cordero.

2. Obtenemos el factor de participación del modo i en la dirección del análisis, lo cual implica que el vector J se obtenga según la expresión (4.136):

$$\Gamma_i = \phi_i^T M J, \quad i = 1, 2, ..., q \tag{4.180}$$

3. Determinamos el vector $A_{ij} \in \mathbb{R}^{nx1}$ como el aporte al signo de un modo i al combinarse con un modo j, siendo n el número de grados de libertad considerados. Para determinarlo, la respuesta del sistema al cuadrado dividirá al producto del módulo de las respuestas combinadas por medio del coeficiente de correlación ε_{ij} por el signo de la respuesta R_i y por el factor de participación del modo i al cuadrado:

$$A_{ij} = 100\Gamma_i^2 \frac{|\varepsilon_{ij} R_i R_j|}{R^2} sign\,(R_i)\,, \quad i, j = 1, 2, ..., q \tag{4.181}$$

siendo $sign\,(R_i)$ una función que proporcionará un vector, de n componentes, con valores iguales a 1 o -1, dependiendo del signo de cada componente k del vector R_i.

$$sign(R_i)(k) = \begin{cases} 1 \; si\, R_i(k) \geq 0 \\ -1 \; si\, R_i(k) < 0 \end{cases}, \quad k = 1, 2, ..., n \tag{4.182}$$

Cada uno de los productos y cocientes vectoriales que aparecen en la expresión (4.181) se calcularán elemento a elemento.

4. Sumamos los aportes al signo en valor absoluto, generando un vector S de n componentes:

$$S(k) = \sum_{i=1}^{q}\sum_{j=1}^{q} |A_{ij}(k)|, \; k = 1, 2, ..., n \tag{4.183}$$

5. Sumamos los aportes positivos al vector P y los aportes negativos al vector N, ambos de n componentes:

$$\begin{aligned} P(k) &= \sum_{i=1}^{q} \sum_{j=1}^{q} |Aij\,(k) > 0|, \quad k = 1, 2, ..., n \\ N(k) &= \sum_{i=1}^{q} \sum_{j=1}^{q} |Aij\,(k) < 0|, \quad k = 1, 2, ..., n \end{aligned} \tag{4.184}$$

6. Determinamos los porcentajes de aportes positivos y negativos al signo:

$$\begin{aligned} \%P(k) &= 100\ \tfrac{P(k)}{S(k)}, \quad k = 1, 2, ..., n \\ \%N(k) &= 100\ \tfrac{N(k)}{S(k)}, \quad k = 1, 2, ..., n \end{aligned} \tag{4.185}$$

7. Generamos un vector llamado *Signo* de n elementos con el signo de cada grado de libertad del resultado, dependiendo de cual de los dos porcentajes es mayor. En consecuencia, almacenamos en la posición k de este vector un 1 indicando que el elemento k del vector R será positivo o un -1 si dicho valor será negativo:

$$Signo(k) = \begin{cases} 1 \text{ si } \%P(k) \geq \%N(k) \\ -1 \text{ si } \%P(k) < \%N(k)\ , \quad k = 1, 2, ..., n \end{cases} \tag{4.186}$$

Capítulo 5

La Computación Científica

Este capítulo está dedicado a las diferentes técnicas que permiten obtener el mejor rendimiento de los computadores actuales, lo cual implica que las aplicaciones de simulación de ciencia e ingeniería que rueden sobre ellos ofrezcan las mejores prestaciones, en tiempos de ejecución, requerimientos de memoria y precisión en los resultados, a los usuarios. Para ello, se describe en qué consiste la Computación de Altas Prestaciones, cuáles son los diferentes tipos de supercomputadores existentes y qué modelos de programación paralela pueden emplearse en cada uno de ellos. Finalmente, se expone el software numérico más avanzado que existe actualmente diseñado para desarrollar aplicaciones científicas, resolver problemas de álgebra lineal básica, sistemas de ecuaciones lineales, problemas de valores propios y particionado de mallas y grafos.

5.1. Introducción

La Computación Científica es una amplia disciplina focalizada en usar los computadores como herramientas del descubrimiento científico. Tradicionalmente, la Computación Científica ha jugado un papel muy importante en el avance de la informática y, en consecuencia, en numerosos campos de la ciencia y de la

ingeniería. En los comienzos de la informática, la Computación Científica era la principal fuerza impulsora, proporcionando los problemas que eran la motivación para el desarrollo tanto de software como de hardware.

Posteriormente los ordenadores se fueron aplicando a un número incontable de áreas. Surgió la Computación Paralela, técnica que permite desarrollar aplicaciones en las cuales varios elementos de proceso (procesadores o núcleos computacionales) trabajan de manera conjunta en la resolución de un mismo problema. Con ello, la Computación Científica se convirtió, allá por el principio de los años 90, en sinónimo de cálculo intensivo y de grandes supercomputadores, con un número muy limitado de usuarios que se ocupaban de problemas extremadamente técnicos y científicos.

No obstante, si bien es cierto que la Computación Científica ha mostrado su importancia tradicionalmente en aquellas áreas en las cuales las necesidades de velocidad y memoria son más acuciantes, también es cierto que está sufriendo un gran auge dada su aplicabilidad a arquitecturas de bajo coste, como son los ordenadores personales, los cuales ofrecen a un precio muy asequible unas excelentes prestaciones, al estar compuestos a día de hoy por múltiples núcleos de cómputo.

Así, en 1994, el CESDIS (Center of Excellence in Space Data and Information Sciences) conectó mediante una red Ethernet un conjunto de PCs, bajo el proyecto llamado Beowulf, como una alternativa de bajo coste a los caros supercomputadores. El proyecto fue un éxito y su idea se extendió rápidamente a la comunidad científica y académica internacional.

Posteriormente, los desarrollos y la disponibilidad de software de cálculo numérico de dominio público (BLAS, LAPACK, PETSc, etc.) capaz de resolver eficientemente los modelos matemáticos en los que se basa la Computación Científica, y el desarrollo de estándares de facto referentes a diferentes paradigmas de programación paralela como OpenMP [11] y MPI [12], hicieron posible el crear software portable a cualquier plataforma. Surge de este modo la Computación de Altas Prestaciones, la cual, englobando a la Computación Paralela y a todo un conjunto de librerías de cálculo numérico, intenta obtener las máximas prestaciones de un sistema de computación.

A medida que la comunidad investigadora ha confiado en la Computación

Científica y ésta ha ido madurando, han sido mayores los diferentes campos a los que se ha aplicado, aumentando en gran medida el tamaño y la complejidad de los problemas abordables.

Un último desarrollo que está teniendo un gran impacto en el mundo de la informática, la ciencia y la ingeniería es el de Computación Cloud, o computación en la nube, mayor inclusive que el que tuvo hace una década la Computación Grid. El objetivo final de ambas tecnologías es que desde cualquier punto de acceso a la red se puedan utilizar todo un conjunto de recursos computacionales de los cuales no se dispone, de una manera transparente e independientemente de las arquitecturas de la máquinas utilizadas y de la ubicación física de las mismas.

Para aprovechar todo este potencial computacional deben interaccionar todo un conjunto de recursos distintos y distantes, compuestos no sólo por el hardware sino además por toda una serie de capas software que van desde las herramientas de despliegue de máquinas virtuales a la planificación de tareas, los protocolos de comunicaciones y de seguridad, la propia aplicación desplegada y las dependencias que ella misma requiere para ponerse en marcha. Y todo esto realizado de manera transparente al usuario final, de modo que su utilización no sea más complicada que la de un recurso puramente local.

A modo de resumen podemos decir que la Computación Científica sigue siendo a día de hoy una fuerza importante en el desarrollo de la informática, poniendo especial énfasis en las siguientes áreas:

- Técnicas que permiten extraer el máximo rendimiento de un ordenador y de la aplicación que allí se ejecutará, desarrollando algoritmos que minimicen tanto el movimiento de datos entre la jerarquía de memorias como el número de operaciones aritméticas, optimizando los accesos a disco y empleando a la vez librerías numéricas avanzadas que recojan el estado del arte en el campo de los métodos numéricos.
- El desarrollo de aplicaciones basadas en técnicas de paralelización, capaces de emplear múltiples procesadores simultáneamente y sacar el mejor provecho de los múltiples núcleos de los que disponga la plataforma computacional, y en el uso de librerías numéricas paralelas, con el objetivo de resolver problemas diversos de ciencia e ingeniería en los que reduzcamos

los tiempos de computación y podamos abordar problemas más realistas y complejos.

- El diseño de software orientado a la visualización (preproceso y postproceso) de las enormes cantidades de datos habitualmente generadas por los programas de cálculo masivo, usuales en este campo.

- El desarrollo de componentes software que hagan posible el ejecutar tareas de manera sencilla y transparente en una infraestructura de cálculo distribuido compuesta por múltiples computadores geográficamente repartidos, así como el almacenar y gestionar un volumen ingente de datos entre los mismos.

- La creación de aplicaciones, basadas en la Computación en Grid y Cloud, y preferiblemente también en Computación de Altas Prestaciones, que nos permitan reducir en gran medida los tiempos de simulación en aquellos campos de ingeniería o la ciencia que impliquen la necesidad de ejecutar multitud de simulaciones paramétricas e independientes entre sí, así como todas aquellas en las cuales es necesario gestionar un gran volumen de datos, bien sea de entrada o de salida al problema.

- El desarrollo y despliegue de servicios Grid y Cloud que ofrezcan el uso robusto, seguro, sencillo y transparente de un conjunto de aplicaciones, basadas en una Arquitectura Orientada a Servicio y de forma deseable en Computación de Altas Prestaciones, en un amplio abanico de sectores científicos y empresariales, así como de los recursos computacionales que proporcionen la capacidad de cómputo y el espacio de almacenamiento de datos requerido por las mismas.

5.2. La Computación de Altas Prestaciones

El volumen de datos requerido y la complejidad computacional de problemas de ámbito científico en áreas tales como la biomedicina, la predicción del tiempo, la modelización del comportamiento de los océanos, del universo o del cambio climático en nuestro planeta, por citar algunos ejemplos, han ido creciendo cada

vez más. Históricamente, la potencia computacional de los computadores no ha estado en sintonía con la potencia demandada por numerosas aplicaciones, como las citadas, motivo por el cual los científicos e ingenieros resolvían problemas simplificados, que pudieran ser ejecutados en los ordenadores de la época.

Sin embargo, con el surgimiento de la Computación Paralela [146, 147, 148], un número determinado de computadores previamente interconectados, cada uno de ellos compuesto incluso por múltiples procesadores y núcleos computacionales, trabajan simultáneamente en la resolución de un mismo problema, el cual, por su complejidad subyacente, podría sobrepasar la capacidad de un único computador.

Ello, unido al hecho de que la potencia de los computadores ha ido aumentando vertiginosamente, ha supuesto que los ingenieros y científicos actualmente puedan, por un lado, abordar problemas mucho más realistas, obteniendo bajo unos tiempos de respuesta muy razonables unos resultados más precisos y fiables que los obtenidos hasta el momento y, por otro, resolviendo problemas que por su naturaleza parecían impensables hace tan sólo algunos años.

Hasta hace una década, el crecimiento exponencial en el rendimiento de los computadores venía dado por el incremento en la frecuencia de reloj, por el diseño de arquitecturas más complejas y por la incorporación creciente en el procesador de un número mayor de transistores, cada vez más rápidos y a la vez más pequeños y con un menor consumo energético. Esto conllevaba que la capacidad de cómputo de un computador se mejoraba sustancialmente cada 18 o 24 meses, duplicando el número de transistores presentes en la misma unidad de área, de acuerdo a la ley de Moore.

A la vez, se fueron incorporando otro tipo de tecnologías que trataban de incrementar la potencia de los procesadores, como el incremento de la longitud de palabra, la disponibilidad de múltiples unidades funcionales, la ejecución segmentada de las instrucciones y las técnicas *multihilo*, o multithreading. Dichas técnicas, permiten la ejecución de varios procesos ligeros denominados *hilos*, o threads, simultáneamente sobre un mismo procesador, lo cual equivale a disponer de varios procesadores lógicos, o virtuales, que ejecutan flujos de instrucciones diferentes y de manera intercalada sobre un mismo procesador físico, siempre con el objetivo de sacar el mejor provecho a la arquitectura disponible. Si distintos

hilos no necesitan utilizar un mismo recurso (CPU, memoria, disco, etc.) al mismo tiempo, se ejecutarán de manera concurrente. Por el contrario, y como es lógico, si varios hilos necesitan acceder a un recurso en común, uno de ellos lo hará y el resto se esperarán.

No obstante, y a pesar de dichas técnicas de mejora de las prestaciones, la imposibilidad de seguir reduciendo al mismo tiempo el tamaño de los transistores y el consumo energético de los mismos dio lugar a la necesidad de intentar proporcionar, de un modo alternativo, un mayor rendimiento sin elevar el consumo asociado. Surgieron entonces los chips que incorporaban en su interior múltiples elementos o *núcleos* de procesamiento, denominados *multicores* o *multinúcleos*, como una alternativa más eficiente a los procesadores tradicionales que integraban, hasta el momento, un único núcleo. La diferencia es clara: mientras que la mejora de las prestaciones de un procesador había venido dada por aumentar notablemente su número de transistores, con la dificultad, la elevada inversión y el consumo energético insostenible que ello supone, dicha mejora podría venir, sencillamente, por la replicación del diseño del mismo, aumentando su número de elementos de proceso. En adelante, cuando hablemos de procesador, nos referiremos al chip que incorpora en su interior uno o más núcleos de cómputo, cada uno de los cuales tiene la capacidad de ejecutar instrucciones de forma independiente y simultánea. Con todo ello, el sistema operativo trata a cada núcleo como si de un procesador independiente se tratara, con todos sus recursos asociados. A efectos prácticos, estamos hablando de un multiprocesador implementado en un sólo chip.

La figura 5.1 muestra un diagrama de bloques de un procesador multicore compuesto por dos núcleos, que no tiene por qué corresponderse con un ejemplo real. Como puede observarse, existen dos niveles de caché. La más cercana y privada a cada núcleo, de nivel 1, y otra compartida entre ambos de nivel 2.

Si cada uno de estos núcleos que forman parte de un procesador tiene a su vez la capacidad de ejecutar varios hilos simultáneamente, el sistema operativo o las propias aplicaciones pueden dividir su carga de trabajo en hilos cuya ejecución se puede planificar de forma independiente sobre los múltiples elementos de cómputo virtuales que están a su disposición. Esto es lo que se conoce como tecnología *hyperthreading*. Por ese motivo, un simple PC de sobremesa está compuesto a día

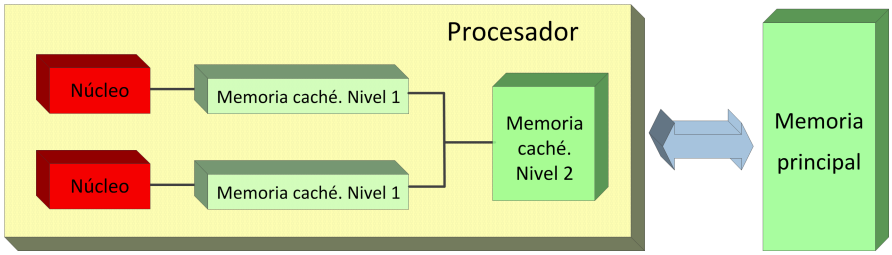


Figura 5.1: Diagrama de bloques de un procesador dual core genérico.

de hoy por múltiples núcleos de cómputo, cada uno de ellos multihilo, aumentando paulatinamente dicho número en un futuro inmediato.

Sin embargo, aprovechar todo el potencial computacional que tenemos disponible no es tarea fácil, puesto que requiere el desarrollo de aplicaciones paralelas eficientes que son difíciles de programar [149]. Esto ha supuesto que el ritmo en el desarrollo de aplicaciones comerciales paralelas sea muy lento y la disponibilidad de las mismas en el mercado profesional sea realmente baja. Tómese como ejemplo el caso que nos conlleva en esta tesis doctoral, relativo al cálculo de estructuras, y el casi nulo número de aplicaciones de simulación paralelas disponibles en el mercado.

Para sacar provecho de la Computación Paralela, una aplicación debe ser previamente *paralelizada*, lo cual supondrá que la ejecución de la misma ocurra en más de un *proceso* o *hilo de ejecución* (según el modelo de paralelismo escogido) donde, según veremos, cada uno de ellos representa a una simple instancia de la aplicación o a uno de los subprogramas de los que consta, ejecutándose autónomamente en un elemento de cómputo. De este modo, el objetivo primero de la Computación Paralela es incrementar las prestaciones computacionales de una aplicación paralelizada frente a su ejecución secuencial en un simple elemento de proceso, obteniendo por ejemplo la solución del problema en un tiempo inferior al emplear la potencia de cómputo de los diferentes elementos de cómputo partícipes en la resolución del mismo. A la vez, bajo un modelo de memoria distribuida, como veremos, se habilita la posibilidad de resolver problemas de una mayor dimensión, superando la limitación de la capacidad de memoria de un procesador, al tener a nuestra disposición la memoria de todos ellos.

Idealmente, si una aplicación paralelizada se ejecuta en p elementos de proce-

samiento, parece lógico pensar que será p veces más rápida que si se ejecuta en un solo elemento. Sin embargo, esto es extremadamente difícil de llevarlo a cabo en la práctica, debido a multiples razones, como son el hecho de que hay partes del código puramente secuenciales, los procesos y los hilos de ejecución necesitan un tiempo para ponerse en marcha, los elementos de proceso invierten tiempo en comunicarse datos entre ellos, la carga debe estar equilibradamente distribuida entre las tareas y el coste de los accesos a disco es elevado y, en la mayor parte de los casos, no se realiza en paralelo.

Uno de los requerimientos de la programación paralela consiste en descomponer, de una manera eficiente, una aplicación y los datos con los que trabaja en diferentes subproblemas, cada uno de ellos asignado a un elemento de procesamiento distinto. A este procedimiento se le denomina *descomposición del problema* y podemos distinguir entre un *paralelismo de datos* o un *paralelismo de tareas*. El primero consiste en la ejecución simultánea del mismo código en elementos de proceso diferentes sobre distintos datos, los cuales se reparten de manera equitativa entre los mismos. Cada elemento trabajará con la porción de datos asignados, aunque necesitará comunicarse periódicamente con otros para intercambiar información. Suele ser la aproximación más habitual y apropiada en aplicaciones de Computación Científica. El segundo caso consiste en dividir un problema, o aplicación, en un número de tareas más pequeñas, asignando cada una de ellas a un elemento de cómputo diferente. Suele ser menos frecuente, ya que no es fácil encontrar grandes fragmentos de código independientes entre sí en una misma aplicación. Tanto en un caso como en otro es fundamental repartir la carga de manera equitativa entre los elementos de procesamiento, en lo que se conoce como *equilibrado de la carga*, bien sea a nivel de datos o a nivel de tareas, asegurando que no existan elementos de proceso ociosos mientras otros se mantienen activamente trabajando.

Denominamos Computación de Altas Prestaciones a todas aquellas técnicas que intentan incrementar las prestaciones computacionales de un algoritmo o aplicación. En primer lugar, entre dichas técnicas encontramos aquellas que tienen como objetivo mejorar las prestaciones de una aplicación en su ejecución en un único elemento de procesamiento y que dan lugar al desarrollo de una aplicación secuencial óptima. A modo de ejemplo podemos citar el aprovechamiento de la

jerarquía de memorias y reducir al máximo el movimiento de datos entre las mismas, el empleo de estructuras de datos que minimicen el consumo de memoria, como puede ser el formato Compressed Sparse Row (CSR) a la hora de trabajar con matrices dispersas, la gestión eficiente del sistema de E/S, el uso de núcleos computacionales como BLAS o LAPACK y el de librerías numéricas secuenciales que recojan el estado del arte en el campo de los métodos numéricos a la hora de abordar problemas tales como la resolución de sistemas de ecuaciones lineales o no lineales, problemas de valores propios, problemas de optimización, ecuaciones diferenciales, etc.

En segundo lugar, bajo el nombre de Computación de Altas Prestaciones nos referimos a todas aquellas técnicas que permiten que una aplicación se ejecute en múltiples elementos de proceso, como es el caso de la Computación Paralela, así como al equilibrado óptimo de la carga y la minimización de las comunicaciones entre los mismos, empleando al mismo tiempo núcleos computacionales paralelos como PBLAS, BLACS, ScaLAPACK, etc. y las librerías numéricas avanzadas ya paralelizadas que estén a nuestra disposición.

En cuanto al aprovechamiento de la jerarquía de memorias, debemos emplear algoritmos que minimicen el movimiento de datos entre las diferentes memorias que componen un ordenador, de modo que los accesos a los datos o a las instrucciones se realicen, en mayor medida, en las componentes superiores de la jerarquía, lo que reducirá notablemente los tiempos de ejecución de dichos algoritmos. No olvidemos que, aproximadamente, el tiempo de ejecución de una instrucción en un ordenador es la suma del tiempo invertido por parte de la CPU en realizar las operaciones aritméticas más los tiempos de espera en el acceso a los datos presentes en memoria.

Entre las componentes de dicha jerarquía, como podemos observar en la figura 5.2, encontramos a la memoria secundaria, o de disco, a la memoria RAM, a la memoria caché, en sus múltiples niveles, y a los registros del procesador, yendo de mayor a menor capacidad de almacenamiento y de menor a mayor velocidad de acceso. A modo de ejemplo de algoritmos eficientes, en el aprovechamiento de las mejores prestaciones ofrecidas por la memoria caché frente a la memoria central, podemos citar a los denominados *algoritmos a bloques*, los cuales intentan realizar todas las operaciones oportunas con los datos en el momento en el que están en

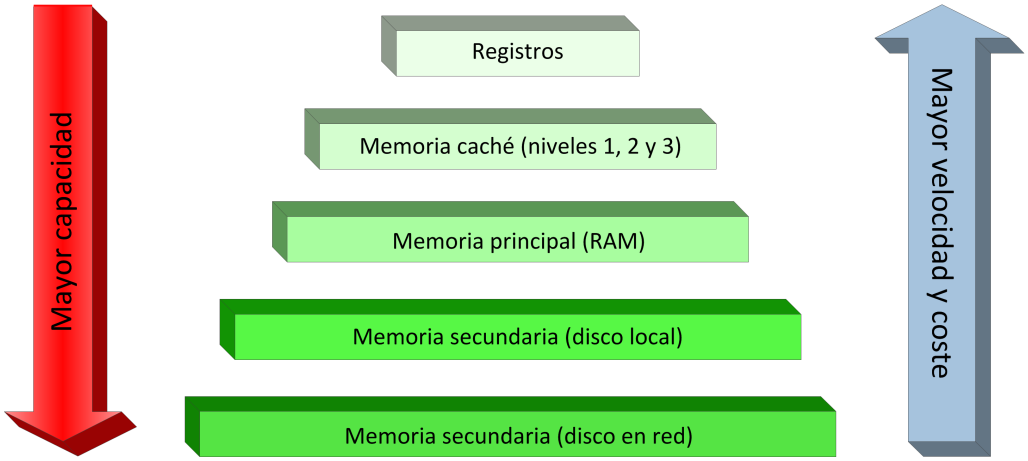


Figura 5.2: Jerarquía de memorias de un computador.

la memoria caché, tratando de evitar el costoso trasiego de ida entre la memoria principal y la memoria caché de los mismos cuando se necesita utilizarlos y de vuelta cuando se deja de hacerlo.

5.3. Los computadores de altas prestaciones

Un computador paralelo está formado por un simple computador con múltiples elementos de cómputo internos o por múltiples computadores interconectados entre sí para formar una plataforma de computación de altas prestaciones [150]. Como es evidente, dicho multiprocesador, en el primer caso, o multicomputador, en el segundo, resuelve grandes problemas en un tiempo mucho más reducido que el que invertiría un sencillo computador de sobremesa.

Estos computadores, a veces compuestos hasta por cientos o miles de elementos de procesamiento, presentan cada día un mayor uso, debido a la cada vez mayor demanda de computación en multitud de campos de la ciencia, la ingeniería o las finanzas. Ello supone que, sin la presencia de dichos computadores, muchas de estas aplicaciones no podrían ejecutarse, o que el tiempo de ejecución de las mismas resultaría tan elevado como inaceptable.

Son muchas las disciplinas en las cuales se usan habitualmente estos computadores paralelos de altas prestaciones. Por citar sólo algunos ejemplos, podemos

hablar de las predicciones metereológicas, el análisis de secuencias de ADN por parte de los biólogos, el diseño de nuevos fármacos por las empresas farmacéuticas, las exploraciones geológicas por parte de las empresas petrolíferas, las inversiones en bolsa en los mercados de finanzas, el diseño de vehículos aeroespaciales por parte de la NASA o en el desarrollo de efectos especiales o películas de animación por parte de la industria del cine. Todas estas aplicaciones presentan un característica en común: su alta demanda computacional asociada, en ocasiones, a la entrada o salida de un gran volumen de datos.

En 1972, Michael Flynn propuso una clasificación de los ordenadores, conocida como Taxonomía de Flynn [151], la cual los catalogaba en función de la ausencia o presencia de un control global que regule el flujo de las instrucciones y los datos. Si bien los computadores con un único elemento de proceso pertenecerían al tipo SISD (*Single Instruction Single Data*), los multiprocesadores se clasificarían en el tipo MIMD (*Multiple Instruction Multiple Data*), donde diferentes flujos de instrucciones (probablemente incluso diferentes programas) se ejecutan sobre diferentes conjuntos de datos de manera asíncrona. Esto se corresponde por tanto con un computador con distintos elementos de proceso, cada uno de ellos con su propia unidad de control. Existen tres tipos de arquitecturas MIMD de acuerdo a la organización del espacio de memoria, dos de ellas ya clásicas, como son la de *Memoria Compartida* y de *Memoria Distribuida* y una nueva denominada *Memoria Compartida Distribuida* que es una combinación de las dos anteriores. Procedemos a continuación a describir cada una de ellas.

5.3.1. Los multiprocesadores de memoria compartida

Un multiprocesador con Memoria Compartida, también denominado un multiprocesador simétrico (SMP), consta de un conjunto de elementos de proceso, cada uno de ellos con una pequeña memoria caché, que comparten un conjunto de módulos de memoria a través de una red de interconexión, como muestra la figura 5.3.

La comunicación entre los distintos elementos de proceso se realiza mediante la memoria. Para ello, basta con que uno de ellos escriba una variable en memoria y ésta sea leída por otro. Cualquier elemento de proceso puede acceder a cualquier

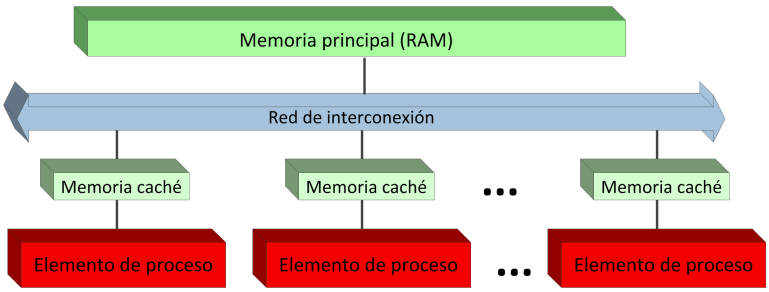


Figura 5.3: Arquitectura de un sistema de memoria compartida.

posición de memoria, siendo uniforme el tiempo de acceso a cualquiera de ellas, en lo que se conoce como máquinas de tipo UMA (Uniform Memory Access). La ventaja de este tipo de arquitecturas es que son más fáciles de programar, ya que el programador no tiene que implementar comunicaciones explícitas entre los elementos de proceso, aunque debe gestionar la compartición de datos entre los diferentes hilos de ejecución.

Sin embargo, debido a que la comunicación se realiza compartiendo zonas comunes de memoria, el tráfico entre los elementos de proceso y la memoria será cuantioso y pueden producirse conflictos de acceso si varios elementos intentan acceder, en un mismo instante, a las mismas posiciones con el objetivo de modificar una o más variables comunes, las cuales deben ser gestionadas por el programador mediante secciones críticas, semáforos, etc. Como consecuencia, los accesos a memoria se convierten en un cuello de botella que condiciona el rendimiento del sistema y que implican que estos sistemas no escalen adecuadamente.

Una forma de mejorar el acceso a memoria consiste en utilizar una estructura jerárquica de memorias, como la ya citada con anterioridad, incluyendo por ejemplo diferentes niveles de memoria caché independiente para cada elemento de proceso. Existen además otras alternativas como son el uso de memorias con varios puertos de acceso y el permitir que se pueda acceder simultáneamente a datos situados en módulos diferentes.

Aunque el uso de la memoria caché supone una mejora notable en los tiempos de acceso a memoria, bien es cierto que da lugar a otro problema denominado *coherencia* o *consistencia de cachés* que debe ser solucionado. Ya que la memoria se comparte por los diferentes elementos de proceso, una misma variable puede ser

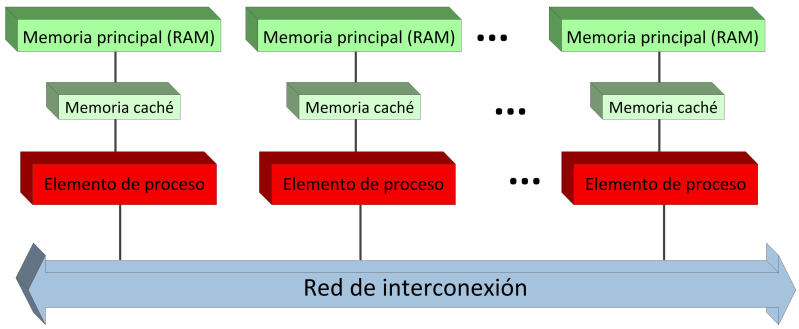


Figura 5.4: Arquitectura de un sistema de memoria distribuida.

leída y modificada por cualquiera de ellos. Si varios elementos de proceso poseen una copia propia de una variable compartida, almacenada en su memoria caché, y uno de ellos la modifica, la copia que el otro posee dejará de ser válida y deberá ser actualizada.

5.3.2. Los multiprocesadores de memoria distribuida

En estos multiprocesadores, también denominados multicomputadores, cada elemento de proceso posee su propia memoria local donde guarda sus datos, la cual representa la única memoria a la que puede acceder (ver figura 5.4). Los distintos elementos de proceso, también denominados *nodos*, se unen mediante una red de interconexión y la comunicación entre ellos se realiza mediante el paso de mensajes, lo cual ocurre cuando un proceso necesita datos que no posee y que están almacenados en la memoria de otro proceso.

El factor que más afecta a la eficiencia de estas máquinas es el tipo de red utilizada, influyendo en el tiempo asociado a las comunicaciones entre los elementos de proceso dentro del tiempo de ejecución del algoritmo paralelo. Como es obvio, el acceso a los datos presentes en la memoria local es mucho más rápido que el acceso a los datos que posea otro proceso. Ello supone que a la hora de programar dichos multiprocesadores, el código y los datos deberían estructurarse de manera que la mayoría de accesos a los datos, por parte de un elemento de proceso, tengan lugar dentro de su memoria local, evitando así las costosas comunicaciones entre los distintos elementos. Los multiprocesadores basados en memoria distribuida presentan estas ventajas frente a los basados en memoria compartida:

- Admiten un número muy elevado de elementos de procesamiento, dando lugar a los denominados MPPs (massively parallel computers).
- Son fácilmente escalables, permitiendo aumentar el número de procesadores y con ello las prestaciones de la máquina.
- La tecnología de diseño y fabricación suele ser más asequible.

A su vez, estos multiprocesadores presentan estos inconvenientes:

- Su programación es más compleja y alejada de la programación secuencial, siendo complicado diseñar algoritmos paralelos eficientes.
- No es fácil equilibrar la carga computacional entre los distintos procesadores, lo cual tiene una relación directa con las prestaciones obtenidas.
- Las comunicaciones añaden un tiempo adicional en la ejecución de los programas. A veces, equilibrar la carga y disminuir el número de mensajes pueden llegar a ser objetivos contrapuestos.
- Debe replicarse el núcleo del sistema operativo y el código de los programas en cada procesador, limitando así en parte el aprovechamiento de la memoria.

5.3.3. Los multiprocesadores de memoria compartida distribuida

Como podemos contemplar en la figura 5.5, estos multiprocesadores son en realidad un modelo híbrido entre los dos anteriores, de manera que el procesador que veíamos en el caso del multiprocesador de memoria distribuida, junto con su memoria local, se reemplaza por un multiprocesador de memoria compartida. Este tipo de sistemas, denominados DSM (Distributed Shared Memory), presenta dos posibles alternativas, como a continuación se detalla.

Por un lado aquella en la cual se intenta ocultar la naturaleza distribuida de la memoria, presentándose como un espacio único de memoria compartida, aunque

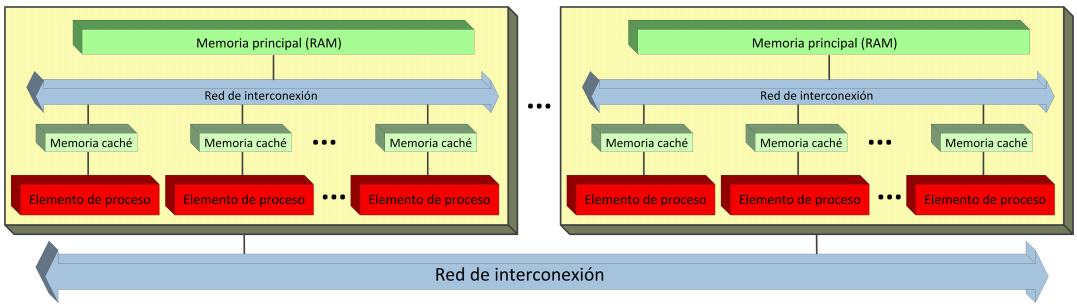


Figura 5.5: Arquitectura de un sistema de memoria compartida distribuida.

implementado sobre un sistema de memoria distribuida. La memoria estará por tanto lógicamente compartida, pero físicamente distribuida. Esto supone que los tiempos de acceso a los datos no serán uniformes, motivo por el cual la localidad de los datos vuelve a ser muy importante en las prestaciones de los algoritmos paralelos allí ejecutados. Por ese motivo, a esta arquitectura se le denomina NUMA (Non-Uniform Memory Access). Puede decirse que estos multiprocesadores combinan las ventajas de los multiprocesadores anteriores. Así, poseen la escalabilidad de los multiprocesadores de memoria distribuida y la facilidad de programación de los multiprocesadores de memoria compartida.

Por otro lado, la segunda alternativa es aquella en la cual no se pretende ocultar al usuario el hecho de que la memoria está físicamente distribuida. Esto implica que estos sistemas se presenten como un sistema de memoria distribuida donde cada nodo es en realidad un sistema de memoria compartida. Es lo que se conoce como un cluster de arquitecturas SMPs. Un buen ejemplo de este tipo de configuración son las basadas en grupos (clusters) de PCs (ver figura 5.6), o clusters Beowulf. Hoy en día, los PCs de sobremesa presentan un diseño donde se reflejan todos los adelantos tanto tecnológicos como de organización, siendo máquinas paralelas propiamente dichas basadas en memoria compartida, al estar compuestas por múltiples procesadores que comparten una memoria común. Los avances producidos en el campo de las redes de interconexión, que han dado lugar al desarrollo de redes rápidas tales como Gigabit Ethernet, InfiniBand, etc., junto con el de las herramientas de programación paralela, como MPI, permiten trabajar con un conjunto de máquinas independientes y conectadas entre sí como si se tratara de una sola. Con ello se consigue disponer de un multicomputador cuyos nodos son en realidad máquinas paralelas de memoria compartida muy potentes,

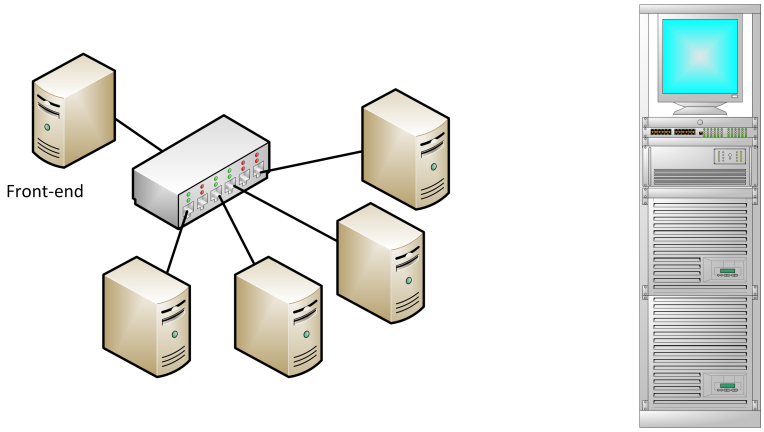


Figura 5.6: Cluster de PCs independientes o agrupados en un armario.

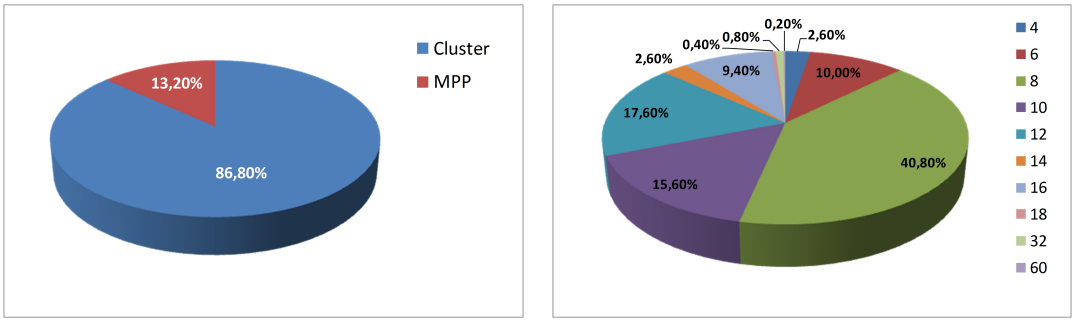
desbancando incluso en potencia computacional a los grandes multiprocesadores, pero con un coste económico muy inferior, ofreciendo una excelente relación precio/prestaciones y siendo a la vez fácilmente escalables a un número mayor de nodos.

Si echamos un vistazo a la lista de los 500 supercomputadores más potentes del mundo [152], pondremos de manifiesto lo que estamos comentando. Así, un 86.8 % de dichos supercomputadores presentan una arquitectura basada en clusters, frente al 13.2 % restantes catalogados como MPPs, como refleja la figura 5.7(a). Por otro lado, la figura 5.7(b) muestra el porcentaje de supercomputadores de acuerdo al número de núcleos (de 4 a 60) por procesador. Como podemos observar, un 40.8 % de los supercomputadores están compuestos por procesadores de 8 núcleos, seguidos de un 17.6 % compuestos por 12 y un 15.6 % por 10.

5.4. Paradigmas de la Programación Paralela

En la actualidad, los diferentes modelos de programación en paralelo tienen su origen de acuerdo a la plataforma hardware en la que se implementarán por defecto. Los modelos existentes son los siguientes:

- Hilos de ejecución explícitos.
- Paso de mensajes.



(a) Según su arquitectura.

(b) Según el número de núcleos por procesador.

Figura 5.7: Porcentajes de supercomputadores en la lista Top500 de acuerdo a su arquitectura y al número de núcleos por procesador.

- Directivas del compilador.
- Programación paralela multinivel.

5.4.1. Hilos de ejecución explícitos

En esta aproximación de programación paralela, una aplicación crea explícitamente varios *hilos*, o *threads*, paralelos que se ejecutan dentro del mismo espacio de memoria. Dichos hilos se comunican entre sí usando áreas compartidas en memoria, de manera que los conflictos de acceso a la misma se gestionan mediante el uso de bloqueos, semáforos y exclusiones mutuas. Se deja en manos del programador la descomposición de la aplicación en las diferentes tareas paralelas, o hilos, así como su sincronización en memoria entre los mismos.

Lógicamente esta aproximación está pensada para trabajar con máquinas multiprocesador con memora compartida y no puede ser implementada bajo máquinas con memoria distribuida, las cuales no ofrecen el soporte apropiado. Por otro lado, es muy habitual encontrarla en aplicaciones de servidores de datos, no siendo muy común en Computación Científica.

Actualmente el interfaz más usado para desplegar este paradigma es el que ofrece la librería Pthreads (POSIX Threads), basada en POSIX [153], el estándar soportado por UNIX, Linux, Mac OS X y Solaris que define un API (Application Programming Interface) para crear, gestionar y coordinar hilos de ejecución.

Windows no implementa el estándar POSIX directamente, pero hay diferentes soluciones que permiten que los programas basados en POSIX pueden ejecutarse en dicha plataforma.

5.4.2. Paso de mensajes

Este otro modelo considera a la memoria del sistema como un conjunto de espacios de direcciones diferentes, cada uno de ellos asociado a un proceso, los cuales se comunican a través del envío y recepción de datos.

La idea consiste en dividir a una aplicación en un conjunto de procesos que puedan ejecutarse concurrentemente sobre un conjunto distinto de datos, almacenados en su memoria local. Si en algún momento alguno de los procesos necesita datos que no posee para continuar trabajando, estos le son enviados por otro proceso por medio de un mensaje. Al igual que en la aproximación anterior, es aquí también responsabilidad del programador la descomposición de una aplicación en las diferentes tareas paralelas, así como la sincronización entre las mismas.

Se dispone por tanto de primitivas para llevar a cabo la comunicación, como *envía un mensaje al proceso i* o como *recibe un mensaje del proceso j*. Dichos mensajes pueden ser clasificados de dos formas distintas:

1. Dependiendo del número de procesos participantes a nivel emisor o receptor. Podemos tener:

   a) Mensajes punto a punto: Participan un solo emisor y un solo receptor.

   b) Difusión de uno a todos: Forman parte un solo emisor y múltiples receptores. Uno de los procesos envía el mismo mensaje al resto de procesos.

   c) Difusión de todos a uno: Trabajan múltiples emisores y un solo receptor. Todos los procesos envían un mensaje al mismo proceso destino.

   d) Multidifusión de todos a todos: Colaboran múltiples emisores y múltiples receptores. Cada proceso envía un mensaje al resto.

2. Dependiendo del comportamiento bloqueante o no bloqueante del emisor y del receptor.

   De acuerdo al comportamiento del emisor tendremos:

   a) Comunicación con emisor bloqueante: Se da cuando el emisor queda bloqueado hasta que el receptor reciba o consulte el mensaje.

   b) Comunicación con emisor no bloqueante: Se produce en caso contrario, es decir, cuando el proceso que envía los datos reanuda su ejecución inmediatamente tras enviar el mensaje.

   De acuerdo al comportamiento del receptor tendremos:

   a) Comunicación con receptor bloqueante: El receptor se bloquea tras ejecutar la instrucción de recepción de un mensaje, hasta que éste llegue. Se da cuando el receptor necesita los datos para continuar.

   b) Comunicación con receptor no bloqueante: Se produce cuando el receptor no se bloquea tras la instrucción de recepción del mensaje, volviendo de inmediato y dedicando ese tiempo a otras tareas si el dato no es imprescindible para continuar. De manera obligada, el receptor deberá consultará periódicamente si el mensaje ya ha llegado o no. Otra posibilidad intermedia es que el receptor quede bloqueado durante un tiempo determinado a la espera del mensaje, transcurrido el cual retornaría al trabajo.

El enfoque de paso de mensajes está claramente pensado para trabajar con multiprocesadores con memoria distribuida, aunque es fácilmente implementable en multiprocesadores con memoria compartida. En Computación Científica, el paso de mensajes se ha estandarizado bajo el uso de la librería MPI [12], que más tarde describiremos, aunque previamente existieron otras alternativas ampliamente utilizadas como PVM [154] .

5.4.3. Directivas del compilador

Este paradigma de programación consiste en que el programador incorpore una serie de comentarios especiales, denominados *directivas,* sobre el código de un

programa secuencial, indicando qué regiones del programa deben ser paralelizadas y cómo se deben distribuir el trabajo y los datos entre los elementos de proceso.

Obviamente, esto requiere un compilador especial que interprete dichos comentarios o directivas, generando el código apropiado que o bien usará hilos de ejecución explícitos, habitualmente, o paso de mensajes para implementar las comunicaciones.

La descomposición de una aplicación en tareas paralelas la lleva a cabo el compilador, con ayuda del programador, por medio de las citadas directivas, lo que, en muchas ocasiones, supone una reestructuración o nueva programación del código secuencial ya existente. Por otro lado, la gestión de las regiones críticas en memoria y la sincronización entre las tareas es responsabilidad exclusiva del compilador, aunque vendrán definidas por el programador.

Considerando que esta aproximación abstrae al programador de la capa de comunicación empleada, podemos decir que puede usarse tanto en los sistemas de memoria compartida como en los de memoria distribuida, aunque en la práctica es en los primeros en los que se usa habitualmente, ya que la disponibilidad de un espacio de memoria global simplifica el trabajo a realizar por el compilador.

En los últimos años, OpenMP [11] se ha convertido en una de las librerías más utilizadas basadas en este paradigma de programación, dirigida exclusivamente a sistemas con memoria compartida. La programación mediante OpenMP es mucho más sencilla que mediante Pthreads. Además, mientras que el uso de Pthreads conlleva que el código de partida sufra muchas modificaciones para incorporar toda la funcionalidad asociada, con OpenMP los programas quedarán menos alterados, permitiendo la codificación de forma progresiva y aumentando la expresividad del código, lo que puede facilitar su comprensión por parte de programadores no expertos. Inclusive, se permite la compilación de código paralelizado en compiladores que no contemplen las directivas de OpenMP, dando lugar, como resulta evidente, a un ejecutable secuencial.

Menos usado es el estándar High Performance Fortran (HPF) [155], el cuál consiste en un lenguaje de alto nivel basado en Fortran 90 que incorpora directivas interpretadas por el compilador que especifican el paralelismo de datos. Está diseñado para trabajar tanto con sistemas de memoria compartida como

distribuida.

La directiva más utilizada tanto en OpenMP como en HPF es la paralelización de bucles, en contraste con una paralelización a nivel de programa llevada a cabo cuando usamos MPI.

5.4.4. Programación paralela multinivel

El modelo de programación paralela multinivel es una combinación entre el paso de mensajes y, o bien las directivas del compilador, o bien los hilos de ejecución explícitos. Si bien es cierto que dicha aproximación parte del reconocimiento de considerar a los sistemas de memoria compartida como el modelo que ofrece una mayor sencillez en el desarrollo de aplicaciones paralelas, también lo es el hecho de que una gran parte de las aplicaciones de carácter científico están basadas en el modelo de paso de mensajes.

Si a ambas premisas le unimos que los sistemas de memoria compartida distribuida representan el modelo más empleado entre las arquitecturas paralelas, parece claro que dicho paradigma de programación es la alternativa natural más adecuada para desarrollar aplicaciones que se ejecutarán desde potentes supercomputadores hasta sencillos clusters de PCs.

Esta aproximación ofrece un considerable potencial en el incremento de prestaciones de una aplicación paralela y no incrementa en demasía la dificultad en la programación frente a usar exclusivamente paso de mensajes. En particular, la escalabilidad es seguramente el argumento más apropiado para usar este paradigma de programación.

Supongamos una aplicación paralela basada en MPI, ejecutada sobre un cluster de PCs multinúcleos, compuesto cada uno de ellos únicamente por dos núcleos, en la cual se mantiene el tamaño del problema pero se aumenta paulatinamente el número de procesos, empleando únicamente uno de los dos núcleos de cada PC. Obviamente ocurrirá que, a partir de un número concreto de procesos empleados, las prestaciones de la aplicación paralela comenzarán a disminuir, debido a que los tiempos invertidos en las comunicaciones tomarán mayor fuerza frente los tiempos de cómputo. En ese caso, la programación multinivel permitiría aprovechar

de forma más natural los diferentes niveles de la arquitectura del computador, creando para ello dos hilos de ejecución mediante OpenMP por cada proceso MPI. Ello supondrá, como es lógico, un incremento de las prestaciones.

Es precisamente esta aproximación multinivel que combina el paso de mensajes, mediante MPI, y las directivas del compilador, mediante OpenMP, la que se ha escogido y seguido en esta tesis doctoral, y bajo la cual se han diseñado e implementado los diferentes algoritmos de cálculo estático y dinámico estructural.

5.5. Entornos de programación paralela

De igual manera que las arquitecturas paralelas han ido prosperando y mejorando a lo largo de los años, la programación paralela ha ido también evolucionando a fin de usar eficientemente dichas arquitecturas.

Actualmente, MPI representa el estándar de facto empleado por excelencia en máquinas con memoria distribuida. Por otro lado, OpenMP es la librería de programación paralela más utilizada en máquinas con memoria compartida. Una característica primordial de ambos entornos es la portabilidad que ofrecen a las aplicaciones paralelas desarrolladas con los mismos, las cuales se podrán ejecutar, tras el proceso de compilación oportuno, en un simple PC o sobre un potente multicomputador. Existe además una tendencia a usar ambos entornos conjuntamente, debido al creciente protagonismo que están adquiriendo las arquitecturas de memoria compartida distribuida y la llamada programación paralela multinivel.

5.5.1. Message Passing Interface (MPI)

En sus comienzos, las diferentes librerías o paquetes de comunicación basados en pasos de mensajes diferían tanto en su implementación y presentaban soluciones tan diversas que hacían realmente difícil la portabilidad de las aplicaciones paralelas de unas máquinas a otras. Surgió entonces la idea de desarrollar un estándar consensuado, que ofreciera un interfaz común a las aplicaciones y que fuera independiente de la arquitectura. Dicho estándar recibió el nombre de MPI

(Message Passing Interface) [12] y fue desarrollado entre los años 1992 y 1994 por el llamado MPI Forum, compuesto por más de 60 personas de 40 organizaciones pertenecientes a la industria, el ámbito universitario y el gubernamental.

El resultado de dicho esfuerzo define una especificación de funciones, junto con su interfaz y el comportamiento de las mismas, que pueden ser usadas para enviar y recibir datos entre los elementos de proceso que forman parte de un sistema que emplee el paradigma de paso de mensajes. A diferencia de otros muchos paquetes de comunicación, MPI es un estándar único, siendo los programas, en consecuencia, completamente portables sin necesidad de realizar cambio alguno. Aunque inicialmente fue diseñado para trabajar con sistemas con memoria distribuida, funciona también sobre procesadores con memoria compartida. Incluso puede ser simulado en máquinas con un solo procesador, mediante la concurrencia de varios procesos en una misma máquina.

Hay que tener en cuenta que MPI es tan sólo una especificación. En consecuencia, los fabricantes de multiprocesadores (Cray, SGI, Sun, etc.) y de software, como Microsoft, desarrollaron posteriormente librerías de rutinas que reflejaban el comportamiento y los requerimientos de dicho estándar, optimizadas de acuerdo a la arquitectura concreta de la máquina o a su red de interconexión, pudiendo ser empleadas desde lenguajes de programación ampliamente utilizados por la comunidad científica como C, C++ o Fortran. A día de hoy, algunas de las implementaciones de MPI más utilizadas son MPICH [156], Open MPI [157] y Microsoft MPI (MS-MPI) [158].

En realidad, un programa paralelizado mediante MPI consiste en dos o más procesos autónomos, identificados con un número o rango, que ejecutan su propio código y se comunican por medio de las rutinas definidas en el estándar. En su especificación original, MPI no permitía la creación dinámica de procesos durante la ejecución de un programa, sino que dicho número de procesos permanecía fijo durante la ejecución de la aplicación paralela y se determinaba por parte del usuario en el momento de su lanzamiento. El estándar MPI incluye rutinas para las siguientes operaciones entre procesos:

- Comunicaciones punto a punto, para el envío y la recepción de datos entre dos procesos.

- Comunicaciones colectivas, que permiten transmitir y recibir datos entre más de dos procesos. Estas rutinas son más eficientes, en términos del tiempo asociado a las comunicaciones entre los procesos, que las de punto a punto, a la vez de proporcionar un código mas legible y más sencillo de mantener y de depurar. Existen a su vez diferentes tipos de comunicaciones colectivas:

  - Gestión del entorno y consultas: Se trata de un conjunto de rutinas usadas, por ejemplo, para inicializar o terminar el entorno de ejecución en MPI, para finalizar la ejecución de los procesos que forman parte de un grupo, para consultar el número de procesos dentro de un grupo o para averiguar el identificador que le corresponde a cada uno.
  - De movimiento de datos: Entre otras operaciones, permite que un proceso envíe una serie de datos idénticos al resto de procesos de un grupo, que un proceso distribuya una serie de datos entre un número concreto de procesos, que un procesador reciba de parte de un conjunto de procesos una serie datos distribuidos entre los mismos y los agrupe, o que todos los procesos envíen un dato al resto de procesos o distribuyan una serie de datos entre ellos mismos.
  - Computación colectiva, u operación de reducción, donde un proceso recibe un conjunto de datos de otros procesos de su mismo grupo y realiza alguna operación aritmética determinada con ellos o calcula el máximo, el mínimo, etc., de los mismos.

- Gestión dinámica de grupos de procesos: Un grupo es simplemente un conjunto ordenado de procesos, cada uno de los cuales recibe un identificador. Se permite que un proceso pase a formar parte de uno o más grupos o que deje de hacerlo.

- Sincronización, donde todos los procesos esperan hasta que todos los integrantes de un mismo grupo hayan alcanzado un punto del código determinado.

- Gestión de tipos de datos ya predefinidos por la especificación o de cualquier otro tipo creados por el usuario.

- Gestión de topologías de procesos: En MPI, una topología es un mecanismo para asociar diferentes esquemas de identificación a los procesos pertenecientes a un grupo, a fin de obtener una mayor eficiencia en las comunicaciones y una programación más sencilla. Dicha topología, que no guarda relación alguna con la distribución física y espacial de los elementos de proceso dentro de la máquina, describe un mapeo u ordenación virtual de los procesos de acuerdo a alguna forma geométrica. MPI da soporte a dos tipos de topologías: Cartesiana o Grid y en forma de grafo.

Con posterioridad, MPI se vio mejorado al incorporarle nuevas funcionalidades. Surge así, en 1997, el estándar MPI-2 [159], el cual proporciona nuevas peculiaridades, como la gestión dinámica de procesos, las operaciones de acceso a memoria remota, la definición de los enlaces desde C++ o desde Fortran 90 y una especificación de E/S de datos en paralelo (MPI-IO), donde múltiples procesos leen o escriben datos simultáneamente en un fichero compartido.

Entre las implementaciones del sistema de E/S de ficheros en paralelo encontramos la de la librería ROMIO [160], la cual forma parte actualmente de MPICH. Otras implementaciones de MPI como Open MPI o las de Cray o IBM incorporan variantes de ROMIO en sus implementaciones de MPI-IO.

En el año 2012 surge MPI-3 [161], que incluye, en sus versiones 3.0 y 3.1, la extensión de las operaciones colectivas para incluir versiones no bloqueantes, algunas de las cuales tienen que ver con rutinas de E/S, y la especificación de los enlaces desde Fortran 2008. Además, se han eliminado algunos tipos de datos y enlaces de C++ en desuso. Actualmente, el MPI Forum está ya trabajando en la especificación 4.0.

5.5.2. OpenMP

OpenMP [11] es una especificación que define un API para paralelizar programas escritos en Fortran, C o C++ sobre arquitecturas de memoria compartida multiplataforma, que van desde un sencillo PC de sobremesa a un multiprocesador.

Esta iniciativa fue plasmada en el año 1997 por un grupo de especialistas de la industria y de diferentes centros de investigación con el objetivo de desarrollar un interfaz consensuado sobre el cual se pudieran desarrollar aplicaciones paralelas que fueran portables a máquinas de memoria compartida de diferentes fabricantes. Hasta ese momento, existía una amplia diversidad de modelos de programación, todos ellos diferentes y únicamente válidos para una plataforma de memoria compartida concreta, lo que impedía la portabilidad entre las mismas.

Básicamente, OpenMP está compuesto por los tres siguientes componentes:

- Un conjunto de directivas, o constructores, usados por el programador para informar al compilador de los diferentes aspectos relacionados con la paralelización de la aplicación. Con ellas, el programador indica qué instrucciones se deben ejecutar en paralelo y cómo se deben distribuir entre los hilos que las ejecutarán. A modo de ejemplo, OpenMP ofrece una directiva para indicarle al compilador que paralelice un bucle. Cualquier directiva se expresa del siguiente modo:

  #pragma omp <directiva específica>

- Una librería *runtime* de alto nivel compuesta por rutinas que permiten determinar o consultar parámetros del algoritmo paralelo, tales como como el número de hilos de ejecución participantes o el identificador de un hilo de ejecución concreto.

- Un conjunto de variables de entorno que pueden ser usadas para definir parámetros paralelos en el sistema de *runtime*.

Eso supone que OpenMP no es un lenguaje de programación, sino una notación que puede ser incorporada a un programa secuencial ya existente, para describir cómo se comparte el trabajo entre los diferentes hilos que lo ejecutarán sobre diferentes núcleos de cómputo y cómo se accede a los datos compartidos a medida que se necesitan.

El primer paso para desarrollar un programa paralelo con OpenMP, a partir de un código secuencial, es identificar aquellas partes del código que pueden ser ejecutadas concurrentemente entre diferentes elementos de proceso. En ocasiones,

ello conlleva la necesidad de reemplazar un algoritmo por otro alternativo que ofrezca un mayor nivel de paralelismo.

El segundo paso consiste en ir incorporando, de manera incremental, las directivas que vayan paralelizando las diferentes partes del código, sin que esto tenga influencia alguna sobre el resto del programa que esté aún sin modificar y que se ejecutará de manera secuencial.

En OpenMP, la entidad básica es el *hilo*, o thread, el cual es capaz de ejecutar de manera autónoma e independiente un flujo de instrucciones. A la hora de ejecutar un programa, el sistema operativo crea un proceso para ello y le asignará recursos tales como celdas de memoria y registros del procesador. Inicialmente, el programa comenzará con un sólo hilo de ejecución, como si de un programa secuencial se tratase. Cuando se encuentra con una directiva que le indica la necesidad de ejecutar una parte de código en paralelo, lo que se denomina una *región paralela*, se crea un equipo de hilos que colaboran en la ejecución de las instrucciones conjuntamente. Al final de dicha región paralela, los hilos se quedan inactivos a la espera de alcanzar otra región paralela, a excepción del hilo inicial, que continua su ejecución.

Si múltiples hilos colaboran en la ejecución del programa, todos ellos compartirán los recursos del proceso, incluyendo el espacio de direccionamiento en memoria, aunque debe haber partes de la misma con un acceso privado y restringido para cada hilo. Ello viene dado por el hecho de que el programador debe especificar si las variables que usa cada hilo son privadas o compartidas.

Durante todos estos años, se han ido generando nuevas especificaciones de OpenMP, siendo la última especificación la 4.1 [162], presentada en julio de 2015.

5.5.3. Uso conjunto de MPI y OpenMP

Como hemos visto con anterioridad, los dos modelos de programación más extendidos son el modelo de paso de mensajes, a través de MPI, y el de memoria compartida, con la utilización de OpenMP. La disponibilidad de máquinas de memoria compartida distribuida y la combinación de MPI y OpenMP, con el objetivo de explotar todo el potencial computacional que dichas máquinas nos

ofrecen, representa una aproximación prometedora en el campo de la programación paralela.

Para ello, inicialmente se debe llevar a cabo una distribución de los datos entre los nodos del multicomputador, donde los diferentes procesos MPI se ejecutarán independientemente y se comunicarán entre sí mediante pasos de mensajes. Al mismo tiempo, teniendo en cuenta que cada nodo es a su vez una máquina de memoria compartida, se lanzarán múltiples hilos de ejecución por proceso, empleando OpenMP.

Las aplicaciones híbridas son aquellas que combinan ambos modelos de programación con el fin de aprovechar, en cada caso, las bondades y potencialidades específicas de cada modelo e incrementar su escalabilidad. Ello conlleva, por tanto, una paralelización de grano grueso de toda la aplicación al usar MPI y una paralelización más fina en aquellas secciones paralelizadas con OpenMP. En consecuencia, las aplicaciones desarrolladas incorporan un doble nivel de paralelismo, uno entre los diferentes procesos MPI y otro entre los diferentes hilos lanzados por OpenMP, lo cual las convierte, simultáneamente, en versátiles y complejas.

Dependiendo de las características del hardware utilizado, se plantearán diferentes alternativas a la hora de ejecutar la aplicación, considerando el número de procesos MPI a lanzar por cada nodo y, en consecuencia, el número de hilos de ejecución a poner en marcha por cada proceso MPI. Por ejemplo, en una máquina compuesta por varios nodos de 4 núcleos, se podría lanzar un proceso MPI por nodo y 4 hilos de ejecución por proceso. De este modo, cada proceso dispondría de toda la memoria que le ofrece el nodo y la comunicación entre los procesos se llevaría a cabo mediante la red que interconecta los distintos nodos. Sin embargo, caben también otras posibilidades, como lanzar dos procesos MPI por cada nodo y 2 hilos de ejecución por proceso, realizándose ahora la comunicación entre los procesos MPI de un mismo nodo mediante su red de interconexión interna y donde ambos procesos compartirían la memoria del nodo. Incluso, llegados al extremo, habrá casos en los cuales será más eficiente lanzar 4 procesos MPI por nodo y ningún hilo de ejecución, teniendo en cuenta la sobrecarga que supone la creación de hilos y su sincronización.

Todas estas variantes de ejecución, unidas a otras características intrínsecas

de la máquina, como la disponibilidad y ubicación de los diferentes niveles de memorias caché, y a la eficiencia de cada tipo de paralelismo implementado en las distintas partes del código, darán lugar a cambios significativos en el rendimiento y en las prestaciones ofrecidas por la aplicación.

Otro aspecto importante a considerar es si las regiones paralelas de OpenMP están compuestas por invocaciones a funciones de MPI o no. En caso de que no sea así, no existirá mayor inconveniente, ya que solamente el hilo maestro estará activo durante la comunicación. En caso contrario, en el cual varios hilos invoquen conjuntamente a una misma rutina de MPI, será necesario emplear una librería MPI segura a hilos (threaded-safe). Esto tiene la ventaja de poder solapar comunicaciones y cálculo, pero requiere poner una mayor atención a la interacción entre MPI y OpenMP. Mientras que el estándar MPI-1 no incluye soporte multihilo, éste sí que se incorpora en diferentes niveles en la especificación de MPI-2.

5.6. Evaluación de los algoritmos paralelos

La Computación Paralela tiene como objetivo incrementar las prestaciones que se obtienen al ejecutar un programa paralelo en una máquina paralela frente al mismo programa resuelto de forma secuencial. Surgen por tanto conceptos que nos ayudan a evaluar las prestaciones de estos programas paralelos. Algunos de ellos son los que se citan a continuación.

5.6.1. Speedup o Incremento de Velocidad

Sea T_1 el tiempo del mejor algoritmo secuencial que resuelve un problema y T_p el tiempo del mejor algoritmo paralelo, empleando p elementos de cómputo. Definimos el *speedup* o *incremento de velocidad* S_p como el cociente entre T_1 y T_p:

$$S_p = \frac{T_1}{T_p} \tag{5.1}$$

Este concepto nos indica la ganancia de velocidad de un programa paralelo frente al secuencial y tiene como límite teórico $S_p \leq p$. Algunos de los motivos por los cuales en la práctica no se alcanza dicho valor teórico son los siguientes:

- Existe poca carga computacional asignada a los elementos de procesamiento y se invierte un tiempo sustancial en las comunicaciones. Este problema aparece, por ejemplo, cuando se aumenta paulatinamente el número de elementos de proceso involucrados en la resolución de un problema, sin variar el tamaño de éste, ocurriendo que el incremento de velocidad no aumenta en consonancia, debido a que cada vez será mayor el tiempo dedicado a las comunicaciones y menor el tiempo invertido en la computación por parte de cada uno de ellos.

- El algoritmo a paralelizar presenta partes de código que, por naturaleza, son secuenciales. En ese caso, y si es posible, deberíamos reemplazar esas partes por otras que puedan presentar un grado mayor de paralelismo.

- La carga del problema está mal equilibrada entre los elementos de proceso. Esto supondrá que algunos de los elementos tendrán más trabajo asignado que otros, de manera que aquellos con menor carga finalizarán antes. Sin embargo, la aplicación paralela no acabará hasta que termine el último de los elementos que participan en la resolución del problema.

- Demasiada contención en la memoria principal. Cuando múltiples elementos de proceso quieren acceder al mismo tiempo para leer o escribir datos en memoria, deberán esperar hasta que la memoria esté libre, con el retraso asociado que ello supone. Una forma apropiada para reducir la contención de memoria consiste en reescribir la aplicación mejorando la localidad de los datos, utilizando correctamente la memoria caché.

- Existe una cantidad de tiempo elevada, en la ejecución de la aplicación, dedicada a leer o escribir datos en disco en comparación con el tiempo de computación.

- La sobrecarga paralela (creación y finalización de regiones paralelas, creación, sincronización y finalización de hilos de ejecución, creación, sincroni-

zación, comunicación y finalización de procesos, etc.) es excesiva comparada con el tiempo de cómputo.

No obstante puede ocurrir incluso que el incremento de velocidad sea mayor que p, lo que se conoce como *super-speedup*. Habitualmente la explicación a dicha situación está detrás del uso de la memoria. Es posible que una aplicación secuencial y lo datos con los que trabaja no quepan completamente en la memoria principal del computador, motivo por el cual se invierte un tiempo extra de paginación y movimiento de datos entre la memoria principal y la memoria secundaria o de disco. En cambio, al ejecutar la versión paralela no se produciría dicho fenómeno de paginación, puesto que la aplicación y los datos sí que tendrían cabida en la memoria, en caso de trabajar con una máquina basada en memoria distribuida.

Existe también la posibilidad de que dicho super-speedup no sea debido a la memoria principal sino a la memoria caché. De manera similar a la explicación anterior es posible que la aplicación paralela haga un uso más eficiente de la memoria caché que la aplicación secuencial, produciéndose un número inferior de fallos de página, o puede ser que la aplicación paralela y sus datos asociados quepan en caché de manera completa, mientras que la aplicación secuencial y los datos la superan en tamaño. Tanto en un caso como en otro, en la aplicación secuencial se producirían un movimiento de datos entre la memoria caché y la memoria principal que no tendrían lugar con la aplicación paralela, con el incremento de tiempo adicional que ello supone. Un ejemplo habitual de super-speedup es el que se presenta a la hora de resolver problemas densos de álgebra lineal.

5.6.2. Eficiencia

La *eficiencia* de un algoritmo paralelo mide el grado de utilización y aprovechamiento de un sistema multiprocesador o multicomputador, relacionando el incremento de velocidad obtenido por dicho algoritmo frente al número de elementos de cómputo empleados:

$$E_p = \frac{S_p}{p} \tag{5.2}$$

Se suele expresar en tanto por cien, teniendo el límite en $E_p \leqslant 1(100\,\%)$, lo cual significa que todos los elementos de proceso involucrados están siendo totalmente utilizados. Obviamente este valor no se alcanza en la práctica, por los mismos motivos que comentábamos en el caso del incremento de velocidad.

5.6.3. Escalabilidad

Si bien es cierto que el hecho de aumentar el número de elementos de cómputo, manteniendo constante el tamaño del problema, suele traducirse en una disminución del incremento de velocidad y como consecuencia de la eficiencia, una forma de seguir manteniendo unas buenas prestaciones consiste en aumentar a la vez la dimensión del problema.

Definimos la *escalabilidad* como la capacidad de un determinado algoritmo de mantener sus prestaciones cuando aumenta el número de elementos de procesamiento. Un algoritmo paralelo se dice que es *escalable* si es capaz de mantener su eficiencia constante cuando aumentamos el número de elementos de cómputo, incrementando de forma equivalente el tamaño del problema.

5.7. Software de Computación Científica

La incesante labor de investigación por equipos especializados en numerosas disciplinas de la ciencia o la ingeniería ha dado lugar a desarrollos software muy eficientes, en términos de consumo de memoria y tiempos de ejecución, acompañados de una elevada fiabilidad en sus resultados. Buen ejemplo de ello viene dado en el campo de la Computación Numérica, donde muchos de esos avances se han plasmado a modo librerías de dominio público que están a disposición de ser utilizadas por cualquier usuario. Gran parte de dichas implementaciones incorporan incluso técnicas de Computación de Altas Prestaciones, ofreciendo una versión secuencial o paralela de las mismas [163].

Simultáneamente, existen iniciativas para fomentar el uso de dicho software numérico de dominio público, entre las que destacamos al DOE ACTS (Advanced CompuTational Software) Collection Project, llevado a cabo de manera exitosa

por el Lawrence Berkeley National Laboratory (LBNL), en colaboración con el National Energy Research Scientific Computing Center (NERSC), cuyo objetivo fue promocionar, durante 13 años, el uso de software financiado total o parcialmente por el Departamento de Energía (DOE) de EEUU y asegurar su disponibilidad y su aplicabilidad en herramientas software de carácter científico [164]. Precisamente, el doctorando desea agradecer su participación como asistente y como ponente invitado en el *ACTS Collection Workshop, Robust and High Performance Tools for Scientific Computing* organizado por el LBNL, en Berkeley, en el año 2002.

Lejos están los años en los cuales el programador de una aplicación desarrollaba todas las partes de las misma. A día de hoy, las aplicaciones software en ciencia o ingeniería conllevan tal complejidad que requieren de un equipo multidisciplinar formado, según sea el caso, por informáticos, matemáticos e investigadores expertos en la materia. Más aún, dicho equipo debe saber aprovechar todo aquel software disponible desarrollado por terceros, a modo de librerías o componentes que destaque considerablemente en su campo, siempre que sea posible basado en estándares y que nos sea útil en la incorporación a nuestra aplicación.

Las ventajas que dichos componentes proporcionan a nuestro código son muchas. Por un lado, implican un aumento lógico de productividad, por nuestra parte. Por otro lado, aportan portabilidad a diferentes plataformas, eficiencia, robustez y fiabilidad en los resultados, al estar previamente testados por numerosos usuarios.

A continuación se detallan diferentes librerías numéricas ampliamente utilizadas por la comunidad científica, muchas de las cuales han sido empleadas en esta tesis doctoral. En realidad, hablar de *librerías* es una mala traducción del inglés muy extendida y deberíamos emplear en su lugar la palabra *biblioteca*.

5.7.1. Núcleos computacionales

Bajo la nomenclatura de núcleos computacionales se agrupan todo un conjunto de funciones que resuelven operaciones elementales de algebra lineal entre vectores y matrices. Si dichas operaciones están recogidas en una librería y están

implementadas de manera eficiente, muchas veces incluso optimizada para una determinada plataforma computacional, nos evita la programación de las mismas desde un código de más alto nivel, al tiempo que proporciona eficiencia, fiabilidad y portabilidad a nuestra aplicación. Entre dichos núcleos computacionales se encuentran BLAS, LAPACK, PBLAS, BLACS y ScaLAPACK.

5.7.1.1. BLAS

BLAS (Basic Linear Algebra Subprograms) es una especificación de un conjunto de rutinas de bajo nivel para la realización de operaciones elementales de álgebra lineal entre vectores y matrices [165, 166, 167]. Representa un estándar de facto que ha sido implementado de manera óptima por los fabricantes de hardware para un amplio abanico de plataformas computacionales. Debido a su eficiencia, portabilidad e implementación óptima para múltiples arquitecturas, se usa habitualmente por parte de otras librerías numéricas de más alto nivel.

BLAS está estructurado en 3 niveles distintos. El nivel 1 incluye operaciones entre vectores. El nivel 2 incorpora operaciones entre matrices y vectores. Por último, el nivel 3 está formado por operaciones entre matrices mediante algoritmos orientados a bloques que reaprovechan los datos y minimizan sus movimientos en la jerarquía de memorias.

5.7.1.2. LAPACK

LAPACK (Linear Algebra PACKage) es una librería estándar compuesta por rutinas encargadas de resolver problemas de álgebra lineal sobre matrices densas y matrices banda, tales como sistemas de ecuaciones lineales, problemas de mínimos cuadrados y problemas de valores propios o de valores singulares [168]. Proporciona además diversas factorizaciones matriciales como la LU, Cholesky, QR, SVD, Schur, Schur generalizado, etc. Aunque inicialmente se programó en Fortran 77, en el año 2008 se reescribió en Fortran 90.

Está compuesta por tres tipos de rutinas o funciones: *rutinas driver*, que resuelven un sistema de ecuaciones, calculan los valores propios de una matriz, etc., *rutinas computacionales*, que realizan factorizaciones matriciales, como la

LU, QR, etc. y *rutinas auxiliares*, y de más bajo nivel, orientadas a bloques y a modo de extensiones de BLAS.

El objetivo original del LAPACK fue reescribir las librerías estándar LINPACK, orientada a la resolución de sistemas de ecuaciones lineales y problemas de mínimos cuadrados, y EISPACK, dedicada a la resolución de problemas de valores propios, para que ambas utilizaran las rutinas del nivel 3 de BLAS siempre que fuera posible. De esta forma, estarían basadas en algoritmos orientados a bloques y aprovecharían la jerarquía de memorias, además de ser portables a un amplio número de plataformas computacionales. Más aún, si los diferentes fabricantes proporcionaban versiones de BLAS nativas y optimizadas para sus máquinas, LAPACK se beneficiaría y conseguiría buenas prestaciones.

5.7.1.3. BLACS

BLACS (Basic Linear Algebra Communication Subprograms) incorpora un conjunto de rutinas de comunicaciones orientadas a operaciones básicas de álgebra lineal [169]. Las rutinas, simples y a la vez eficientes, están basadas en el modelo de paso de mensajes y emplean funciones de MPI o de PVM. De este modo, dichas rutinas pueden implementarse homogéneamente y eficientemente en un amplio abanico de plataformas de memoria distribuida. BLACS puede emplearse desde otras aplicaciones de álgebra lineal, lo que hará más sencillo su desarrollo y les proporcionará portabilidad. De hecho, representa la capa de comunicación de ScaLAPACK, como veremos más adelante, motivo por el cual se desarrolló.

5.7.1.4. PBLAS

PBLAS (Parallel Basic Linear Algebra Subprograms) es una librería formada por implementaciones paralelas portables de las rutinas de BLAS para máquinas de memoria distribuida [170]. Presenta un interfaz muy similar a BLAS y, de igual manera, está estructurada en 3 niveles, que realizan en paralelo el mismo tipo de operaciones. PBLAS emplea BLAS para realizar las operaciones numéricas elementales y BLACS para llevar a cabo las comunicaciones entre los nodos.

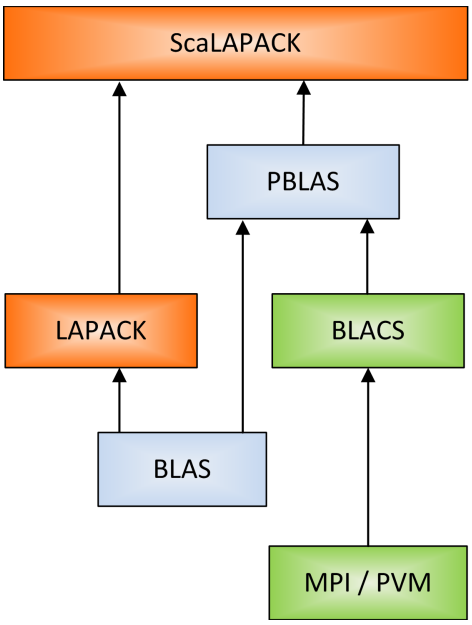


Figura 5.8: Diagrama de dependencias de los núcleos computacionales.

5.7.1.5. ScaLAPACK

ScaLAPACK (Scalable Linear Algebra PACKage) implementa en paralelo gran parte de las rutinas de álgebra lineal recogidas en LAPACK [171]. Resuelve, sobre máquinas de memoria distribuida, problemas computacionales donde la matriz de coeficientes es densa o en forma de banda, tales como sistemas de ecuaciones lineales, problemas de mínimos cuadrados y problemas de valores propios y valores singulares.

La librería está escrita en Fortran, con excepción de unas pocas rutinas auxiliares dedicadas a problemas de valores propios escritas en C, y emplea una distribución cíclica bidimensional a bloques de los datos. Pretende ofrecer eficiencia, escalabilidad, fiabilidad, portabilidad y facilidad de uso, compartiendo su interfaz con LAPACK siempre que sea posible. Muchos de esos objetivos los consigue empleando LAPACK y PBLAS, los cuales se apoyan en BLAS y BLACS, tal y como muestra la figura 5.8. Es por tanto portable a máquinas de memoria distribuida que empleen MPI o PVM.

5.7.1.6. Implementaciones alternativas

Los 5 núcleos computacionales que hemos descrito en las secciones previas se encuentran accesibles, sin optimizar para ninguna plataforma concreta, en la página de NETLIB [172], un repositorio de distribución gratuita de software numérico. De allí puede descargarse el código fuente y compilarlas posteriormente. Alternativamente, estos núcleos se han implementado por diferentes desarrolladores de software o distintos fabricantes de hardware. Algunos ejemplos son los siguientes:

- CBLAS [173] y CLAPACK [174]: Se corresponden con una implementación de BLAS y LAPACK en lenguaje C generadas a partir de la utilidad f2c, un programa desarrollado en los laboratorios Bell para convertir código de Fortran 77 a C.
- OpenBLAS [175]: Incluye una versión optimizada y multihilo de BLAS basada en GotoBLAS, el cual dejó de mantenerse.
- ATLAS (Automatically Tuned Linear Algebra Software) [176]: Tiene la capacidad de compilarse de manera óptima y automática para diferentes arquitecturas. Incluye una implementación multihilo de BLAS y algunas funciones de LAPACK.
- Intel MKL (the Intel Math Kernel Library) [177]: Incluye versiones optimizadas multihilo para los procesadores de Intel de BLAS, LAPACK, BLACS, PBLAS y ScaLAPACK bajo los sistemas operativos Linux y Windows.
- ACML (AMD Core Math Library) [178]: Formada por implementaciones óptimas multihilo para procesadores de AMD de BLAS y LAPACK.
- PLAPACK [179]: Es un paquete con una funcionalidad similar a ScaLAPACK pero con un interfaz diferente, más próxima a una programación orientada a objetos.
- FLAME [180]: Se trata de un proyecto que intenta facilitar el desarrollo de las operaciones de álgebra lineal densa y del software basado en ella, incluyendo una nueva notación de los algoritmos y nuevos interfaces del

software desarrollado diferentes a los tradicionalmente ofrecidos por BLAS y LAPACK. Aun así, el usuario puede configurar las librerías para trabajar con el nuevo API o con el tradicional de BLAS y LAPACK en Fortran 77, manteniendo la portabilidad de todas las aplicaciones basadas en dichos núcleos computacionales. Está formado por las librerías BLIS, libFLAME y Elemental.

- PLASMA (Parallel Linear Algebra Software for Multicore Architectures): Consiste en una librería basada en Fortran y C diseñada para obtener mejores prestaciones en máquinas multinúcleo bajo memoria compartida [181]. Incorpora la misma funcionalidad que LAPACK, a diferencia de matrices banda y la resolución de problemas de valores propios y valores singulares. No reemplaza a ScaLAPACK como software para máquinas de memoria distribuida, ya que sólo es compatible en memoria compartida.

- MAGMA (Matrix Algebra on GPU and Multicore Architectures): Se trata de un software con una funcionalidad similar a LAPACK desarrollado para atender las necesidades de sistemas híbridos y heterogéneos equipados con aceleradores hardware, tales como GPUs [182].

5.7.2. Particionado de mallas y grafos

El desarrollo de soluciones paralelas eficientes en muchas áreas de ciencia e ingeniería depende de algoritmos que particionen apropiadamente grafos no estructurados. Como ejemplo, podemos citar todas aquellas aplicaciones basadas en elementos finitos, las cuales requieren la distribución de la malla de elementos entre los elementos de proceso, como ocurre en esta tesis doctoral.

Dicha distribución se debe realizar de manera que el número de elementos asignados a cada partición sea lo más igualado posible, a fin de equilibrar la carga computacional y, a la vez, que el número de elementos adyacentes asignados en común a elementos de cómputo diferentes sea mínimo, reduciendo así las comunicaciones. Las técnicas de particionado de grafos, encargadas de satisfacer dichas condiciones, comienzan convirtiendo la malla de elementos finitos en un grafo, el cual particionan a continuación en partes iguales.

Alternativamente, dichos algoritmos son también válidos para reordenar matrices dispersas, a fin de reducir el llenado que se produce en su proceso de factorización a la hora de resolver un sistema de ecuaciones lineales. Dicha reordenación, que decide el orden en el cual se eliminan las filas de la matriz durante el citado proceso, reduce sustancialmente la cantidad de memoria requerida y el tiempo invertido en la resolución del sistema. Adicionalmente, el objetivo de la reordenación de la matriz es también el de equilibrar la carga computacional entre los elementos de procesamiento, en el momento de obtener su descomposición. Por último, cabe destacar que dichos algoritmos de particionado son además aplicables en la resolución de problemas de optimización.

Entre las librerías de particionado de grafos que están a nuestro alcance destacan METIS [183] y su implementación paralela ParMETIS [184], SCOTCH [185], y su versión paralela PT-SCOTCH [186, 187] o Zoltan [188].

5.7.2.1. METIS

METIS es un software de particionado de grafos irregulares, mallas no estructuradas y ordenación de matrices dispersas desarrollado en lenguaje C por la Universidad de Minnesota [183]. Los algoritmos aquí implementados están basados en el paradigma de particionado de grafos multinivel, el cual particiona un grafo de adyacencias, con posibles pesos en sus vértices y aristas, en un tiempo reducido.

El objetivo consiste en particionar el grafo en k subconjuntos disjuntos, de manera que la suma de todos los pesos de los vértices pertenecientes a cada uno de los subgrafos sea la misma para todos ellos y la suma de los pesos de las aristas cuyos vértices pertenecen a subconjuntos diferentes sea mínima.

Puede decirse que muchas implementaciones eficientes de aplicaciones paralelas en ingeniería pasan por llevar a cabo un problema de particionado de grafos, donde los vértices pueden representar tareas computacionales y las aristas implican una comunicación o un intercambio de datos entre los elementos de procesamiento. De este modo, asignaremos pesos a los vértices de manera proporcional a la cantidad de computación que conlleva la tarea a la que se asocia. De un modo

similar, los pesos de las aristas vienen reflejados por la cantidad de datos que necesitan ser intercambiados.

En consecuencia, si particionamos el grafo para asignar las distintas tareas a los k procesadores, el equilibrio de la carga quedará garantizado, puesto que las tareas asociadas a cada elementos de proceso presentarán el mismo peso. Por otro lado, ya que el algoritmo habrá minimizado el número de enlaces que unen los diferentes subconjuntos, la sobrecarga de comunicación será a su vez mínima.

A la hora de definir la malla de elementos finitos, METIS ofrece la flexibilidad necesaria para que pueda estar compuesta, conjuntamente, por elementos de diferente tipo. Inicialmente, la malla se convierte en un *grafo dual*, donde cada elemento finito se transforma en un vértice del grafo y dos vértices se unen si los elementos de los que provienen tienen algún nodo en común, o en un *grafo nodal*, donde cada nodo de la malla da lugar a un vértice del grafo y dos vértices se unen si los nodos pertenecen a un mismo elemento. Como resultado de salida, los algoritmos indican el índice de la partición a la que pertenece cada vértice del grafo.

La división del grafo en k particiones se puede obtener mediante los algoritmos de *Bisección Recursiva Multinivel* [189] o *Particionado k-way Multinivel* [190], siendo el segundo de ellos preferible en cuanto a la flexibilidad que proporciona su configuración. El objetivo es dar lugar a un conjunto de particiones donde el número de aristas del grafo que unen nodos que pertenecen a diferentes particiones sea mínimo. A este objetivo se le denomina habitualmente *corte de arista* (*edge-cut*) y representa una aproximación del coste real de comunicación en la etapa de simulación asociado al particionado.

En ambos casos, dichos particionados de grafos multinivel constan de 3 etapas: *agregación*, *particionado inicial* y *desagregación*, como muestra la figura 5.9. En la fase de agregación, se genera una secuencia de grafos cada vez más pequeños, a partir del grafo de entrada, agrupando juntos un conjunto de tamaño máximo de vértices adyacentes hasta alcanzar un grafo de unos pocos cientos de vértices. En esta etapa se emplean algoritmos tales como *Random Matching* o *Sorted Heavy-Edge Matching*.

Se lleva a cabo entonces la fase de particionado inicial, usando alguna apro-

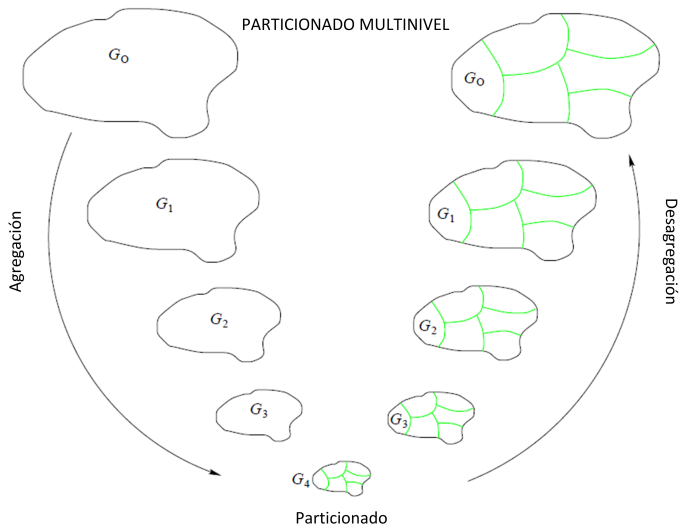


Figura 5.9: Etapas del particionado de grafos multinivel.

ximación como la desarrollada por Kernighan-Lin [191], minimizando el número de aristas cortadas. Como el tamaño del grafo es reducido, se requiere muy poco tiempo. Finalmente, se van generando un conjunto de grafos, cada vez de mayor tamaño, desagregando en una misma partición todos aquellos vértices que fueron agrupados previamente y refinando el particionado mediante métodos heurísticos [191, 192] que mueven vértices entre particiones a fin de mejorar la calidad de la solución final.

La ordenación de la matriz se realiza mediante el algoritmo de *Disección Anidada Multinivel* [189], el cual particiona el grafo en el que se convierte la matriz de entrada, donde cada vértice del mismo se corresponde con una fila de la matriz y donde dos vértices i y j se conectan si el elemento que ocupa dichas posiciones de fila i y columna j en la matriz es distinto de cero. Dicha técnica de ordenación de la matriz es equivalente a la reordenación de los nodos de la malla de elementos finitos.

Los algoritmos de Disección Anidada dividen el grafo de manera recursiva en dos mitades aproximadamente iguales, seleccionando un separador de vértices, obtenido a partir del separador de aristas del grafo, hasta que se alcanza el número de particiones deseadas. En cada paso, los nodos del separador de vértices son los últimos en numerarse. La eficiencia y la complejidad de estos algoritmos depende

del algoritmo que calcule dichos separadores. En general, pequeños separadores dan lugar a un llenado inferior. Dentro de los algoritmos más utilizados para particionar un grafo en dos, podemos mencionar a los algoritmos de Particionado de Grafos Multinivel, citados anteriormente.

Finalmente, conviene resaltar que METIS proporciona un API con un conjunto de funciones que pueden ser invocadas desde programas escritos en C, C++ o Fortran. Por otro lado, ofrece también un conjunto de programas autónomos que reproducen la funcionalidad del API y que trabajan a nivel de ficheros.

5.7.2.2. ParMETIS

ParMETIS es una librería paralela basada en MPI que implementa una variedad de algoritmos de particionado de mallas y grafos no estructurados y ordenación de matrices dispersas, todo ellos de gran tamaño [184]. Dichos algoritmos están basados en las mismas técnicas implementadas a nivel secuencial en METIS. Sin embargo, ParMETIS extiende la funcionalidad de METIS e incluye rutinas especialmente apropiadas para la simulación numérica en paralelo a gran escala, tales como mejorar la calidad de una partición ya existente o la repartición de un grafo correspondiente a una malla refinada adaptativamente.

La técnica de particionado de grafos empleado por ParMETIS en paralelo está basado en el algoritmo de Particionado k-way Multinivel, descrito en [189, 190] y paralelizado en [193, 194]. Se ofrecen además diferentes rutinas de particionado de grafos, provenientes de mallas de elementos finitos, a partir de las coordenadas de los nodos.

5.7.3. Resolución de sistemas de ecuaciones lineales

La resolución de grandes sistemas de ecuaciones lineales donde la matriz de coeficientes es dispersa supone la base para resolver muchos problemas de la ingeniería moderna. Habitualmente, las metodologías empleadas para llevar a cabo la resolución de este problema son diversas y pueden enmarcarse en dos grandes grupos: métodos directos [195, 196] y métodos iterativos [197, 198]. Es la propia

naturaleza del problema la que determina qué método resulta más apropiado.

5.7.3.1. Método directos

Un método directo que resuelve un sistema de ecuaciones lineales de la forma $Ax = b$ implica una factorización de la matriz A de coeficientes, de forma genérica mediante la descomposición LU, tal que $A = LU$, siendo L una matriz triangular inferior unidad y U una matriz triangular superior, además de la resolución de dos sistemas de ecuaciones triangulares correspondientes, de la forma $Ly = b$ seguido de $Ux = y$.

En diferentes áreas de aplicación, la matriz de coeficientes será simétrica y definida positiva, por lo que puede ser factorizada mediante la descomposición de Cholesky, donde $A = LL^T$, siendo L una matriz triangular inferior, resolviendo a continuación los sistemas $Ly = b$ y finalmente $L^T x = y$. Puesto que sólo hay que calcular una matriz triangular, el tiempo invertido en la descomposición de Cholesky es la mitad que el de la descomposición LU.

Una descomposición alternativa a la de Cholesky es la llamada descomposición LDL^T, en la cual ocurre que $A = LDL^T$, siendo L una matriz triangular inferior unidad y D una matriz diagonal. Algunas matrices simétricas indefinidas, para las cuales no existe la descomposición de Cholesky, podrán tener este tipo de descomposición, con elementos negativos en D. Por este motivo, esta descomposición es a veces preferida, con un coste computacional y un consumo de memoria idéntico al que presenta la descomposición de Cholesky. En este caso, el sistema $Ax = b$, equivalente a $LDL^T x = b$, se resolverá del siguiente modo. Inicialmente, supondremos que $y = L^T x$ y que $z = Dy$. A partir de ello, comenzamos resolviendo el sistema triangular inferior $Lz = b$, seguido de un sistema diagonal trivial $Dy = z$ para, finalmente, obtener la solución tras resolver un sistema triangular superior $L^T x = y$.

Aunque el tiempo empleado en dichas factorizaciones es altamente considerable, al igual que lo es la memoria RAM consumida, debido al llenado producido en la matriz de coeficientes que altera su patrón inicial de dispersidad, los métodos directos destacan por su alta aplicabilidad y su robustez. Cuando hablamos de

llenado, nos estamos refiriendo a todos aquellos elementos que son diferentes de cero en la matriz L o U como resultado de la factorización, pero que eran iguales a cero en la matriz A original. Lógicamente, eso supone que los factores LU, LL^T o LDL^T son mucho más densos que lo era la matriz de partida, lo cual conlleva un mayor número de operaciones aritméticas y un mayor consumo de memoria para almacenar dichos elementos, siempre teniendo en cuenta que, a la hora de trabajar con matrices dispersas, se deben emplear métodos de almacenamiento en memoria como el de fila comprimida o CSR (del inglés Compress Sparse Row) que únicamente guardan los elementos de la matriz no nulos.

El problema del llenado pretende ser aliviado, en cierta medida, con técnicas de ordenación de matrices, como el algoritmo de Mínimo Grado [199] o el de Disección Anidada Multinivel implementado en METIS o ParMETIS. Sin embargo, encontrar una ordenación de la matriz que de lugar a una cantidad de llenado mínimo es un problema NP-completo [200].

A nivel matemático, dicho ordenación viene expresada por una matriz P de permutación de manera que, ahora, el nuevo factor de Cholesky L se obtendrá como $PAP^T = LL^T$, siendo dicha matriz de permutación también de utilidad en la fase de resolución de los sistemas triangulares $Ly = Pb$, $L^T z = y$, permitiéndonos así obtener finalmente la solución x como $x = P^T z$.

En la práctica, estas factorizaciones han sido muy usadas en su versión secuencial y han sido implementadas eficientemente en arquitecturas paralelas con memoria distribuida para matrices densas, en librerías como ScaLAPACK. Sin embargo, la implementación paralela en dichas arquitecturas para matrices dispersas ha supuesto desde hace muchos años un verdadero desafío, que ha estado sujeto a una intensa tarea de investigación debida a sus altos requerimientos de memoria y elevados tiempos de computación, teniendo en cuenta el gran tamaño que pueden presentar los sistemas en muchos campos de aplicación [201, 202].

Con todo ello, también hay que decir que, en el caso de que el sistema de ecuaciones posea múltiples vectores partes derecha, los métodos directos serán muy efectivos, ya que la factorización se llevará a cabo una sola vez, resolviendo posteriormente diferentes tandas de sistemas triangulares. En el caso del análisis estructural, es habitual que el sistema de ecuaciones posea diferentes vectores

partes derecha, tanto en un análisis estático, trabajando con diferentes hipótesis básicas de carga, como en un análisis dinámico lineal a lo largo del tiempo, donde el vector de carga dinámica efectiva va variando para cada instante y donde se considere incluso la simulación de distintas cargas dinámicas.

Dada una matriz simétrica y definida positiva, existe un *árbol de eliminación* asociado con su factor de Cholesky L [203]. Dicho árbol está formado por un conjunto de n nodos, donde cada nodo representa una columna de la matriz L y existe un enlace entre dos nodos i y j, siendo $i > j$, si L_{ij} es el primer elemento no nulo, fuera de la diagonal, situado en la columna j. En ese caso, se considera que el nodo i es el padre del nodo j y el nodo j es el hijo del nodo i. Si algunos nodos numerados de manera consecutiva forman una cadena en el árbol de eliminación, y las correspondientes filas de L tienen una estructura idéntica, entonces a dicha cadena se le malla supernodo.

Una aplicación de estos árboles de eliminación la tenemos en el caso de los métodos supernodales y multifrontales que citaremos más adelante, siendo extremadamente útiles para determinar una asignación de las columnas a los elementos de cómputo, en el caso de trabajar con arquitecturas de memoria distribuida. Una columna j sólo puede ser eliminada después de que las columnas que vienen representadas por los descendientes del nodo j en el árbol de eliminación hayan sido eliminadas previamente. En una implementación paralela, la eliminación de columnas en diferentes ramas del árbol puede realizarse concurrentemente.

Un ejemplo de estrategia de asignación de la matriz a los elementos de proceso es la denominada *subárbol a subcubo* [204], acompañada de un particionado unidimensional o bidimensional de la matriz. Estas estrategias pretenden distribuir la carga apropiadamente y reducir las comunicaciones. Aunque inicialmente se diseñó específicamente para arquitecturas en forma de hipercubo, el algoritmo es perfectamente aplicable al resto de arquitecturas de memoria distribuida.

A modo de resumen, las cuatro etapas de las que consta la resolución del sistema considerado mediante la factorización de Cholesky son las siguientes. Si bien la factorización numérica es la fase más costosa temporalmente, es necesario paralelizar también el resto de etapas para mantener la escalabilidad del proceso completo y evitar problemas de memoria en un solo procesador:

- *Ordenación*: El objetivo es encontrar una buena matriz de permutación P para que el factor de Cholesky L, obtenido como $PAP^T = LL^T$ o $PAP^T = LDL^T$ en la etapa de la factorización numérica, presente un llenado mínimo, reduciendo así el número de operaciones aritméticas y los requerimientos de memoria. A la vez, dicha matriz de permutación determina el grado de paralelismo en la factorización. Gran parte de las prestaciones que se obtengan en la resolución paralela del sistema dependerán, en buena medida, del algoritmo de ordenación utilizado.

  Algoritmos secuenciales por naturaleza como Cuthill-McKee o Grado Mínimo producen muy buenas ordenaciones, pero dan lugar a árboles de eliminación con bastante altura, que no están equilibrados, limitando el incremento de velocidad que puede obtenerse en la implementación paralela de la factorización numérica. En cambio, otras técnicas como la de Disección Anidada generan árboles de eliminación con menor altura que están bien balanceados, reduciendo el tiempo completo de la factorización numérica a medida que el número de elementos de procesamiento aumenta.

- *Factorización simbólica*: Obtiene la estructura de L, con las posiciones de sus elementos no nulos, y reserva espacio en memoria. Esta factorización es básica para incrementar las prestaciones de la fase siguiente. Puede llevarse a cabo en paralelo eficientemente usando la misma distribución de los datos a los elementos de cómputo que la empleada en la factorización numérica. La estructura de la matriz L se crea de abajo a arriba. Primero, se determina la estructura de elementos no nulos de los nodos hoja y el resultado se envía hacia arriba en el árbol a los elementos de proceso que almacenan el supernodo del nivel superior. Estos elementos obtienen la estructura de elementos no nulos de su supernodo y la unen con la estructura recibida de sus nodos hijos.

- *Factorización numérica*: Obtiene el factor de Cholesky L tal que $PAP^T = LL^T$ o tal que $PAP^T = LDL^T$. Entre las múltiples técnicas posibles, puede llevarse a cabo mediante la factorización de Cholesky *multinodal* [201, 205] o *multifrontal* [201, 206, 207, 208]. El método multinodal presenta dos alternativas: *right-looking o fan-out* [209] y *left-looking o fan-in* [210]. La diferencia entre ellos radica en que están basados en diferentes versiones

del algoritmo original de la factorización de Cholesky, además de emplear distintas distribuciones de los datos y las tareas a realizar entre los elementos de cómputo, si hablamos de su implementación paralela sobre máquinas de memoria distribuida. En realidad, tanto los métodos multinodales como el método multifrontal son algoritmos que tratan de aprovechar los datos, tanto como sea posible, cuando están en la memoria caché.

La variante *right-looking* trabaja con una fila y columna, o un bloque de las mismas, en cada paso y las usa inmediatamente para actualizar todas las filas y columnas de la matriz que todavía no se han factorizado. En cambio, en la variante *left-looking*, las actualizaciones no se aplican inmediatamente, sino que una columna, o bloque de columnas, se elimina cuando se le aplican todas las actualizaciones de las columnas previas de L. Existen también versiones híbridas que combinan ambas variantes llamadas *left-right looking*.

El *método frontal*, precursor del método multifrontal, es una variante de los métodos de factorización, presentado inicialmente por Irons en el año 1970 [211] bajo la necesidad de resolver, en máquinas que poseían poca memoria, grandes sistemas dispersos de ecuaciones resultantes de la aplicación del método de los elementos finitos al análisis estructural. Irons observó que no es necesario calcular toda la matriz para luego factorizarla, sino que se puede realizar un secuencia de factorizaciones a medida que cada fila y columna se van completando. En ese sentido, el método frontal se basa en ir trabajando en memoria con bloques, o frentes, de la matriz, en lugar de trabajar con la matriz completa.

Una de las principales extensiones de estos métodos frontales vino por parte de Duff y Reid [207] en los denominados métodos multifrontales, los cuales están diseñados para trabajar con varios bloques o frentes simultáneamente y de forma independiente, calculando la factorización de un frente y guardando los aportes del mismo a otros frentes hasta que sea necesario utilizarlos. En estos métodos multifrontales, a cada nodo del árbol de eliminación se le asocia la factorización de una matriz frontal pequeña y densa, que podrá ser factorizada cuando se hayan factorizado todas las matrices frontales de los nodos hijos de los que depende.

La principal ventaja de todos los métodos mencionados es que trabajan con

matrices densas y pueden ser factorizadas eficientemente mediante rutinas de BLAS de nivel 3. Además, las referencias a memoria son más eficientes que las realizadas con otros métodos, explotando los accesos a memoria caché.

Tanto en los métodos multifrontales como en los supernodales, los diferentes nodos del árbol de eliminación, donde cada uno se corresponde con una matriz densa, se asignan a los elementos de procesamiento, de modo que diferentes ramas del mismo pueden procesarse en paralelo y sin ninguna sobrecarga de comunicación, estableciendo dependencias de datos y comunicaciones de abajo a arriba. Al paralelismo establecido se le denomina paralelismo a grano grueso. Cuando las dimensiones de los submatrices correspondientes a un nodo son importantes, se pueden dividir en distintos bloques, resolviendo cada uno de ellos en paralelo en el denominado paralelismo de grano medio. Finalmente, para cada bloque se llevan a cabo operaciones con BLAS de nivel 3, las cuales también pueden ser paralelizadas bajo un paralelismo de grano fino con ScaLAPACK.

Tanto los algoritmos multinodales como los multifrontales están detrás de muchos códigos comerciales o de dominio público y han demostrado ser eficientes y escalables aplicados en la resolución de sistemas de ecuaciones lineales dispersos, simétricos y definidos positivos sobre plataformas con memoria distribuida.

- *Solución triangular*: Obtiene la solución del sistema de ecuaciones, tras resolver inicialmente el sistema triangular inferior $Ly = Pb$, mediante el algoritmo de *Sustitución Directa*, seguido del sistema triangular superior $L^T z = y$, mediante la técnica de *Sustitución Inversa*. Para finalizar, la solución del sistema se obtiene como $x = P^T z$.

  Con el objetivo de que no se convierta en el cuello de botella, esta etapa debe ser resuelta también de manera eficiente en paralelo. Aunque el desafío en desarrollar un algoritmo paralelo escalable se encuentra en la secuencialidad de la dependencia de datos y en la poca cantidad de computación asignada a los elementos de procesamiento, dicha dependencia puede ser dotada de concurrencia si se sigue una distribución apropiada de la matriz entre los mismos. Al igual que en el caso de la factorización numérica, los algoritmos

pueden estar organizados en términos de operaciones con matrices densas.

La implementación paralela del algoritmo de Sustitución Directa es muy similar a la seguida en la factorización numérica, puesto que se sigue la misma estrategia subárbol a subcubo de asignación de nodos del árbol de eliminación a los elementos de cómputo. Dicha sustitución comienza con los supernodos del árbol que representan el nivel inferior, para ir ascendiendo y finalizar en el supernodo raíz. Una vez resueltos los subárboles de más bajo nivel, cada supernodo sucesivamente ejecuta una operación de suma extendida, donde se recoge los resultados obtenidos de sus supernodos hijos, seguida de una sustitución directa trapezoidal densa en paralelo. Esta distribución asegura que, como mucho, tendrán lugar dos operaciones de comunicación punto a punto entre pares de elementos de procesamiento.

La implementación del algoritmo paralelo de Sustitución Inversa es similar, pero cada elemento de proceso recorre su parte del árbol supernodal de arriba a abajo. En cada supernodo de los niveles más altos se ejecuta un algoritmo de Sustitución Inversa para matrices densas en paralelo y los vectores solución calculados en los supernodos padres son propagados a los supernodos hijo. Este proceso continua con una comunicación punto a punto como mucho entre pares de elemento de cómputo.

5.7.3.2. Métodos iterativos

A partir de una solución inicial, los métodos iterativos generan una sucesión de soluciones aproximadas con la esperanza de alcanzar la solución del problema [197]. Sin embargo, el principal inconveniente que presentan es que pueden no converger a la solución o que el número de iteraciones necesarias, y en consecuencia el tiempo de respuesta asociado, sea excesivamente elevado.

No obstante, estos métodos no modifican la matriz de coeficientes en la resolución del sistema de ecuaciones, motivo por el cual consumen menos memoria y están basados, principalmente y para cada iteración, en el producto de una matriz dispersa por un vector y en diferentes operaciones con vectores, lo cual supone que sean altamente paralelizables [212].

Se distinguen dos tipos de métodos iterativos, los estacionarios y los no estacionarios. Los métodos estacionarios son más antiguos y más sencillos de comprender e implementar, pero habitualmente presentan una velocidad de convergencia menor. Entre dichos métodos estacionarios se encuentran los métodos de Jacobi, Gauss-Seidel, Sobrerrelajación Sucesiva (SOR) y Sobrerrelajación Sucesiva Simétrica (SSOR). Los métodos no estacionarios, mucho más efectivos, incluyen métodos basados en el uso de los subespacios de Krylov [213] y métodos no basados en dichos subespacios, como el método de Chebyshev.

El subespacio de Krylov $\mathcal{K}_m(A, r_0)$ de dimensión m, asociado con un sistema lineal $Ax = b$, para un vector inicial x_0 y un vector residuo $r_0 = b - Ax_0$, se define como el subespacio generado por los vectores r_0, Ar_0, A^2r_0, .., $A^{m-1}r_0$. Entre dichos métodos, no estacionarios y basados en los subespacios de Krylov, encontramos los siguientes:

- *Gradiente Conjugado (CG)*: Su nombre proviene del hecho de generar una secuencia de vectores conjugados (o ortogonales) que son los residuos de las iteraciones. Dichos vectores son además los gradientes de una función cuadrática, cuya minimización es equivalente a resolver el sistema de ecuaciones. Se trata de un método muy efectivo cuando la matriz es simétrica y definida positiva, a lo cual hay que añadir que sólo necesita almacenar un número limitado de vectores en la secuencia citada.

- *Residuo Mínimo Generalizado (GMRES)*: Obtiene una secuencia de vectores ortogonales pertenecientes al subespacio de Krylov, los cuales se combinan y actualizan por medio de la resolución de un problema de mínimos cuadrados. Requiere almacenar la secuencia completa de vectores, lo que conlleva un consumo elevado de memoria. Por ese motivo, han surgido versiones con reinicio, las cuales sólo almacenan un número determinado de vectores. Es un método muy útil para matrices no simétricas.

- *Gradiente BiConjugado (BiCG)*: Este método genera dos secuencias de vectores, una con la matriz A y otra con A^T, que se ortogonalizan conjuntamente. Es aplicable cuando la matriz no es simétrica y no requiere almacenar todos los vectores aunque su convergencia puede ser irregular, con oscilaciones que dan lugar a criterios de parada con condiciones erróneas.

- Residuo Mínimo (MINRES) y LQ Simétrico (SYMMLQ): Dichos métodos son alternativas al método de CG aplicables cuando al matriz sea simétrica pero posiblemente indefinida. El método SYMMLQ genera las mismas iteraciones que CQ si la matriz es simétrica y definida positiva.

- *Residuo Cuasi-Mínimo (QMR)*: Mejora al método BiCG al suavizar su convergencia irregular e incrementar su robustez, lo que conduce a soluciones más fiables. Para ello, resuelve un problema de mínimos cuadrados y actualiza los residuos del método BiCG.

- *Gradiente BiConjugado Estabilizado (BiCGSTAB)*: Es una variante del método BiCG que usa diferentes actualizaciones en la secuencia de vectores obtenida a partir de A^T, a fin de suavizar y mejorar su convergencia.

- *Gradiente Conjugado Cuadrático (CGS)*: Es una variante del método BiCG que no necesitan generar la secuencia de vectores con la matriz A^T. Sin embargo, la convergencia puede ser incluso más irregular que la del método BiCG.

- *Residuo Casi Mínimo Sin Transpuesta (TFQMR)*: Es una variante del método CGS, la cual utiliza todos los vectores disponibles en la dirección de búsqueda. Además, los vectores se combinan mediante un parámetro obtenido mediante una cuasi minimización del residuo. El método es extremadamente robusto y no requiere el cálculo de productos matriz-vector con la matriz A^T.

La efectividad de dichos métodos no estacionarios para un problema dado, en lo que a velocidad de convergencia se refiere, depende de las propiedades de la matriz de coeficientes, por ejemplo que sea simétrica y definida positiva o que no lo sea, además de la distribución de sus valores propios.

Para acelerar dicha velocidad de convergencia, se usan los denominados *precondicionadores* de la matriz, los cuales modifican la matriz de coeficientes y, en consecuencia, su número de condición, sin modificar el resultado del sistema, lo que implica una reducción notable en el número de iteraciones asociadas a su resolución. La idea de usar un método iterativo con un precondicionador M es convertir al sistema inicial $Ax = b$ en otro sistema equivalente $M^{-1}Ax = M^{-1}b$,

el cual converge más rápidamente puesto que las propiedades espectrales de la nueva matriz de coeficientes serán más favorables, pero conlleva la resolución de un sistema de ecuaciones de la forma $My = z$ que debe ser resuelto una o más veces para cada iteración. Consecuentemente, una de las restricciones del uso de precondicionadores es que este último sistema sea fácilmente resoluble.

Esto ha dado lugar a dos tipos de métodos encargados de obtener la matriz M. Por un lado, están todos aquellos que construyen explícitamente la inversa de la matriz M, de modo que el sistema $My = z$ se convierte en un producto matriz por vector. Por otro lado, estarán aquellos métodos basados en una factorización LU o Cholesky incompleta [214] de la matriz A. Se trata de factorizar la matriz pero sin introducir todo el llenado que se produce durante el proceso, lo cual equivale a que $A = LU - R$ o $A = LL^T - R$. Obtenida dicha factorización, habrá que resolver los dos sistemas triangulares asociados.

El precondicionador que hemos visto es el llamado precondicionador por la izquierda. Cabe también la posibilidad de emplear el denominado precondicionador por la derecha, en el cual el sistema $Ax = b$ se convierte en $AM^{-1}y = b$, donde $y = Mx$. Incluso, es también posible aplicar el precondicionador por los dos lados, el cual no es más que una combinación de ambos, de modo que el sistema se transforma en $M_1^{-1}AM_2^{-1}M_2x = M_1^{-1}b$.

No obstante, el esfuerzo que conlleva determinar y calcular un buen precondicionador de la matriz puede ser casi tan elevado como el coste temporal de la factorización directa. Además, las prestaciones obtenidas en la paralelización de precondicionadores como la factorización LU o Cholesky incompleta no han sido tradicionalmente todo lo buenas que sería deseable.

Un precondicionador más sencillo de implementar y más escalable a nivel paralelo en una máquina con memoria distribuida, aunque menos eficiente, es el de *Jacobi a Bloques* [197], en el cual la matriz M es una matriz diagonal a bloques, donde cada bloque tiene un tamaño igual al número de filas asignadas a cada elemento de procesamiento y está compuesto por los elementos de la diagonal de la matriz A.

En el caso que nos ocupa, cabe decir que las matrices de rigidez de una estructura tienen habitualmente un número de condición muy alto, lo cual desaconseja

el uso de métodos iterativos en la resolución del sistema de ecuaciones que conlleva el cálculo de los desplazamientos en los nudos de la estructura, debido al elevado número de iteraciones y, en consecuencia, a los excesivos tiempos de respuesta.

5.7.3.3. Software numérico disponible

Son muchos los desarrollos de software relativos a la resolución de sistemas de ecuaciones lineales dispersos mediante métodos directos o iterativos, bien sean códigos comerciales o de código abierto. En [215] se encuentra disponible un repositorio actualizado de software de código abierto dedicado a la resolución de problemas de algebra lineal, incluyendo diferentes núcleos computacionales, resolución de sistemas de ecuaciones lineales, precondicionadores y resolución de problemas de valores propios. A continuación describiremos algunas de esas implementaciones, atendiendo a un conjunto de requerimientos. Aunque en el caso de la librería WSMP no se cumple, es deseable que el software pueda ser invocado desde PETSc (Portable Extensible Toolkit for Scientific Computing) [13], gracias al desarrollo de un interfaz (wrapper) específico. Esta conveniencia viene dada teniendo en cuenta que los desarrollos de esta tesis doctoral están basados en dicho entorno software. Los requerimientos a los que hacíamos alusión eran:

- Que se haya publicado alguna actualización en los últimos 3 años, entendiendo de ese modo que se siguen manteniendo.
- Que estén disponibles tanto para máquinas de memoria compartida como de memoria distribuida.
- Que resuelvan un sistema de ecuaciones donde la matriz sea dispersa y, al menos, simétrica y definida positiva, como ocurre en el caso del análisis estructural.

MUMPS

MUMPS (MUltifrontal Massively Parallel Solver) [14] es un paquete software, implementado en Fortran 90 y en C, de resolución de sistemas de ecuaciones lineales de la forma $Ax = b$, donde A es una matriz dispersa que puede ser simétrica general, simétrica y definida positiva, o no simétrica. MUMPS implementa

un método directo basado en una aproximación multifrontal [216] que lleva a cabo una factorización Gaussiana $A = LU$, si la matriz no es simétrica, o $A = LDL^T$, si la matriz es simétrica, siendo L y U matrices triangular inferior y superior, respectivamente, y D una matriz diagonal.

De acuerdo a lo descrito en la sección 5.7.3.1, el sistema se resuelve en tres etapas: análisis, que incluye la ordenación y la factorización simbólica, factorización numérica y resolución de los sistemas triangulares. A la hora de ordenar la matriz de coeficientes para reducir el llenado, incorpora diferentes métodos de reordenación como QAMD [217], aunque también puede invocar a librerías externas como PORD [218], SCOTCH, PT-SCOTCH, METIS o ParMETIS. Incluso el propio usuario puede proporcionar dicha ordenación.

MUMPS emplea una aproximación completamente asíncrona en las comunicaciones, para solapar computación y comunicaciones entre las tareas de cómputo [219]. A su vez, y en tiempo de ejecución, emplea una planificación dinámica de trabajo y datos para balancear la carga entre los procesos en la etapa de la factorización numérica [220]. En su etapa final, emplea un refinamiento iterativo para mejorar la solución obtenida tras resolver el sistema de ecuaciones.

Puede ejecutarse en máquinas de memoria distribuida, memoria compartida o en máquinas híbridas. Su implementación puramente secuencial depende únicamente de BLAS. Por el contrario, su implementación paralela sobre una arquitectura de memoria distribuida depende de MPI, BLAS, BLACS y ScaLAPACK. Adicionalmente, MUMPS incorpora directivas de OpenMP en su código [221], a fin de ejecutarse eficientemente en máquinas de memoria compartida y en máquinas híbridas de memoria compartida distribuida, combinado con MPI [222]. Para obtener las mejores prestaciones, la librería de BLAS utilizada también debería ser multihilo y estar basada en OpenMP. MUMPS puede ser utilizado desde PETSc.

PaStiX

PaStiX [223, 224] es una librería paralela multihilo que combina MPI y Pthreads en la resolución de un sistema de ecuaciones lineales disperso y de gran dimensión, sobre una plataforma computacional de memoria compartida distribuida. Si la matriz A de coeficientes es simétrica y definida positiva, podemos usar la

descomposición de Cholesky $A = LL^T$, o la descomposición $A = LDL^T$ cuando la matriz sea simétrica, en ambos casos con o sin pivotación numérica. Si la matriz no es simétrica, emplearemos la descomposición LU, $A = LU$, con pivotación.

Dada una máquina de memoria compartida distribuida, el modo por defecto de ejecutar PaStiX equivale a lanzar un proceso MPI por nodo y tantos hilos de ejecución como núcleos computacionales tenga el nodo. En cualquier caso, es posible ejecutar PaStiX sin MPI, siguiendo un paradigma único de memoria compartida, o sin hilos de ejecución, bajo un modelo de programación puro de memoria distribuida.

Para reordenar la matriz de coeficientes, se utiliza SCOTCH, PT-SCOTCH o METIS, generando a partir de ahí el árbol de dependencias para distribuir la matriz entre los elementos de cálculo por bloques de columnas de una dimensión. Se admiten incluso ordenaciones propuestas por el usuario. Si se emplea SCOTCH, el algoritmo de ordenación es el de disección anidada, en el cual el grafo se particiona en dos mediante un separador de vértices de tamaño mínimo. A los nodos del separador se les asigna el índice de numeración más alto y el algoritmo de partición se repite para cada subgrafo, hasta alcanzar un tamaño determinado, momento en el cual se emplea la técnica de Grado Mínimo Aproximado con Halo para numerar los nudos, asignando los números más pequeños a los nodos con los grados más bajos.

A la hora de factorizar la matriz, se emplea un método supernodal y, dentro de sus posibles variantes (*left-looking* o *right-looking*), implementa la técnica de left-looking, la cual es más eficiente en paralelo, ya que tiene la ventaja de reducir considerablemente el volumen de comunicación [225, 226].

PaStiX depende únicamente de BLAS. Como ya comentamos, las técnicas supernodales están basadas en matrices densas, empleando rutinas de BLAS de nivel 3 para mejorar la eficiencia.

PaStiX incorpora también una versión out-of-core experimental que trabaja únicamente con hilos, no con MPI, y que utiliza un dispositivo de almacenamiento secundario para factorizar la matriz, a fin de reducir sustancialmente la cantidad de memoria necesitada. La solución del sistema de ecuaciones puede ser refinada empleando un método iterativo, como GMRES o Gradiente Conjugado. Por

último, hay que destacar que se incluye una factorización ILU(k) adaptativa y orientada a bloques que puede usarse como precondicionador en paralelo [227].

La librería puede utilizarse desde PETSc.

PARDISO

PARDISO [228, 229] es un software de resolución de grandes sistemas de ecuaciones lineales dispersos y de gran dimensión en máquinas de memoria compartida y distribuida. Permite resolver sistemas donde la matriz es no simétrica, simétrica en general o simétrica y definida positiva, llevando a cabo las factorizaciones $A = LU$, $A = LDL^T$ o $A = LL^T$, respectivamente, en multiprocesadores de memoria compartida, mediante OpenMP, y distribuida, mediante MPI. Está escrito en C y en Fortran 77 y, su versión del año 2006, forma parte de la librería MKL de Intel, en la cual únicamente se ofrecía una paralelización a nivel de memoria compartida y que puede utilizarse desde PETSc. Como reordenación de la matriz, emplea el algoritmo de mínimo grado [230] o el disección anidada multinivel [189] mediante METIS o ParMETIS. En cuanto a la factorización numérica, implementa una combinación híbrida de las variantes supernodales left-looking y right-looking junto con rutinas de BLAS de nivel 3 [231, 232, 233, 234, 235]. Incorpora también técnicas de refinamiento iterativo de la solución.

PARDISO incluye también un método iterativo multirrecursivo de resolución de sistemas lineales simétricos indefinidos y un precondicionador para sistemas no simétricos que combina métodos directos e iterativos para acelerar la convergencia.

WSMP

Watson Sparse Matrix Package (WSMP) es una colección de algoritmos de resolución eficiente de grandes sistemas dispersos de ecuaciones lineales. Puede usarse en secuencial, en un entorno multiprocesador de memoria compartida, basado en Pthreads, o en uno de memoria distribuida, bajo el paradigma de paso de mensajes mediante MPI, donde cada proceso puede ser secuencial o multihilo. Este software está formado por las tres siguientes partes, encargadas de:

- La resolución de los sistemas mediante métodos directos donde la matriz

es simétrica, mediante la descomposición de Cholesky o la descomposición LDL^T [236].

- La resolución de sistemas mediante métodos directos donde la matriz es no simétrica, mediante la descomposición LU [237].
- La resolución de sistemas mediante métodos iterativos. En este caso, únicamente se ejecutará en máquinas de memoria compartida [238].

La ordenación se lleva a cabo en paralelo mediante el método de disección anidada multinivel [239]. En el caso de que la matriz sea simétrica y definida positiva, WSMP emplea una versión modificada del algoritmo de la factorización de Cholesky multifrontal [240, 241]. Bajo una plataforma de memoria compartida, todos los hilos se asignan a la raíz del nodo del árbol de eliminación, o al subárbol perteneciente al proceso MPI. A partir de ahí, y recursivamente, los hilos de cada padre se van asignado a su hijos, a fin de equilibrar la carga. Los sistemas triangulares también se resuelven en paralelo [242]. El software se encapsula en dos librerías: WSMP, que incluye la versión secuencial o multihilo para un único proceso, con capacidad de ejecutarse en una máquina de memoria compartida y la librería PWSMP, que incluye la versión paralela para máquinas de memoria distribuida y donde cada proceso MPI puede desplegar múltiples hilos. Requiere de una versión de BLAS que no se ejecute como multihilo. Desafortunadamente, no existe un interfaz en PETSc desde el cual podamos utilizarlo.

hypre

Hypre (High Performance Preconditioners) [243] es un software que incluye métodos iterativos y precondicionadores para la resolución de sistemas de ecuaciones lineales, dispersos y de gran dimensión, sobre plataformas de memoria distribuida. Parte de código es multihilo, con el objetivo de mejorar las prestaciones en máquinas con memoria compartida.

Ofrece al usuario cuatro interfaces diferentes para interaccionar con la librería: *Structured-Grid System Interface* (Struct), *Semi-Structured-Grid System Interface* (SStruct), *Finite Element Interface* (FEI) y *Linear-Algebraic System Interface* (IJ). El usuario debería elegir cuál de ellos se adecúa mejor a su aplicación, ya que ello le permitirá usar unos métodos u otros.

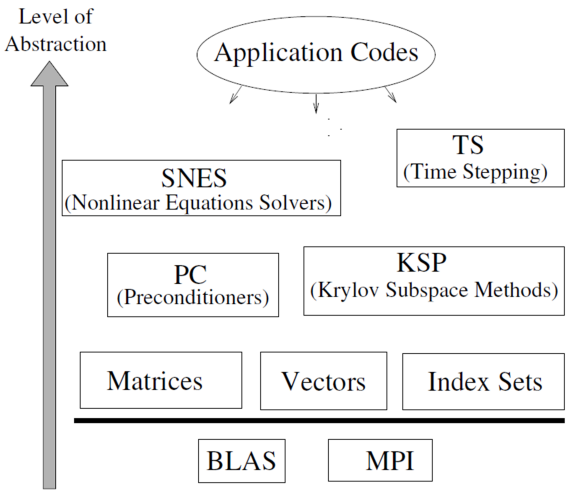


Figura 5.10: Organización de las librerías que componen PETSc.

Escrito en C, a excepción del interfaz FEI que está escrito en C++, depende de MPI, OpenMP y de BLAS. Como métodos iterativos de Krylov paralelizados sobre memoria distribuida incluye, entre otros, Gradiente Conjugado, GMRES, FlexGMRES, LGMRES o BiCGSTAB. Como antes hemos comentado, ofrece múltiples precondicionadores, algunos de ellos basados en la factorización incompleta como Euclid [244] y PILUT, basados en inversas aproximadas dispersas, como ParaSails [245], o basados en multigrid [246] como SMG, PFMG, SysPFMG o BoomerAMG [247], este último con capacidad de comportarse también como precondicionador. Existe un interfaz desde PETSc que permite su uso.

5.7.4. El paquete PETSc

PETSc (Portable Extensible Toolkit for Scientific Computing) [13] es un paquete software para la resolución numérica de ecuaciones en derivadas parciales y problemas relacionados en computadores de altas prestaciones. Proporciona un conjunto de funciones y de estructuras de datos que constituyen los bloques básicos para la implementación de aplicaciones tanto secuenciales como paralelas. En la figura 5.10 podemos observar los principales componentes de PETSc. Como vemos, usa MPI como estándar en la comunicación entre los procesos mediante paso de mensajes y BLAS como núcleo computacional.

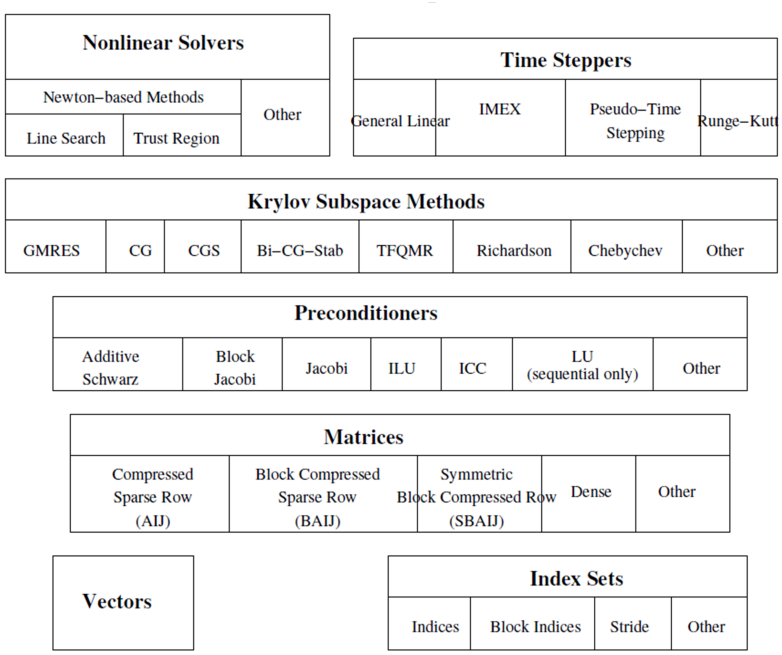


Figura 5.11: Componentes numéricos de PETSc.

A pesar de estar escrito en C, ofrece técnicas de programación orientada a objetos, las cuales proporcionan una flexibilidad notable a los usuarios. El hecho de estar organizado de manera jerárquica, permite que el usuario emplee el nivel de abstracción más apropiado de acuerdo al problema que pretende resolver. PETSc proporciona rutinas de ensamblado y de álgebra lineal para trabajar en paralelo con vectores y matrices. Además, incluye un conjunto de métodos, recogidos en la figura 5.11, que pueden ser invocados desde aplicaciones escritas en Fortran, C, C++, Python y Matlab para resolver:

- Sistemas de ecuaciones lineales mediante las descomposiciones LU o Cholesky en secuencial.

- Sistemas de ecuaciones lineales dispersos mediante métodos iterativos en paralelo, como Chebyshev, CG, BiCG, BiCGSTAB, GMRES y algunas variantes como FGMRES y DGMRES, TFQMR, etc. Se incluyen además, precondicionadores en paralelo, como los de Jacobi, Jacobi a Bloques, Multigrid Algebraico, Additive Schwarz, etc. o en secuencial, como Cholesky Incompleto (ICC), LU incompleto (ILU) o SOR.

- Sistemas de ecuaciones no lineales, muchos de ellos basados en Newton-Raphson o métodos de Krylov no lineales, como CG o GMRES.
- Ecuaciones diferenciales ordinarias y algebraicas, mediante métodos como Runge-Kutta.

Con todo ello, una característica muy apreciada de PETSc es que ofrece al usuario la posibilidad de utilizar software externo, incrementando su funcionalidad. Así por ejemplo, entre otros, presenta un interfaz a:

- Núcleos computacionales, como BLAS, LAPACK o Elemental [248].
- Paquetes de generación de mallas, como Triangle [249] y TetGen [250].
- Librerías de particionado de grafos, como METIS, ParMETIS, Chaco, Party [251], PT-SCOTCH o Zoltan.
- Librerías de resolución de sistemas de ecuaciones mediante métodos directos, como MUMPS, PaStiX, PARDISO, SuperLU [252] o ESSL [253].
- Librerías de resolución de sistemas de ecuaciones lineales mediante métodos iterativos y precondicionadores, tales como hypre, SPAI [254] o TRILINOS/ML [255].
- Software de resolución de ecuaciones diferenciales ordinarias, como CVODE de Sundials [256].
- Otro tipo de software: ADIFOR [257], MATLAB, Mathematica, etc.

5.7.5. El cálculo de valores propios

5.7.5.1. Introducción

La necesidad de resolver problemas de valores propios es habitual en numerosas áreas de ciencia e ingeniería. En su formulación estándar, un problema de valores propios se escribe como:

$$Ax = \lambda x \tag{5.3}$$

donde $A \in \mathbb{C}^{nxn}$ es una matriz real o compleja, $\lambda \in \mathbb{C}$ es un valor propio y $x \in \mathbb{C}^{nx1}$ es un vector propio. Si la matriz es simétrica y real (o Hermitiana y compleja) entonces la ecuación (5.3) se cumple para n pares (λ_i, x_i), todos los valores propios λ_i son reales y los n vectores propios x_i son ortogonales entre sí, lo cual significa que $x_i^T x_j = 0$ para $i \neq j$. Es habitual normalizar los vectores propios calculados de manera que $x_i^T x_i = 1$. Al conjunto de todos los valores propios de una matriz se le denomina *espectro*.

Si expresamos el problema en su forma generalizada, tendremos que:

$$Ax = \lambda Bx \tag{5.4}$$

siendo B una matriz cuadrada también de tamaño n x n. En el caso del análisis estructural que nos ocupa, y como ya vimos en la ecuación (4.94), ocurre que las matrices A y B del problema de valores propios generalizado a resolver son respectivamente las matrices de rigidez y masa, ambas dispersas, formadas por números reales y definidas positivas. En este contexto, los valores propios λ_i, es decir, las frecuencias naturales elevadas al cuadrado, son reales y positivos y los vectores propios x_i asociados, o formas modales, son ortogonales con respecto a la matrices A y B o, lo que es lo mismo, $x_i^T A x_j = 0$ y $x_i^T B x_j = 0$ para $i \neq j$. Habitualmente, los vectores propios se normalizan de forma que $x_i^T B x_i = 1$.

Recordemos además que, en dicho contexto, sólo era necesario calcular unos pocos de aquellos valores propios más pequeños, junto con sus correspondientes vectores propios. Sin embargo, si la estructura a resolver tiene una gran dimensión, es de vital importancia ajustar, en la medida que sea posible, la cantidad de modos de vibración a calcular, a fin de reducir el tiempo de cálculo y la memoria requerida para almacenar los vectores calculados. Hay que tener en cuenta que, aunque las matrices de rigidez y masa sean dispersas, los vectores propios calculados serán densos y su almacenamiento puede convertirse en un problema.

A continuación se describen brevemente diferentes técnicas y métodos de resolución de problemas de valores propios.

5.7.5.2. Métodos de resolución

Aunque los algoritmos numéricos de resolución de problemas de valores propios se desarrollaron hace ya muchos años, este problema conlleva todavía una actividad investigadora importante y altamente relevante dentro del álgebra lineal numérica [258, 259].

En particular, en el contexto de grandes problemas dispersos, se han propuesto muchos métodos para obtener buenas aproximaciones de un subconjunto de la solución con un coste computacional moderado, preservado la dispersidad de la matriz. En esta subsección describiremos brevemente algunos de esos métodos. Para obtener más información, pueden consultarse los monográficos dedicados a esta materia, tales como [260, 261], o algunos incluso con métodos más específicos dentro del análisis estructural [262].

Los métodos de resolución de problemas de valores propios dispersos preservan la dispersidad de la matriz A y están basados en productos matriz por vector, fácilmente paralelizables. Si tratamos de resolver un problema generalizado, como en la ecuación (5.4), la mayoría de los métodos requerirán su transformación al problema estándar. Por ejemplo, si B es no singular, el problema se formula como:

$$Cx = \lambda x \tag{5.5}$$

siendo $C = B^{-1}A$. De manera alternativa, si B es una matriz real simétrica y definida positiva, puede ser factorizada mediante la descomposición de Cholesky, tal que $B = LL^T$. Eso supone que el problema queda expresado ahora como:

$$Cy = \lambda y \tag{5.6}$$

donde $C = L^{-1}AL^{-T}$ es una matriz simétrica, junto con $y = L^T x$. De este modo, si λ e y son respectivamente un valor y vector propio de la ecuación (5.6) entonces λ junto con $L^{-T}y$ lo son a su vez del problema original. Habitualmente a C se le denomina *operador matricial*. Obsérvese que, en ambos casos, la matriz C no se debería construir explícitamente, sino que cada vez que se multiplique dicha matriz por un vector, como por ejemplo $y = L^{-1}AL^{-T}x$, seguiremos estos pasos:

1. Resolvemos un sistema triangular superior $L^T u = x$.
2. Multiplicamos $v = Au$.
3. Resolvemos un sistema triangular inferior $Ly = v$.

Una aproximación diferente supone formular el problema de este otro modo:

$$Cx = \theta x \tag{5.7}$$

siendo C aquel operador matricial tal que los vectores propios permanecen inalterados con respecto al problema original pero los valores propios han cambiado. La explicación a ello es la siguiente. Todos los métodos que se describen en la literatura son iterativos por naturaleza y su convergencia depende de las propiedades espectrales, o distribución de los valores propios, del operador matricial. En particular, los valores propios extremos y bien separados son los primeros en converger. En muchos casos, ocurre que los valores propios requeridos no están bien separados o forman parte del interior del espectro. En esos casos, la convergencia puede ser inaceptablemente lenta. La alternativa a emplear en dichas situaciones, habitualmente denominada *transformación espectral*, consiste en elegir un operador matricial tal que los valores propios se sitúen en posiciones más favorables, mientras que los vectores propios permanecen inalterados. Dicho de otro modo, pretendemos que los valores propios deseados del problema original se correspondan con los valores extremos del problema modificado. Entre todos aquellos tipos de transformación espectral, la de *desplazamiento e inversión* (shift-and-invert) es sin duda una de las más usadas [263, 264], la cual formulamos del siguiente modo. Si a ambos miembros de la igualdad (5.4) les restamos σBx, obtendremos que:

$$(A - \sigma B)x = (\lambda - \sigma)\, Bx \tag{5.8}$$

de donde concluimos que el problema generalizado inicial se reduce a este otro problema estándar:

$$(A - \sigma B)^{-1} Bx = \theta x \tag{5.9}$$

siendo $\theta = 1/(\lambda - \sigma)$. Dicha transformación se usa para mejorar la convergencia, ya que los valores propios de la formulación original (5.4) más próximos a σ se convierten en los de mayor valor absoluto del problema transformado (5.9), como

ilustra la figura 5.12. Por tanto, una vez obtenidos los valores propios θ del problema transformado, obtendremos los del problema original mediante $\lambda = \sigma + 1/\theta$. De nuevo, la matriz C no debe obtenerse explícitamente. La transformación espectral de desplazamiento e inversión conlleva la solución de un sistema de ecuaciones lineales cada vez que se requiere el producto de la matriz C por un vector, el cual puede resolverse mediante los métodos directos o iterativos descritos en la sección 5.7.3. Es decir, si queremos obtener $y = (A - \sigma B)^{-1} Bx$ entonces:

1. Factorizamos, de forma genérica, la matriz $(A - \sigma B) = LU$.
2. Multiplicamos $u = Bx$.
3. Resolvemos un sistema de ecuaciones triangular inferior $Lv = u$.
4. Resolvemos un sistema de ecuaciones triangular superior $Uy = v$.

Debemos tener en cuenta que U será igual a L^T si A y B son simétricas y definidas positivas. En general, se recomienda usar métodos directos para resolver el sistema con una mayor precisión. Además, la matriz de coeficientes se factoriza una sola vez al comienzo del método de valores propios y, en cada iteración, se aprovecha al resolver los sistemas triangulares.

Otra ventaja de los métodos directos es que, a partir de la factorización de la matriz, proporcionan información de la *inercia* del problema, que puede usarse para determinar si todos los valores propios de un intervalo dado se han calculado ya o no. Dada una matriz Hermitiana $A \in \mathbb{C}^{nxn}$, se llama inercia de A a la terna formada por el número de valores propios negativos, positivos o iguales a cero. La inercia se utiliza por tanto para conocer el número de valores propios, de una matriz hermitiana, en el interior de un intervalo dado $[\alpha, \beta]$. Así, la cantidad de valores propios de una matriz A en un intervalo $[\alpha, \beta]$ se obtiene como la diferencia de la inercia de las matrices $(A - \beta I)$ y $(A - \alpha I)$. Es ésta precisamente la base de una técnica llamada *disección del espectro* (spectrum slicing), la cual emplea la inercia con múltiples desplazamientos σ para obtener todos los valores propios que están comprendidos en dicho intervalo. Para ello, se resuelve una serie de subproblemas de valores propios asociados a un secuencia de desplazamientos seleccionados. El conjunto de vectores propios contenido en el intervalo deseado se obtiene a partir de los vectores propios obtenidos por los subproblemas resueltos.

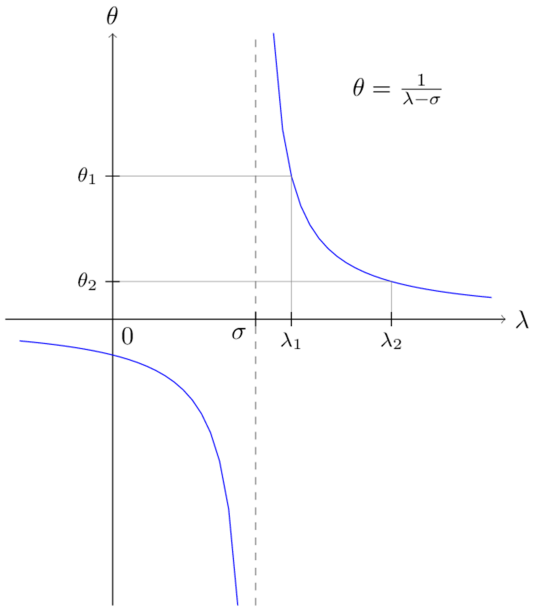


Figura 5.12: Transformación espectral de desplazamiento e inversión.

Por el contrario, si usáramos un método iterativo, la precisión de la solución debería ser ajustada de acuerdo a la tolerancia empleada en la convergencia del método de valores propios. Habitualmente, la tolerancia del método iterativo del sistema de ecuaciones suele ser más exigente que la del método de valores propios.

Obsérvese que el operador matricial $C = (A - \sigma B)^{-1} B$ de la ecuación (5.9) no es simétrico y, por tanto, no pueden emplearse todos aquellos métodos que aprovechen la simetría de la matriz. Una posibilidad para superar este inconveniente es factorizar la matriz B como ocurre en la ecuación (5.6). Otra alternativa diferente supone reemplazar en los algoritmos el producto escalar estándar $\langle x, y \rangle = x^T y$ por este otro $\langle x, y \rangle_B = x^T B y$. Puesto que el operador matricial C es autoadjunto con respecto a este producto escalar, la simetría se recupera.

El esquema común de los métodos iterativos de resolución de problemas de valores propios dispersos supone aplicar el operador matricial a uno o a unos pocos vectores iniciales, calcular las aproximaciones a los valores propios a partir de subespacios de poca dimensión y continuar iterando hasta que se alcance la convergencia, lo cual puede llevarse a cabo de maneras muy distintas. En ese sentido, podemos distinguir entre los siguientes tipos de métodos:

- Métodos con iteraciones sobre uno o varios vectores.
- Métodos basados en los subespacios de Krylov.
- Métodos de optimización y precondicionado.
- Métodos de subestructuración multinivel.

Métodos con iteraciones sobre uno o varios vectores

El método de la *Iteración de la Potencia* es el método más sencillo de todos. Su idea básica consiste en multiplicar, repetidamente, la matriz del problema por un vector de partida normalizado en la iteración anterior. Implícitamente, construye la sucesión de potencias de la matriz, de ahí su nombre. El método únicamente obtiene el valor propio de mayor valor absoluto y su correspondiente vector propio.

Normalmente, su convergencia es muy lenta (lineal), la cual puede mejorarse significativamente en el método de la *Iteración Inversa*, el cual combina el método de la Iteración de la Potencia con la técnica de desplazamiento e inversión previamente descrita. Dicha iteración modificada puede usarse para encontrar valores propios interiores, ya que proporciona el valor propio más cercano al desplazamiento σ dado. Es por tanto particularmente eficiente cuando σ es una buena aproximación al valor propio buscado. El inconveniente que presenta es su mayor coste computacional por iteración, ya que requiere resolver un sistema de ecuaciones lineales. Como decíamos con anterioridad, y en caso de emplear un método directo, la matriz se factorizará una sola vez, reaprovechando dicha factorización en la resolución de los sistemas triangulares que tienen lugar en cada iteración.

El método de la *Iteración del Cociente de Rayleigh* (RQI) es una extensión natural del método de la Iteración Inversa, donde el desplazamiento σ varía en cada paso. En el caso de matrices simétricas, el método ofrece una velocidad de convergencia cúbica, necesitando pocas iteraciones para encontrar un valor propio.

Los tres métodos que acabamos de mencionar obtienen una aproximación a un único valor propio y su correspondiente vector propio. Una vez que se ha alcanzado la convergencia, podrían emplearse técnicas de deflación para continuar obteniendo el siguiente valor y vector propio. Sin embargo, este procedimiento

sería menos eficiente que usar un método que aproxime varios valores y vectores propios al mismo tiempo, como los que se describen a continuación.

El método de la *Iteración del Subespacio* [265] es una sencilla generalización del método de la Iteración de la Potencia, en el cual el operador matricial se multiplica repetidamente y de manera simultánea por un conjunto de vectores, forzando la ortogonalidad entre ellos para evitar que todos converjan al mismo valor propio. Este método está presente en muchas aplicaciones dedicadas al cálculo estructural. Como ventajas, ofrece una implementación sencilla. Sin embargo, es mucho más lento que otros métodos, como el de Lanczos. Puesto que la ortogonalización puede ser un proceso costoso, puede ser efectivo realizarla sólo cada cierto número de iteraciones, el cual no debe ser demasiado grande para evitar la pérdida de ortogonalidad entre dichos vectores.

Métodos basados en subespacios de Krylov

Estos métodos están basados en encontrar aproximaciones a los vectores propios que pertenezcan al subespacio de Krylov de orden m asociado con el operador matricial C y un vector inicial x_0:

$$\mathcal{K}_m(C, x_0) = \text{span}\left\{x_0, Cx_0, C^2x_0, ..., C^{m-1}x_0\right\} \tag{5.10}$$

Tal y como ocurre en el método de la Iteración de la Potencia, estos métodos generan una secuencia de vectores a partir de un vector inicial de forma que, en la iteración k, el vector $C^k x_0$ aproxima la dirección del vector propio dominante. Sin embargo, mientras que el método de la Iteración de la Potencia se queda con el último vector obtenido en cada iteración, descartando los anteriores, los métodos basados en subespacios de Krylov conservan los vectores, manteniendo toda la información contenida en el subespacio generado.

De manera más precisa, dada una matriz V de tamaño n x m, donde $m << n$, cuyas columnas v_i constituyen una base ortonormal del subespacio de Krylov, es decir $V^TV = I_m$ y $\mathcal{K}_m(C, x_0) = \text{span}\{v_1, v_2, ..., v_m\}$, y dada la matriz $H = V^TAV$ de tamaño m x m cuyos valores y vectores propios vienen dados por $Hy_i = \theta_i y_i$, los valores y vectores propios aproximados $\tilde{\lambda}_i, \tilde{x}_i$ del problema original son $\tilde{\lambda}_i = \theta_i$ y $\tilde{x}_i = Vy_i$, los cuales se denominan *valores de Ritz* y *vectores de Ritz*, respectivamente.

Los métodos basados en el subespacio de Krylov calculan V y H simultáneamente, resolviendo posteriormente el problema de valores propios citado y obteniendo los valores y vectores de Ritz como solución aproximada del problema original. En caso de que el problema sea simétrico, la matriz H será tridiagonal, lo cual da lugar al método de Lanczos [266]. Sin embargo, en el caso no simétrico, la matriz H será de tipo Hessenberg, lo que se traduce en el método de Arnoldi [267, 268].

El método de *Lanczos* es simple, ya que en aritmética exacta es posible mantener los vectores base ortogonales simplemente ortogonalizando cada vector con respecto a los dos anteriores mediante un procedimiento como el de Gram-Schmidt clásico o modificado. Sin embargo, en precisión finita, pueden surgir complicaciones, ya que los errores de redondeo destruyen la ortogonalidad tan pronto como converge el primer valor propio, con la necesidad por tanto añadida de aplicar algún tipo de reortogonalización de entre las múltiples disponibles (completa, selectiva, parcial, periódica o sin reortogonalizar).

Pese a ello, el método es muy popular en dinámica estructural, especialmente su variante orientada a bloques [269], que habitualmente se comporta mejor cuando los valores propios están cercanos y agrupados. En dicha variante a bloques se generan varias secuencias de vectores simultáneamente, razón por la cual se puede implementar mediante operaciones de tipo matriz por matriz que aprovechan mejor la jerarquía de memorias. Sin embargo, su implementación resulta bastante compleja y la velocidad de convergencia no suele mejorar con respecto al método con un solo vector a no ser que se cumpla, como decíamos, que los valores propios estén muy cercanos entre sí.

Al igual que Lanczos, el método de *Arnoldi* construye el subespacio de Krylov generando un nuevo vector por cada iteración. Sin embargo, el proceso de Gram-Schmidt para garantizar la ortogonalidad se aplica a todos los vectores de la base calculados en iteraciones anteriores. En consecuencia, el inconveniente principal del método de Arnoldi, que limita seriamente su aplicabilidad, viene dado por el crecimiento de su coste computacional y sus requerimientos de memoria cuando el número de iteraciones, y en consecuencia el número m de vectores, aumenta. Conviene aclarar que esto también le ocurre al método de Lanczos si se usa alguna técnica de reortogonalización.

La necesidad obvia de mantener limitado el tamaño de la base ortogonal da lugar al desarrollo de técnicas de *reinicio*. Una vez que la base calculada ha alcanzado un determinado tamaño, estas técnicas obtienen un nuevo vector a partir de los ya generados, con el cual el proceso comienza tras eliminar a todos los vectores anteriores.

Entre las diferentes técnicas de reinicio, destacan la de *reinicio explícito*, la cual reemplaza el vector de partida por un vector mejorado generado como combinación de los vectores anteriores, a partir del cual se pone de nuevo en marcha el método de Arnoldi o de Lanczos. Una aproximación más eficiente es el *reinicio implícito* [270], el cual combina el método de la iteración QR desplazada implícitamente, aplicado a problemas densos, con el proceso de Lanczos o de Arnoldi. En esta variante, el tamaño de la base ortogonal se limita y actualiza continuamente, hasta que su subespacio contenga un conjunto de vectores propios. En muchos casos, el número de vectores propios que convergen mediante estas técnicas de reinicio coincide con los que convergerían con los métodos de Lanczos y Arnoldi originales, usando un número superior de productos matriz-vector pero reduciendo significativamente el espacio de almacenamiento y el esfuerzo de ortogonalización. Esta técnica se hizo muy popular tras ser implementada e incorporada en la librería ARPACK [16].

El proceso de reinicio se mejoró hace unos años con el método de Krylov-Schur [271], el cual es más simple de implementar que el reinicio implícito y evita la inestabilidad numérica asociada al algoritmo QR usado en la deflación. La versión de Krylov-Schur para matrices simétricas se conoce como método de Lanczos con reinicio grueso (thick restart) [272].

Cuando se quiere resolver el problema de valores propios generalizado simétrico, como es nuestro caso, con la técnica de desplazamiento e inversión, ya vimos que la matriz de la expresión (5.9) dejaba de ser simétrica. Sin embargo, como comentamos, la matriz sí que será simétrica si consideramos el producto escalar asociado a B. En este caso, el método de Lanczos podrá ser aplicado y se le conoce como *B-Lanczos*.

Métodos de optimización y precondicionado

La principal motivación de este tipo de métodos es evitar el elevado coste compu-

tacional asociado a la solución del sistema lineal de ecuaciones en la transformación espectral de desplazamiento e inversión. Estos métodos llevan el nombre de precondicionadores debido a su similitud con la idea del precondicionador aplicado junto a un método iterativo para resolver el sistema de ecuaciones lineales, el cual es una aproximación a la inversa de la matriz que requiere poco esfuerzo computacional.

Basados en el método de *Davidson* [273], los métodos más conocidos de este tipo son el de *Davidson Generalizado* [274] y el de *Jacobi-Davidson* [275]. Estos métodos son similares a los métodos de Lanczos y Arnoldi, en el sentido de que construyen un subespacio en el cual se proyecta el problema de valores propios, y pueden ser especialmente competitivos para obtener valores propios interiores. Sin embargo, la diferencia está en que el subespacio se genera a partir de una matriz precondicionada desplazada, bajo la denominada ecuación de corrección. Además, el sistema de ecuaciones que hay que resolver puede calcularse, sin pérdida de robustez, de manera aproximada mediante métodos iterativos, en lugar de obtener la solución exacta como requiere la transformación espectral. Para reducir el coste de estos métodos, pueden aplicarse también técnicas de reinicio, similares a las utilizadas en los métodos de Krylov, además de plantearse variantes a bloques.

Surgen además otros métodos basados en la idea de combinar un precondicionador con alguna técnica de optimización, como Gradiente Conjugado. Entre ellos, podemos citar al método de *Gradiente Conjugado Precondicionado Localmente Óptimo a Bloques* (Locally-Optimal Block Preconditioned Conjugate Gradient o LOBPCG) [276] y el método de Gradiente Conjugado Acelerado con Deflación (Deflation-Accelerated Conjugate Gradient o DACG) [277, 278].

Métodos de subestructuración multinivel

Los métodos descritos hasta ahora pueden ser aplicados como una caja negra a un operador matricial C dado. Por el contrario, los métodos de subestructuración se conciben como métodos específicos para una aplicación particular. Estos métodos ahondan en el contexto de las implementaciones paralelas y explotan los desarrollos en el área de la descomposición de dominios para resolver ecuaciones en derivadas parciales. La idea básica consiste en dividir el problema en diferentes dominios, los cuales se resuelven separadamente y, finalmente, se combinan los

resultados para dar lugar a una solución global. Este esquema puede aplicarse recursivamente bajo un esquema multinivel.

5.7.5.3. Software numérico disponible

En los últimos años, han sido muchas las ocasiones en las cuales la investigación en el campo de los métodos numéricos dedicados a la resolución de problemas de valores propios dispersos se han materializado en software accesible gratuitamente. En [279] se incluye una recopilación de dicho software, junto con sus características principales, y en [215] podemos encontrar un repositorio actualizado del mismo. De igual manera, es también numeroso el software disponible a nivel comercial. A continuación, describiremos algunas librerías que resuelven el problema generalizado y pueden ser utilizadas desde la librería SLEPc, en la cual están basados los desarrollos de esta tesis doctoral.

ARPACK

ARPACK (ARnoldi PACKage) [16] es una colección de funciones escritas en Fortran 77 diseñadas para resolver problemas de gran dimensión de valores propios estándar y generalizados simétricos y no simétricos. Está basado en el método de Arnoldi con reinicio implícito, en el caso general. Cuando el problema es simétrico, se reduce al método de Lanczos con reinicio implícito con reortogonalización completa. Estas técnicas pueden ser vistas como una síntesis de los procesos de Arnoldi y de Lanczos con técnicas de desplazamiento QR implícitamente.

ARPACK depende de LAPACK y de BLAS y puede usarse como paquete externo desde SLEPc. Su implementación paralela sobre máquinas de memoria compartida se denomina PARPACK [280] y está basada en MPI y en BLACS.

BLZPACK

BLZPACK (Block LancZos PACKage) [281] recoge una implementación paralela, sobre MPI, del algoritmo de Lanczos a bloques diseñado para resolver un problema estándar o generalizado con matrices reales y simétricas. El desarrollo de este código, en Fortran 77, estuvo motivado inicialmente por el estudio del problemas vibratorios en ingeniería estructural, aunque posteriormente se amplió para que

pudiera aplicarse a un abanico mayor de aplicaciones. Combina variantes de reortogonalización parcial y selectiva e incorpora la técnica de disección del espectro (spectrum slicing) para obtener todos los valores propios de un intervalo dado. Puede usarse como paquete externo desde SLEPc.

FEAST

La librería FEAST [282] obtiene los valores y vectores propios, pertenecientes a un intervalo dado, en un problema simétrico y no simétrico estándar o generalizado. Implementa un método basado en la Iteración del Subespacio usando proyectores espectrales capaz de ser ejecutado en paralelo en máquinas con memoria compartida, sobre OpenMP, o distribuida, sobre MPI. La versión 2.1, paralelizada para máquinas de máquina compartida, forma parte de la librería INTEL MKL. Invoca a BLAS y a LAPACK y puede ser usada desde SLEPc. Requiere de software externo para resolver sistemas de ecuaciones dispersos, tales como Intel MKL PARDISO o MUMPS.

BLOPEX

BLOPEX (Block Locally Optimal Preconditioned Eigenvalue Xolvers) [283] es una librería, escrita en C y Matlab, que recoge el método de Gradiente Conjugado Precondicionado Localmente Óptimo a Bloques (LOBPCG) para resolver problemas simétricos y generalizados de valores propios. Se puede usar a nivel secuencial o en paralelo, mediante MPI. BLOPEX está incorporado en la librería hypre y puede usarse como paquete externo desde SLEPc y PETSc, lo que le proporciona la posibilidad de ejecutarse en paralelo sobre MPI. Depende de BLAS y de LAPACK.

SLEPc

SLEPc (Scalable Library for Eigenvalue Problem Computations) es un paquete software diseñado para resolver en paralelo, sobre MPI, problemas de valores propios dispersos y de gran dimensión expresados tanto en forma estándar como generalizada. Incluye problemas simétricos y no simétricos, con aritmética real y compleja. Puede ser usado desde C y desde Fortran. Resuelve además problemas lineales, no lineales y polinomiales, además de abordar la resolución de problemas de descomposición de valores singulares.

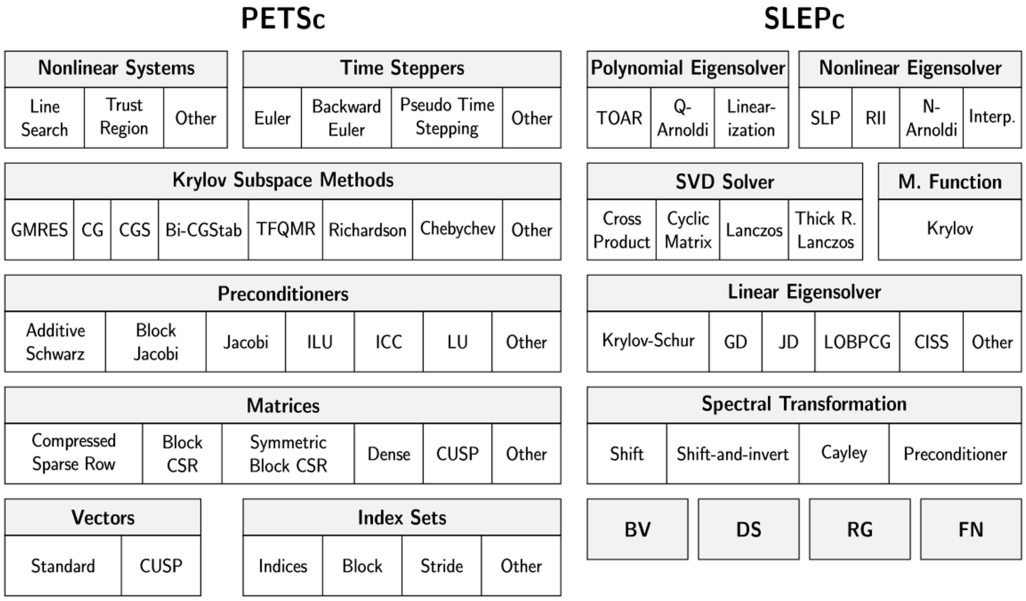


Figura 5.13: Componentes numéricos de PETSc y de SLEPc.

SLEPc depende de BLAS y LAPACK y está construido sobre PETSc, del cual puede ser considerado como una extensión que le proporciona la funcionalidad necesaria para resolver problemas de valores propios. La figura 5.13 muestra los diferentes componentes que conforman PETSc y SLEPc.

SLEPc ofrece un amplio abanico creciente de métodos de resolución, tales como Krylov-Schur, Arnoldi con reinicio explícito, Lanczos con reinicio explícito y con diferentes técnicas de reortogonalización, LOBPCG, Davidson Generalizado, Jacobi-Davidson, CISS (Contour Integral Spectrum Slicing) y otros como Iteración del Subespacio, Iteración del Cociente de Rayleigh, Potencia Inversa o Iteración de la Potencia. Incluye también diferentes tipos de transformaciones espectrales.

Adicionalmente, SLEPc proporciona un acceso transparente a otros software externos que resuelven problemas estándar, como PRIMME [284] y TRLAN [285], o generalizados, como ARPACK, BLZPACK, FEAST y BLOPEX.

Capítulo 6

Las Tecnologías Grid y Cloud

En este capítulo abordaremos las características que presentan la computación Grid y la computación Cloud. Si bien es cierto que en ambos casos el usuario puede utilizar un software o un conjunto de recursos computacionales de los que no disponía a priori, también es verdad que la resolución de los problemas y el uso de los recursos se lleva a cabo empleando tecnología diferente y bajo un modelo colaborativo o comercial, respectivamente. El capítulo describe en qué consiste la Arquitectura Orientada a Servicio, la computación basada en Grid o la computación en la nube. Además, se incluye una descripción del paquete software Globus Toolkit, de la infraestructura de Microsoft Azure y del proyecto VENUS-C, los cuales nos han permitido desarrollar diferentes servicios Grid y Cloud orientados al cálculo estructural, como era objetivo de esta tesis doctoral.

6.1. Las Arquitecturas Orientadas a Servicio

Una Arquitectura Orientada a Servicio (SOA) es una forma lógica de diseñar un sistema software que proporcione servicios o bien a usuarios finales de una aplicación o bien a otros servicios débilmente acoplados y distribuidos en la red, por medio de interfaces bien definidos a los que se accede de un modo seguro y

uniforme. En una SOA, los recursos software se empaquetan como servicios, los cuales consisten en aplicaciones autocontenidas y bien definidas que proporcionan una funcionalidad determinada y que se describen mediante un lenguaje estándar. La madurez adquirida por los servicios Web habilita la creación de servicios que pueden ser accedidos a demanda y de una forma uniforme.

Este paradigma cambia sustancialmente el modo en el cual se diseñan, se exponen y se utilizan las aplicaciones. No en vano, una ventaja importante de las aplicaciones SOA que se ejecutan en una plataforma computacional distribuida es que les permite llegar a más usuarios, con un enfoque comercial o colaborativo. Es por ello que una SOA bien construida puede potenciar un entorno de negocio, gracias a una infraestructura flexible compuesta por un software y una plataforma de procesamiento. Sin embargo, este paradigma da lugar a nuevas dificultades y requerimientos, como es por ejemplo el intercambio de los datos de entrada y salida entre las aplicaciones, que puede llegar a convertirse en una etapa muy costosa en tiempo con respecto a las aplicaciones tradicionales.

De igual modo, la Computación Orientada a Servicio (SOC) es un paradigma de computación distribuida donde el servicio proporcionado viene dado por los elementos computacionales, que pueden ser descritos, publicados, descubiertos, orquestados y programados mediante protocolos estándar con el objetivo de construir redes de aplicaciones distribuidas colaborativas o comerciales dentro y fuera de los límites de una organización.

Los servicios Web y los servicios Grid fueron durante años la forma habitual de implementar aplicaciones orientadas a servicio, dando paso hoy en día a los servicios Cloud. Los servicios Web proporcionan la base del desarrollo y la ejecución de procesos de negocio que están distribuidos y accesibles en la red mediante interfaces y protocolos estándar. Los servicios Grid se orientan a la integración de recursos disponibles en diferentes dominios administrativos, ofreciendo un interfaz estandarizado, coherente y con estado. Por el contrario, los servicios Cloud están dirigidos generalmente al aprovechamiento bajo demanda de los recursos disponibles en infraestructuras computacionales públicas o privadas, a menudo virtualizadas y bajo un modelo de pago por uso.

6.2. La Computación basada en Grid

Durante la década de los años 80, un importante número de investigadores de diferentes disciplinas comenzaron a trabajar juntos para resolver problemas tradicionalmente denominados como grandes desafíos. Se trataba de problemas de ciencia e ingeniería para los cuales las infraestructuras computacionales a gran escala de la época proporcionaron una herramienta fundamental para lograr nuevos descubrimientos científicos. Para ello, fue fundamental un modelo de colaboración multidisciplinar que tuvo un tremendo impacto en la forma en la cual entendemos la ciencia a día de hoy, en disciplinas como la física de altas energías, donde trabajan juntos investigadores de áreas muy dispares de conocimiento.

Dicha idea de colaboración conjunta entre investigadores pertenecientes a zonas dispares del planeta y los avances en internet, en las redes de comunicaciones y en las infraestructuras de cómputo impulsaron la idea de enlazar distintos recursos computacionales geográficamente distribuidos y gestionados en dominios administrativos diferentes, con el objetivo de dar lugar a una plataforma de computación distribuida, dedicada a la ciencia y la ingeniería avanzada, conocida como el Grid [286, 287].

La idea del Grid es por tanto la de compartir, de forma transparente, eficiente y segura mediante organizaciones virtuales dinámicas y multi-institucionales, datos, software y recursos de cómputo, de almacenamiento o de otro tipo, para resolver problemas cuyos requisitos superan las capacidades de los centros individuales [288]. Una Organización Virtual (VO) es una coalición temporal o permanente de individuos, grupos u organizaciones que comparten, bajo unas reglas bien establecidas, información, conocimiento, capacidades y recursos hardware y software, con el objetivo de resolver un conjunto de problemas en un ámbito común, independientemente de su ubicación geográfica.

En consecuencia, si el objetivo del Grid es crear un potente supercomputador virtual autogestionado a partir de un conjunto de sistemas heterogéneos interconectados y compartirlos [289], podemos decir el Grid tiene la capacidad de ampliar la potencia computacional que tenemos a nuestra disposición, la cual deja de estar limitada y restringida a los recursos computacionales de los que dispongamos.

Como es evidente, el establecimiento, gestión y explotación de los recursos por parte de dichas VOs requiere el desarrollo de una nueva tecnología, formada por componentes software que deben interaccionar entre sí, cuyo principio de diseño ha sido el uso de interfaces genéricos estándar.

Todas estas herramientas componen la denominada tecnología Grid, sobre la cual se desarrollarán y desplegarán las posteriores aplicaciones de usuario. Un buen ejemplo de dicho software es Globus Toolkit, el cual se convirtió en el estándar de facto para el despliegue de Grids computacionales y que describiremos en el apartado siguiente.

A lo largo de estos años, han sido diversos los proyectos de gran envergadura relacionados con el Grid. Un ejemplo de ello es NEES [290] una organización orientada a servicio que gestiona la infraestructura computacional de la denominada *Network for Earthquake Engineering and Simulation* en EEUU. La misión de NEES es la de proporcionar los servicios necesarios que permitan a los investigadores de fenómenos sísmicos, de forma colaborativa, tener a su alcance un conjunto de datos, software y recursos computacionales de vanguardia que les sirva para planificar, realizar y publicar sus experimentos. Todo ello está claramente dirigido a lograr una mejor protección contra los desastres causados por los terremotos. Dentro del software que tienen a su alcance los investigadores, denominado NEESgrid Software Suite, se encuentran diferentes herramientas de simulación como OpenSees (Open System for Earthquake Engineering Simulation) [291], una aplicación de código abierto desarrollada por el PEER (Pacific Earthquake Engineering Research Center) encargada de simular la respuesta sísmica de sistemas estructurales y geotécnicos.

A nivel europeo, el proyecto EGEE (The Enabling Grids for E-sciencE) [292] construyó una infraestructura de computación distribuida que permitió a varios miles de usuarios, agrupados en más de 200 Organizaciones Virtuales diferentes, acceder a los recursos computacionales de más de 300 centros. Dicha infraestructura, compuesta por más 200000 núcleos de procesamiento y que ofrecía varios Petabytes de almacenamiento, ejecutaba trabajos gestionados por un software desarrollado en el marco del proyecto llamado gLite (Lightweight Middleware for Grid Computing). La comunidad de usuarios dedicados a la física de altas energías fue la más numerosa en usar la infraestructura Grid. De hecho, los 4 experimentos

que se llevaron a cabo en el acelerador de partículas LHC (Large Hadron Collider) del CERN suponían la ejecución de más de 150000 trabajos diarios.

A día de hoy, dicha infraestructura de computación la gestiona EGI (European Grid Infrastructure) [293], la cual coordina además las iniciativas de Grid de cada país que dan lugar a la plataforma Grid europea. La infraestructura, compuesta conjuntamente por plataformas Grid y Cloud de más de 350 centros, la componen unos 657000 núcleos computacionales, en la cual se lanzan más de 1400 millones de trabajos al día.

En nuestro país, la Iniciativa Grid Nacional española (ES-NGI) [294] integra recursos computacionales interconectados de más de 20 centros, integrados en EGI, que componen una infraestructura de computación virtual y distribuida, utilizando la tecnología Grid. Mediante la compartición de estos recursos, y su gestión de forma conjunta y eficiente, se pretende ofrecer un servicio de soporte computacional a los investigadores.

Existen diversos paquetes software desarrollados para gestionar y desplegar sistemas Grid. Ejemplos de ello son Globus Toolkit, que lo describiremos a continuación, Unicore (Uniform Interface to Computing Resources) [295] o el ya citado gLite, que en la actualidad se distribuye a través de la Distribución de Middleware Unificado (UMD por sus siglas en inglés), disponible en [296].

6.3. Globus Toolkit

6.3.1. Introducción

Globus Toolkit (GT) [297] es un software para desarrollar aplicaciones y desplegar infraestructuras de computación distribuida orientadas a servicio implementado por la Alianza Globus, una asociación internacional dedicada a desarrollar tecnologías necesarias para construir infraestructuras de computación Grid. Su primera versión surge en el año 1998. La última, la versión 6.0, apareció a finales del año 2014. Sin lugar a dudas, la versión más utilizada y la que contó con mayor éxito fue la 4.0 (GT4) [17], publicada en el año 2005. Es precisamente dicha versión, la cual cuenta con un elevado grado de compatibilidad con la ver-

sión 6.0, la que se principalmente se describe a continuación, puesto que es en la que está basado el Servicio Grid de Análisis Estructural desarrollado en el marco de esta tesis doctoral.

Esta herramienta software incluye servicios y librerías para monitorizar, descubrir y gestionar recursos, abarcando también aspectos relativos a la seguridad, la detección de fallos y la gestión de ficheros. Sus componentes pueden usarse en su totalidad o con independencia para desarrollar aplicaciones. Además de convertirse en una parte central de muchos proyectos de investigación de ciencia e ingeniería, representa también una capa sobre la cual algunas compañías informáticas han construido sus productos Grid comerciales.

Cada organización posee su modo particular de trabajar y la colaboración entre múltiples organizaciones se ve a menudo frustrada por la incompatibilidad de recursos, tales como los formatos de los ficheros, los ordenadores usados o las redes de comunicación. GT fue concebido para reducir el impacto de estos obstáculos. Así, sus principales servicios, interfaces y protocolos permiten que los usuarios accedan a recursos remotos como si de sus recursos locales se tratara, a la vez de preservar el control local sobre quién y cuándo accede a ellos mismos. A continuación se incluyen algunos de los principios en los que se basa GT4:

- *Arquitectura orientada a servicio*: GT4 puede usarse para construir aplicaciones e infraestructuras orientadas a servicio. Dichas aplicaciones se construirán por medio de componentes que deben definir su interfaz de servicio.
- *Servicios de infraestructura*: Se incluyen servicios que permiten acceder, monitorizar, gestionar y controlar los elementos computacionales y de almacenamiento de la infraestructura.
- *Servicios Web*: GT4 hace un uso extensivo de los protocolos estándar de los servicios Web [298], así como de los mecanismos de descripción y descubrimiento de los servicios, donde se lleve a cabo la autenticación y la autorización de los usuarios que pretendan acceder a los mismos.
- *Contenedores*: GT4 incluye componentes que pueden ser usados para construir contenedores, los cuales albergan servicios Web escritos en diferentes lenguajes como C, Java o Python. Un contenedor no es más la agrupación

de todo el software necesario para desplegar un servicio Web, quedando así a disposición de los clientes.

- *Seguridad*: Los sistemas de seguridad, basados en estándares, están orientados a la protección de la información, así como a la autenticación y autorización de los usuarios.

- *Estándares*: Siempre que sea posible, los desarrollos de Globus están basados en estándares o especificaciones ampliamente adoptadas por la comunidad, a fin de implementar componentes portables y reutilizables.

Esta evolución en GT4 a que la arquitectura de un sistema Grid estuviera basada en los conceptos y tecnologías de los servicios Web vino precisamente por acogerse a OGSA (Open Grid Services Architecture) [299]. Esta especificación, desarrollada por el Open Grid Forum, pretende estandarizar todos y cada uno de los servicios que forman parte del Grid, confiando para ello en los servicios Web, los cuales destacan por proporcionar mecanismos estándar para describir, descubrir y utilizar una funcionalidad concreta en la red, facilitando el desarrollo de las arquitecturas orientadas a servicio.

Son diversas las definiciones que podemos encontrar de los servicios Web. Por un lado, un *servicio Web* es una tecnología, basada en un conjunto de protocolos y estándares, que permitir intercambiar datos entre dos aplicaciones que podrán estar implementadas en lenguajes diferentes y ejecutadas sobre cualquier plataforma. Por otro, y de acuerdo a la definición del W3C Web Services Architecture Working Group [300], un servicio Web es un sistema software que permite la interacción con otro sistema diferente a través de la red. Dicho servicio presenta un interfaz, descrito en un formato entendible por una máquina, que otros sistemas emplean para interaccionar con él, de acuerdo a la descripción empleada y mediante mensajes basados en un formato concreto, empleando un protocolo de transporte.

Más en detalle, los protocolos en los que están basados los servicios Web son los siguientes, de acuerdo a la arquitectura recogida en la figura 6.1:

- SOAP (Simple Object Access Protocol) y REST (Representational State Transfer): Son dos de los protocolos de comunicación entre las aplicaciones,

pertenecientes a la capa de Invocación. Suelen estar basados en el intercambio de datos de tipo XML (Extensible Markup Language) y usan habitualmente HTTP (Hypertext Transfer Protocol) como protocolo de transporte subyacente.

- WSDL (Web Services Description Language): Es el lenguaje que permite describir el interfaz a través del cual se puede acceder al servicio Web, indicando los requisitos necesarios para comunicarse con él. Dicha descripción estará basada en XML. De este modo, un programa cliente puede acceder a la descripción de un servicio, conociendo qué funciones ofrece a través de la red. Dentro de la arquitectura del servicio, forma parte de la capa de Descripción.

- UDDI (Universal Description, Discovery and Integration): Se trata del protocolo que permite almacenar y publicar la información de un servicio Web, con el objetivo de que el resto de aplicaciones puedan localizarlo y utilizarlo. En realidad, UDDI es un repositorio público para almacenar de forma estructurada información sobre empresas y los servicios que éstas ofrecen, junto con los interfaces de uso de los mismos. Dentro de la arquitectura de un servicio Web, el protocolo UDDI pertenece a la capa de Procesos.

El acceso a los servicios Web se lleva a cabo mediante los denominados URI (Uniform Resource Identifier). Se trata de direcciones que se almacenan en el repositorio UDDI, similares a las URL (Uniform Resource Locator) de las páginas web, aunque más completas.

Con todo ello, cuando una aplicación cliente quiere acceder a un servicio Web, necesita conocer en primer lugar su URI, bajo una etapa denominada *descubrimiento*. A continuación, el cliente interacciona con el servicio encontrado, quien le proporciona su descripción y funcionalidad expresada mediante WSDL. De acuerdo a la funcionalidad necesitada y según el modo de invocar al servicio recogido en su descripción, el cliente lo invoca proporcionándole los datos de entrada y recogiendo los de salida, vía REST o SOAP, los cuales se transmiten por HTTP.

Pero OGSA va más allá y una premisa básica de esta especificación es que todo en un Grid sean servicios, entendiendo como tal a una entidad accesible

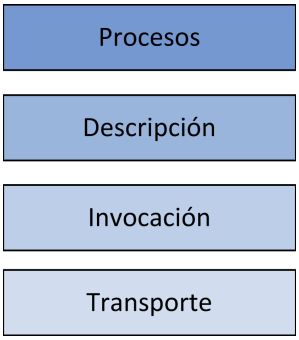


Figura 6.1: Arquitectura de los servicios Web.

vía red que proporciona alguna capacidad mediante el intercambio de mensajes. Esto significa que todos aquellos recursos computacionales o de almacenamiento, el software o las bases de datos son, para OGSA, servicios.

Es más, OGSA define la arquitectura de un sistema Grid en base a los denominados *servicios Grid*, los cuales representan servicios Web con estado. Precisamente, una de las características de los servicios Web tradicionales es que carecen de estado, lo cual significa que están diseñados para implementar acciones que no requieran persistencia entre diferentes invocaciones. Surge entonces la especificación WSRF (Web Services Resource Framework) [301], definida por OASIS (Advancing Open Standards for the Information Society) y adoptada por GT4, la cual se encarga de definir un marco abierto y genérico para modelar y acceder a recursos con estado usando servicios Web, a través de los interfaces que ellos mismos proporcionen. Para ello, emplea la especificación WS-Resource (Web Service Resource) [302], la cual representa la unión de un recurso y un servicio Web a través del cual se accede al recurso, describiendo su estado mediante una descripción WSDL.

La figura 6.2 muestra un diagrama de capas recogiendo la relación entre OGSA, WSRF y los servicios Web. GT4 implementa muchos de los servicios requeridos y estandarizados por OGSA (gestión de VOs, seguridad, gestión de trabajos y de recursos, etc.), además de implementar WSRF. Las aplicaciones Grid de usuario estarán basadas en los servicios de más alto nivel definidos por OGSA, quien además plantea incorporar otras capacidades y protocolos a los servicios Web. Es el caso por ejemplo del protocolo denominado WS-Notification (Web Service Notification) [303], el cual está formado por las especificaciones que permiten a

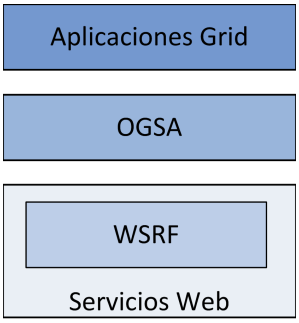


Figura 6.2: Diagrama de capas de OGSA, WSRF y los servicios Web.

los servicios Web difundir o notificar entre sí cambios de estado. A través de dicho protocolo, OGSA define interfaces estándar de servicios para la subscripción y la entrega de notificaciones.

6.3.2. Componentes principales

GT4 está compuesto por los tres siguientes tipos de componentes, los cuales se describirán a continuación con más detalle, agrupados por funcionalidades:

- Un conjunto de servicios básicos encargados de la gestión de la ejecución (GRAM), el acceso y movimiento de los datos (GridFTP, RFT, etc.), la gestión de réplicas (RLS), la monitorización y el descubrimiento de recursos (Index Service, Trigger Service, MDS), la gestión de credenciales y otros. Aunque algunos están implementados en otros lenguajes y puedan usar otros protocolos, la mayoría son servicios Web implementados en Java.
- Tres contenedores encargados de albergar servicios desarrollados por el usuario en Java, C o Phyton. Estos contenedores proporcionan mecanismos de seguridad, descubrimiento, gestión del estado, etc. requeridos a la hora de construir servicios, enriquecidos con un conjunto de especificaciones relativas a los servicios Web, tales como WSRF, WS-Notification y WS-Security.
- Un API que permite a los programas cliente escritos en C, Java o Python invocar directamente a los servicios básicos de GT4 e incorporarlos a los desarrollados de un usuario.

6.3.2.1. Gestión de recursos y ejecución de trabajos

El componente que proporciona la ejecución remota, homogénea y estandarizada de los trabajos y gestiona el estado de su ejecución es el llamado GRAM (Globus Resource Allocation Manager) [304]. Su interfaz permite que los clientes puedan indicar el tipo y la cantidad de recursos computacionales necesitados, el fichero ejecutable y sus argumentos, los datos de entrada y salida, las credenciales del usuario a usar y toda aquella información adicional y necesaria para la ejecución de un trabajo. Además, los clientes tienen la posibilidad de monitorizar el estado tanto de sus tareas como de los recursos computacionales en los que se ejecutan, mediante la subscripción a notificaciones en las que se informa del estado de las mismas.

Una de las ventajas del uso de GRAM es que abstrae a los usuarios de los diferentes gestores de trabajos, como PBS, TORQUE, Sun Grid Engine, etc. que a nivel local pueda tener cada recurso, delegando en ellos la gestión de los nodos locales.

6.3.2.2. Acceso y comunicación de los datos

Como es lógico, las aplicaciones necesitarán acceder a los datos y moverlos entre diferentes recursos computacionales dentro del Grid o desde/hacia un cliente externo, los cuales podrán ser de un tamaño considerable en multitud de ocasiones. GT4 implementa distintos mecanismos para ello, tales como el componente GridFTP [305], una implementación Grid del protocolo FTP que proporciona una herramienta fiable y segura de movimiento optimizado de grandes volúmenes de datos, o el servicio RFT (Reliable File Transfer) [306] que gestiona de manera fiable múltiples transferencias mediante GridFTP. Se dispone también de un sistema de gestión de ficheros replicados, denominado RLS (Replica Location Service) [307].

6.3.2.3. Servicios de información

La monitorización y el descubrimiento de recursos son dos tareas primordiales a considerar en un sistema distribuido. La monitorización nos permite detectar y diagnosticar problemas que puedan estar ocurriendo en un recurso Grid, mientras que el descubrimiento nos permite encontrar e identificar aquel recurso que posee unas características deseadas.

Ambas tareas requieren de la capacidad de recoger información de fuentes diversas y distribuidas. GT4 proporciona MDS (Monitoring and Discovery Service) [308], el cual facilita el acceso y la publicación de la información estática y dinámica de los recursos, como puede el sistema operativo en el que está basado, la cantidad de memoria de la que dispone o la carga del procesador.

Las propiedades de un recurso se expresan en formato XML y es posible acceder a esta información mediante una petición directa o mediante una subscripción. Estos mecanismos son básicamente implementaciones de las especificaciones WSRF y WS-Notification construidas sobre cada servicio y contenedor de GT4 que pueden ser incorporadas a un servicio desarrollado por un usuario.

6.3.2.4. Gestión de la seguridad

La seguridad es un aspecto particularmente importante a la hora de ejecutar trabajos de forma remota. GT4 proporciona, gracias a GSI (Grid Security Infrastructure) [309], los componentes necesarios para garantizar dicha seguridad en las infraestructuras Grid. Incluye la especificación WS-Security [310] de OASIS, la cual ofrece mecanismos de intercambio de mensajes SOAP seguros entre servicios Web. Algunos conceptos clave que se manejan son los siguientes:

- *Autenticación*: Es el proceso que verifica la identidad de los usuarios, los recursos, los servicios, las aplicaciones, etc. que interaccionan en el Grid.
- *Autorización*: Consiste en el control de acceso a un servicio, garantizando que sólo pueden acceder al mismo aquellos usuarios que tengan permiso.
- *Integridad de los datos*: Asegura que los datos no se modifican o se destruyen

sin la requerida autorización.

- *Confidencialidad de los datos*: También denominada privacidad, consiste en garantizar que los datos que se envían, almacenan o reciben del Grid no puedan ser entendidos o interpretados por terceras personas.

En el nivel más bajo, los componentes de seguridad de GT4, basados en estándares, proporcionan la apropiada protección en las comunicaciones, gracias a la denominada *criptografía de clave pública o asimétrica*, la cual emplea una clave pública y otra privada en el proceso de cifrado y descifrado de la información intercambiada entre emisor y receptor.

Además de los protocolos necesarios, GT4 se basa en el uso de nombre y contraseña o en los certificados digitales X.509 para proporcionar autorización, autenticación y delegación de credenciales. Un *certificado digital* es un documento digital que certifica que una clave pública pertenece a un usuario particular. Este documento vendrá firmado por una entidad denominada *Entidad o Autoridad Certificadora* quien da fe de lo recogido en el certificado. Por tanto, cualquier usuario que quiera utilizar una infraestructura Grid debe disponer de antemano de su certificado digital para autenticarse, en el cual se recoge su nombre, organización y país al que pertenece, su clave pública, la entidad de la Autoridad Certificadora que ha emitido el certificado y una firma digital para garantizar que la información del certificado no se haya modificado.

Adicionalmente, los recursos computacionales también dispondrán de su certificado de tipo X.509, garantizando así su identidad. Los protocolos implementados permiten por tanto que todas las entidades (usuarios y recursos) validen mutuamente sus credenciales, las usen para establecer un canal seguro de comunicación y creen y transporten las credenciales delegadas mediante los certificados *proxy*, que permitan que un componente remoto actúe en nombre de un usuario durante un periodo de tiempo limitado.

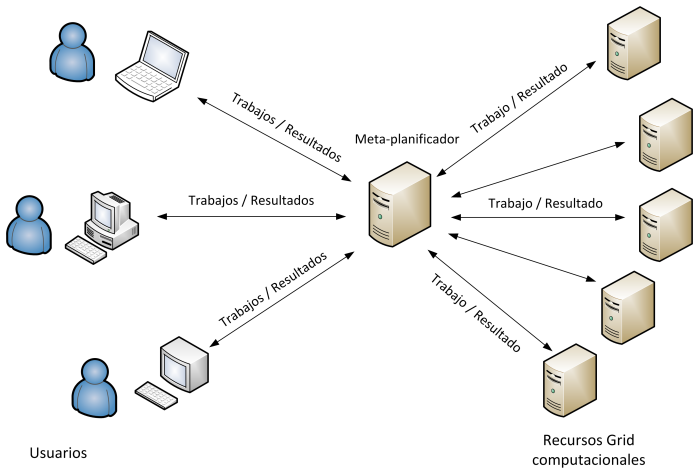


Figura 6.3: Planificación de tareas a recursos computacionales.

6.4. Meta-planificación de tareas en el Grid

Como hemos podido comprobar, GT4 incluye multitud de servicios y funcionalidades básicas y necesarias para desplegar una infraestructura Grid. Sin embargo, para poder ejecutar trabajos en un Grid computacional, es necesario disponer de una herramienta encargada de llevar a cabo la meta-planificación de las tareas a los recursos computacionales, de la cual carece GT4.

Dicha meta-planificación consiste en gestionar eficientemente la ejecución remota de las tareas en los recursos computacionales de una o más organizaciones que forman parte de una infraestructura Grid (ver figura 6.3). Entre las tareas a realizar por parte de un meta-planificador se encuentran el descubrimiento de los recursos, el filtrado y la selección de los recursos, la asignación eficiente de las tareas a los recursos, la gestión de los datos, la monitorización del progreso de la ejecución de las tareas y la tolerancia a fallos multinivel, ante caídas del recurso o del propio meta-planificador.

Son diversos los meta-planificadores existentes para poder ser utilizados sobre una infraestructura basada en GT. Ejemplos de ello son HTCondor, GridBus, GridWay, GMarte, etc., como podemos apreciar en la figura 6.4. A continuación describimos brevemente cada uno de ellos, poniendo especial énfasis en GMarte, al ser el meta-planificador empleado en el Servicio Grid de Análisis Estructural desarrollado.

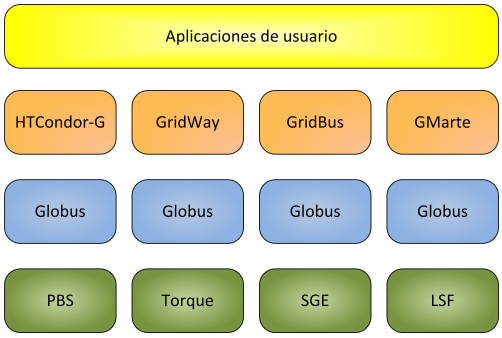


Figura 6.4: Diagrama de componentes de un sistema software Grid basado en GT.

HTCondor [311] es un sistema de gestión de carga especializado en trabajos de computación intensiva capaz de crear un entorno de computación de alta productividad en máquinas Linux, Mac OS y Windows. HTCondor se puede usar para gestionar un cluster de nodos de computación dedicados, como un cluster de PCs, o para aprovechar la potencia de cómputo de máquinas ociosas de escritorio. La versión denominada HTCondor-G [312] incorpora un interfaz GRAM que le permite conectarse a infraestructuras Grid compuestas por máquinas gestionadas por las versiones 2 y 5 de GT.

GridBus Broker [313] ofrece un API de alto nivel, desarrollado en Java, que puede incorporarse en una aplicación con el objetivo de llevar a cabo la gestión de una infraestructura Grid computacional o de datos heterogénea. Ofrece un acceso transparente para el usuario a diferentes software Grid, entre los que se encuentra GT4 o HTCondor. Los mecanismos de planificación tienen en consideración el número de trabajos finalizados para evaluar las prestaciones de cada recurso computacional, lo que determina, junto con el límite máximo de trabajos a enviar a cada uno de ellos, el procedimiento de selección de un recurso a la hora de asignarle una tarea. De este modo, por cada tarea pendiente de ejecución, el planificador selecciona el recurso que supuestamente completará el trabajo lo antes posible.

GridWay [314] es un planificador de tareas en infraestructuras computacionales de una misma organización gestionadas por diferentes sistemas DRM (Distributed Resource Management) como Sun Grid Engine, HT-Condor, PBS, etc. que incorpora la especificación DRMAA (Distributed Resource Management Application API) [315]. A su vez, actúa también como meta-planificador en infraestruc-

turas Grid de una o más organizaciones, interaccionando con otros software Grid intermedios como GT o gLite. GridWay permite ejecutar tareas independientes, así como otro tipo de tareas con dependencias diversas entre ellas.

GMarte [18] es un meta-planificador con un API de alto nivel desarrollado en Java para interaccionar con los recursos computacionales remotos de una infraestructura Grid basada en GT o en gLite y llevar a cabo la ejecución remota de las tareas. Alternativamente a dicho API, el usuario dispone también de la posibilidad de usar un fichero XML en el que se especifica la descripción de las tareas computacionales, los recursos Grid a emplear y la configuración del meta-planificador. Un aspecto diferenciador con respecto a otros meta-planificadores es que permite recoger ficheros generados por una simulación antes de que ésta haya acabado su ejecución por completo. En nuestro caso particular, esta característica es de especial interés a la hora de llevar a cabo una simulación dinámica remota a lo largo del tiempo, de manera que podamos recibir los resultados de la misma para aquellos instantes que ya han sido calculados, solapando de este modo la computación con la recogida de resultados y reduciendo notablemente el tiempo dedicado a la recuperación de una cantidad considerable de datos.

6.5. La computación en la nube

El mundo de las tecnologías de la información está experimentando un cambio notable en la forma de proceder, que ya comenzó con la computación en el Grid, y que supone emplear software y recursos computacionales distribuidos y disponibles a través de internet, vistos como un servicio de uso a demanda, en lugar de adquirir y gestionar aplicaciones software y recursos de cómputo en propiedad.

Los avances en el hardware de los computadores, a nivel de los procesadores multinúcleo y de las técnicas de virtualización, las tecnologías de internet, como los servicios Web y las arquitecturas orientadas a servicio, la computación distribuida en clusters o en una infraestructura Grid y las mejoras en la gestión de los sistemas de la información han dado lugar a un punto común de convergencia de todas estas tecnologías en lo que a día de hoy denominamos como Cloud Computing o computación en la nube.

Son múltiples las definiciones que existen del Cloud [316, 317]. Una de ellas, dice que el Cloud, o la nube, es un sistema de computación paralela y distribuida, compuesto por un conjunto de computadores virtualizados e interconectados que son dinámicamente aprovisionados y presentados como uno o más recursos de computación unificados, basados en un acuerdo de servicio establecido a través de una negociación entre el proveedor del servicio y el consumidor. Muy aceptada y citada es la definición del National Institute of Standards and Technology (NIST) de EEUU [318], la cual incluimos a continuación: *"Cloud computing es un modelo tecnológico que permite el acceso por red ubicuo, adaptado y bajo demanda a un conjunto compartido de recursos de computación configurables (redes, servidores, equipos de almacenamiento y servicios) que puede ser rápidamente aprovisionado y liberado con un mínimo esfuerzo de gestión o interacción por parte del proveedor del servicio".*

Algunas de las características de la computación en la nube que la diferencian de otro tipo de tecnologías son las siguientes [319]:

- *Acceso ubicuo*: El usuario puede acceder a los servicios que le ofrece la nube desde cualquier ubicación geográfica, siempre que disponga, claro está, de una conexión a internet, y desde multitud de dispositivos físicos diferentes, como ordenadores, tabletas, dispositivos móviles, etc.

- *Elasticidad*: Permite que los usuarios aumenten o reduzcan su cantidad de recursos computacionales de manera rápida y a medida que se necesiten. Se distingue entre la *elasticidad vertical*, cuando el recurso mejora en su número de núcleos computacionales o en la memoria de la que dispone, escalando por tanto a un recurso más potente, y entre la *elasticidad horizontal*, cuando incorporamos más recursos con las mismas características de los ya existentes. Este aumento o disminución en el número de recursos puede llevarse a cabo de manera manual o automatizada. Los usuarios pueden llegar a pensar que el conjunto de recursos disponible bajo demanda es infinito.

- *Pago por uso*: En la nube, los usuarios pagan sólo por aquellos recursos hardware o software que utilizan, liberándolos tan pronto como no los necesiten para reducir costes. Los servicios deben tener un precio conocido a priori y los proveedores deben disponer de herramientas de monitorización

que midan la cantidad de recursos consumidos por cada usuario y lleven a cabo la contabilidad y la facturación.

- *Particularización*: En un entorno multi-distribuido, puede existir mucha disparidad entre las necesidades de cada usuario. Por tanto, los recursos alquilados en la nube deben ser altamente amoldables a los requerimientos del cliente. En el caso de los servicios de infraestructura, la particularización significa permitir que los usuarios desplieguen recursos virtuales del tipo deseado a los que tendrán

  un acceso privilegiado como superusuarios. Otros modelos de más alto nivel ofrecen menos flexibilidad y no son apropiados para una computación de propósito general, aunque también poseerán un cierto nivel de particularización.

- *Agrupación de recursos*: Los recursos de computación del proveedor se agrupan para servir a múltiples usuarios utilizando un modelo multi-distribuido con diferentes recursos físicos y virtuales asignados dinámicamente conforme a la demanda del consumidor. El cliente no tiene control ni conocimiento de la ubicación física de los recursos proporcionados, con la excepción del centro de datos o región del proveedor.

Los servicios de computación en la nube se dividen en tres tipos, de acuerdo al nivel de abstracción de la capacidad proporcionada y al servicio proporcionado por el proveedor: Infraestructura como Servicio (*Infrastructure as a Service o IaaS*), Plataforma como Servicio (*Platform as a Service o PaaS*) y Software como Servicio (*Software as a Service o SaaS*). Estos niveles de abstracción proporcionan un arquitectura de tres capas, donde la capa superior emplea servicios de la capa inferior. El software de más bajo nivel gestiona los recursos físicos y las máquinas virtuales desplegadas en los mismos, proporcionando además la funcionalidad necesaria para llevar a cabo las tareas de contabilidad y facturación bajo un modelo de pago por uso. Los entornos de desarrollo en la nube se construyen sobre dichos servicios de infraestructura, para ofrecer el desarrollo de aplicaciones científicas o empresariales y el despliegue de las mismas. En la parte superior, y una vez desplegadas en la nube, dichas aplicaciones podrán ser utilizadas por los usuarios.

6.5.1. Infraestructura como Servicio (IaaS)

Una Infraestructura como Servicio (IaaS) proporciona un conjunto de recursos virtualizados, bajo diferentes sistemas operativos, de computación, almacenamiento y comunicación. Estas infraestructuras permiten un aprovisionamiento de recursos a demanda por parte de los usuarios, quienes ejecutan un software personalizado. Mientras que el proveedor se encarga del mantenimiento del hardware, el usuario gestiona e instala las aplicaciones software que desee. Claramente, los servicios de Infraestructura representan la capa más baja de los sistemas de computación en la nube.

Un ejemplo de IaaS es Amazon Web Services (AWS), cuyo servicio denominado Elastic Compute Cloud (EC2) [320] ofrece máquinas virtuales con un software que puede ser particularizado como si de una máquina física se tratase. Los usuarios pueden realizar tareas de administración sobre la infraestructura, tales como ponerla en marcha o detenerla, instalar paquetes software concretos, incorporar discos virtuales, configurar cortafuegos y gestionar los permisos de acceso. Otro ejemplo es Microsoft Azure Compute, que veremos en la sección 6.6.

6.5.2. Plataforma como Servicio (PaaS)

Las Plataformas como Servicio (PaaS) ofrecen un nivel de abstracción situado por encima de la capa IaaS, proporcionando una plataforma de cómputo y un conjunto de soluciones software a partir de las cuales los desarrolladores crean y despliegan sus aplicaciones, sin tener el control de los detalles de la infraestructura. Google App Engine o Microsoft Azure son ejemplos de PaaS que ofrecen el entorno necesario para desarrollar, ejecutar y albergar aplicaciones. Otro ejemplo, es el proyecto VENUS-C, que describiremos en el apartado 6.7.

6.5.3. Software como Servicio (SaaS)

Claramente, las aplicaciones representan la parte superior de un sistema software en la nube, las cuales pueden ser accedidas como servicios por parte de los usuarios, bien sea a través de un navegador Web o de una aplicación cliente.

Cada vez más, los usuarios tienden a usar aplicaciones on-line que no necesitan instalar localmente en sus computadores, sin que ello suponga una pérdida de funcionalidad. De hecho, este modelo de uso, denominado SaaS, favorece a los usuarios, ya que siempre tienen acceso a la última versión de una aplicación y no tienen que preocuparse por las diferentes actualizaciones de la misma ni por la gestión de la infraestructura en la cual se ejecuta. A su vez, el modelo también beneficia a los proveedores de las aplicaciones, ya que simplifica el proceso de desarrollo y elimina el mantenimiento de distintas versiones. Por contra, el usuario necesita una conexión permanente a internet para poder utilizar la aplicación.

Como ejemplos SaaS, podemos citar a los servicios Cloud desarrollados bajo el paraguas de esa tesis doctoral o a otros muy conocidos como Gmail, Google Maps, YouTube o Dropbox.

6.5.4. Tipos de infraestructuras Cloud

De acuerdo al modelo de despliegue, las infraestructuras Cloud se clasifican en públicas, privadas (on-premises) o híbridas. Las infraestructuras *públicas* son aquellas que están a disposición de cualquier usuario, para ser usadas a demanda y bajo el modelo de pago por uso. Aunque a priori se tienda a pensar que la computación en la nube tiene lugar únicamente bajo infraestructuras públicas, hay que considerar que existen otros modelos de despliegue de acuerdo a su ubicación física. Es el caso de las infraestructuras *on-premises*, las cuales siempre pertenecen a una determinada empresa u organización, para el uso exclusivo del personal que forme parte de la misma. En la mayoría de los casos, desplegar una infraestructura privada supone virtualizarla y ejecutar el software Cloud que sea necesario.

Finalmente, las infraestructuras *híbridas* emplean una plataforma *on-premises* suplementada con la capacidad de cómputo de una plataforma pública, la cual se puede emplear cuando la carga del sistema sobrepasa la capacidad de la plataforma privada. Otro ejemplo de uso de una plataforma híbrida es aquél en el cual una organización ejecuta aplicaciones fundamentales y opera y guarda datos sensibles en su plataforma privada y externaliza la ejecución del resto de aplicaciones a la plataforma pública.

Aunque son muchos los diferentes proveedores de servicios Cloud, a continuación nos centraremos en Microsoft Azure, por ser una de las plataforma Cloud utilizada en el marco de esta tesis doctoral.

6.6. Microsoft Azure

Microsoft Azure [19, 321] proporciona una plataforma computacional y un conjunto de herramientas software, bajo el nombre de Azure SDK (Software Development Kit), para desarrollar, desplegar y gestionar aplicaciones en la nube en diferentes lenguajes de programación como .NET, Java o Python, entre otros. Está formado por servicios de cómputo, de datos, de aplicación y de red.

6.6.1. Modelos de ejecución

Entre los diferentes modelos de ejecución que componen Azure encontramos *Azure Virtual Machines*, *Azure Websites* y *Azure Cloud Services*. Un servicio Cloud de Azure contiene o bien una colección de máquinas virtuales o bien un conjunto de instancias de tipo Web y Worker roles, como veremos a continuación.

Azure Virtual Machines representa, junto con el servicio *Virtual Networks* de interconexión entre las máquinas, el núcleo central del comportamiento de Azure como IaaS. Soporta el despliegue de máquinas virtuales (MVs), en un centro de datos de Microsoft, bajo los sistemas operativos Windows Server o Linux, a partir de una imagen virtual creada por el usuario o disponible en el catálogo.

Bajo este modelo, el usuario posee un control elevado en lo que respecta a la configuración de la MV, quien es responsable de la instalación, configuración y mantenimiento del software a instalar sobre ella. Adicionalmente, las MVs poseen dos tipos de disco: uno, obligatorio, donde reside el sistema operativo (Windows o Linux) y otro, opcional, donde se almacenan las aplicaciones y los datos.

A nivel IaaS, un servicio Cloud de Azure es un contenedor de múltiples MVs que ofrece seguridad, gestión y conectividad de red. El punto de entrada al servicio Cloud vendría dado por su dirección IP, la cual se conoce como una dirección VIP

(IP virtual), vinculada al distribuidor de carga de Azure y quien se encargará de equilibrar dicha carga entre las diferentes MVs que lo componen. Azure posee dos tipos de MVs, llamadas *básica* y *estándar*. A su vez, cada una de ellas puede presentar un *tamaño* diferente, el cual viene dado por las características de la CPU, la memoria RAM y el tipo de disco elegido. Azure ofrece elasticidad, a fin de incrementar o disminuir el número de MVs en función de la carga, siempre de manera automática y de acuerdo a unas reglas preestablecidas.

Azure Websites es un servicio Cloud que permite desplegar una aplicación web, escrita en .NET, Java, PHP, Node.js o Python, para ser utilizada por los usuarios desde internet. Se trata de un servicio PaaS, motivo por el cual el usuario no tiene que preocuparse por la gestión de las MVs en las que se ejecute la aplicación. Azure proporciona elasticidad, aumentando o reduciendo el número de instancias, así como el equilibrado de la misma en las diferentes MVs.

Otro tipo distinto de servicio Cloud es el que está compuesto por diferentes recursos computacionales que, de manera colectiva, interaccionan entre sí o con el exterior y procesan información. A los recursos o instancias de este tipo se le denominan *web role* y *worker role*. La única diferencia entre ellas es que las instancias de tipo web role ejecutan IIS (*Microsoft Internet Information Services*). Mientras que las instancias de tipo web role se usan para aplicaciones web, así como para toda clase de aplicaciones que requieran IIS, las instancias de tipo worker role son de propósito general y se suelen dedicar a procesar tareas no interactivas. Es habitual, como muestra la figura 6.5, que las instancias web role y worker role se comuniquen por una cola, en la cual las web role posicionan las peticiones que van llegando al servicio Cloud, en forma de mensajes, y las worker role las leen y las procesan.

El usuario escoge el tamaño de máquina virtual que desea utilizar, determinando el número de núcleos computacionales de los que consta, la cantidad de memoria RAM y el tipo de disco a utilizar. Cabe la posibilidad de escalar el servicio Cloud e incorporar automáticamente nuevas instancias o eliminar algunas de las ya existentes. Para ello, ello el usuario puede configurar el periodo de tiempo en el que desea que se desplieguen nuevas máquinas, o el escalado puede venir dado en función del porcentaje de uso de las instancias actuales o del número de trabajos en cola.

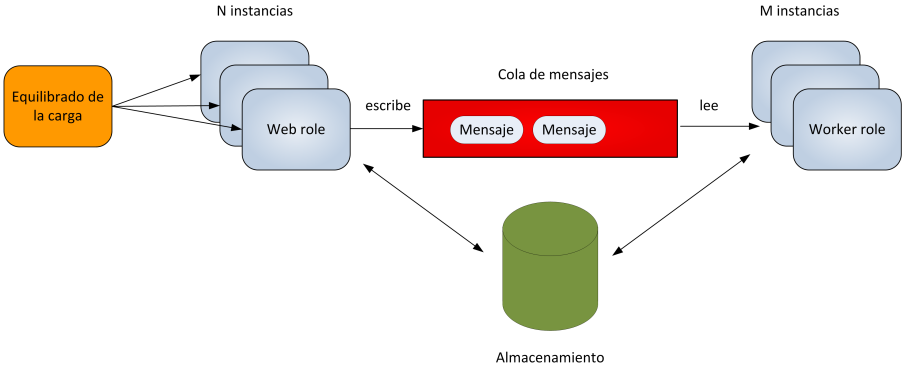


Figura 6.5: Aplicación habitual con web roles y worker roles.

6.6.2. Gestión de los datos

El servicio de gestión de datos de Microsoft Azure proporciona un almacenamiento seguro, escalable, redundante y de alta disponibilidad. Está compuesto por cuatro tipos de servicios: blobs, tablas, colas y un sistema de ficheros compartido.

Los ficheros *blob* (binary large object) son archivos del estilo de los que todos conocemos y que se almacenan en cualquier ordenador. Es posible crear contenedores, similares a las tradicionales carpetas, en los cuales se guardan los blobs, con la limitación de que sólo se permite un nivel de jerarquía, lo cual implica que un contenedor no puede incluir a otro. Son accesibles mediante una URL, mediante algunas de las librerías disponibles en el SDK de Azure para utilizar como cliente o mediante un interfaz REST (Representational State Transfer) [322], el cual proporciona una semántica de interfaz uniforme para el acceso a los recursos frente al uso de interfaces arbitrarias específicas de la aplicación.

Las *tablas* representan un sistema de almacenamiento que permite guardar importantes cantidades de datos semi-estructurados y no relacionales y acceder a los mismos de forma eficiente mediante un modelo basado en el esquema de clave-valor. Cada entidad que se almacena en una tabla puede estar compuesta por un conjunto de datos heterogéneos diferente a los de otras entidades.

El servicio de *colas* se emplea para almacenar mensajes de hasta 64 KBytes que se procesan de forma asíncrona mediante un protocolo FIFO donde, como ejemplo, un servicio pone en la cola tareas de cómputo, a modo de mensajes, y otro los toma de la cola y los procesa.

Adicionalmente, Azure habilita la posibilidad de configurar y gestionar un sistema de ficheros compartido al que diferentes máquinas virtuales tienen un acceso común.

6.7. El proyecto VENUS-C

VENUS-C (Virtual multidisciplinary EnviroNments USing Cloud Infrastructures) [323] fue un proyecto financiado por el séptimo Programa Marco de la Comisión Europea en el cual científicos y proveedores de tecnología trabajaron conjuntamente para desarrollar una infraestructura de computación en la nube dirigida a la comunidad científica y a las PyMES europeas.

VENUS-C dio lugar a una plataforma fácil de usar y orientada a servicio que permite a los investigadores emplear la computación en la nube en multitud de campos de la ciencia y la ingeniería. Siete escenarios de distintos ámbitos formaron parte del proyecto, entre los que se encontraba el dedicado al análisis estructural, bajo el cual se han desarrollado los servicios Cloud que forman parte de esta tesis doctoral. Adicionalmente, el proyecto contó con 15 aplicaciones piloto y 5 experimentos que demostraron la viabilidad de los desarrollos.

VENUS-C está asociado al concepto de PaaS, lo que abstrae a los desarrolladores de software de la infraestructura hardware subyacente, centrándose en la funcionalidad de sus aplicaciones, a las cuales se dota de elasticidad, seguridad y alta disponibilidad.

El proyecto tuvo como objetivo facilitar la migración de una aplicación ya existente a la nube y garantizar la interoperabilidad entre diferentes infraestructuras. En este sentido, VENUS-C ofrece un servicio web con un interfaz que cumple con el estándar OGSA-BES (Basic Execution Service) [324], además de una definición de trabajos basada en JSDL (Job Submission Description Language) [325] y librerías cliente disponibles para Java y .NET. Mientras que la especificación OGSA-BES describe un interfaz basado en servicios Web, estandarizado por el Open Grid Forum, que incluye la creación, monitorización y el control de las tareas computacionales, el estándar JSDL se emplea para describir los requerimientos de dichas tareas para ser ejecutadas en recursos remotos pertenecientes

a infraestructuras Grid o Cloud.

La plataforma de VENUS-C está compuesta por varios componentes que proporcionan diferentes servicios. Entre ellos destacan el *Gestor de Trabajos* y el *Gestor de Datos*. El Gestor de la Ejecución de los Trabajos, denominado *Programming Model Enactment Service* (PMES) [326], permite que los usuarios se aprovisionen de recursos computacionales, envíen tareas al Cloud y monitoricen el estado de las mismas. Por otro lado, el Gestor de los Datos es el responsable de transferir los datos de entrada al Cloud y de recoger los resultados, incorporando la especificación CDMI (Cloud Data Management Interface) [327] propuesta por la SNIA (Storage Networking Industry Association). Dicho estándar especifica el interfaz, basado en REST, para acceder a los datos almacenados en la nube. Entre el resto de servicios que forma parte de VENUS-C se encuentran aquellos dedicados a la elasticidad, a la contabilidad y la facturación de los recursos consumidos por los usuarios.

Los dos componentes principales que implementan la gestión de los trabajos, tal y como se muestra en la figura 6.6, son el Generic Worker, desarrollado por Microsoft Advanced Technology Labs, y COMPSs Enactment Service [328], implementado por el Centro de Supercomputación de Barcelona (BSC), ambos socios del proyecto. Cada componente posee su propio modelo de programación. Mientras que el primero se ejecuta en máquinas virtuales basadas en Windows Server pertenecientes a la plataforma de Microsoft Azure, el segundo lo hace en máquinas virtuales bajo el sistema operativo Linux, desplegadas en el BSC, el Royal Institute of Technology (KTH) y la empresa Engineering Group.

VENUS-C proporciona, por tanto, un API para interactuar con PMES basado en .NET, para enviar trabajos a la infraestructura gestionada por el Generic Worker, o en Java, para enviar los trabajos a la plataforma gobernada por COMPSs.

6.7.1. Generic Worker

El Generic Worker (GW) [329, 330] es una implementación web role para Microsoft Azure que facilita el despliegue y la invocación remota de una aplicación

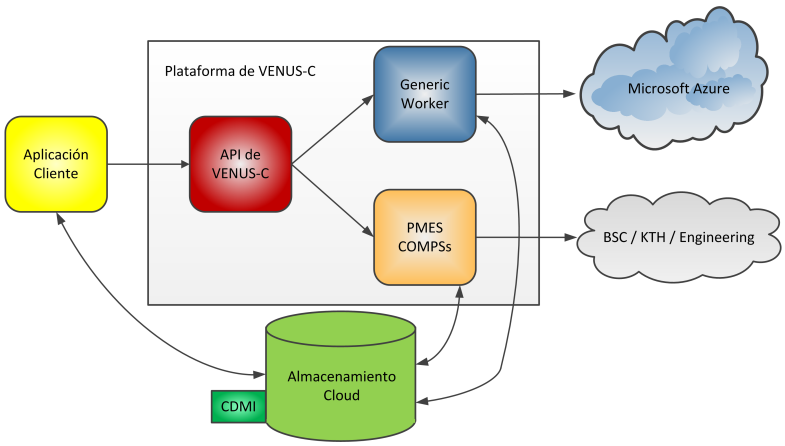


Figura 6.6: Arquitectura básica de VENUS-C.

en la nube residente en la infraestructura de Azure. Dicho componente atiende peticiones por parte de los clientes para registrar un trabajo, obtener su estado, finalizarlo, además de realizar toda la gestión necesaria para procesarlo. Es posible desplegar múltiples instancias del GW trabajando conjuntamente, ejecutándose cada una de ellas en una MV de Azure. El ciclo de vida de un trabajo comienza con el envío del mismo por parte del cliente a través del servicio PMES. Dicho trabajo incluirá una descripción de la aplicación a ejecutar, incluyendo su fichero ejecutable y los parámetros necesarios para su ejecución, además de la ubicación, en el servicio de almacenamiento Cloud, de la aplicación y de sus ficheros de entrada, junto con la localización futura de los ficheros de salida. Por su parte, la instancia de GW que atienda la petición de procesamiento del trabajo lo incorporará a una cola de trabajos pendientes.

Cualquiera de las instancias existentes de Generic Worker que recupere un trabajo no procesado de dicha cola revisará si está listo para su ejecución, lo cual se produce cuando todos sus ficheros de entrada están disponibles. Si eso es así, dicha instancia procede a procesar la tarea, copiando al disco local de su MV los ficheros de la aplicación y sus datos de entrada. El siguiente paso supone, evidentemente, la ejecución de la aplicación por parte de dicha instancia, así como el cambio del estado del trabajo, el cual pasa a *en ejecución*. A partir de ese momento, la instancia del GW monitoriza el trabajo y espera a que finalice. Si acaba satisfactoriamente, transfiere los ficheros de salida a las localizaciones de almacenamiento especificadas en la descripción del fichero. Cuando se completa

dicha transferencia, el GW actualiza el estado del trabajo a *finalizado* y recupera otro trabajo sin procesar de la cola. En caso de error en la ejecución, el estado del trabajo será *fallido*. Conviene aclarar que todo el movimiento de datos se realizará mediante la implementación de CDMI, gracias a ficheros de tipo blobs.

Para acceder al GW, se lleva a cabo un proceso de autorización y autenticación de usuarios, empleando el nombre de usuario y la contraseña. Por otro, las comunicaciones se protegen de acuerdo a la especificación WS-Security de OASIS.

De manera añadida, el GW ofrece un interfaz adicional al administrador del servicio a través del cual se puede configurar la política relativa a la elasticidad. Finalmente, también permite la suscripción a notificaciones, con el objetivo de que el proceso cliente consulte periódicamente el estado de sus tareas e informe, en consecuencia, al usuario.

6.7.2. PMES COMPSs

COMP Superscalar (COMPSs) [331] es un entorno de programación cuyo principal objetivo es facilitar el desarrollo de aplicaciones en entornos distribuidos y mejorar las prestaciones de las mismas explotando su concurrencia inherente. Está basado en lograr que el programador no sea consciente ni del entorno de ejecución de una aplicación ni de los detalles de paralelización de la misma. Únicamente debe seleccionar qué partes de una aplicación secuencial se ejecutarán remotamente, lo que en la terminología de COMPSs se denominan las *tareas*, dando lugar a una paralelización automática de la misma sobre una infraestructura computacional Grid o Cloud.

La implementación de COMPSs como interfaz BES, o COMPSs Enactment Service (en adelante PMES COMPSs), incluye un *Gestor de Trabajos* (Job Manager) y un *Distribuidor de Trabajos* (Job Dispatcher). Cuando llega al servicio PMES COMPSs una petición de procesamiento de un nuevo trabajo, el Gestor de Trabajos lo registra en la cola de las tareas a procesar. Periódicamente, alguno de los hilos que forman parte del Distribuidor de Trabajos tomará la tarea de la cola y pondrá en marcha la creación una nueva MV por medio de OpenNebula [332] o EMOTIVE (Elastic Management of Tasks in Virtualized Environments)

[333].

En el momento en el cual esta máquina esté lista, el Distribuidor de Trabajos desplegará en ella la ejecución de COMPSs y copiará en su disco local los diferentes ficheros que configuran la aplicación y los datos de entrada de la misma, desde el repositorio de almacenamiento especificado por el usuario, a través de un servicio CDMI. Esta nueva instancia de COMPSs, encargada de ejecutar el trabajo, posee la capacidad de planificar la asignación de las tareas concurrentes que forman parte del mismo a un conjunto de recursos computacionales, o *workers*, de los que se dispone, siendo todos ellos MVs creadas previamente a demanda.

A su vez, esta instancia de COMPSs puede incrementar el número de workers, desplegando para ello nuevas MVs, si detecta que la carga de los recursos disponibles es demasiado alta o si ninguno de ellos satisface las restricciones computacionales de las tareas que componen el trabajo. Una vez que dichas MVs están listas para utilizarse, se añadirán al conjunto de *workers* disponibles. De manera alternativa, una instancia de COMPSs también puede liberar MVs, en caso de que exista un nivel bajo de carga en los recursos o incluso cuando un trabajo finalice su ejecución. Antes de eliminar un worker, la instancia de COMPSs garantiza que no se está ejecutando ninguna tarea en él y que se ha realizado una copia de los resultados generados por dicho recurso al disco local de la MV en el que se está desplegado COMPSs. Conviene aclarar que dicho aumento o reducción en el número de MVs se puede realizar de modo automático por parte del sistema o puede ser una petición expresa del usuario a la hora de enviar un trabajo para su ejecución.

Una vez que se ha completado el trabajo, los datos de salida se llevan al almacenamiento especificado en su descripción JSDL por parte del usuario, eliminando de la plataforma la MV que ha albergado la ejecución de la instancia de COMPSs.

Como inconveniente, hay que decir que el API de PMES COMPSs no contemplaba la suscripción a notificaciones, lo cual no nos impidió monitorizar e informar al usuario acerca del porcentaje de progreso de una simulación a lo largo del tiempo.

Capítulo 7

Implementación en Paralelo del Simulador Estructural

Este capítulo está dedicado a la implementación paralela del Simulador Estructural, el cual está basado en el modelo de programación paralela multinivel gracias a la combinación de MPI y OpenMP, tanto en lo que se refiere al cálculo estático como a las diferentes variantes del cálculo dinámico. Como veremos, el Simulador está basado en PETSc y emplea MUMPS, PaStiX, PARDISO, *hypre* y el propio PETSc para resolver los sistemas de ecuaciones lineales, además de SLEPc y ARPACK para resolver los problemas de valores propios. El particionado de la malla se lleva a cabo mediante la librería METIS, de acuerdo a dos estrategias de reparto de los elementos estructurales bien diferenciadas, las cuales determinarán el volumen de las comunicaciones en las diferentes etapas del cálculo.

7.1. Introducción

El esquema general de funcionamiento del Simulador Estructural implementado viene recogido en la figura 7.1. Como dato de entrada, tendremos un archivo

que almacena la siguiente información:

- Las coordenadas espaciales de los nudos.
- Los nudos vértices de cada barra o elemento finito.
- Las propiedades de las secciones que forman las barras, tales como inercias, módulo de torsión, etc. o propiedades de los elementos finitos 2D, como su espesor.
- Las características de los materiales que componen la estructura, como los módulos de deformación longitudinal y transversal, el módulo de Poisson, el peso específico, el coeficiente de dilatación, etc.
- La definición de los apoyos de la estructura.
- Las cargas estáticas aplicadas sobre los nudos, las barras y los elementos finitos.
- Las cargas dinámicas genéricas aplicadas sobre los nudos.
- La carga sísmica aplicada expresada por medio de un acelerograma o un espectro de respuesta.

Para simplificar, a dicho fichero con los datos de entrada le llamaremos simplemente el fichero con la *geometría*, de aquí en adelante. Como dato de salida, se proporcionarán diferentes resultados, siempre dependientes del tipo de análisis realizado. Cabe la posibilidad de almacenar los resultados en un mismo fichero o en ficheros independientes. Así, en un análisis estático, dichos archivos de salida almacenarán:

- Los desplazamientos en los nudos.
- Las reacciones en los apoyos.
- Los esfuerzos y las deformaciones en los puntos intermedios de las barras.
- Los esfuerzos en los nudos de los elementos finitos 2D y las deformaciones unitarias y las tensiones en los nudos de cualquier elemento finito.

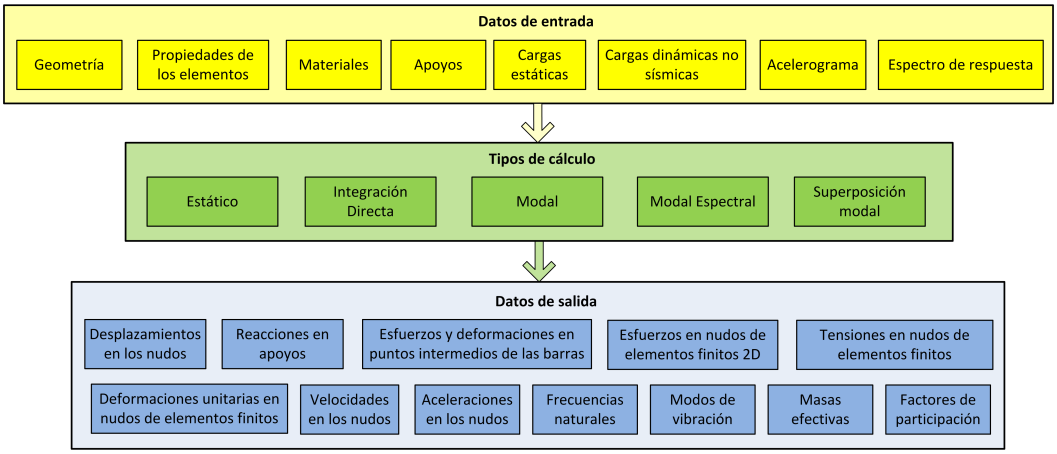


Figura 7.1: Esquema de funcionamiento y componentes del Simulador Estructural.

En el caso de un análisis dinámico por integración directa, dichos resultados se generarán para cada instante de tiempo, junto con las velocidades y las aceleraciones en los nudos. Si trabajamos con ficheros independientes, cada uno de ellos almacenará un tipo de resultado concreto para un instante de tiempo determinado.

Si llevamos a cabo un análisis modal, obtendremos las frecuencias naturales, los modos de vibración y las masas efectivas. Si hablamos de un análisis por superposición modal, los resultados serán idénticos a los obtenidos mediante los métodos de integración directa, añadiendo los factores de participación de cada modo. Por último, en el caso de un análisis modal espectral, los resultados serán los mismos que en un análisis estático, tras combinar los resultados obtenidos para cada modo de vibración, junto con las velocidades y las aceleraciones en los nudos y los factores de participación de cada modo.

El primer paso, antes de realizar el desarrollo de la versión paralela del Simulador Estructural, ha consistido en implementar en secuencial los algoritmos que nos permiten llevar a cabo un análisis lineal estático o dinámico (modal, modal espectral, superposición modal e integración directa en el tiempo) tal y como describimos en los capítulos 3 y 4, incluyendo todas y cada una de distintas etapas de las que consta cada tipo de cálculo, así como los diferentes métodos de integración o combinación modal mencionados. Cuando hablamos de versión secuencial, nos estamos refiriendo a la versión que se ejecuta empleando un único elemento de

procesamiento.

En dicha implementación, hemos procurado minimizar el consumo de memoria, con el objetivo de simular estructuras de gran dimensión. Al mismo tiempo, hemos intentado reducir en la medida de lo posible los tiempos de simulación. Para ello, en cualquier etapa del cálculo:

- Se ha minimizado el número de operaciones aritméticas que aparecen en cualquier expresión matemática implementada.
- Las operaciones de algebra lineal entre matrices densas se llevan a cabo invocando a funciones de las que dispone la librería BLAS. Es el caso, por ejemplo, de los productos de matrices que pueden aparecer al generar la matriz de rigidez de una barra o de un elemento finito, las cuales se realizan invocando a funciones de BLAS de nivel 3.
- Se ha cuidado el diseño de las estructuras de datos, así como el de los vectores y las matrices intermedias con las que trabajamos, para que coincida el orden en el que las recorren los algoritmos con aquél en el que se almacenan en memoria. De este modo, conseguimos reducir sustancialmente los tiempos de cálculo, al minimizar los movimientos de datos entre la memoria principal y la memoria cache.

Los algoritmos implementados están escritos en lenguaje C y están basados por completo en la librería PETSc. El ensamblaje de los elementos que componen las matrices de rigidez y de masa o los vectores de cargas estáticas y dinámicas se ha llevado a cabo invocando a funciones del propio PETSc. Lo mismo podemos decir de las operaciones de algebra lineal que haya que realizar entre matrices dispersas y/o vectores, donde hemos empleado siempre las funciones que dicha librería nos facilita.

Es más, en el momento de determinar los desplazamientos de los nudos de la estructura y resolver el sistema de ecuaciones correspondiente, podemos emplear los métodos iterativos de PETSc, acompañados de sus propios precondicionadores o de los implementados en *hypre*, o podemos invocar a librerías externas como MUMPS, PaStiX o PARDISO y resolver dichos sistemas mediante métodos directos, gracias a los interfaces que PETSc nos proporciona.

A la hora de realizar la implementación paralela, hemos escogido el modelo de programación paralela multinivel, el cual nos permite aprovechar la potencia computacional que nos ofrecen los multiprocesadores de memoria compartida distribuida. Para ello, los desarrollos están basados conjuntamente en MPI y en OpenMP. Es por ello que nuestro programa en paralelo se ejecutará en un número de procesos o tareas MPI (que se numerarán desde el identificador 0 en adelante), los cuales se comunicarán mediante paso de mensajes, y donde cada proceso creará un número de hilos de ejecución (que también se numerarán comenzando en 0) desplegados gracias a OpenMP, que se comunicarán por memoria compartida. Se abre en consecuencia un abanico de posibilidades a la hora de ejecutar el Simulador Estructural desarrollado, dependiendo de la plataforma computacional de destino, las cuales incluyen:

- Máquinas con un único elemento de procesamiento: Emplearemos la versión secuencial de los algoritmos.
- Supercomputadores de memoria compartida y PCs multinúcleo: Por defecto, lanzaremos un único proceso MPI que creará múltiples hilos de ejecución mediante OpenMP.
- Supercomputadores de memoria distribuida: Emplearemos múltiples procesos MPI pero cada uno de ellos con un solo hilo de ejecución.
- Supercomputadores de memoria compartida distribuida: Por defecto, ejecutaremos un único proceso MPI por nodo que creará múltiples hilos de ejecución mediante OpenMP.

En cualquier caso, dichas posibilidades de ejecución no son rígidas, pudiendo por ejemplo lanzar varios procesos MPI en máquinas de memoria compartida o en cada uno de los nodos de una máquina de memoria compartida distribuida. No obstante, es importante descartar que la última versión de PETSc no incorpora técnicas multihilo en la implementación de ninguna de sus funciones, motivo por el cual no podremos sacar partido del paradigma de memoria compartida al utilizarlo, lo que nos habría resultado de mucha utilidad a la hora de realizar operaciones de álgebra lineal entre matrices y vectores. Del mismo modo, PETSc no es seguro frente a la invocación simultánea a sus funciones por parte de diferentes

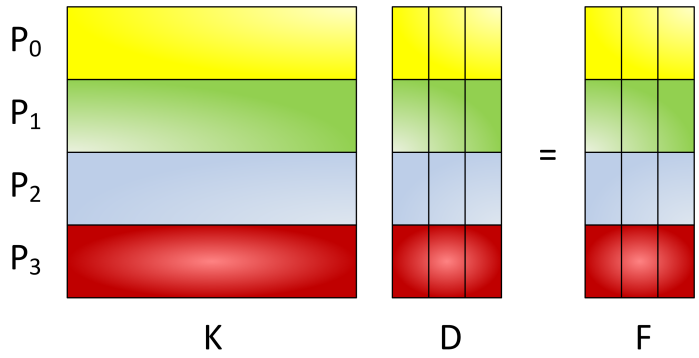


Figura 7.2: Partición en PETSc de la matriz y los vectores que forman parte de un sistema de ecuaciones lineales entre 4 procesos.

hilos de ejecución, motivo por el cual tendremos que tener la precaución de que ello no ocurra.

PETSc trabaja con un formato de matrices y vectores en paralelo que implica un particionado por bloques de filas consecutivas entre los procesos, asignando a cada uno todo el conjunto comprendido entre una fila inicial y una fila final. La figura 7.2 nos muestra la distribución que tendríamos de los elementos de la matriz de coeficientes y de los múltiples vectores parte derecha y solución a la hora de resolver un sistema de ecuaciones. Evidentemente, esa distribución deberá ser idéntica a la que sigamos en las diferentes fases del cálculo de la estructura incluyendo, para comenzar, el ensamblaje de dicha matriz y vectores parte derecha.

A continuación se detallará la estrategia de paralelización que se ha seguido en los distintos tipos de cálculo estructural implementado.

7.2. Paralelización del cálculo estático

El cálculo estático de una estructura está formado por una serie de etapas recogidas en el algoritmo 3. En las diferentes apartados de esta sección, iremos abordando la paralelización, si procede, de cada una de ellas.

Algoritmo 3 Cálculo estático de una estructura

Entrada: *ntarea* (identificador de la tarea MPI)
Salida: Resultados de cálculo

si *ntarea*=0 **entonces**
    Lectura del fichero con la geometría
    Envío de la geometría al resto de tareas
si no
    Recepción de la geometría
fin si
si *ntarea*=0 **entonces**
    Particionado de la malla de la estructura
    Envío del particionado al resto de tareas
si no
    Recepción del particionado
fin si
Reordenación y clasificación de los nodos de la estructura
Generación de la matriz de rigidez
Generación del vector de cargas aplicadas
Imposición de las condiciones de contorno
Cálculo de los desplazamientos en los nudos
Obtención de solicitaciones en los extremos de las barras
Cálculo de reacciones en apoyos
Generación de esfuerzos y deformaciones en puntos intermedios de las barras
Cálculo de esfuerzos en nudos de elementos finitos 2D
Cálculo de deformaciones unitarias, esfuerzos y tensiones en nudos de elementos finitos
Escritura en disco de los ficheros de resultados

7.2.1. Lectura, envío y recepción de los datos de entrada

De entre todas la tareas disponibles, la tarea 0 será la encargada de leer el fichero la geometría y almacenarlo en memoria, enviando dicha información al resto de tareas mediante las correspondientes invocaciones a la función MPI_Bcast, a modo de difusión de mensajes de uno a todos. Podemos decir que los datos de entrada al problema estarán replicados por igual en todas las tareas que participan en el cálculo estructural.

7.2.2. Particionado de la estructura

De aquí en adelante, llamaremos *dominio*, *malla de elementos* o simplemente *mallado* al conjunto de barras y elementos finitos de diferente tipo que componen una estructura. La idea inicial que gobierna al análisis estructural distribuido consiste en dividir o descomponer la malla en grupos formados por un conjunto de nodos y elementos, llamados *subdominios*, y asignar cada uno de ellos a uno de los procesos MPI que participarán, de manera colaborativa, en el cálculo. El número de particiones coincide por tanto con el número de tareas MPI, de modo que asignaremos la partición i a la tarea i.

Sin lugar a dudas, podemos decir que el tipo y la calidad del particionado que llevemos a cabo determinará las prestaciones de la aplicación paralela desarrollada. Es por ello que debemos ser conscientes de la necesidad de que el trabajo asociado a cada subdominio sea lo más equitativo posible, consiguiendo así que todos los procesos acaben su labor de simulación al mismo tiempo. Adicionalmente, la descomposición del dominio entre los procesos debe minimizar las comunicaciones entre estos últimos, las cuales incrementan notablemente los tiempos de respuesta de la aplicación.

A la hora de dividir el mallado en los diferentes subdominios, se han implementado las dos siguientes aproximaciones, a las cuales denominaremos *particionado por corte de nodos* y *particionado por corte de elementos*. En un caso o en otro, las particiones adyacentes al corte compartirán o bien nodos o bien elementos, lo cual conlleva unas estrategias de paralelización y unas necesidades de comunicación que serán diferentes, para cada etapa del cálculo, de acuerdo al tipo de

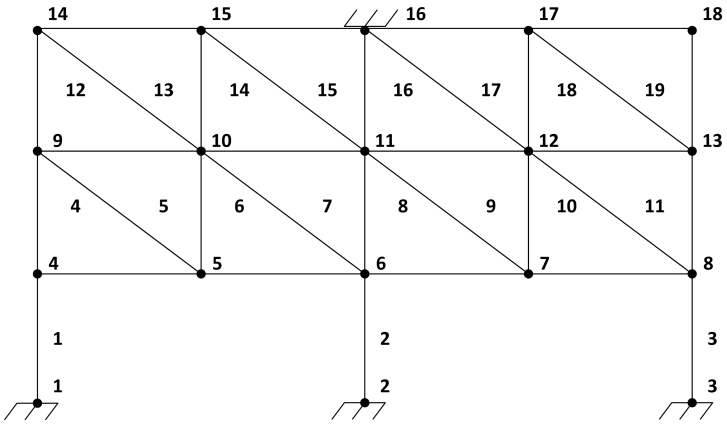


Figura 7.3: Estructura a descomponer en dos subdominios.

particionado escogido. A continuación describimos las peculiaridades de dichas aproximaciones, basándonos en el ejemplo de la figura 7.3, formado por 3 barras (numeradas de la 1 a la 3), 16 triángulos (con numeración del 4 al 19) y 18 nodos, 4 de los cuales (1, 2, 3 y 16) serán apoyos. Como hemos visto, no se distingue entre las barras y los triángulos a la hora de numerar los elementos que componen la estructura, con el objetivo de simplificar la explicación.

7.2.2.1. Particionado por corte de nodos

La forma más lógica de descomponer una malla de elementos supone dividirla a través de sus aristas o caras, dependiendo de que se traten de elementos finitos en 2D o en 3D, respectivamente. De este modo, cada elemento se asigna a una única partición, pero los nodos de dichas aristas o caras, a través de las cuales se lleva a cabo el corte, se comparten entre distintas particiones. En el caso de una barra, el corte tendrá lugar a lo largo de la misma, la cual se asignará a una u otra partición aunque ambos nodos se compartirán.

En distintas etapas del cálculo estructural, estos nodos compartidos necesitarán una contribución conjunta entre todos aquellos elementos de los que son vértices, los cuales, si están ubicados en distintas particiones, darán lugar a comunicaciones. A priori, distinguimos dos tipos de nodos en una partición. Aquellos denominados *locales*, asignados de forma exclusiva a una partición y que, por el momento, están ubicados en el interior de la misma, o nodos *compartidos*, que

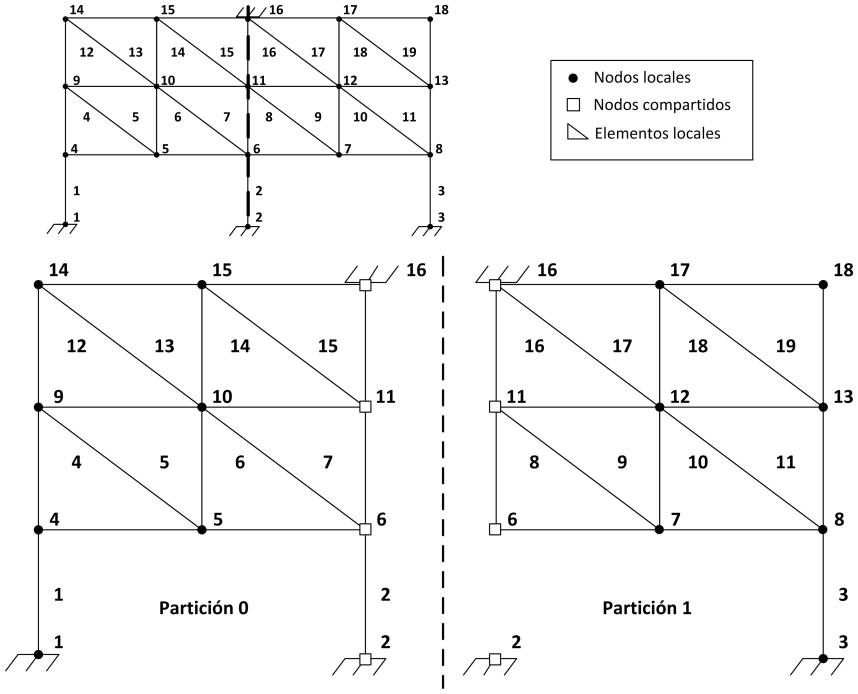


Figura 7.4: Particionado en dos subdominios mediante la técnica de corte de nodos.

pertenecerán a varias particiones y que están ubicados en la zona de corte.

La figura 7.4 nos muestra el particionado de la estructura de la figura 7.3 en dos subdominios mediante esta aproximación. Como podemos observar, el corte se realiza a través de los nodos 2, 6, 11 y 16, que pasan a estar compartidos entre ambas particiones, mientras que el resto de nodos serán locales a una u otra partición.

Para llevar a cabo el particionado de la estructura mediante esta aproximación, hemos empleado la librería METIS, la cual, como ya vimos en la sección 5.7.2.1, subdivide grafos o mallas de elementos finitos. Como primer paso para emplear dicha librería, debemos construir la malla de la estructura partiendo del fichero de entrada que incluye la geometría. En METIS, dicha malla viene dada por sendos vectores que almacenan, para cada elemento de la estructura, la numeración de sus nodos vértice. Conviene destacar que se permite que el mallado esté formado por elementos heterogéneos, como ocurrirá en nuestro caso, compuesto por barras, triángulos, tetraedros y hexaedros.

En el caso particular del particionado por corte de nodos, invocaremos a las funciones METIS_PartMeshNodal o METIS_PartMeshDual, las cuales nos proporcionan dos vectores de salida con la partición a la que pertenece cada nodo y cada elemento de la estructura. Aunque la funcionalidad será similar, el modo de proceder por parte de cada una de ellas es diferente. En primer lugar, ambas funciones convierten, a nivel interno, la malla de elementos finitos en un grafo de adyacencias, a partir del cual se realiza la partición. Sin embargo, la función METIS_PartMeshNodal genera un *grafo nodal*, donde cada vértice del grafo viene dado por un nodo de la malla y dos vértices se unen por una arista si pertenecen a un mismo elemento. Por contra, la función METIS_PartMeshDual obtiene un *grafo dual*, en el cual cada vértice del grafo se corresponde con un elemento de la malla, los cuales se unirán por una arista si los elementos a los que representan comparten algún nodo. Conviene tener en cuenta que la generación del grafo dual, a partir de la malla de elementos, será más costosa en tiempo que la del grafo nodal. Además, un grafo dual tendrá más nodos a subdividir que un grafo nodal.

A partir de ahí, el grafo se divide internamente en tantos subgrupos como se indique, mediante las funciones llamadas METIS_PartGraphRecursive o METIS_PartGraphKway, las cuales implementan los métodos de *Bisección Recursiva Multinivel* o *Particionado k-way Multinivel*, proporcionando un vector que almacena la partición a la que se asignará cada vértice del grafo. Una vez obtenido dicho vector, las funciones de METIS distribuirán las aristas del grafo.

En consecuencia, en primer lugar se distribuirán los nodos y a partir de ahí los elementos, si el grafo es nodal. Como norma general, un elemento se asignará a la partición a la que pertenezcan la mayoría de sus nodos. Por el contrario, en un grafo dual, en primer lugar se asignarán los elementos a las particiones. A continuación, cada nodo formará parte de la partición a la que estén asignados la mayoría de sus elementos a los que pertenece como vértice. Aunque ambas posibilidades son válidas, como acabamos de ver, lo más lógico parece generar el grafo dual, en lugar del nodal, para distribuir los elementos y, a partir de ellos, los nodos.

Como función objetivo del particionado, la función METIS_PartGraphKway nos da a escoger entre minimizar el número de aristas cortadas en el grafo o el volumen total de comunicación. En mallas regulares, donde todos los nodos tienen un

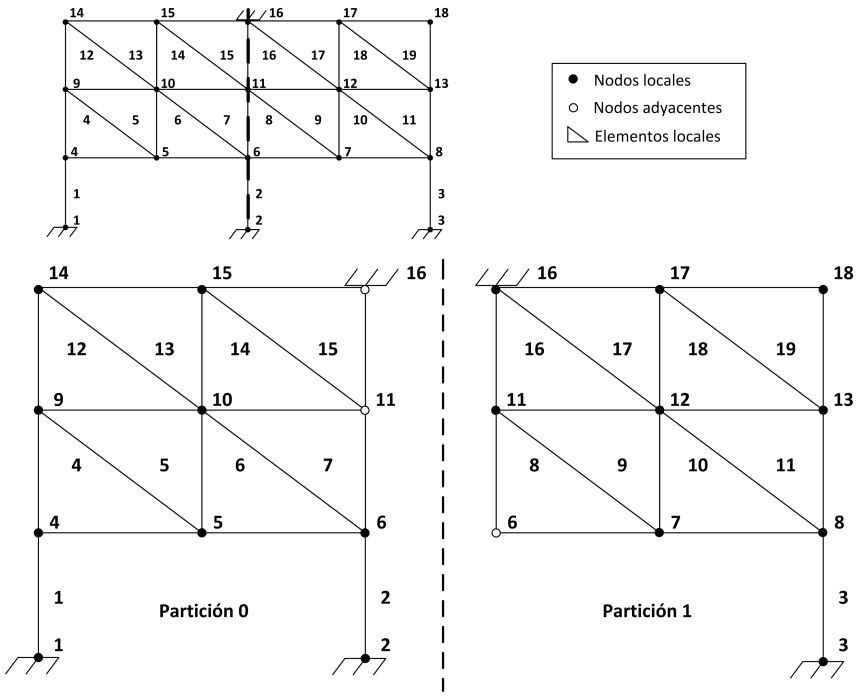


Figura 7.5: Particionado en dos subdominios mediante la técnica de corte de nodos y con asignación de cada nodo a una única partición.

mismo número de grados de libertad, los resultados son similares entre una función objetivo u otra. Conviene también destacar que es más rápido reducir el número de aristas cortadas que minimizar el volumen total de comunicación. En el caso de la función METIS_PartGraphRecursive, sólo se permite escoger, como función objetivo, la minimización de aristas cortadas.

Resulta evidente, por tanto, que cada nudo y cada elemento se asignarán de manera única a un subdominio. Aplicado sobre nuestra estructura de ejemplo, el resultado podría ser el que tenemos en la figura 7.5. Como puede verse los nodos 2 y 6, que eran compartidos, se han asignado finalmente a la partición 0, siendo ahora locales a la misma. Lo mismo podemos decir de los nodos 11 y 16, que han pasado a pertenecer a la partición 1.

Si consideramos única y exclusivamente a todos aquellos nodos que sean vértices de los elementos pertenecientes a una partición, denominaremos *nudos locales* a aquellos que estén asignados a la misma partición, siendo el resto los llamados *nudos adyacentes*, los cuales serán nudos locales en una partición diferente.

7.2.2.2. Particionado por corte de elementos

En este otro tipo de particionado, el corte se lleva a cabo a través de los propios elementos. Así, cada nodo se asignará a una única partición, pero los elementos cortados formarán parte, de manera conjunta, de las particiones adyacentes al corte. A dichos elementos asignados a más de una partición les denominaremos *elementos compartidos* y darán lugar a una franja fronteriza entre particiones. Los elementos que formen parte de un sólo subdominio se denominarán *elementos no compartidos*. La unión de los elementos compartidos y no compartidos de un partición forman el conjunto de *elementos locales*, asignados a dicha partición.

El inconveniente de esta aproximación es la replicación en el procesamiento de aquellos elementos finitos que pertenecen conjuntamente a distintos subdominios, lo cual parece indicar que será menos eficiente que la anterior. Sin embargo, también es cierto que este tipo de particionado repercute en una reducción de las comunicaciones.

No obstante, y dada una partición, hay que considerar que los elementos compartidos tendrán como vértices a nodos que nos serán asignados como propios, a los cuales designaremos como *nodos locales*, frente a otros que serán propiedad exclusiva de las particiones vecinas, a los que denominaremos *nodos adyacentes*.

La figura 7.6 incluye un posible particionado de la estructura 7.3 en dos subdominios. El corte se realiza a través de los elementos 2, 8, 7, 14 y 15, que pasarán a estar compartidos entre ambas particiones. Claramente, el corte realizado en dicho ejemplo no resulta ser el más apropiado, y así debería entenderse, a fin de dividir la estructura en dos. Su propósito es únicamente el de clarificar y explicar las diferentes peculiaridades que pueden presentarse bajo dicha aproximación, como veremos más adelante.

En caso de optar por un particionado por corte de elementos, la aproximación más simple es sin duda la de agrupar los nodos por subdominios de acuerdo a su numeración inicial, a la cual podemos denominar como *particionado por corte de elementos natural*. Esto supone que si nuestra malla tiene N nodos y queremos dividirla en p particiones, la primera partición estaría formada por los nodos del 1 al $\lceil N/p \rceil$, la segunda de los nodos $\lceil N/p \rceil + 1$ al $2\lceil N/p \rceil$ y así sucesivamente.

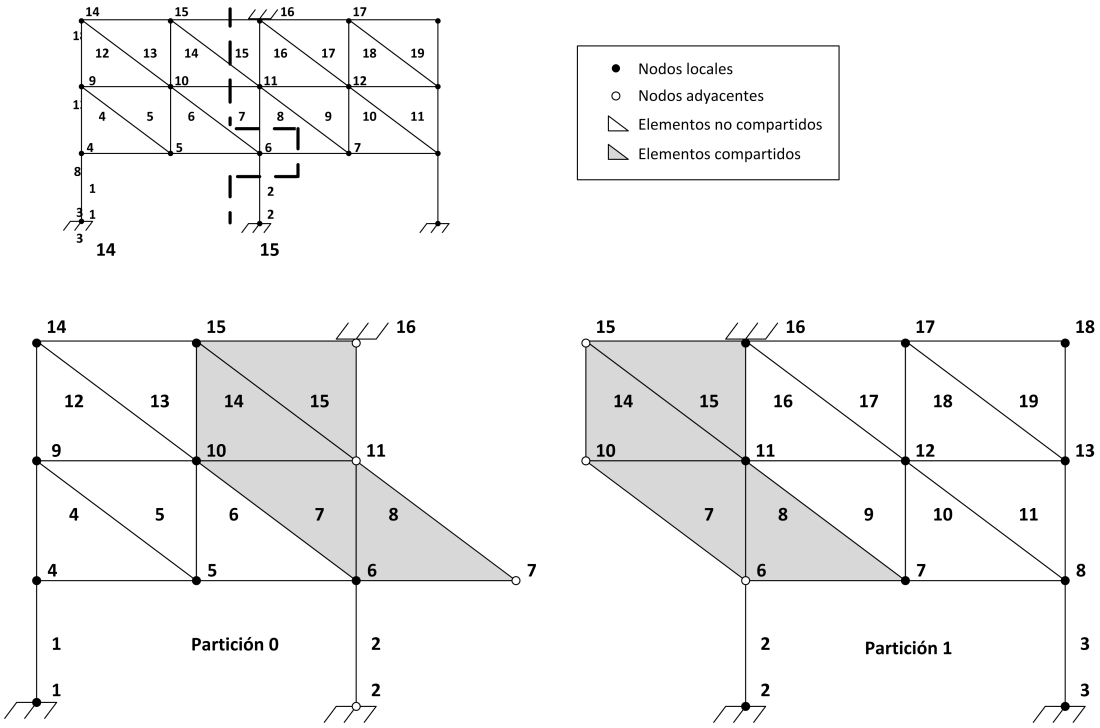


Figura 7.6: Particionado en dos subdominios mediante la técnica de corte de elementos.

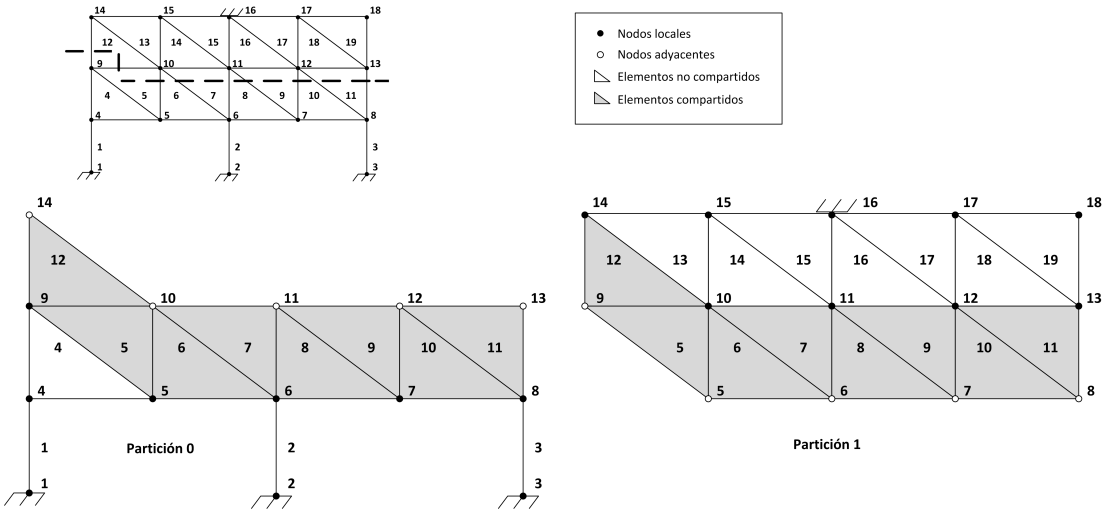


Figura 7.7: Particionado en dos subdominios mediante la técnica de corte de elementos natural.

Mientras que esta aproximación equilibra el número de nodos por partición, no tiene en cuenta, sin embargo, el coste de las comunicaciones, lo que puede dar lugar a que los algoritmos paralelos implementados ofrezcan pobres prestaciones. En la figura 7.7 podemos ver cuál sería el resultado de dicha aproximación aplicada sobre la estructura de la figura 7.3.

Mejores prestaciones deberíamos obtener si el particionado por corte de elementos lo llevamos a cabo empleando las funciones de METIS. Para ello, debemos generar un grafo a partir de la malla de la estructura y subdividirlo. De este modo, comenzaremos invocando a la función METIS_MeshToNodal, la cual nos transforma nuestra malla inicial a un grafo nodal de adyacencias que distribuimos mediante las funciones METIS_PartGraphRecursive o METIS_PartGraphKway. Dichas funciones nos devuelven, como resultado de salida, un vector con la partición a la que pertenecerá cada vértice del grafo o, lo que es lo mismo, cada nodo de la malla. A partir de ahí, debemos ser nosotros los que llevemos a cabo la asignación de los elementos de la malla a las diferentes particiones.

7.2.2.3. Tareas pendientes en el particionado

Hasta el momento, todo el trabajo realizado para obtener la asignación de los nodos y los elementos a las particiones, si nos referimos a la aproximación por corte

de nodos, o únicamente la asignación de los nodos, si hablamos del particionado por corte de elementos, las ha realizado tan sólo el proceso 0. A continuación, dicho proceso enviará los citados vectores al resto, los cuales llevarán a cabo en paralelo las siguientes tareas con cada una de las particiones que les han sido asignadas:

- *Reordenación de los nodos de la malla*: Teniendo en cuenta la distribución de las filas de una matriz o los elementos de un vector en PETSc, la cual suponía que las tareas tuvieran asignadas un conjunto de filas o de elementos consecutivos, respectivamente, será necesario reordenar los nodos de la estructura, comenzando por la partición inicial. Dada una partición cualquiera, la nueva numeración que adjudicaremos a su primer nodo se corresponderá con una unidad más de la que asignamos al nodo último de la partición anterior, recorriendo todos los nodos asignados por orden de acuerdo a la numeración de menor a mayor que ya poseían, hasta alcanzar el nodo final. Obsérvese que, en el caso de un particionado por corte de elementos natural, los nodos mantendrán su numeración inicial.

- *Distribución de los elementos a las particiones*: En el caso de la partición por corte de elementos, cada tarea deberá obtener, en paralelo, los elementos locales (compartidos y no compartidos) que formen parte de su partición, a partir de los nodos previamente asignados. Un elemento cualquiera de la malla pasará a integrarse en una partición siempre y cuando posea uno o más vértices ya asignados a dicha partición.

- *Identificación de los nodos adyacentes*: Si bien es cierto que, llegados a este punto, cada tarea sabrá cuáles son los nodos asignados a su partición (los que hemos denominado como nodos locales), también lo es que cada tarea debe identificar, en paralelo, todos aquellos nodos que sean adyacentes a su partición, tanto en el caso de un particionado por corte de nodos como por corte de elementos, puesto que dichos nodos adyacentes serán los que darán lugar a las comunicaciones.

  Las figuras 7.8 y 7.9 nos muestran el particionado final de nuestra estructura de ejemplo de acuerdo al particionado por corte de nodos o corte de elementos, respectivamente, teniendo en consideración la reordenación de

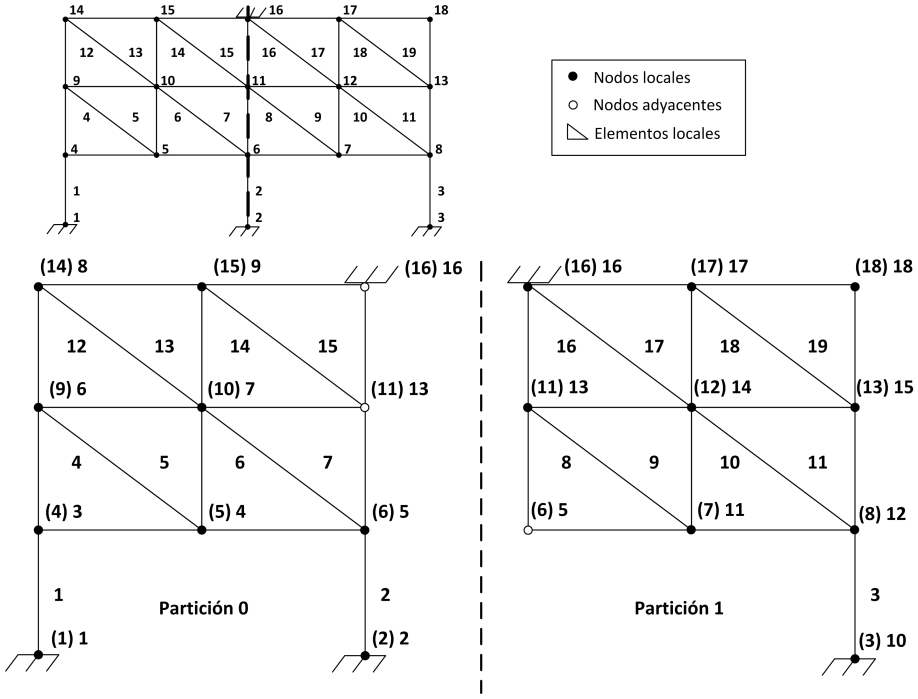


Figura 7.8: Particionado en dos subdominios mediante la técnica de corte de nodos y reordenación de nodos.

los nodos del mallado. La numeración original de los nodos vendrá dada entre paréntesis.

7.2.2.4. Métricas a emplear en el particionado

Además del número de aristas cortadas o el volumen total de comunicación que nos proporciona la librería METIS, se proporcionan a continuación diferentes métricas que nos servirán para estimar la bondad y poder comparar la calidad de diferentes particionados:

- Porcentaje de elementos solapados, entendiéndose como tal aquellos que forman parte de más de una partición. Lo podemos calcular de forma individual para cada tipo de elemento (barras, triángulos, tetraedros o hexaedros) o de forma colectiva entre todos ellos. Como es lógico, sólo tiene sentido en el caso de aplicar la técnica de particionado por corte de elementos, siendo igual

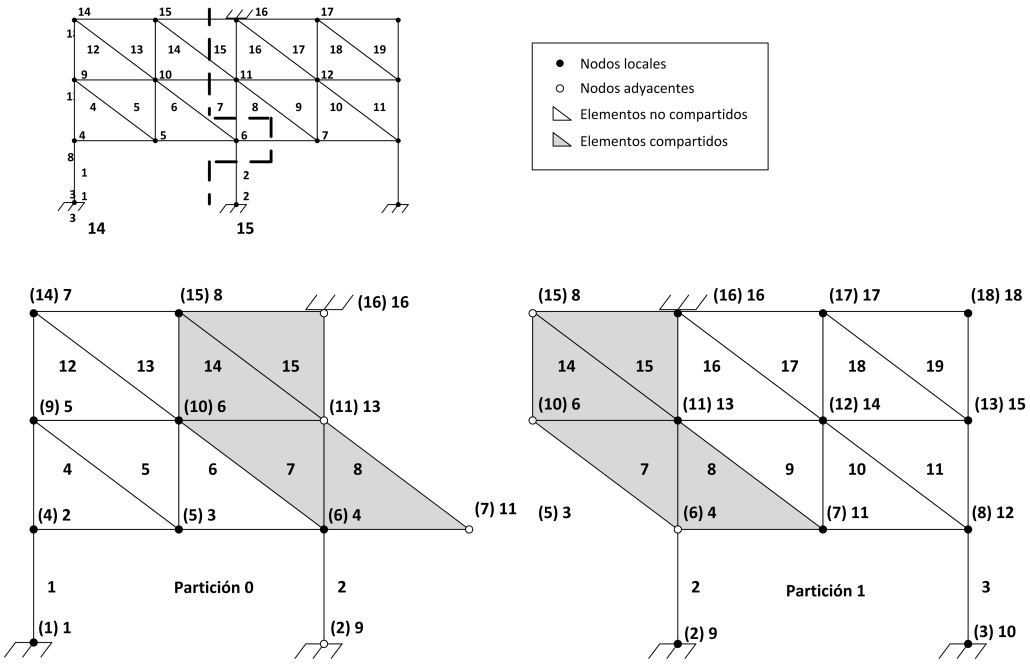


Figura 7.9: Particionado en dos subdominios mediante la técnica de corte de elementos y reordenación de nodos.

a 0 cuando empleamos un particionado por corte de nodos. Lo calculamos del siguiente modo:

$$Elementos\_Compartidos(\,\%) = \frac{\sum_{p=0}^{np-1} E_p - E}{E} * 100 \qquad (7.1)$$

siendo np el número de particiones, E el número total de elementos en la malla (en total o de un tipo concreto) y E_p el número de elementos locales (compartidos y no compartidos, bien de todo tipo o de un tipo concreto) de la partición p. Su valor ideal será del 0 %.

- Porcentaje de nodos adyacentes, siendo un nudo adyacente a una partición el que es compartido entre ella misma y otras particiones y no le pertenece como nodo local. El modo de calcularlo será el siguiente:

$$Nudos\_Adyacentes(\,\%) = \frac{\sum_{p=0}^{np-1} Na_p}{N} * 100 \qquad (7.2)$$

donde N representa el número total de nodos de la estructura y Na_p el número de nodos adyacentes de una partición p. Como en el caso anterior, una particionado será mejor a medida que dicho número se acerque al 0 %.

- Equilibrado de los elementos o porcentaje de distribución de los elementos entre las particiones, bien sea en conjunto o para cada tipo de elemento. Se obtiene a partir del número máximo de elementos que forman parte de todas las particiones:

$$Equilibrado\_elementos(\,\%) = \frac{E}{np * \underset{p=0,\,...,\,np-1}{\text{máx}} \{E_p\}} * 100 \tag{7.3}$$

  Deberá tomar valores lo más próximos al 100 %.

- Equilibrado de los nudos o porcentaje de distribución de los nodos entre las particiones, calculado gracias al número máximo de nodos locales Nl asignados a las distintas particiones:

$$Equilibrado\_nodos(\,\%) = \frac{N}{np * \underset{p=0,\,...,\,np-1}{\text{máx}} \{Nl_p\}} * 100 \tag{7.4}$$

  Idealmente, debería estar cercano al 100 %.

7.2.3. Generación de la matriz de rigidez de la estructura

Cada tarea generará en paralelo su conjunto de filas consecutivas de la matriz de rigidez de acuerdo a la nueva numeración de los nodos asignados en el particionado de la malla. Eso significa que si los índices de los nodos inicial y final asignados a una partición son respectivamente nudo_inic y nudo_fin, las filas de la matriz de rigidez que generará la tarea que gestiona dicha partición irán de la fila (nudo_inic-1)*6 a la nudo_fin*6-1. Ello será así siempre y cuando se genere la matriz *completa* de la estructura, de tamaño *6N* siendo N el número total de nodos. En nuestro ejemplo, la matriz de rigidez completa será una matriz de tamaño 108 x 108, donde las filas de la 0 a la 53 se corresponden con la tarea 0 y las filas de la 54 a la 107 pertenecen a la tarea 1, en el caso del particionado por corte de nodos.

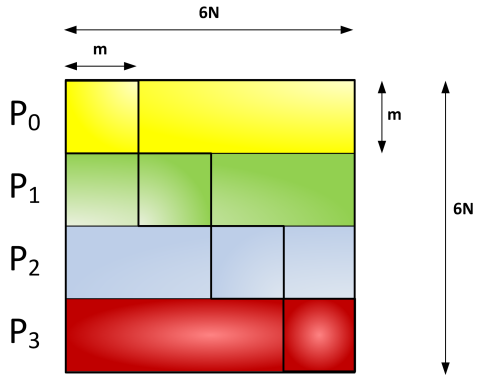


Figura 7.10: División de las filas asignadas a una tarea en submatrices diagonales y no diagonales.

Por el contrario, si no consideramos todas las ecuaciones del sistema, daremos lugar a la que llamaremos la matriz *condensada*. Esto ocurre por ejemplo si no incluimos las ecuaciones de los grados de libertad empotrados, las de aquellos grados de libertad que no se tienen en cuenta en un análisis 2D o las ecuaciones de los giros en los nudos de los elementos finitos en 3D. En nuestro caso particular, supongamos que los grados de libertad de los nudos empotrados 1, 2, 3 y 16 no forman parte del cálculo porque ya sabemos que sus desplazamientos serán nulos. Ello implicará que la matriz condensada del problema será de tamaño 84 x 84, donde la tarea 0 se encargará de las filas que van de la 0 a la 41 y la tarea 1 de las filas de la 42 a la 83, para el particionado por corte de corte de nodos.

Es imprescindible que antes de generar la matriz de rigidez se estime y reserve el espacio necesario para la misma. De no ser así, las prestaciones se degradarán sustancialmente a la hora de generarla, debido a las copias de datos que tendrán lugar en memoria ante las continuas reservas de espacio, con unos tiempos de respuesta exageradamente altos. Para ello, cada tarea obtendrá el número de valores diferentes de cero para cada una de las filas que le corresponden, el cual vendrá dado por la cantidad y el tipo de elementos (barras o elementos finitos 2D o 3D) que confluyen en cada nodo. En realidad, para cada fila debemos proporcionar dos valores, correspondientes al número de elementos diferentes de cero que están dentro o fuera de una submatriz a modo de bloque diagonal de tamaño mxm, siendo m el número de filas asignadas a la tarea, como se muestra en la figura 7.10.

PETSc nos proporciona el tipo de datos Mat para trabajar con matrices y cada tarea debe generar una secuencia de invocaciones a funciones de la librería para crear una matriz. A modo de ejemplo, se muestra dicha secuencia para generar en paralelo la matriz de rigidez K de tipo *fila comprimida a bloques* (sbaij) acorde con las librerías MUMPS o PaStiX, las cuales emplearíamos posteriormente para resolver el sistema de ecuaciones que proporciona los desplazamientos en los nudos. Se entiende que la matriz es de tamaño $6N$ x $6N$, que el número de filas locales asignadas a la tarea es m y que los vectores que almacenan el número de elementos diferentes de 0 para cada fila dentro y fuera del bloque diagonal son d_nnz y o_nnz respectivamente:

```
MatCreate(PETSC_COMM_WORLD,&K); /* Creamos una matriz genérica */
MatSetSizes(K,m,m,6*N,6*N); /* Asignamos las dimensiones */
MatSetType(K,"sbaij");          /* Indicamos el tipo de matriz */
MatSetFromOptions(K);           /* Leemos los parámetros de la línea de
                                   comandos */
MatSeqSBAIJSetPreallocation(K,1,0,d_nnz); /* Reservamos espacio */
MatMPISBAIJSetPreallocation(K,1,0,d_nnz,0,o_nnz);
MatSetOption(K,MAT_SPD,PETSC_TRUE); /* Matriz simétrica y definida
                                                    positiva */
MatSetOption(K,MAT_IGNORE_LOWER_TRIANGULAR,PETSC_TRUE); /* Sólo
    consideramos la parte triangular superior de la matriz */
```

Una vez que la matriz está creada, debemos incorporarle los elementos que forman parte de la misma. Para ello, tendremos que generar, en ejes globales, la matriz de rigidez de cada elemento que compone la estructura, además de ensamblarla en sus posiciones correspondientes, de acuerdo a lo descrito en el apartado 3.2.

La generación de la matriz de rigidez en ejes globales de cualquier elemento de la estructura (barra, triángulo, tetraedro o hexaedro) supondrá expresar las operaciones matriciales necesarias en funciones de BLAS 1 o de BLAS 3. Para ello, todas las matrices se expresarán en forma de vectores, guardando sus elementos columna a columna. Más concretamente, las funciones empleadas han sido las siguientes:

- dsymm: Realiza las operaciones $C = \alpha AB + \beta C$ ó $C = \alpha BA + \beta C$, siendo A una matriz simétrica, B y C matrices cualquiera y α y β dos números reales.

- dgemm: Se encarga de la operación $C = \alpha op(A)op(B) + \beta C$, donde A, B y C son matrices cualesquiera, α y β son números reales y $op(X)$ puede ser o bien la propia matriz X o su transpuesta X^T.

- daxpy: Lleva a cabo la operación $y = \alpha x + y$, siendo x e y vectores y α un número real.

- dscal: Escala un vector x, de modo que $x = \alpha x$, siendo α un número real.

- dcopy: Copia un vector a otro, es decir, $y = x$.

Si dispusiéramos de una implementación multihilo de BLAS, basada por ejemplo en OpenMP, las operaciones citadas se podrían llevar a cabo en paralelo.

Si nos referimos a la aportación de las barras a la matriz de rigidez global de la estructura, el procedimiento a realizar es el que viene recogido en el algoritmo 4. Como podemos ver en él, cada tarea se encarga de generar la matriz de rigidez en ejes globales de aquellas barras asignadas a su partición, llevando a cabo el ensamblaje de las mismas. Valga dicho algoritmo para extrapolar la labor similar a realizar con el resto de elementos (triángulos, tetraedros y hexaedros) que configuran la estructura.

En el caso de una partición por nodos (ver figura 7.8), la tarea 0 generará las matrices de rigidez de las barras 1 y 2 y de los triángulos 4, 5, 6, 7, 12, 13, 14 y 15, mientras que la tarea 1 generará las matrices de la barra 3 y de los triángulos 8, 9, 10, 11, 16, 17, 18, 19. Como puede observarse, no hay ningún elemento que se calcule de modo repetitivo entre ambas tareas. Sin embargo, en el caso de una partición por elementos (ver figura 7.9), la tarea 1 generará las matrices de sus elementos no compartidos, es decir de la barra 3 y de los triángulos 9, 10, 11, 16, 17, 18 y 19, pero además las de los elementos compartidos que también generará la tarea 0, como son la barra 2 y los triángulos 7, 8, 14 y 15, en un claro esfuerzo computacional añadido frente al otro tipo de particionado.

Algoritmo 4 Aportación de las matrices de rigidez de las barras asignadas a una tarea a la matriz K de rigidez global

Entrada: K (matriz de rigidez global), *barras_particion* (vector con el identificador de la barras asignadas a la tarea), *nBarrasParticion* (número total de barras asignadas a la tarea)
Salida: K (matriz de rigidez global)

```
1: #pragma omp parallel for private (b, K'^b, R^b, H^b, K^b)
2: para barra = 1 hasta nBarrasParticion hacer
3:    b=barras_particion[barra]
4:    Calcular la matriz de rigidez en ejes locales K'^b
5:    Calcular la matriz de rotación R^b
6:    si la barra b es excéntrica entonces
7:       Calcular la matriz de excentricidad H^b
8:    fin si
9:    Calcular la matriz de rigidez en ejes globales K^b
10:   Ensamblar la matriz K^b en la matriz K
11: fin para
```

En lo que respecta al paralelismo mediante OpenMP, se ha empleado el constructor paralelo #pragma omp parallel for. Dicho constructor paraleliza el bucle principal que recorre todas la barras (y si extrapolamos todos los elementos finitos) pertenecientes a una partición, dividiendo las iteraciones en los diferentes hilos de ejecución. Esto supone, por tanto, que cada hilo se encargará de generar y ensamblar la matriz de rigidez de un subconjunto de los elementos asignados a la tarea.

En el uso de OpenMP, el programador debe distinguir el ámbito o tipo de acceso a cada variable, bien sea bajo un acceso *compartido* entre los diferentes hilos o mediante un acceso *privado* por parte de cada uno de ellos. En el caso de que la variable sea compartida, únicamente existirá una instancia de la misma y cada hilo podrá leer o modificar su valor. En cambio, una variable privada estará replicada entre los hilos de modo que cada uno tendrá su propia copia local, con acceso exclusivo. De este modo, los cambios que un hilo realice sobre una variable no son visibles para los otros hilos.

Precisamente, la cláusula private es la que nos permite indicar el nombre de todas aquellas variables que sean privadas para cada hilo, siendo el resto, por defecto, variables compartidas, a excepción de la variable de control del bucle for,

en nuestro caso la variable *barra*, que por defecto será privada. Lógicamente, las variables privadas son las que se completan en cada iteración de manera individual por cada hilo y no son necesarias para iteraciones futuras, como son el identificador de la barra, la matriz de rigidez en ejes locales, la matriz de rotación, la matriz de excentricidad y la matriz de rigidez en ejes globales. El tamaño de dichas matrices es pequeño, lo cual permite que el alojamiento local en la pila que OpenMP lleva a cabo sobre dichas variables, al ser de ámbito privado, sea adecuado. Por el contrario, las variables que almacenan el número de barras asignadas a una partición o el vector con los identificadores de las mismas serán compartidas, aunque sólo a nivel de lectura.

A la hora de llevar a cabo el proceso de ensamblaje, habrá que tener un especial cuidado con la variable K, que almacena la matriz de rigidez global de la estructura, ya que se tratará de una variable compartida a la que los diferentes hilos aportarán las matrices de rigidez de los elementos estructurales.

Otro aspecto a determinar y que también puede tener un fuerte impacto en las prestaciones es el de distribuir las iteraciones del bucle a los hilos, lo cual viene dado por la cláusula schedule. En nuestro caso, hemos elegido el tipo static, que es el que se usa por defecto, por varios motivos. En este tipo de distribución, las iteraciones se dividen por grupos de tamaño determinado formados por un rango continuo de iteraciones, cada uno de los cuales se asigna estáticamente a los hilos, siguiendo un esquema cíclico o de *round-robin*. Claramente, el último grupo asignado puede tener un número de iteraciones inferior al resto. Si no se especifica el tamaño, a cada hilo se le asignará un rango de iteraciones aproximadamente idéntico. Frente a otros tipos de distribuciones de iteraciones a hilos, como puede ser dynamic, en el cual las iteraciones se asignan por grupos a los hilos a medida que han acabado con un grupo de iteraciones previo, la distribución static presenta una menor sobrecarga, siendo muy apropiada cuando todas las iteraciones presentan una carga computacional similar, como claramente es nuestro caso.

Sin lugar a dudas, la parte más delicada de la generación de la matriz de rigidez viene dada por el proceso de ensamblaje, en lo que se refiere a gestionar eficientemente y a nivel de memoria la aportación de la matriz de rigidez de cada elemento de la estructura a sus posiciones correspondientes en la matriz de rigidez global, a lo cual unimos la problemática adicional, al trabajar con OpenMP, de que

diferentes hilos intenten leer y escribir simultáneamente en posiciones idénticas de la matriz. Más aún, es fundamental considerar que las diferentes tareas MPI llevarán a cabo el ensamblaje de los elementos estructurales de su partición sobre las filas correspondientes de la matriz de rigidez K, las cuales unas veces les corresponderán de acuerdo al reparto por bloques de filas consecutivas y otras veces no, dando lugar a comunicaciones entre las tareas.

La función de PETSc que lleva a cabo la suma de los elementos de la matriz de rigidez de un elemento estructural a los elementos de la matriz de rigidez global se denomina MatSetValues. Entre otros parámetros, la función necesita que le pasemos los valores de la matriz y sus índices de fila y columna globales, indicando si queremos asignar directamente un valor o sumarlo a los ya existentes, mediante las opciones INSERT_VALUES o ADD_VALUES, respectivamente. Podemos invocar a la función pasándole o bien elementos individuales, o bien todos los que pertenecen a una misma fila o bien los que forman parte de un bloque de filas.

De este modo, en el caso de un particionado por corte de nodos, todos los elementos de la matriz de rigidez de cada elemento estructural asignados a una tarea pasarán a formar parte de la matriz de rigidez global invocando a la función MatSetValues, a excepción de aquellas filas y columnas no consideradas si trabajamos con la matriz condensada. Sin embargo, las filas de la matriz del elemento correspondientes a sus nodos locales a la partición se ensamblarán dentro del bloque de filas de la matriz global correspondientes a la tarea, no siendo así en el caso de las filas de los nodos adyacentes, las cuales deberán ser enviadas a las particiones vecinas que posean a dichos nodos como locales. Si nos fijamos en la figura 7.8, la tarea 0 enviará a la tarea 1 las filas de los nodos 13 y 16 correspondientes a la matriz de rigidez de los elementos 7 y 15. A la vez, esta tarea deberá recibir de la tarea 1 las filas del nodo 5 que forman parte de la matriz de rigidez del elemento 8.

La situación es muy distinta si aplicamos la técnica del particionado por corte de elementos. En este caso, cada tarea sólo ensamblará las filas de la matriz de rigidez de un elemento correspondientes a los nodos locales a su partición. Si nos fijamos por ejemplo en el elemento 7 de la figura 7.9, la tarea 0 generará la matriz de rigidez de dicho elemento y ensamblará, por medio de la función MatSetValues,

las filas de los nodos 4 y 6, dejando sin ensamblar las filas del nodo 13. Por su parte, la tarea 1 generará también la matriz de rigidez del mismo elemento, al ser compartido, pero ensamblará las filas del nodo 13 y no las de los nodos 4 y 6. Es evidente, por tanto, que este tipo de particionado conlleva la ventaja de que no existen comunicaciones entre las tareas a la hora de ensamblar las matrices de rigidez de sus elementos y al generar, en consecuencia, la matriz de rigidez global de la estructura.

Como hemos citado anteriormente, al trabajar con OpenMP, la matriz de rigidez global K será una variable compartida entre los distintos hilos de una misma tarea, a la cual incorporarán valores que hay que sumar a los ya existentes. Eso significa que cabrá la posibilidad de que dos o más hilos intenten modificar, al mismo tiempo, valores situados en posiciones idénticas de la matriz. Lógicamente, el acceso a dicha variable debe controlarse, garantizando que sólo un hilo de ejecución pueda modificar simultáneamente la matriz. Para ello, emplearemos el constructor #pragma omp critical que OpenMP nos ofrece, a fin de definir una región crítica compuesta únicamente por la invocación a la función MatSetValues:

```
#pragma omp critical
{
  MatSetValues(K,nFilas,indices_filas,nColumnas,indices_columnas,valores,
               ADD_VALUES);
}
```

De este modo, cuando un hilo de ejecución alcance este constructor, se esperará hasta que ningún otro hilo esté modificando los elementos de la matriz. Sin embargo, el inconveniente que presenta toda sección crítica es que los hilos de ejecución permanecerán detenidos durante un tiempo hasta que tengan acceso a dicha sección, lo cual repercutirá en las prestaciones del algoritmo paralelo. Para tratar de aliviar en la medida de lo posible dicho inconveniente, las invocaciones a la función MatSetValues deberán ser mínimas. Así, en nuestro caso, sólo llamaremos una vez a dicha función por cada matriz de rigidez de un elemento a ensamblar, proporcionándole conjuntamente todos los valores de la matriz que tengan que incorporarse a dicha matriz global.

Como paso final, debemos invocar a las funciones MatAssemblyBegin y MatAssemblyEnd que son las que realmente se encargan de llevar a cabo las comunica-

ciones entre las tareas, de forma totalmente transparente para el usuario, en caso de que los datos aportados por una tarea no estén situados en el bloque de filas asignado a la misma.

Si no vamos a incorporar futuros valores a la matriz, tendremos que llamar a las funciones con la opción MAT_FINAL_ASSEMBLY. Si por el contrario todavía tenemos pendiente modificar el valor de algún elemento, como ocurre en nuestro caso al imponer las condiciones de contorno, llamaremos a las funciones con la opción MAT_FLUSH_ASSEMBLY. Hay que tener en cuenta además que PETSc no permite mezclar fases de inserción y de adición de valores a una matriz, motivo por el cual hay que invocar entre ellas a las funciones MatAssemblyBegin y MatAssemblyEnd con el parámetro MAT_FLUSH_ASSEMBLY.

7.2.4. Generación del vector de cargas

Cada tarea tendrá la responsabilidad de incorporar, al vector de cargas de la estructura, el aporte de cada una de las acciones aplicadas bien sea directamente sobre los nodos locales a su partición o sobre los elementos estructurales asignados a la misma, distribuyendo dichas cargas en los nodos que forman los elementos, de acuerdo a lo descrito en la sección 3.3. Puesto que serán múltiples las distintas hipótesis básicas de carga aplicadas sobre la estructura (peso propio, concargas, cargas de nieve, cambios de temperatura, etc.), el tipo de datos encargado de almacenarlas equivaldrá en realidad a una matriz densa, compuesta por tantas columnas como número de hipótesis de carga diferente tengamos, almacenada en memoria también por columnas. Por otro lado, el número total de filas de dicha matriz de cargas coincidirá con el de la matriz de rigidez global de la estructura, aplicando el mismo tipo de distribución de bloques de filas consecutivas a las tareas de MPI descrito en la sección anterior. A modo de ejemplo la figura 7.2 recoge la distribución de la matriz F de cargas, compuesta por 3 hipótesis básicas, entre 4 tareas.

Bajo el tipo de datos Vec, PETSc nos ofrece la posibilidad de trabajar con dos tipos de vectores, uno secuencial, en el que todos sus componentes pertenecen a una misma tarea, o paralelo, en el cual sus elementos están distribuidos entre las distintas tareas. Respectivamente, dichos vectores se podrán crear por medio

de las funciones VecCreateSeq o VecCreateMPI, indicando el tamaño global de los mismos. En caso de que el vector sea paralelo, podemos determinar el número de elementos consecutivos que almacenará cada tarea, o dejar que PETSc lo decida por nosotros. Tanto en un caso como en otro, PETSc reservará el espacio en memoria necesitado para guardar los elementos del vector.

Sin embargo, cabe también la posibilidad de que sea el usuario el que realice previamente la reserva de espacio en memoria y le indique a PETSc que dicho espacio quede a disposición de un vector concreto. En este caso, la librería nos ofrece las funciones VecCreateSeqWithArray o VecCreateMPIWithArray.

Aunque PETSc dispone de funciones como VecSetValues, VecGetValues o VecGetArray para modificar los valores de un vector o acceder al valor de los mismos, bien es verdad que dispondremos de una mayor flexibilidad a la hora de trabajar con los elementos de un vector si somos nosotros quienes reservamos la memoria, lo cual además no nos resta de ninguna de las funcionalidades que PETSc nos ofrece para trabajar con este tipo de datos. A modo de ejemplo donde se muestra esta ventaja, podemos citar el hecho de obtener un vector a partir de otros por medio de operaciones matriciales realizadas con BLAS. Si una vez obtenido este vector tuviéramos que invocar a la función VecSetValues, para asignar los valores al vector con el que vaya a trabajar PETSc, tendríamos una copia de datos innecesaria, además de un consumo de memoria adicional que a continuación deberíamos reajustar.

Para almacenar en paralelo una matriz como la encargada de guardar las cargas aplicadas en paralelo, disponer de la flexibilidad mencionada y poder aprovechar la funcionalidad que PETSc nos ofrece a nivel del tipo de datos Vec, hemos creado un nuevo tipo de datos llamado Vdmc (vector distribuido con múltiples columnas) compuesto, entre otros por los siguientes campos:

```
struct Vdmc {
  long nFilasLocales;  /* Número de filas locales a la tarea */
  long nFilasGlobales; /* Número total de filas */
  long nColumnas;      /* Número de columnas */
  double *valores;     /* Vector de tamaño nFilasLocales*nColumnas */
  Vec *vec_petsc;      /* Vector de longitud nColumnas */
};
```

A partir por tanto de una variable del tipo Vdmc, dispondremos de un vector de vectores de tipo Vec secuenciales o paralelos. Reservaremos, como decíamos, un vector llamado valores, de longitud igual al número de filas locales nFilasLocales multiplicado por el número de columnas nColumnas, donde almacenaremos los elementos de la matriz asignados a cada tarea, repartidos por columnas, además de un vector de punteros llamado vec_petsc a vectores de tipo Vec. Por medio de invocaciones a las funciones VecCreateSeqWithArray y VecCreateMPIWithArray (tantas como número de columnas tengamos) determinaremos que el espacio asignado a cada vector columna se corresponderá con posiciones concretas ya reservadas del vector valores. De este modo, podemos tratar cada columna de forma individual, por medio de las funciones de PETSc, o podemos tratar la matriz en su conjunto, realizando por ejemplo alguna operación matricial mediante las rutinas de BLAS.

El algoritmo 5 nos muestra la rutina que distribuye las cargas estáticas aplicadas en los triángulos de una partición sobre sus nodos vértices, aportando dichos valores a la matriz F global de cargas. El algoritmo es totalmente válido para otro tipo de elementos, como barras, hexaedros o tetraedros.

Algoritmo 5 Aportación de las cargas aplicadas sobre los triángulos de una tarea a la matriz F de cargas global

Entrada: F (matriz de cargas global), *triangulos_particion* (vector con el identificador de los triángulos asignados a la tarea), *nTriangulosParticion* (número total de triángulos asignados a la tarea), *nHipotesisBasicas* (número de hipótesis de carga aplicadas)
Salida: F (matriz de cargas global)

```
1: #pragma omp parallel for private (t, h, nelems, indices_filas, valores)
2: para triangulo = 1 hasta nTriangulosParticion hacer
3:    t=triangulos_particion[triangulo]
4:    para h = 1 hasta nHipotesisBasicas hacer
5:       Calcular el valor de las cargas aplicadas en los nodos del triángulo t
6:       Completar los vectores con los índices de las filas y los valores de la carga,
         de longitud nelems, de aquellos nodos a incorporar a la matriz F
7:       #pragma omp critical /* Ensamblaje de las cargas en los nodos */
8:          VecSetValues(F->vec_petsc[h], nelems, indices_filas, valores,
9:                       ADD_VALUES);
10:   fin para
11: fin para
```

La tarea recorre todos los triángulos correspondientes a su partición, distri-

buyendo el cálculo de la carga aplicada sobre los nodos de cada triángulo y su posterior ensamblaje entre los diferentes hilos de ejecución, por medio del constructor #pragma omp parallel for. En un particionado por corte de nodos, cada triángulo aportará a la matriz F los valores de las cargas aplicados sobre sus 3 nodos. Al igual que ocurría en el caso de la matriz de rigidez, los valores de las cargas aplicadas sobre los nodos locales se ensamblarán en filas de la matriz F pertenecientes a la tarea, mientras que los valores de las cargas aplicadas sobre los nodos adyacentes se corresponderán con filas de la matriz F asignadas a tareas vecinas, a las que lógicamente se las deberemos enviar. En un particionado en cambio por corte de elementos, cada triángulo sólo aportará a la matriz F los valores de las cargas de sus nodos locales, dejando sin ensamblar las cargas de sus nodos adyacentes, lo cual supone que este tipo de particionado no conlleve comunicaciones a la hora de generar la matriz de cargas.

Como podemos ver en el algoritmo 5, la función que sumará los valores de las cargas en los nodos a las posiciones correspondientes de los diferentes vectores de hipótesis es la función VecSetValues, la cual trabaja con índices de filas globales. Puesto que diferentes hilos de ejecución podrían modificar el valor de un mismo elemento, estamos obligados a convertir la invocación a la función en una sección crítica.

Cada tarea deberá también aportar a la matriz F los valores de las cargas asignadas directamente sobre sus nodos locales, independientemente del tipo de particionado. De manera similar al procedimiento que acabamos de ver, cada hilo de ejecución ensamblará las cargas correspondientes a un grupo de nodos por medio de la función VecSetValues, la cual se convertirá de nuevo en una sección crítica que implementamos por medio del constructor #pragma omp critical.

Cuando hayamos acabado de incorporar todos los elementos a los vectores de carga, deberemos invocar a las funciones VecAssemblyBegin y VecAssemblyEnd, las cuales llevan a cabo las comunicaciones entre las tareas de todos aquellos valores no locales previamente insertados.

7.2.5. Imposición de las condiciones de contorno

La imposición de las condiciones de contorno puede suponer la modificación de los valores de la diagonal de la matriz de rigidez K y del vector de fuerzas F, de acuerdo a lo que vimos en la sección 3.4.

En el caso de considerar a todas las ecuaciones del problema, e independientemente del tipo de particionado, cada tarea impondrá las condiciones de contorno (mediante la función MatSetValues, bien sea con las opciones ADD_VALUES o INSERT_VALUES) en los elementos de la diagonal de la matriz K correspondientes a sus nodos locales coaccionados. Es el caso, por ejemplo, de los nodos empotrados 1 y 2 asignados a la tarea 0 o los nodos 3 y 16 asignados a la tarea 1 en la figura 7.8. Además, la tarea invocará a la función VecSetValues, con la opción INSERT_VALUES, para modificar los elementos de la matriz F cuando alguno de sus nodos locales presente un movimiento impuesto.

En el caso de trabajar con el problema condensado, donde no habremos incorporado todas aquellas ecuaciones en la que sabemos que el valor de los desplazamientos es nulo, cada tarea deberá imponer las condiciones de contorno en sus nodos locales que sean apoyos elásticos o presenten un movimiento impuesto.

Una vez finalizada la etapa de imponer las condiciones de contorno, y puesto que habremos modificado los valores de las matrices K o F, debemos invocar a las funciones MatAssemblyBegin y MatAssemblyEnd, con el argumento MAT_FINAL_ASSEMBLY. Si además hemos modificado la matriz F debido a la presencia de algún movimiento impuesto, tendremos que invocar a las funciones VecAssemblyBegin y VecAssemblyEnd. A partir de este momento ya podremos operar con dichas matrices a fin de llevar a cabo cualquier operación de algebra lineal como es, en nuestro caso particular, la resolución del sistema de ecuaciones que obtiene los desplazamientos en los nudos de la estructura.

Debido al nulo coste computacional que conlleva la imposición de las citadas condiciones de contorno, las tareas no crearán ningún hilo de ejecución, encargándose únicamente el hilo principal de llevar a cabo el trabajo mencionado.

7.2.6. Cálculo de los desplazamientos en los nudos

El cálculo de los desplazamientos en los nudos de la estructura supone resolver el sistema de ecuaciones, con múltiples partes derecha, formado por la matriz de rigidez K y la matriz de fuerzas F, a fin de obtener la matriz de desplazamientos D, tal y como recogimos en la sección 3.5.

La distribución de los elementos de la matriz D coincidirá plenamente con la de la matriz F, como se observa en la figura 7.2. Eso supone que el tipo de datos que vamos a emplear para almacenar la matriz D es el denominado Vdmc, el cual hemos descrito en la sección anterior para guardar los valores de la matriz F. No obstante, dicho tipo de datos presenta algunos parámetros adicionales necesarios para guardar la matriz D frente a los que hemos visto en el caso de la matriz F.

Es bien sabido que los desplazamientos en los nodos que vayamos a calcular en esta etapa representan los datos de entrada de etapas próximas, a partir de los cuales se obtendrán, por ejemplo, los esfuerzos en los extremos de una barra o las tensiones en los nudos un elemento finito. Esto supone, por tanto, que cada elemento de la estructura debe tener conocimiento de los desplazamientos de todos sus nodos vértices. Dicho de otro modo, cada proceso tendrá que conocer los desplazamientos en sus nodos locales y en sus nudos adyacentes. Si bien es cierto que cada tarea obtendrá los desplazamientos en sus nodos locales tras resolver el sistema de ecuaciones, también es cierto que la necesidad de conocer los desplazamientos en sus nodos adyacentes implica que tendrá recibirlos de parte de aquellas tareas vecinas que posean a dichos nodos como locales.

Para afrontar este problema, PETSc nos permite trabajar con los que denomina nodos *ghost* (o fantasmas), los cuales se corresponden con nuestros nodos adyacentes de cada partición. Debemos por tanto enriquecer el tipo de datos Vdmc, incluyendo el número de filas correspondientes a los nodos ghost, o adyacentes, a la partición, reservando espacio para las mismas en el vector valores donde almacenamos los desplazamientos. De hecho, dichas filas de los nodos adyacentes estarán siempre situadas en la parte final de cada columna. La figura 7.11 nos muestra la distribución de la matriz de desplazamientos entre cuatro tareas. Como puede observarse, cada tarea P_i reserva m_i filas en las que almacenará los desplazamientos de sus nodos locales y n_i filas en las que guardará los desplaza-

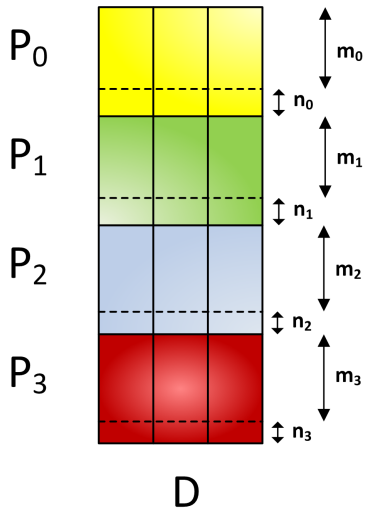


Figura 7.11: Distribución de la matriz de desplazamientos, distinguiendo entre las filas locales m_i y las filas ghost n_i de cada tarea P_i.

mientos de sus nodos adyacentes. La nueva estructura de datos es por tanto la siguiente. Obsérvese que, en el caso de la matriz de fuerzas F, el número de filas reservadas para los nodos adyacentes de cada tarea será igual a 0:

```
struct Vdmc {
   long nFilasLocales;  /* Número de filas locales (no incluye filas ghost) */
   long nFilasGlobales; /* Número total de filas (no incluye filas ghost) */
   long nFilasGhost;    /* Número de filas ghost locales a la tarea */
   long nColumnas;      /* Número de columnas */
   double *valores;     /* Vector de longitud (nFilasLocales+nFilasGhost)*
                           *nColumnas */
   Vec *vec_petsc;      /* Vector de longitud nColumnas */
};
```

Una característica de la parte ghost de los vectores es que no participa en las operaciones de álgebra lineal llevadas a cabo por medio de las funciones que nos proporciona PETSc.

Por otro lado, PETSc ofrece numerosas funciones para gestionar los vectores con filas ghost. De entre todas ellas, y para comenzar, emplearemos la función

VecCreateGhostWithArray para crear un vector paralelo con filas ghost, tras haber reservado previamente el espacio en memoria por nuestra parte. Si preferimos que sea PETSc quien reserve la memoria necesaria, entonces deberíamos utilizar la función VecCreateGhost. Para cada vector a crear, dichas funciones necesitan que les indiquemos el número de filas locales a la tarea, el número total de filas (suma del número de filas locales de todas las tareas) y el número de filas ghost de esta tarea, además de los índices globales de dichas filas ghost. Los vectores secuenciales los seguiremos creando con la función VecCreateSeqWithArray, entendiendo que estos vectores no incluirán filas ghost.

Una vez que hayamos resuelto el sistema de ecuaciones y cada tarea haya obtenido los desplazamientos de sus nodos locales, debe comenzar el proceso de envío y recepción de datos entre las tareas. Esto supone que cada tarea debe recibir, de las tareas que gestionan las particiones vecinas, los desplazamientos de sus nodos catalogados como adyacentes. Además, cada tarea debe enviar los desplazamientos de sus nodos locales que sean adyacentes en otras particiones vecinas. Este proceso de comunicación entre tareas tiene lugar independientemente del tipo de particionado que tengamos. Así, en el caso del particionado de corte de nodos de la figura 7.8, la tarea 0 debe recibir los desplazamientos de los nodos 11 y 16 por parte de la tarea 1. Adicionalmente, la tarea 0 deberá enviar los desplazamientos del nodo 6 a la tarea 1. En el caso del particionado por corte de elementos (ver figura 7.9), la tarea 0 enviará los desplazamientos de los nodos 6, 10 y 15 a la tarea 1, de la cual recibirá los movimientos de los nodos 7, 11 y 16.

De manera sencilla, dicho proceso de comunicación entre las tareas se llevará a cabo invocando a las funciones VecGhostUpdateBegin y VecGhostUpdateEnd, proporcionándole los diferentes vectores columna de la matriz D acompañados de los parámetros INSERT_VALUES y SCATTER_FORWARD.

Para resolver el sistema de ecuaciones y determinar así los desplazamientos en los nodos, PETSc dispone del tipo de dato llamado KSP, el cual nos proporciona un acceso uniforme y eficiente a una amplia variedad de métodos directos e iterativos, secuenciales o paralelos, presentes en el propio PETSc o en otras librerías externas de las que actúa como interfaz. Una de las ventajas que ofrece dicho tipo de dato, u objeto, es que la secuencia de instrucciones que debemos escribir para resolver el sistema es independiente del tipo de método y de librería numérica

empleada. Es más, será precisamente el usuario quien, en tiempo de ejecución, determine la librería numérica a emplear, el método de resolución escogido y los diferentes parámetros que acompañan a éste. Dicha secuencia de instrucciones a la que hacíamos alusión para resolver el sistema $KD = F$ es la siguiente:

```
KSP solver;                    /* Declaramos la variable de tipo KSP */
KSPCreate(PETSC_COMM_WORLD,&solver); /* Creamos el contexto */
KSPSetOperators(solver,K,K); /* Indicamos la matriz de coeficientes y la matriz
                                asociada con un posible precondicionador */
KSPSetFromOptions(solver);  /* Leemos los parámetros de la línea de comandos */
KSPSetUp(solver);              /* Creamos las estructuras de datos internas */
/* Resolvemos un sistema para cada columna de la matriz F */
for (h=0;h<F.nColumnas;h++)
  KSPSolve(solver,F.vec_petsc[h],D.vec_petsc[h]);
KSPDestroy(&solver);           /* Destruimos el solver y liberamos memoria */
```

En nuestro caso, hemos empleado las librerías MUMPS, PaStiX y PARDISO para dar solución a los citados sistemas de ecuaciones mediante métodos directos, además de la propia librería PETSc para resolver los mismos sistemas mediante métodos iterativos, acompañados de algún tipo de precondicionador presente en el propio PETSc o en la librería *hypre*. MUMPS, por ejemplo, posee una paralelización basada en MPI y en OpenMP, además de invocar a funciones de BLAS, lo cual nos resulta ideal debido al modelo de programación paralela multinivel que hemos escogido en la implementación del Simulador Estructural. Sin embargo, la versión 3.6.3 de PETSc, que responde a la última versión publicada de la librería, está únicamente basada en MPI, debiendo confiar en las invocaciones que lleve a cabo sobre funciones de BLAS y en el hecho de disponer de una librerías BLAS multihilo para obtener algún tipo de paralelismo a nivel de memoria compartida.

Aprovechando la flexibilidad que PETSc no ofrece, invocaríamos a nuestro simulador paralelo acompañado de los datos necesarios para resolver el sistema de ecuaciones. Para ello, las opciones -ksp_type y -pc_type nos permiten determinar, en tiempo de ejecución, el método iterativo y el precondicionador encargado de resolver el sistema de ecuaciones. Si en cambio deseamos resolver nuestro sistema mediante el método directo que nos ofrece la librería MUMPS, algunos de los pa-

rámetros con los que invocaríamos al Simulador Estructural serían los siguientes:

```
-ksp_type preonly -pc_type cholesky -pc_factor_mat_solver_package mumps
```

7.2.7. Cálculo de solicitaciones en extremos de barras

A nivel de la implementación paralela, cada tarea obtendrá los esfuerzos en los extremos de todas aquellas barras que le han sido asignadas, repartiendo el cálculo de todas ellas entre los hilos de ejecución creados gracias al constructor `#pragma omp parallel for` que paralelizará el bucle encargado de recorrer todas las barras de la partición.

Para calcular los esfuerzos en sus extremos, cada barra necesitará los desplazamientos en los mismos, los cuales ya serán valores conocidos de acuerdo a las comunicaciones que tienen lugar una vez resuelto del sistema de ecuaciones, según hemos descrito en la sección anterior. Ello supone que, independientemente del tipo de particionado aplicado, no habrá ningún tipo adicional de comunicaciones entre las tareas para obtener dichas solicitaciones en extremos.

No obstante, puede ocurrir que diferentes tareas calculen, de manera repetitiva, los esfuerzos de una misma barra compartida entre ambas, en el caso del particionado con corte de elementos. Aunque podríamos haber implementado un algoritmo que decidiera, llegados a este punto, cuáles de dichas barras replicadas entre varias particiones se calculan por una tarea o por otra, eliminando de este modo el cálculo repetitivo, hemos decidido dejarlo así como base de la implementación futura de un cálculo estático no lineal por incrementos de carga, en el cual los esfuerzos acumulados en los extremos de una barra en una iteración de un método como Newton-Raphson son necesarios para obtener la matriz de rigidez geométrica de la barra en la iteración siguiente.

A nivel global, las solicitaciones de las diferentes barras se guardarán en una variable de tipo `Vdmc`, con un número global de filas igual a la suma del número de barras asignadas a cada tarea multiplicado por 12 y un número de columnas igual al número de hipótesis básicas de carga. La matriz estará repartida por bloques de filas entre las distintas tareas, donde el número de filas de cada una de ellas dependerá del número de barras asignadas a su partición. El número de

filas ghost por tarea sera igual a 0. Para sumar el número de barras asignadas a las distintas tareas, teniendo en cuenta que podrá haber barras replicadas, se ha empleado la función MPI_Allreduce.

A su vez, el cálculo de los esfuerzos para cada barra, de acuerdo a la expresión (3.169), se realiza invocando a la función dgemm de BLAS, operando con los desplazamientos, los esfuerzos de empotramiento y las solicitaciones en los extremos de la barra para las diferentes hipótesis de manera conjunta, como si de matrices densas se tratara.

7.2.8. Cálculo de reacciones en apoyos

Para almacenar las reacciones en aquellos nudos de la estructura que sean un apoyo, para las diferentes hipótesis de carga, emplearemos una variable de tipo Vdmc distribuida entre todas las tareas. Dicha variable tendrá un número global de filas equivalente al producto del número de nudos de la estructura que sean apoyos multiplicado por 6. El número de columnas vendrá dado por el número de hipótesis básicas aplicadas y cada tarea trabajará con un grupo de filas consecutivas correspondientes a multiplicar por 6 la cantidad de nodos locales a su partición que sean apoyos. El número de filas ghost será igual a 0.

La reacción en un apoyo viene dada por las fuerzas que le transmiten los diferentes elementos que a él confluyen, a lo que habrá que restar el valor de la fuerza aplicada directamente sobre él mismo. De este modo, cada tarea deberá calcular la fuerza que transmiten los elementos de su partición a sus nodos vértices que sean apoyos, independientemente de que el nodo correspondiente al apoyo sea local o adyacente a la partición, en el caso de un particionado por corte de nodos. Sin embargo, en el caso de un particionado por corte de elementos, la tarea sólo debe considerar a aquellos elementos que incluyan entre sus vértices a apoyos que sean nodos locales a la partición. Dicho conjunto de elementos con los que trabajaremos en un tipo de particionado u otro se repartirá entre los diferentes hilos de ejecución, creados por medio del constructor #pragma omp parallel for.

La fuerza que transmite cada elemento a un apoyo viene expresada como un vector de 6 componentes, para cada hipótesis de carga, el cual habrá que

incorporar a la matriz que almacena las reacciones de todos los apoyos y que calcularemos por medio de invocaciones a la función dgemm de BLAS. Dicha incorporación la realizaremos gracias a la función VecSetValues, acompañada por el parámetro ADD_VALUES, la cual deberá formar parte de una sección crítica, implementada mediante el constructor #pragma omp critical, puesto que diferentes hilos que trabajan con distintos elementos podrán tener un apoyo en común y llevar a cabo su aportación simultáneamente a las mismas posiciones en memoria de la matriz que almacena las reacciones. Finalizada la incorporación de todos los elementos de una partición a sus apoyos habrá que invocar a las funciones VecAssemblyBegin y VecAssemblyEnd, bajo las cuales se llevarán a cabo las posibles comunicaciones.

En el caso de la partición por corte de nodos, toda tarea que posea un elemento con algún apoyo adyacente deberá enviar la fuerza que le transmite a dicho apoyo a aquella tarea que lo posea como nodo local. Así por ejemplo, si nos fijamos en la figura 7.8, vemos que la partición 0 tiene como apoyos locales a los nodos 1 y 2, mientras que el apoyo del nudo 16 será adyacente a esta partición pero local a la partición 1. Esto implica que la tarea 0 calculará la fuerza que las barras 1 y 2 transmiten a los apoyos con idéntica numeración, lo cual no da lugar a comunicaciones al tratarse de nodos locales, además de calcular la fuerza que el triángulo 15 aporta al nudo 16, lo que conllevará el envío de dicha aportación a la tarea 1, quien la sumará a las fuerzas transmitidas por los triángulos 16 y 17 que dicha tarea calcula.

La situación es muy diferente en el caso de un particionado por corte de elementos, ya que en este caso cada tarea sólo considera a los elementos cuyos nodos apoyos sean en realidad locales a la partición. Eso implica, por tanto, que no habrá comunicaciones bajo este tipo de particionado, ya que todos los elementos que confluyen a un apoyo local estarán asignados a una misma partición. Es el caso del ejemplo recogido en la figura 7.9, donde vemos que la tarea 0 solamente obtendrá la fuerza que la barra 1 aporta al nodo 1, sin preocuparse por la barra 2 y el triángulo 15 puesto que los apoyos a los que estos elementos confluyen no son locales a la partición. Paralelamente, la partición 1 calculará la aportación de las fuerzas de las barras 2 y 3 y los triángulos 15, 16 y 17 a los nodos 2, 3 y 16, sin que ello conlleve comunicación alguna.

Finalmente, cada tarea recorrerá todos sus apoyos locales restando, si es el caso, la carga en ejes globales aplicada directamente sobre cada uno de ellos. La labor a realizar sobre los diferentes apoyos quedará repartida entre los distintos hilos de ejecución creados con dicho fin, ensamblando los valores a restar por medio de la función VecSetValues que ahora no es necesario que forme parte de una sección crítica.

7.2.9. Cálculo de esfuerzos y deformaciones en puntos intermedios de las barras

A la hora de calcular los esfuerzos y las deformaciones en los puntos intermedios de las barras asignadas a cada partición, nos encontramos que, si hemos particionado la estructura mediante la técnica de corte de elementos, existirán barras compartidas entre diferentes particiones. En realidad, carece de sentido que diferentes tareas calculen de manera repetitiva los esfuerzos y deformaciones de un mismo conjunto de barras.

Para evitarlo, hemos decidido que toda barra que forme parte de más de una partición se asignará, a fin de calcular sus esfuerzos y deformaciones, a aquella que posea el menor índice. De este modo y a efectos de cálculo de este tipo de resultados, la barra 2 del ejemplo recogido en la figura 7.9 pasará a formar parte única y exclusivamente de la partición 0.

Previamente al cálculo de los esfuerzos y las deformaciones para cada barra, debemos reservar en memoria la variable que almacenará dicha información. Se tratará de nuevo de dos variables de tipo Vdmc, una para guardar los esfuerzos y otra las deformaciones, que estarán formadas por un número de filas igual al número de puntos intermedios en los que se dividen todas las barras de la estructura multiplicado por 6. El número de columnas vendrá dado por el número de hipótesis básicas de cálculo. Cada tarea dispondrá de un conjunto de filas consecutivas en las que almacenará los esfuerzos o las deformaciones en los puntos intermedios de las barras asignadas a la partición que gestiona. Además, las diferentes tareas podrán obtener el número global de filas de la matriz sumando el número de filas asignado a cada una de ellas, empleando para ello la función MPI_Allreduce.

En definitiva, este tipo de resultado no presenta ninguna dificultad para calcularlo en paralelo entre las diferentes tareas MPI, ya que no habrá ninguna comunicación entre ellas. A nivel de una misma tarea, crearemos diferentes hilos de ejecución que se repartirán el cálculo de los esfuerzos, en primer lugar, y de las deformaciones, en segundo lugar, de las barras asignadas a la partición. Como los diferentes hilos no accederán a posiciones comunes de los resultados, no existirá tampoco la necesidad de definir ninguna región crítica.

7.2.10. Cálculo de deformaciones, esfuerzos y tensiones en elementos finitos

Cuando nos referimos a las deformaciones unitarias, los esfuerzos y las tensiones en los elementos finitos, conviene recordar que dichas magnitudes las proporcionaremos a modo de resultados en nudos concretos de la estructura, obtenidos como valor promedio de la aportación de dichas magnitudes por parte de determinados elementos finitos que tienen al nudo como vértice.

De acuerdo a lo descrito en el apartado 3.10, las deformaciones unitarias y las tensiones se obtendrán en cualquier nudo al que confluya algún elemento finito, bien sea 2D o 3D, como valor medio de la aportación de las deformaciones y las tensiones obtenidas en dicho nudo por parte de esos elementos. Sin embargo, sólo proporcionaremos esfuerzos en un nudo de la estructura si es vértice de algún elemento finito 2D, a partir del promedio de los valores de esfuerzos obtenidos por parte de dichos elementos bidimensionales en ese nudo.

Cada tarea será por tanto responsable de obtener los valores promediados de las deformaciones, las tensiones y los esfuerzos en los nudos locales a su partición que pertenezcan a algún elemento finito. Esto supone que, en un particionado por corte de nodos, la tarea calculará las deformaciones, las tensiones y los esfuerzos (si procede) en los nodos de los elementos finitos asignados a su partición, promediando dichos resultados en sus nodos locales, lo cual no conllevará comunicaciones, y en sus nodos adyacentes, lo cual evidentemente supondrá enviar dicha información a aquella tarea que posea a dicho nodo como local.

Pongamos un ejemplo de acuerdo a la figura 7.8. Tras calcular, por parte de

la tarea 0, las deformaciones unitarias, la tensiones y los esfuerzos en los tres nodos del elemento 7, los valores obtenidos en los nodos 6 y 11 se promediarán, por parte de la citada tarea, con la aportación de los triángulos que confluyen a dichos nodos, lo cual conllevará la necesidad de recibir la información calculada en el nudo 6 debida al triángulo 8 por parte de la tarea 1. Consecuentemente, la tarea 0 deberá enviar los valores obtenidos en el nodo 11 a la tarea 1, a fin de ser promediados por esta última.

En cambio, bajo un particionado por corte de elementos, cada tarea sólo calculará y promediará los resultados de deformaciones, tensiones y esfuerzos en los nodos locales a la partición de cada uno de sus elementos finitos, lo cual supone que no habrá comunicaciones. Esto significa que, en el caso de la figura 7.9, la tarea 0 sólo calculará los valores mencionados en el nodo 6 del elemento 8, pero no así en los nodos 7 y 11 de dicho elemento, labor ésta que realizará la tarea 1. Hay que tener en cuenta que, en un particionado por corte de elementos, todos los elementos finitos que rodean a un nodo local a una partición forman parte de la misma partición, motivo por el cual no es necesaria la aportación de ningún otro elemento que pudiera pertenecer a una partición distinta. Dicha circustancia no es cierta, como bien sabemos, en el caso del particionado por corte de nodos, y es por ello la necesidad de las comunicaciones.

En el caso de las deformaciones unitarias y las tensiones, reservaremos en memoria sendas matrices distribuidas, de tipo Vdmc, del siguiente tamaño. Las matrices tendrán un número global de filas equivalente al producto del número de nodos a los que confluye algún elemento finito multiplicado por 6. A nivel local, cada tarea trabajará con un número de filas consecutivas acorde al número de nodos locales a su partición, que son vértices de algún elemento finito, multiplicado por 6. El número de columnas, como siempre, vendrá dado por el número de hipótesis de carga y no trabaremos con filas ghost.

En el caso de los esfuerzos en los nodos de los elementos finitos 2D, obtendremos cortantes y momentos de placa, almacenados en dos matrices distribuidas también de tipo Vdmc, con un número global de filas correspondiente al producto del número de nodos multiplicado por 3 o por 6, respectivamente. En cuanto al número de filas asignadas a cada tarea, de manera consecutiva, dicho valor se obtendrá como el producto de la cantidad de nudos locales a la partición a los que

confluye algún elemento finito en 2D multiplicado por 3 o por 6. Evidentemente tendremos tantas columnas como número de hipótesis de carga y no es necesario reservar espacio para filas ghost.

Para calcular las deformaciones unitarias, las tensiones y los esfuerzos en los nodos de cada elemento finito emplearemos principalmente la función dgemm de BLAS, acompañada en algún ocasión, como en el caso de las deformaciones unitarias en triángulos, por la función daxpy.

El algoritmo 6 nos muestra en qué consiste el trabajo a realizar por parte de una tarea para calcular las deformaciones unitarias y las tensiones en sus nodos locales a partir de dichos valores obtenidos en los nodos de sus tetraedros. Tras crear los hilos de ejecución convenientes y repartir las iteraciones entre los mismos, recorremos todos los tetraedros de la partición, calculando las deformaciones unitarias y las tensiones en todos sus nodos (en el caso de un particionado por corte de nodos) o únicamente en aquellos nodos que sean locales a la partición (si se trata de un particionado por corte de elementos). Dichos valores, expresados con respecto a los ejes locales del elemento, se transforman a ejes globales para ser promediados con la aportación del resto de elementos que confluyan a un mismo nodo. La aportación a los vectores de deformaciones unitarias y tensiones se realiza por medio de la función VecSetValues, la cual debe formar parte de una región crítica, ya que diferentes hilos encargados de tetraedros diferentes podrían intentar aportar simultáneamente la deformación o la tensión en uno de los nodos que tengan en común.

Finalizada toda aportación en el cálculo de deformaciones y tensiones por parte de los diferentes elementos finitos, debemos invocar a las funciones VecAssemblyBegin y VecAssemblyEnd, las cuales llevarán a cabo las comunicaciones pertinentes entre las distintas tareas, únicamente en el caso de haber aplicado un particionado por corte de nodos.

El algoritmo implementado para calcular en paralelo los esfuerzos en los nodos de los triángulos asignados a un partición es idéntico al que acabamos de comentar relativo al cálculo de las deformaciones y las tensiones de todos aquellos nodos a los que confluye algún elemento finito.

Algoritmo 6 Aportación de las deformaciones unitarias y las tensiones en los tetraedros de una tarea a las deformaciones unitarias ϵ y las tensiones σ globales en los nodos de la estructura

Entrada: ϵ (matriz de deformaciones unitarias globales), σ (matriz de tensiones globales), *tetraedros_particion* (vector con el identificador de los tetraedros asignados a la tarea), *nTetraedrosParticion* (número total de tetraedros asignados a la tarea), *nHipotesisBasicas* (número de hipótesis de carga aplicadas)

Salida: ϵ (matriz de deformaciones unitarias globales) y σ (matriz de tensiones globales)

```
1: #pragma omp parallel for private (t, h, nelems, indices_filas, valores_deform, valores_tension)
2: para tetraedro = 1 hasta nTetraedrosParticion hacer
3:    t=tetraedros_particion[tetraedro]
4:    Calcular la deformación unitaria en ejes locales en los nodos a ensamblar del tetraedro t
5:    Obtener la tensión en ejes locales en los nodos a ensamblar del tetraedro t
6:    para h = 1 hasta nHipotesisBasicas hacer
7:       Transformar a ejes globales y promediar los valores de la deformación unitaria (valores_deform) y las tensiones (valores_tension) en los nodos a ensamblar
8:       Completar el vector indices_filas, de longitud nelems, con los índices de las filas a incorporar a las matrices ε y σ.
9:       #pragma omp critical  /* Ensamblaje de deformaciones y tensiones */
10:      {
11:         VecSetValues(ε->vec_petsc[h], nelems, indices_filas, valores_deform,
12:                      ADD_VALUES);
13:         VecSetValues(σ->vec_petsc[h], nelems, indices_filas, valores_tension,
14:                      ADD_VALUES);
15:      }
16:   fin para
17: fin para
```

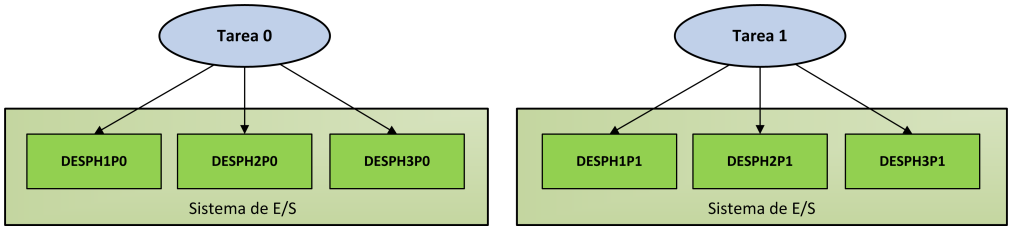


Figura 7.12: Alternativa 1 en la que cada tarea escribe sus resultados de forma individual.

7.2.11. Escritura de los resultados en disco

El tiempo dedicado a la escritura de los resultados de la simulación en ficheros de disco puede llegar a ser una parte importante frente al tiempo total de simulación, especialmente en estructuras de barras donde los esfuerzos y las deformaciones en puntos intermedios ocupan una cantidad de espacio considerable o en un cálculo dinámico a lo largo del tiempo donde hay que escribir resultados periódicamente. En ese sentido resulta conveniente paralelizar también esta etapa final del análisis estructural, al igual que hemos hecho con el resto.

A la hora de llevar a cabo la escritura de los datos de salida en los archivos de disco, se han implementado las siguientes alternativas:

1. *Alternativa 1*: Cada proceso MPI escribe sus resultados en ficheros totalmente independientes entre sí, en lo que respecta a los diferentes tipos de resultados, a las diferentes hipótesis de un mismo resultado y con respecto al resto de procesos. Esto supone que cada proceso generará tantos ficheros como tipo de resultados haya obtenido, uno además por cada tipo de hipótesis de carga, siendo ficheros diferentes a los generados por cada proceso.

   Supongamos una estructura con de 3 hipótesis de carga aplicadas calculada mediante 2 tareas MPI. Los ficheros de resultados generados, en lo que respecta únicamente a los desplazamientos, serían los que se muestran en la figura 7.12. Convendría extrapolar el ejemplo mostrado con los desplazamientos para el resto de resultados.

   La ventaja de esta aproximación es que no conlleva ningún tipo de comunicación ni de sincronización entre las diferentes tareas a la hora de realizar la escritura de los datos. Además, no necesita un sistema de ficheros compar-

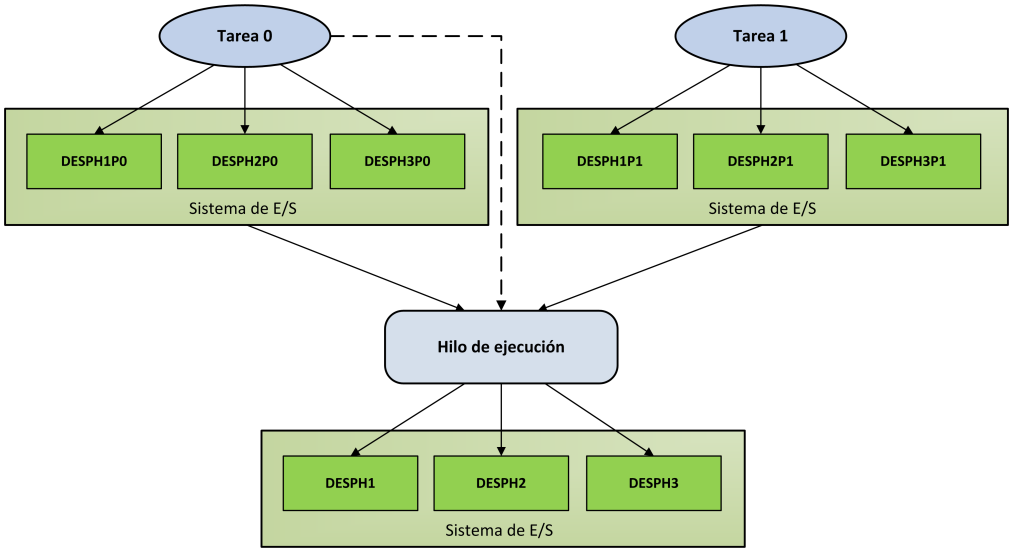


Figura 7.13: Alternativa 2 en la que cada tarea escribe sus resultados de forma individual pero se unen por medio de un hilo de ejecución.

tido para escribirlos, ya que cada tarea puede escribir en su disco local o en un directorio compartido para las diferentes tareas, si lo hubiera. El principal inconveniente es que los resultados estarán distribuidos entre un número más que considerable de ficheros, dificultando la recogida o la lectura de los mismos.

2. Una posible mejora frente a la aproximación anterior consistiría en que los resultados de un mismo tipo y una misma hipótesis estén agrupados en un solo fichero. Para llevarlo a cabo, se han implementado las siguientes alternativas:

   a) *Alternativa 2*: Una vez que los resultados se han almacenado en disco mediante la aproximación citada, la tarea 0 lanza un hilo encargado de empaquetar los resultados de un mismo tipo y generados por los diferentes procesos para una misma hipótesis, en un solo fichero. El resultado sería el que se muestra en la figura 7.13. Resulta evidente que, en este caso, es necesario disponer de un sistema de ficheros compartido, para que el hilo pueda leer los datos de salida generados por cada tarea.

   b) *Alternativa 3*: Únicamente la tarea 0 escribe los resultados de disco.

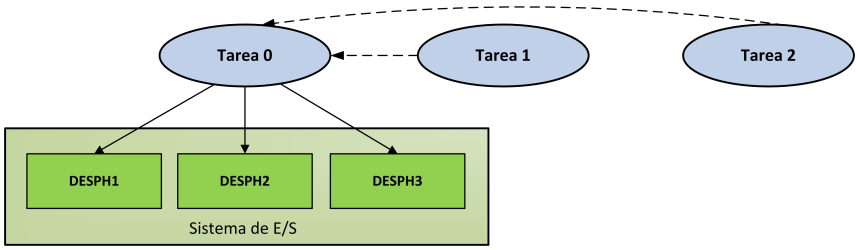


Figura 7.14: Alternativa 3 en la que sólo la tarea 0 escribe los múltiples ficheros de resultados.

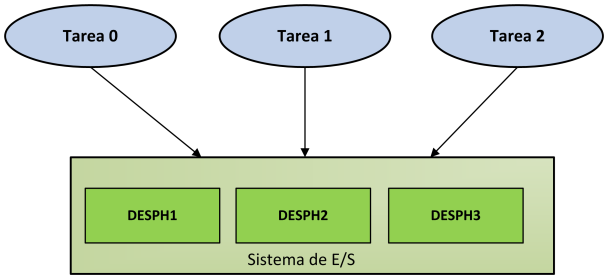


Figura 7.15: Alternativa 4 en la que todas las tareas escriben conjuntamente los múltiples ficheros de resultados.

Para ello, es necesario que el resto de tareas le envíen previamente los datos de salida, mediante la función MPI_Gatherv. Es evidente que, bajo esta aproximación, la escritura de los resultados se realizaría de forma secuencial. A ello hay que añadir además la sobrecarga del tiempo invertido en el envío de los datos por parte de todas las tareas a la tarea principal. El procedimiento que se sigue es el que podemos observar en la figura 7.14, donde participan 3 tareas en el cálculo de la estructura.

c) *Alternativa 4*: Todos los procesos escriben sus resultados, de manera sincronizada y en paralelo, como muestra la figura 7.15, mediante las funciones MPI_File_open, MPI_File_write_at_all y MPI_File_close de MPI.

3. Generamos un fichero único de salida, donde todos los procesos almacenan todos sus resultados. Para ello, caben las dos siguientes alternativas.

   a) *Alternativa 5*: Sólo escribe el proceso 0, quien recibe previamente los resultados del resto tras invocar a la función MPI_Gatherv. Esta aproximación, recogida en la figura 7.16, presenta los mismos inconvenientes que hemos visto en el caso de escribir los resultados en múltiples fi-

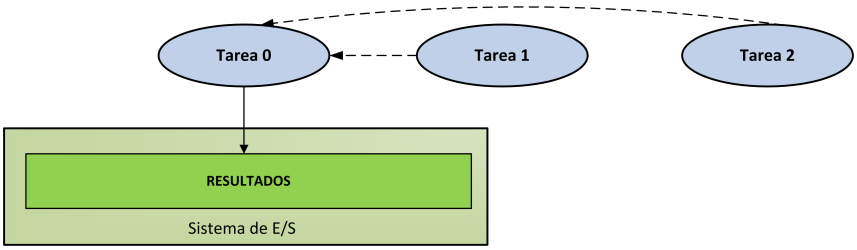


Figura 7.16: Alternativa 5 en la que sólo la tarea 0 escribe los resultados en un único fichero.

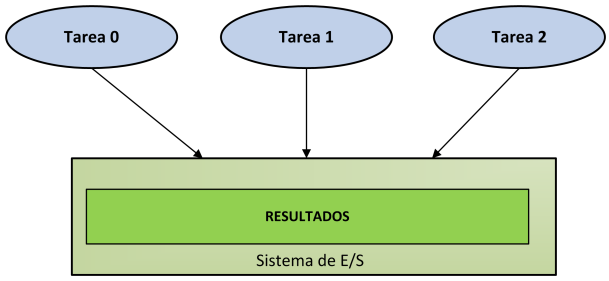


Figura 7.17: Alternativa 6 en la que todas las tareas escriben conjuntamente los resultados en un único fichero.

cheros, relativos a la sobrecarga que supone el envío de los datos y la secuencialidad en la escritura de los mismos.

b) *Alternativa 6*: Cada proceso escribe en paralelo, junto con el resto de procesos, los resultados que él mismo ha generado, como se muestra en la figura 7.17, empleando para ello las siguientes funciones de MPI-IO: MPI_File_open,MPI_File_get_position, MPI_File_write_at_all y MPI_File_close.

Aunque las alternativas en las que se genera un único fichero de salida ofrecen una mayor comodidad a aquellas otras aplicaciones encargadas de leer o recoger los resultados generados tras el cálculo, bien es verdad que el hecho de generar múltiples ficheros nos permite que, a la hora de realizar un análisis dinámico a lo largo del tiempo de forma remota, la aplicación cliente pueda ir recuperando los resultados de salida a medida que van siendo generados por el servicio Grid o Cloud, sin necesidad de esperar a que acabe la simulación para poder comenzar la recogida de los mismos, solapando así la recepción de los resultados con la simulación de la estructura y reduciendo considerablemente los tiempos de respuesta.

7.3. Paralelización del cálculo dinámico

Como ya vimos en el capítulo 4, el cálculo dinámico de una estructura puede realizarse mediante un análisis modal, con el objetivo de obtener sus frecuencias naturales y sus modos de vibración, mediante un análisis modal espectral, calculando así la respuesta máxima ante la actuación de un sismo, o mediante métodos de integración directa o por superposición modal, determinando la respuesta de la estructura a lo largo del tiempo tras la actuación de una carga dinámica cualquiera. Abordamos por tanto en esta sección la paralelización de los diferentes tipos de cálculo dinámico mencionados, junto con las diferentes etapas de las que cada uno consta. En todos ellos, el particionado de la estructura entre los distintos procesos MPI se llevará a cabo mediante las técnicas de particionado por corte de nodos y por corte de elementos descritas en el apartado 7.2.2.

7.3.1. Generación de las matrices de rigidez, masa y amortiguamiento

Para calcular la matriz de rigidez de la estructura ante un cálculo dinámico, seguiremos los mismos pasos ya mencionados en la sección 7.2.3 a la hora de generar dicha matriz en el caso de un análisis estático. Lo mismo podemos decir de la generación de la matriz de masa, cuyo cálculo en paralelo sigue las mismas pautas que la generación de la matriz de rigidez, teniendo en cuenta que generaremos la matriz de masa consistente y no la concentrada. Esto supone que cada proceso MPI será el encargado de generar, en paralelo con el resto de procesos, el conjunto de filas consecutivas de dicha matriz de masa que le haya sido asignado, en función de la nueva numeración de los nodos de su partición. Para ello, cada proceso generará la matriz de masa en ejes globales de los elementos estructurales (barras, triángulos, tetraedros y hexaedros) pertenecientes a su partición, empleando para ello las funciones de BLAS oportunas, ensamblando su aportación a la matriz global de la estructura tras invocar a la función MatSetValues de PETSc. A la aportación de la matriz de masa de los elementos estructurales hay que incorporar además la masa debida a las cargas aplicadas directamente sobre los nudos de la partición.

A nivel de OpenMP, paralelizaremos mediante el constructor #pragma omp parallel for los bucles que recorren los distintos elementos de la partición para generar su matriz de masa e incorporarla a la matriz de masa global, definiendo el ámbito (privado o compartido) de cada variable y empleando una distribución de tipo static de las iteraciones a los hilos de ejecución. A modo de ejemplo, el algoritmo 7 recoge la generación de la matriz de masa de los triángulos asignados a la partición de una tarea concreta y su aportación a la matriz de masa global. De nuevo, la operación más delicada será precisamente dicha aportación a la matriz global o ensamblaje, donde cada hilo sumará la matriz de masa de un elemento a la matriz M de masa global de la estructura, la cual se definirá como una variable compartida. Puesto que las funciones de PETSc no son seguras frente a la invocación simultánea a las mismas de diferentes hilos de ejecución, la llamada a la función MatSetValues debe formar parte de una región crítica, definida por el constructor #pragma omp critical, consiguiendo así que sólo sea un hilo el que pueda modificar al mismo tiempo el valor de dicha matriz.

Algoritmo 7 Aportación de las matrices de masa de los triángulos asignados a una tarea a la matriz M de rigidez global

Entrada: M (matriz de masa global), *triangulos_particion* (vector con el identificador de los triángulos asignados a la tarea), $nTriangulosParticion$ (número total de triángulos asignados a la tarea)
Salida: M (matriz de masa global)

1: **#pragma** omp **parallel** for **private** (t, M'^t, R^t, M^t)
2: **para** $triangulo = 1$ **hasta** $nTriangulosParticion$ **hacer**
3: t=*triangulos_particion[triangulo]*
4: Calcular la matriz de masa con comportamiento de lámina en ejes locales M'^t
5: Calcular la matriz de rotación R^t
6: Calcular la matriz de masa en ejes globales M^t
7: Ensamblar la matriz M^t en la matriz M
8: **fin para**

Finalmente, cada tarea MPI deberá llamar a las funciones MatAssemblyBegin y MatAssemblyEnd, gracias a las cuales tienen lugar las comunicaciones entre las tareas que participan conjuntamente en la generación de la matriz, siempre dependiendo del tipo de particionado de la malla de la estructura empleado.

Conviene aclarar que, en el caso de un análisis dinámico mediante métodos

de integración directa, las matrices globales de rigidez y masa de la estructura podrán ser o bien matrices completas, en las cuales consideramos a todos los grados de libertad de la estructura, o bien matrices condensadas, en las cuales no incorporaremos la aportación de todos aquellos grados de libertad que estén empotrados, que no formen parte del cálculo por realizarse un análisis en 2D o que se correspondan con los giros en los nudos de los elementos finitos tridimensionales (tetraedros y hexaedros). Sin embargo, en el caso de un análisis modal, modal espectral o modal por superposición en el tiempo, las matrices de rigidez y masa generadas siempre serán matrices condensadas.

Una vez generadas en paralelo las matrices globales de rigidez y masa, generaremos la matriz de amortiguamiento, de acuerdo al método de Rayleigh recogido en la expresión (4.23). Esto supone que cada tarea MPI generará en paralelo su bloque de filas de la matriz de amortiguamiento correspondiente, como combinación lineal de sus matrices de rigidez y masa. Se trata en definitiva de una operación realizada en paralelo en la que no habrá comunicaciones entre las tareas, implementada mediante la invocación a las funciones MatDuplicate, MatScale y MatAXPY de PETSc, de acuerdo a la siguiente secuencia de código:

```
/* Pretendemos que C=α0M+α1K */
MatDuplicate(M,MAT_COPY_VALUES,&C);    /* Copiamos la matriz M a C */
MatScale(C,alpha0);                    /* C = α0 * C */
MatAXPY(C,alpha1,K,SAME_NONZERO_PATTERN); /* C = C + α1 * K */
```

Desafortunadamente, PETSc no ofrece una implementación multihilo de ninguna de sus funciones, ni siquiera de aquellas relacionadas con operaciones básicas de álgebra lineal, motivo por el cual no podremos sacar partido de una paralelización a nivel de memoria compartida en lo que respecta a la generación de la matriz de amortiguamiento.

7.3.2. Generación del vector de cargas dinámicas

En el caso de un análisis dinámico a lo largo del tiempo, a partir del método de superposición modal o mediante métodos de integración directa, debemos generar para cada instante de tiempo el vector con la carga dinámica aplicada sobre los

nudos de la estructura. Dicho vector estará repartido de forma consecutiva entre las tareas, de acuerdo a la numeración de los nodos asignados a su partición. Dispondremos por una variable de tipo Vdmc con un número global de filas y un número local de filas en cada cada tarea idéntico al de las matrices de rigidez, masa o amortiguamiento (en el caso de la matriz completa, el número de filas locales se corresponderá con el número de nudos asignados a la partición multiplicado por 6, siendo dicho número inferior para la matriz condensada), además de un número de columnas equivalente al número de hipótesis de cargas dinámicas aplicadas.

Dicho vector se obtendrá, según lo descrito en la expresión (4.27) y en el caso de una carga sísmica, tras multiplicar la matriz de masa M por un vector obtenido como combinación de productos de vectores J de influencia por la aceleración del terreno para cada eje o, según la expresión (4.29), como producto de la matriz de masa por el vector J, multiplicado previamente por el valor de la cargas dinámicas tipo o genéricas aplicadas sobre los nudos expresadas en forma de aceleración, a lo que hay que añadir el vector con las cargas dinámicas expresadas mediante fuerzas aplicadas directamente sobre los nudos de la estructura.

Cada proceso MPI generará por tanto en paralelo sus filas correspondientes al vector J de influencia en uno de los ejes, multiplicadas por el vector de la aceleración de la carga aplicada en sus nudos y lo sumará, si procede, a la aportación del resto de vectores de influencia para los restantes ejes. Tras ello, dicho vector resultante se multiplicará en paralelo a la matriz global de masa mediante la función MatMult (tantas veces como número de hipótesis dinámicas diferentes tengamos en el cálculo de la estructura) y el resultado se sumará, en paralelo, al vector con las fuerzas dinámicas aplicadas sobre los nudos, por medio de la función VecAxpy y en caso de que dichas fuerzas se apliquen. Más en detalle, dichas funciones de PETSc consisten en:

- MatMult: Lleva a cabo la operación $y = Ax$, multiplicando una matriz A dispersa por un vector x. Si la operación se realiza en paralelo, como es nuestro caso, A e y deberán estar distribuidos apropiadamente entre las diferentes tareas, tras lo cual el vector y queda repartido de igual modo.

- VecAXPY: Implementa en la operación $y = y + \alpha x$, donde α se corresponde con un número real y donde x e y son vectores.

En la generación del vector de cargas dinámicas no hemos empleado el paradigma de memoria compartida. En primer lugar, debido a la escasa complejidad computacional subyacente y, en segundo lugar, porque las funciones MatMult y VecAXPY, como ocurre con el resto de funciones de PETSc, no están basadas en técnicas multihilo.

7.3.3. Paralelización del análisis dinámico mediante métodos de integración directa

El cálculo dinámico lineal de una estructura empleando métodos de integración directa está compuesto por diferentes etapas, tal y como nos indica el algoritmo 8. El bucle principal que allí aparece recorre todos los instantes de tiempo en los que obtendremos la respuesta de la estructura, ante una carga aplicada que va cambiando para cada instante. Dicho bucle no puede ser paralelizado a nivel de OpenMP, ya que los desplazamientos, las velocidades y las aceleraciones en los nudos dependen, por lo general, de dichos valores obtenidos en el instante de tiempo anterior. En consecuencia, tendremos que aplicar la paralelización, tanto en lo que se refiere a MPI como a OpenMP, para cada instante de tiempo y, más concretamente, a título particular en las distintas etapas de las que consta dicho cálculo. Otra cuestión a reseñar es que sólo obtendremos resultados adicionales a los desplazamientos, velocidades y aceleraciones en los nudos en aquellos instantes de tiempo en los que vayamos a almacenar la respuesta estructural en ficheros de disco.

Para llevar a cabo este tipo de cálculo, hemos paralelizado los 8 siguientes métodos de integración directa, todos ellos detallados en la sección 4.7: Diferencias Centradas, Newmark, Wilson-θ, Houbolt Monopaso, HHT-α, WBZ-α, Generalizado-α y SDIRK. Básicamente, todos ellos consisten en resolver un sistema de ecuaciones lineales en el que determinan los desplazamientos en los nudos. Para ello, previamente generan una matriz de coeficientes, o matriz de rigidez efectiva del sistema, y un vector parte derecha denominado vector de carga dinámica efectiva. Si bien la matriz de rigidez efectiva no varía para cada instante de tiempo, por tratarse de un análisis lineal, sí que cambia el vector de la carga efectiva, debido a la naturaleza dinámica de la carga aplicada. A partir de los

Algoritmo 8 Cálculo dinámico mediante métodos de integración

Entrada: *ntarea* (identificador de la tarea MPI), $TMAX$ (tiempo de simulación), ΔT_s (incremento de tiempo en la simulación), ΔT_g (incremento de tiempo en la grabación de los resultados)

Salida: Resultados de cálculo

si *ntarea*=0 **entonces**
  Lectura del fichero con la geometría
  Envío de la geometría al resto de tareas
si no
  Recepción de la geometría
fin si
si *ntarea*=0 **entonces**
  Particionado de la malla de la estructura
  Envío del particionado al resto de tareas
si no
  Recepción del particionado
fin si
Reordenación y clasificación de los nodos de la estructura
Generación de la matrices de rigidez, masa y amortiguamiento
Cálculo de los desplazamientos, velocidades y aceleraciones iniciales en los nudos
Generación de la matriz de rigidez efectiva
Imposición de las condiciones de contorno
$npasos = TMAX/\Delta T_s$ /* Número de pasos de simulación */
$g = 0$ /* Contabiliza los pasos de grabación */
para $p = 0$ **hasta** *npasos* **hacer**
  Cálculo del vector de cargas dinámicas aplicadas
  Generación del vector de cargas dinámicas efectivas
  Cálculo de los desplazamientos, velocidades y aceleraciones en los nudos
  si $(t * \Delta T_s = g * \Delta T_g)$ **entonces**
    /* Hay que escribir resultados en disco para este instante de tiempo */
    Obtención de solicitaciones en los extremos de las barras
    Cálculo de reacciones en apoyos
    Generación de esfuerzos y deformaciones en puntos intermedios de las barras
    Cálculo de esfuerzos en nudos de elementos finitos 2D
    Cálculo de deformaciones unitarias, esfuerzos y tensiones en nudos de elementos finitos
    Escritura de los ficheros de resultados
    $g = g + 1$
  fin si
fin para

desplazamientos en los nudos, en el instante actual y tal vez en el anterior, y de las velocidades y aceleraciones en el instante previo, determinaremos los nuevos valores de velocidades y aceleraciones en el instante actual.

Cada tarea obtendrá por tanto los desplazamientos, las velocidades y las aceleraciones en sus nudos locales, para cada instante de tiempo. En consecuencia, las tres variables que almacenen dicha información para cada tarea será del tipo Vdmc, con un número de filas locales igual al número de grados de libertad considerados en todos sus nudos locales (en el caso del problema condensado) o del número del nudos locales multiplicado por 6 (en el caso del problema completo) y con un número de columnas equivalente al de hipótesis dinámicas aplicadas. Dicha distribución de datos será por tanto idéntica a los vectores que almacenan las cargas dinámicas aplicadas, en cada instante, o los que guardan las cargas dinámicas efectivas.

Sin embargo, conviene recordar que cada tarea deberá conocer adicionalmente los desplazamientos en sus nudos adyacentes, puesto que dicha información será necesaria para proceder con el resto de etapas del cálculo, no siendo así en el caso de las velocidades y las aceleraciones. Ello implica la necesidad de trabajar, únicamente en el caso de los desplazamientos, con los denominados nudos ghost, los cuales se corresponden, como ya vimos, con los nudos adyacentes de cada partición. De este modo y a la hora de reservar espacio para el vector de los desplazamientos, cada tarea reservará un número de filas ghost correspondiente a los grados de libertad considerados en sus nudos adyacentes de la partición, el cual será igual al número de nodos adyacentes multiplicado por 6 para el problema completo, indicando además los índices de las filas globales de dichos grados de libertad.

De este modo, aunque única y exclusivamente para aquellos pasos de tiempo en los que haya que calcular los resultados del resto de etapas y escribirlos en disco, cada tarea deberá recibir del resto de tareas vecinas los desplazamientos en sus nudos adyacentes. Como ya vimos en el caso del análisis estático, dicha comunicación se realizará por medio de las funciones VecGhostUpdateBegin y VecGhostUpdateEnd, empleando los parámetros INSERT_VALUES y SCATTER_FORWARD.

Antes de comenzar el proceso iterativo de cálculo, debemos obtener los des-

plazamientos, las velocidades y las aceleraciones iniciales en los nudos. En nuestro caso, supondremos que los desplazamientos y las velocidades son nulos, aunque podrían tomar un valor distinto de partida, obteniendo las aceleraciones fruto de la resolución de un sistema de ecuaciones lineales en paralelo donde aparece la matriz de masa como matriz de coeficientes.

A partir de dichos valores iniciales, la siguiente etapa a paralelizar es la generación de la matriz de rigidez efectiva, la cual se obtiene como combinación de las matrices de rigidez, masa y amortiguamiento. Para ello, emplearemos las funciones MatDuplicate, MatScale y MatAXPY de PETSc que ya citamos en el cálculo de la matriz de amortiguamiento. Dicha matriz de rigidez efectiva quedará distribuida entre las diferentes tareas de igual a modo a como lo están el resto de matrices del problema.

En los diferentes métodos de integración, la generación del vector de carga dinámica efectiva tiene lugar, básicamente, por medio de sumas de vectores, multiplicados con anterioridad por una constante, y productos matriz por vector. Entre los vectores, aparece la carga dinámica aplicada en los nudos en el instante de tiempo actual y, dependiendo del método, en el instante anterior, junto con los desplazamientos, velocidades y aceleraciones en los nudos calculados en el instante de tiempo previo. Entre las matrices, encontramos a las de rigidez, masa y amortiguamiento. Todas estas operaciones las realizaremos en paralelo por medio de las funciones VecCopy, VecScale, VecAXPY, VecAYPX, VecWAXPY, MatMult y MatMultAdd de PETSc. A continuación detallamos el comportamiento de aquellas funciones que no hemos descrito hasta el momento. Tanto los vectores como las matrices que empleemos estarán distribuidos apropiadamente entre las tareas:

- VecCopy: Copia el contenido de un vector a otro, es decir, y=x.
- VecAYPX: Implementa la operación $y = x + \alpha y$, donde α se corresponde con un número real y donde x e y son vectores.
- VecWAXPY: Realiza la operación $w = \alpha x + y$, siendo α un número real y siendo w, x e y vectores.
- VecAXPBY: Lleva a cabo la operación $y = \alpha x + \beta y$, donde α y β son dos números reales y donde x e y son vectores.

- MatMultAdd: Se encarga de la operación $z = y + Ax$, siendo A una matriz y siendo x, y y z vectores.

A la hora de calcular los desplazamientos en los nudos, los diferentes métodos implementados deben resolver, para cada instante de tiempo, un sistema de ecuaciones lineales con tantos vectores partes derecha como número de hipótesis dinámicas diferentes aparezcan en el cálculo. Dichos sistemas los resolveremos en paralelo mediante métodos iterativos, empleando para ello los métodos de los PETSc dispone, así como sus propios precondicionadores o los que forman parte de *hypre*, o mediante métodos directos, por medio de las librerías MUMPS, PaStiX o PARDISO. Puesto que la matriz de rigidez efectiva permanece constante a lo largo del tiempo, los métodos directos serán a priori mucho más eficientes que los iterativos, ya que llevarán a cabo la fase de ordenación, factorización simbólica y factorización numérica en el primer instante de tiempo, resolviendo únicamente sistemas de ecuaciones triangulares en el resto de instantes. Como es bien sabido, los métodos iterativos no podrán beneficiarse de la peculiaridad de que la matriz de coeficientes permanezca inalterada durante todo el proceso de análisis.

Conviene destacar que el método SDIRK necesita resolver, para cada instante de tiempo y para cada hipótesis de cálculo, un sistema de ecuaciones con un número de vectores partes derecha equivalente al número de etapas (2, 3 ó 4) empleadas en el método, siendo por tanto más costoso a nivel computacional que el resto. Por otro lado, los vectores solución obtenidos con este método deben combinarse con los desplazamientos y las velocidades en el instante de tiempo anterior para obtener los desplazamientos en el instante actual.

Finalmente, una vez calculados los desplazamientos en los nudos, determinaremos las velocidades y las aceleraciones en ese mismo instante de tiempo. Dichos valores se obtendrán mediante operaciones vectoriales en paralelo, actuando normalmente como operandos los desplazamientos actuales y los desplazamientos, las velocidades y aceleraciones en el instante de tiempo anterior. Para ello, emplearemos las funciones VecCopy, VecScale, VecAXPY, VecAYPX y VecWAXPY, ya descritas con anterioridad.

De nuevo la excepción viene del lado del método SDIRK, el cual obtiene las aceleraciones para cada instante de tiempo resolviendo un sistema de ecuaciones,

en el cual la matriz de masa actuará como matriz de coeficientes. En cuanto al vector parte derecha, estará formado por productos matriz por vector y restas de vectores entre las matrices de rigidez y amortiguamiento y los vectores que almacenan la carga dinámica, los desplazamientos y las velocidades en el instante de tiempo actual. El cálculo de dichas aceleraciones es totalmente opcional, ya que no se necesitan para calcular los desplazamientos o las velocidades en el instante de tiempo siguiente.

La paralelización del resto de etapas necesitadas en este tipo de análisis (reacciones en apoyos, esfuerzos y deformaciones en puntos intermedios de las barras o deformaciones unitarias, tensiones y esfuerzos en nudos de elementos finitos) ya han sido descritas previamente, en el caso del cálculo estático.

7.3.4. Paralelización del análisis dinámico lineal mediante técnicas de análisis modal

El objetivo primordial del análisis modal consiste en obtener las frecuencias naturales y los modos de vibración de la estructura. A continuación, este tipo de cálculo proporcionará además los porcentajes de masas efectivas para cada modo de vibración y el resto de resultados ya conocidos para cada modo de vibración. Los pasos necesarios para realizar este tipo de análisis están recogidos en el algoritmo 9.

Una vez que cada tarea ha generado su parte local de las matrices condensadas de rigidez y masa y ha sumado la rigidez de los apoyos elásticos de su partición a los elementos correspondientes de la diagonal de la matriz de rigidez, el paso siguiente y primordial en este tipo de análisis consiste en calcular las frecuencias naturales y los modos de vibración de la estructura, resolviendo en paralelo el problema de valores propios generalizado de la expresión (4.94). Para ello, hemos empleado la librería SLEPc, la cual ha sido elegida por sus buenas prestaciones y por su flexibilidad y facilidad de uso. Además de usar sus propias implementaciones, SLEPc nos permite emplear otros métodos de resolución presentes en diferentes librerías numéricas, como ARPACK (la cual también hemos utilizado), BLOPEX o FEAST. Como ya comentamos en el apartado 5.7.5.3, SLEPc está

Algoritmo 9 Análisis modal

Entrada: *ntarea* (identificador de la tarea MPI)
Salida: Resultados de cálculo

si *ntarea*=0 **entonces**
    Lectura del fichero con la geometría
    Envío de la geometría al resto de tareas
si no
    Recepción de la geometría
fin si
si *ntarea*=0 **entonces**
    Particionado de la malla de la estructura
    Envío del particionado al resto de tareas
si no
    Recepción del particionado
fin si
Reordenación y clasificación de los nodos de la estructura
Generación de la matrices de rigidez y masa
Imposición de las condiciones de contorno en la matriz de rigidez
Cálculo de las frecuencias naturales y los modos de vibración
Cálculo de las masas efectivas
Obtención de solicitaciones en los extremos de las barras
Cálculo de reacciones en apoyos
Generación de esfuerzos y deformaciones en puntos intermedios de las barras
Cálculo de esfuerzos en nudos de elementos finitos 2D
Cálculo de deformaciones unitarias, esfuerzos y tensiones en nudos de elementos finitos
Escritura de los ficheros de resultados

basado en PETSc y puede considerarse una extensión al mismo, motivo por el cual emplearemos sus propios métodos iterativos para resolver los sistemas de ecuaciones lineales que aparecen en cada paso de la resolución del problema de valores propios o aquellos métodos que forman parte de otras librerías numéricas de las cuales actúa como interfaz, como MUMPS, PaStiX o PARDISO. Al igual que ocurría con PETSc, SLEPc tampoco ofrece una paralelización multihilo.

El tipo de datos principal de SLEPc para resolver el problema de valores propios es EPS, el cual presenta un uso similar al de KSP en PETSc. Todos los parámetros que recogen el problema a resolver y el método a utilizar se pueden proporcionar en tiempo de ejecución desde la línea de comandos o se pueden emplear las funciones específicas, invocándolas desde nuestro código.

Adicionalmente, es también necesario configurar el tipo de transformación espectral a aplicar. El tipo de datos ST encapsula toda la funcionalidad requerida por las técnicas de aceleración basadas en transformaciones espectrales. Su uso pretende ser análogo al tipo de dato PC de PETSc. Como es habitual, es posible especificar dicha transformación desde la línea de comandos o invocar a las funciones oportunas. Puesto que algunas de las transformaciones, como la denominada de *desplazamiento e inversión*, conllevan la solución de un sistema de ecuaciones lineales, tendremos también que indicar el método de resolución a emplear.

La selección del desplazamiento σ empleado en las técnicas de transformación espectral no es trivial y merece alguna consideración. Una primera aproximación podría consistir en asignar un valor nulo a dicho desplazamiento, lo cual sería perfectamente válido para nuestro tipo de problema. Sin embargo, la convergencia del método empleado es más rápida cuanto más cercano sea el desplazamiento empleado a los valores propios buscados. En ese sentido, en [334] se propone la siguiente estrategia heurística para calcular el valor del desplazamiento σ_h:

$$\sigma_h = \frac{1}{\sqrt{n}\sum_{i=1}^{n}\frac{M_{ii}}{K_{ii}}} \tag{7.5}$$

siendo n el número de grados de libertad considerados en la estructura. En todos nuestras pruebas realizadas, el desplazamiento σ_h calculado siempre satisface que $0 < \sigma_h < \omega_1^2$, estando en todos los casos alejado del valor propio más pequeño.

Sin embargo, conviene saber que un valor de σ_h mayor que ω_1^2 supondría que la matriz de coeficientes $K - \sigma M$ que aparece en el sistema de ecuaciones a resolver debido a la transformación espectral no sería definida positiva, lo que tendría importantes consecuencias en el método de resolución de sistemas de ecuaciones lineales. En particular, el método directo que aplicáramos debería ser válido para matrices indefinidas y el método iterativo vería seriamente perjudicada su convergencia a la solución del problema. No en vano si el valor del desplazamiento fuera demasiado elevado, correríamos el riesgo de no obtener los valores propios más pequeños, los cuales son precisamente los requeridos y de mayor importancia en nuestro tipo problema. En consecuencia, dicha técnica heurística será la que consideraremos.

A continuación se incluye el código que nos serviría de ejemplo de uso de la librería SLEPc para calcular las frecuencias naturales y los modos de vibración de una estructura, empleando la técnica de desplazamiento e inversión como transformación espectral. Aunque calculemos los valores propios más grandes y más próximos al desplazamiento, en realidad se tratará de los más pequeños, debido a la transformación empleada:

```
EPS eps; /* Declaramos la variable de tipo EPS */
ST st;   /* Declaramos la variable de tipo ST */
EPSCreate(PETSC_COMM_WORLD,&eps); /* Creamos el contexto */
/* Establecemos a las matrices de rigidez y masa como operadores del problema */
EPSSetOperators(eps,K,M);
/* Indicamos el tipo de problema a resolver: simétrico y generalizado */
EPSSetProblemType(eps,EPS_GHEP);
EPSSetFromOptions(eps); /* Leemos los parámetros de la línea de comandos */
EPSGetST(eps,&st);      /* Obtenemos la transformación espectral */
STSetShift(st,σ_h);     /* Asignamos el valor al desplazamiento */
/* Determinamos la estructura de la matriz del problema transformado */
STSetMatStructure(st,SAME_NONZERO_PATTERN);
/* Indicamos el modo almacenar la matriz en el problema transformado por la
   transformación espectral */
STSetMatMode(st,STMATMODE_COPY);
/* Determinamos la magnitud de los valores propios a calcular */
```

```
EPSSetWhichEigenpairs(eps,EPS_TARGET_MAGNITUDE);
EPSSolve(eps); /* Resolvemos el problema */
/* Obtenemos el número de valores y vectores propios que convergen */
EPSGetConverged(eps,&nconv);
/* Obtenemos dichos valores y vectores propios */
for (i=0;i<nconv;i++)
  EPSGetEigenpair(eps,i,&valor_propio.valores[i],PETSC_NULL,
                        vector_propio.vec_petsc[i],PETSC_NULL);
EPSDestroy(&eps); /* Destruimos el contexto y liberamos memoria */
```

En dicho ejemplo, valor_propio y vector_propio serán dos variables de tipo Vdmc, la primera de las cuales consistirá en un vector secuencial, replicado entre todas las tareas y de longitud igual al número de frecuencias naturales a calcular, y la segunda se tratará de un vector distribuido por bloques de filas consecutivas con un tamaño local equivalente al número de grados de libertad considerados en cada partición. Dicha segunda variable tendrá una longitud global igual al número total de grados de libertad del problema condensado y tantas columnas como número de frecuencias naturales hayamos considerado, sin emplear filas ghost.

Mediante diferentes opciones, podremos configurar el problema en tiempo de ejecución y el modo de resolverlo. Así por ejemplo, las opciones -eps_nev y -eps_ncv recogen respectivamente el número de valores propios a calcular y el número de vectores, o longitud máxima del subespacio de trabajo, usados por el algoritmo de resolución, el cual se recomienda que sea al menos el doble del número de valores propios. El método de resolución viene especificado por la opción -eps_type. Otras posibles opciones a emplear son -eps_maxit y -eps_tol, las cuales ajustan el número máximo de iteraciones permitidas en el algoritmo de resolución y la tolerancia a emplear como criterio de parada.

Por otro lado, es necesario también configurar la técnica de transformación espectral empleada. Para ello, SLEPc dispone entre otras de las opciones -st_type, para recoger el tipo de transformación elegida, o -st_ksp_type y -st_pc_type, las cuales nos permiten indicar el método de resolución del sistema de ecuaciones del problema transformado y el precondicionador empleado, respectivamente.

A modo de ejemplo, la siguiente línea de ejecución la emplearíamos junto a nuestro fichero ejecutable para calcular 15 frecuencias naturales de vibración empleando para ello el método de Lanczos, con un número máximo de 100000 iteraciones y una tolerancia de 1e-6, aplicando la transformación espectral de desplazamiento e inversión y la resolución de los sistemas de ecuaciones mediante MUMPS:

-eps_nev 15 -eps_ncv 30 -eps_type lanczos -eps_tol 1e-6 -eps_max_it 100000 -st_type sinvert -st_ksp_type preonly -st_pc_type cholesky -st_pc_factor_mat_solver_package mumps

Una vez hallados dichos valores propios, calcularemos la raíz cuadrada de los mismos para obtener las frecuencias angulares de vibración ω, expresadas en radianes por segundo, y dividiremos por 2π para transformarlas a hercios, disponiendo así de las frecuencias naturales de vibración de la estructura.

La normalización en paralelo de los vectores propios calculados, o modos de vibración del sistema, según la expresión (4.97) conlleva realizar operaciones colectivas de álgebra lineal entre las diferente tareas, entre las que se encuentran, para cada vector propio, el producto de la matriz de masa por dicho vector propio, llevado a cabo mediante la función MatMult, y el producto escalar del vector resultante por el vector propio de nuevo, empleando para ello la función VecDot de PETSc, la cual calcula en paralelo el producto escalar de dos vectores, replicando el resultado en las diferentes tareas participantes. A continuación, mediante la directiva #pragma omp parallel for, cada tarea puede paralelizar el cociente de los elementos de cada vector propio que le han sido asignados entre la raíz cuadrada del producto escalar anterior.

De modo muy similar a la paralelización de la normalización de los vectores propios, el cálculo en paralelo de la masa efectiva de cada modo de vibración como porcentaje de la masa efectiva total de la estructura, de acuerdo a la expresión (4.106), requiere realizar en paralelo diferentes productos matriz por vector y productos escalares entre vectores, donde participan la matriz de masa de la estructura, los modos de vibración calculados y el vector J de influencia, operaciones todas ellas llevadas a cabo de manera colectiva entre las tareas gracias a las funciones MatMult y VecDot.

Los modos de vibración calculados son en realidad los desplazamientos relativos que tienen lugar en los nudos ante una vibración libre de la estructura, para aquellos grados de libertad considerados en el problema condensado, siendo nulos para el resto. Eso supone por tanto que cada tarea habrá obtenido dichos desplazamientos en determinados grados de libertad correspondientes a sus nudos locales. Si a continuación deseamos continuar con el resto de etapas asociadas al cálculo, por ejemplo para obtener y representar gráficamente las formas modales, cada tarea deberá disponer de los valores de los modos de vibración en sus nudos adyacentes.

Es por ello que generaremos una nueva variable de tipo Vdmc encargada de almacenar, de forma distribuida, los valores de los modos de vibración de todos los grados de libertad de la estructura, tratándose por tanto del problema completo. Así, cada tarea copiará a dicha variable los valores de los modos de vibración de sus grados de libertad participantes en el problema condensado, asignando un valor nulo al resto. Como filas ghost, cada tarea reservará espacio e indicará los índices de los grados de libertad correspondientes a sus nudos adyacentes, realizando a continuación las comunicaciones oportunas entre las diferentes tareas por medio de las funciones VecGhostUpdateBegin y VecGhostUpdateEnd de PETSc ya conocidas.

7.3.5. Paralelización del análisis dinámico mediante el método de superposición modal

Como alternativa a los métodos de integración directa, el análisis por superposición modal también proporciona la respuesta de la estructura a lo largo del tiempo aunque, en lugar de integrar la ecuación dinámica de movimiento, la desacopla y la resuelve para cada modo de vibración, obteniendo la respuesta del sistema como combinación lineal de la respuesta de cada modo individual. El algoritmo 10 incluye todas las etapas necesarias para realizar este tipo de análisis estructural en paralelo.

De acuerdo a dicho algoritmo, la primera de las etapas no descritas hasta el momento es la del cálculo en paralelo de los factores de participación de cada

Algoritmo 10 Cálculo dinámico mediante superposición modal

Entrada: *ntarea* (identificador de la tarea MPI), *TMAX* (tiempo de simulación), ΔT_s (incremento de tiempo en la simulación), ΔT_g (incremento de tiempo en la grabación de los resultados)

Salida: Resultados de cálculo

si *ntarea*=0 **entonces**
  Lectura del fichero con la geometría
  Envío de la geometría al resto de tareas
si no
  Recepción de la geometría
fin si
si *ntarea*=0 **entonces**
  Particionado de la malla de la estructura
  Envío del particionado al resto de tareas
si no
  Recepción del particionado
fin si
Reordenación y clasificación de los nodos de la estructura
Generación de la matrices de rigidez y masa
Imposición de las condiciones de contorno en la matriz de rigidez
Cálculo de las frecuencias naturales y los modos de vibración
Obtención de las masas efectivas
Generación de los factores de participación
$npasos = TMAX/\Delta T_s$ /* Número de pasos de simulación */
$g = 0$ /* Contabiliza los pasos de grabación */
para $p = 0$ **hasta** *npasos* **hacer**
  Evaluación del vector de cargas dinámicas aplicadas
  Cálculo del vector de cargas dinámicas efectivas
  Obtención de las amplitudes modales y de sus derivadas
  Cálculo de los desplazamientos, velocidades y aceleraciones en los nudos
  si $(t * \Delta T_s = g * \Delta T_g)$ **entonces**
    /* Hay que escribir resultados en disco para este instante de tiempo */
    Obtención de solicitaciones en los extremos de las barras
    Cálculo de reacciones en apoyos
    Generación de esfuerzos y deformaciones en puntos intermedios de las barras
    Cálculo de esfuerzos en nudos de elementos finitos 2D
    Cálculo de deformaciones unitarias y tensiones en nudos de elementos finitos
    Escritura de los ficheros de resultados
    $g = g + 1$
  fin si
fin para

modo. Como podemos observar en la expresiones (4.119) y (4.120), dicho cálculo supone realizar, para cada modo de vibración y para cada uno de los ejes, un producto matriz por vector y un producto escalar de vectores, usando por tanto las funciones apropiadas de PETSc de forma colectiva entre las diferentes tareas, al igual que ocurría en el cálculo de las masas efectivas para cada modo. De esta manera, el factor de participación de cada uno de los modos quedará replicado, en forma de un vector secuencial, entre las diferentes tareas.

En cuanto al cálculo del vector de amplitud modal y de sus derivadas, debemos resolver un conjunto de q ecuaciones desacopladas de segundo orden, cada una de ellas de un único grado de libertad, como queda recogido en la expresión (4.116). Como primer paso, repartiremos de forma consecutiva y equitativa cada una de las ecuaciones desacopladas entre las tareas involucradas en el cálculo de la estructura, para su resolución. Obviamente si el número de ecuaciones es inferior al número de tareas, algunas de ellas permanecerán ociosas en esta etapa de análisis. Para almacenar dicho vector de amplitudes modales y sus derivadas de forma distribuida entre las tareas, emplearemos 3 variables de tipo Vdmc, sin necesitar filas ghost y con tantas columnas como número de hipótesis dinámicas diferentes estén siendo aplicadas sobre la estructura.

Por medio de la directiva #pragma omp parallel for, cada tarea desplegará un conjunto de hilos de ejecución encargados de resolver cada una de las ecuaciones desacopladas de un grado de libertad que le han sido asignadas. Precisamente para resolver dichas ecuaciones,hemos implementado los métodos de Diferencias Centradas, Newmark, Wilson-θ, Houbolt Monopaso, HHT-α, WBZ-α, Generalizado-α, SDIRK y Duhamel.

No obstante, antes de resolver las citadas ecuaciones, debemos generar en paralelo el vector parte derecha correspondiente, llamado vector de cargas dinámicas efectivas. Para ello, realizaremos un conjunto de q productos escalares, implementados mediante la función VecDot, donde cada tarea multiplica su parte local de los modos de vibración por el vector de carga dinámica aplicada sobre los grados de libertad considerados en su partición. Puesto que el resultado de dicho producto escalar se envía a todas las tareas involucradas, el vector de carga dinámica efectiva será un vector secuencial replicado entre todas ellas, para cada hipótesis de carga dinámica aplicada. En consecuencia, cada tarea descartará aquellos ele-

mentos de dicho vector que no se correspondan con las ecuaciones que le han sido asignadas. El cálculo del vector de carga dinámica aplicada sobre los grados de libertad de cada partición considerados en el problema se obtendrá de acuerdo a lo descrito en el apartado 7.3.2, teniendo en cuenta que estamos trabajando con el problema condensado.

Una vez que se han resuelto dichas ecuaciones desacopladas, los vectores de amplitudes modales y sus derivadas habrán quedado distribuidos entre cada una de las tareas. Consecuentemente, cada tarea deberá enviar al resto los elementos que ha calculado, con el propósito de que todas ellas dispongan de tres vectores secuenciales y replicados, empleando para ello la función MPI_Allgatherv. Seguidamente, cada tarea multiplicará cada elemento de dichos vectores por su parte local del modo de vibración correspondiente, acumulando los diferentes productos y calculando así en paralelo los desplazamientos, las velocidades y las aceleraciones en los grados de libertad considerados de los nudos locales a su partición, de acuerdo a las expresiones (4.114). En este caso, será la función VecMAXPY de PETSc la que emplearemos para calcular dichos resultados, los cuales se quedarán repartidos entre las tareas de acuerdo a una distribución por bloques de filas consecutivas. La citada función VecMAXPY realiza la operación $y = y + \sum_{i=1}^{q} \alpha_i x_i$, siendo y un vector de n componentes, α un vector de q números reales y x un array de q vectores, cada uno de ellos de n elementos.

En caso de que debamos escribir los resultados de este instante de tiempo en uno o más ficheros de disco, será necesario proceder con el resto de etapas del cálculo. Para ello, de igual modo a como vimos en el caso del análisis modal, cada tarea debe disponer de antemano de los desplazamientos en todos los grados de libertad de su partición, bien sean pertenecientes a nudos locales o a nudos adyacentes. En consecuencia, cada tarea reservará espacio para una variable de tipo Vdmc donde se guardarán, de forma repartida entre todas ellas, los desplazamientos de todos los grados de libertad de la estructura, incorporando como filas ghost las correspondientes a los grados de libertad de los nudos adyacentes a su partición. De ese modo, cada una de las tareas copiará los valores del vector de desplazamientos condensados al vector de desplazamientos completos, pondrá a 0 los desplazamientos del resto de grados de libertad de sus nudos locales y realizará las comunicaciones oportunas con el resto de tareas, gracias a las funciones

VecGhostUpdateBegin y VecGhostUpdateEnd, para completar los desplazamientos en los nudos adyacentes.

7.3.6. Paralelización del análisis dinámico mediante el método modal espectral

El método modal espectral, como alternativa a los métodos de integración directa y superposición modal, obtiene la respuesta máxima de la estructura combinando la respuesta individual de cada uno de los modos, en lugar de calcular el comportamiento de la estructura a lo largo del tiempo. El método está compuesto por un conjunto de etapas documentadas en el algoritmo 11.

Para cada una de las hipótesis espectrales aplicadas, cada tarea comienza calculando en paralelo los desplazamientos, las velocidades y las aceleraciones máximas en los grados de libertad que formen parte del problema condensado y pertenezcan a los nudos locales de su partición, de acuerdo a la formulación correspondiente a la expresión (4.122). Eso supone que debemos reservar 3 vectores de tipo Vdmc para almacenar dicha información con unas dimensiones y una distribución de datos por bloques de filas consecutivas entre las tareas idéntica a la que presentan los modos de vibración de la estructura.

A su vez, cada tarea paraleliza mediante OpenMP, y más concretamente mediante la directiva #pragma omp parallel for, el cálculo de los desplazamientos, las velocidades y las aceleraciones máximas en dichos grados de libertad para cada modo de vibración. Para ello, cada hilo de ejecución, de acuerdo al modo de vibración correspondiente a la iteración que le haya sido asignada, debe multiplicar los valores de la forma modal de dicho modo asignados a la tarea por su factor de participación y por su desplazamiento espectral, velocidad espectral o aceleración espectral máxima. Dichos productos y cálculos de los desplazamientos, velocidades y aceleraciones en los nudos se llevarán a cabo por medio de las correspondientes invocaciones a las funciones VecCopy y VecScale de PETSc.

Conviene recordar que el factor de participación consiste en un vector secuencial previamente calculado y replicado en cada tarea, como vimos en el apartado anterior, y que los valores del desplazamiento y la velocidad espectral máxima

Algoritmo 11 Cálculo dinámico mediante análisis modal espectral

Entrada: *ntarea* (identificador de la tarea MPI), *grabar_result_modos_hipot* (indica a 1 o a 0 si grabamos o no los resultados para cada modo de vibración y para cada hipótesis espectral)

Salida: Resultados de cálculo

si *ntarea*=0 **entonces**
  Lectura del fichero con la geometría
  Envío de la geometría al resto de tareas
si no
  Recepción de la geometría
fin si
si *ntarea*=0 **entonces**
  Particionado de la malla de la estructura
  Envío del particionado al resto de tareas
si no
  Recepción del particionado
fin si
Reordenación y clasificación de los nodos de la estructura
Generación de la matrices de rigidez y masa
Imposición de las condiciones de contorno en la matriz de rigidez
Cálculo de las frecuencias naturales y los modos de vibración
Obtención de las masas efectivas
Generación de los factores de participación
para $h = 0$ **hasta** $nHipotesisEspectrales$ **hacer**
  Cálculo de la aceleración, la velocidad y el desplazamiento espectral máximo, para cada modo de vibración
  Obtención de desplazamientos, velocidades y aceleraciones máximos en los nudos, para cada modo de vibración
  Obtención de solicitaciones en los extremos de las barras, para cada modo
  Cálculo de reacciones en apoyos, para cada modo de vibración
  Generación de esfuerzos y deformaciones en puntos intermedios de las barras, para cada modo de vibración
  Cálculo de esfuerzos en nudos de elementos finitos 2D, para cada modo
  Cálculo de deformaciones unitarias, esfuerzos y tensiones en nudos de elementos finitos, para cada modo de vibración
  si *grabar_result_modos_hipot*=1 **entonces**
    Escritura de los ficheros de resultados para cada modo de vibración
  fin si
  Combinación de los resultados de los diferentes modos de vibración mediante técnicas de combinación modal
  Determinación del signo de los resultados
fin para
Escritura de los ficheros de resultados

de cada modo los obtendremos según las expresiones (4.121), tras calcular previamente la aceleración espectral máxima de acuerdo a lo descrito en la sección 4.10.2 y tras combinar los valores de la aceleración espectral máxima obtenidos de cada uno de los ejes, en caso de ser diferentes, por medio de las técnicas de combinación direccional denominadas SRSS, CQC3 o CPAE que vimos en el apartado 4.10.3.

De forma idéntica a lo que ya describimos en el caso del análisis modal o el análisis por superposición modal, a continuación tendrá lugar una etapa de comunicación entre las tareas a fin de disponer de los desplazamientos en los nudos adyacentes de cada partición, definiendo a los grados de libertad de dichos nudos como filas ghost en el vector de desplazamientos de todos los grados de libertad de la estructura a reservar y completar.

Como podemos ver en el algoritmo 11, cada tarea calculará ahora en paralelo el resto de resultados para cada modo de vibración, los cuales podrán ser guardados en disco, a título individual para cada modo de vibración y para una determinada hipótesis espectral, siempre de manera opcional.

El siguiente paso es por tanto el de combinar la respuesta de los diferentes modos de vibración, recurriendo para ello a alguna de las técnicas de combinación descritas en el apartado 4.10.4. Todas las técnicas allí recogidas han sido implementadas y paralelizadas en esta tesis doctoral. Como ya dijimos, cada tipo de resultado se obtendrá, a título individual, gracias a la combinación de sus correspondientes respuestas modales, a excepción de los esfuerzos y las deformaciones en los puntos intermedios de las barras, los cuales se calcularán con posterioridad a partir de los resultados ya combinados de los desplazamientos en los nudos y los esfuerzos en los extremos de la barras. Debido a ello, cada tarea tendrá que conocer los desplazamientos combinados en sus nudos adyacentes para calcular dichos esfuerzos y deformaciones en los puntos intermedios de sus barras, lo cual tendrá lugar mediante las comunicaciones oportunas entre las diferentes tareas, como ya es bien sabido.

En gran medida, las distintas técnicas de combinación no están compuestas más que por diferentes operaciones vectoriales, las cuales se llevarán a cabo de forma colectiva entre las diferentes tareas, empleando para ello las funciones Ve-

cAbs, VecScale, VecAXPY, VecPointwiseMult, VecNorm y VecSqrtAbs de PETSc. De todas ellas, las que no hemos descrito hasta el momento son:

- VecAbs: Reemplaza cada elemento del vector por su valor absoluto.
- VecPointwiseMult: Calcula el producto elemento a elemento de dos vectores, almacenando el resultado en otro vector diferente.
- VecNorm: Obtiene la norma indicada de un vector. En nuestro caso nos será útil para calcular el máximo de los elementos de un vector en valor absoluto, lo cual es equivalente a obtener su infinito-norma.
- VecSqrtAbs: Sobreescribe cada elemento de un vector con la raíz cuadrada de su magnitud.

Únicamente el cálculo de la respuesta conjunta de todos los modos con altas frecuencias, en el método de la Respuesta Rígida Residual, requiere la resolución de un sistema de ecuaciones lineales donde aparece la matriz de rigidez como matriz de coeficientes, el cual resolveremos en paralelo mediante PETSc y *hypre* o mediante MUMPS, PaStiX o PARDISO, y donde el vector parte derecha se obtiene como un producto matriz por vector donde participa la matriz de masa, empleando en consecuencia la función MatMult.

Finalmente, la única etapa restante, antes de escribir los resultados combinados en los ficheros de disco, es la de estimar el signo de cada uno de los resultados, teniendo en cuenta que todos ellos estarán expresados en valor absoluto al haberse obtenido tras el cálculo de una raíz cuadrada. En ese sentido, cada tarea obtendrá en paralelo el signo de todo aquel conjunto de valores locales que ha calculado como combinación de los distintos resultados, aplicando algunas de las técnicas descritas en el apartado 4.10.5.

Así, una posibilidad es que el signo de los resultados combinados se corresponda con el signo que posea el modo de vibración que posea el mayor porcentaje de masa efectiva. Para ello, todas las tareas llevarán a cabo, de manera colectiva, un producto matriz por vector y diferentes productos escalares, usando las funciones MatMult y VecDot, calculando así el porcentaje de masa efectiva frente al total

indicado en la expresión (4.178), calculando a continuación el modo de vibración que da lugar a dicho porcentaje de masa efectiva máxima.

En cuanto a la implementación del método de Cominetti y Jorquera, cada tarea obtendrá en paralelo el signo de los valores del bloque de filas consecutivas de cada tipo de resultado que le han sido asignados, invocando para ello a las funciones VecAbs, VecScale, VecAXPY y VecPointwiseMult encargadas de llevar a cabo distintas operaciones vectoriales de forma colectiva entre todas la tareas.

Capítulo 8

Aplicación de las Tecnologías Grid y Cloud al Análisis Estructural

Como su propio nombre indica, este capítulo incluye la aplicación de las tecnologías Grid y Cloud al cálculo estático y dinámico de estructuras. Más en detalle, el capítulo describe los requerimientos considerados a la hora de desarrollar los servicios Grid y Cloud de análisis estructural, junto con la arquitectura que presenta cada uno de ellos y los diferentes elementos que los componen.

8.1. Diseño e implementación del servicio Grid de análisis estructural

8.1.1. Introducción

En este apartado mostraremos los diferentes componentes que forman parte del sistema Grid desarrollado, el cual ofrece a la comunidad estructural (estudios de arquitectura, empresas de ingeniería y construcción, universidades y centros de investigación) la posibilidad de analizar estructuras bajo demanda de forma

remota, tanto estática como dinámicamente, en una infraestructura Grid basada en Globus Toolkit.

A la hora de implementar dicho servicio remoto de análisis estructural, hemos tenido en cuenta los siguientes requerimientos:

- La definición de los parámetros de cálculo de una simulación debe ser idéntica tanto si el usuario desea calcular en su máquina local como si desea delegar dicho cálculo en el servicio Grid.
- Las simulaciones se ejecutarán en el sistema Grid de una forma totalmente transparente para el usuario.
- El sistema sólo debe estar accesible a aquellos usuarios autorizados y autenticados.
- El servicio Grid de análisis estructural debería estar basado en un modelo de alta productividad, donde múltiples clientes puedan utilizar simultáneamente el servicio y donde cada uno de ellos pueda simular el comportamiento de diferentes estructuras al mismo tiempo. No en vano, uno de los principales objetivos propuestos consistirá en calcular el mayor número de simulaciones por unidad de tiempo que sea posible, reduciendo así los tiempos de espera de los usuarios del servicio.
- Debe intentar amortiguarse la penalización que puede suponer analizar una estructura a nivel remoto. En el caso de llevar a cabo un análisis dinámico a lo largo del tiempo, mediante métodos de integración directa o por superposición modal, la considerable cantidad de resultados generados puede convertirse en un serio problema a la hora de ser recogidos por parte del cliente. Para intentar reducir dicha sobrecarga, las fases de simulación y recogida de datos deberían realizarse de forma simultánea, lo cual reduciría considerablemente los tiempos de espera.
- Debe proporcionarse un acceso privado a los datos. Únicamente el propietario de una simulación puede tener acceso a su estado, así como a los datos de entrada y salida de la misma.

- El usuario debe estar permanentemente informado del estado de sus simulaciones remotas.
- El sistema debe garantizar que todas las peticiones de análisis se atienden satisfactoriamente, teniendo en cuenta los posibles fallos que puedan ocurrir en los recursos de cómputo o en el recurso que alberga al propio servicio Grid.

Como ya describimos en la sección 6.3, Globus Toolkit (GT) incluye un conjunto de herramientas software que proporcionan una plataforma para el desarrollo de aplicaciones Grid. En su versión 4, este software incluye varios servicios de alto nivel, que satisfacen los requerimientos del estándar OGSA, como la monitorización de recursos, el descubrimiento de servicios, la gestión del envío de trabajos, el soporte de los sistemas de seguridad y los servicios para la gestión de datos, entre otros. Como resultado de la evolución natural de GT4 hacia los Servicios Web, surgieron los Servicios Grid, a modo de Servicios Web que aplicados al dominio Grid establecen un estándar en el descubrimiento y acceso a recursos Grid. Todo ello convirtió a GT4 en el estándar de facto empleado en la comunidad de usuarios Grid. Además, como vimos, estos servicios están implementados sobre el estándar WSRF, ya que GT4 incluye una implementación completa del mismo.

Partiendo por tanto del mencionado GT4, se ha desarrollado y desplegado un Servicio Grid de Análisis Estructural cuya arquitectura, recogida en la figura 8.1, está compuesta principalmente por estos 5 componentes: el Gestor del Servicio, el Comunicador del Estado, el Simulador Estructural, el Planificador de Trabajos y el Colector de Datos.

El Gestor del Servicio es el componente principal del sistema y se encarga de atender las peticiones de los clientes, así como de interconectar los diferentes elementos que componen el servicio. *El Planificador* es el elemento que asigna las simulaciones, también llamadas tareas, a los recursos computacionales que componen la infraestructura Grid. A partir de un Grid Computacional, este componente lleva a cabo la etapa de descubrimiento de recursos, para obtener una lista de máquinas candidatas a ejecutar la tarea, selecciona el recurso más adecuado de entre todas ellas, realiza la transferencia de los ficheros de entrada, ejecuta y monitoriza la tarea y recoge sus resultados. *El Comunicador del Estado* es el responsable de

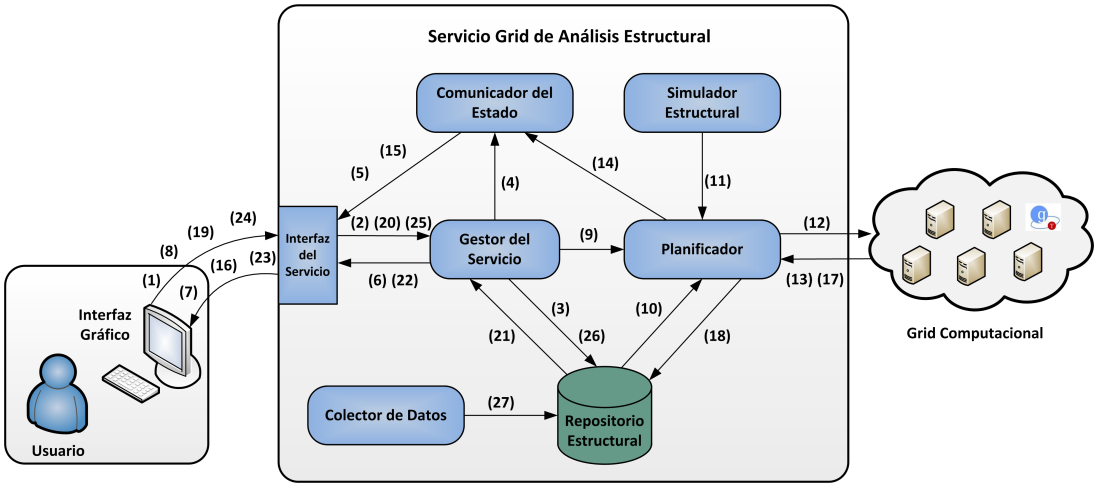


Figura 8.1: Arquitectura del Servicio Grid de Análisis Estructural.

informar al usuario de los cambios de estado en la ejecución de sus simulaciones, de modo que éste sepa en cada momento en qué estado se encuentran sus tareas (en espera, en ejecución, finalizada, fallida, etc.). *El Simulador Estructural*, descrito en el capítulo 7, es la herramienta de análisis basada en Computación de Altas Prestaciones desarrollada en el marco de esta tesis, la cual será ejecutada por el Planificador en el recurso Grid seleccionado, encargándose de realizar el análisis estático o dinámico de la estructura. Todos los datos de entrada y salida, correspondientes a cada simulación, se almacenan temporalmente en el *Repositorio Estructural*, ofreciendo un servicio adicional de almacenamiento de datos a los clientes y permitiendo, al mismo modo, la ejecución de nuevo de una tarea en caso de fallo. Finalmente, *el Colector de Datos* elimina todos aquellos ficheros de resultados que, una vez transcurrido un espacio de tiempo determinado desde su generación, permanecen todavía en el servicio, sin ser borrados por petición expresa de sus propietarios.

8.1.2. Gestión de las simulaciones

El proceso de análisis de una estructura conlleva una secuencia de pasos en los cuales intervienen e interaccionan todos los agentes integrantes del servicio. Este proceso se inicia cuando el cliente lanza una petición de cálculo (1), invocando al método *CalculateStructure*, enviando al servicio la geometría de la estructura a

analizar, sus propiedades, las hipótesis de carga y los parámetros que definen el tipo de cálculo, mediante un mensaje SOAP. Es importante destacar que cuando un cliente envía una petición de cálculo, éste no se queda bloqueado a la espera de su finalización. Las peticiones son atendidas por el *Gestor del Servicio* (2), el cual procesa los datos, genera y almacena (3), en el *Repositorio Estructural*, el fichero de entrada requerido por el *Simulador Estructural* y notifica al *Comunicador del Estado* (4) que se ha recibido una nueva simulación, de cuyos cambios de estado hay que informar al usuario. Para ello, dicho Comunicador del Estado creará un elemento de notificación (5), al que el cliente tendrá que suscribirse. A su vez, el Gestor del Servicio devuelve al cliente un identificador de la simulación, junto con la ruta del Repositorio Estructural donde se almacenarán los diferentes ficheros de entrada y salida de la simulación (6).

Por su parte, el cliente obtendrá la ruta en la cual se guardarán los archivos correspondientes a su simulación, junto con el identificador de la misma (7), el cual empleará en posteriores invocaciones al servicio con el objetivo de identificar de manera unívoca a dicha tarea. A continuación, el cliente se suscribirá al elemento de notificación asociado a su simulación, para estar informado de sus cambios de estado (8).

Una vez guardado el fichero de entrada, el Gestor del Servicio creará la tarea a ejecutar (9), la cual incluye una descripción de las características de la simulación que será ejecutada en el Grid Computacional mediante el Planificador. Entre las características que definen la tarea a ejecutar se encuentra el fichero ejecutable del Simulador Estructural, el fichero de entrada que dicha aplicación requiere para su ejecución, los ficheros de salida que se obtendrán al finalizar la simulación o, en caso de un análisis dinámico a lo largo del tiempo, los ficheros de resultados que se irán generado para cada instante. Inclusive, se puede especificar también una lista de requisitos hardware que debe cumplir un recurso computacional para poder realizar la ejecución de la tarea, tales como la memoria RAM o el número mínimo de elementos de procesamiento libres de los que debe disponer dicho recurso. Más aún, es posible limitar el número máximo de procesos encargados de ejecutar en paralelo una determinada tarea, lo cual resulta conveniente si la aplicación no escala convenientemente a partir de un número concreto de ellos.

Una vez definida la tarea, el Planificador se desplegará mediante un hilo indi-

vidual que velará por la correcta ejecución de dicha simulación, empleando para ello el API proporcionado por el meta-planificador GMarte, que describimos en la sección 6.4. Así, el Planificador realizará, gracias a dicho API y de manera totalmente transparente para el usuario, la lectura del fichero de entrada (10), la lectura del fichero ejecutable del Simulador Estructural (11), la selección del recurso más apropiado, la transferencia del fichero de entrada y del Simulador al recurso computacional seleccionado y la ejecución de la tarea mediante dicho Simulador Estructural (12). Se ha definido una política de selección de recursos basada en un modelo de alta eficiencia y productividad, el cual tiene en cuenta las características de cada simulación, tales como el tamaño de la estructura, el tipo de cálculo a llevar a cabo, los privilegios asociados al tipo de cliente, etc. para decidir en qué recurso se ejecuta el Simulador Estructural y con qué número de procesos MPI. De este modo, pretendemos aprovechar la distribución óptima de las tareas a los recursos Grid y el grado de eficiencia ofrecido por el Simulador Estructural, de modo que todo ello contribuya a mejorar las prestaciones del sistema global, en lugar de actuar en beneficio de la ejecución de una simulación a título individual.

A medida que la tarea progresa, el Planificador va monitorizando su estado (13), notificando al Comunicador del Estado cualquier cambio que tenga lugar (14), quien a su vez informará al usuario (15), lo cual le permite conocer el progreso de sus simulaciones en el Grid (16).

Dependiendo del tipo de análisis, el Comunicador del Estado enviará al usuario un tipo de notificación u otro. Así, tras la ejecución de un análisis estático o un análisis modal, el Planificador llevará a cabo la recogida de los resultados (17) y su posterior almacenamiento en el Repositorio Estructural (18). En ese momento, el Comunicador del Estado notifica al usuario que tiene todos los resultados disponibles y que puede proceder a recogerlos, puesto que ha finalizado la simulación. Por otro lado, en el caso de un análisis dinámico a lo largo del tiempo, se informará al usuario cuando el Planificador haya almacenado un número adecuado de resultados parciales en el Repositorio estructural, independientemente de que la ejecución haya o no finalizado.

En ambos casos, el cliente enviará en una o más ocasiones, dependiendo del tipo de análisis, la petición de recogida de resultados (19), indicando el identi-

ficador de la simulación y los pasos de tiempo de los cuales queremos recoger los resultados. Dicha petición será procesada por parte del Gestor del Servicio (20), quien lee los ficheros de resultados del Repositorio (21) y construye con ellos un fichero XML a modo de mensaje SOAP que enviará (22) y será recibido por el cliente (23). Como una alternativa a la que acabamos de describir, cabe la posibilidad de emplear el protocolo GridFTP ofrecido por GT4 para recoger los resultados. De este modo, el cliente accederá al espacio de almacenamiento de su simulación, dentro del Repositorio Estructural, y se descargará directamente los ficheros binarios de resultados.

Finalmente, el servicio publica un método de borrado, que debería ser empleado por los clientes una vez que hayan recogido los resultados de una simulación (24). Como consecuencia, el Gestor del Servicio atiende dicha petición (25) y elimina del Repositorio Estructural todos los datos referentes a dicha simulación (26). Sin embargo, puesto que no es obligatoria su invocación, se ha implementado un demonio, mediante un hilo de ejecución, llamado *Colector de Datos* que elimina automáticamente del repositorio los datos de todas aquellas simulaciones ya finalizadas, una vez excedido un tiempo acordado entre el cliente y el proveedor del Servicio (27). Dicho tiempo será diferente para cada usuario.

8.1.3. Comunicaciones entre el cliente y el servicio Grid

Uno de los principales inconvenientes asociados a la utilización de un arquitectura orientada a servicio es sin duda el que viene dado por las comunicaciones entre el cliente y servicio, sobre todo cuando éstas requieren el intercambio de un gran volumen de datos. Como es sabido, los servicios Web y en consecuencia los servicios Grid se basan en mensajes de texto para llevar a cabo el intercambio de datos con otras aplicaciones. Esta característica garantiza la comunicación independientemente de la plataforma, pero conlleva una notable sobrecarga en cuanto al tamaño de dichos mensajes, teniendo en cuenta que los ficheros binarios de resultados se convertirán en caracteres de texto. Dicho problema vendrá además agravado por el enorme volumen de datos generado por un análisis dinámico de estructuras de gran dimensión a lo largo del tiempo. Aunque lo ideal sería poder incorporar directamente los datos binarios dentro del fichero XML, de acuerdo al

protocolo WS-Attachments de los servicios Web, GT4 no permite trabajar con mensajes SOAP que incorporen ficheros binarios como adjuntos.

Para tratar de mitigarlo en lo posible, hemos empleado esquemas de codificación hexadecimal o en Base64 de datos binarios a cadenas de texto, a fin de introducir dichos datos embebidos en el fichero XML, de manera que se reduzca notablemente el tamaño de los mensajes en comparación a si los datos binarios se convirtieran directamente a caracteres de texto. La codificación hexadecimal transforma cada byte binario en dos caracteres, lo que incrementa en un 100 % el tamaño original de los datos. Sin embargo, la codificación y la decodificación es trivial. Por otro lado, en la codificación en Base64 ocurre que cada grupo de tres bytes binarios consecutivos se convierten en cuatro caracteres de texto, incrementando en consecuencia en un 33 % el tamaño original, con una codificación y decodificación más compleja que la anterior. Con todo ello, la codificación en Base64 será preferible a la hexadecimal, ya que el cuello de botella no lo tenemos en el tiempo empleado en la codificación y la decodificación de la información, sino en el tamaño de la misma.

Alternativamente, hemos implementado otra aproximación, la cual consiste en que las comunicaciones se realicen mediante GridFTP, donde los datos binarios pueden ser recogidos directamente desde el Repositorio Estructural por parte del cliente, de manera segura y eficiente. Para llevar a cabo esta segunda opción, se ha desarrollado una herramienta cliente GridFTP, que se ha incorporado al API del cliente, que proporciona una recogida de datos transparente por parte de los usuarios. A cambio, es necesario la instalación de algunas librerías de GT4 en la máquina cliente.

8.1.4. Gestión de la transferencia de los resultados

En función del tipo de análisis a realizar, la transferencia de los resultados se lleva a cabo de distinto modo tanto entre el servicio y el cliente como entre el recurso Grid y el servicio. En el caso de un análisis estático, el Planificador recogerá automáticamente todos los resultados una vez finalizada la ejecución de la tarea, almacenándolos en el Repositorio Estructural. A su vez, se informa al usuario de la disponibilidad de los mismos. De este modo, el usuario puede someter

una petición de recogida de datos, lo cual conlleva una acción de incorporar, bajo una codificación hexadecimal o en Base64, todos los ficheros de salida en un único mensaje de tipo SOAP o recogerlos vía GridFTP.

Por otro lado, en el caso de un análisis dinámico a lo largo del tiempo, debido principalmente al gran volumen de datos generado, se debe emplear un esfuerzo extra si queremos gestionar de manera óptima la recepción de los resultados. En lo que respecta a la transferencia de los mismos entre el recurso Grid y el servicio, se ha implementado un sistema de recogida periódica de resultados parciales generados por el Simulador Estructural para cada paso de tiempo. Mediante el uso de un sistema de sufijos en el nombre de los ficheros que distingue el instante de tiempo (t0, t1, t2, t3, etc.) al que corresponde cada uno de ellos y conociendo el intervalo de tiempo que transcurre en la generación de resultados entre dos instantes de tiempo consecutivos (dato este último aportado por la herramienta de cálculo), el Planificador configura periódicamente la recogida de datos, mediante el API de GMarte, solapándola de este modo con la ejecución de la simulación. Esto supone que, a medida que los resultados del análisis dinámico van siendo generados, estos se transfieren al servicio Grid sin necesidad de esperar a la finalización de la simulación, aproximación esta última que conllevaría un mayor tiempo de respuesta.

En cuanto a la transferencia de resultados entre el cliente y el servicio, se ha seguido la misma política de solapar la recogida de los resultados y el tiempo de ejecución de la simulación. De este modo, el Comunicador de Trabajos envía una notificación al cliente cada vez que se dispone de un conjunto determinado de resultados en el Repositorio Estructural. En consecuencia, el usuario puede comenzar a descargar y visualizar resultados parciales sin esperar a que su simulación haya finalizado, agilizándose de este modo la transferencia y reduciendo enormemente los tiempos de espera.

8.1.5. Planificación Grid

Como ya se ha explicado anteriormente, la ejecución de las simulaciones en los recursos computacionales disponibles se debe gestionar eficientemente por parte de un planificador Grid, el cual distribuya las tareas de los usuarios entre los

distintos recursos computacionales que componen la infraestructura. En nuestro caso, como ya hemos comentado, el servicio emplea como planificador a GMarte, el cual ofrece servicios de ejecución remota de tareas y meta-planificación eficiente sobre máquinas basadas en GT. En concreto, GMarte se encarga del descubrimiento de recursos, de la selección de los mismos, de la transferencia de los ficheros de entrada, de la ejecución de las simulaciones, de la monitorización del estado de los trabajos y de la recogida de los resultados, todo ello de manera transparente para el usuario. A continuación se describen dichas etapas con un mayor nivel de detalle:

- *Descubrimiento de Recursos*: Si el usuario no proporciona una lista de recursos computacionales, esta fase se encarga de obtener una lista potencial de máquinas en las que podamos realizar las ejecuciones, empleando para ello el servicio de GT denominado *Index Service*, que proporciona información agregada sobre los recursos computacionales de una organización.

- *Filtrado de Recursos*: Una vez que se ha obtenido una lista preliminar de máquinas, esta fase nos permite descartar todas aquellas que no sean accesibles al usuario por medio de sus credenciales o que no satisfacen ciertos requisitos computacionales necesitados por la aplicación que se pretende ejecutar. Este proceso da lugar a una lista definitiva de máquinas candidatas para la ejecución de las simulaciones.

  Es importante destacar que las dos fases previamente descritas sólo tienen lugar antes de comenzar el proceso de asignación de tareas y, por lo tanto, únicamente se ejecutan una vez para cada caso de estudio, compuesto este último por múltiples simulaciones independientes entre sí. Sin embargo, para cada una de las simulaciones que formen parte de dicho caso de estudio, será necesario llevar a cabo los pasos que a continuación se describen.

- *Selección de Recursos*: Partiendo del citado subconjunto de recursos computacionales, esta etapa se encarga de elegir una máquina que albergue la ejecución de la tarea que está pendiente de ser planificada. Dicho procedimiento de asignación o planificación conlleva una notable complejidad, donde es necesario puntuar a los recursos en base a los requisitos demandados por las aplicación a ejecutar. Este proceso involucra el desarrollo de

una función de rango que trata de cuantificar la idoneidad de cada máquina para la ejecución de una cierta tarea. En nuestro caso, la función de rango hace uso de un modelo de prestaciones que tiene en cuenta el coste de la transferencia de los datos desde el servicio a los recursos candidatos, así como un componente de distribución de carga que trata de evitar la asignación de todas las simulaciones a una única máquina. El modelo de prestaciones se basa además en las capacidades hardware de las máquinas remotas (número de núcleos computacionales y memoria RAM) de manera que máquinas más potentes reciben una mayor puntuación. Una vez que se ha seleccionado el recurso en el que se debe ejecutar la tarea, tienen lugar las restantes fases.

- *Transferencia de ficheros*: El Simulador Estructural paralelo y el fichero de entrada que describe la estructura y las cargas externas aplicadas se transfieren desde el servicio Grid a la máquina remota vía GridFTP.

- *Ejecución*: La aplicación comienza su ejecución en el recurso Grid seleccionado, tras ser previamente integrada en el gestor de trabajos de dicho recurso, siendo periódicamente monitorizada para conocer su estado o a fin de detectar si ha ocurrido algún error.

- *Transferencia de Resultados*: Esta fase se encarga de transferir al servicio Grid, mediante el protocolo GridFTP, todos los resultados del proceso de simulación generados en la máquina destino, borrando posteriormente los ficheros remotos.

Aunque podríamos haber usado otros planificadores, como HTCondor, GridWay o GridBus, nos hemos decantado por GMarte por su capacidad para recuperar los resultados ya generados de una simulación antes de que ésta finalice. Esta característica, que no está presente en el resto de planificadores, nos permite solapar la recogida de resultados con la ejecución de una simulación dinámica a lo largo del tiempo, reduciendo los tiempos de respuesta de los usuarios.

Por otro lado, GMarte está basado en técnicas multihilo, tanto en sus diferentes etapas como a nivel de cada una de ellas, lo que proporciona la implementación de sistema de alta productividad. Ello permite gestionar eficientemente y

de forma concurrente la ejecución remota de las diferentes tareas que se envíen simultáneamente al servicio, una situación que se produce con frecuencia si múltiples usuarios interaccionan con el servicio al mismo tiempo, tal y como deseamos. A modo de ejemplo, esto resulta de vital importancia en la fase de selección de recursos, la cual puede convertirse en un cuello de botella y donde, bajo esta aproximación, se obtienen importantes incrementos de velocidad en la asignación de la tareas.

Hay también que tener en cuenta que este planificador considera los requerimientos computacionales especificados por las tareas, así como la información dinámica del estado de los recursos (número de procesadores libres, memoria RAM disponible o nivel de carga de la CPU), accediendo para ello al servicio MDS de la máquina destino, para realizar la asignación de las mismas.

Otra característica a destacar de dicho planificador es el esquema de tolerancia a fallos multinivel ofrecido, dirigido a gestionar los errores ocurridos tanto durante la transferencia de ficheros como durante la ejecución de la tarea en la máquina remota. Esto garantiza que todas las simulaciones enviadas al servicio se ejecutarán satisfactoriamente siempre y cuando existan recursos computacionales Grid disponibles.

GMarte soporta ejecuciones en GT 2.4 y 4.0.2. Su funcionalidad está disponible como un API en Java, lo que permite integrar su funcionalidad como parte de un servicio Grid, así como en XML, permitiendo de este modo utilizarlo sin necesidad de emplear el lenguaje Java. Finalmente, cabe destacar la capa de abstracción que proporciona GMarte en el acceso a la información computacional ofrecida por los diferentes recursos heterogéneos. De esta manera, es posible consultar de un modo idéntico la información computacional de máquinas basadas en diferentes versiones de GT, a partir de los métodos de alto nivel ofrecidos para ser usados por el usuario.

8.1.6. Notificaciones

Las notificaciones son un patrón de diseño software que permite informar a los clientes de los cambios producidos en el servicio. De este modo, ya no es el

cliente el que consulta el estado de sus simulaciones, sino que es el servicio quien le informa, por medio de mensajes, de los cambios producidos en las mismas.

Cuando el cliente envía una petición de cálculo, se crea un nuevo elemento de notificación en el servicio referente a dicha simulación. Este elemento se publica, de manera que el cliente se suscribe a él, pasando a partir de ese momento a recibir las notificaciones referentes a los cambios de estado en el ciclo de vida de su petición (en cola, en ejecución, finalizada, fallida, etc.). De este modo, se informa al cliente inmediatamente de la finalización de su tarea, en caso de un análisis estático o modal, teniendo disponibles todos los resultados de cálculo, o cuando haya una cantidad considerable de los mismos que pueden ser accedidos, si se lleva a cabo un cálculo dinámico a lo largo del tiempo.

Conviene aclarar que si no empleáramos esta aproximación basada en notificaciones, sería el cliente el que periódicamente pediría al servicio que le informara del estado de sus simulaciones. Esta alternativa introducía una sobrecarga notable en el sistema, máxime en un servicio multiusuario que puede estar atendiendo simultáneamente diferentes peticiones de múltiples clientes.

8.1.7. Tolerancia a fallos

La tolerancia a fallos es uno de los aspectos más relevantes en un sistema que ofrece un servicio bajo demanda, en el cual el cliente exigirá las máximas garantías y donde se pretende ofrecer un alto nivel de calidad de servicio. Son varios los niveles de tolerancia a fallos implementados en el sistema, tanto a nivel del servicio Grid como a nivel de la planificación y la ejecución en la plataforma Grid.

En cuanto al servicio, se ha implementado un sistema de persistencia mediante un fichero XML en el cual el Gestor del Servicio almacena una descripción de las simulaciones activas, entre las que se encuentran las que están pendientes de ser ejecutadas, las que están ejecutándose y las que ya han finalizado pero aún tienen resultados en el Repositorio Estructural pendientes de ser recogidos por el cliente. Dicho fichero de persistencia se actualiza por parte del Gestor del Servicio o por el Planificador cuando:

- Llega una nueva tarea al servicio, creando un nuevo registro en el mismo.
- Comienza la ejecución de una tarea, modificando un registro ya existente.
- Ha finalizado la ejecución de la tarea, pero el cliente aún no ha recuperado todos sus datos de salida, modificando de nuevo el registro oportuno.
- El cliente ya ha recuperado todos los resultados, eliminando el registro pertinente.

De este modo, ante una caída del servicio, dicho fichero se procesará de nuevo y se volverán a lanzar todas las tareas que estaban en ejecución, además de procesar a todas aquellas que estaban en cola y a las que aún tienen datos pendientes de ser recogidos del servicio.

En cuanto a la ejecución de la tarea en el recurso Grid, el planificador GMarte es el que nos proporciona la tolerancia a fallos necesaria tal que, ante un fallo en la ejecución de una tarea, ésta sea migrada automáticamente a otro recurso, previa planificación de nuevo de la misma. En el caso de que se trate de un análisis dinámico, la ejecución se relanzaría desde el principio, y el usuario quedaría a la espera de la generación de los resultados que le restan por descargarse. En caso de fallo en la red, la comunicación se intenta hasta 3 veces, antes de notificar dicho problema al servicio Grid.

Con estos esquemas de tolerancia a fallos implementados, se garantiza que toda petición que llegue al servicio será satisfactoriamente atendida, incluso ante caídas del propio servicio.

8.1.8. Seguridad en el servicio Grid

La seguridad es un aspecto fundamental a tener en cuenta en un sistema con el que van a interactuar multitud de usuarios. Los mecanismos de seguridad del Servicio Grid de Análisis Estructural puestos en práctica emplean la capa de seguridad llamada GSI (Grid Security Infrastructure) de GT4, cubriendo aspectos como la autorización y la autenticación del usuario, además de la privacidad y la integridad de los datos.

Por un lado, es necesaria la implantación de un sistema de autorización y autenticación que regule el acceso a los servicios publicados y permita saber qué usuario está realizando cada acción en cada momento. Para ello, basándonos en el modelo de autorización implementado por medio de los ficheros Gridmap de GSI, se emplea un fichero de configuración en el cual aparece un listado de todos los usuarios autorizados para interactuar con el servicio. La autenticación se lleva a cabo por medio de un certificado X.509, que identifica inequívocamente al usuario y que éste envía al servicio al iniciar la comunicación. El nombre del cliente se extraerá de dicho certificado, el cual se contrasta con los nombres presentes en el fichero de usuarios autorizados para poder acceder al sistema. Por otro lado, conviene destacar que el servicio se ha implementado catalogando a los usuarios en diferentes niveles de privilegios, determinando así las acciones que puede realizar cada uno de ellos.

La privacidad e integridad de los datos se consigue utilizando sistemas de clave pública y privada, que utilizan el mismo certificado X.509 del usuario para llevar a cabo el cifrado y la firma de los datos que viajan entre el servicio y el cliente.

Conviene además recordar que, a la hora de ejecutar una tarea remota en un recurso gestionado por Globus, son necesarias las credenciales del usuario. En nuestro caso, hemos generado un único certificado desde el servicio Grid, que se suministra a GMarte, con los privilegios necesarios para ejecutar las simulaciones en los recursos computacionales. Esta aproximación facilita la incorporación de nuevos usuarios al servicio, ya que evita la modificación de los esquemas de seguridad de los recursos Grid, incorporando a dichos nuevos usuarios. De este modo se supone que todos los usuarios autorizados a usar el servicio podrán ejecutar sus simulaciones en la infraestructura Grid, sin necesidad de disponer de un permiso explícito para ello.

8.1.9. Monitorización del Servicio Grid de Análisis Estructural

En un entorno dinámico como es el de una infraestructura Grid, resulta esencial ofrecer mecanismos que informen de las características de los recursos que

componen la infraestructura y del estado de los mismos. Dicha información presenta una relevancia destacable, puesto que es empleada por ejemplo para llevar a cabo la planificación y la ejecución de las tareas. Sin embargo, ocurre que, en una plataforma orientada a servicio, el propio servicio puede ser una importante fuente de información, principalmente si está integrado en una infraestructura multiservicio. Si ese es el caso, necesitamos conocer el estado de cada uno de los servicios que componen la infraestructura del servicio global, con el objetivo de distribuir apropiadamente las simulaciones, enviadas directamente por parte de un cliente o por un planificador de servicio de más alto nivel.

En consecuencia, se ha incorporado el servicio MDS (Monitoring and Discovery Services) de GT con el objetivo, por un lado, de ofrecer información sobre el estado del Servicio Grid de Análisis Estructural y, por otro, de habilitar el desarrollo de una infraestructura de un nivel superior que estuviera compuesta por múltiples Servicios Grid de Análisis Estructural.

El mecanismo de publicación vía MDS del Servicio Grid de Análisis Estructural está basado en un conjunto de ficheros log, generados por dicho servicio, que registran todos los eventos ocurridos, junto con un proveedor MDS encargado de transformar la información almacenada en dichos ficheros al formato de texto definido en el esquema MDS. Como consecuencia, una vez que dicho proveedor ha sido incorporado y activado en el servicio, el proceso de publicación de la información se llevará a cabo bajo demanda y de forma transparente. Dentro de la información proporcionada por el servicio Grid se incluye el número de usuarios conectados, el número de simulaciones activas y el número de nodos disponibles en la infraestructura Grid.

8.1.10. Métodos implementados para interaccionar con el servicio

Gracias a la flexibilidad ofrecida por los servicios Web, es posible recoger en diferentes métodos la funcionalidad del Servicio Grid de Análisis Estructural. Todos ellos se publican por medio del protocolo WSDL, el cual permite al cliente conocer el modo de invocarlos. Posteriormente, cada uno de esos métodos se invo-

ca mediante peticiones SOAP, quien establece además el formato de los mensajes entre el cliente y el servidor, los cuales se transportan usando el protocolo HTTP. A continuación se recogen los diferentes métodos implementados y publicados por el servicio:

- *CalculateStructure*: Se trata del primer método que debe invocar el cliente para simular una estructura en la plataforma Grid. Como parte del mensaje SOAP, este método debe recibir la geometría y las propiedades de la estructura, además de la cargas externas aplicadas y los parámetros que definen la simulación, los cuales se corresponden directamente con los requeridos por el Simulador Estructural.

  A continuación, la estructura a simular se registra en la lista de tareas del servicio, almacenada a modo de un fichero XML de persistencia que será leído por el Planificador para llevar a cabo su ejecución en la infraestructura Grid. Adicionalmente, el método crea un elemento de notificación para cada simulación recibida, el cual se gestionará mediante el método *Subscribe* que describiremos más adelante. Finalmente, el método devuelve al cliente un identificador de tarea único, que será usado en futuras invocaciones al servicio para identificar la simulación.

- *CancelCalculation*: Este método se encarga de cancelar la ejecución de una simulación, cuyo identificador se proporciona como dato de entrada, en el recurso Grid, empleando para ello el API ofrecido por el Planificador. El método borra además todos los datos correspondientes a dicha simulación almacenados en el Repositorio Estructural. Si la ejecución no había comenzado todavía, se elimina del fichero de persistencia que guarda las tareas pendientes de ejecución.

- *GetCalculationStatus*: Cualquier tarea enviada al servicio Grid pasa por un conjunto de estados durante su ciclo de vida. Entre dichos estados tenemos los de enviada, en cola, en ejecución, completada, cancelada, fallida y pendiente de recibir resultados. En cualquier momento, los clientes pueden obtener el estado de una tarea invocando al método GetCalculationStatus, pasando como dato de entrada el identificador de la misma.

Aunque la invocación a dicho método por parte del cliente nos proporcionaría el estado de una simulación de forma satisfactoria, conviene aclarar que es preferible emplear la aproximación basada en notificaciones, la cual introduce mucho menor tráfico en la red y mucha menor sobrecarga en el servicio que la debida a las múltiples consultas que pueden realizar los distintos usuarios del sistema para conocer el estado de sus tareas.

- *Subscribe*: Este método permite que un cliente se suscriba a un servicio de notificaciones que le informará periódicamente del estado de sus tareas. Como hemos comentado, esta alternativa resulta ser mucho más apropiada que la anterior, a fin de no colapsar el servicio con multitud de peticiones y reducir considerablemente el número de mensajes intercambiados entre él y los clientes.
- *DeleteData*: Se trata del método que borra del Repositorio Estructural todos los datos de entrada y salida de aquella simulación cuyo identificador se proporciona.
- *GetResults*: Consiste en el método al que deben invocar los clientes, una o más veces, para recoger los resultados de una simulación. Para ello, el usuario debe indicar el identificador de dicha simulación y los instantes de tiempo de los que desea recuperar sus datos de salida.
- *GetServiceInformation*: Se trata del método que proporciona información relevante del servicio, como el número de simulaciones activas, el número de usuarios propietarios de dichas simulaciones activas o el número de recursos computacionales disponibles en la infraestructura.

8.1.11. El cliente gráfico

El Servicio Grid de Análisis Estructural debe ir acompañado de aplicaciones cliente, instaladas sobre PCs convencionales, que le envíen peticiones de cálculo y recojan los resultados de las mismas. Para ello, se desarrolló una aplicación (ver figura 8.2) con un avanzado interfaz gráfico de usuario que le asiste en la etapa de entrada de datos, o preproceso, en la definición de los parámetros de cálculo y en la interpretación de los resultados de salida, o postproceso.

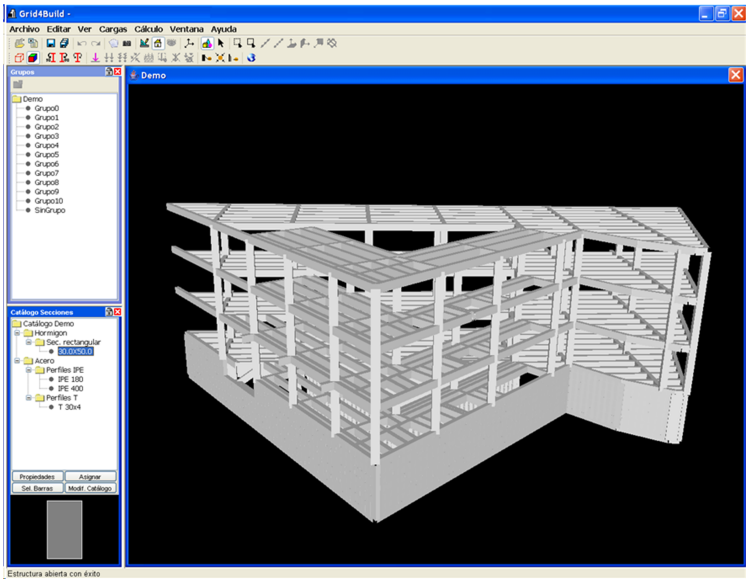


Figura 8.2: Cliente gráfico que interacciona con el Servicio Grid de Análisis Estructural.

La aplicación gráfica analiza las estructuras por medio del Servicio Grid desarrollado, interaccionando a través de su interfaz pública. Al sistema compuesto por dicha aplicación gráfica y el servicio Grid implementado se le denominó Grid4Build. De este modo ocurre que, ante una estructura a analizar, el cliente envía al servicio las propiedades de dicha estructura, así como los diferentes parámetros asociados al cálculo, mediante un mensaje SOAP. Seguidamente, el cliente se suscribe a las notificaciones de su simulación y comienza a estar informado del progreso de la misma. Finalmente, cuando se le comunica que ya hay resultados disponibles, estos se recogen por medio de otro mensaje SOAP o mediante GridFTP, pudiendo ser borrados del servicio a voluntad del cliente. A partir de estos datos comienza la etapa de postproceso, siendo a su vez representados gráficamente en el interfaz de modo que puedan ser fácilmente interpretados.

La tolerancia a fallos incluida en el cliente es otro aspecto a destacar, aprovechando el desacoplamiento existente con la etapa de análisis de la estructura. Esto garantiza que si el cliente cierra la aplicación o ésta incluso falla mientras se está llevando a cabo remotamente el análisis, no se producirá ni la cancelación de la simulación ni la pérdida de los resultados, que podrán ser recuperados posteriormente gracias al uso del identificador del trabajo, suscribiéndose además a

las notificaciones.

En cuanto al análisis dinámico, debido a la existencia de múltiples ficheros de resultados para los diferentes instantes de tiempo, el cliente implementa un esquema de persistencia en el que registra automáticamente el identificador del último paso de tiempo descargado, permitiéndose fácilmente la reanudación de la descarga justo en el punto en el que se quedó con anterioridad. De este modo, se optimizan al máximo las transferencias de resultados entre el cliente y el servicio, garantizándose que todo resultado recibido y almacenado por el cliente no vuelve a ser recuperado.

En lo que respecta a los fallos que puedan existir en la red durante la transferencia de datos, el cliente llevará a cabo una serie de intentos hasta que la conexión con el servicio se dé por perdida. Con posterioridad, el cliente volverá a solicitar, en cualquier momento, la comunicación con el servicio, a fin de recoger los resultados pendientes de su simulación.

8.2. Diseño e implementación de los servicios Cloud de análisis estructural

8.2.1. Introducción

A los requerimientos ya comentados que debía satisfacer el Servicio Grid de Análisis Estructural conviene añadir los siguientes, en lo que respecta al desarrollo y despliegue de diferentes Servicios Cloud de Análisis Estructural:

- El servicio debería incrementar y decrementar automáticamente el número de máquinas virtuales utilizadas en el cálculo, en función de la carga del sistema.
- La infraestructura computacional desplegada en la nube debería ser multiplataforma, considerando diferentes sistemas operativos.
- Las simulaciones estáticas no deberían verse penalizadas en su ejecución en

el servicio Cloud por las simulaciones dinámicas, las cuales conllevan unos tiempos superiores de simulación.

Como ya vimos en la sección 6.7, VENUS-C es una plataforma Cloud compuesta por varios componentes que proporcionan diferentes servicios. Entre los esenciales se encuentran la gestión de los datos y la gestión de los trabajos. El primero de ellos incluye la transferencia de los datos de entrada al Cloud, así como la recuperación de los resultados, ambas llevadas a cabo mediante la especificación CDMI. El segundo en cambio permite que los usuarios puedan reservar recursos computacionales en la nube para su uso particular, enviar tareas para su ejecución y gestionar su ciclo de vida, es decir, monitorizar el estado de su ejecución y cancelar la ejecución del trabajo, si es necesario.

Los dos componentes principales que implementan el servicio de gestión de los trabajos enviados al Cloud son el Generic Worker y PMES COMPSs, respectivamente desarrollados por Microsoft y por el BSC, los cuales presentan un modelo de programación diferente. Mientras que el primero está implementado para ser desplegado en la plataforma de Microsoft Azure, ejecutando máquinas virtuales basadas en Windows Server 2008 bajo .NET, el segundo se despliega sobre una plataforma de máquinas virtuales Linux gestionadas por EMOTIVE y OpenNebula, sobre recursos computacionales del BSC, de la universidad KTH y de la empresa Engineering.

Las simulaciones en la nube se gestionarán mediante Architrave [335], un sistema avanzado de diseño, cálculo y visualización de estructuras de edificación e ingeniería civil desarrollado gracias a la colaboración del DMMCTE de la UPV y el GRyCAP. Architrave está compuesto por estos tres componentes:

- *Architrave Diseño*: Es una aplicación, basada en AUTOCAD, en la que el usuario dibuja el modelo y define las propiedades de los elementos estructurales y las cargas externas aplicadas.
- *Architrave Cálculo*: Se trata de una aplicación con un interfaz gráfico, mostrada en la figura 8.3, donde el usuario puede modificar las propiedades estructurales y las cargas externas, además de analizar la estructura y visualizar los resultados.

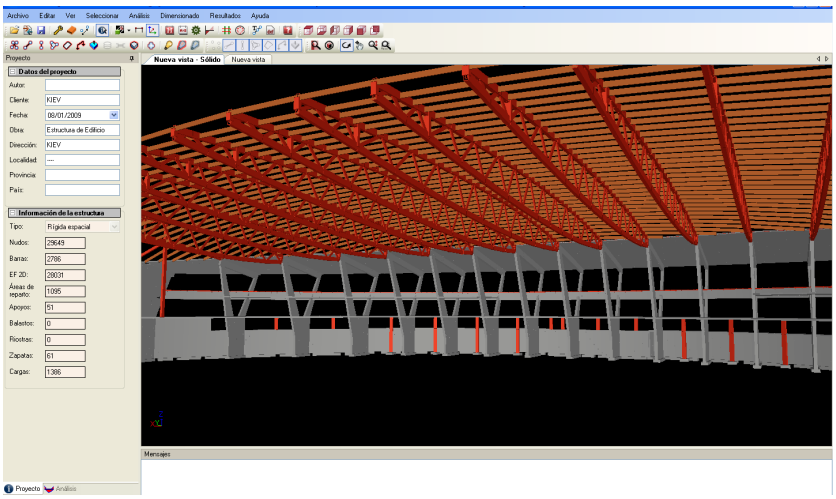


Figura 8.3: Cliente gráfico que interacciona con el Servicio Cloud de Análisis Estructural.

- *El Simulador Estructural*: Consiste en la aplicación de cálculo estructural basada en técnicas de Computación de Altas Prestaciones desarrollada en el marco de esta tesis, capaz de simular estática y dinámicamente la respuesta de una estructura mediante el método de los elementos finitos.

A la hora de simular una estructura de forma remota, será necesario configurar algunas opciones que determinan la cantidad de resultados a almacenar en el servicio de almacenamiento del que dispongamos en la nube, así como su posterior transferencia a la máquina local del usuario y el modo de llevarla a cabo. En ese sentido, se ha incorporado a Architrave Cálculo el cuadro de diálogo mostrado en la figura 8.4. Como podemos apreciar en la imagen, el usuario debe proporcionar un nombre a la simulación e indicar la plataforma Cloud en la que vamos a analizar la estructura, bien sea la proporcionada por Microsoft Azure, gestionada mediante Generic Worker, o la del BSC, KTH y Engineering, gestionada por PMES COMPSs.

En cuanto a los resultados a almacenar en la nube, podremos elegir entre guardar todos los resultados de la simulación, la cual representa la opción elegida por defecto para los diferentes tipos de cálculo, o sólo los tres instantes de tiempo más desfavorables, que sólo podremos seleccionar en el caso de realizar un análisis dinámico a lo largo del tiempo. Conviene aclarar que el criterio que hemos escogido

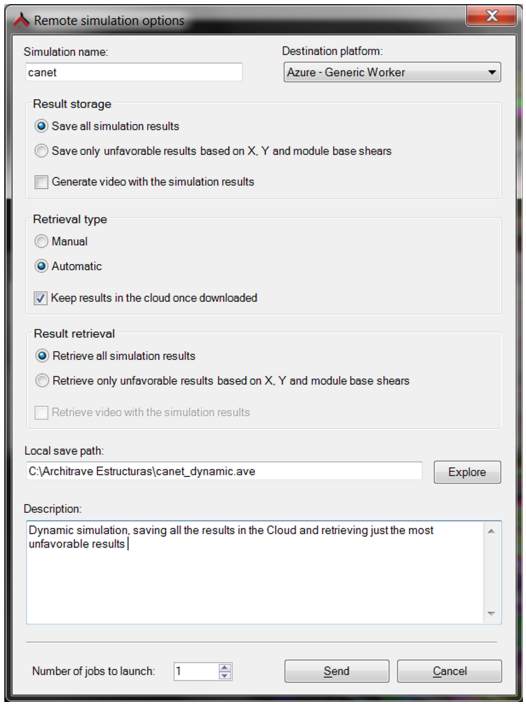


Figura 8.4: Definición de una simulación remota.

para determinar dichos instantes más desfavorables es el de los *cortantes en la base*, obteniendo dicho valor en los ejes X e Y globales de la estructura, junto con el valor del módulo en ambos ejes. Dichos cortantes en la base representan una estimación de la fuerza lateral máxima que tiene lugar en la base de la estructura debida a un movimiento sísmico, proporcionándonos una estimación de la bondad de una estructura ante la acción de un terremoto. Se facilita además la posibilidad de generar y almacenar un vídeo que recoge la respuesta de la estructura a lo largo del tiempo.

Por otro lado, el usuario puede indicar si desea descargar los resultados automáticamente, una vez que estén disponibles y sin necesidad de ninguna interacción por su parte, o de forma manual, llevándolo a cabo más adelante y siempre tras petición expresa de él mismo. Una vez descargados dichos datos a la máquina local, el usuario configura además si desea que sean automáticamente borrados del Cloud o no.

Por último, el usuario debe seleccionar qué tipo de resultados desea descargar, bien sean todos los de la simulación, sólo los más desfavorables e incluso, si lo

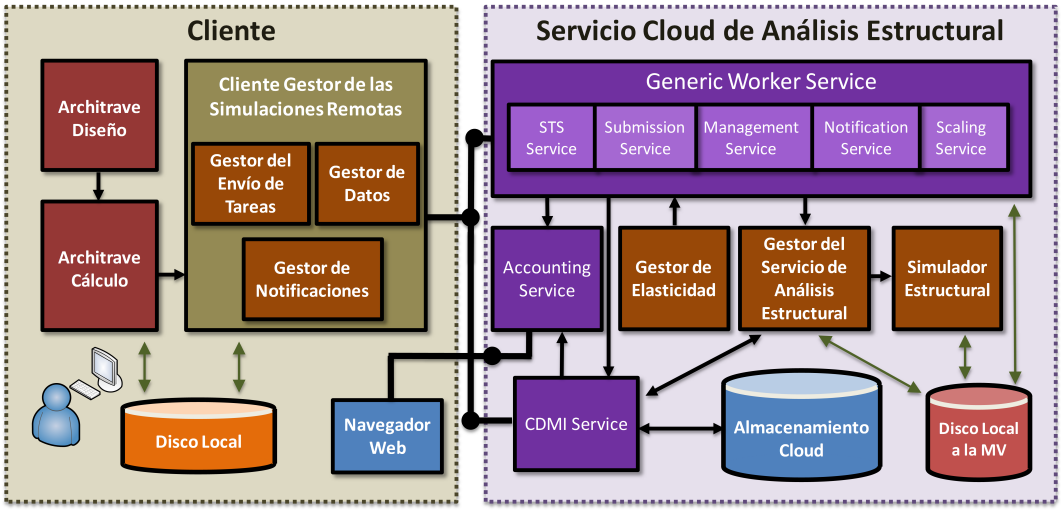


Figura 8.5: Arquitectura del sistema Cloud de Análisis Estructural Basado en Generic Worker.

hubiera generado previamente, un vídeo con los resultados de la simulación a lo largo del tiempo.

A continuación procedemos a explicar el diseño adoptado en caso de emplear Generic Worker o PMES COMPSs para ofrecer un análisis estructural a demanda en una plataforma basada en computación en la nube.

8.2.2. Desarrollos basados en Generic Worker

8.2.2.1. Introducción

La arquitectura del sistema Cloud desarrollado basado en Generic Worker es la que podemos observar en la figura 8.5, compuesto principalmente por una aplicación cliente, ejecutada en la máquina local del usuario, y un Servicio Cloud de Análisis Estructural, desplegado en la nube.

Tras crear la estructura en *Architrave Diseño*, el usuario ejecuta la aplicación *Architrave Cálculo* desde la cual, además de modificar las propiedades estructurales o las cargas aplicadas, define los parámetros de análisis de la simulación, configura las características de la simulación remota y visualiza los resultados. Adicionalmente, se ha desarrollado una herramienta denominada el *Cliente Ges-*

tor de la Simulación Remota, incorporada a Architrave Cálculo, formada por diferentes componentes y encargada de enviar y gestionar las simulaciones en la nube y recoger posteriormente los resultados. Toda comunicación entre el cliente local y el servicio Cloud se llevará a cabo mediante el servicio CDMI.

Por su parte, el servicio Cloud atenderá satisfactoriamente las diferentes peticiones de cálculo, informando al usuario del estado de sus simulaciones. Para ello, estará formado por un conjunto de elementos, algunos de ellos pertenecientes al *Generic Worker* y otros implementados por nuestra parte, tales como el *Gestor del Servicio de Análisis Estructural*, el *Simulador Estructural* o el *Gestor de la Elasticidad*.

En las siguientes subsecciones proporcionaremos un explicación más detallada de los citados componentes.

8.2.2.2. El Cliente Gestor de las Simulaciones Remotas

En la parte cliente, las simulaciones estáticas o dinámicas a ejecutar a nivel local o remoto se definen exactamente del mismo modo, por medio de Architrave Cálculo, configurando la simulación remota mediante el cuadro de diálogo de la figura 8.4. Para poder enviar y gestionar las simulaciones en la nube, conocer su estado en tiempo real y descargar los resultados, se ha desarrollado una herramienta en .NET a la que hemos denominado el Cliente Gestor de la Simulación Remota, cuyo interfaz gráfico podemos ver en la figura 8.6. Esta nueva aplicación se lanza desde Architrave Cálculo, al cual se ha incorporado como un ejecutable independiente que puede continuar ejecutándose a pesar de que Architrave Cálculo se haya cerrado. Esta herramienta está a su vez formada por los siguientes componentes:

- *El Gestor del Envío de Tareas*: Para cada simulación a analizar de forma remota, este componente genera un documento JSDL que envía al servicio Cloud desarrollado. Se trata en realidad de un fichero XML que incorpora:
  - El nombre del trabajo y su identificador.
  - Una referencia al blob del servicio de almacenamiento donde se encuentra la aplicación a ejecutar en la nube y sus dependencias, junto

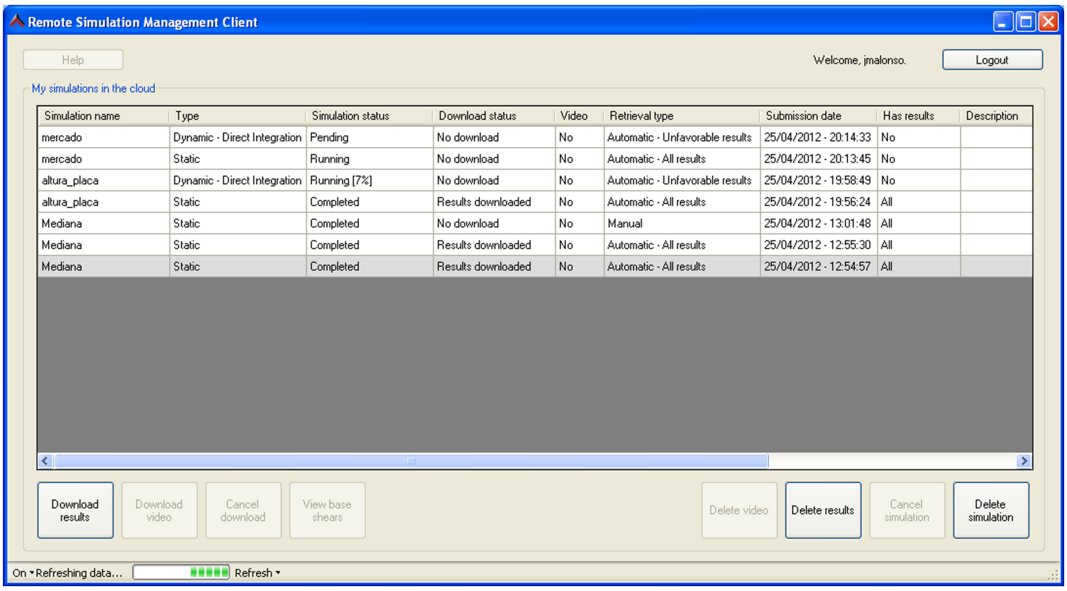


Figura 8.6: Aspecto gráfico del Cliente Gestor de las Simulaciones Remotas.

con la plantilla que describe el modo de invocar a la aplicación y los parámetros que la acompañan.

- El valor de los parámetros empleados para invocar a la aplicación, indicando las referencias al servicio Cloud de almacenamiento donde se almacenarán los ficheros de entrada y salida de la aplicación. Dichos ficheros de entrada deberán ser descargados de dicho servicio antes de que la aplicación se ponga en marcha. Con posterioridad, los ficheros de salida se enviarán también al servicio de almacenamiento del Cloud.

Este módulo proporciona además la posibilidad de poder consultar el estado de una tarea y de cancelar la ejecución de una simulación previamente enviada al Cloud.

- *El Gestor de Datos*: Este elemento es el encargado de enviar los datos de entrada de una simulación y descargar los resultados de salida, usando para ello o bien el servicio CDMI o el servicio de almacenamiento de Microsoft Azure (Microsoft Azure Storage service). Para el primero de los casos, se ha desarrollado un módulo cliente que interacciona con el servicio CDMI con el objetivo de enviar y recibir ficheros entre el cliente y el servicio Cloud. Dicho módulo cumple con el interfaz REST de CDMI. Para el segundo de

los casos, se utilizan los métodos proporcionados por Microsoft Azure para transferir o recoger datos de su servicio de almacenamiento.

- *El Gestor de Notificaciones*: De forma periódica, este módulo revisa el estado de todas las simulaciones enviadas al servicio Cloud (enviada, pendiente, en ejecución, finalizada, fallida, cancelada, etc.), informando en consecuencia al usuario. Adicionalmente, le informa del progreso en la ejecución de sus tareas remotas. Para tener conocimiento del estado de una simulación, el Gestor debe suscribirse a un sistema de notificaciones, en el cual se le informa del estado de la misma. A nivel de implementación, el servicio Cloud trabajará en realidad con dos colas de mensajes, en las cuales va almacenando el cambio de estado de una tarea y el progreso en su ejecución. Periódicamente, este componente consultará los mensajes existentes en dichas colas, informando en consecuencia al usuario.

Cada vez que el Cliente Gestor de las Simulaciones Remotas se pone en marcha, el usuario debe proporcionar su nombre de usuario y su contraseña, a fin de identificarse y autenticarse en el servicio Cloud. Si el usuario tiene permiso para acceder al mismo, se le mostrará la información correspondiente a sus simulaciones remotas, pudiendo además llevar a cabo diferentes acciones, entre las que se encuentran las de enviar nuevos trabajos de simulación, descargar resultados, cancelar la ejecución de una tarea, cancelar una descarga de resultados, eliminar los resultados en el servicio de almacenamiento o ver información relativa a los cortantes en la base para los diferentes instantes de tiempo, en caso de una simulación dinámica a lo largo del tiempo, con el objetivo de que el usuario pueda recuperar de manera selectiva todos aquellos que considere significativos, como muestra la figura 8.7.

Entre la información que se proporciona al usuario de cada simulación se encuentra el nombre de la misma y las características estructurales de la estructura a analizar, el tipo de cálculo a realizar y sus parámetros asociados, el estado de la simulación y su progreso, en forma de porcentaje, en caso de estar ejecutándose, el tipo de datos de salida a recuperar (todos, sólo los más desfavorables e incluso un vídeo de la simulación), el tipo de resultados ya disponibles en el servicio, el modo (manual o automático) de recuperarlos, el estado del proceso de recuperación de los resultados a la máquina local del usuario (pendiente, descargados, en

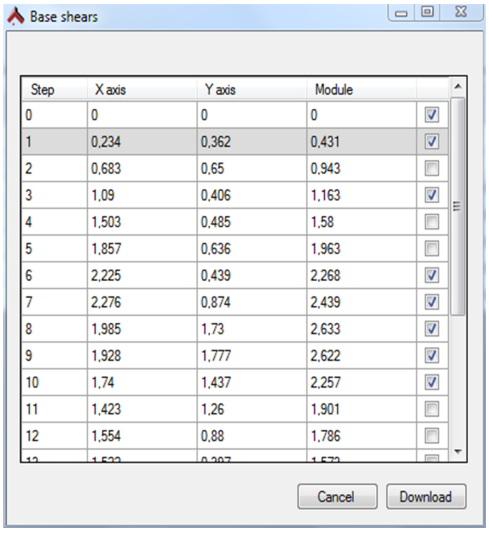


Figura 8.7: Selección de los pasos de tiempo a ser descargados de acuerdo a los cortantes en la base.

proceso de descarga), la fecha y la hora del envío de la simulación a la nube y una descripción asociada a la simulación remota que haya proporcionado el usuario.

8.2.2.3. El Servicio Cloud de Análisis Estructural

En la parte Cloud, el Servicio de Análisis Estructural implementado está basado en los siguientes componentes, de los cuales hemos desarrollado el *Gestor del Servicio de Análisis Estructural*, el *Simulador Estructural* y el *Gestor de la Elasticidad*:

- *Generic Worker*: Como ya comentamos, se trata de un servicio de tipo web role que gestiona la ejecución de las tareas en la infraestructura de Microsoft Azure. Dicho servicio está formado a su vez por este otro conjunto de elementos, algunos de los cuales interaccionan entre sí:
  - *Security Token Service (STS)*: Implementa la autorización y autenticación de los clientes que quieren acceder al servicio, mediante su nombre de usuario y su contraseña.
  - *Job Submission Service*: Se encarga de atender y registrar las nuevas peticiones de ejecución de tareas que llegan al servicio Cloud.

  - *Job Management*: Gestiona la ejecución de las tareas y proporciona información relativa a su estado.
  - *Scaling Service*: Aumenta o reduce dinámicamente el número de máquinas virtuales encargadas de ejecutar las simulaciones o, lo que es lo mismo, el número de instancias del propio Generic Worker.
  - *Notification Service*: Se emplea para registrar el progreso en la ejecución de las simulaciones, si procede, junto con sus cambios de estado.

- *Gestor del Servicio de Análisis Estructural*: Consiste en el componente encargado de lanzar, con los parámetros apropiados y dependiendo del tipo de cálculo, al *Simulador Estructural*. Para ello, habrá generado previamente el fichero de entrada necesitado en la ejecución del Simulador.

  Este módulo también se encarga de que la simulación remota tenga lugar de acuerdo a la definición establecida por parte del usuario. En ese sentido, en el caso de un análisis estático, modal o modal espectral, guardará los resultados de cálculo en el servicio de almacenamiento Cloud una vez que la simulación haya finalizado, tras agrupar en un mismo archivo los diferentes ficheros de salida generados por el Simulador. Sin embargo, en el caso de un análisis dinámico a lo largo del tiempo, mediante métodos de integración directa o por superposición modal, caben dos posibilidades. Si el usuario ha escogido almacenar todos los resultados de tiempo ocurrirá que, a medida que el Simulador Estructural genere los resultados para cada instante de tiempo, el Gestor de la Simulación los agrupará y los almacenará en el servicio Cloud. Con ello se consigue solapar la simulación con la subida de estos datos, lo cual permite que el usuario comience a descargar dichos ficheros de salida antes de que concluya la ejecución de la simulación. Sin embargo, si el usuario sólo desea almacenar los resultados de los instantes más desfavorables, el Gestor identificará de qué instantes se trata, una vez finalizada la simulación, compactará sus resultados y los almacenará en el servicio de almacenamiento. Asimismo, si procede, generará un vídeo con la respuesta de la estructura que también moverá al servicio de almacenamiento.

- *Simulador Estructural*: Se trata de la aplicación de cálculo desarrollada en esta tesis doctoral, basada en técnicas de Computación de Altas Prestaciones, encargada de obtener el comportamiento de la estructura ante las

cargas externas aplicadas. Para poder ejecutarse en la plataforma computacional de Microsoft Azure, la aplicación debe compilarse, junto con las diferentes librerías numéricas de las que depende, en Microsoft Windows.

- *Gestor de la Elasticidad*: Este componente se encarga de implementar la elasticidad del servicio Cloud, ajustando automáticamente su número de recursos computacionales en función de la carga del sistema y de acuerdo a la política que establezcamos. Para aumentar o reducir el número de instancias en el sistema, se emplea el *Scaling Service* del Generic Worker.

  Este módulo, que se ejecuta en una instancia independiente al resto, monitoriza la carga del sistema cada cierto periodo de tiempo que nosotros establecemos como apropiado, decidiendo como consecuencia la acción a llevar a cabo. Dicho intervalo de tiempo no deberá ser inferior al tiempo de despliegue de una nueva máquina virtual, puesto que entonces no se vería reflejado un posible aumento del número de instancias en una acción anterior, pero tampoco muy elevado, ya que no respondería en tiempo real ante una variación del número de simulaciones en el sistema.

  Adicionalmente, debemos considerar un número mínimo (MIN_MV) y máximo (MAX_MV) de máquinas virtuales que pueden formar parte del despliegue Cloud, junto con un umbral inferior (UMBRAL_INF) y un umbral superior (UMBRAL_SUP) de máquinas que estarán, en un momento dado, sin ejecutar ninguna simulación. Si el número de máquinas ociosas, en un momento dado es menor o igual UMBRAL_INF, se entiende que la carga de procesamiento del sistema será elevada y se deberán desplegar una o más instancias automáticamente (siempre que no se supere, claro está, el número máximo MAX_MV permitido), con el objetivo de garantizar que las tareas sufran el menor retraso posible en su ejecución remota, como medida de calidad de servicio, manteniendo la productividad del sistema. Si por el contrario el número de instancias ociosas es mayor o igual a UMBRAL_SUP, se entiende que tenemos poca carga en el servicio o, lo que es lo mismo, un número notable de máquinas ociosas, siendo conveniente deshacernos de alguna de ellas, teniendo en cuenta que no podemos tener menos máquinas que el valor indicado en MIN_MV.

  El inconveniente principal que presenta Microsoft Azure, y en consecuencia

el Scaling Service, a la hora de liberar una máquina virtual es que no permite que sea el Gestor de Elasticidad quien especifique qué instancia se desea apagar. Es precisamente el propio Microsoft Azure quien se encarga automáticamente de decidirlo, sin tener en cuenta el hecho de que la máquina virtual escogida podrá estar ejecutando una simulación. Llegado el caso, si eso ocurriera, Microsoft Azure relanzaría de nuevo la ejecución de la tarea en una de las máquinas que sigan formando parte de la infraestructura. Sin embargo, el usuario se vería evidentemente penalizado en sus tiempos de respuesta.

Por tanto, como es lógico, caben contemplar diferentes alternativas. Una de ellas sería que no tuviéramos en cuenta la constante UMBRAL_SUP y sólo nos planteáramos liberar máquinas cuando todas las que estén desplegadas en un momento determinado estén ociosas. En ese caso, cuando no haya ninguna tarea en ejecución en el sistema, reduciríamos el número de máquinas de la plataforma al mínimo establecido (MIN_MV). Con ello, ningún usuario se vería penalizado, ya que nunca se apagaría una máquina que estuviera ejecutando una simulación. Sin embargo, esta política presenta el inconveniente de que los costes económicos serían altos, ya que las instancias permanecerían en marcha a pesar de no ejecutar ningún trabajo. La otra alternativa sería asignar un valor a la constante UMBRAL_SUP, liberando una o más instancias simultáneamente cuando alcancemos o estemos por encima de dicho valor, alcanzando un valor mínimo de MIN_MV máquinas en marcha. Esta aproximación podrá incrementar el tiempo de ejecución de algunas simulaciones y reducir la productividad del sistema, pero el precio de albergar el servicio en una plataforma de pago por uso como la de Microsoft sería inferior.

- *CDMI Service*: Este componente transporta los datos de entrada y salida al servicio Cloud a través de la implementación del protocolo estándar CDMI. Para ello, incorpora un conjunto de operaciones que forman parte de dicha especificación encargadas de la gestión de los contenedores y los datos mediante operaciones de creación, lectura, actualización (o escritura) y borrado. Este módulo se ejecuta como una instancia particular, a nivel local o en la nube, diferente a las que ejecutan el Generic Worker, el Gestor del

Servicio de Análisis Estructural y el Simulador Estructural.

- *Accounting Service*: Es el componente, proporcionado como fruto de los desarrollos del proyecto VENUS-C, que monitoriza todas las acciones realizadas por los usuarios en el Servicio Cloud de Análisis Estructural y lleva a cabo la contabilidad y la facturación en el uso del mismo. Estos servicios están accesibles al usuario a través de un navegador web, en el que puede consultar su consumo y su facturación.

Para incrementar la productividad del sistema y conseguir que las simulaciones estáticas no se vean penalizadas por los elevados tiempos de respuesta que pueden presentar las simulaciones dinámicas a lo largo del tiempo, se ha optado por desplegar dos Servicios Cloud de Análisis Estructural. Uno de ellos, encargado de atender la simulaciones en las que se requiere un análisis estático, modal o modal espectral, y otro en el que se atienden peticiones de análisis dinámicos a lo largo del tiempo, bien sea por métodos de integración directa o por superposición modal.

A modo de ejemplo, supongamos que disponemos de un despliegue de 30 instancias, todas ellas del mismo tipo. Si 2 de ellas las empleamos para albergar el servicio CDMI y la ejecución del Gestor de Elasticidad, podríamos dedicar de 2 a 10 máquinas para atender las peticiones que lleguen al primero de los servicios y de 4 a 18 máquinas para las simulaciones que requieran al segundo. El intervalo de tiempo bajo el cual el Gestor de Elasticidad monitorizaría el estado del sistema podríamos fijarlo a 15 minutos, siendo 2 el número de instancias que pueden permanecer ociosas, por debajo del cual desplegaremos nuevas máquinas virtuales, y por encima del cual eliminaremos a alguna de ellas. El número de máquinas virtuales que pondremos en marcha simultáneamente y que eliminaremos de la infraestructura también será igual a dos.

Evidentemente los números indicados no dejan de ser una estimación, los cuales dependerían como es lógico del número de instancias totales en el despliegue y de la carga real del sistema, una vez que éste pasara a estar en modo de producción.

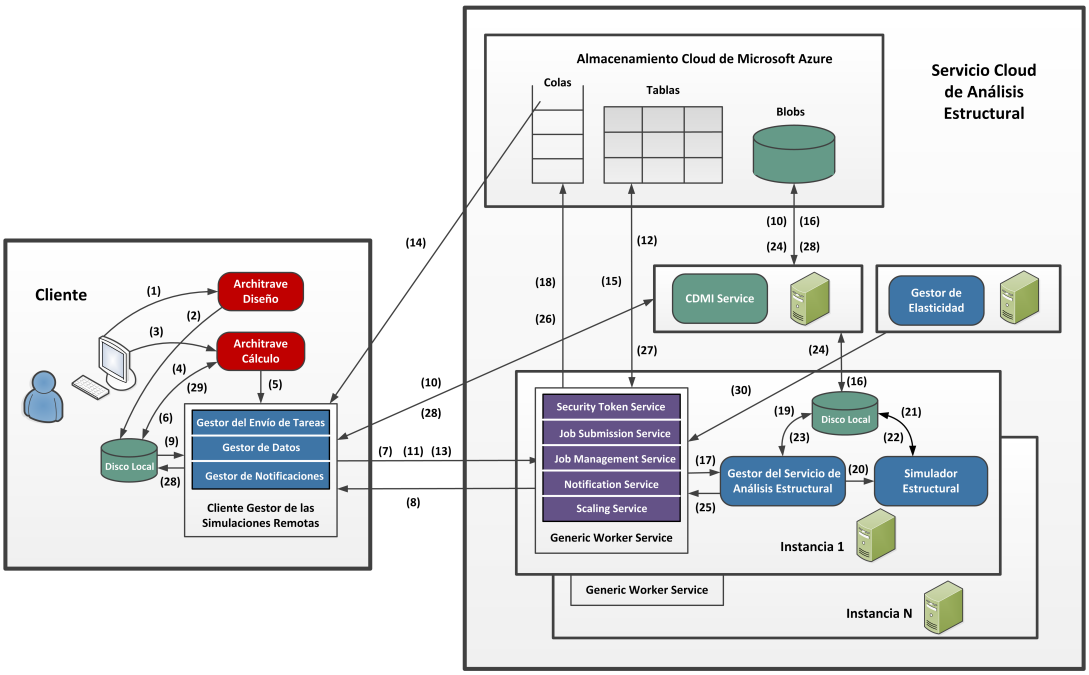


Figura 8.8: Gestión de la simulación de una estructura en la nube mediante Generic Worker.

8.2.2.4. Gestión de las simulaciones

La figura 8.8 recoge el ciclo de vida del análisis de una estructura en el servicio Cloud, el cual comienza por su diseño y finaliza con la visualización de los resultados, una vez llevado a cabo el análisis remoto de la misma en dicho servicio de acuerdo al tipo de cálculo elegido por el usuario.

El primer paso comienza desde Architrave Diseño (1), donde el usuario dibuja la geometría de la estructura, asigna las propiedades a los elementos estructurales y define las cargas externas aplicadas, almacenando dicha información en un fichero XML (2). A continuación, el usuario ejecutará Architrave Cálculo (3), desde donde leerá el fichero generado por Architrave Diseño (4), podrá modificar las propiedades de algún elemento que forme parte de la estructura, así como las cargas aplicadas, escogerá los parámetros que determinan el tipo de cálculo a realizar y configurará además la simulación remota. Tras ello, Architrave Cálculo lanzará al Cliente Gestor de las Simulaciones Remotas (5), informándole de la configuración de la simulación y generará dos ficheros binarios, uno con la geometría de la estructura y otro con los parámetros correspondientes al cálculo de

la misma (6).

La ejecución del Cliente Gestor de las Simulaciones Remotas comienza pidiéndole al usuario que proporcione su nombre y su contraseña, las cuales serán validadas en la nube por medio del Security Token Service (7). Si el usuario tiene permisos para acceder al servicio Cloud, el Gestor del Envío de Tareas obtendrá del Job Management Service un listado actualizado de las simulaciones lanzadas previamente al servicio, junto con el estado real de las mismas (8), informando al usuario. Será a partir de entonces cuando el cliente podrá cancelar la ejecución de alguna de ellas, recuperar los resultados o enviar nuevos trabajos al servicio.

Simultáneamente, se llevará a cabo el envío de la tarea. Para ello, el Gestor de Datos lee los ficheros binarios relativos a la geometría de la estructura y a los parámetros de cálculo (9) y los copia al servicio Cloud de almacenamiento (10), empleando el servicio CDMI. Por su parte, el Gestor del Envío de Tareas enviará el trabajo al servicio Cloud, a modo de un documento JSDL con su descripción (11), siendo el Job Submission Service, de una de las instancias, el componente del servicio que se encargará de registrar la simulación (12) en una tabla de Azure. A continuación, el Gestor de Notificaciones se suscribe a las notificaciones de cambio de estado y del progreso, si procede, de la tarea (13) por medio del Notification Service y consulta periódicamente dichos cambios de estado (14), informando en consecuencia al usuario de los mismos y, en el caso de un análisis dinámico a lo largo del tiempo, del avance en la ejecución de la simulación.

Cada una de las instancias que forman parte del despliegue del servicio Cloud y que permanecen ociosas consultan periódicamente la tabla del servicio de almacenamiento en busca de tareas a ejecutar. Como resultado, una de ellas se encargará de procesar la simulación, obteniendo por medio del Job Management Service la información referente a dicha simulación (15), junto con los ficheros que guardan la geometría de la estructura y los parámetros de cálculo (16). Una vez copiados dichos ficheros al disco local de la máquina virtual, el Job Management Service ejecutará al Gestor del Servicio de Análisis Estructural (17) y el Notification Service registrará el cambio en el estado de la tarea (18).

Tras ponerse en marcha, el Gestor del Servicio de Análisis Estructural leerá el fichero binario con la geometría y las cargas aplicadas y lo transformará en un

fichero que será leído por el Simulador Estructural (19), al cual invocará con los parámetros apropiados en función del tipo de cálculo escogido por el usuario (20). Tras leer dicho fichero de entrada (21), el Simulador Estructural lleva a cabo el análisis de la estructura, escribiendo los ficheros de resultados en disco, bien sea al final de la simulación o a medida que ésta transcurre (22). Simultáneamente, el Gestor del Servicio de Análisis Estructural empaquetará los diferentes ficheros generados (23) y los moverá al almacenamiento Cloud de Azure mediante el servicio CDMI (24), para cada instante de tiempo y de forma solapada con la ejecución o una vez finalizada, siempre dependiendo del tipo de cálculo y de la definición de la simulación remota llevada a cabo por el usuario. Adicionalmente, el Gestor del Servicio de Análisis Estructural irá informando al Notification Service del progreso (proporcionándole un número entero correspondiente al instante de tiempo ya simulado) o de la finalización de la simulación (25), quien incorporará a su vez dichos avances o cambios de estado en los sistemas de colas, con el fin de informar al usuario (26). Finalmente, transcurrida la simulación, el Job Management Service accede a la tabla de trabajos y deja registrado que la tarea ha concluido con éxito (27).

Por otro lado, si el usuario ha escogido la opción de descargar los resultados automáticamente, el Gestor de Datos llevará a cabo la recogida de los ficheros de salida una vez transcurrida la simulación o, en el caso de un análisis dinámico a lo largo del tiempo en el que hay que recoger los resultados de todos los instantes, de manera simultánea con la ejecución de la misma (28), tras ser informado de su disponibilidad por parte del Gestor de Notificaciones. A partir de ese momento, la respuesta de la estructura podrá ser visualizada e interpretada desde Architrave Cálculo (29).

Por último, conviene destacar que el Gestor de Elasticidad analizará constantemente el número de instancias que se encuentran disponibles y a la espera de ejecutar una tarea. Cuando dicho número sea inferior o superior a los umbrales establecidos, se pondrá en contacto con el Scaling Service con el objetivo de incorporar o eliminar de la infraestructura computacional un número determinado de máquinas virtuales, quien llevará a cabo la labor encomendada.

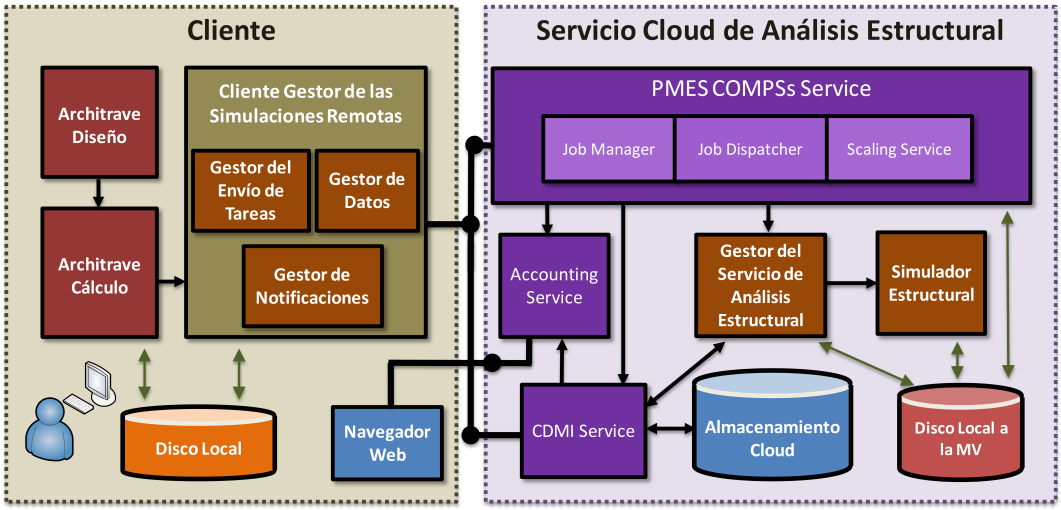


Figura 8.9: Arquitectura del sistema Cloud de Análisis Estructural basado en PMES COMPSs.

8.2.3. Desarrollos basados en PMES COMPSs

La figura 8.9 nos muestra la arquitectura del sistema Cloud desarrollado y desplegado basado en PMES COMPSs. Mientras que las máquinas virtuales que forman parte de la infraestructura gestionada por el Generic Worker consultan una tabla en la que se registran los trabajos que han llegado al sistema, con intención de atender la ejecución de uno de ellos, en el caso de PMES COMPSs ocurre que los trabajos se asignan a las máquinas virtuales de forma orquestada, desplegando para ello nuevas máquinas de forma automática, mediante OpenNebula o EMOTIVE, a medida que se incrementa el número de trabajos en el servicio.

Como podemos observar, hemos mantenido los mismos componentes del lado del cliente, aunque han sido acordemente modificados para poder interaccionar con PMES COMPSs. Esto supone que el usuario dispondrá de un mismo interfaz gráfico para definir las simulaciones remotas, para estar informado de sus cambios de estado o para recibir los resultados, independientemente por tanto de la plataforma destino en la que se realice el análisis estructural, la cual le será completamente transparente.

Al igual que ocurría en el caso de los desarrollos bajo Generic Worker, las comunicaciones entre el cliente y el servicio Cloud se realizan mediante el servicio

CDMI, el cual estará desplegado en una máquina virtual independiente.

Por la parte Cloud, PMES COMPSs posee estos tres componentes:

- *Job Manager Service*: Se encarga de ir registrando las diferentes simulaciones a ejecutar por parte del sistema y proporcionar el estado de un trabajo.
- *Job Dispatcher Service*: Es el responsable de la ejecución de los trabajos.
- *Scaling Service*: Ajusta, de forma dinámica, el número de máquinas virtuales que forman parte de la plataforma computacional, incrementando o decrementando su número en función de la carga del sistema.

Como podemos observar, PMES COMPSs no posee un servicio de notificaciones, razón por la cual no podremos suscribirnos a él desde el cliente y no sacaremos partido del Gestor de Notificaciones del que disponemos. Para conocer por tanto el estado de una simulación, el Gestor del Envío de Tareas debe enviar una petición expresa para obtenerlo. Adicionalmente, para determinar el progreso de un trabajo bajo un análisis dinámico a lo largo del tiempo, el Gestor de Datos debe consultar periódicamente la disponibilidad de los resultados en el almacenamiento Cloud para un instante de tiempo concreto, en lo que se conoce como un modelo gestionado por datos o *data-driven*.

Por otro lado, no es necesario incluir en el servicio Cloud el componente al que denominábamos el Gestor de la Elasticidad, ya que PMES COMPSs ofrece automáticamente esta funcionalidad, desplegando tantas máquinas virtuales como sean necesarias a la hora de ejecutar individualmente cada trabajo, eliminándolas cuando éste concluya.

Mantenemos, como no podía ser de otro modo, al Gestor del Servicio de Análisis Estructural y al Simulador Estructural, los cuales son componentes idénticos a los ya descritos en el caso del servicio basado en Generic Worker, con la única salvedad de que ahora deben estar compilados para trabajar en máquinas virtuales desplegadas sobre el sistema operativo Linux. A priori, el principal inconveniente viene de la mano del Gestor del Servicio de Análisis Estructural, que está escrito completamente en .NET. No obstante, ese inconveniente lo hemos solventado gracias a los desarrollos del proyecto Mono [336], el cual posee la capacidad de

poder ejecutar código escrito en .NET en diferentes plataformas, entre las que se encuentra Linux. Tendremos por tanto que instalar el runtime de Mono en las máquinas virtuales en las que vayamos a lanzar nuestras tareas. El Simulador Estructural está sin embargo basado en ANSI C, lo cual garantiza su portabilidad a cualquier plataforma destino, aunque para poder ejecutarlo es necesario compilar también en Linux las diferentes librerías numéricas de las que depende.

COMPSs ofrece al usuario la posibilidad de ejecutar su aplicación original en el Cloud, en paralelo, sin realizar en ella ninguna modificación. No obstante, para ello será necesario que desarrolle una aplicación COMPSs compuesta por los tres siguientes ficheros en código Java:

- El código de la aplicación principal: Consiste en el código que se ejecuta secuencialmente, compuesto por llamadas a los métodos seleccionados por el usuario que serán ejecutados en el Cloud.
- El código de los métodos remotos: Consiste en la implementación de las tareas a ejecutar en paralelo.
- Interfaz Java: Declara los métodos seleccionados por el usuario para poder ejecutarse como tareas remotas, junto con la información necesaria para planificar la ejecución de cada una de ellas. Entre dicha información se incluyen los requisitos hardware, especificando el número de núcleos computacionales o el tamaño de la memoria RAM necesitada.

La aplicación principal se ejecutará por parte de COMPSs en la máquina virtual llamada *Master*, reemplazando las invocaciones a los métodos seleccionados por el usuario con la creación de las tareas remotas, las cuales se ejecutarán en paralelo en una o más máquinas virtuales denominadas *Workers*. En nuestro caso, el método remoto estará únicamente formado por una invocación al Gestor del Servicio de Análisis Estructural, para su puesta en marcha, el cual se encargará a su vez de lanzar al Simulador Estructural. Ambos componentes se ejecutarán en la misma máquina virtual.

La figura 8.10 nos muestra la gestión de una simulación en la nube mediante el Servicio de Análisis Estructural basado en PMES COMPSs. Al igual que ocurría

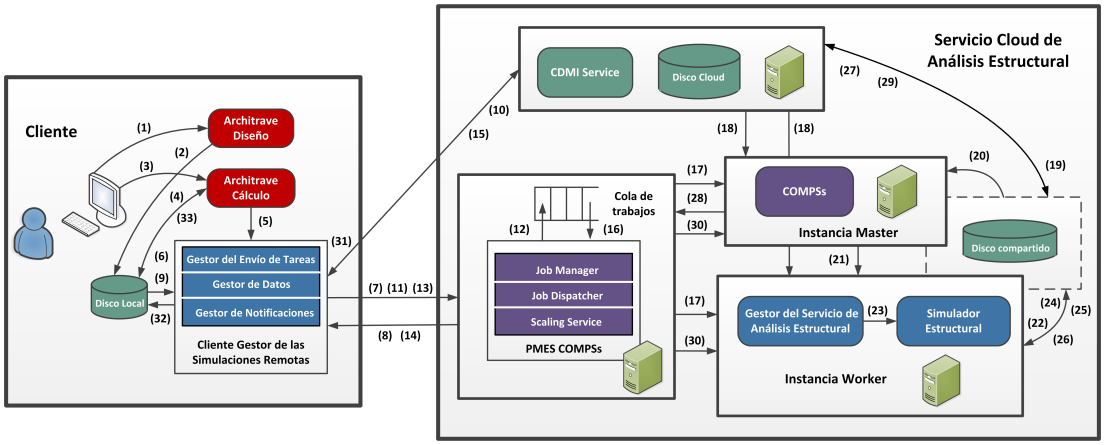


Figura 8.10: Gestión de la simulación de una estructura en la nube mediante PMES COMPSs.

en el caso del uso del Generic Worker, el usuario debe identificarse y autenticarse (7) antes de acceder al servicio. Para ello, PMES COMPSs nos permite utilizar o bien el nombre del usuario y la contraseña o bien certificados de tipo X.509.

El Gestor del Envío de Tareas le pedirá al Job Manager que le indique el estado de las simulaciones enviadas al servicio con anterioridad (8), informando por pantalla al usuario. A su vez, el Gestor de Datos leerá de disco los ficheros con la geometría y la cargas aplicadas y los parámetros de cálculo (9) y los copiará al servicio de almacenamiento del que dispone PMES COMPSs, gracias al servicio CDMI (10). A continuación, el Gestor del Envío de Tareas enviará el documento JSDL con la descripción del trabajo al servicio Cloud gestionado por PMES COMPSs (11), siendo recogido por el Job Manager, quien creará el trabajo y lo incorporará a un sistema de colas del que forman parte los diferentes trabajos a ejecutar en el servicio (12), delegando por completo su ejecución en manos del Job Dispatcher. Entre otros, el fichero XML que almacena la descripción JSDL del trabajo está formado por:

- Su nombre y su identificador.
- El nombre de la aplicación COMPSs a ejecutar en las máquinas Master y Workers y el de los ficheros con la geometría de la estructura, los parámetros de cálculo y los resultados de salida, junto con su ubicación en el servicio de almacenamiento Cloud.

- El nombre de la imagen a desplegar en las máquinas virtuales. Dicha imagen, que habremos preparado e incorporado previamente al repositorio de imágenes de máquinas virtuales de VENUS-C, estará compuesta por las dependencias software de nuestras aplicaciones a ejecutar, tales como Mono, OpenMP, MPI o las diferentes librerías numéricas de las que depende el Simulador Estructural.
- El número mínimo y máximo de máquinas virtuales que se desplegarán para ejecutar el trabajo, junto con el tamaño del disco de almacenamiento compartido entre todas ellas.

Llegado este momento, el Gestor del Envío de Tareas consultará periódicamente el estado del trabajo (13), siendo notificado por el Job Manager (14). Asimismo, para poder conocer cómo avanza la simulación de un cálculo dinámico a lo largo del tiempo, el Gestor de Datos consultará con una determinada frecuencia la disponibilidad de los resultados en el repositorio Cloud para cada instante de tiempo (15).

El propio Job Dispatcher dispone de un conjunto de hilos de ejecución que van recogiendo los trabajos del citado sistema de colas (16). Para cada uno de ellos, el Job Manager actualizará el estado del trabajo y solicitará la puesta en marcha, por medio del Scaling Service, de dos máquinas virtuales en las que se ejecutará el trabajo, a la cuales llamaremos las instancias *Master* y *Worker* (17). En la primera de ellas se pondrá en marcha el entorno de ejecución de COMPSs (18), mientras que en la segunda se desplegará la imagen bajo Linux especificada en el documento JSDL (18). Una vez que se han desplegado y configurado dichas máquinas virtuales, el Job Dispatcher copiará a un disco compartido, desde el servicio Cloud de almacenamiento, los ficheros binarios del Gestor del Servicio de Análisis Estructural y del Simulador Estructural, así como los ficheros de entrada de los que dependen y la aplicación COMPSs desarrollada en Java (19).

A continuación, COMPSs se pondrá en marcha desde la instancia Master, leerá y ejecutará la aplicación Java principal (20) y lanzará la ejecución de la tarea remota en la máquina Worker (21), la cual a su vez invocará al Gestor del Servicio de Análisis Estructural, quien leerá la geometría estructural y proporcionará el fichero de entrada necesitado por parte del Simulador Estructural (22), ejecu-

tándolo de acuerdo a sus parámetros de cálculo (23). El Simulador Estructural leerá los datos de entrada (24) e irá progresando con la ejecución de la estructura, escribiendo los datos en disco (25) para cada instante, en el caso de un análisis dinámico a lo largo del tiempo, o una vez finalizada la ejecución. Simultáneamente, el Gestor del Servicio de Análisis Estructural empaquetará los resultados de salida generados por el Simulador (26) y, en el caso de un análisis a lo largo del tiempo, los moverá al repositorio estructural del servicio Cloud (27), de forma solapada con la ejecución de la simulación. Finalizada la simulación, COMPSs informará a PMES COMPSs, siendo el Job Manager el encargado de modificar el estado del trabajo (28) y el Job Dispatcher el responsable de mover al repositorio estructural los datos de salida generados ante un cálculo diferente al dinámico a lo largo del tiempo (29). Una vez finalizada la ejecución del trabajo, las dos máquinas virtuales desplegadas para ejecutarlo serán eliminadas de la infraestructura (30). Entre tanto, el Gestor del Envío de la Tarea habrá consultado periódicamente el estado del trabajo y el Gestos de Datos habrá comprobado la cantidad de archivos disponibles en el repositorio de datos, informando al usuario del estado de la tarea y de su progreso, descargando automáticamente los resultados si el usuario así lo ha decidido, de manera solapada con la ejecución de la simulación o tras su finalización (31), almacenándolos en la máquina local del usuario (32). Dichos resultados serán leídos y mostrados gráficamente por Architrave Cálculo (33) e interpretados por parte del usuario.

8.2.4. Métodos disponibles para interaccionar con el servicio

Como ya sabemos, la arquitectura de VENUS-C permite la coexistencia de dos modelos de programación, englobados bajo el denominado PMES (Programming Model Enactment Service), los cuales satisfacen los requerimientos específicos en la ejecución de las tareas de los usuarios. PMES expone su funcionalidad mediante interfaces basados en servicios web, los cuales satisfacen la especificación *Basic Execution Service* (BES) de OGSA y donde los trabajos a ejecutar se expresan mediante documentos JSDL.

Para acceder desde una aplicación cliente a los servicios que gestionan la

ejecución remota de las tareas, VENUS-C proporciona a modo de API un conjunto de métodos para ser invocados desde aplicaciones Java o .NET, accediendo a la plataforma gestionada por PMES COMPSs o Generic Worker, respectivamente. A continuación se recoge una lista de aquellos métodos a utilizar para acceder a los citados servicios desde .NET:

- *SubmitVENUSJob*: Envía un trabajo al servicio Cloud. Como dato de entrada, necesita un documento JSDL que describe la aplicación a ejecutar. Como resultado de salida, proporciona el identificador de la simulación, a emplear en invocaciones futuras al servicio.
- *GetActivityStatuses*: Proporciona el estado de un conjunto de trabajos.
- *GetActivityDocuments*: Devuelve los documentos JSDL con la descripción de un conjunto de trabajos.
- *TerminateActivities*: Finaliza la ejecución de un conjunto de trabajos.
- *GetJobs*: Lista los trabajos enviados por un cliente en concreto.
- *GetAllJobs*: Obtiene un listado de todos los trabajos presentes en el sistema.
- *GetAllJobsByGroup*: Obtiene un listado de todos los trabajos de un cliente el cual pertenece a un grupo determinado de usuarios.
- *CreateSubscription*: Crea una subscripción a las notificaciones de cualquier cambio de estado de un trabajo.
- *CreateSubscriptionForStatuses*: Crea una subscripción a las notificaciones de cambios de ciertos estados de un trabajo.
- *Unsubscribe*: Elimina la subscripción a las notificaciones de un trabajo.
- *ListDeployments*: Obtiene una lista de los recursos desplegados.
- *UpdateDeployment*: Modifica el número de instancias desplegadas en la infraestructura.

Capítulo 9

Resultados experimentales

Este capitulo incluye los resultados experimentales realizados a fin de evaluar las prestaciones del Simulador Estructural, sobre un supercomputador de memoria compartida distribuida y sobre un PC tradicional, comparando sus tiempos de respuesta con otros programas de cálculo estructural. Adicionalmente, se han ejecutado dos casos de estudio para analizar el comportamiento de los servicios Grid y Cloud desarrollados.

9.1. Resultados del Simulador Estructural sobre un cluster

9.1.1. Plataforma computacional de ejecución

Todas las simulaciones se han realizado sobre un pequeño cluster llamado Kahan, propiedad del DSIC, el cual consta de 6 nodos biprocesador conectados mediante una red InfiniBand QDR 4X, con un ancho de banda de 40 Gbps y una tasa efectiva de 32 Gbps. Como podemos observar en la figura 9.1, cada nodo está formado a su vez por:

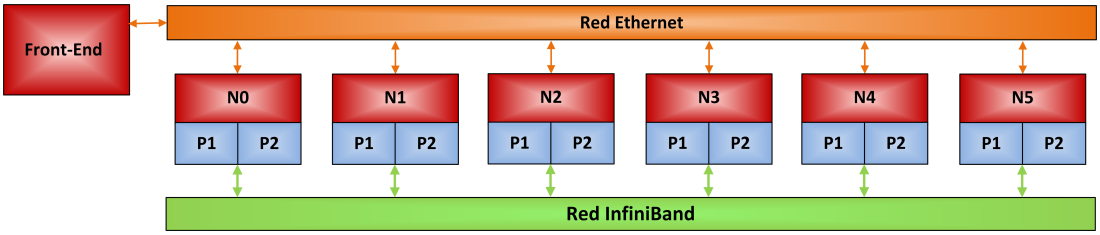


Figura 9.1: Arquitectura del cluster Kahan.

- Dos procesadores AMD Opteron 6272, cada uno de 16 núcleos computacionales con una frecuencia de reloj de 2.1 GHz y una memoria cache de nivel 3 de 16 MBytes.
- 32 GBytes de memoria RAM del tipo DDR3 SDRAM 1600.
- 500 GBytes de disco duro, con una controladora SATA de 6 Gbps.

Disponemos por tanto de 12 procesadores, 192 núcleos computacionales y 192 GBytes de memoria RAM. La máquina front-end, compuesta por un procesador Intel Core 2 Duo a 3 GHz con 4 GBytes de memoria, no pertenece al cluster. El directorio HOME del usuario está montado tanto en el front-end como en los nodos. El software instalado es el siguiente:

- Sistema operativo: CentOS 6.3.
- Compiladores: GNU 4.4.6 (gcc y gfortran) e Intel 13.1 (icc e ifort), los cuales incorporan la especificación 3.0 de OpenMP.
- MPI: Open MPI 1.6.2, configurado con soporte para InfiniBand.
- Sistema de colas de trabajos: Se gestiona mediante Torque y emplea a Maui como planificador.
- Librerías numéricas: LAPACK 3.2 e Intel MKL (versión Composer XE 2013), esta última formada por versiones multihilo optimizadas de BLAS, LAPACK, BLACS, PBLAS, ScaLAPACK y la versión 3.1 de PARDISO, la cual incorpora únicamente una paralelización a nivel de memoria compartida.

Nudos	Gdl problema completo	Gdl problema condensado	Barras	Triángulos	Hipótesis
12175	73050	71106	550	45174	2

Tabla 9.1: Características de la estructura NSDol.

9.1.2. Librerías numéricas adicionales

Para ejecutar las diferentes simulaciones, se han instalado y compilado sobre el cluster Kahan, empleando el compilador de Intel anteriormente mencionado, las siguientes librerías numéricas, todas ellas en sus últimas versiones: PETSc 3.6.3, MUMPS 5.0.0, PaStiX 5.2.2.20, hypre 2.10.0b, METIS 5.1.0, ParMETIS 4.0.3, SCOTCH y PT-SCOTCH 6.0.3, SLEPc 3.6.2, ARPACK 2.1 y PARPACK.

9.1.3. Estructuras seleccionadas como batería de test

Se han seleccionado las siguientes estructuras para llevar a cabo las simulaciones pertinentes:

- NSDol: Se corresponde con una modelización, mediante barras (550) y elementos finitos bidimensionales (842 triángulos y 11083 rectángulos), de la estructura de la iglesia de Nuestra Señora de los Dolores, ubicada en Valencia (ver figura 9.2). Construida de hormigón y acero, presenta como cimentación zapatas aisladas, zapatas corridas bajo muros y apoyos empotrados. Las características geométricas de la estructura consideradas a efectos de cálculo están recogidas en la tabla 9.1, en la cual se muestra el número de grados de libertad del problema completo, con 6 grados de libertad por nudo, y del problema condensado, en el cual sólo se tienen en cuenta aquellos grados de libertad de los cuales desconocemos, a priori, el valor de sus desplazamientos.

- Altura: Consiste en un edificio no construido de viviendas y oficinas de 60 plantas, representado en la figura 9.3. Está modelizado mediante vigas y pilares (1500 barras) y forjados discretizados por elementos finitos bidimensionales (138240 rectángulos). Sus características estructurales consideradas en la simulación aparecen en la tabla 9.2.

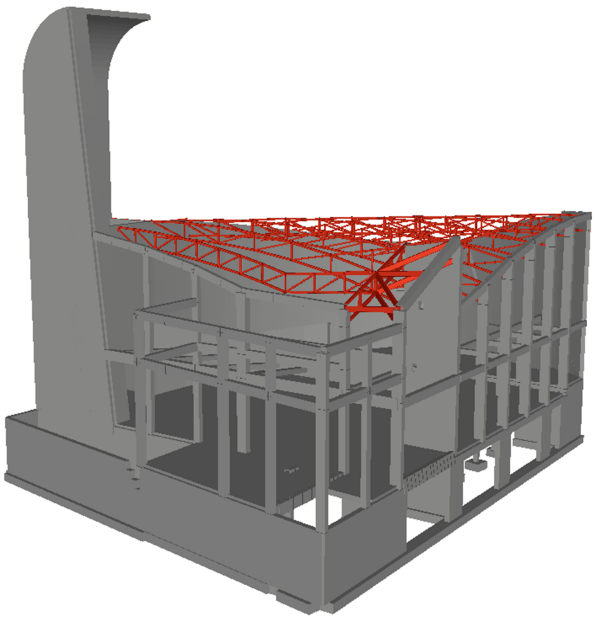


Figura 9.2: Modelización de la estructura NSDol.

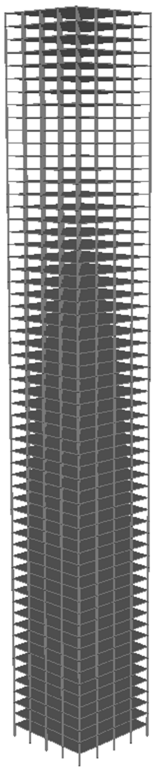


Figura 9.3: Modelización de la estructura Altura.

Nudos	Gdl problema completo	Gdl problema condensado	Barras	Triángulos	Hipótesis
144085	864510	864360	1500	552960	2

Tabla 9.2: Características de la estructura Altura.

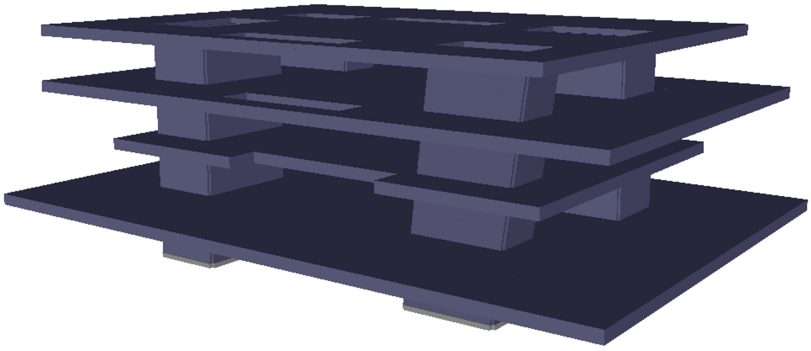


Figura 9.4: Modelización de la estructura Veles e Vents.

- Veles e Vents (Velas y Vientos): Situado en el interior del puerto de Valencia, se trata del edificio más emblemático y representativo de la Organización de la Copa América celebrada en dicha ciudad en el año 2007, representado en la figura 9.4. Fue construido entre 2005 y 2006 y recibió el premio europeo Leaf Awards. Dispone de cuatro plataformas horizontales, con amplias terrazas voladizas entre ellas, malladas mediante elementos finitos bidimensionales. Sus peculiaridades geométricas están recogidas en la tabla 9.3.

9.1.4. Análisis estático

El sistema de ecuaciones lineales a resolver, necesario para calcular los desplazamientos en los nudos de la estructura, representa el problema numérico más

Nudos	Gdl problema completo	Gdl problema condensado	Barras	Triángulos	Hipótesis
147605	885630	883692	0	293488	1

Tabla 9.3: Características de la estructura Veles e Vents.

relevante en un análisis estático tridimensional, más aún si hablamos de estructuras de gran dimensión. Como es bien conocido, podemos aplicar métodos directos o iterativos para resolver dichos sistemas.

A fin de comparar las prestaciones que ofrecen ambos tipos de métodos, en términos de tiempo de ejecución en secuencial y poner de manifiesto su aplicabilidad al análisis estructural, la figura 9.4 incluye los tiempos de cálculo de los desplazamientos de la estructura NSDol, resolviendo el sistema de ecuaciones con múltiples partes derecha por métodos directos, mediante las librerías MUMPS, PaStiX y PARDISO, o mediante el método iterativo del Gradiente Conjugado (GC), presente en PETSc, combinado con diferentes precondicionadores del propio PETSc o de *hypre*, con una tolerancia de convergencia relativa (rtol) de 1e-10.

Más concretamente, hemos empleado, por un lado, los precondicionadores paralelos de Jacobi a Bloques (JB) o Additive Schwarz (ASM), combinados en cada bloque individual con otro precondicionador como el de la descomposición incompleta de Cholesky con un nivel de llenado k, también conocida como ICC(k), o con el precondicionador de Eisenstat, el cual representa una variación del método iterativo estacionario SSOR, usándose como precondicionador por la izquierda y por la derecha del sistema de ecuaciones. Por otro lado, hemos analizado también diferentes precondicionadores de *hypre*, como alternativa a los citados de PETSc, entre los cuales se encuentran ParaSails, a modo de precondicionador paralelo basado en la inversa aproximada dispersa, o BoomerAMG, el cual es una implementación paralela del método multigrid algebraico descrito en [337] y del que se ha empleado su configuración de parámetros definida por defecto. Desafortunadamente, no ha sido posible emplear el precondicionador Euclid, el cual implementa una descomposición LU incompleta (ILU) en paralelo, ya que no forma parte de los precondicionadores de *hypre* que pueden utilizarse desde PETSc. Cualquier otra combinación de métodos iterativos y precondicionadores de PETSc o *hypre* proporcionó peores resultados.

Como podemos observar, debido al mal condicionamiento que presentan las matrices de rigidez, la resolución mediante los métodos iterativos y los precondicionadores de PETSc y *hypre* es mucho más lenta que la realizada mediante los métodos directos implementados en MUMPS, PaStiX o PARDISO. Además, cuando empleamos los métodos iterativos, el sistema de ecuaciones debe resolver-

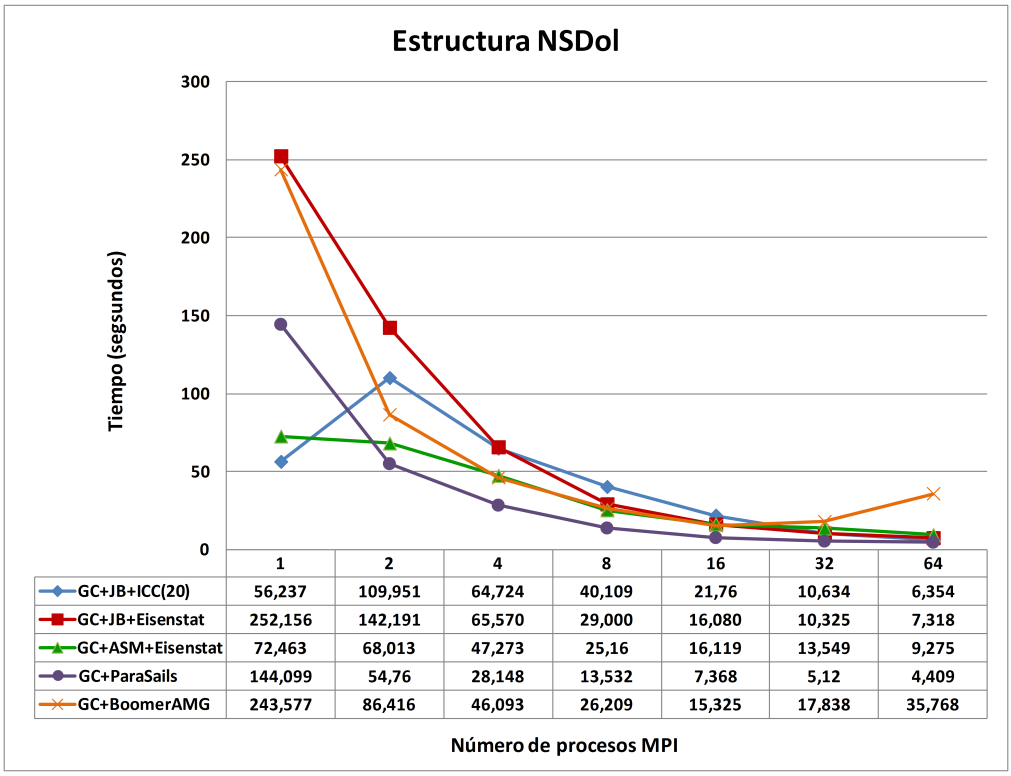
Librería o método de resolución	Tiempo (segs.)
MUMPS	1.58
PaStiX	3.12
PARDISO	3.29
GC+JB+ICC(20)	56.24
GC+JB+Eisenstat	252.16
GC+ASM+Eisenstat	72.46
GC+ParaSails	144.10
GC+BoomerAMG	243.58

Tabla 9.4: Tiempos invertidos en el cálculo secuencial de los desplazamientos de la estructura NSDol.

se para cada hipótesis de carga, o vector parte derecha del sistema, lo que resulta mucho menos eficiente que la resolución de los sistemas de ecuaciones triangulares mediante las sustituciones regresivas y progresivas llevadas a cabo por parte de los métodos directos. Es por ello que, en los análisis del resto de estructuras consideradas, emplearemos las librerías MUMPS, PaStiX y PARDISO descartando, en consecuencia, a los métodos iterativos.

No obstante, dichos métodos iterativos pueden resultar mucho más escalables que los métodos directos, motivo por el cual se ha calculado la estructura NSDol empleando un número de procesos MPI comprendido entre 1 y 64. Los tiempos de resolución del sistema de ecuaciones subyacente mediante dichos métodos iterativos se recogen en la figura 9.5. En dichas simulaciones, se ha empleado la técnica del particionado por corte de elementos, a partir de un grafo nodal, empleando el método de Particionado k-way Multinivel y tratando de minimizar el número de aristas cortadas.

Mientras que los mejores resultados, en secuencial, vienen dados por el método de Gradiente Conjugado con Jacobi a Bloques y la descomposición ICC con nivel 20 de llenado, los tiempos inferiores de cálculo se obtienen, para un número de procesos MPI comprendido entre 2 y 64, al emplear Gradiente Conjugado y el precondicionador ParaSails de *hypre*. El principal inconveniente de los precondicionadores de Jacobi a Bloques y Additive Schwarz es que el número de iteraciones necesarias hasta alcanzar la convergencia del sistema varía al incrementar el número de procesos, lo que se traduce por ejemplo en un incremento en el tiempo invertido al emplear 2 o 4 procesos frente al caso secuencial en la



	1	2	4	8	16	32	64
GC+JB+ICC(20)	56,237	109,951	64,724	40,109	21,76	10,634	6,354
GC+JB+Eisenstat	252,156	142,191	65,570	29,000	16,080	10,325	7,318
GC+ASM+Eisenstat	72,463	68,013	47,273	25,16	16,119	13,549	9,275
GC+ParaSails	144,099	54,76	28,148	13,532	7,368	5,12	4,409
GC+BoomerAMG	243,577	86,416	46,093	26,209	15,325	17,838	35,768

Figura 9.5: Tiempos invertidos en el cálculo en paralelo de los desplazamientos de la estructura NSDol.

combinación de Jacobi a Bloques e ICC(20).

Los incrementos de velocidad correspondientes de las ejecuciones paralelas frente a la ejecución secuencial se muestran en la tabla 9.5 y en la figura 9.6, donde los mejores resultados se obtienen con la combinación de Jacobi a Bloques y Eisenstat y donde resulta destacable el incremento superlineal de velocidad obtenido al emplear los precondicionadores de *hypre*.

Una vez descartados los métodos iterativos para resolver los sistemas de ecuaciones lineales, el siguiente paso consiste en proporcionar los resultados obtenidos en la escritura de los resultados en disco ante las diferentes alternativas descritas en la sección 7.2.11, en el caso de la estructura Altura, trabajando siempre con un disco compartido entre los diferentes nodos. El espacio de disco ocupado por los distintos tipos de resultados es de 58.66 MBytes y la estrategia de particionado de la estructura ha sido la del corte de nodos, a partir de un grafo nodal y empleando el método de Particionado k-way Multinivel de METIS, teniendo a

Método	Número de procesos MPI						
	1	2	4	8	16	32	64
GC+JB+ICC(20)	1	0.51	0.87	1.40	2.58	5.29	8.85
GC+JB+Eisenstat	1	1.77	3.85	8.70	15.68	24.42	34.46
GC+ASM+Eisenstat	1	1.07	1.53	2.88	4.50	5.35	7.81
GC+ParaSails	1	2.63	5.12	10.65	19.56	28.14	32.68
GC+BoomerAMG	1	2.82	5.28	9.29	15.89	13.65	6.81

Tabla 9.5: Incrementos de velocidad en el cálculo en paralelo de los desplazamientos de la estructura NSDol.

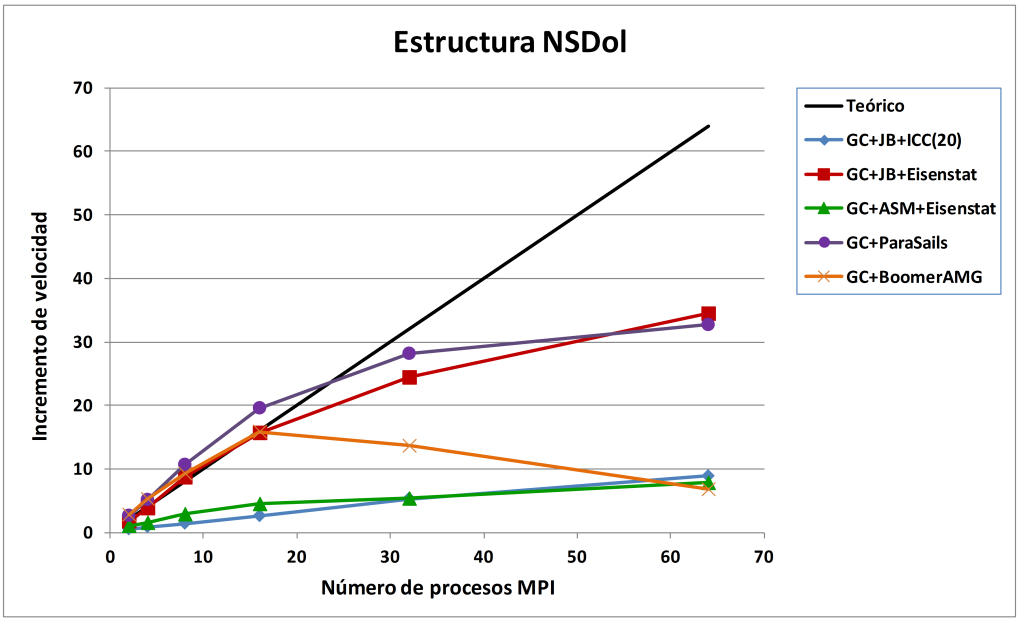


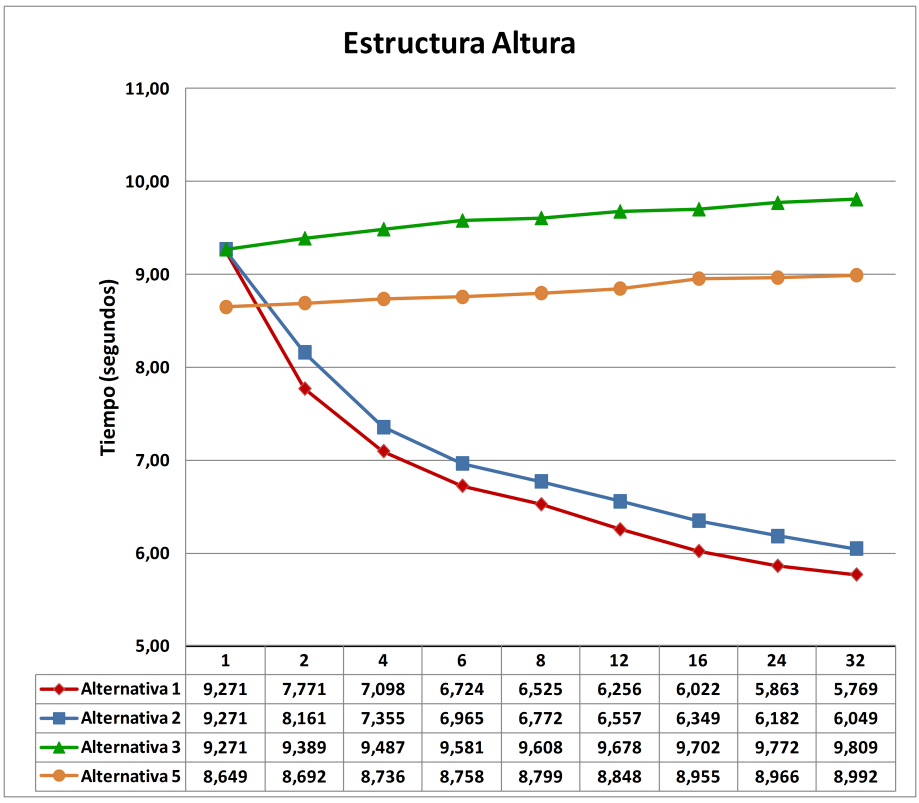
Figura 9.6: Incremento de velocidad en el cálculo en paralelo de los desplazamientos de la estructura NSDol.

la minimización del número de aristas cortadas como función objetivo. En este sentido, la figura 9.7 nos indica que las alternativas 1 y 2, en las cuales cada tarea escribe sus datos en ficheros diferentes a los del resto de tareas, consiguen reducir el tiempo de escritura de los datos a medida que se incrementa el número de procesos, existiendo una leve diferencia entre ambas debida a la sobrecarga de concatenar diferentes ficheros en uno solo por parte del hilo de ejecución desplegado en la alternativa 2. Por su parte, las alternativas 3 y 5 muestran que los tiempos de envío de todos los resultados a la tarea 0 para que los escriba en disco son pequeños, aumentando ligeramente a medida que se incrementa el número de procesos. Como era de esperar, resulta más rápido escribir todos los resultados en un mismo fichero que escribirlos en diferentes archivos.

Conviene resaltar que no se han incluido en esta memoria los resultados de las alternativas 4 y 6 basadas en MPI-IO, dado que el cluster utilizado en la experimentación utiliza como soporte a un sistema de ficheros POSIX convencional y no a un sistema de ficheros paralelo como GPFS (General Parallel File System) [338]. En ese sentido, los experimentos realizados demostraron que dichas alternativas 4 y 6 tuvieron un mejor comportamiento que las alternativas clásicas 3 y 5, pero peor que las alternativas 1 y 2, mostrando además una falta de escalabilidad cuando se planificó más de un proceso por nodo. Es por ello que en las próximas simulaciones emplearemos la alternativa 1.

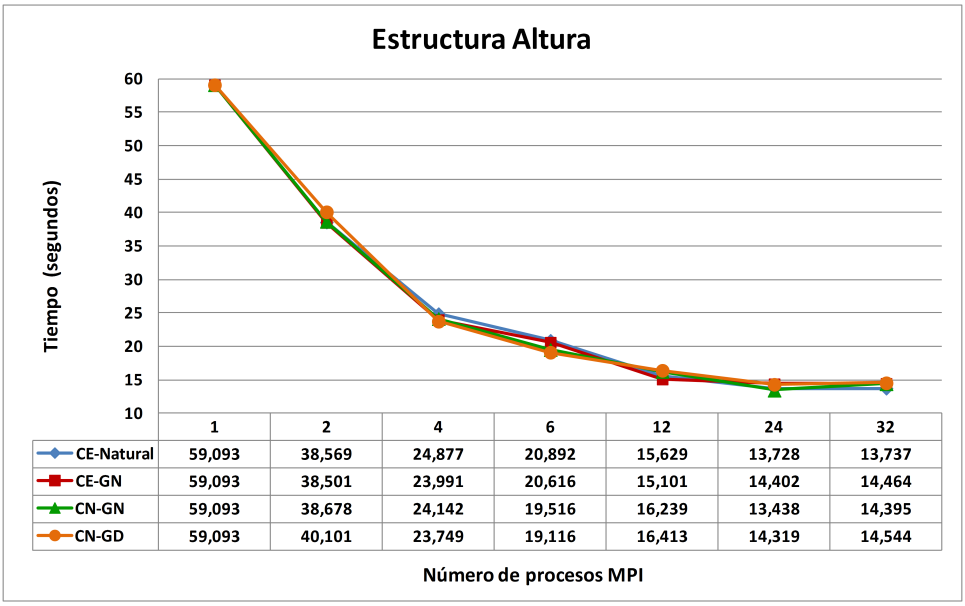
Otro aspecto relevante es el de identificar cuál de las diferentes estrategias de particionado de la estructura entre los procesos, descritas en la sección 7.2.2, proporciona las mejores prestaciones a nivel de tiempo de ejecución. La figura 9.8 detalla los tiempos completos de la simulación estática de la estructura Altura, incluyendo la escritura en disco, empleando las 4 siguientes posibilidades: particionado por corte de elementos natural (CE-Natural), donde no empleamos la librería METIS, particionado por corte de elementos a partir de un grado nodal (CE-GN), particionado por corte de nodos a partir de un grafo nodal (CN-GN) y particionado por corte de nodos a partir de un grafo dual (CN-GD).

Como podemos observar, los tiempos son muy similares entre las diferentes estrategias de particionado, lo cual demuestra que la técnica del particionado por corte de elementos escala prácticamente por igual que la técnica del particionado por corte de nodos. Curiosamente, la técnica de particionado por corte de ele-



	1	2	4	6	8	12	16	24	32
Alternativa 1	9,271	7,771	7,098	6,724	6,525	6,256	6,022	5,863	5,769
Alternativa 2	9,271	8,161	7,355	6,965	6,772	6,557	6,349	6,182	6,049
Alternativa 3	9,271	9,389	9,487	9,581	9,608	9,678	9,702	9,772	9,809
Alternativa 5	8,649	8,692	8,736	8,758	8,799	8,848	8,955	8,966	8,992

Figura 9.7: Tiempos de escritura en disco en paralelo de los resultados de cálculo estático de la estructura Altura.



	1	2	4	6	12	24	32
CE-Natural	59,093	38,569	24,877	20,892	15,629	13,728	13,737
CE-GN	59,093	38,501	23,991	20,616	15,101	14,402	14,464
CN-GN	59,093	38,678	24,142	19,516	16,239	13,438	14,395
CN-GD	59,093	40,101	23,749	19,116	16,413	14,319	14,544

Figura 9.8: Tiempos del cálculo estático en paralelo de la estructura Altura con diferentes técnicas de particionado.

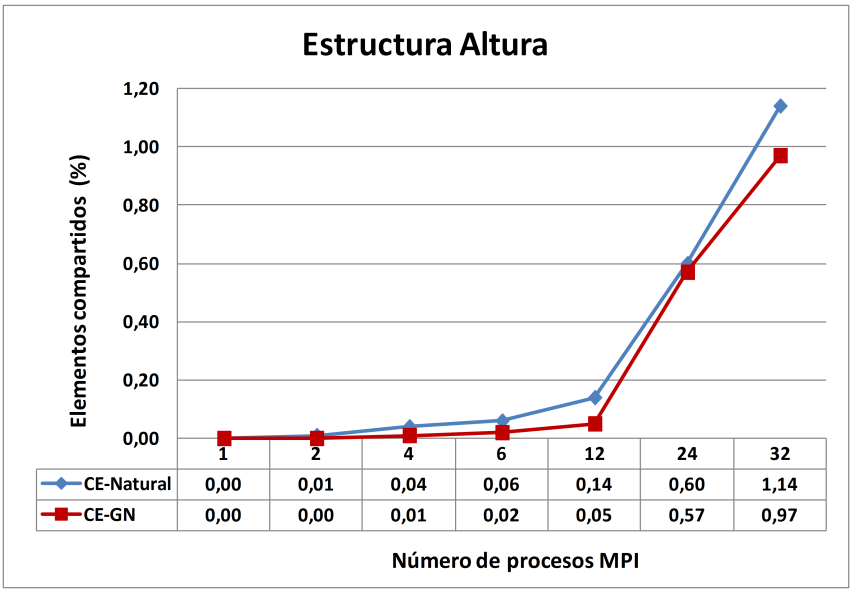
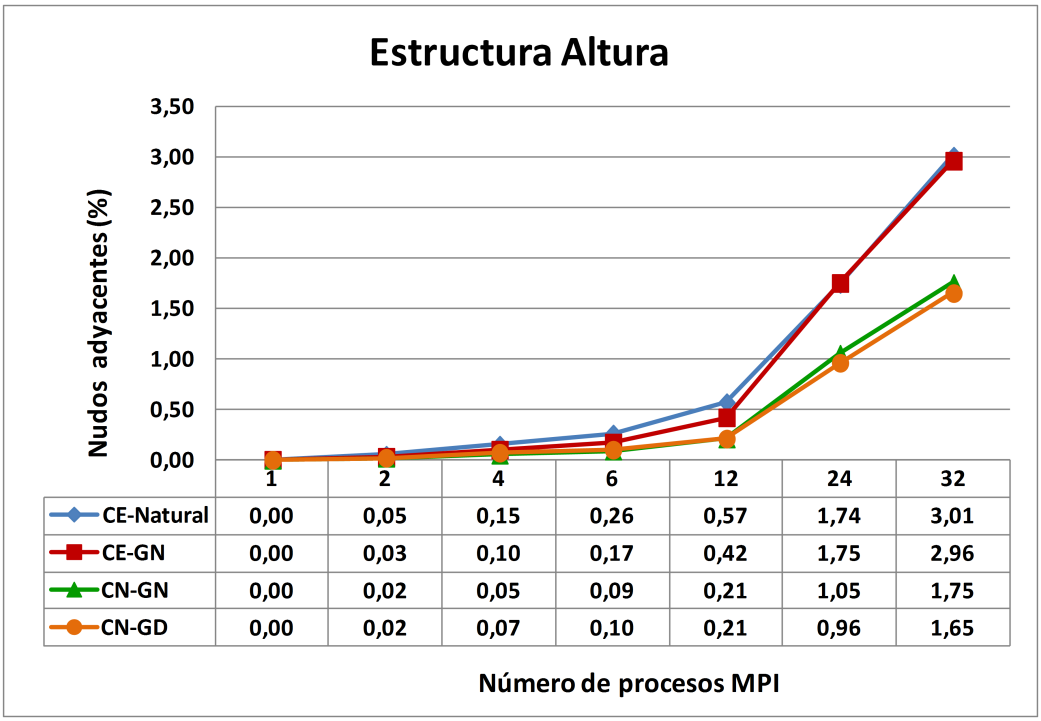


Figura 9.9: Porcentaje de elementos compartidos en el cálculo de la estructura Altura.

mentos natural se comportó igual de bien que el resto, lo cual es debido al alto grado de simetría y regularidad que presenta la estructura, tras partir además de una numeración inicial de los nodos, no arbitraria, de abajo a arriba.

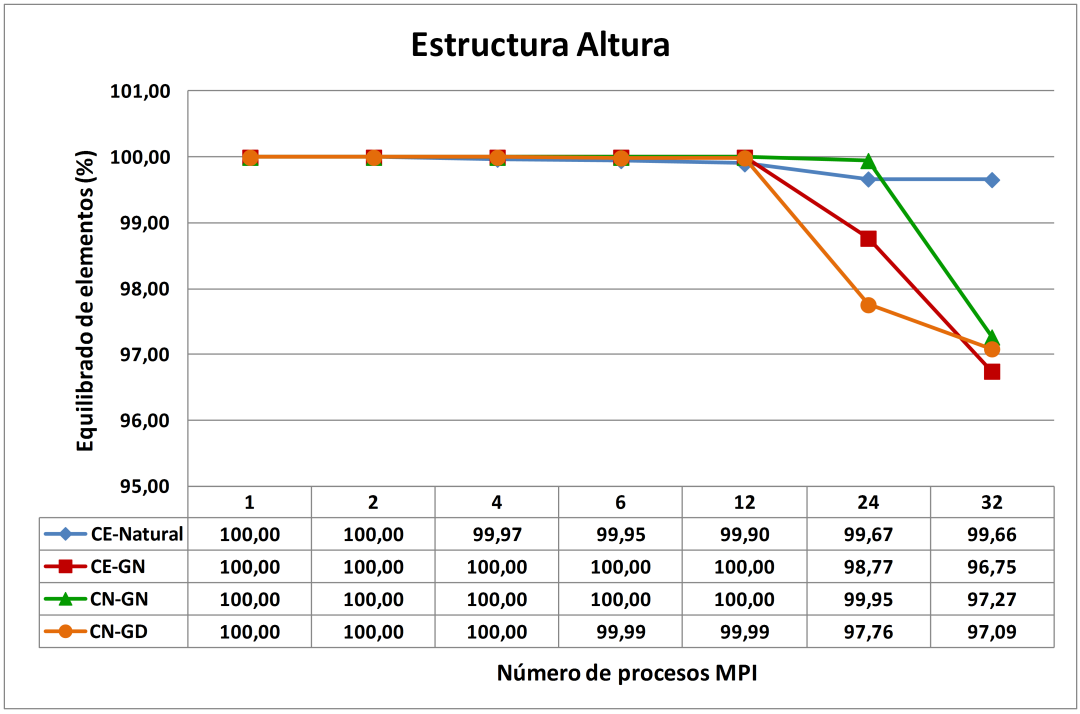
Si comparamos más en detalle los particionados por corte de elementos, observamos en la figura 9.9 que el particionado natural presenta un mayor porcentaje de elementos compartidos frente al particionado con METIS a partir de un grafo nodal, siendo mayor la diferencia existente a medida que aumenta el número de particiones. Por otro lado, la figura 9.10 refleja el porcentaje de nudos adyacentes para las 4 estrategias empleadas, siendo menor en el caso del particionado por corte de nodos, lo cual supone que el volumen de datos correspondiente a las comunicaciones de los desplazamientos en los nudos de unas tareas a otras, una vez calculados, será mayor en las técnicas de particionado por corte de elementos.

Finalmente, las figuras 9.11 y 9.12 nos muestran el porcentaje de distribución de los elementos y nodos de la estructura entre las distintas particiones. En este caso, es el particionado por corte de elementos natural el que presenta los mejores porcentajes. El resto de particionados muestran una línea descendente pronunciada para más de 12 particiones, retrasándose a más de 24 particiones en el caso del corte de nodos a partir de un grado nodal.



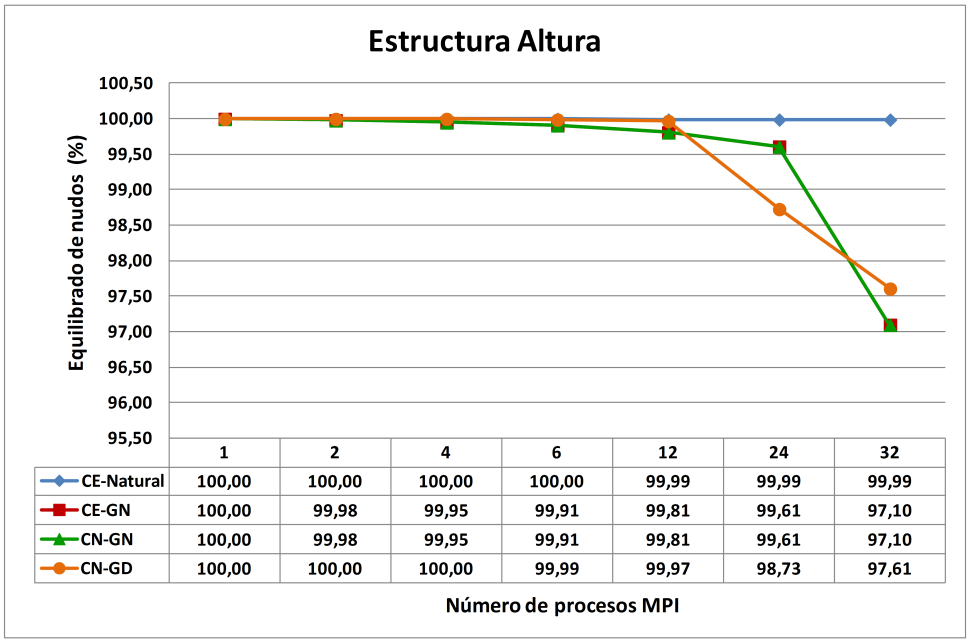
	1	2	4	6	12	24	32
CE-Natural	0,00	0,05	0,15	0,26	0,57	1,74	3,01
CE-GN	0,00	0,03	0,10	0,17	0,42	1,75	2,96
CN-GN	0,00	0,02	0,05	0,09	0,21	1,05	1,75
CN-GD	0,00	0,02	0,07	0,10	0,21	0,96	1,65

Figura 9.10: Porcentaje de nudos adyacentes en el cálculo de la estructura Altura.



	1	2	4	6	12	24	32
CE-Natural	100,00	100,00	99,97	99,95	99,90	99,67	99,66
CE-GN	100,00	100,00	100,00	100,00	100,00	98,77	96,75
CN-GN	100,00	100,00	100,00	100,00	100,00	99,95	97,27
CN-GD	100,00	100,00	100,00	99,99	99,99	97,76	97,09

Figura 9.11: Porcentaje de distribución de los elementos entre las particiones, en el cálculo de la estructura Altura.



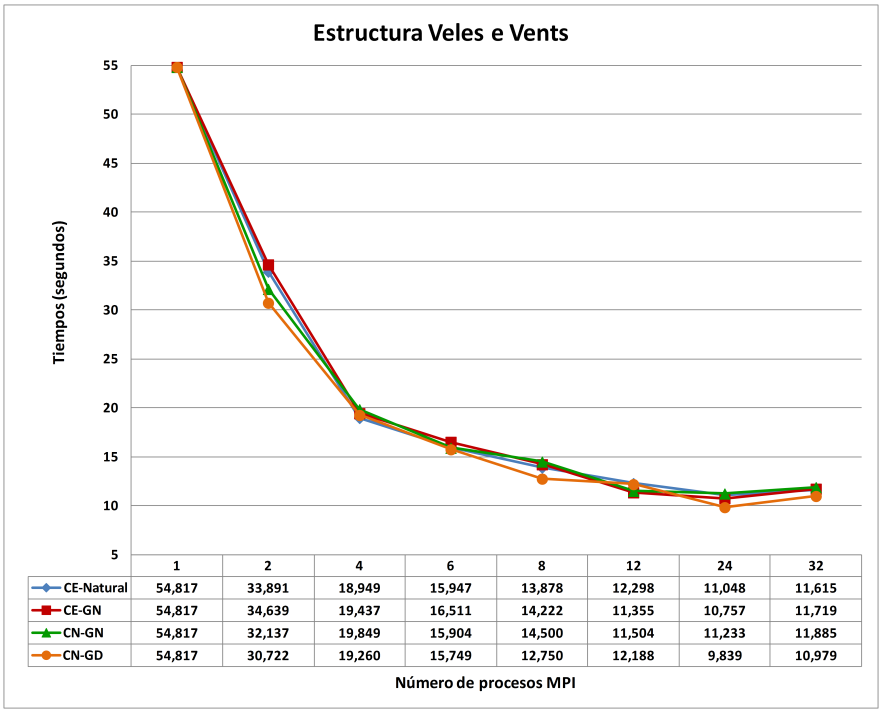
	1	2	4	6	12	24	32
CE-Natural	100,00	100,00	100,00	100,00	99,99	99,99	99,99
CE-GN	100,00	99,98	99,95	99,91	99,81	99,61	97,10
CN-GN	100,00	99,98	99,95	99,91	99,81	99,61	97,10
CN-GD	100,00	100,00	100,00	99,99	99,97	98,73	97,61

Figura 9.12: Porcentaje de distribución de los nodos entre las particiones, en el cálculo de la estructura Altura.

En todas estas pruebas se ha usado el método del Particionado k-way Multinivel de METIS, con la minimización de aristas cortadas como función objetivo, y se han resuelto los sistemas de ecuaciones mediante la librería MUMPS, empleando un único hilo de ejecución y múltiples procesos MPI. Desde el propio MUMPS, se ha ordenado la matriz de coeficientes empleado METIS. Otras librerías como ParMETIS, SCOTCH o PT-SCOTCH han dado lugar a tiempos de respuesta superiores en la resolución de los sistemas de ecuaciones.

Si repetimos la aplicación de las diferentes técnicas de particionado en el cálculo estático de la estructura de Veles e Vents, tal y como nos muestra la figura 9.13, observamos que las diferencias en los tiempos de simulación son ahora ligeramente mayores a las ocurridas en el caso de la estructura Altura, siendo normalmente mejores los obtenidos con el particionado por corte de nodos a partir de un grafo dual.

Al igual que en el caso de la estructura Altura, se incluyen a continuación los porcentajes de elementos compartidos (figura 9.14), nudos adyacentes (figura 9.15), distribución de los elementos (figura 9.16) y distribución de los nodos (figura 9.17) entre las particiones. Claramente, observamos los inconvenientes del

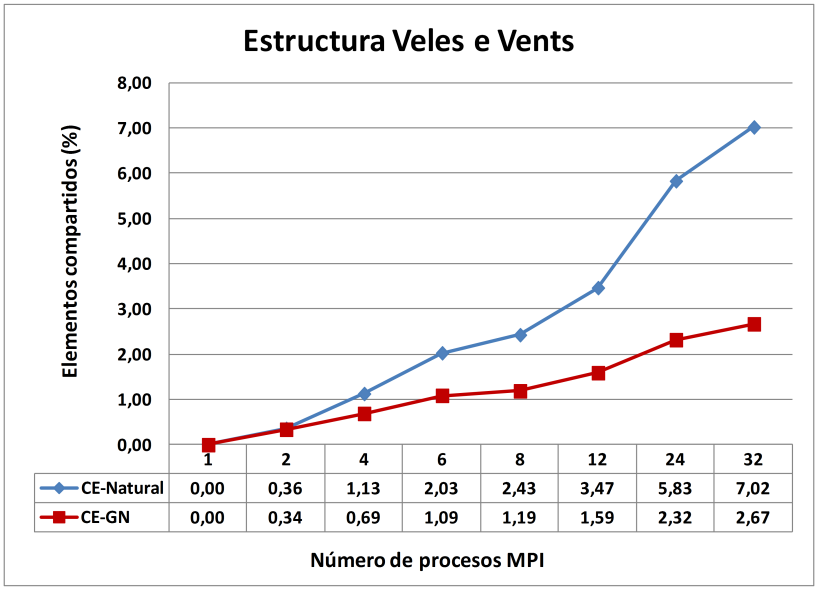


	1	2	4	6	8	12	24	32
CE-Natural	54,817	33,891	18,949	15,947	13,878	12,298	11,048	11,615
CE-GN	54,817	34,639	19,437	16,511	14,222	11,355	10,757	11,719
CN-GN	54,817	32,137	19,849	15,904	14,500	11,504	11,233	11,885
CN-GD	54,817	30,722	19,260	15,749	12,750	12,188	9,839	10,979

Figura 9.13: Tiempos del cálculo estático en paralelo de la estructura Veles e Vents con diferentes técnicas de particionado.

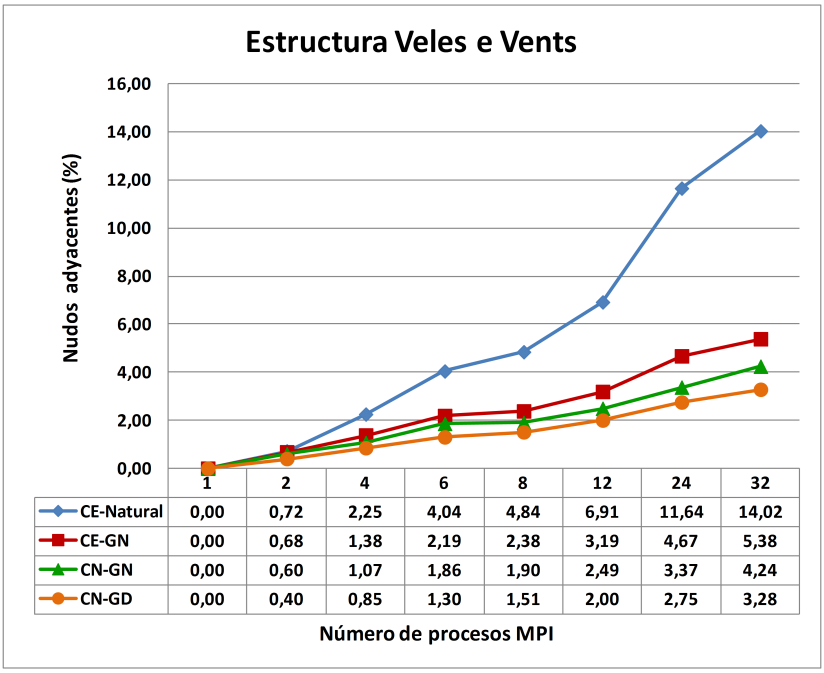
particionado por corte de elementos natural, el cual presenta los peores porcentajes en términos de elementos compartidos, nudos adyacentes y distribución de elementos. El porcentaje de nudos adyacentes es inferior en las técnicas por corte de nodos, como ocurría en el caso de la estructura Altura, las cuales ofrecen también un mejor equilibrado en la distribución de los elementos entre las particiones, con valores que se van alternando con el particionado por corte de elementos a partir de un grafo nodal en el caso de la distribución de los nodos. A su vez, el particionado por corte de nodos a partir de un grafo dual posee un porcentaje de nudos adyacentes inferior al de dicho particionado partiendo de un grafo nodal, lo cual supone que el volumen de comunicación de datos entre las tareas en las diferentes etapas de cálculo será inferior.

Dicho edificio de Veles e Vents será también el que utilizaremos para identificar cuál de las librerías numéricas dedicadas a la resolución de los sistemas de ecuaciones mediante métodos directos, como MUMPS, PaStiX o PARDISO, resulta ser la más apropiada para ser aplicada al análisis estructural. En este sentido, la figura 9.18 recoge los tiempos de ejecución dedicados al cálculo de los



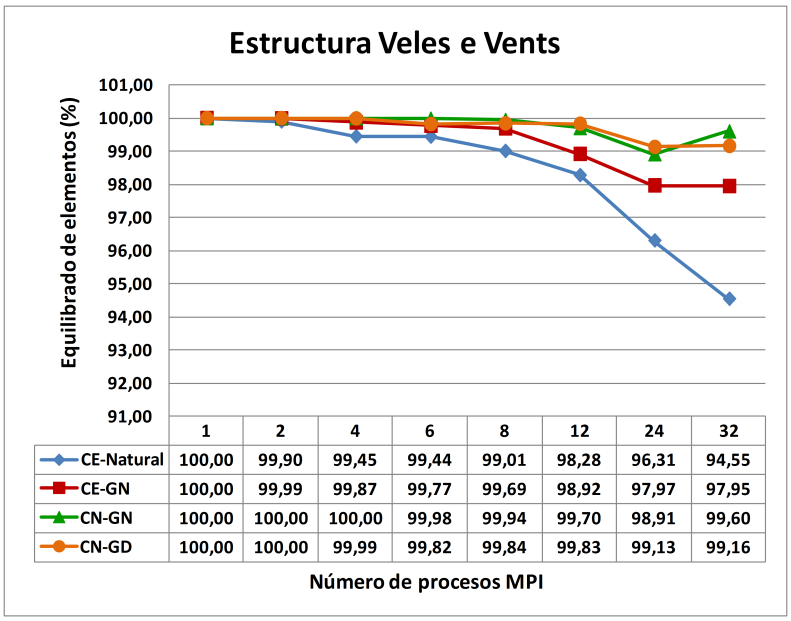
	1	2	4	6	8	12	24	32
CE-Natural	0,00	0,36	1,13	2,03	2,43	3,47	5,83	7,02
CE-GN	0,00	0,34	0,69	1,09	1,19	1,59	2,32	2,67

Figura 9.14: Porcentaje de elementos compartidos en el cálculo de la estructura Veles e Vents.



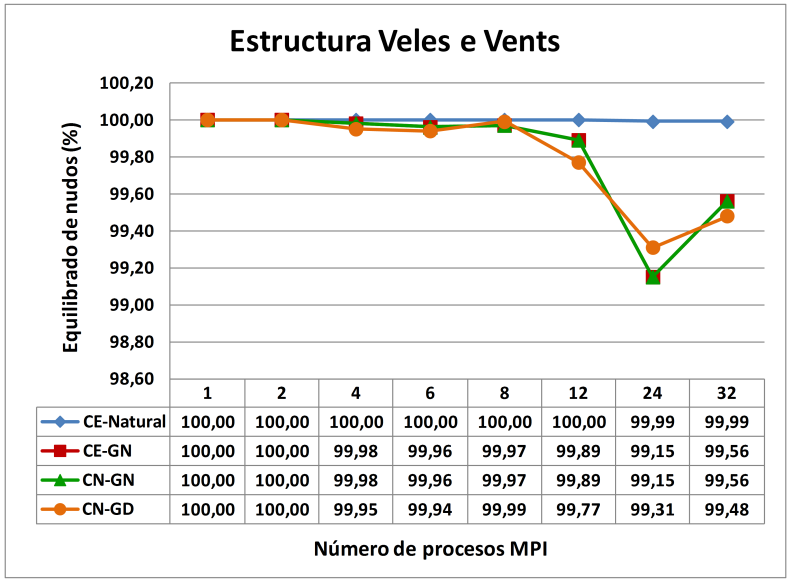
	1	2	4	6	8	12	24	32
CE-Natural	0,00	0,72	2,25	4,04	4,84	6,91	11,64	14,02
CE-GN	0,00	0,68	1,38	2,19	2,38	3,19	4,67	5,38
CN-GN	0,00	0,60	1,07	1,86	1,90	2,49	3,37	4,24
CN-GD	0,00	0,40	0,85	1,30	1,51	2,00	2,75	3,28

Figura 9.15: Porcentaje de nudos adyacentes en el cálculo de la estructura Veles e Vents.



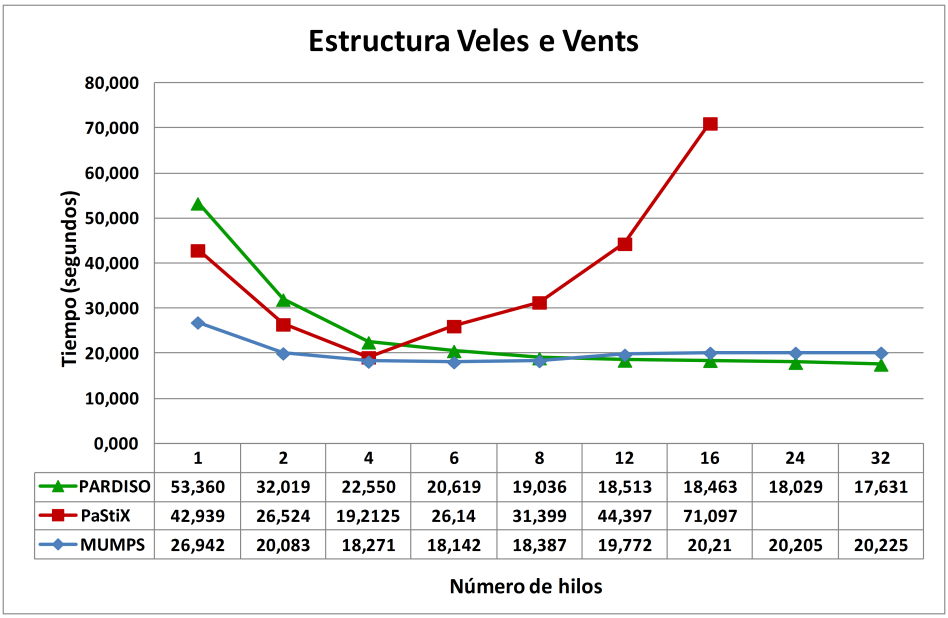
	1	2	4	6	8	12	24	32
CE-Natural	100,00	99,90	99,45	99,44	99,01	98,28	96,31	94,55
CE-GN	100,00	99,99	99,87	99,77	99,69	98,92	97,97	97,95
CN-GN	100,00	100,00	100,00	99,98	99,94	99,70	98,91	99,60
CN-GD	100,00	100,00	99,99	99,82	99,84	99,83	99,13	99,16

Figura 9.16: Porcentaje de distribución de los elementos entre las particiones, en el cálculo de la estructura Veles e Vents.



	1	2	4	6	8	12	24	32
CE-Natural	100,00	100,00	100,00	100,00	100,00	100,00	99,99	99,99
CE-GN	100,00	100,00	99,98	99,96	99,97	99,89	99,15	99,56
CN-GN	100,00	100,00	99,98	99,96	99,97	99,89	99,15	99,56
CN-GD	100,00	100,00	99,95	99,94	99,99	99,77	99,31	99,48

Figura 9.17: Porcentaje de distribución de los nodos entre las particiones, en el cálculo de la estructura Veles.



	1	2	4	6	8	12	16	24	32
PARDISO	53,360	32,019	22,550	20,619	19,036	18,513	18,463	18,029	17,631
PaStiX	42,939	26,524	19,2125	26,14	31,399	44,397	71,097		
MUMPS	26,942	20,083	18,271	18,142	18,387	19,772	20,21	20,205	20,225

Figura 9.18: Tiempos invertidos en el cálculo en paralelo de los desplazamientos de la estructura Veles e Vents empleando múltiples hilos.

desplazamientos de la estructura por parte de las 3 librerías citadas, empleando un proceso MPI y múltiples hilos de ejecución. A nivel secuencial, la librería MUMPS presenta el menor tiempo de respuesta, seguida de PaStiX y de PARDISO. A medida que aumentamos el número de hilos, el comportamiento de las librerías es dispar. Así, podemos apreciar en la tabla 9.6 que la librería MUMPS es la menos escalable, aunque hay que tener en cuenta que es la que parte con el tiempo de respuesta inferior al emplear un solo hilo. Para un número de hilos superior a 6, los tiempos de resolución comienzan a incrementarse ligeramente. Muy diferente es el comportamiento de PaStiX, que consigue reducir los tiempos de ejecución hasta emplear 4 hilos, igualando casi el tiempo de MUMPS. A partir de ahí, los tiempos crecen de manera exagerada e inadmisible, motivo por el cual no se incluyen para 24 y 32 hilos. Por el contrario, la única librería que consigue decrementar los tiempos de respuesta hasta alcanzar los 32 hilos de ejecución es PARDISO, la cual ofrece el mejor tiempo de ejecución, empleando 32 hilos, y los mejores incrementos de velocidad, a pesar de ser poco destacables.

Por otro lado, la figura 9.19 nos muestra como se reducen los tiempos de respuesta en la resolución del sistema de ecuaciones al incrementar de 1 a 32 el

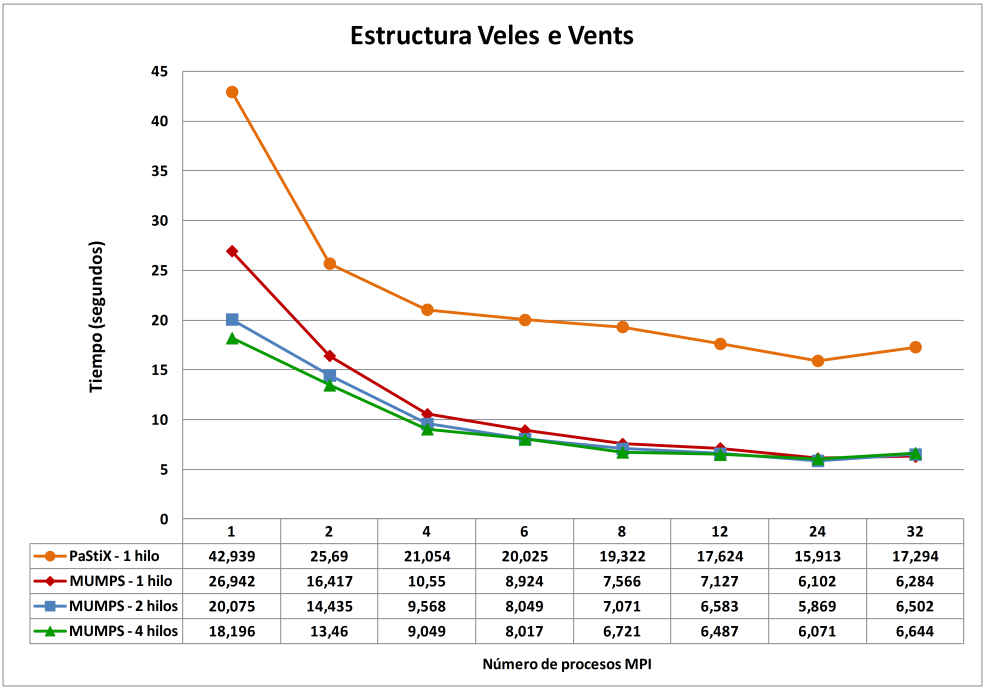
Método	Número de hilos								
	1	2	4	6	8	12	16	24	32
MUMPS	1.00	1.34	1.48	1.49	1.47	1.36	1.33	1.33	1.33
PaStiX	1.00	1.62	2.23	1.64	1.37	0.97	0.60	0.04	0.02
PARDISO	1.00	1.67	2.37	2.59	2.80	2.88	2.89	2.96	3.03

Tabla 9.6: Incrementos de velocidad en el cálculo en paralelo de los desplazamientos de la estructura Veles e Vents empleando múltiples hilos.

número de procesos MPI. En el caso de la librería MUMPS, hemos lanzado 1, 2 o 4 hilos de ejecución por cada proceso MPI. Como ya citamos con anterioridad, no es posible emplear en esta prueba a la librería PARDISO, ya que únicamente ofrece una paralelización a nivel de memoria compartida en la versión que forma parte de la librería MKL. Por otro lado, tampoco ha sido posible lanzar más de un hilo por cada proceso MPI en el caso de PaStiX. Ello es debido a que dicha librería necesita que la versión de MPI en la que se apoya sea de tipo MPI_THREAD_MULTIPLE, de modo que múltiples hilos puedan llamar conjuntamente a funciones de MPI, lo cual no ocurre con la versión de Open MPI instalada en la máquina Kahan, que es de tipo MPI_THREAD_SINGLE.

Como dicha figura nos indica, los tiempos de PaStiX son siempre superiores a los de MUMPS, la cual ha demostrado ser la librería más apropiada, reduciendo incluso los tiempos de respuesta en el caso de utilizar 2 o 4 hilos por proceso frente a emplear uno solo, para un buen número de procesos MPI. Tanto en el caso de PaStiX como en el de MUMPS, se ha empleado la librería METIS para reordenar la matriz de coeficientes. Será MUMPS por tanto la librería escogida para realizar, de ahora en adelante, el resto de simulaciones.

En términos de incrementos de velocidad (ver tabla 9.7 y figura 9.20) y de eficiencia (figura 9.21), observamos un bajo grado de escalabilidad tanto por parte de PaStiX como de MUMPS. Aunque MUMPS ofrece unas mejores prestaciones que PaStiX usando un único hilo, la mejora a la hora de usar más de un hilo resulta ser poco destacable. Conviene comentar que los incrementos de velocidad los hemos medido con respecto a la ejecución que emplea un solo proceso MPI y un solo hilo de ejecución. Por otro lado, las eficiencias se han calculado dividiendo el incremento de velocidad por el número de elementos de procesamiento empleados, obtenido como el producto del número de procesos MPI por el número de hilos



	1	2	4	6	8	12	24	32
PaStiX - 1 hilo	42,939	25,69	21,054	20,025	19,322	17,624	15,913	17,294
MUMPS - 1 hilo	26,942	16,417	10,55	8,924	7,566	7,127	6,102	6,284
MUMPS - 2 hilos	20,075	14,435	9,568	8,049	7,071	6,583	5,869	6,502
MUMPS - 4 hilos	18,196	13,46	9,049	8,017	6,721	6,487	6,071	6,644

Figura 9.19: Tiempos invertidos en el cálculo en paralelo de los desplazamientos de la estructura Veles e Vents empleando múltiples procesos MPI y 1 o más hilos.

de ejecución.

Como configuración de la librería MKL en las distintas pruebas realizadas, hemos mantenido las variables de entorno MKL_DYNAMIC igual a TRUE, de modo que sea la misma librería la que decida el número más apropiado de hilos de ejecución a lanzar en sus propias funciones de acuerdo a la dimensión del problema, y la variable OMP_NESTED igual a FALSE, con el objetivo de que la librería MKL sólo use un hilo de ejecución cuando sea invocada desde una región paralela de nuestro programa.

Método	Número de procesos MPI							
	1	2	4	6	8	12	24	32
PaStiX - 1 hilo	1	1.67	2.04	2.14	2.22	2.44	2.70	2.48
MUMPS - 1 hilo	1	1.64	2.55	3.02	3.56	3.78	4.42	4.29
MUMPS - 2 hilos	1.34	1.87	2.82	3.35	3.81	4.09	4.59	4.14
MUMPS - 4 hilos	1.48	2.00	2.98	3.36	4.01	4.15	4.44	4.06

Tabla 9.7: Incrementos de velocidad en el cálculo en paralelo de los desplazamientos de la estructura Veles e Vents.

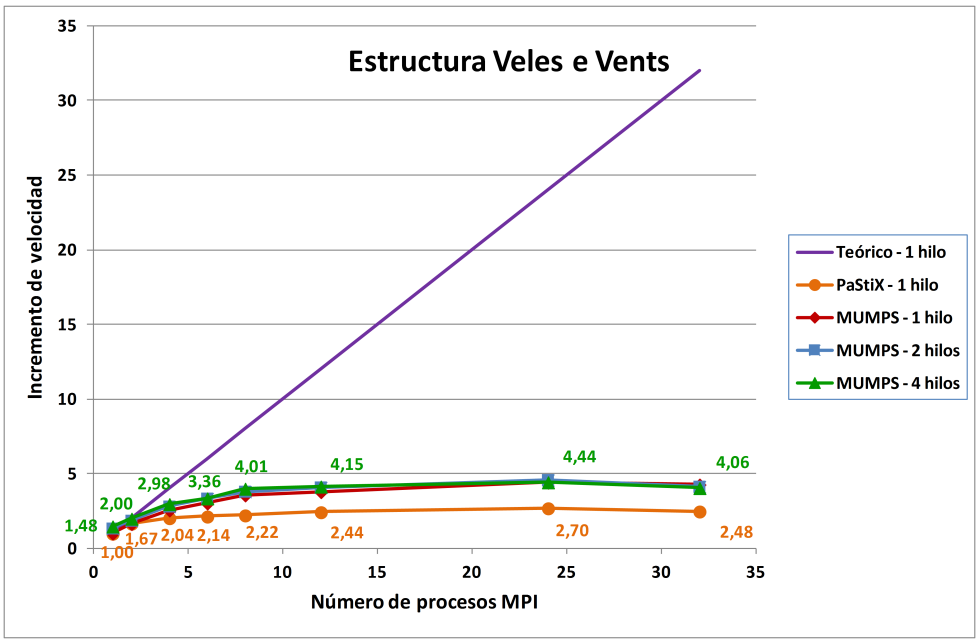
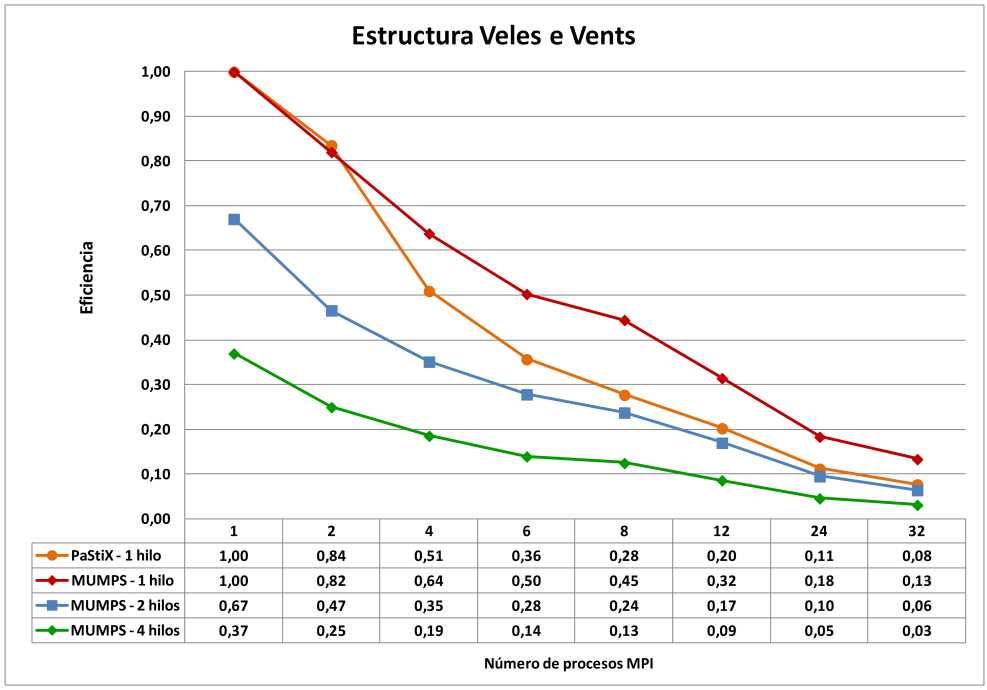


Figura 9.20: Incremento de velocidad en el cálculo en paralelo de los desplazamientos de la estructura Veles e Vents empleando múltiples procesos MPI y 1 o más hilos.



	1	2	4	6	8	12	24	32
PaStiX - 1 hilo	1,00	0,84	0,51	0,36	0,28	0,20	0,11	0,08
MUMPS - 1 hilo	1,00	0,82	0,64	0,50	0,45	0,32	0,18	0,13
MUMPS - 2 hilos	0,67	0,47	0,35	0,28	0,24	0,17	0,10	0,06
MUMPS - 4 hilos	0,37	0,25	0,19	0,14	0,13	0,09	0,05	0,03

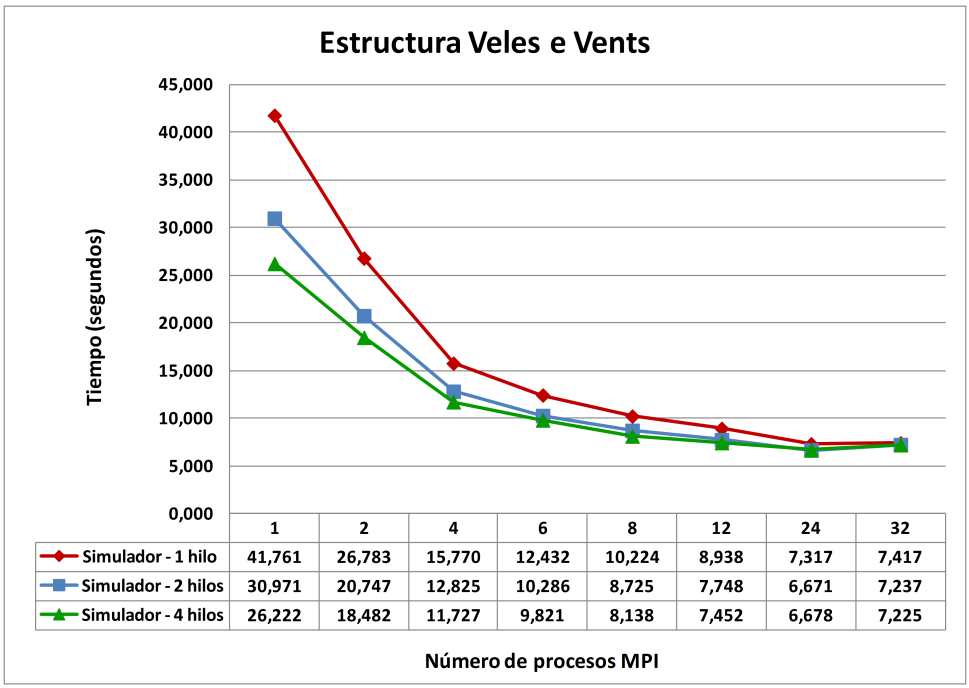
Figura 9.21: Eficiencia en el cálculo en paralelo de los desplazamientos de la estructura Veles e Vents empleando múltiples procesos MPI y 1 o más hilos.

Como hemos tenido ocasión de contemplar, la eficiencia de las simulaciones que utilizan múltiples hilos es muy reducida. Esto es así sobre todo en el caso de MUMPS, cuyo código incluye unas pocas directivas OpenMP de modo experimental y preliminar, delegando este aprovechamiento en una librería BLAS multihilo compatible con OpenMP, como la recogida en la librería MKL. Se observa además, en varias ocasiones, que es más adecuado planificar varios procesos MPI en el mismo nodo que utilizar el soporte multihilo (por ejemplo, 24 procesos MPI con un solo hilo frente a 6 procesos MPI con 4 hilos), ya que la paralelización a nivel de memoria distribuida abarca una mayor parte del código y tiene mayor repercusión en los tiempos de respuesta. No obstante, la escalabilidad de los procesos MPI está limitada, pudiendo apreciar un peor comportamiento para 32 procesos que para 24.

En lo que respecta a los tiempos invertidos por el Simulador Estructural al usar conjuntamente MPI y OpenMP, hay que decir que la figura 9.22 nos muestra el tiempo del cálculo estático de la estructura de Veles y Vents empleando de 1 a 32 procesos MPI, combinados con 1, 2 o 4 hilos de ejecución de OpenMP. En dicha gráfica aparece reflejado el tiempo dedicado a la generación de la matriz de rigidez y de los vectores de cargas, el cálculo de los desplazamientos en los nudos mediante MUMPS, la obtención de las reacciones en apoyos y el cálculo de las deformaciones unitarias, los esfuerzos y las tensiones en los nudos de los elementos finitos. Como podemos observar, los tiempos de simulación se van reduciendo a medida que se incrementa el número de procesos MPI y el número de hilos de ejecución, obteniendo el tiempo inferior al emplear 24 procesos MPI y 2 hilos.

Por su parte, podemos apreciar en la tabla 9.8 y en la figura 9.23, en primer lugar, el incremento de velocidad en el cálculo estático de la estructura, además de la eficiencia obtenida, en la figura 9.24. Dichos resultados están directamente influenciados por aquellos relativos a la resolución del sistema de ecuaciones, aunque son ligeramente mejores debido a un aumento en el incremento de velocidad y en la eficiencia presentes en el resto de etapas de cálculo.

Más detalladamente, las figuras 9.25, 9.26 y 9.27 recogen, respectivamente, el tiempo dedicado a la generación de la matriz de rigidez, además del incremento de velocidad y la eficiencia obtenida. Por su parte, las figuras 9.28, 9.29 y 9.30 nos muestran el tiempo invertido en el cálculo de las deformaciones unitarias,



	1	2	4	6	8	12	24	32
Simulador - 1 hilo	41,761	26,783	15,770	12,432	10,224	8,938	7,317	7,417
Simulador - 2 hilos	30,971	20,747	12,825	10,286	8,725	7,748	6,671	7,237
Simulador - 4 hilos	26,222	18,482	11,727	9,821	8,138	7,452	6,678	7,225

Figura 9.22: Tiempo de simulación estática de la estructura de Veles e Vents empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.

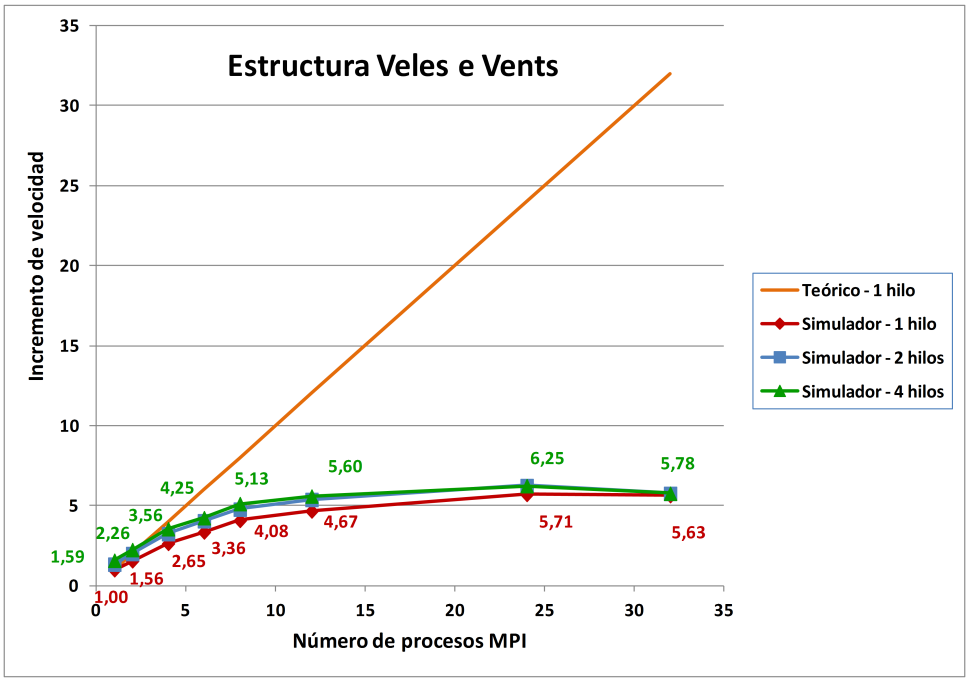


Figura 9.23: Incremento de velocidad en la simulación estática de la estructura de Veles e Vents empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.

Método	Número de procesos MPI							
	1	2	4	6	8	12	24	32
Simulador - 1 hilo	1	1.56	2.65	3.36	4.08	4.67	5.71	5.63
Simulador - 2 hilos	1.35	2.01	3.26	4.06	4.79	5.39	6.26	5.77
Simulador - 4 hilos	1.59	2.26	3.56	4.25	5.13	5.60	6.25	5.78

Tabla 9.8: Incrementos de velocidad en la simulación estática de la estructura Veles e Vents.

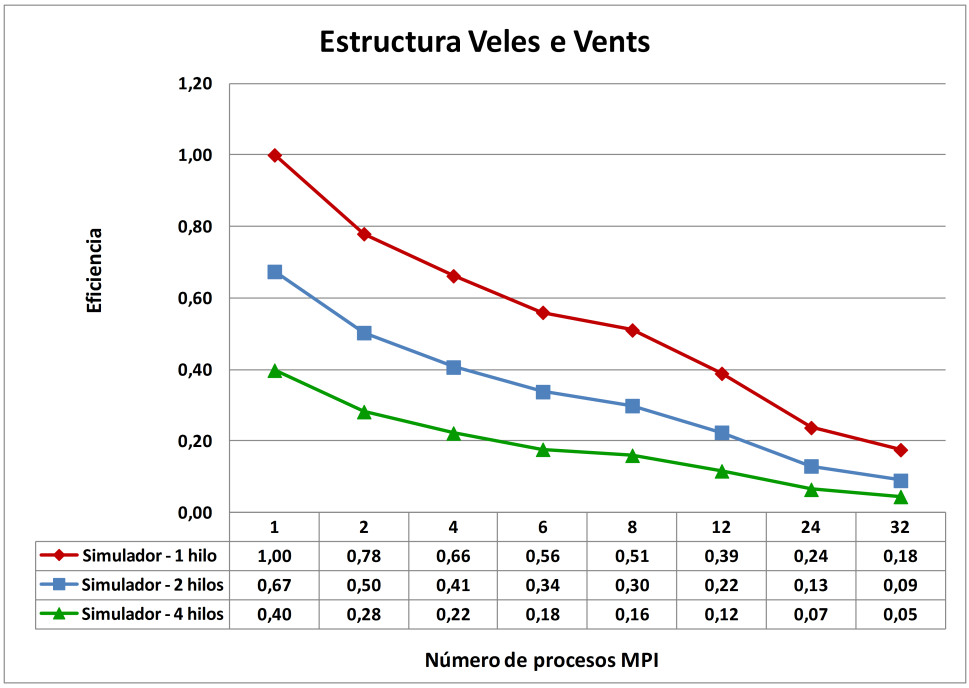
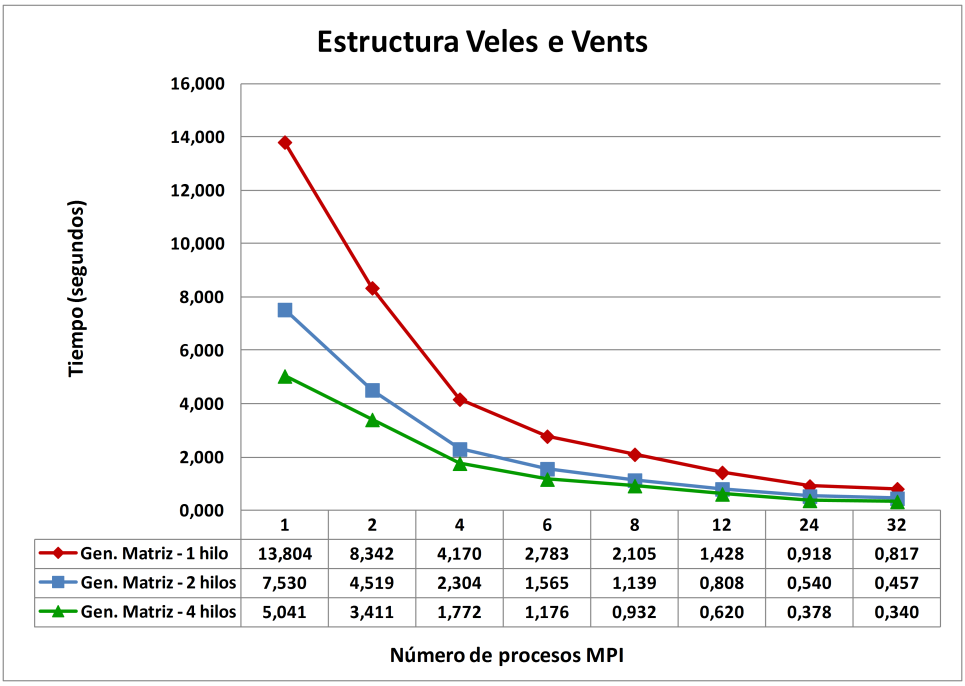


Figura 9.24: Eficiencia en la simulación estática de la estructura de Veles e Vents empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.



	1	2	4	6	8	12	24	32
Gen. Matriz - 1 hilo	13,804	8,342	4,170	2,783	2,105	1,428	0,918	0,817
Gen. Matriz - 2 hilos	7,530	4,519	2,304	1,565	1,139	0,808	0,540	0,457
Gen. Matriz - 4 hilos	5,041	3,411	1,772	1,176	0,932	0,620	0,378	0,340

Figura 9.25: Tiempo de generación de la matriz de rigidez correspondiente a la estructura de Veles e Vents empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.

los esfuerzos y las tensiones en los nudos de los elementos finitos, además del incremento de velocidad logrado y la eficiencia. Como es evidente, tanto la generación de la matriz como el cálculo de las deformaciones unitarias, los esfuerzos y las tensiones proporcionan mejores prestaciones paralelas que el cálculo de los desplazamientos.

9.1.5. Análisis dinámico

9.1.5.1. Análisis mediante métodos de integración directa

Pretendemos analizar dinámicamente, mediante el método de integración directa denominado Generalizado-α, la estructura de Veles e Vents, empleando para ello el acelerograma de la figura 9.31 correspondiente al terremoto que ocurrió en Turquía en el año 1999. El comportamiento dinámico de la estructura a lo largo del tiempo lo simularemos durante 15 segundos, empleando incrementos de

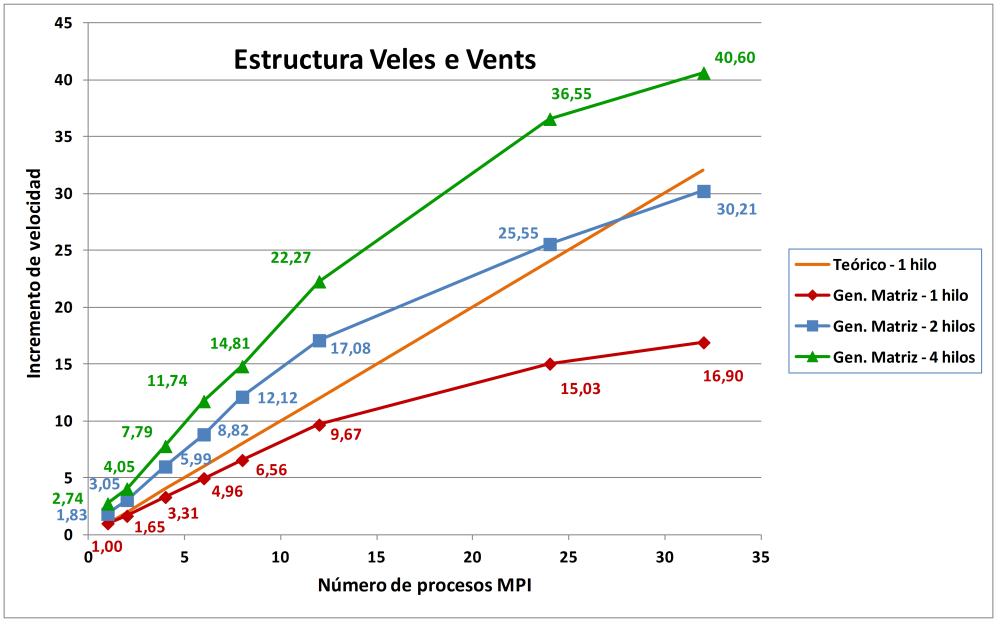
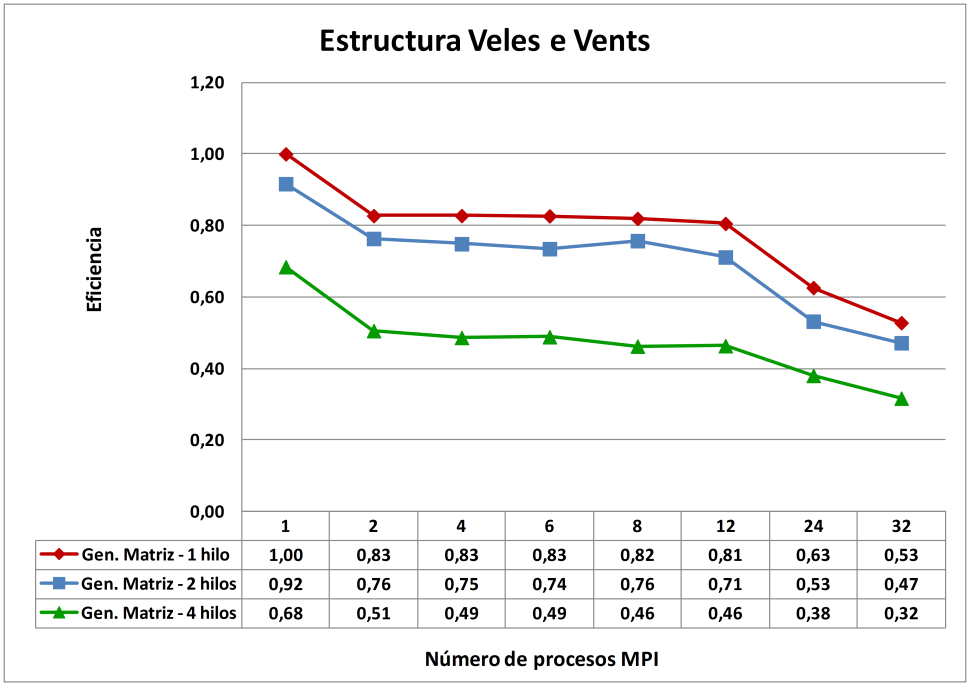
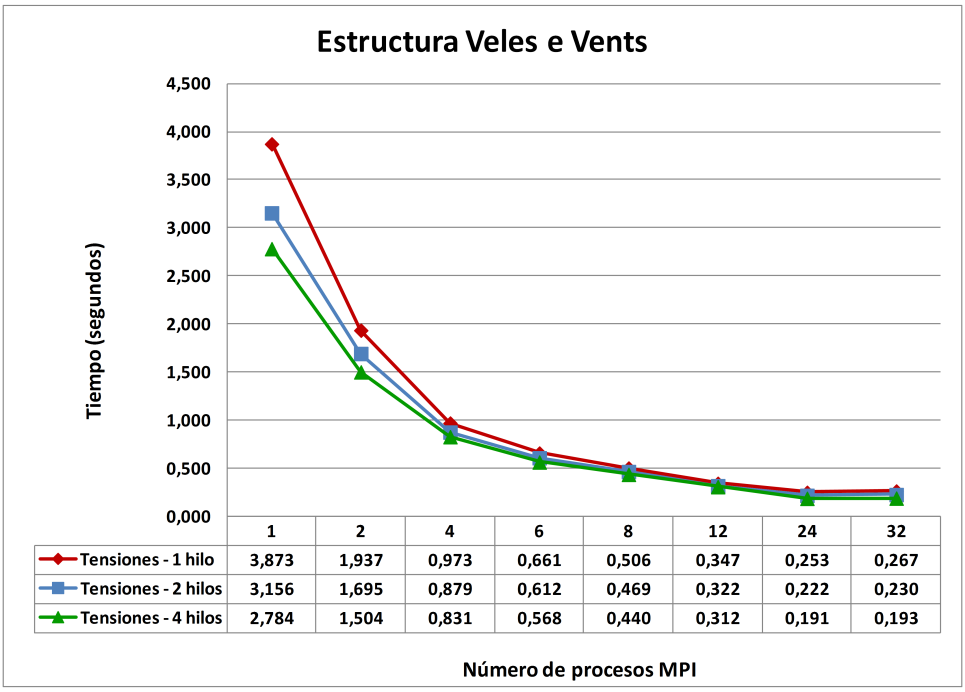


Figura 9.26: Incremento de velocidad en la generación de la matriz de rigidez correspondiente a la estructura de Veles e Vents empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.



	1	2	4	6	8	12	24	32
Gen. Matriz - 1 hilo	1,00	0,83	0,83	0,83	0,82	0,81	0,63	0,53
Gen. Matriz - 2 hilos	0,92	0,76	0,75	0,74	0,76	0,71	0,53	0,47
Gen. Matriz - 4 hilos	0,68	0,51	0,49	0,49	0,46	0,46	0,38	0,32

Figura 9.27: Eficiencia en la generación de la matriz de rigidez correspondiente a la estructura de Veles e Vents empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.



	1	2	4	6	8	12	24	32
Tensiones - 1 hilo	3,873	1,937	0,973	0,661	0,506	0,347	0,253	0,267
Tensiones - 2 hilos	3,156	1,695	0,879	0,612	0,469	0,322	0,222	0,230
Tensiones - 4 hilos	2,784	1,504	0,831	0,568	0,440	0,312	0,191	0,193

Figura 9.28: Tiempo de cálculo de las deformaciones unitarias, los esfuerzos y las tensiones correspondiente a la estructura de Veles e Vents empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.

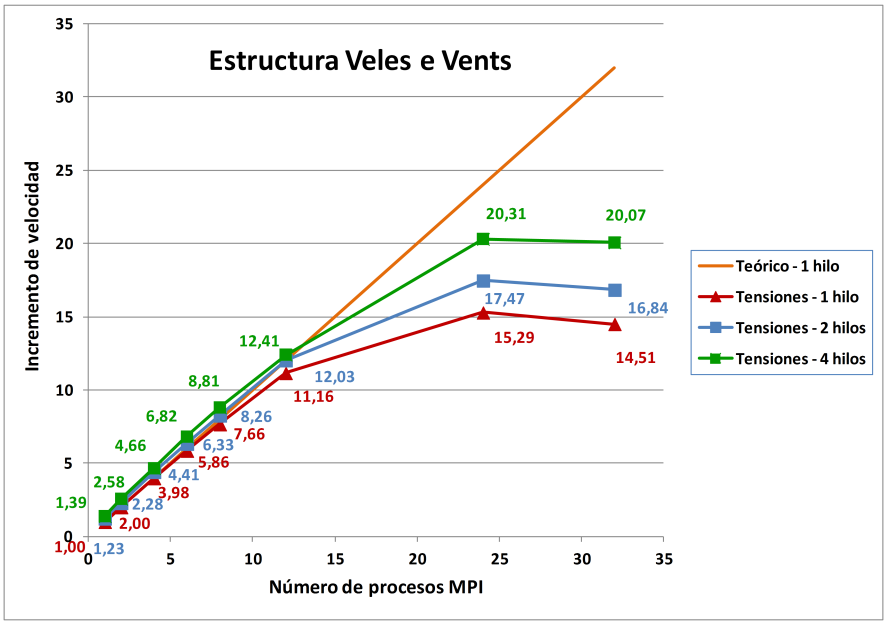
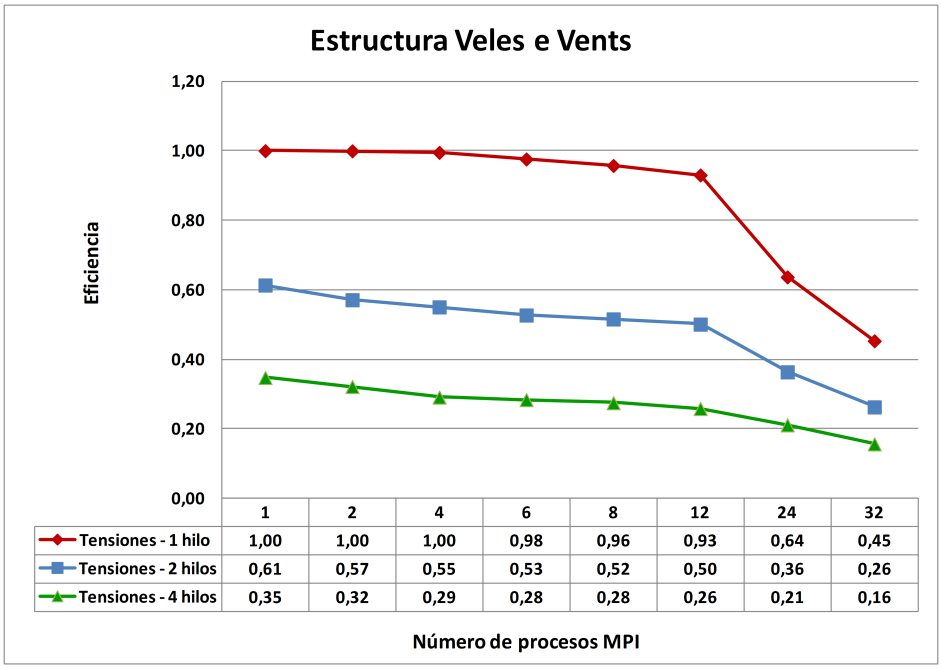


Figura 9.29: Incremento de velocidad en el cálculo de las deformaciones unitarias, los esfuerzos y las tensiones correspondiente a la estructura de Veles e Vents empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.



	1	2	4	6	8	12	24	32
Tensiones - 1 hilo	1,00	1,00	1,00	0,98	0,96	0,93	0,64	0,45
Tensiones - 2 hilos	0,61	0,57	0,55	0,53	0,52	0,50	0,36	0,26
Tensiones - 4 hilos	0,35	0,32	0,29	0,28	0,28	0,26	0,21	0,16

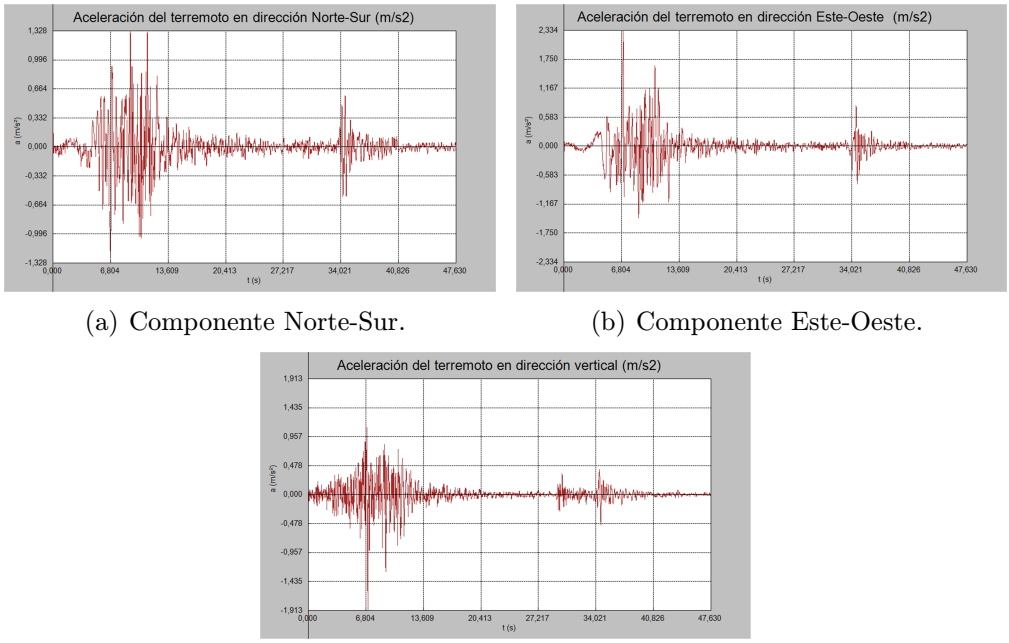
Figura 9.30: Eficiencia en el cálculo de las deformaciones unitarias, los esfuerzos y las tensiones correspondiente a la estructura de Veles e Vents empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.

tiempo de 0.01 segundos, lo que supone un total de 1500 pasos de simulación.

La figura 9.32 almacena el tiempo de la fase inicial de cálculo, compuesto por la generación de las matrices de rigidez, masa, amortiguamiento y la generación y la triangularización de la matriz de rigidez efectiva mediante MUMPS, junto con el incremento de velocidad y la eficiencia recogido por las figuras 9.33 y 9.34.

A su vez, la figura 9.35 recoge los tiempos medios para cada paso de tiempo, en los cuales calculamos los desplazamientos, las velocidades y las aceleraciones en los nudos, las reacciones en los apoyos y las deformaciones unitarias, los esfuerzos y las tensiones en los nudos de los elementos finitos. Como podemos observar, el tiempo de ejecución se reduce de 5.44 a 0.53 segundos, empleando 24 procesos MPI y 4 hilos. Por su parte, las figuras 9.36 y 9.37 nos muestran los resultados obtenidos del incremento de velocidad y la eficiencia para cada paso de tiempo.

Conviene aclarar que los tiempos de respuesta que obtendríamos con el resto de los métodos de integración que hemos desarrollado serían muy similares, a excepción del método SDIRK, que duplicaría dicho tiempo para esta estructura



(a) Componente Norte-Sur.

(b) Componente Este-Oeste.

(c) Componente vertical.

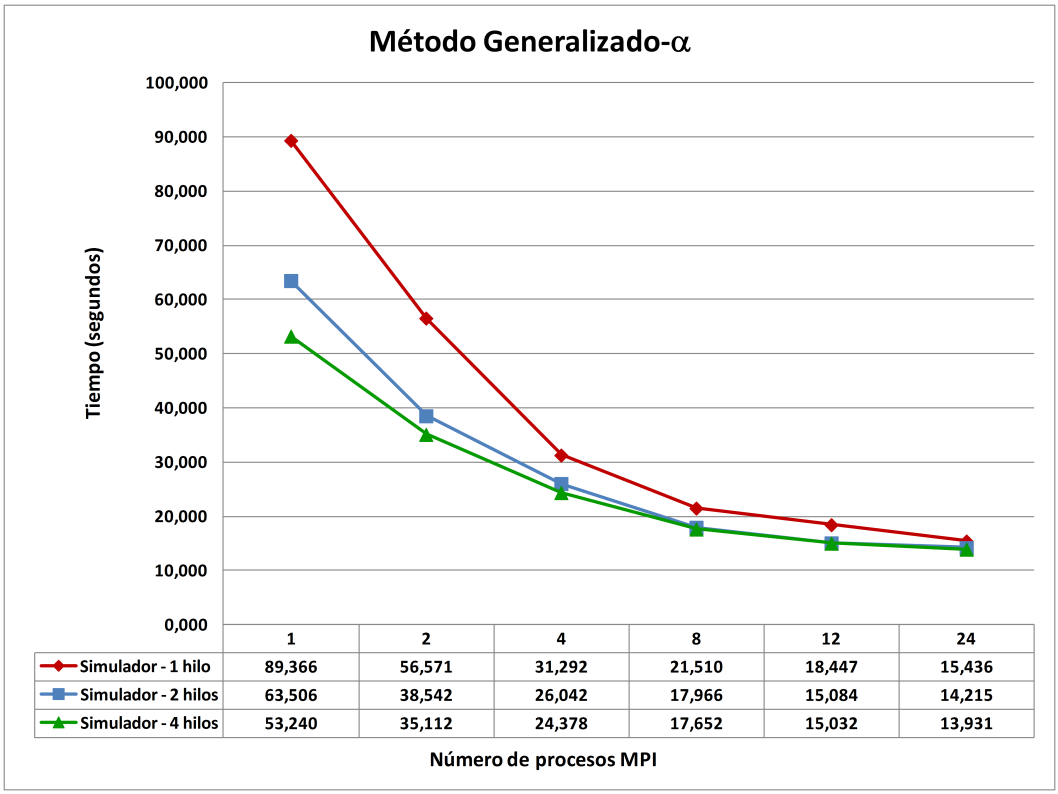
Figura 9.31: Acelerograma del terremoto ocurrido en Turquía en el año 1999.

en particular, suponiendo que empleáramos la variante de 4 etapas y obtuviéramos además las aceleraciones en los nudos previa resolución de los sistemas de ecuaciones triangulares pertinentes.

Por último, las figuras 9.38, 9.39 y 9.40 nos muestran el tiempo total de la simulación dinámica de la estructura empleando entre 1 y 24 procesos MPI y 1, 2 o 4 hilos por cada proceso, además del incremento de velocidad obtenido y la eficiencia. En dichos resultados no están considerados los tiempos dedicados a la escritura de los ficheros de disco. El tiempo de simulación de la estructura pasa por tanto de 137.41 a 13.55 minutos, empleando para ello 24 procesos MPI y 4 hilos.

9.1.5.2. Análisis modal

Para concluir esta fase experimental relacionada con el Simulador Estructural, hemos realizado un análisis modal de la estructura Veles e Vents, calculando sus 50 primeros modos de vibración mediante los métodos de Krylov-Schur y Arnoldi con reinicio explícito, presentes en la librería SLEPc, y Arnoldi con reinicio



	1	2	4	8	12	24
Simulador - 1 hilo	89,366	56,571	31,292	21,510	18,447	15,436
Simulador - 2 hilos	63,506	38,542	26,042	17,966	15,084	14,215
Simulador - 4 hilos	53,240	35,112	24,378	17,652	15,032	13,931

Figura 9.32: Tiempo inicial, en segundos, en el cálculo dinámico de Veles e Vents mediante el método Generalizado empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.

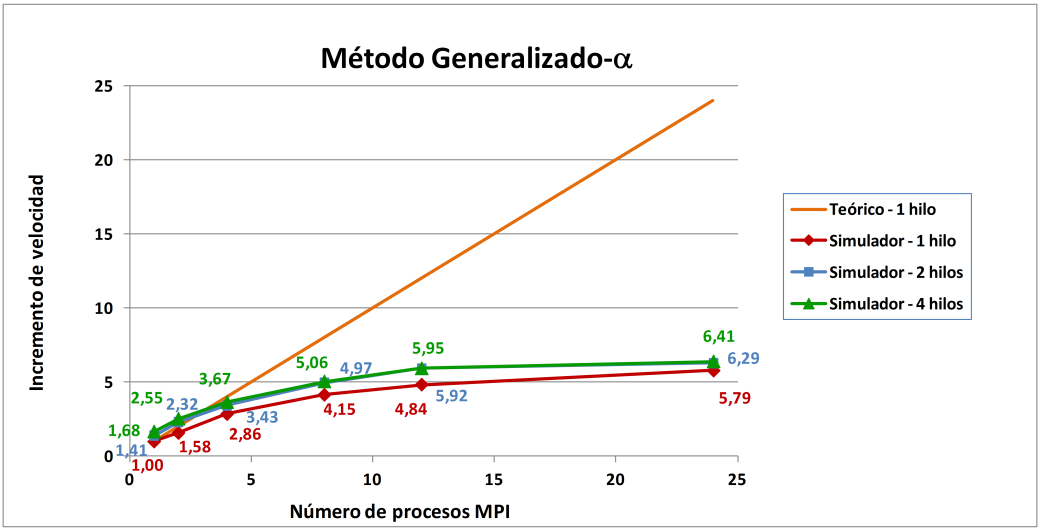
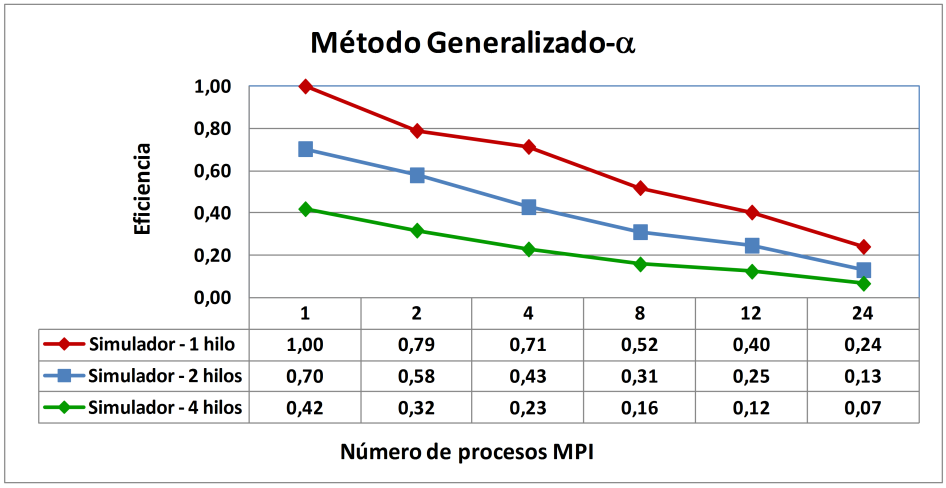
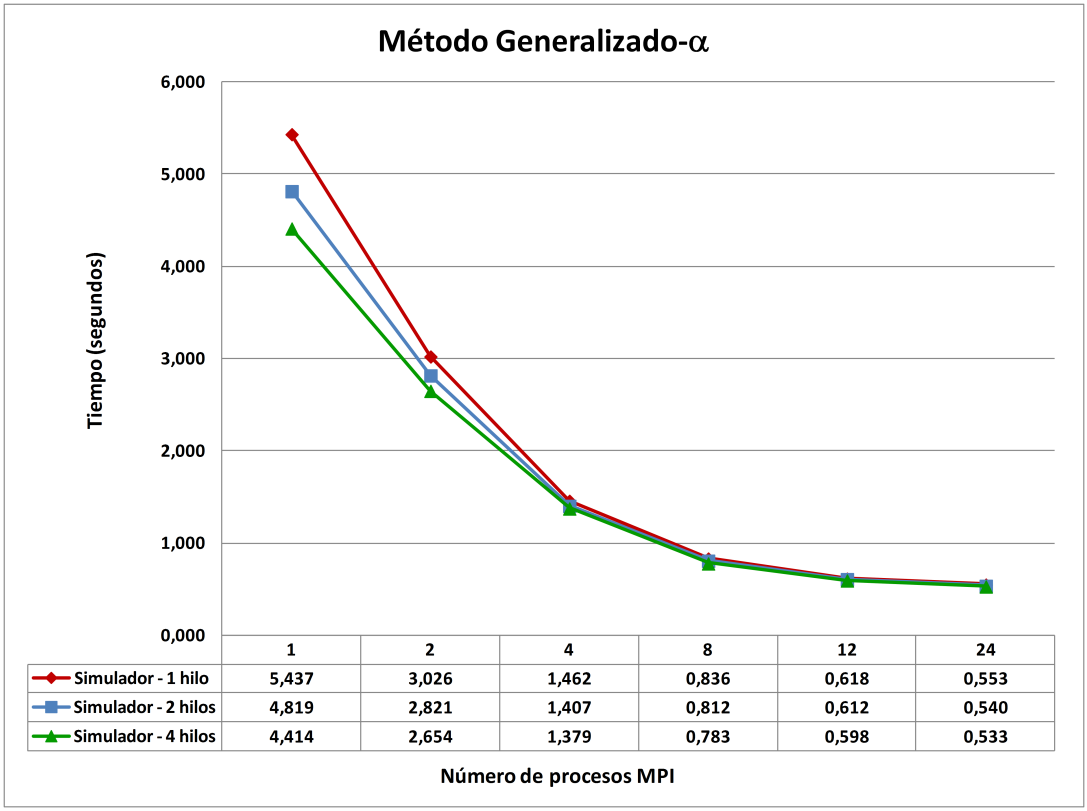


Figura 9.33: Incremento de velocidad en el tiempo inicial del cálculo dinámico de Veles e Vents mediante el método Generalizado empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.



	1	2	4	8	12	24
Simulador - 1 hilo	1,00	0,79	0,71	0,52	0,40	0,24
Simulador - 2 hilos	0,70	0,58	0,43	0,31	0,25	0,13
Simulador - 4 hilos	0,42	0,32	0,23	0,16	0,12	0,07

Figura 9.34: Eficiencia en el tiempo inicial del cálculo dinámico de Veles e Vents mediante el método Generalizado empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.



	1	2	4	8	12	24
Simulador - 1 hilo	5,437	3,026	1,462	0,836	0,618	0,553
Simulador - 2 hilos	4,819	2,821	1,407	0,812	0,612	0,540
Simulador - 4 hilos	4,414	2,654	1,379	0,783	0,598	0,533

Figura 9.35: Tiempo medio por iteración, en segundos, en el cálculo dinámico de Veles e Vents mediante el método Generalizado empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.

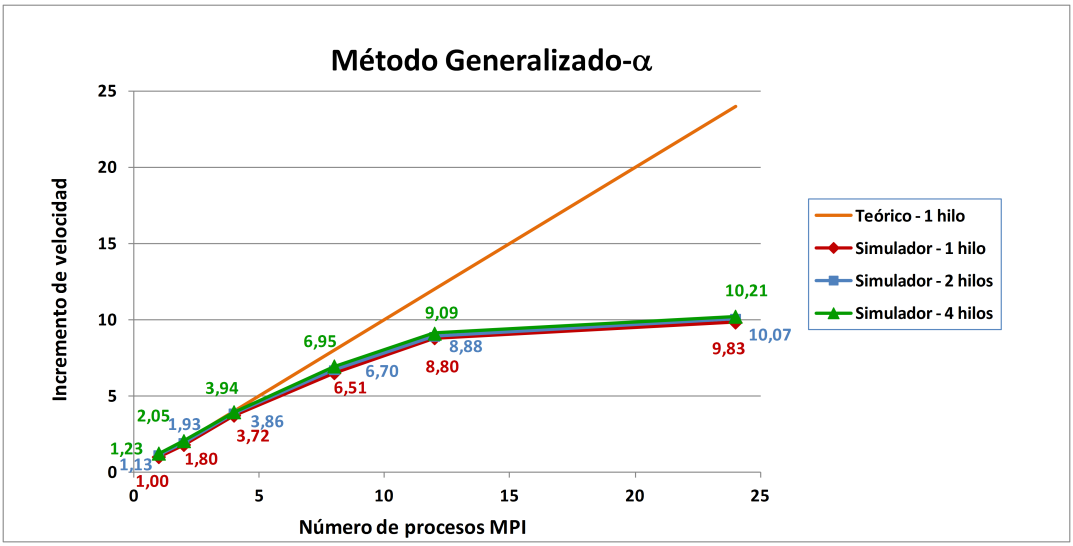
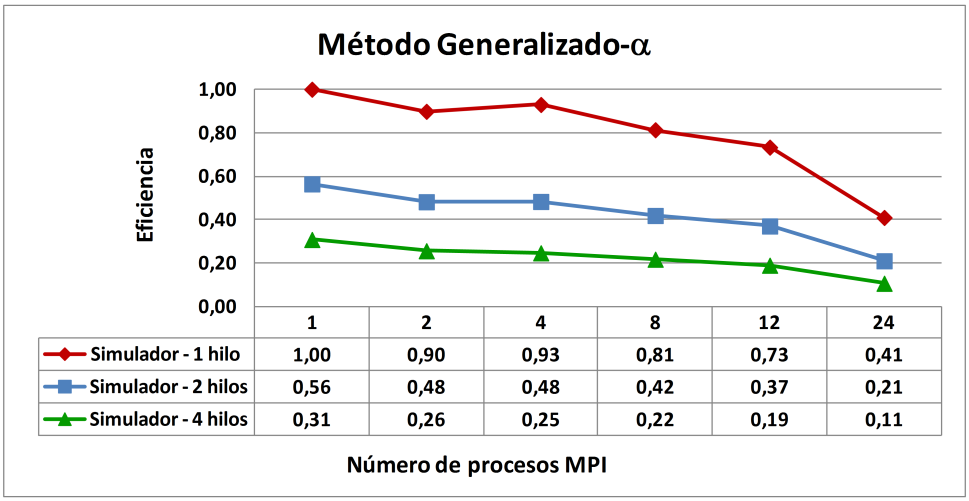
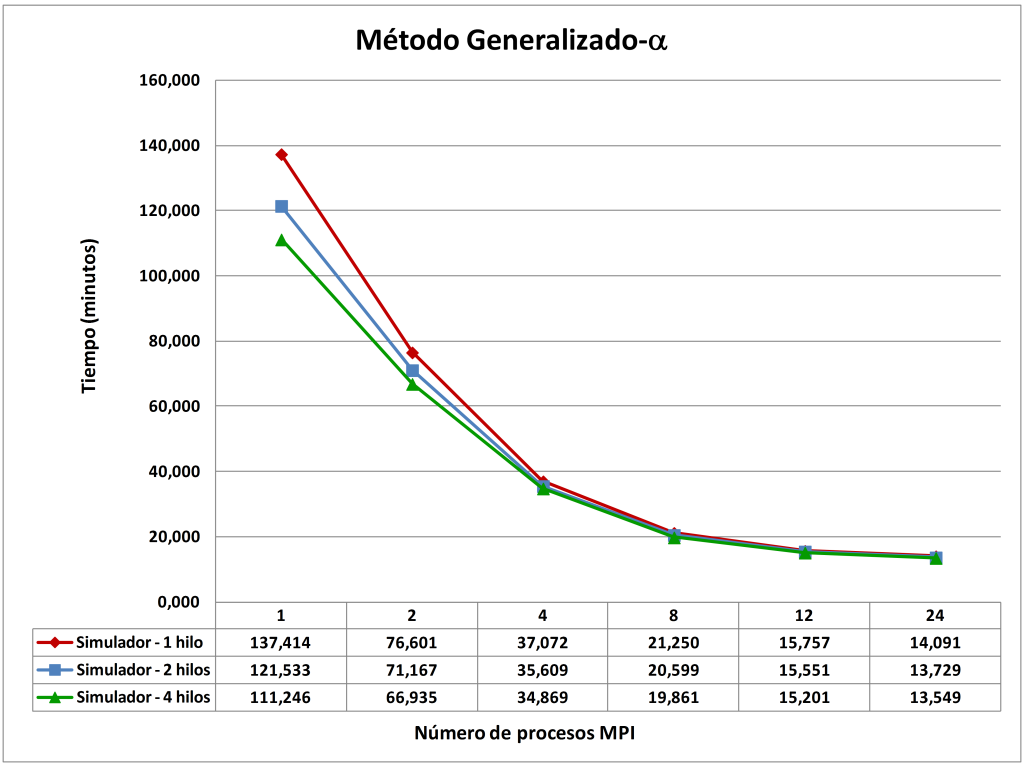


Figura 9.36: Incremento de velocidad, para cada instante de tiempo, del cálculo dinámico de Veles e Vents mediante el método Generalizado empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.



	1	2	4	8	12	24
Simulador - 1 hilo	1,00	0,90	0,93	0,81	0,73	0,41
Simulador - 2 hilos	0,56	0,48	0,48	0,42	0,37	0,21
Simulador - 4 hilos	0,31	0,26	0,25	0,22	0,19	0,11

Figura 9.37: Eficiencia, para cada instante de tiempo, del cálculo dinámico de Veles e Vents mediante el método Generalizado empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.



	1	2	4	8	12	24
Simulador - 1 hilo	137,414	76,601	37,072	21,250	15,757	14,091
Simulador - 2 hilos	121,533	71,167	35,609	20,599	15,551	13,729
Simulador - 4 hilos	111,246	66,935	34,869	19,861	15,201	13,549

Figura 9.38: Tiempo, en minutos, de la simulación dinámica de la estructura de Veles e Vents mediante el método Generalizado empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.

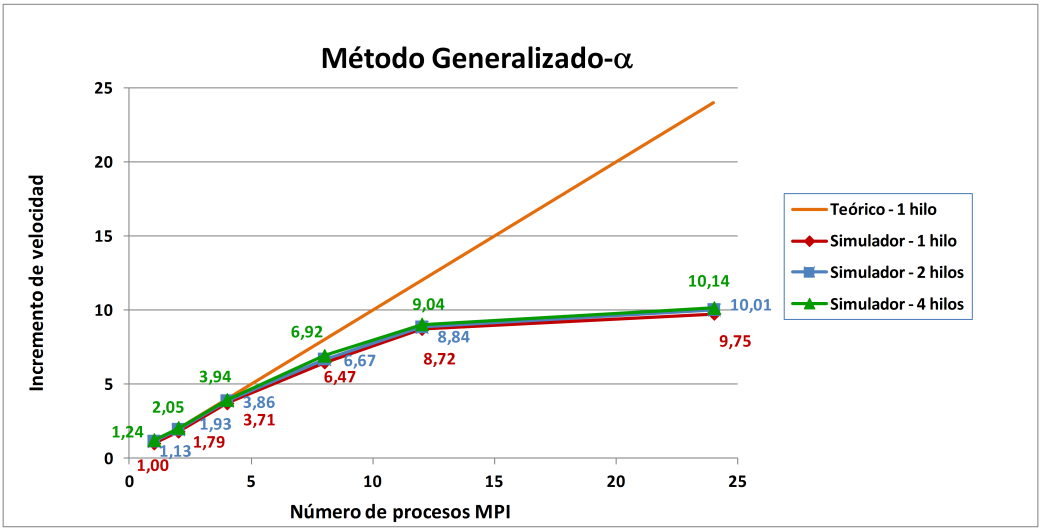
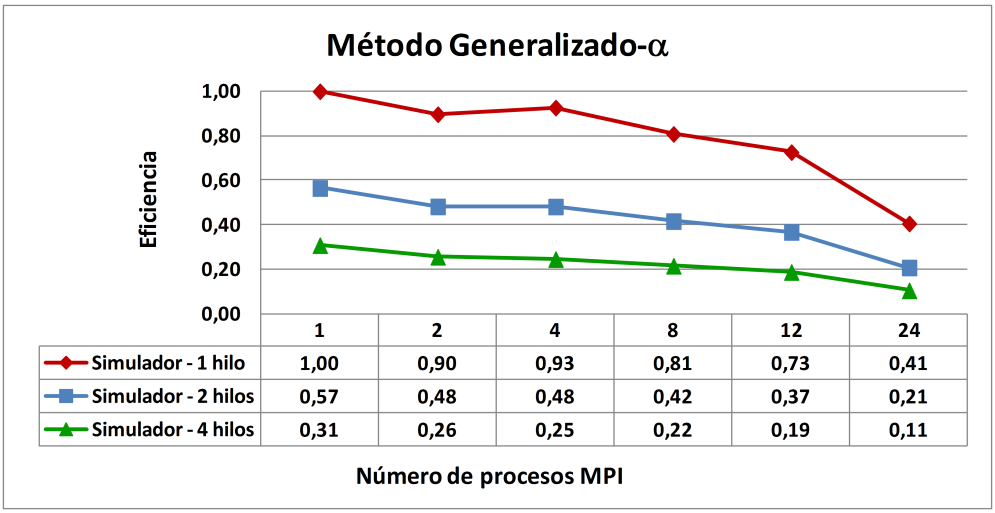


Figura 9.39: Incremento de velocidad en la simulación dinámica de la estructura de Veles e Vents mediante el método Generalizado empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.



	1	2	4	8	12	24
Simulador - 1 hilo	1,00	0,90	0,93	0,81	0,73	0,41
Simulador - 2 hilos	0,57	0,48	0,48	0,42	0,37	0,21
Simulador - 4 hilos	0,31	0,26	0,25	0,22	0,19	0,11

Figura 9.40: Eficiencia en la simulación dinámica de la estructura de Veles e Vents mediante el método Generalizado empleando múltiples procesos MPI y 1, 2 o 4 hilos de ejecución.

implícito, implementado en la librería ARPACK, todos ellos combinados con la transformación espectral de desplazamiento e inversión. Como resolución de los sistemas de ecuaciones lineales resultantes de aplicar la citada transformación espectral, hemos empleado la librería MUMPS. Cualquier otro método de cálculo de valores propios o técnica de trasformación espectral disponible en SLEPc, así como cualquier otro método de resolución de sistemas de ecuaciones lineales que forme parte de una librería diferente a MUMPS ha proporcionado tiempos superiores de ejecución.

La tabla 9.9 recoge los 10 primeros valores propios calculados y sus frecuencias naturales asociadas, además del error relativo Er calculado como:

$$Er = \left\| K\phi_i - \omega_i^2 M\phi_i \right\|_2 / \left\| \omega_i^2 \phi_i \right\|_2 \tag{9.1}$$

En las diferentes pruebas experimentales, se han empleado entre 1 y 24 procesos MPI, con 1, 2 y 4 hilos. Desafortunadamente, la versión multihilo apenas logra una ventaja sustancial con respecto a la aproximación de emplear un único hilo. Ello es debido a que las librerías PETSc y SLEPc no poseen, en sus últimas versiones, una paralelización a nivel de memoria compartida y los sistemas de ecuaciones triangulares que se resuelven con MUMPS en cada iteración tienen

Modo	Valor propio	Frecuencia Natural (Hz)	Error relativo
1	99.003999	1.583604	9.714844e-08
2	119.723277	1.741444	3.102101e-08
3	173.917774	2.098902	1.999275e-08
4	176.812017	2.116294	6.640812e-08
5	196.227077	2.229460	1.415815e-08
6	216.115816	2.339717	2.342394e-08
7	225.651672	2.390779	5.341030e-08
8	263.246888	2.582271	3.375073e-08
9	276.670772	2.647292	1.618135e-08
10	290.102400	2.710789	9.142914e-08

Tabla 9.9: Valores propios, frecuencias naturales y errores relativos cometidos en el análisis modal de la estructura Veles e Vents.

poca carga computacional para poder sacar provecho de las funciones multihilo de BLAS que forman parte de la librería MKL.

En todos los análisis realizados, la tolerancia de convergencia se ha fijado a 1e-8 y el desplazamiento inicial se ha correspondido con el resultado de aplicar la técnica descrita en la expresión (7.5), la cual proporciona un resultado igual a 0.29471, bastante alejado del primer valor propio. Conviene aclarar que con dicha técnica siempre hemos obtenido desplazamientos comprendidos entre 0 y el valor propio más pequeño, tanto en lo que respecta al análisis modal de esta estructura como de cualquier otra.

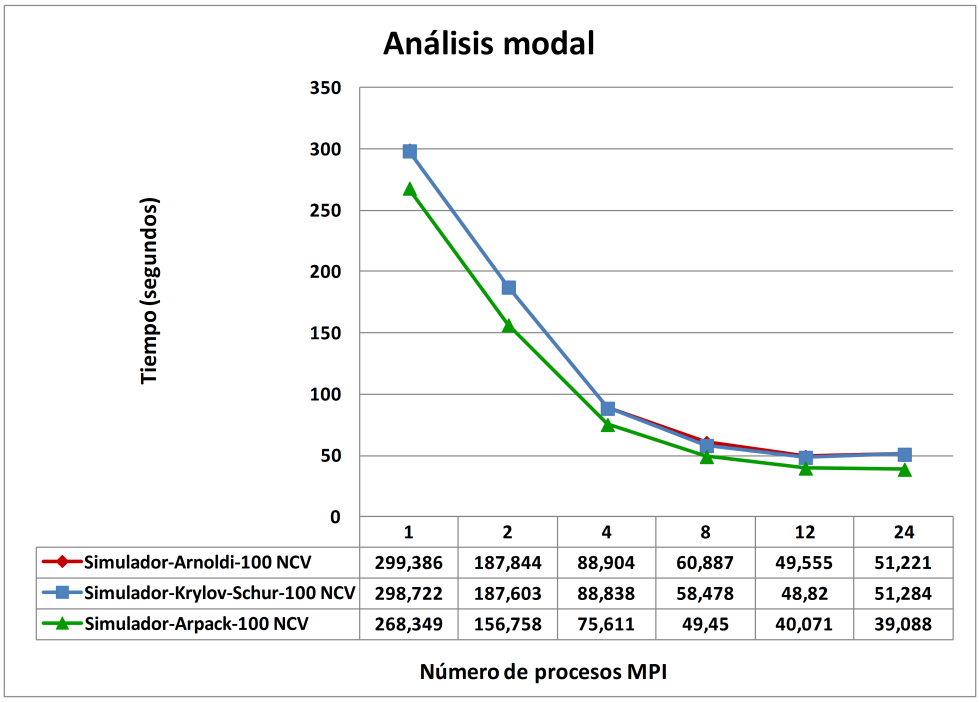
En la figuras 9.41, 9.42 y 9.43 aparecen los tiempos invertidos en la generación de las matrices de rigidez y masa y en el cálculo de las frecuencias naturales y sus modos de vibración correspondientes. Como podemos observar, hemos empleado 75 y 100 vectores columna (NCV), o longitud máxima del subespacio, para analizar su relevancia en los tiempos de respuesta dedicados al cálculo de los 50 valores y vectores propios mencionados. Si el número de vectores columna es suficientemente grande con respecto al número de valores propios a calcular, sólo serán necesarios unos pocos reinicios y la diferencia entre las variantes implícita y explícita desaparece, consumiendo en consecuencia más memoria, lo que podemos apreciar en la figura 9.41, donde los tiempos tras emplear Krylov-Schur o Arnoldi con reinicio explícito son idénticos y ligeramente superiores a los de Arnoldi con reinicio implícito.

Sin embargo, si el número de valores propios a calcular es elevado y el número de vectores columna es sólo ligeramente superior, la situación que se produce es la contraria, ya que son necesarios muchos reinicios hasta alcanzar la convergencia, de modo que los métodos con reinicio explícito pierden efectividad con respecto a los de reinicio implícito. Dicha situación es precisamente la que reproduce la figura 9.42, donde los tiempos del método de Arnoldi con reinicio explícito son muy superiores a los que presentan los otros dos métodos, basados en técnicas de reinicio implícito. Con todo ello, hay que decir que no es posible conocer de antemano el número óptimo de vectores columna, aunque un valor recomendable, por ejemplo, sería que dicho número fuera el doble que el de los valores propios a calcular. Un aspecto importante a recordar en el comportamiento típico de los métodos con reinicio explícito es que los valores propios convergen de uno en uno. En cambio, en los métodos con reinicio implícito, todos los valores propios van convergiendo simultáneamente.

Por último, la figura 9.43 compara los tiempos de ejecución, empleando 75 y 100 vectores columna, mediante los métodos de Krylov-Schur y Arnoldi con reinicio implícito, los cuales se ven poco afectados por dicho número, aunque siempre proporcionan tiempos de respuesta inferiores para 100 vectores columna. Como podemos observar, los mejores tiempos de ejecución, en el análisis modal de la estructura, se han obtenido gracias al método de Arnoldi con reinicio implícito.

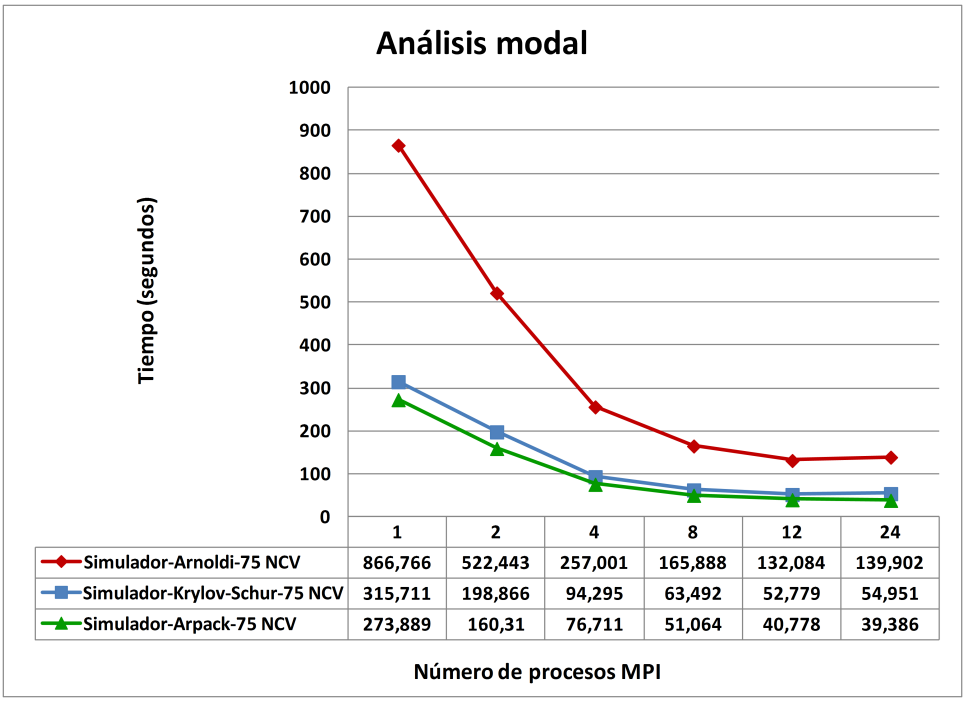
Por su parte, las figuras 9.44 y 9.45 detallan los incrementos de velocidad y la eficiencia al emplear los métodos de Krylov-Schur y Arnoldi con reinicio implícito con 100 vectores columna. Como podemos observar, los resultados son muy similares y siguen una idéntica tendencia, siendo ligeramente mejores en el caso de Arnoldi con reinicio implícito.

Finalmente, la tabla 9.10 almacena la suma acumulada de las masas efectivas, expresadas en forma de porcentaje de la masa efectiva total, de los 50 primeros modos de vibración calculados de la estructura Veles e Vents. Como podemos observar, la suma de las masas efectivas vale 76.607 %, no siendo superior al 90 % de la masa total movilizada como exige la NCSE-02 o el Eurocódigo 8, para lo cual deberíamos calcular hasta 80 modos de vibración, lo que supondría que la suma de masas efectivas sería igual a 90.004 % y que los tiempos de ejecución fueran de 460.13 segundos con Krylov-Schur y de 432.667 con Arnoldi con reinicio implícito,



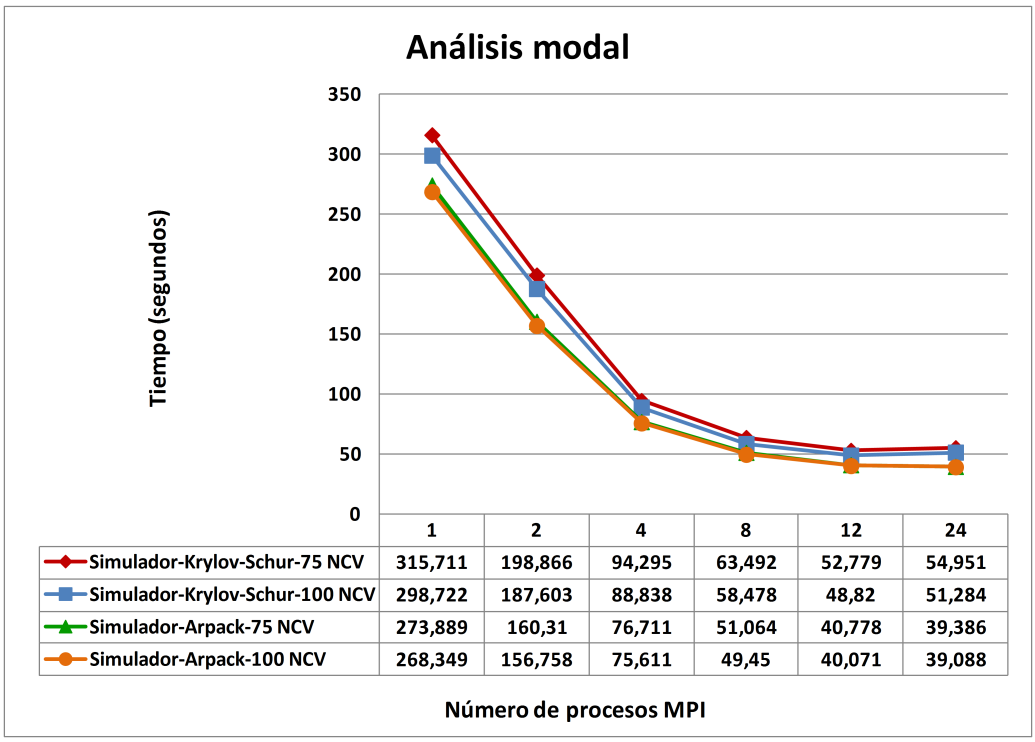
	1	2	4	8	12	24
Simulador-Arnoldi-100 NCV	299,386	187,844	88,904	60,887	49,555	51,221
Simulador-Krylov-Schur-100 NCV	298,722	187,603	88,838	58,478	48,82	51,284
Simulador-Arpack-100 NCV	268,349	156,758	75,611	49,45	40,071	39,088

Figura 9.41: Tiempo dedicado al análisis modal de la estructura de Veles e Vents empleando 100 vectores columna y múltiples procesos MPI.



	1	2	4	8	12	24
Simulador-Arnoldi-75 NCV	866,766	522,443	257,001	165,888	132,084	139,902
Simulador-Krylov-Schur-75 NCV	315,711	198,866	94,295	63,492	52,779	54,951
Simulador-Arpack-75 NCV	273,889	160,31	76,711	51,064	40,778	39,386

Figura 9.42: Tiempo dedicado al análisis modal de la estructura de Veles e Vents empleando 75 vectores columna y múltiples procesos MPI.



	1	2	4	8	12	24
Simulador-Krylov-Schur-75 NCV	315,711	198,866	94,295	63,492	52,779	54,951
Simulador-Krylov-Schur-100 NCV	298,722	187,603	88,838	58,478	48,82	51,284
Simulador-Arpack-75 NCV	273,889	160,31	76,711	51,064	40,778	39,386
Simulador-Arpack-100 NCV	268,349	156,758	75,611	49,45	40,071	39,088

Figura 9.43: Comparativa en el análisis modal de la estructura de Veles e Vents empleando 75 o 100 vectores columna y múltiples procesos MPI.

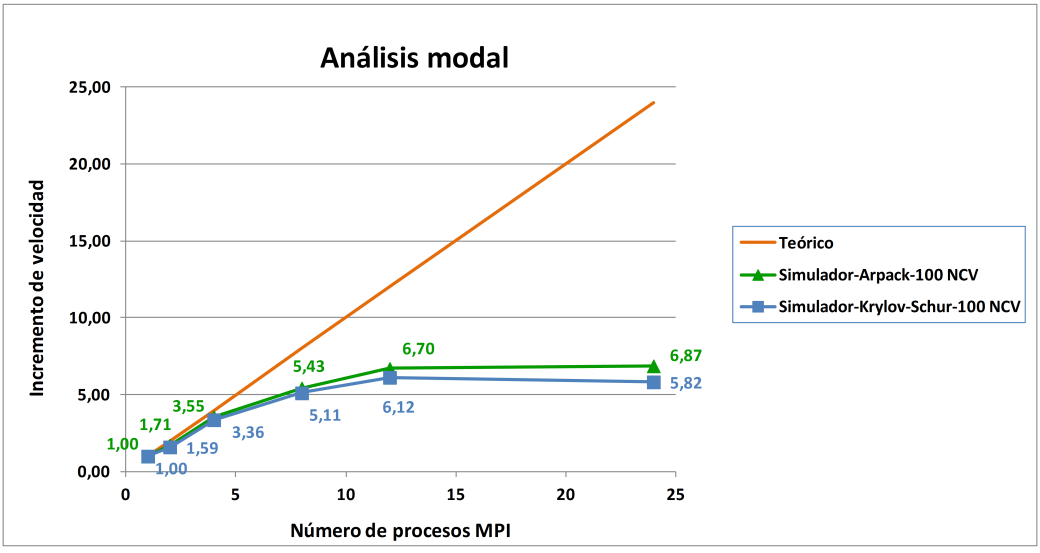


Figura 9.44: Incremento de velocidad en el análisis modal de la estructura de Veles e Vents empleando 100 vectores columna y múltiples procesos MPI.

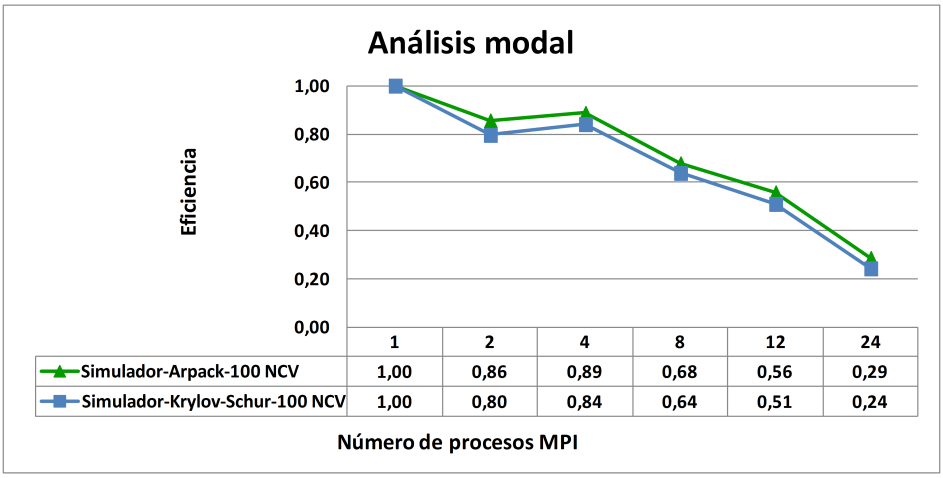


Figura 9.45: Eficiencia en el análisis modal de la estructura de Veles e Vents empleando 100 vectores columna y múltiples procesos MPI.

	Eje X	Eje Y	Eje Z	Global
%Meft acumulada	78.305 %	85.854 %	67.828 %	76.607 %

Tabla 9.10: Sumas acumuladas de las masas efectivas, como porcentaje de la masa efectiva total, para los 50 primeros modos de vibración de la estructura Veles e Vents.

empleando para ello un único proceso MPI y 160 vectores columna.

9.2. Resultados del Simulador Estructural sobre un PC tradicional

En primer lugar, pretendemos comparar en este apartado los tiempos de ejecución del Simulador Estructural frente a dos programas de cálculo de estructuras, como son el programa Angle, desarrollado por el profesor Adolfo Alonso Durá, codirector de esta tesis doctoral, empleado en tareas de investigación por parte del DMMCTE y el programa SAP2000 v.16 [339] desarrollado y comercializado a nivel mundial por la empresa Computers and Structures, Inc. (CSI).

Para realizar la comparativa, se han empleado las cuatro siguientes estructuras, cuyo número de nudos, grados de libertad, barras y elementos finitos que componen su mallado aparecen detallados en la tabla 9.11:

Estructura	Nudos	Gdl	Barras	EF 2D	EF 3D	Hipótesis
Kiev	29649	1174951	2793	27490	0	3
Rascacielos	182501	1094856	1900	175104	0	2
Veles e Vents	147599	883590	0	293474	0	1
San Juan Hospital	75227	262998	0	13809	53785	1

Tabla 9.11: Características de las estructuras empleadas en la comparativa entre diferentes programas de cálculo.

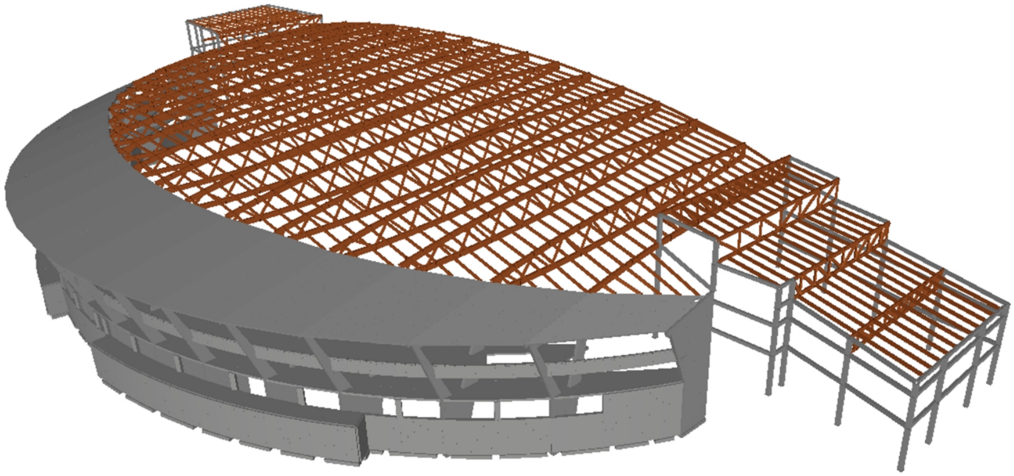


Figura 9.46: Modelización de la estructura Kiev.

- Kiev: Modelo de la estructura del parque acuático de la ciudad de Kiev (Ucrania), formado por barras y elementos finitos 2D, que aparece en la figura 9.46.

- Rascacielos: Edificio de plantas compuestas por losas de elementos finitos planos sobre pilares similar a la estructura Altura, descrita previamente, pero de mayor dimensión, al estar formada por 76 plantas (ver figura 9.47).

- Veles e Vents: Edificio del puerto de Valencia descrito en la sección anterior cuyo mallado se visualiza en la figura 9.48.

- San Juan del Hospital: Primera iglesia construida en Valencia tras la reconquista. Está compuesta por elementos finitos en dos y tres dimensiones (ver figura 9.49).

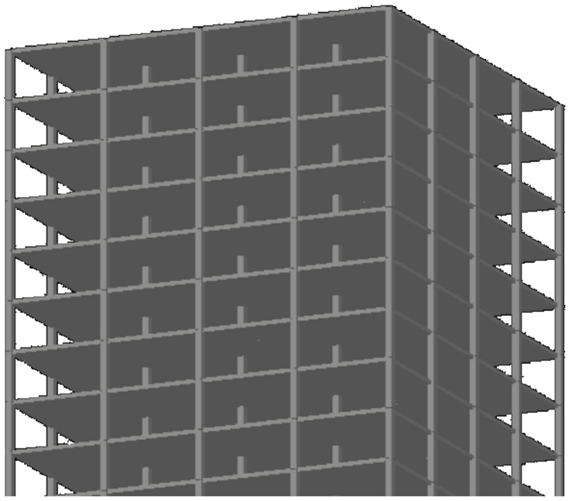


Figura 9.47: Modelización de las últimas plantas de la estructura Rascacielos.

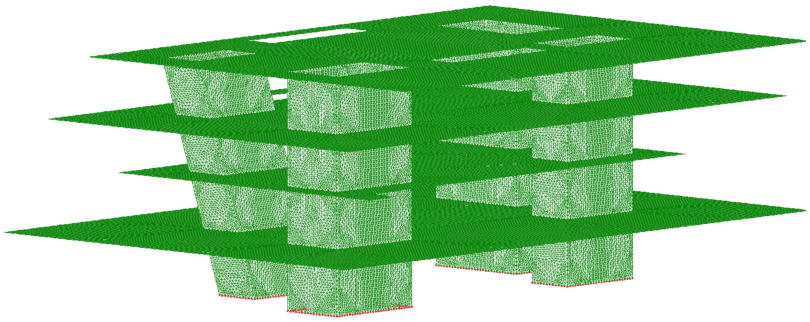


Figura 9.48: Mallado de la estructura Veles e Vents.

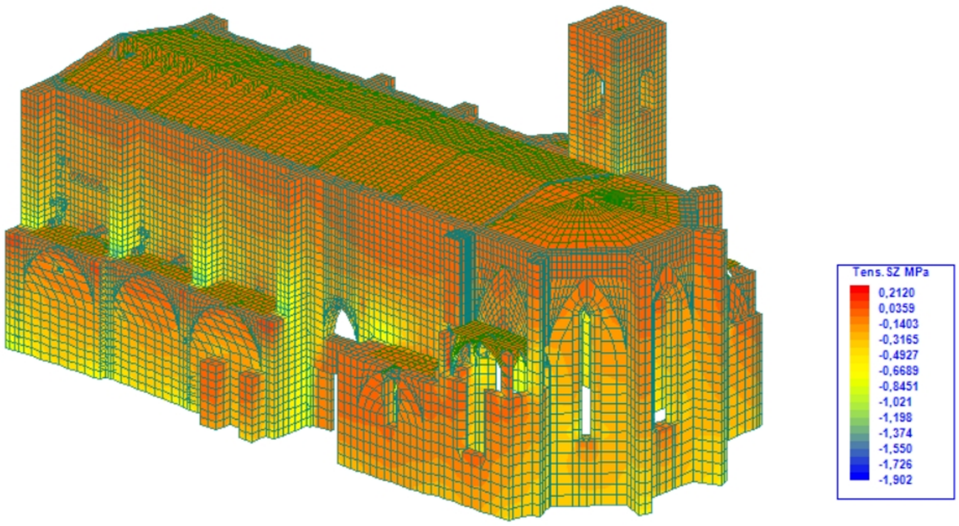


Figura 9.49: Mallado y visualización de resultados de la estructura de San Juan del Hospital.

La figura 9.50 incluye los tiempos del cálculo estático de las 4 estructuras mencionadas. Es importante destacar que el Simulador Estructural se ha utilizado únicamente en su versión secuencial, sin sacar provecho por tanto de ser ejecutado en paralelo con múltiples procesos MPI o con diferentes hilos de ejecución. Como podemos apreciar, dicho Simulador Estructural ofrece unos tiempos de cálculo muy competitivos frente a los otros dos programas de cálculo, siendo sobre todo apreciable la mejora sobre el programa SAP2000 en la estructura Rascacielos, compuesta por más de un millón de grados de libertad. Mientras que SAP2000 realiza el análisis de la estructura en 18 minutos y 29 segundos, el Simulador Estructural lo lleva a cabo en solo 34 segundos, siendo por tanto 32.6 veces más rápido, como podemos apreciar en la figura 9.51, donde se muestran los incrementos de velocidad del Simulador Estructural frente a los dos programas mencionados. Destacable resulta también el hecho de que nuestro Simulador Estructural permita trabajar con elementos finitos en 3D (tetraedros, hexaedros o prismas triangulares) a diferencia de SAP2000, que no calcula estructuras compuestas por tetraedros o prismas triangulares.

Todas las pruebas se han realizado sobre un PC tradicional compuesto por un procesador Intel i7 a 2.0 GHz (frecuencia máxima de 3.1 GHz con la tecnología

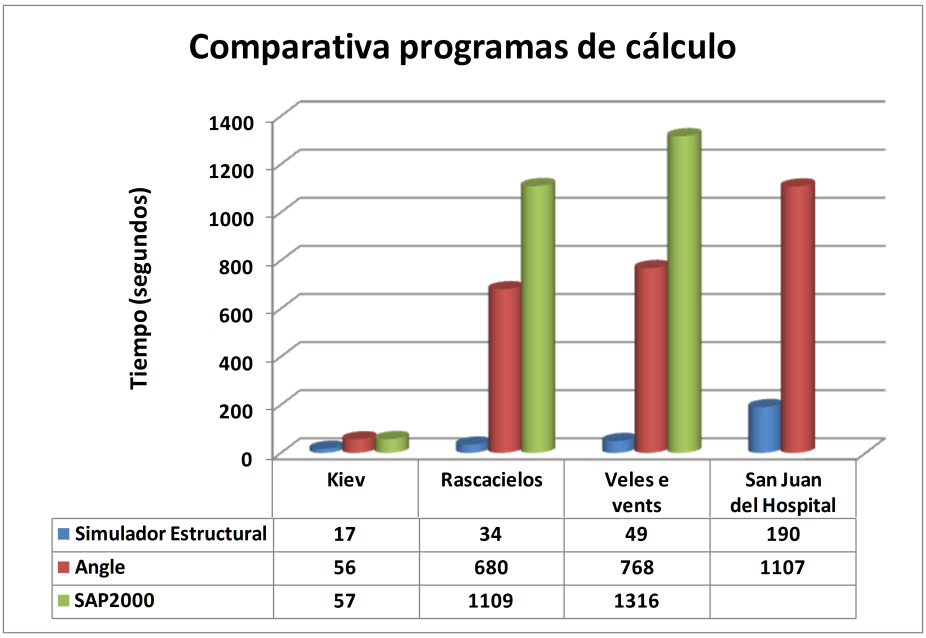


Figura 9.50: Comparativa de tiempos ante un cálculo estático entre el Simulador Estructural, Angle y SAP2000.

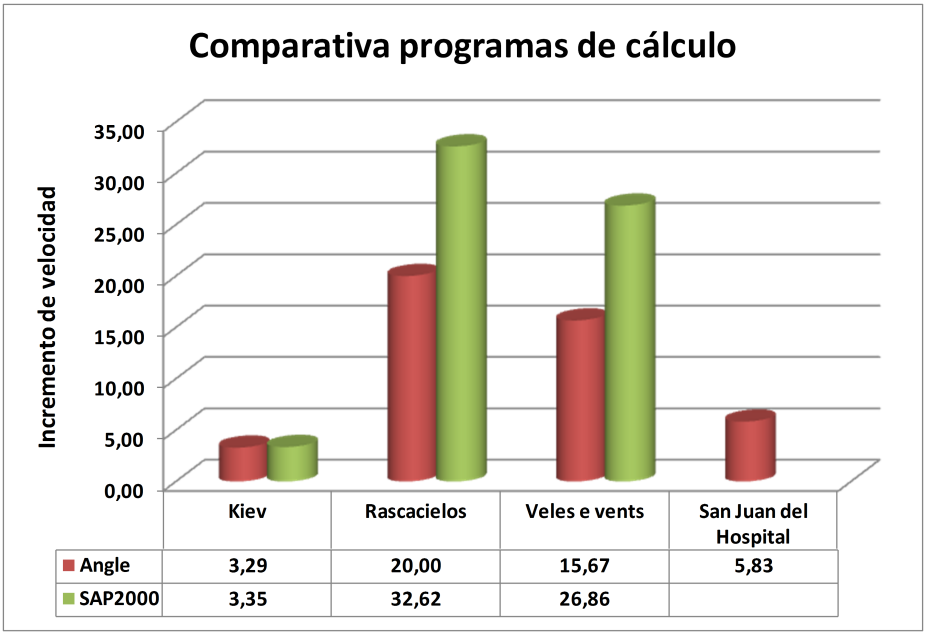


Figura 9.51: Incremento de velocidad del Simulador Estructural, ante un cálculo estático, frente a Angle y SAP2000.

Turbo Boost) con 16 GBytes de memoria RAM de tipo DDR3, bajo el sistema operativo Windows 8.

9.3. Diseño estructural en una infraestructura Grid computacional

Con el objetivo de evaluar los beneficios de la utilización de una infraestructura de computación distribuida, basada en tecnologías Grid, en el dominio del cálculo dinámico de estructuras, se ha realizado la ejecución de un caso de estudio mediante el Servicio Grid de Cálculo Estructural desarrollado. Dicho caso de estudio ha consistido en el diseño de un edificio de viviendas recogido en la figura 9.52 del que se han considerado 8 soluciones estructurales diferentes, a fin de encontrar la más óptima de todas ellas. Cada uno de los diseños estructurales está compuesto por unos 17300 nudos, esto es, unos 103800 grados de libertad y alrededor de 35000 barras. Teniendo en cuenta la Normativa de Construcción Sismorresistente Española (NCSE-02), cada una de las diferentes soluciones preliminares fue simulada bajo la influencia de 5 terremotos representativos y diferentes, de acuerdo a la localización geográfica del edificio. Esto dio lugar, por tanto, a un total de 40 simulaciones diferentes.

Los acelerogramas empleados presentaban valores correspondientes a la aceleración del terreno cada 0.01 segundos, con una duración de 5 a 10 segundos. Para obtener la respuesta estructural a lo largo del tiempo, empleamos el método Generalizado-α, basado como bien sabemos en técnicas de integración directa. El tiempo total de simulación se fijó a 5 segundos en todos los análisis, empleando pasos de tiempo en la integración de 0.01 segundos y grabando los resultados cada 0.5 segundos. Los sistemas de ecuaciones lineales se resolvieron mediante la librería MUMPS. Cada ejecución necesitó un total de 417 MBytes de memoria RAM y generó un total de 82.73 MBytes de resultados, lo cual dio lugar a que el caso de estudio generara un total de 3.2 GBytes de ficheros de salida.

Para la ejecución de este caso de estudio, empleamos una infraestructura Grid compuesta por 3 clusters de PCs pertenecientes a nuestro grupo de investigación, cuyas características principales se detallan en la tabla 9.12. Los recursos com-

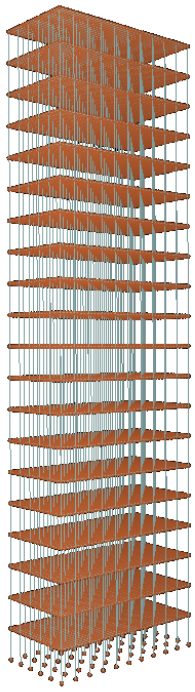


Figura 9.52: Edificio de viviendas analizado en la plataforma Grid.

Máquina	Procesadores	Velocidad	Memoria
Seker	Intel Pentium 4 Xeon	2.8 GHz	4 GBytes
Ramses	12 Intel Pentium III biprocesadores	866 MHz	512 MBytes
Kefren	20 Intel Pentium Xeon biprocesadores	2.0 GHz	1 GByte
Odin	55 Intel Pentium Xeon biprocesadores	2.8 GHz	2 GBytes

Tabla 9.12: Características de la infraestructura Grid.

putacionales estaban conectados por una red de área local de 100 Mbits/seg. y en todos ellos se instaló Globus Toolkit. Aunque la utilización de una infraestructura distribuida con recursos de múltiples centros resultaría, a primera vista, más atractiva, la única diferencia destacable residiría en la cantidad de tiempo involucrado en las transferencias de datos.

La tabla 9.13 resume el proceso de asignación de tareas. En ella se indica el número inicial de nodos de computación disponibles en cada cluster al inicio de la ejecución del caso de estudio, así como el número de tareas asignadas a cada uno de ellos. La máquina llamada Seker, en la que se desplegó el servicio Grid, estaba situada en la misma red de área local que los clusters empleados. A priori,

Máquina	Procesadores inicialmente disponibles	Tareas asignadas
Ramses	10	10
Kefren	19	17
Odin	53	13

Tabla 9.13: Distribución de las simulaciones en la infraestructura Grid.

quizá cabría pensar que Odin, la máquina más potente, recibiría la práctica totalidad de las simulaciones. Sin embargo, la política de planificación implementada en el servicio Grid trata de distribuir las ejecuciones en los diferentes recursos, reduciendo el impacto causado en caso de fallo de una máquina. Se trata además de una estrategia de planificación amigable, que evita sobrecargar una única máquina con la mayoría de las ejecuciones, lo que iría en detrimento del resto de usuarios.

De media, cada ejecución en Odin requirió unos 11.37 segundos para llevar a cabo la fase de envío de datos, 530.96 segundos para simular la estructura y 88.36 segundos para recoger los resultados.

El proceso de simulación de todas las soluciones estructurales requirió 34 minutos, desde el comienzo de la planificación hasta que se completó la transferencia de los datos de salida de la última simulación a la máquina que alberga el servicio. Una ejecución secuencial de todo el caso de estudio, ejecutando una simulación tras otra empleando un único procesador del cluster Odin, duró un total de 354 minutos (casi 6 horas), con lo cual la aproximación basada en Grid fue 10.41 veces más rápida que la ejecución tradicional.

Consideremos un cluster de PCs de tamaño medio compuesto por 8 nodos de cálculo, lo que puede ser bastante habitual en un centro de investigación. Así, la ejecución del caso de estudio realizando dos ejecuciones simultáneas con 4 procesos MPI, en el cluster Odin, precisó de 63.73 minutos. Por lo tanto, la aplicación basada en Grid fue casi 2 veces más rápida que la aproximación basada en Computación de Altas Prestaciones.

Obviamente, el tiempo involucrado en la transferencia de datos supone el mayor handicap de la aplicación diseñada. En ese sentido, es importante reducir al máximo la cantidad de información generada por el Simulador y emplear técnicas avanzadas de compresión de datos. Si los resultados de una misma solución

Nudos	Barras	Elementos finitos 2D	Grados de libertad
42302	2657	68787	253812

Tabla 9.14: Características estructurales del Banco Nórdico.

estructural ante diferentes terremotos se combinaran en el Grid, reduciríamos considerablemente la cantidad de datos que tienen que ser recuperados.

En cualquier caso, la mejora en los tiempos de ejecución del caso de estudio completo es siempre dependiente de la cantidad y de las características de los recursos computacionales de los que disponga la infraestructura Grid, de la velocidad de la red de interconexión, de la dimensión de la estructura, del tipo de análisis a llevar a cabo y de la cantidad de datos a ser descargados a la máquina del usuario.

9.4. Simulación estructural mediante los servicios Cloud

Si bien en el apartado siguiente ejecutaremos un caso de estudio compuesto por múltiples simulaciones con la intención de obtener el diseño más apropiado de una estructura mediante el servicio Cloud basado en Generic Worker, en este caso llevaremos a cabo la ejecución remota de una única simulación correspondiente a obtener la respuesta de una estructura ante la acción de un sismo, usando para ello lo servicios Cloud basados en Generic Worker y en COMPSs.

La estructura a analizar consiste en una interpretación del Banco Nórdico, situado en Helsinki, obra del arquitecto Alvar Aalto y construido en el año 1962, recogida en la figura 9.53. Consiste en una estructura compuesta por losas macizas sobre pilares interiores de hormigón y pilares exteriores de acero y una losa de cimentación, debido a la presencia de estrato freático. A nivel estructural, está compuesta por barras para modelizar vigas y columnas y elementos finitos en 2D en la modelización de losas y muros, cuyo número viene indicado en la tabla 9.14.

Para obtener el comportamiento dinámico de dicha estructura a lo largo del

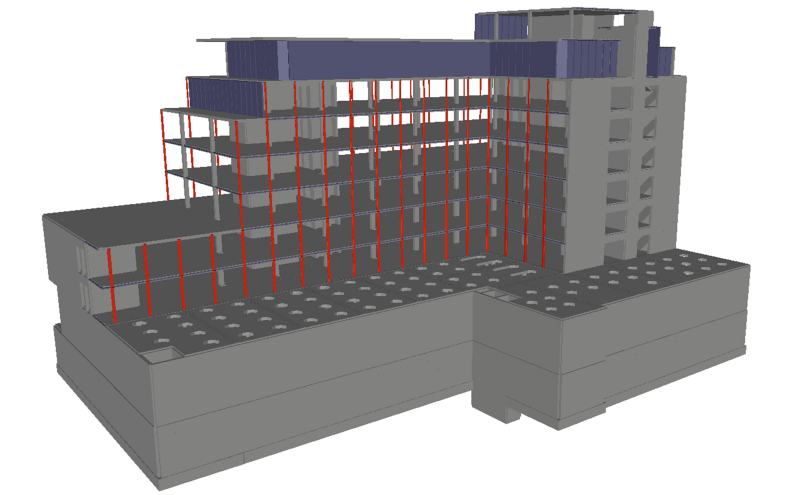


Figura 9.53: Banco Nórdico calculado dinámicamente en los servicios Cloud.

tiempo, aplicamos un acelerograma de 12 segundos de duración, con un incremento de tiempo en la simulación de 0.01 segundos, grabando los resultados en disco cada 0.05 segundos, lo que requirió 2.48 GBytes para almacenar los ficheros de resultados. Como plataformas computacionales, empleamos:

- A nivel local, un PC compuesto por un procesador Intel Core i5 a 3.20 GHz y 4 GBytes de memoria RAM.
- En la infraestructura de Microsoft Azure, una instancia de tipo medio (B2), formada por una CPU de doble núcleo a 1.60 GHz y 3.5 GBytes de memoria RAM, accesible mediante el servicio Cloud basado en Generic Worker.
- En el despliegue del BSC y por medio del Servicio Cloud basado en PMES COMPSs, una máquina virtual de 8 núcleos computacionales sobre una máquina física compuesta por 2 AMD Opteron 6140 de 8 núcleos cada una a 2.6 GHz y 32 GBytes de memoria RAM.

La figura 9.54 muestra los tiempos de respuesta de la simulación bajo las 3 plataformas citadas. Eso incluye por tanto los tiempos de envío y recogida de resultados de todos los instantes de tiempo generados y, en el caso de PMES COMPSs, el tiempo de despliegue de las dos máquinas virtuales. En la infraestructura Cloud del BSC, el Simulador Estructural se lanzó empleando 1, 2 y 4 tareas MPI empleando la técnica de particionado por corte de elementos natural, las cuales se

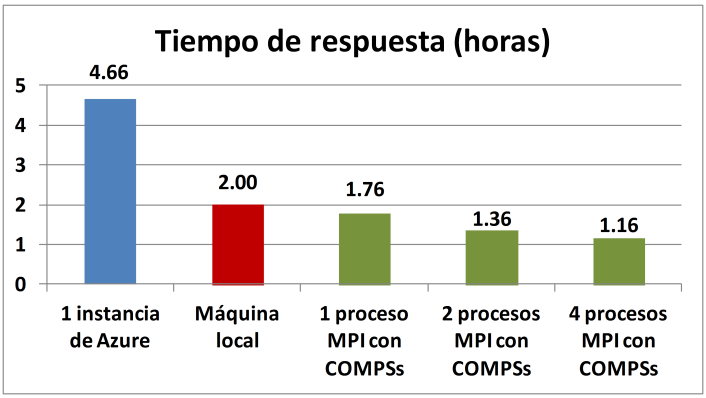


Figura 9.54: Tiempos de respuesta del Banco Nórdico en 3 plataformas distintas.

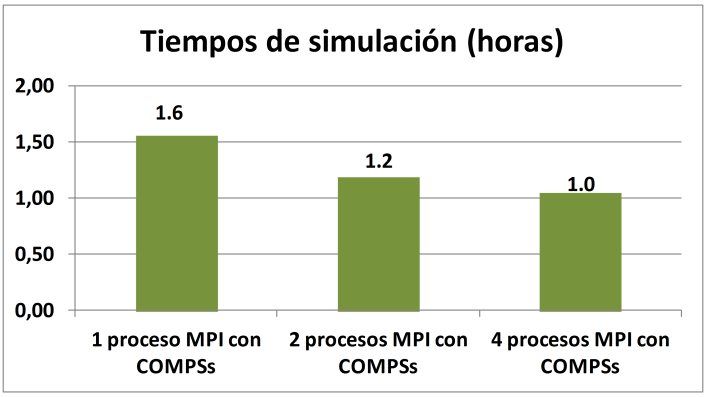


Figura 9.55: Tiempos de simulación del Banco Nórdico en la plataforma de PMES COMPSs.

ejecutaron sobre una misma máquina virtual, junto con el Gestor del Servicio de Análisis Estructural. El método de integración directa empleado fue el de Generalizado-α. Como podemos observar, los tiempos de respuesta mediante el servicio Cloud de Microsoft Azure son mucho mayores a los tiempos obtenidos al ejecutar a nivel local o por medio del servicio Cloud basado en PMES COMPSs. Básicamente, la diferencia del tiempo de simulación entre las 3 plataformas es debido a las características particulares de cada una de ellas, especialmente en el tipo y frecuencia de la CPU y del tiempo de acceso a disco.

Finalmente, la figura 9.55 nos muestra los tiempos invertidos por el Simulador Estructural, empleando un número diferente de procesos MPI en la plataforma Cloud gestionada por PMES COMPSs.

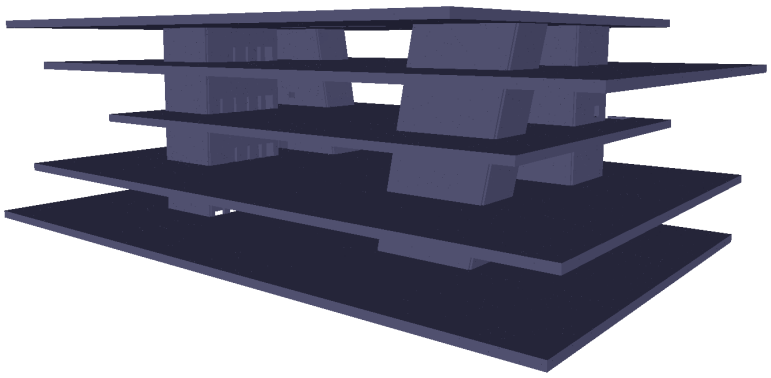


Figura 9.56: Edificio Veles e Vents, incluyendo el parking.

Nudos	Grados de libertad	Barras	Elementos finitos 2D
120238	721428	-	237892

Tabla 9.15: Características estructurales del edificio Veles e Vents.

9.5. Diseño estructural en la plataforma Cloud de Microsoft Azure

La estructura seleccionada para realizar este caso de estudio ha sido de nuevo el edificio Veles e Vents, aunque en este caso, como se puede observar en la figura 9.56, dispone de un parking, además de las cuatro plataformas horizontales.

El principal objetivo de este caso de estudio consistió en obtener el diseño estructural más apropiado del edificio, de acuerdo a diferentes criterios económicos y de seguridad. Para ello, se propusieron 8 soluciones estructurales diferentes, con variaciones en los materiales empleados, en el mallado de los elementos finitos y en las dimensiones de los elementos estructurales. De media, cada solución estructural estaba compuesta por el número nudos y elementos finitos triangulares que se muestra en la tabla 9.15.

Para cada solución estructural, se calculó su respuesta dinámica a lo largo del tiempo bajo la influencia de 5 terremotos distintos, de 10 segundos de duración cada uno de ellos. Esto dio lugar a 40 simulaciones diferentes e independientes para ser lanzadas en la nube. En cada una de ellas, empleamos el método HHT-α, con incrementos de tiempo en la simulación de 0.01 segundos y con incrementos

de tiempo en la grabación de resultados de 0.05 segundos, lo cual equivale a decir que fueron necesarios 1000 pasos de tiempo para simular el comportamiento de la estructura, 200 de los cuales se almacenaron en disco.

La ejecución de cada simulación dinámica duró 21.48 minutos en un PC convencional, compuesto por un procesador Intel core i5 a 3.20 GHz y 4 GBytes de RAM. El fichero de entrada de cada simulación, compuesto por una de las soluciones estructurales y un acelerograma, ocupaba 12.07 MBytes. En cambio, los ficheros de resultados correspondientes a los 200 pasos de tiempo necesitaron un total de 10.36 GBytes.

La simulación del caso de estudio en la nube se llevó a cabo en la infraestructura de Microsoft Azure, empleando máquinas de tamaño medio (B2), las cuales constan de una CPU de doble núcleo con un ciclo de reloj de 1.60 GHz y 3.5 GBytes de memoria RAM. Dicho caso de estudio fue ejecutado bajo 3 tipos de configuraciones diferentes, cuyas características se muestra a continuación:

- *Configuración A*: Los resultados de los 200 pasos de tiempo de cada simulación se generan y se almacenan en el servicio Cloud y se descargan en su totalidad a la máquina local del cliente.

- *Configuración B*: Generamos remotamente los resultados de los 200 pasos de tiempo, pero únicamente se almacenan en el servicio Cloud los resultados de los 3 instantes más desfavorables, de acuerdo a los cortantes en la base, los cuales se descargan a la máquina del cliente.

- *Configuración C*: Los resultados de los 200 pasos de tiempo se generan y almacenan en el servicio Cloud, pero no se descarga ninguno a la máquina local del usuario. Se entiende que el usuario se los descargará, todos o parte de ellos, con posterioridad.

La tabla 9.16 nos muestra, entre otras, una comparativa de los tiempos de ejecución de una única simulación en la máquina local del usuario o en una de las instancias de la infraestructura de Azure, empleando las configuraciones mencionadas. Como puede observarse en este caso en particular, el tiempo de simulación en la nube es superior al tiempo invertido en la máquina del cliente. En breve

Tipo de Ejecución	Tiempo de respuesta	Sobrecarga	Datos de entrada	Datos de salida
Local	27.48 min.	-	12.07 MBytes	10.36 GBytes
Remota - Conf. A	47.53 min.	20.05 min.	6.91 MBytes	7.90 GBytes
Remota - Conf. B	47.27 min.	19.78 min.	6.91 MBytes	89.14 MBytes
Remota - Conf. C	46.43 min.	18.95 min.	6.91 MBytes	0 Bytes

Tabla 9.16: Resultados de una única simulación en local o en la nube.

aclararemos a qué es debida dicha sobrecarga, recogida en la tercera columna. En lo que respecta al espacio ocupado por los datos de salida, recogido en la quinta columna de la tabla mencionada, lo consideramos desde el punto de vista de aquellos que recogerá el usuario en su máquina local como resultado de la simulación. Para intentar reducir el más que considerable volumen de datos a enviar y recibir entre el cliente y el servicio Cloud, dichos datos se comprimirán previamente. Precisamente, la información reflejada en la tabla 9.16 para las 3 configuraciones se corresponde con los datos comprimidos, no siendo así en el caso de la ejecución local.

Como cabía esperar, la ejecución remota más rápida tuvo lugar en el caso de la Configuración C, en la cual no hay recogida de resultados. Puede llamar la atención al lector que las tres configuraciones presenten unos tiempos de simulación muy similares, a pesar de la notable diferencia de datos a recibir por parte del cliente en cada una de ellas. Ello es así porque, en la Configuración A, los movimientos de datos desde la máquina virtual en la que se ejecuta la tarea al almacenamiento Cloud, y de este último a la máquina del cliente, se llevan a cabo de manera solapada con la ejecución de la simulación. En la Configuración B, sin embargo, la subida de los datos al servicio de almacenamiento y su posterior recogida por parte del cliente relativa a los instantes de tiempo más desfavorables no puede comenzar hasta que la simulación haya finalizado por completo y hayamos determinado de qué 3 instantes de tiempo se trata frente a los 200 disponibles.

A fin de analizar a qué es debida la sobrecarga de tiempo relativa al cálculo mediante el servicio Cloud, la tabla 9.17 nos muestra el tiempo dedicado a las diferentes etapas que forman parte de la simulación remota, así como la sobrecarga asociada al compararlo con el tiempo invertido al analizar la estructura en la máquina local. En la segunda columna, se indica el tiempo relativo al en-

vío del trabajo al servicio Cloud, junto con sus datos de entrada. En la tercera columna, se muestra el tiempo de descargar del Simulador Estructural desde el almacenamiento Cloud y la inicialización del trabajo. En la cuarta columna, aparece recogido el tiempo invertido por el Simulador Estructural en el análisis de la estructura, junto con el dedicado por el Gestor del Servicio de Análisis Estructural a compactar los diferentes ficheros generados por el Simulador en uno solo. Finalmente, en la última columna se indica el tiempo correspondiente a mover los resultados desde la máquina virtual empleada en la simulación al almacenamiento Cloud y su descarga posterior, si procede, a la máquina local del usuario.

Como podemos ver, el mayor tiempo invertido es debido a la simulación de la estructura y a la compactación de los resultados. Ello es así, principalmente, por las diferentes características computacionales que presenta la instancia de tipo medio escogida de Azure, comparada con la máquina local. Aunque la sobrecarga asociada al resto de etapas relacionadas con el análisis de forma remota está presente, tiene una repercusión mucho menor ya que siempre está, en conjunto, por debajo de los 2 minutos. Únicamente conviene resaltar que el hecho de que la Configuración B presente unos tiempos de simulación mayores es debido al tiempo invertido por el Gestor del Servicio de Análisis Estructural en determinar los 3 instantes de tiempo que presentan los resultados más desfavorables. Obsérvese que en Microsoft Azure no tenemos un tiempo dedicado al despliegue de la máquina virtual en la cual se ejecuta la simulación, como habría sido necesario si hubiéramos usado COMPSs, puesto que dicho tiempo tuvo lugar durante el despliegue del servicio y no durante la ejecución de una simulación o del caso de estudio completo.

En lo que respecta a la ejecución del caso de estudio en la máquina local, fueron necesarias 18.32 horas (1099.31 minutos) para analizar las 40 simulaciones, calculando una detrás de otra, con un total de 482.66 MBytes de ficheros de entrada y un volumen de resultados de 414.42 GBytes.

En cuanto a la ejecución del caso de estudio completo en la nube, hemos empleado 4 despliegues diferentes compuestos por 1, 10, 20 y 40 instancias de tipo medio de Azure, bajo los 3 tipos de configuraciones mencionadas. Los tiempos de respuesta para dichos despliegues y configuraciones están recogidos en la tabla 9.18 y en la figura 9.57. Como hemos dicho con anterioridad, los datos que se

	Envío de la tarea	Inicialización del trabajo	Simulación estructural	Subida de resultados y descarga local
Local	-	-	27.48 min.	-
Config. A	34 seg.	3 seg.	45.58 min.	80 seg.
Sobrecarga Config. A	34 seg.	3 seg.	18.10 min.	80 seg.
Config. B	34 seg.	3 seg.	45.72 min.	56 seg.
Sobrecarga Config. B	34 seg.	3 seg.	18.25 min.	56 seg.
Config. C	34 seg.	3 seg.	45.58 min.	14 seg.
Sobrecarga Config. C	34 seg.	3 seg.	18.10 min.	14 seg.

Tabla 9.17: Tiempos de las diferentes etapas de una simulación en local y en la nube.

envían o reciben entre el cliente y el servicio Cloud estarán comprimidos. Esto supone que, al ejecutar el caso de estudio, sean necesarios 276 MBytes datos de entrada y 315.87 GBytes como cantidad de resultados a recibir por el cliente en el caso de la configuración A o de 3.48 GBytes para la configuración B.

En el caso de las configuraciones B y C, los tiempos de respuesta se reducen gradualmente a medida que el número de instancias aumenta. Inicialmente, esto es también así para la configuración A, hasta que empleamos 10 máquinas de Azure. A partir de dicho número, el tiempo de ejecución se mantiene alrededor de las 8 horas, donde la descarga de los resultados a la máquina del cliente se convierte en un auténtico cuello de botella, ocasionando que los tiempos de recogida de datos superan en gran medida a los tiempos de simulación en la nube.

En cualquier caso, conviene destacar la reducción del tiempo que ha tenido lugar al calcular el caso de estudio completo por medio del servicio Cloud. Si bien, a nivel local, necesitábamos alrededor de 18 horas para ejecutar las 40 simulaciones, gracias a la plataforma computacional desplegada en la nube podremos disponer de la totalidad de los resultados en menos de la mitad de ese tiempo, o en menos incluso de 1 hora si nos es suficiente con recoger sólo los resultados más desfavorables o dejar para más adelante la recepción de los datos de salida.

Como métricas para evaluar las prestaciones del servicio Cloud, calcularemos el incremento de velocidad y la eficiencia al ejecutar el caso de estudio completo.

Número de instancias	Tiempo Local	Tiempo Config. A	Tiempo Config. B	Tiempo Config. C
1	18.32 h.	31.45 h.	31.34 h.	31.00 h.
10	-	9.12 h.	3.30 h.	3.25 h.
20	-	8.51 h.	1.73 h.	1.69 h.
40	-	8.76 h.	0.92 h.	0.90 h.

Tabla 9.18: Tiempos de ejecución del caso de estudio completo en local y en la nube.

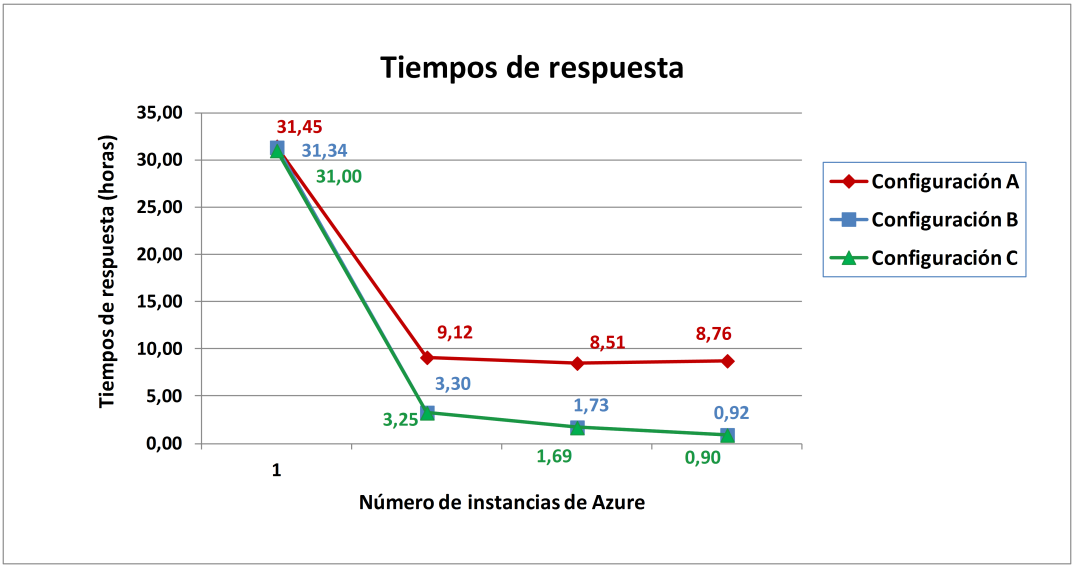


Figura 9.57: Tiempos de respuesta del caso de estudio completo en la nube.

Por un lado, expresaremos dichos valores con respecto a la aproximación secuencial, en la cual lanzamos todas las simulaciones en la máquina local del usuario, a fin de comparar las ventajas que ofrece el servicio para reducir los tiempos de respuesta de múltiples simulaciones independientes. Por otro lado, proporcionaremos dichas métricas con respecto a emplear solo una instancia de Azure, analizando así la escalabilidad del servicio Cloud desplegado.

Las figuras 9.58 y 9.59 nos muestran el comportamiento del sistema Cloud en términos de incremento de velocidad tras aumentar el número de instancias de Azure para ejecutar el caso de estudio completo, con respecto a la aproximación tradicional o con respecto a los resultados de ejecución en el sistema Cloud con una única instancia, respectivamente.

En la primera de dichas figuras, el incremento de velocidad toma un valor cercano a 0.6 al emplear una única instancia, lo que nos indica que la ejecución de una simulación en el servicio Cloud es mucho lenta que su ejecución en local, para cualquiera de las configuraciones. Esto es así, como ya hemos comentado, por las diferencias de las características computacionales de la máquina local y la máquina virtual de Azure que hemos empleado. Mientras que el incremento de velocidad de las configuraciones B y C, con respecto a una instancia de Azure, toma valores cercanos al ideal (alrededor de 34 para 40 máquinas), dicho incremento con respecto a la aproximación secuencial resulta ser igual a 20 para 40 instancias. En el caso de la configuración A, es fácil apreciar que dicho incremento de velocidad permanece prácticamente inalterado a pesar de emplear 10, 20 ó 40 máquinas, debido a las razones comentadas relativas al cuello de botella de la red en la recepción de los resultados.

El comportamiento del servicio Cloud en términos de eficiencia se refleja en las figuras 9.60 y 9.61, las cuales nos muestran el resultado con respecto a la ejecución en local y con respecto a emplear una única instancia de Azure, para las 3 configuraciones. Como era de esperar, el decremento en la eficiencia es lento y lineal en las configuraciones B y C tras aumentar el número de instancias, ofreciendo en consecuencia las mejores eficiencias (0.86 para 40 instancias), mientras que dicha tendencia a la baja es mucho más acuciada en el caso de la configuración A, la cual presenta unos resultados claramente peores.

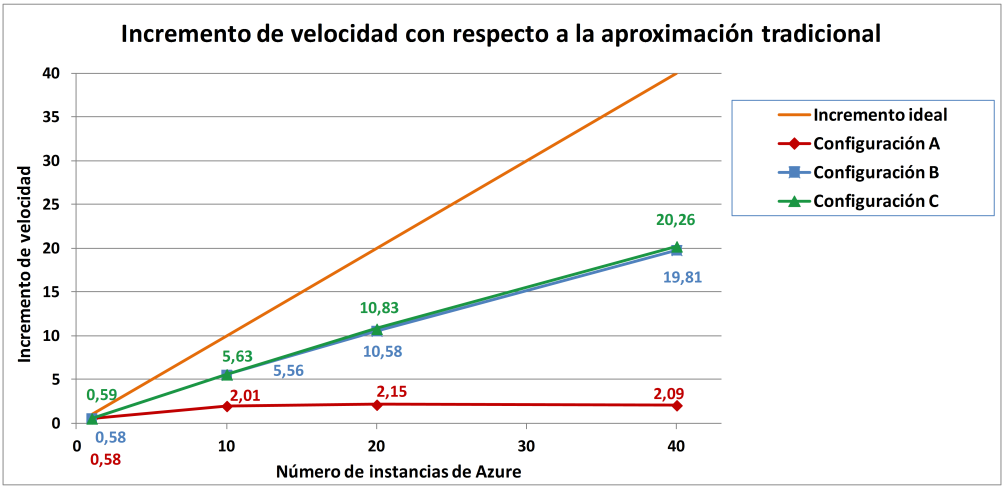


Figura 9.58: Incrementos de velocidad con respecto a la ejecución en local.

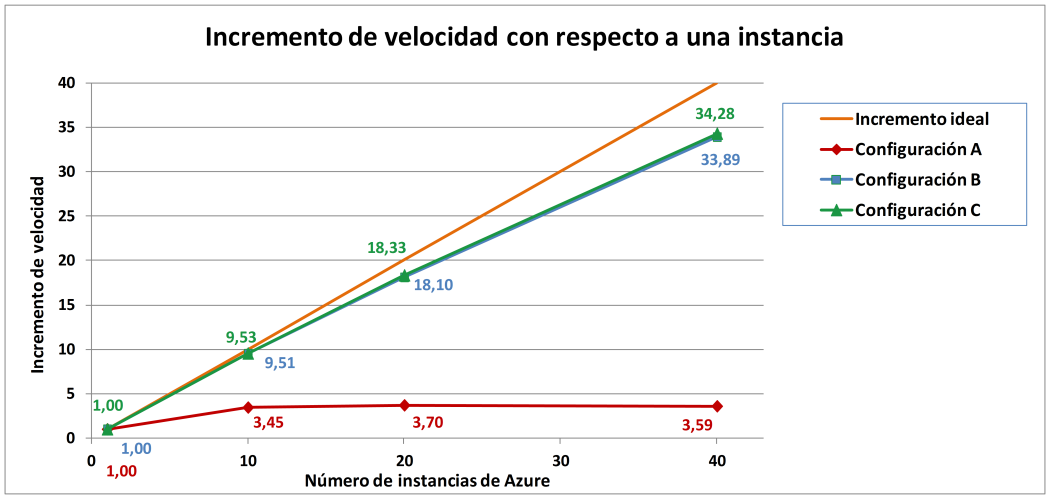


Figura 9.59: Incrementos de velocidad con respecto a una instancia de Azure.

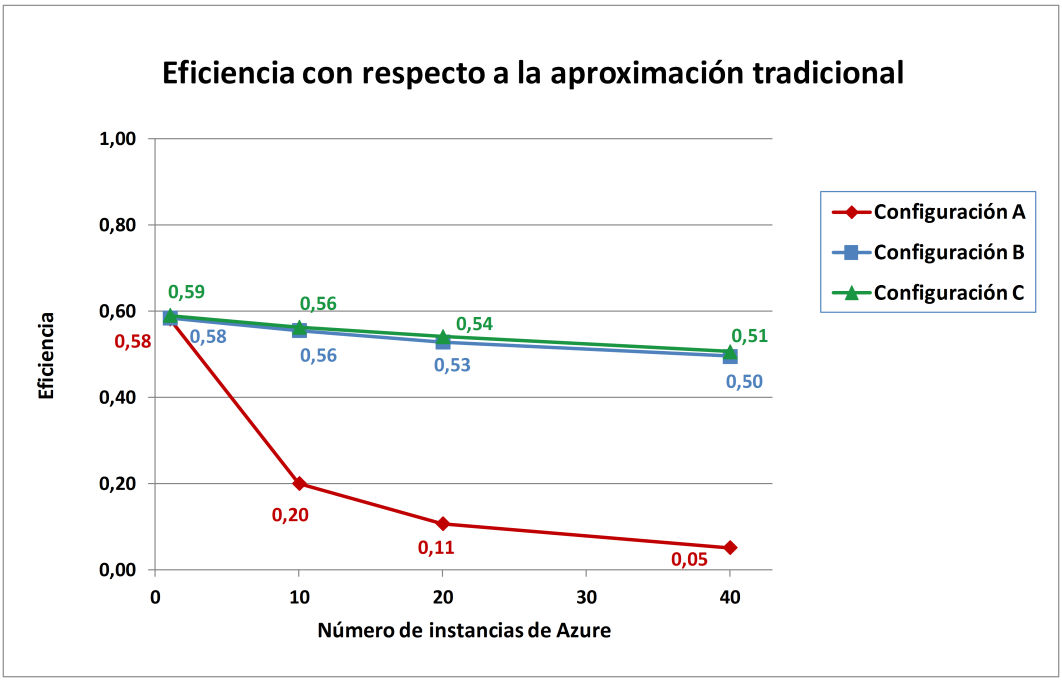


Figura 9.60: Eficiencia del servicio con respecto a la ejecución en local.

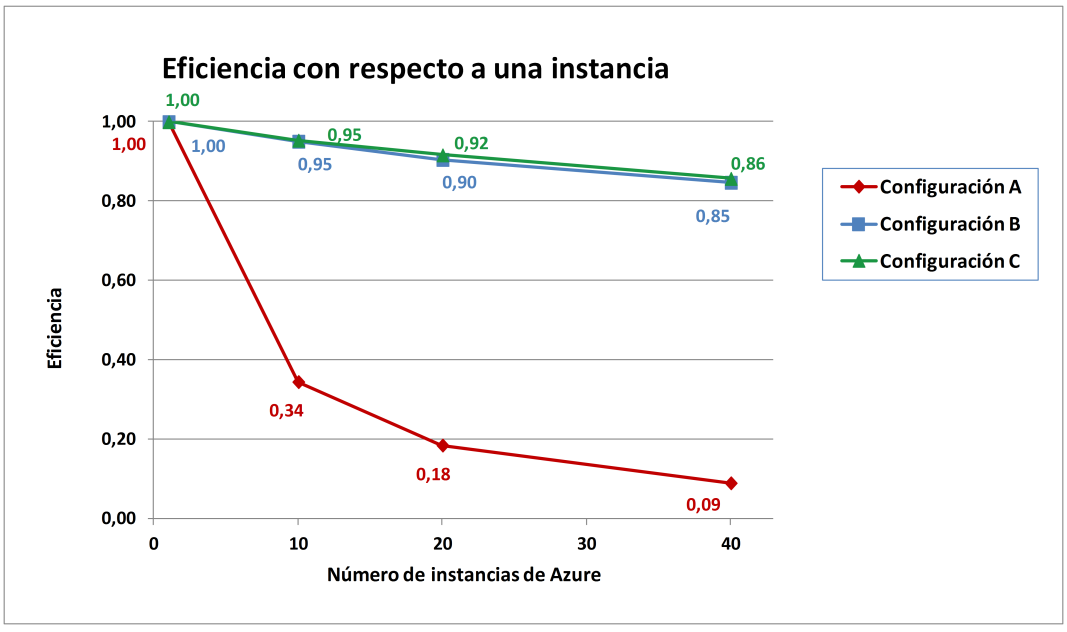


Figura 9.61: Eficiencia del servicio con respecto a una instancia de Azure.

Concepto	Coste (en euros)
Consumo de CPU	0.1265 por hora
Almacenamiento al mes	0.0203 por GByte
Transferencia de salida	0.0734 por GByte

Tabla 9.19: Precios de Microsoft Azure por una instancia de tipo B2.

Conf.	Número de Instancias	Coste de Simulación	Coste de Almacenamiento	Coste de Transferencia	Coste Total
A	1	3.92 €	0.28 €	23.18 €	27.39 €
A	10	4.11 €	0.08 €	23.18 €	27.38 €
A	20	4.28 €	0.08 €	23.18 €	27.54 €
A	40	4.58 €	0.08 €	23.18 €	27.84 €
B	1	3.96 €	0.00 €	0.26 €	4.22 €
B	10	4.17 €	0.00 €	0.26 €	4.43 €
B	20	4.38 €	0.00 €	0.26 €	4.64 €
B	40	4.68 €	0.00 €	0.26 €	4.93 €
C	1	3.92 €	1.50 €	11.59 €	17.01 €
C	10	4.11 €	1.50 €	11.59 €	17.20 €
C	20	4.28 €	1.50 €	11.59 €	17.37 €
C	40	4.58 €	1.50 €	11.59 €	17.67 €

Tabla 9.20: Coste de ejecución del caso de estudio para las distintas configuraciones.

Los precios ofertados por Microsoft para una instancia de tipo B2 son los que podemos apreciar en la tabla 9.19. Conviene aclarar que Microsoft sólo nos cobra por las transferencias de salida, pero no así por las de entrada. Por otro lado, tampoco se paga por los datos almacenados en el disco local de la máquina virtual, sino por aquellos datos que subamos al almacenamiento Cloud. En ese sentido, el precio mostrado en la tabla se refiere a almacenar 1 GByte durante un mes completo. Si el tiempo es inferior, pagaremos únicamente por el periodo transcurrido.

A partir de dichos precios, obtendremos los costes correspondientes a la ejecución del caso de estudio completo, mostrados en la tabla 9.20 para las diferentes configuraciones consideradas y para un número distinto de instancias empleadas.

Como podemos observar, el precio por simulación en las configuraciones A y C es idéntico, puesto que ambas invierten el mismo tiempo en ejecutar las

simulaciones y en subir al servicio de almacenamiento los resultados de los 200 instantes de tiempo generados. En cambio, dicho tiempo es ligeramente superior para la configuración B, al calcular los 3 instantes de tiempo más desfavorables, lo que conlleva un coste un poco mayor.

Para calcular el coste del almacenamiento en las configuraciones A y B, hemos considerado el periodo de tiempo que transcurre desde que comienzan a almacenarse los datos del primer instante de tiempo de la primera simulación hasta que el usuario se descarga los datos correspondientes a la última de las simulaciones. Se entiende que, pasado ese momento, los datos se borran por completo del almacenamiento Cloud. Como podemos observar, este coste es nulo para la configuración B, debido al reducido tamaño que ocupan los 3 instantes de tiempo a almacenar y del escaso tiempo que permanecen guardados en el Cloud.

Con respecto a la configuración C, sería irreal pensar que el usuario no se descargará los resultados de todas o de parte de las simulaciones en algún momento. En ese sentido, hemos estimado que el usuario guardará los datos de todas las simulaciones en el Cloud durante una semana, que será el tiempo que necesitará, a modo de ejemplo, para interpretar los resultados. Del mismo modo, para obtener el coste de transferencia de dicha configuración C y a partir de los cortantes en la base, entendemos que el usuario podría descargarse los datos de salida de unas pocas simulaciones o podría descargarse los de la mayoría de ellas. Como término medio, hemos estimado que recogerá los datos del 50 % de las simulaciones.

En el caso por tanto de la configuración A, en la que el usuario almacena en el Cloud y se descarga los resultados de todos los instantes de tiempo para todas las simulaciones, el coste por ejecución del caso de estudio alcanza los 27.84 euros al emplear 40 instancias, lo cual supone que el coste de cada simulación será de 0.70 euros. En el extremo contrario tenemos el caso de la configuración B, en la cual el usuario sólo almacena y descarga los 3 instantes de tiempo más desfavorables. Como podemos ver, el coste del caso de estudio completo ronda los 5 euros, empleando para ello 40 instancias de Azure. Como es evidente, esto implica que el coste por simulación será de 0.12 euros.

Dichos costes ponen claramente de manifiesto la ventaja, a nivel económico, que supone el ejecutar las simulaciones en el Cloud empleando para ello, en este

ejemplo, hasta 40 instancias, comparado con el coste que supondría adquirir y mantener una infraestructura hardware compuesta por el número de máquinas que estimemos oportuno, a nivel local.

Capítulo 10

Conclusiones y Trabajos Futuros

Este capítulo final incluye las contribuciones principales llevadas a cabo en esta tesis doctoral. Se recoge además un listado de las publicaciones en congresos y en revistas de investigación llevadas a cabo, además de las diferentes tareas de difusión realizadas a partir de los resultados obtenidos. Finalmente, se describen distintas aplicaciones software o líneas de investigación que cuentan como componente al Simulador Estructural desarrollado, junto con los trabajos futuros.

10.1. Contribuciones principales

El trabajo realizado en esta tesis doctoral ha consistido en la aplicación de computación avanzada, entendiendo como tal a la Computación de Altas Prestaciones, las Tecnologías Grid y las Tecnologías Cloud, al cálculo estático y dinámico lineal de estructuras de edificación e ingeniería civil mediante el método de los elementos finitos. Dichas estructuras podrán estar compuestas, en su modelización, por barras, triángulos, cuadriláteros, tetraedros, hexaedros o prismas triangulares.

Gracias a la eficiencia de los métodos numéricos actuales, plasmados en libre-

rías numéricas de dominio público, y a la disponibilidad de librerías de comunicación y sincronización entre los elementos de proceso, bien sea mediante paso de mensajes o por memoria compartida, la aplicación de la Computación de Altas Prestaciones al cálculo estructural ha permitido llevar a cabo un análisis riguroso y realista de estructuras de gran dimensión y complejidad, con unos tiempos de respuesta razonablemente reducidos.

Más concretamente, la aplicación implementada está basada en PETSc, uno de los entornos de desarrollo de computación científica más conocidos, el cual nos permite resolver problemas numéricos con matrices dispersas en paralelo, como la resolución de sistemas de ecuaciones lineales o la realización de operaciones básicas de álgebra lineal. A la vez, los algoritmos de cálculo estático y dinámico desarrollados utilizan el paradigma de programación paralela multinivel, gracias al uso conjunto de MPI y OpenMP, a fin de garantizar la portabilidad y obtener las mejores prestaciones en distintos sistemas operativos, como Linux y Windows, sobre diferentes plataformas computacionales paralelas, las cuales van desde un simple PC a costosos supercomputadores de memoria compartida distribuida, pasando por las redes de PCs.

Con el objetivo de distribuir la malla de barras y elementos finitos entre los distintos elementos de cómputo, hemos aplicado dos estrategias diferentes, basadas en un particionado por corte de nodos o por corte de elementos. Mientras que la primera estrategia conlleva que cada barra o elemento finito se asigne a una única partición, la segunda supone que un mismo elemento estructural podrá formar parte, conjuntamente, de más de una de ellas. Claramente, esta segunda alternativa implicará el cálculo solapado de aquellos elementos replicados en distintas particiones pero, al mismo tiempo, conllevará reducir notablemente el volumen de las comunicaciones entre las tareas en las diferentes etapas de análisis. Para distribuir la dicha malla de elementos que conforman la estructura, hemos empleado la librería METIS.

El cálculo estático se ha llevado a cabo mediante el Método de la Rigidez, donde se han paralelizado todas las etapas que lo componen, incluyendo la de la escritura de resultados mediante rutinas de MPI-IO. En el caso del análisis dinámico, la resolución de la ecuación dinámica de segundo orden se ha implementado mediante la paralelización de numerosos métodos de integración directa, así co-

mo mediante técnicas de análisis modal, superposición modal y análisis modal espectral. En el caso concreto del análisis modal espectral, se han paralelizado un amplio número de métodos de combinación modal de los resultados, acompañados de distintas técnicas de combinación direccional y diferentes alternativas que intentan proporcionar el signo de los resultados. El uso de librerías numéricas paralelas de dominio público ha dotado de robustez y eficiencia a la aplicación desarrollada. Así, los sistemas de ecuaciones lineales se han resuelto mediante las librerías MUMPS, PaStiX y PARDISO, basadas en métodos directos, o mediante los métodos iterativos presentes en PETSc acompañados de sus precondicionadores o algunos otros que forman parte de *hypre*. Además, se han empleado las librerías SLEPc y ARPACK para resolver el problema de valores propios generalizado y obtener las frecuencias naturales y los modos de vibración de la estructura.

Con el objetivo de que dicha aplicación paralela desarrollada pueda estar a disposición de una amplia comunidad de arquitectos e ingenieros estructurales, se han implementado y desplegado sendos servicios Grid y Cloud que ofrecen un análisis estructural por internet seguro, fiable y de alta productividad bajo una arquitectura orientada a servicio.

Así, el servicio Grid se ha desarrollado sobre Globus Toolkit 4, como software que ha ofrecido los servicios Grid básicos, integrando componentes software como GMarte, capaz de llevar a cabo la planificación de tareas, la recogida de resultados y la tolerancia a fallos. Adicionalmente, el servicio Cloud se ha implementado bajo los desarrollos del proyecto europeo VENUS-C, desplegándose sobre la infraestructura de Microsoft Azure o sobre una infraestructura Cloud "on-premises" gestionada por COMPSs.

Aunque el ámbito de aplicación de los desarrollos de esta tesis doctoral se ha ceñido principalmente al de las estructuras históricas y de edificación, conviene resaltar que dichos desarrollos serían aplicables a cualquier otro tipo de estructuras, como sería el caso de los puentes, túneles, depósitos, silos, invernaderos, etc. Más aún, su extrapolación a otros ámbitos de la ingeniería de estructuras, la resistencia de materiales, la ingeniería aeroespacial, la ingeniería mecánica y la de automoción podría ser directo, siendo incluso extensible a la ingeniería hidráulica y la ingeniería del terreno.

10.2. Publicaciones docentes y de investigación

La realización de esta tesis doctoral ha dado lugar a la publicación de los siguientes artículos de investigación:

1. A. Pérez-García, F. Gómez-Martínez, A. Alonso, V. Hernández, J.M. Alonso, P. de la Fuente, and P. Lozano. Architrave: Advanced Analysis of Building Structures Integrated in Computer-Aided Design. *Construction and Building Research*, 123–139. Springer Netherlands, 2014.

2. M. Caballer, P. de La Fuente, P. Lozano, J. M. Alonso, C. de Alfonso, and V. Hernández. Easing the Structural Analysis Migration to the Cloud. *IBERGRID 2013: 7th Iberian Grid Infrastructure Conference Proceedings*, pp. 159–171. Universitat Politècnica de València. Madrid, 2013.

3. J. M. Alonso, A. Alonso, P. de la Fuente, F. Gómez, V. Hernández, P. Lozano, and A. Pérez. A Cloud System Implementation for the Analysis of Civil Engineering Structures. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'12)*, pp. 210–216. CSREA Press. Las Vegas (Estados Unidos), 2012.

   Congreso de tipo B según la clasificación proporcionada por CORE2013.

4. J. M. Alonso, V. Hernández, R. López, and G. Moltó. GRID4BUILD: A High Performance Grid Framework for the 3D Analysis and Visualisation of Building Structures. *IBERGRID: 1st Iberian Grid Infrastructure. Conference Proceedings*, pp. 353–364. Fundación CESGA, Gesbiblo, S.L. Santiago de Compostela, 2007.

5. J. M. Alonso, V. Hernández, and G. Moltó. A High-Throughput Application for the Dynamic Analysis of Structures on a Grid Environment. *Advances in Engineering Software*, 39(10):839–848, 2008.

   Revista indexada en el *Journal Citation Reports* del *Science Citation Index* con un factor de impacto de 1.188 en el año 2008, recogida en el segundo cuartil (Q2).

6. J. M. Alonso, C. Alfonso, G. García, and V. Hernández. Grid Technology for Structural Analysis. *Advances in Engineering Software*, 38(11-12):738–749, 2007.

   Revista indexada en el *Journal Citation Reports* del *Science Citation Index* con un factor de impacto de 0.529 en el año 2007, recogida en el tercer cuartil (Q3).

7. J. M. Alonso, V. Hernández, R. López, and G. Moltó. A Service Oriented System for On Demand Dynamic Structural Analysis over Computational Grids. *High Performance Computing for Computational Science. VECPAR 2006*. LNCS 4395:13–26, 2007.

   Congreso de tipo B según la clasificación proporcionada por CORE2008.

8. J. M. Alonso, V. Hernández, R. López, and G. Moltó. Una Aproximación Orientada a Servicios Grid para el Análisis Estático y Dinámico de Estructuras de Edificación. *Actas de la 32a Conferencia Latinoamericana de Informática*, pp. 1–9. Santiago de Chile, 2006.

9. J. M. Alonso, V. Hernández, R. López, and G. Moltó. Experiences Using Grid Services in Structural Dynamics. *eWork and eBusiness in Architecture, Engineering and Construction. Proceedings of the 6th European Conference on Product and Process Modelling (ECPPM 2006)*, pp. 359–366. Valencia, 2006.

10. J. M. Alonso, V. Hernández, R. López, and G. Moltó. A Grid Service Development for Three-Dimensional Structural Analysis. *Proceedings of the Fifth International Conference on Engineering Computational Technology (ECT 2006)*. B. H. V. Topping, G. Montero and R. Montenegro (Editors), Civil-Comp Press, paper 122. Las Palmas de Gran Canaria, 2006.

11. J. M. Alonso and V. Hernández. A Parallel Implementation of Three-Dimensional Modal Analysis of Building Structures. *Proceedings of the Fifth International Conference on Engineering Computational Technology (ECT 2006)*. B. H. V. Topping, G. Montero and R. Montenegro (Editors), Civil-Comp Press, paper 120. Las Palmas de Gran Canaria, 2006.

12. J. M. Alonso, V. Hernández, and G. Moltó. Aplicación de la Computación Paralela y las Tecnologías Grid en el Cálculo Dinámico de Estructuras de Edificación. *Proceedings of the 12th International Congress on Computer Science Research (CIICC'05)*, pp. 3–14. Monterrey (México), 2005.

13. J. M. Alonso, V. Hernández, and G. Moltó. Biomedical and Civil Engineering Experiences Using Grid Computing Technologies. *Parallel Computing: Current and Future Issues of High-End Computing*, NIC Series Volume 33, pp. 647–654. Málaga, 2005.

    Congreso de tipo C según la clasificación proporcionada por CORE2008.

14. J. M. Alonso, C. Alfonso, V. Hernández, and G. Moltó. A Grid-based Application of the Three-dimensional Dynamic Analysis of High-Rise Buildings. *Proceedings of The Tenth International Conference on Civil, Structural and Environmental Engineering Computing (CC 2005)*, B. H. V. Topping (Editor), Civil-Comp Press, paper 68. Roma (Italia), 2005.

15. J. M. Alonso and V. Hernández. Three-dimensional Structural Dynamic Analysis using Parallel Direct Time Integration Methods. *Proceedings of The Tenth International Conference on Civil, Structural and Environmental Engineering Computing (CC 2005)*, B. H. V. Topping (Editor), Civil-Comp Press, paper 232. Roma (Italia), 2005.

16. J .M. Alonso, C. Alfonso, and V. Hernández. Desarrollo de un Servicio Grid para el Análisis 3D de Estructuras de Edificación. *Anales de Ingeniería Mecánica. Revista de la Asociación Española de Ingeniería Mecánica*, Año 15, Volumen No. 4:2387–2395, 2004.

    Revista indexada por Latindex.

17. J. M. Alonso, G. García, and V. Hernández. Minimizando los Tiempos de Análisis 3D de Estructuras de Edificación de Gran Dimensión. *Anales de Ingeniería Mecánica. Revista de la Asociación Española de Ingeniería Mecánica*, Año 15, Volumen No. 4:2407–2415, 2004.

    Revista indexada por Latindex.

18. J. M. Alonso, C. de Alfonso, G. García, and V. Hernández. Parallel and Grid Computing in 3D Analysis of Large Dimension Structural Systems. *Proceedings of the 10th International Euro-Par Conference*. LNCS 3036:487–496, 2004.

    Congreso de tipo A según la clasificación proporcionada por CORE2008.

19. J. M. Alonso, C. de Alfonso, G. García, and V. Hernández. Integrating HPC and Grid Computing for 3D Structural Analysis of Large Buildings. *Proceedings of the Fourth International Conference on Engineering Computational Technology (ECT 2004)*. B.H.V. Topping and C.A. Mota Soares, (Editors), Civil-Comp Press, paper 92. Lisboa (Portugal), 2004.

20. J.M. Alonso, C. de Alfonso, G. García, and V. Hernández. Desarrollo de una Aplicación de Alta Productividad para el Cálculo Estructural de Edificios de Gran Dimensión. *Actas de las XV Jornadas de Paralelismo*, pp. 277–282. Universidad de Almería. Almería, 2004.

21. J. M. Alonso, V. Hernández, and A. M. Vidal. HIPERBUILD: An Efficient Parallel Software for 3D Structural Analysis of Buildings. *Parallel Computing: Fundamentals Applications. Proceedings of the International Conference ParCo99*. Imperial College Press, pp. 235–242. Delft (Holanda), 1999.

    Congreso de tipo B según la clasificación proporcionada por CORE2008.

22. J. M. Alonso, V. Hernández, A. M. Vidal, and E. Abdilla. Parallel Computing and 3D Structural Analysis of Buildings. *18e Conférence Internationale Sur la CFAO et les Nouvelles Technologies de Conception et de Fabrication*, pp. 247-254. HERMES Science Publications. París (Francia), 1999.

23. J. M. Alonso, F. Alvarruiz, V. Hernández, and A. M. Vidal. High Performance Computing in the Building Construction Sector. *The Life-Cycle of Construction IT Innovations. Technology Transfer from Research to Practice*. Vol. 1, pp. 45–54. Royal Institute of Technology. Estocolmo (Suecia), 1998.

Además, se ha publicado este artículo docente:

1. A. Pérez-García, A. Guardiola, F. Gómez, and J. M. Alonso. Experiencias Docentes en Laboratorios Virtuales Orientadas a Mejorar la Eficiencia del Aprendizaje Autónomo del Análisis y Diseño Estructural. *Actas de las III Jornadas Internacionales de Enseñanza de la Ingeniería Estructural.* Asociación Científico-Técnica del Hormigón Estructural (ACHE). Valencia, 2013.

Adicionalmente, se han llevado a cabo las siguientes comunicaciones sin publicación, todas ellas relacionadas con la presente tesis doctoral:

1. HiperBUILD: Herramientas de Simulación y Visualización 3D de Altas Prestaciones. *Conferencias del Aula del Salón de Novedades del Salón Internacional de la Construcción.* Construmat'99. Barcelona, 1999.
2. Desarrollo de una Aplicación Grid de Altas Prestaciones para el Análisis Dinámico y la Visualización en 3D de Estructuras de Edificación (Grid4Build). *4º Reunión de la Red Temática en Grid Middleware.* Universidad Complutense de Madrid. Madrid, 2006.
3. Grid4Build: Sistema Avanzado para el Análisis y la Visualización de Estructuras de Edificación en 3D. *Jornadas FITEC de I+D+I en Construmat 2007, Salón Internacional de la Construcción.* Barcelona, 2007.
4. User Scenarios in VENUS-C - Focus on Structural Analysis. *Cloudscape III - Taking European Cloud Infrastructure Forward.* Siena, 2011.
5. VENUS-C - Architrave: Un Sistema para el Diseño y Análisis de Estructuras en la Nube. *6º Reunión Plenaria de la Red Española de e-Ciencia.* Centro Superior de Investigaciones Científicas. Madrid, 2012.

10.3. Aplicaciones que incorporan al Simulador Estructural

A medida que el Simulador Estructural ha ido progresando en sus desarrollos, se ha ido incorporando como núcleo de cálculo en diferentes aplicaciones de diseño

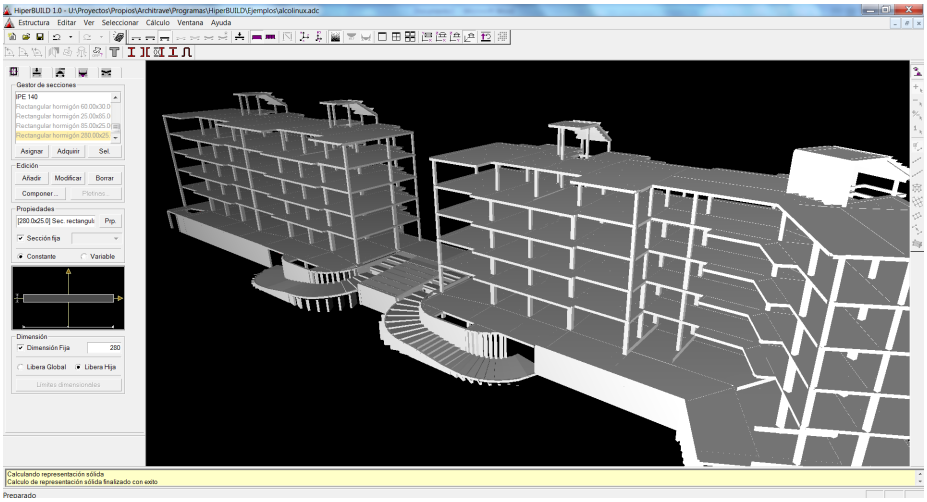


Figura 10.1: Captura de pantalla de la aplicación HiperBUILD.

y análisis estructural, dotadas todas ellas de un interfaz gráfico que permite al usuario proporcionar los datos de entrada y visualizar los resultados. Entre dichas aplicaciones, se encuentran las siguientes:

- HiperBUILD: Esta aplicación, ver figura 10.1, surge como resultado de una primera colaboración entre el DMMCTE de la UPV y el GRyCAP. Proporciona un cálculo estático en 3D de estructuras de barras basado en MPI. Previamente a la colaboración, el doctorando y el GRyCAP partían de la experiencia del proyecto europeo HIPERCOSME, donde habían aplicado la computación paralela al cálculo estático en 2D de estructuras de barras, mediante PVM. Con posterioridad, esta aplicación pasó a denominarse ATLANTES-HPCN.

- Grid4Build: Esta otra aplicación (ver figura 10.2) fue desarrollada por parte del GRyCAP bajo la financiación de los proyectos GRID4BUILD y GRID-IT. Permitía realizar un análisis estático y dinámico de estructuras de barras en una infraestructura GRID, para lo cual se desarrolló y se desplegó un servicio Grid basado en Globus Toolkit del que formaba parte el Simulador Estructural. A nivel paralelo, dicho Simulador estaba basado en MPI, e incorporaba métodos de integración directa, análisis modal y análisis por superposición modal. En lo que respecta a librerías numéricas, estaba basado en WSMP, MUMPS o PETSc para resolver los sistemas de ecuaciones

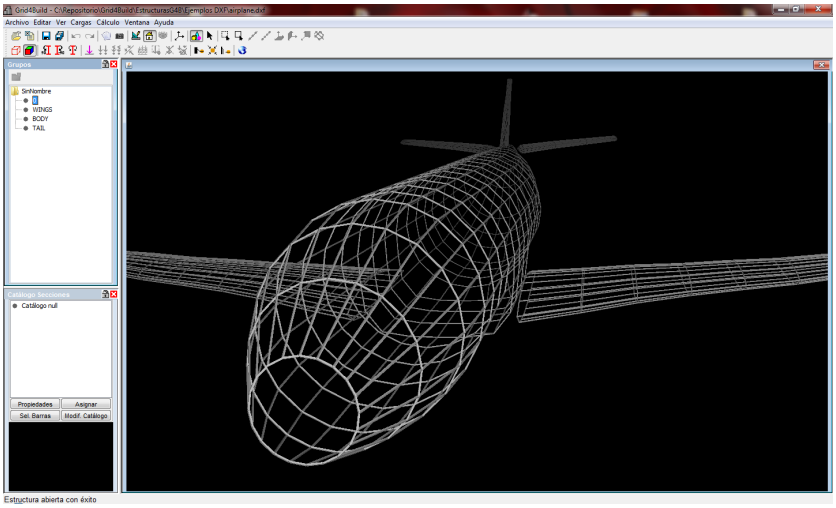


Figura 10.2: Interfaz gráfico de la aplicación Grid4Build.

lineales y en SLEPc para resolver el problema de valores propios generalizado. Grid4Build forma parte de la base de datos de aplicaciones de EGI [340].

- Architrave [335]: Se trata de un sistema software, que surge tras la colaboración del DMMCTE y el GRyCAP, compuesto por tres componentes, todos ellos totalmente desacoplados: Architrave Diseño, Architrave Cálculo y el Simulador Estructural.

  Architrave Diseño proporciona un interfaz gráfico de diseño estructural que funciona como una aplicación plug-in que opera en entornos de CAD de uso habitual en el desarrollo de proyectos de arquitectura e ingeniería (ver figura 10.3). En este entorno se define y genera el modelo de la estructura para su posterior análisis y dimensionado desde Architrave Cálculo.

  Architrave Cálculo consiste en una aplicación compuesta por un interfaz gráfico que asiste al usuario en las etapas de pre y post-proceso del análisis de la estructura. Proporciona una visualización gráfica realista y eficiente de modelos estructurales en 3D, como vemos en la figura 10.4, al tiempo que permite representar, visualizar y consultar los resultados del análisis. Adicionalmente, genera los listados y los planos de ejecución de la estructura e incorpora un módulo de dimensionado de elementos de hormigón armado

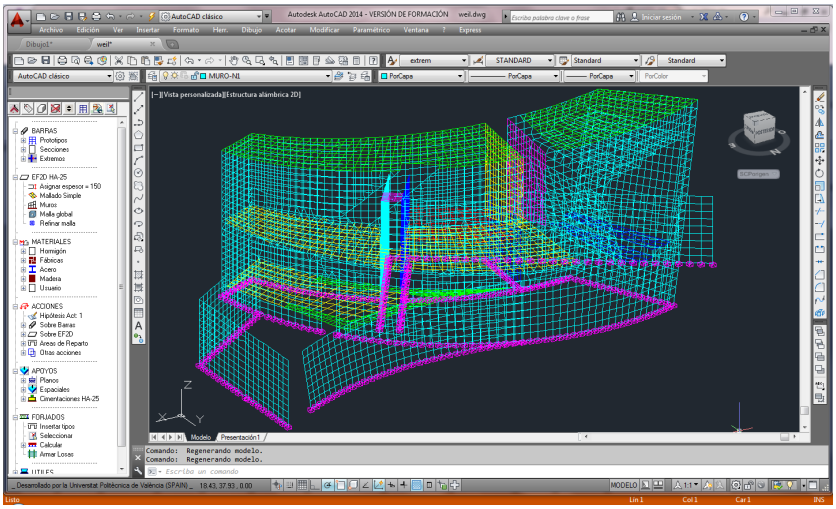


Figura 10.3: Interfaz gráfico de la aplicación Architrave Diseño.

y de acero, el cual calcula según las especificaciones del Código Técnico de la Edificación, de la Instrucción EHE-08 de hormigón estructural y de la Normativa de Construcción Sismorresistente NCSE-02. El cálculo de la estructura se realiza invocando al Simulador Estructural, el cual incorpora todos los desarrollos descritos en esta tesis doctoral.

Architrave sirvió como aplicación base para desarrollar el servicio de cálculo estructural en la nube en el marco del proyecto VENUS-C. En ese sentido, el Simulador Estructural está integrado dentro del servicio de cálculo remoto, y es el encargado de llevar a cabo todas las peticiones de análisis. A su vez, el usuario puede decidir desde el interfaz gráfico de Architrave Cálculo entre calcular la estructura en su máquina local o enviarla para ser calculada en la nube, siendo informado en todo momento de cómo va progresando el análisis de la misma.

Actualmente, el sistema Architrave se distribuye a nivel académico, de modo gratuito para alumnos, profesores e investigadores, y a nivel profesional, siendo la empresa Preference la encargada de llevar a cabo su comercialización. Desde la publicación de la primera versión en diciembre de 2010, el crecimiento del número de usuarios de Architrave ha sido constante, cuya evolución viene recogida en la figura 10.5. A día de hoy, cuenta con 3557 usuarios y 7730 descargas. A nivel profesional se ha descargado en 22 países y a nivel académico cuenta con usuarios (alumnos y profesores) de 54

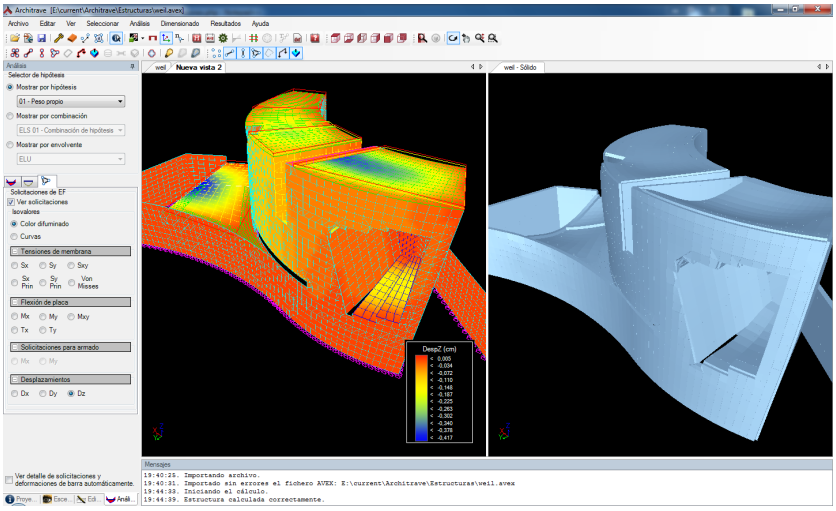


Figura 10.4: Interfaz gráfico de la aplicación Architrave Cálculo.

universidades de 8 países diferentes (ver figura 10.6), 37 de las cuales son españolas y 17 extranjeras. Por citar un ejemplo, Architrave se utiliza a nivel académico en asignaturas de las siguientes titulaciones de la UPV:

- Grado en Fundamentos de la Arquitectura, impartido por la Escuela Técnica Superior de Arquitectura (ETSAV): 5 asignaturas.
- Grado en Arquitectura Técnica, impartido por la Escuela Técnica Superior de Ingeniería de Edificación (ETSIE): 2 asignaturas.
- Máster Universitario en Arquitectura, impartido por la ETSAV: 3 asignaturas.
- Máster Universitario en Arquitectura Avanzada, Paisaje, Urbanismo y Diseño, impartido por la ETSAV: 3 asignaturas.
- Diploma de Especialización en Rehabilitación de Estructuras de Edificación: Seminario de formación de Architrave como herramienta horizontal a utilizar en las asignaturas.

- Calma3D: La empresa Preference S.L. es una empresa informática dedicada a la programación de aplicaciones para el sector de la madera, PVC y del aluminio, la cual desarrolla y comercializa desde hace años su producto PrefSuite [341], un conjunto de aplicaciones para la gestión completa de talleres

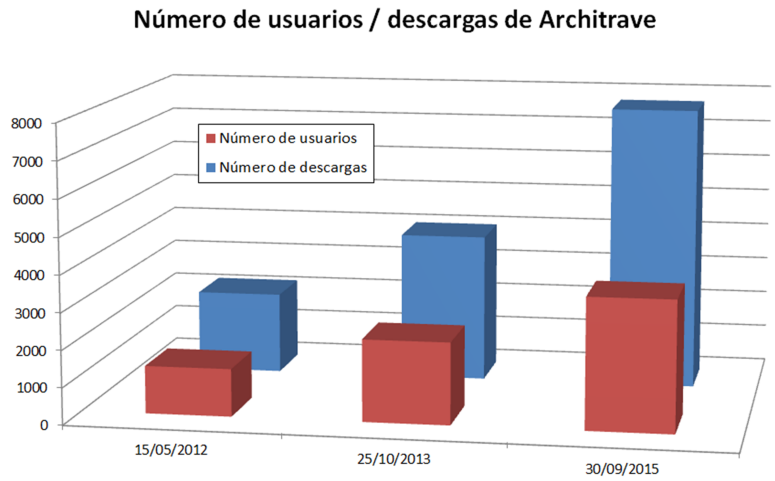


Figura 10.5: Número de usuarios y descargas de Architrave.

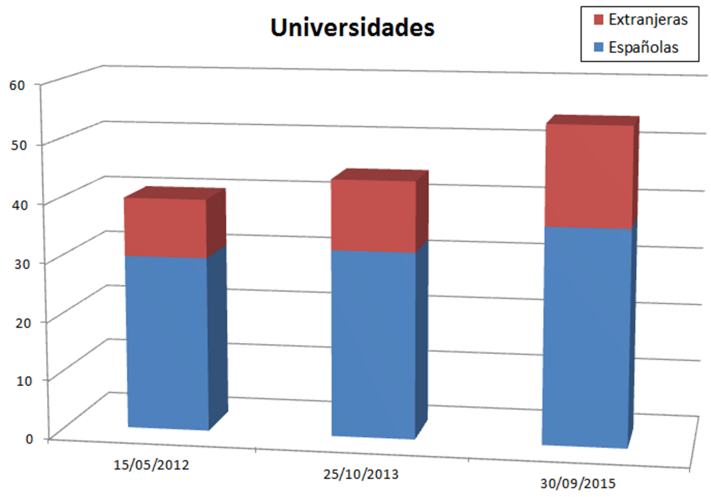


Figura 10.6: Evolución del número de universidades desde las que se ha descargado Architrave.

de aluminio, PVC y madera dirigido a fabricantes, distribuidores y proveedores de ventanas, carpinterías metálicas y de aluminio y a constructores de muros cortina.

A propuesta de dicha empresa, se han llevado a cabo un par de convenios de colaboración para el desarrollo de un software de Cálculo Estructural en 3D de Muros Cortina (Calma3D), el cual forma parte actualmente y se comercializa a nivel mundial bajo el paquete PrefSuite. Parte de los desarrollos del Simulador Estructural implementado en esta tesis doctoral se han integrado como la herramienta de cálculo estático de muros cortina de la que consta dicho paquete.

- Optimizador estructural: Con el objetivo de que pueda ser utilizado desde otras aplicaciones software, se ha desarrollado una capa por encima del Simulador Estructural que le permite ser invocado desde código desarrollado en C, C++ o .NET, a modo de una librería de enlace dinámico o dll de Microsoft Windows. En ese sentido, el Simulador forma parte como herramienta de cálculo de una aplicación de optimización estructural que está siendo desarrollada con fines de investigación desde el DMMCTE de la UPV.

10.4. Trabajos futuros

Son diversas las líneas de trabajo adicionales que surgen tras esta tesis. La primera de ellas, pasaría por adaptar el Simulador Estructural para facilitar la inclusión de nuevas librerías software relacionadas con la resolución de los problemas numéricos. A modo de ejemplo, podríamos particionar la estructura empleando ParMETIS, en lugar de METIS, e incluso evaluar otras librerías como SCOTCH o PT-SCOTCH.

Resulta ser también del máximo interés el realizar un análisis de escalabilidad del Simulador Estructural bajo un supercomputador con mejores prestaciones que el que hemos empleado, el cual disponga de un número más elevado de nodos y donde no se planifiquen diferentes procesos MPI en un mismo nodo, con los inconvenientes en términos de tiempo de ejecución, incremento de velocidad y

eficiencia que ello supone.

Otra línea de actuación podría ser la aplicación de la computación de altas prestaciones y el paradigma de programación multinivel, basado en MPI y en OpenMP, al análisis no lineal de estructuras, tanto a nivel estático como dinámico, teniendo en consideración la no linealidad geométrica (donde la acción de las cargas exteriores puede provocar grandes deformaciones) y la no linealidad debida a la relación tensión-deformación del material.

A nivel estático no lineal, se deben aplicar procedimientos en los cuales la carga se aplica de manera incremental en un conjunto de pasos, determinando la respuesta para cada uno de ellos mediante la resolución de un sistema de ecuaciones no lineales, por métodos como Newton-Raphson o alguna de sus variantes, acompañado de técnicas como la Longitud de Arco que decidan el valor del incremento de la carga en el siguiente paso de actuación. Lo mismo puede decirse del cálculo dinámico no lineal, donde deberíamos resolver también un sistema de ecuaciones no lineales para cada paso de tiempo en el que se aplica la carga exterior, siempre guiados por métodos de integración directa como los que hemos implementado en esta tesis doctoral.

En consecuencia, los servicios Grid y Cloud desplegados también se verían modificados, para acomodarse a este tipo de análisis, mucho más complejo y costoso en el tiempo que el análisis lineal implementado.

Debido a la elevada demanda computacional que presentará el análisis no lineal, parece clara la necesidad de aplicar técnicas de computación de altas prestaciones y su ejecución en máquinas multiprocesador. En ese sentido, una línea de investigación vendría por desplegar clusters virtuales en la nube que nos proporcionaran la plataforma computacional paralela para lanzar las simulaciones. Precisamente, el proyecto CLUVIEM (Clusters Virtuales Elásticos y Migrables Sobre Infraestructuras Cloud Híbridas, TIN2013-44390-R-AR) financiado por el Ministerio de Economía y Competitividad, en el que participa el doctorando, incluye entre sus objetivos la ejecución de un análisis dinámico no lineal en una plataforma computacional formada por clusters virtuales.

Finalmente, y en el marco de la optimización estructural, se podría desplegar una infraestructura Grid de sobremesa, gestionada por HTCondor, en la que se

aprovechara todo un conjunto de recursos computacionales de los cuales se dispone y que puedan estar ociosos en un momento dado, para ejecutar la cantidad considerable de simulaciones estructurales que conllevan las técnicas de optimización.

Bibliografía

[1] H. H. West and L. F. Geschwindner. *Fundamentals of Structural Analysis.* J. Wiley & Sons, New York, 2001.

[2] Ministerio de Fomento. *Norma de Construcción Sismorresistente NCSE-02. Parte General y Edificación*, 2007.

[3] Cotec. Fundación Para la Innovación Tecnológica. *Documentos Cotec Sobre Necesidades Tecnológicas. 8, Sector de la Construcción*, 1997.

[4] V. Bertero. Lessons learned from recent catastrophic earthquakes and associated research. In *Primera Conferencia International Torroja*, Madrid, 1989. Instituto Torroja.

[5] L. M. Bozzo y A. H. Barbat. *Diseño Sismorresistente de Edificios. Técnicas Convencionales y Avanzadas.* Editorial Reverté, S.A., Barcelona, 2000.

[6] S. Wilkinson and D. Thambiratnam. Simplified procedure for seismic analysis of asymmetric buildings. *Computers & Structures*, 79(32):2833–2845, 2001.

[7] R. W. Clough and J. Penzien. *Dynamics of Structures.* Computers and Structures, Inc., 2004.

[8] A. K. Chopra. *Dynamics of Structures: Theory and Applications to Earthquake Engineering.* Prentice-Hall International Series in Civil Engineering and Engineering Mechanics, 2006.

[9] N. M. Newmark. A method of computation for structural dynamics. *Journal of Engineering Mechanics Division, ASCE*, 85:67–94, 1959.

[10] E. L. Wilson. *A Computer Program for the Dynamic Stress Analysis of Underground Structures.* SESM Report 68-1. Division of Structural Engineering and Structural Mechanics, University of California, Berkeley, 1968.

[11] B. Chapman, G. Jost, and R. Van de Pas. *Using OpenMP: Portable Shared Memory Parallel Programming.* MIT Press, Massachusetts, 2008.

[12] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference.* The MIT Press, Massachusetts, 1996.

[13] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W.D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, S. Zampini, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory. Mathematics and Computer Science Division, 2015.

[14] CERFACS, CNRS, ENS Lyon, INP Toulouse, Inria, University of Bordeaux. *MUltifrontal Massively Parallel Solver Users' Guide*, 2015.

[15] J. E. Román, C. Campos, E. Romero, and A. Tomás. SLEPc Users Manual. Scalable Library for Eigenvalue Problem computations. Technical Report DSIC-II/24/02, Universitat Politècnica de València, 2015.

[16] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide, Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods.* SIAM, Philadelphia, 1998.

[17] I. Foster. Globus Toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, pages 2–13. Springer-Verlag LNCS 3779, 2005.

[18] J. M. Alonso, V. Hernández, and G. Moltó. GMarte: Grid middleware to abstract remote task execution. *Concurrency and Computation-Practice & Experience*, 18(15):2021–2036, 2006.

[19] M. Collier and R. Shahan. *Fundamentals of Azure. Microsoft Azure Essentials.* Microsoft Press, 2015.

[20] Fundación Laboral de la Construcción de Aragón. *Estudio del Sector de la Construcción, Perspectivas de Futuro, Renovación Generacional e Inmigración*, 2009.

[21] Fundación Cotec para la Innovación Tecnológica. *Innovación en Construcción. Informes Sobre el Sistema Español de Innovación*, 2000.

[22] Fundación Cotec para la Innovación Tecnológica, Madrid. *Informe Cotec 2014. Tecnología e Innovación en España*, 2014.

[23] Fundación Cotec para la Innovación Tecnológica. *Informe Cotec 2007. Tecnología e Innovación en España*, 2007.

[24] R. K. Livesley. *Métodos Matriciales Para Cálculo de Estructuras*. H. Blume Ediciones, 1978.

[25] W. McGuire, R. H. Gallagher y R. D. Ziemian. *Matrix Structural Analysis*. John Wiley & Sons, New York, etc., 2000.

[26] O. C. Zienkiewicz, R. L. Taylor y J. Z. Zhu. *El Método de los Elementos Finitos. Volumen 1. Las Bases*. Centro Internacional de Métodos Numéricos en Ingeniería, Barcelona, 2010.

[27] E. Oñate. *Cálculo de Estructuras por el Método de Elementos Finitos*. Centro Internacional de Métodos Numéricos en Ingeniería, Barcelona, 1995.

[28] T. R. Chandrupatla y A. D. Belegundu. *Introducción al Estudio del Elemento Finito en Ingeniería*. Prentice Hall, México, 1999.

[29] R. Arguelles Álvarez. *Cálculo de Estructuras. Tomo II*. Escuela Técnica Superior de Ingenieros de Montes. Sección de Publicaciones, Madrid, 1999.

[30] Ministerio de Vivienda. *Código Técnico de la Edificación*, 2006.

[31] Ministerio de La Presidencia. *Instrucción de Hormigón Estructural (EHE-08)*, 2008.

[32] A. Alonso Durá. *Un Modelo de Integración del Análisis Estructural en Entornos CAD Para Estructuras de Edificación*. Tesis Doctoral. Departamento de Mecánica de los Medios Continuos y Teoría de Estructuras. Universidad Politécnica de Valencia, 2003.

[33] J. L. Batoz, K. J. Bathe, and L. W. Ho. A study of three-node triangular plate bending elements. *International Journal for Numerical Methods in Engineering*, 15:1771–1812, 1980.

[34] C. A. Felippa. A study of optimal membrane triangles with drilling freedoms. *Computer Methods in Applied Mechanics and Engineering*, 192:2125–2168, 2003.

[35] R. C. Hibbeler. *Mecánica de Materiales.* Pearson, Prentice Hall, México, 2006.

[36] F. P. Beer, E. R. Johnston, J. T. DeWolf y D. F. Mazurek. *Mecánica de Materiales.* McGraw-Hill, México, 2010.

[37] J. M. Canet y A. H. Barbat. *Estructuras Sometidas a Acciones Sísmicas. Cálculo por Ordenador.* Centro Internacional de Métodos Numéricos en Ingeniería, Barcelona, 1988.

[38] M. Paz. *Dinámica Estructural: Teoría y Cálculo.* Reverté, Barcelona, 2009.

[39] W. L. Wood. *Practical Time-Stepping Schemes. Oxford Applied Mathematics and Computing Science Series.* Oxford University Press, 1990.

[40] T. J. R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis.* Courier Dover Publications, 2000.

[41] T. C. Fung. Numerical dissipation in time-step integration algorithms for structural dynamic analysis. *Progress in Structural Engineering and Materials*, 5:167–180, 2003.

[42] K. Subbaraj and M. A. Dokainish. A survey of direct time-integration methods in computational structural dynamics-I. Explicit methods. *Computers & Structures*, 32(6):1371–1386, 1989.

[43] K. Subbaraj and M. A. Dokainish. A survey of direct time-integration methods in computational structural dynamics-II. Implicit methods. *Computers & Structures*, 32(6):1387–1401, 1989.

[44] G. G. Dahlquist. A special stability problem for linear multistep algorithms. *BIT*, 3:27–43, 1963.

[45] J. C. Houbolt. A recurrence matrix solution for the dynamic response of elastic aircraft. *Journal of the Aeronautical Sciences*, 17(9):540–550, 1950.

[46] J. T. Chung and G.M. Hulbert. A family of single-step Houbolt time integration algorithms for structural dynamics. *Computer Methods in Applied Mechanics and Engineering*, 118(1-2):1–11, 1994.

[47] T. Belytschko and T. J. R. Hughes. *Computational Methods in Transient Analysis*. North Holland, 1983.

[48] K. J. Bathe and E. Wilson. Stability and accuracy methods of direct integration methods. *Earthquake Engineering and Structural Dynamics*, 1:283–291, 1973.

[49] K. C. Park. Improved stiffly stable method for direct integration of nonlinear structural dynamic equations. *Journal of Applied Mechanics-Transactions of the ASME*, 42(2):464–470, 1975.

[50] O. C. Zienkiewicz. New look at Newmark, Houbolt and other time stepping formulas - Weighted residual approach. *Earthquake Engineering & Structural Dynamics*, 5(4):413–418, 1977.

[51] O. C. Zienkiewicz, W. L. Wood, N. W. Hine, and R. L. Taylor. A unified set of single step algorithms. Part 1. General formulation and applications. *International Journal for Numerical Methods in Engineering*, 20(8):1529–1552, 1984.

[52] W.L. Wood. A unified set of single step algorithms. Part 2. Theory. *International Journal for Numerical Methods in Engineering*, 20(12):2303–2309, 1984.

[53] G. Bazzi and E. Anderheggen. The ρ-family of algorithms for time-step integration with improved numerical dissipation. *Earthquake Engineering & Structural Dynamics*, 10(4):537–550, 1982.

[54] M. G. Katona and O. C. Zienkiewicz. A unified set of single step algorithms. Part 3. The beta-m method, a generalization of the Newmark scheme. *International Journal for Numerical Methods in Engineering*, 21(7):1345–1359, 1985.

[55] H. M. Hilber and T. J. R. Hughes. Collocation, dissipation and overshoot for time integration schemes in structural dynamics. *Earthquake Engineering & Structural Dynamics*, 6(1):99–117, 1978.

[56] H. M. Hilber, T. J. R. Hughes, and R. L. Taylor. Improved numerical dissipation for time integration algorithms in structural dynamics. *Earthquake Engineering & Structural Dynamics*, 5(3):283–292, 1977.

[57] M. Bossak W. L. Wood and O. C. Zienkiewicz. An alpha modification of Newmark's method. *International Journal for Numerical Methods in Engineering*, 15(10):1562–1566, 1980.

[58] C. Hoff and P. J. Pahl. Development of an implicit method with numerical dissipation from a generalized single-step algorithm for structural dynamics. *Computer Methods in Applied Mechanics and Engineering*, 67(3):367–385, 1988.

[59] C. Hoff and P. J. Pahl. Practical performance of the $\theta 1$-method and comparison with other dissipative algorithms in structural dynamics. *Computer Methods in Applied Mechanics and Engineering*, 67(1):87–110, 1988.

[60] C. Hoff, T. J. R. Hughes, G. Hulbert, and P. J. Pahl. Extended comparison of the Hilber-Hughes-Taylor α-method and the $\theta 1$-method. *Computer Methods in Applied Mechanics and Engineering*, 76(1):87–93, 1989.

[61] S. Valliappan and K. K. Ang. η method of numerical integration. *Computational Mechanics*, 5:321–336, 1997.

[62] J. Chung and G. M. Hulbert. A time integration algorithm for structural dynamics with improved numerical dissipation - The Generalized-α method. *Journal of Applied Mechanics*, 60(2):371–375, 1993.

[63] E-H Cho J. Chung and K. Choi. A priori error estimator of the Generalized-alpha method for structural dynamics. *International Journal for Numerical Methods in Engineering*, 55:537–554, 2003.

[64] B. Owren and H. H. Simonsen. Alternative integration methods for problems in structural dynamics. *Computer Methods in Applied Mechanics and Engineering*, 122(1-2):1–10, 1995.

[65] K. K. Tamma and R. R. Namburu. A new finite element based Lax-Wendrof/Taylor-Galerking methodology for computational dynamics. *Computer Methods in Applied Mechanics and Engineering*, 71:137–150, 1988.

[66] K. K. Tamma and R. R. Namburu. Applicability and evaluation of an implicit self-starting unconditionally stable methodology for the dynamics of structures. *Computers & Structures*, 34(6):835–842, 1990.

[67] K. K. Tamma and R. R. Namburu. A generalized γ_s-family of self-starting algorithms for computational structural dynamics. In *Proceedings of the 33rd Structures, Structural Dynamics and Materials Conference, AIAA-92-2330*, Dallas, Texas, 1992.

[68] P. D. Lax and B. Wendroff. Difference schemes for hyperbolic equations with high order accuracy. *Communications on Pure and Applied Mathematics*, 181(17):381–398, 1964.

[69] S. Modak and E. D. Sotelino. The Generalized method for structural dynamics applications. *Advances in Engineering Software*, 33:565–575, 2002.

[70] M. Austin. High-order integration of smooth dynamical systems - Theory and numerical experiments. *International Journal for Numerical Methods in Engineering*, 36(12):2107–2122, 1993.

[71] N. Tarnow and J. C. Simo. How to render 2nd-order accurate time-stepping algorithms 4th-order accurate while retaining the stability and conservation properties. *Computer Methods in Applied Mechanics and Engineering*, 115(3-4):233–252, 1994.

[72] T. C. Fung. Unconditionally stable higher-order Newmark methods by sub-stepping procedure. *Computer Methods in Applied Mechanics and Engineering*, 147(1-2):61–84, 1997.

[73] T. C. Fung. Complex-time-step Newmark methods with controllable numerical dissipation. *International Journal for Numerical Methods in Engineering*, 41(1):65–93, 1998.

[74] E. Hairer, G. Wanner, and S. P. Norsett. *Solving Ordinary Differential Equation, vol. I.* Springer, Berlin, 1991.

[75] M. Borri and C. Botasso. A general framework for interpreting time finite element formulations. *Computational Mechanics*, 13:133–142, 1993.

[76] P. W. Moller. High-order hierarchical A- and L-stable integration methods. *International Journal of Numerical Methods in Engineering*, 36:2607–2624, 1993.

[77] M. Gellert. A new algorithm for integration of dynamic systems. *Computers & Structures*, 9:401–408, 1978.

[78] T. C. Fung. Unconditionally stable higher order accurate hermitian time finite elements. *International Journal of Numerical Methods in Engineering*, 39:3475–3495, 1996.

[79] G. M. Hulbert. Time finite element methods for structural dynamics. *International Journal of Numerical Methods in Engineering*, 33:307–331, 1992.

[80] G. M. Hulbert. A unified set of single-step asymptotic annihilation algorithms for structural dynamics. *Computer Methods in Applied Mechanics and Engineering*, 113:1–9, 1994.

[81] B. W Golley and M. Amer. An unconditionally stable time-stepping procedure with algorithmic damping: A weighted integral approach using two general weights functions. *Earthquake Engineering and Structural Dynamics*, 28:1345–1360, 1999.

[82] K. K. Tamma, X. Zhou, and D. Sha. The time dimension: A theory towards the evolution, classification, characterization and design of computational algorithms for transient/dynamic applications. *Archives of Computational Methods in Engineering*, 7(2):67–292, 2000.

[83] M. Ghassemieh, A. A. Gholampour, and M. Karimi-Rad. A new unconditionally stable time integration method for analysis of nonlinear structural dynamics. *Journal of Applied Mechanics*, 80(2):021024–021024–12, 2012.

[84] T. Belytschko and R. Mullen. Explicit integration of structural problems. *Finite Elements in Nonlinear Mechanics*, 2:697–720, 1997.

[85] C. C. Fu. A method for the numerical integration of equations of motion arising from a finite element analysis. *Journal of Applied Mechanics*, 37(3):599–605, 1970.

[86] R. D. Krieg. Unconditional stability in numerical time integration methods. *Journal of Applied Mechanics*, 40(2)(2):417–421, 1973.

[87] D. M. Trujillo. An unconditionally stable explicit algorithm for structural dynamics. *International Journal for Numerical Methods in Engineering*, 11(10):1579–1592, 1977.

[88] J. Kujawski and R. H. Gallagher. A family of higher-order explicit algorithms for the transient dynamic analysis. *Transactions of The Society for Computer Simulation International*, 1:155–166, 1984.

[89] G. D. Hahn. A modified Euler method for dynamic analysis. *International Journal for Numerical Methods in Engineering*, 32(5):943–955, 1991.

[90] C. Hoff and R. L. Taylor. Higher derivative explicit one step methods for non-linear dynamic problems. Part I: Design and theory. *International Journal for Numerical Methods in Engineering*, 29:275–290, 1990.

[91] C. Hoff and R. L. Taylor. Higher derivative explicit one step methods for non-linear dynamic problems. Part II: Practical calculations and comparisons with other higher order methods. *International Journal for Numerical Methods in Engineering*, 29(2):291–301, 1990.

[92] S. Pezeshk and C. V. Camp. An explicit time integration technique for dynamics analyses. *International Journal for Numerical Methods in Engineering*, 38:2265–2281, 1995.

[93] G. M. Hulbert and J. T. Chung. Explicit time integration algorithms for structural dynamics with optimal numerical dissipation. *Computer Methods in Applied Mechanics and Engineering*, 137(2):175–188, 1996.

[94] G. M. Hulbert and I. Jang. Automatic time step control algorithms for structural dynamics. *Computer Methods in Applied Mechanics and Engineering*, 126(1-2):155–178, 1995.

[95] K. K. Tamma and R. R. Namburu. A robust self-starting explicit computational methodology for structural dynamic applications: Architecture and representations. *International Journal for Numerical Methods in Engineering*, 29(7):1441–1454, 1990.

[96] M. Li, K. K. Tamma, and R. R. Namburu. Evaluation and applicability of explicit self-starting formulations for nonlinear structural dynamics. In *Proceedings of the 33rd Structures, Structural Dynamics and Materials Conference, AIAA-92-2330*, Dallas, Texas, 1992.

[97] E. Hairer and G. Wanner. *Solving Ordinary Differential Equation II, Stiff and Differential-Algebraic Problems.* Springer, Berlin, 1991.

[98] K. Dekker. Comparing stabilized Runge-Kutta methods for semi-discretized parabolic and hyperbolic equations. Technical Report Report-NW 45/77, Stichting Mathematisch Centrum, 1977.

[99] F. Y. Cheng. *Matrix Analysis of Structural Dynamics.* Marcel Dekketer, Inc., 2001.

[100] G. Dahlquist and A. Bjorck. *Numerical Methods.* Prentice-Hall, New York, 1974.

[101] L. F. Shampine. Local error controls in codes for ordinary differential equations. *Applied Mathematics and Computation*, 3(3):189–210, 1977.

[102] J. Braekhus and J. O. Aasen. Experiments with direct integration algorithms for ordinary differential equations in structural dynamics. *Computers & Structures*, 13(1-3):91–96, 1981.

[103] J. L. Humar and E. W. Wright. Numerical methods in structural dynamics. *Canadian Journal of Civil Engineering*, 1(2):179–193, 1974.

[104] P. Henrici. *Discrete Variable Methods in Ordinary Differential Equations.* John Wiley, Nueva York, 1968.

[105] H. Armen, A. Pifko, and H. Levine. Nonlinear finite element techniques for aircraft analysis. In *Aircraft Crash Worthiness, edited by K.K. Saczalski et al.*, pp. 517–548. University Press of Virginia, Charlottesville, 1975.

[106] K. C. Park, C. A. Felippa, and J. A. Deruntz. Stabilization of staggered solution procedures for fluid-structure interaction analysis. In *Computational Methods for Fluid-Structure Interaction Problems, edited by T. Belytschko and T. L. Geers*, 95–124, New York: The American Society of Mechanical Engineers, 1977.

[107] T. Belytschko and R. Mullen. Stability of explicit - implicit mesh partition in time integration. *International Journal for Numerical Methods in Engineering*, 12:1575–1586, 1978.

[108] K. J. Bathe. *Formulation and Computational Algorithms in Finite Element Analysis*. US-German Symposium. DTIC Document, 1977.

[109] T. J. R. Hughes and W. K. Liu. Implicit-explicit finite elements in transient analysis: stability theory. *Journal of Applied Mechanics*, 45(2):371–374, 1978.

[110] T. J. R. Hughes and W. K. Liu. Implicit-explicit finite elements in transient analysis: implementation and numerical examples. *Journal of Applied Mechanics*, 45(2):375–378, 1978.

[111] I. Miranda, R. M. Ferencz, and T. J. R. Hughes. An improved implicit-explicit time integration method for structural dynamics. *Earthquake Engineering & Structural Dynamics*, 18(5):643–653, 1989.

[112] W. J. T. Daniel. Explicit/implicit partitioning and a new explicit form of the Generalized alpha method. *Communications in Numerical Methods in Engineering*, 19(11):909–920, 2003.

[113] M. O. Neal and T. Belytschko. Explicit-explicit subcycling with non-integer time step rations for structural dynamics systems. *Computers and Structures*, 31(6):871–880, 1989.

[114] P. Smolinski, S. Sleith, and T. Belytschko. Stability of an explicit multi-time step integration algorithm for linear structural dynamics equations. *Computational Mechanics*, 18:236–244, 1996.

[115] P. Smolinski. Subcycling integration with non-integer time steps for structural dynamic problems. *Computers & Structures*, 59(2):273–281, 1996.

[116] W. J. T. Daniel. Analysis and implementation of a new constant acceleration subcycling algorithm. *International Journal of Numerical Methods in Engineering*, 40(15):2841–2855, 1997.

[117] W. J. T. Daniel. A study of the stability of subcycling algorithms in structural dynamics. *Computer Methods in Applied Mechanics and Engineering*, 156(1–4):1–13, 1998.

[118] Y. S. Wu and P. Smolinski. A multi-time step integration algorithm for structural dynamics based on the modified trapezoidal rule. *Computer Methods in Applied Mechanics and Engineering*, 187(1–4):641–660, 2000.

[119] C. Chen and J. M. Ricles. Development of direct integration algorithms for structural dynamics using discrete control theory. *Journal of Engineering Mechanics*, 134(8):676–683, 2008.

[120] D. Sha, X. Chen, and K. K. Tamma. Virtual-pulse time integral methodology: a new approach for computational dynamics. Part 1. Theory for linear structural dynamics. *Finite Elements in Analysis and Design*, 20(3):179–194, 1995.

[121] X. Chen, D. Sha, and K. K. Tamma. Virtual-pulse time integral methodology: a new approach for computational dynamics. Part 2. Theory for nonlinear structural dynamics. *Finite Elements in Analysis and Design*, 20(3):195–204, 1995.

[122] K. K. Tamma, X. Chen, and D. Sha. An overview of recent advances and evaluation of the virtual-pulse (VIP) time integral methodology for general structural dynamics problems: computational issues and implementation aspects. *International Journal for Numerical Methods in Engineering*, 39(11):1955–1977, 1996.

[123] K. J. Bathe and E. Wilson. Stability and accuracy analysis of direct integration methods. *Earthquake Engineering Structural Dynamics*, 1:283–291, 1973.

[124] Comité Técnico AEN/CTN 140 Eurocódigos Estructurales. *Eurocódigo 8: Proyecto de Estructuras Sismorresistentes. Parte 1: Reglas Generales, Acciones Sísmicas y Reglas Para Edificación*, 2011.

[125] R. Bonett and L. Pujades. Generación de acelerogramas artificiales compatibles con un espectro de respuesta. aplicación a eventos recientes en colombia y españa. *Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería*, 18(2):297–308, 2002.

[126] L. E. Goodman, E. Rosenblueth, and N. M. Newmark. Aseismic design of elastic structures founded on firm ground. In *Proceedings of the ASCE Structural Division, Vol. 79, separate n. 239*, pages 1–27. ASCE, 1953.

[127] E. Rosenblueth and H. Contreras. Approximate design for multicomponent earthquakes. *Journal of Engineering Mechanics Division*, 103(5):881–893, 1977.

[128] W. Smeby and A. Der Kiureghian. Modal combination rules for multicomponent earthquake excitation. *Earthquake Engineering and Structural Dynamics*, 13(1):1–12, 1985.

[129] C. Menun and A. Der Kiureghian. A replacement for the 30 %, 40 % and SRSS rules for multicomponent seismic analysis. *Earthquake Spectra*, 14(1):153–163, 1998.

[130] O. A. López, A. K. Chopra, and J. J. Hernández. Critical response of structures to multicomponent earthquake excitation. *Earthquake Engineering and Structural Dynamics*, 29:1759–1778, 2000.

[131] J. J. Hernández and O. A. López. Response to three-component seismic motion of arbitrary direction. *Earthquake Engineering and Structural Dynamics*, 31(1):55–77, 2002.

[132] O. A. López, A. K. Chopra, and J. J. Hernández. Adapting the CQC3 rule for three seismic components with different spectra. *Journal of Structural Engineering*, 130:403–410, 2004.

[133] A. K. Gupta. *Response Spectrum Method in Seismic Analysis and Design of Structures.* CRC Press, 1992.

[134] R. L. Jennings. *The Response of Multi-Stoned Structures to Strong Ground Motion.* M.Sc. Thesis, University of Illinois, Urbana, 1958.

[135] Comisión Federal de Electricidad. *Manual de Diseño de Obras Civiles, Diseño por Sismo.* Chap. C.1.3. Instituto de Investigaciones Eléctricas, México, 1993.

[136] G. J. O'Hara and R. O. Belsheim. *Interim Design Values for Shock Design of Shipboard Equipment.* NRL Memorandum Report 1936. Mechanics Division. US Naval Research Laboratory, 1963.

[137] S. A. Anagnostopoulos. Response spectrum techniques for three-component earthquake design. *Earthquake Engineering and Structural Dynamics*, 9(5):459–476, 1981.

[138] E. Rosenblueth and J. Elorduy. Response of linear systems in certain transient disturbances. In *Proceedings of the Fourth World Conference on Earthquake Engineering*, pages 185–196, 1969.

[139] U.S. Nuclear Regulatory Commission. *Regulatory Guide 1.92. Combining Modal Responses and Spatial Components in Seismic Response Analysis. Revision 1.* Office of Standards Development, 1976.

[140] A. K. Gupta and K. Cordero. Combination of modal responses. In *Transactions of the 6th International Conference on Structural Mechanics in Reactor Technology. Paper No. K7/5*, 1981.

[141] E. Wilson, A. Der Kiureghian, and E. P. Bayo. A replacement for the SRSS method in seismic analysis. *Earthquake Engineering and Structural Dynamics*, 9(2):187–194, 1981.

[142] A. Der Kiureghian. A response spectrum method for random vibration analysis of MDF systems. *Earthquake Engineering and Structural Dynamics*, 9(5):419–435, 1981.

[143] U.S. Nuclear Regulatory Commission. *Regulatory Guide 1.92. Combining Modal Responses and Spatial Components in Seismic Response Analysis. Revision 2.* Office of Standards Development, 2006.

[144] A. K. Gupta. Modal combination in response spectrum method. In *Eighth World Conference on Earthquake Engineering*, 1984.

[145] S. Cominetti y F. Jorquera. Determinación del signo de los esfuerzos obtenidos del análisis dinámico por superposición modal espectral. In *Congreso Chileno de Sismología e Ingeniería Antisísmica. IX Jornadas*, Concepción, Chile, 2005.

[146] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing. Design and Analysis of Algorithms.* The Benjamin/Cummings Publishing Company, Inc., California, 1994.

[147] B. Wilkinson and M. Allen. *Parallel Programming. Techniques and Applications Using Networked Workstations and Parallel Computers.* Pearson Prentice Hall, United States of America, 2005.

[148] F. Almeida, D. Giménez, J. M. Mantas y A. M. Vidal. *Introducción a la Programación Paralela.* Cengage Learning Paraninfo, Madrid, 2008.

[149] D. Gove. *Multicore Application Programming. For Windows, Linux, and Oracle Solaris.* Addison-Wesley, 2011.

[150] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach.* Morgan Kaufmann, 1999.

[151] M. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, 1972.

[152] The Top500 List. http://www.top500.org. Accedido el 22/08/2015.

[153] D. R. Butenhof. *Programming with POSIX Threads.* Addison-Wesley, 1997.

[154] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing.* Scientific and Engineering Computation, MIT Press, Massachusetts, 1994.

[155] High Performance Fortran Forum. *High Performance Fortran Language Specification. Versión 1.1*, 1995.

[156] W. D. Gropp and E. Lusk. *User's Guide for MPICH, a Portable Implementation of MPI.* Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[157] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, 2004.

[158] Microsoft MPI. http://msdn.microsoft.com/en-us/library/bb524831(v=vs.85).aspx.

[159] Message Passing Interface Forum. *MPI-2: Extensions to the Message Passing Interface*, 2003.

[160] R. Thakur, R. Rosss, E. Lusk, W. D. Gropp, and R. Latham. *Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation.* ANL/MCS-TM-234. Mathematics and Computer Science Division, Argonne National Laboratory, 2010.

[161] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard. Version 3.1*, 2015.

[162] The OpenMP Architecture Review Board. *OpenMP Application Programming Interface. Version 4.1*, 2015.

[163] V. Eijkhout, J. Langou, and J. Dongarra. Parallel linear algebra software. In *Parallel Processing for Scientific Computing. SIAM. Edited by M. A. Heroux, P. Raghavan and H. D. Simon*, Philadelphia, 2006.

[164] L. A. Drummond and O. A. Marques. An overview of the Advanced CompuTational Software (ACTS) Collection. *ACM Transactions on Mathematical Software*, 31:282–301, 2005.

[165] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, 1979.

[166] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–17, 1988.

[167] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A set of level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[168] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Philadelphia, 1999.

[169] J. J. Dongarra and R. C. Whaley. LAPACK Working Note 94. A User's Guide to the BLACS v1.1. Technical Report UT-CS-95-281, University of Tennessee, 1997.

[170] J. Choi, J. J. Dongarra, and D. Walker. PB-BLAS: A set of parallel block basic linear algebra subprograms. *Concurrency Practice and Experience*, 8(7):517–535, 1996.

[171] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, 1997.

[172] NETLIB home page. http://www.netlib.org.

[173] CBLAS home page. http://www.netlib.org/blas.

[174] CLAPACK home page. http://www.netlib.org/clapack.

[175] W. Qian, Z. Xianyi, Z. Yunquan, and Q. Yi. AUGEM: Automatically generate high performance dense linear algebra kernels on x86 CPUs. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*, Denver, 2013.

[176] R. C. Whaley and A. Petitet. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience*, 35(2):101–121, 2005.

[177] Intel Math Kernel Library. http://software.intel.com/en-us/intel-mkl.

[178] Advanced Micro Devices, Inc., Numerical Algorithms Group Ltd. *AMD Core Math Library (ACML). Versión 4.1.0*, 2008.

[179] PLAPACK home page. http://www.cs.utexas.edu/users/plapack.

[180] FLAME home page. http://www.cs.utexas.edu/ flame/web.

[181] PLASMA home page. http://icl.cs.utk.edu/plasma.

[182] MAGMA home page. http://icl.cs.utk.edu/magma.

[183] G. Karypis. *METIS. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. Version 5.1.0.* Department of Computer Science & Engineering. University of Minnesota, 2013.

[184] G. Karypis and K. Schloegel. *PARMETIS. Parallel Graph Partitioning and Sparse Matrix Ordering Library. Version 4.0.* Department of Computer Science & Engineering. University of Minnesota, 2013.

[185] F. Pellegrini. *SCOTCH and LIBSCOTCH 5.1 User's Guide.* INRIA, IPB & LaBRI, Université Bordeaux I, 2010.

[186] F. Pellegrini. *PT-SCOTCH and LIBSCOTCH 5.1 User's Guide.* INRIA, IPB & LaBRI, Université Bordeaux I, 2010.

[187] C. Chevalier and F. Pellegrini. PT-SCOTCH: A tool for efficient parallel graph ordering. *Parallel Computing*, 34(6-8):318–331, 2008.

[188] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine. The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring. *Scientific Programming*, 20(2):129–150, 2012.

[189] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[190] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[191] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[192] C. Fiduccia and R. Mattheyses. A linear time heuristic for improving network partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, 175–181, 1982.

[193] G. Karypis and V. Kumar. A coarse-grain parallel multilevel k-way partitioning algorithm. In *Proceedings of the 8th SIAM conference on Parallel Processing for Scientific Computing*, 1997.

[194] K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of EuroPar-2000*, 2000.

[195] G. H. Golub and C. F. Loan. *Matrix Computations*. John Hopkins U. Press, 1996.

[196] I. Duff, A. Erisman, and J. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 1989.

[197] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2003.

[198] R. Barrett, M. Berry, T. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, 1994.

[199] A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989.

[200] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, 1981.

[201] M. T. Heath, E. Ng, and B. W. Peyton. Parallel algorithms for sparse linear systems. *SIAM Review*, 33(3):420–460, 1991.

[202] E. J. Haunschmid and C. W. Ueberhuber. *Direct Solvers for Sparse Systems*. Institute for Applied and Numerical Mathematics, Technical University of Vienna.

[203] J. W. H. Liu. The role of elimination trees in sparse factorisation. *SIAM Journal on Matrix Analysis and Applications*, 11(1):134–172, 1990.

[204] A. George, J. W.-H. Liu, and E. G.-Y Ng. Communication results for parallel sparse Cholesky factorization on a hypercube. *Parallel Computing*, 10:287–298, 1989.

[205] A. George, J. W.-H. Liu, and E. Ng. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, New York, 1994.

[206] J. W. H. Liu. The multifrontal method for sparse matrix solution: Theory and practice. *SIAM Review*, 34(1):82–109, 1992.

[207] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, 9(3):302–325, 1983.

[208] I. Duff, N. Gould, M. Lescreiner, and J. L. Reid. The multifrontal method in a parallel environment. In *Advances in Numerical Computacion, M. Cox and S. Hammarling, eds., Oxford University Press*, U.K., 1990.

[209] A. George, M. T. Heath, T. Michael, J. Liu, and E. Ng. Sparse Cholesky factorization on a local-memory multiprocessor. *SIAM journal on Scientific and Statistical Computing*, 9(2):327–340, 1988.

[210] C. Ashcraft, S. C. Eisenstat, and J. W. H. Liu. A fan-in algorithm for distributed sparse numerical factorization. *SIAM Journal on Scientific and Statistical Computing*, 11(3):593–599, 1990.

[211] B. M. Irons. A frontal solution program for finite element analysis. *Journal for Numerical Methods in Engineering*, 2(1):5–32, 1970.

[212] Y. Saad. *Krylov Subspace Methods on Parallel Computers*, volume 95. Computer Science Department, University of Minnesota, 1996.

[213] H. A. Van der Vorst. *Iterative Krylov Methods for Large Linear Systems*, volume 13. Cambridge University Press, 2003.

[214] T. F. Chan and H. A. van der Vorst. *Approximate and Incomplete Factorizations*. Springer, 1997.

[215] Freely available software for linear algebra. http://www.netlib.org/utk/people/JackDongarra/la-sw.html.

[216] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, 184(2-4):501–520, 2000.

[217] P. R. Amestoy. Recent progress in parallel multifrontal solvers for unsymmetric sparse matrices. In *Proceedings of the 15th World Congress on Scientific Computation, Modelling and Applied Mathematics, IMACS 97*, Berlin, 1997.

[218] J. Schulze. Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods. *BIT*, 41(4):800–841, 2001.

[219] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal of Matrix Analysis and Applications*, 23(1):15–41, 2001.

[220] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[221] J.-Y. L'Excellent and W. M. Sid-Lakhdar. A study of shared-memory parallelism in a multifrontal solver. *Parallel Computing*, 40(3-4):34–46, 2014.

[222] I. Chowdhury and J.-Y L'Excellent. *Some Experiments and Issues to Exploit Multicore Parallelism in a Distributed-Memory Parallel Sparse Direct Solver*. Nº 7411. INRIA, Grenoble - Rhône-Alpes, 2010.

[223] P. Hénon, P. Ramet, and J. Roman. PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing*, 28(2):301–321, 2002.

[224] *PaStiX User's manual*, 2013.

[225] P. Hénon, P. Ramet, and J. Roman. A Mapping and Scheduling Algorithm for Parallel Sparse Fan-In Numerical Factorization. In *Proceedings of Euro-Par'99*, volume 1685 of *LNCS*, pages 1059–1067, Toulouse, France, 1999. Springer Verlag.

[226] P. Hénon, P. Ramet, and J. Roman. PaStiX: A Parallel Direct Solver for Sparse SPD Matrices based on Efficient Static Scheduling and Memory Management. In *Tenth SIAM Conference on Parallel Processing for Scientific Computing*, Portsmouth, USA, 2001.

[227] P. Hénon, P. Ramet, and J. Roman. On finding approximate supernodes for an efficient ILU(k) factorization. *Parallel Computing*, 34:345–362, 2008.

[228] O. Schenk and K. Gärtner. *Parallel Sparse Direct and Multi-Recursive Iterative Linear Solvers - PARDISO User Guide Version 5.0.0*, 2014.

[229] A. Kuzmin, M. Luisier, and O. Schenk. Fast methods for computing selected elements of the greens function in massively parallel nanoelectronic device simulations. In F. Wolf, B. Mohr, and D. Mey, editors, *Euro-Par 2013 Parallel Processing*, volume 8097 of *Lecture Notes in Computer Science*, pages 533–544. Springer Berlin Heidelberg, 2013.

[230] J. W. H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 11(2):141–153, 1985.

[231] O. Schenk, K. Gärtner, and W. Fichtner. Efficient sparse lu factorization with left-right looking strategy on shared memory multiprocessors. *BIT*, 40(1):158–176, 2000.

[232] O. Schenk and K. Gärtner. Sparse factorization with two-level scheduling in PARDISO. In *Proceedings of the 10th SIAM conference on Parallel Processing for Scientific Computing*, Portsmouth, Virginia, 2001.

[233] O. Schenk and K. Gärtner. Two-level scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems. *Parallel Computing*, 28(2):187–197, 2002.

[234] O. Schenk and K. Gärtner. Solving unsymmetric sparse systems of linear equations with PARDISO. *Future Generation Computer Systems*, 20(3):475–487, 2004.

[235] O. Schenk and K. Gärtner. On fast factorization pivoting methods for symmetric indefinite systems. *Electronic Transactions on Numerical Analysis*, 23(1):158–179, 2006.

[236] A. Gupta and H. Avron. *WSMP: Watson Sparse Matrix Package. Part I - Direct solution of symmetric systems. Version 15.06.* IBM Research Report RC 21886 (98462), 2015.

[237] A. Gupta. *WSMP: Watson Sparse Matrix Package. Part II - Direct solution of general systems. Version 15.06.* IBM Research Report RC 21888 (98472), 2015.

[238] A. Gupta and H. Avron. *WSMP: Watson Sparse Matrix Package. Part III - Iterative solution of sparse systems. Version 15.06.* IBM Research Report RC RC 24398 (W0711-017), 2015.

[239] A. Gupta. Fast and effective algorithms for graph partitioning and sparse matrix ordering. *IBM Journal of Research and Development*, 41(1–2):171–183, 1997.

[240] A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):502–520, 1997.

[241] A. Gupta, S. Koric, and T. George. Sparse matrix factorization on massively parallel computers. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009.

[242] M. Joshi, A. Gupta, G. Karypis, and V. Kumar. Two-dimensional scalable parallel algorithms for solution of triangular systems. In *Proceedings of the 1997 International Conference on High Performance Computing (HiPC)*, 1997.

[243] hypre, High Performance Preconditioners. User's manual. Technical report, 2015.

[244] D. Hysom and A. Pothen. A scalable parallel algorithm for incomplete factor preconditioning. *SIAM Journal on Scientific Computing*, 22(6):2194–2215, 2000.

[245] E. Chow. Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns. *International Journal of High Performance Computing Applications*, 15(1):1–17, 1990.

[246] A. Brandt. Multi-level adaptative solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[247] V. E. Henson and U. M. Yang. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics: Transactions of IMACS*, 41(1):155–177, 2002.

[248] Elemental home page. http://libelemental.org.

[249] Triangle home page. http://www.cs.cmu.edu/ quake/triangle.html.

[250] TetGen home page. http://wias-berlin.de/software/tetgen.

[251] PARTY home page. http://www2.cs.uni-paderborn.de/cs/ag-monien/PERSONAL/ROBSY/party.html.

[252] SuperLU home page. http://crd-legacy.lbl.gov/ xiaoye/SuperLU.

[253] ESSL - IBM's math library home page. http://www-03.ibm.com/systems/power/software/essl.

[254] SPAI home page. http://cccs.unibas.ch/lehre/software-packages.

[255] ML home page. http://trilinos.org/packages/ml.

[256] SUNDIALS home page. http://computation.llnl.gov/casc/sundials.

[257] ADIFOR home page. http://www.mcs.anl.gov/research/projects/adifor.

[258] H. A. van der Vorst and G. H. Golub. 150 years old and still alive: eigenproblems. In I.S. Duff and G.A. Watson, editors, *The State of the Art in Numerical Analysis*, pages 93–120. Oxford University Press, 1997.

[259] G. H. Golub and H. A. van der Vorst. Eigenvalue computation in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1-2):35–65, 2000.

[260] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. SIAM, Philadelphia, 2011.

[261] B. N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, Englewood Cliffs, NJ, 1980.

[262] A. F. Bertolini. Review of eigensolution procedures for linear dynamic finite element analysis. *Applied Mechanics Reviews*, 51(2):155–172, 1998.

[263] T. Ericsson and A. Ruhe. The spectral transformation Lanczos method for the numerical-solution of large sparse generalized symmetric eigenvalue problems. *Mathematics of Computation*, 35(152):1251–1268, 1980.

[264] B. Nour-Omid, B. N. Parlett, T. Ericsson, and P. S. Jensen. How to implement the spectral transformation. *Mathematics of Computation*, 48(178):663–673, 1987.

[265] K. J. Bathe and E. Wilson. Solution methods for eigenvalue problems in structural dynamics. *International Journal for Numerical Methods in Engineering*, 6:213–226, 1973.

[266] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45(4):255–282, 1950.

[267] W. E. Arnoldi. The principe of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9:17–29, 1951.

[268] Y. Saad. Variations on Arnoldi method for computing eigenelements of large unsymmetric matrices. *Linear Algebra and its Applications*, 34:269–295, 1980.

[269] G. H. Golub and R. Underwood. The block Lanczos method for computing eigenvalues. In J. R. Rice, editor, *Mathematical Software III*, pages 364–377. Publ. No. 39 of the Mathematical Research Center, Academic Press, 1977.

[270] D. C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM Journal on Matrix Analysis and Applications*, 13(1):357–385, 1992.

[271] G. W. Stewart. A Krylov Schur algorithm for large eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 23(3):601–614, 2002.

[272] K. Wu y H. Simon. Thick-restart Lanczos method for large symmetric eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(2):602–616, 2000.

[273] E. R. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *Journal of Computational Physics*, 17(1):87–94, 1975.

[274] R. B. Morgan and D. S. Scott. Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices. *SIAM Journal on Scientific and Statistical Computing*, 7(3):817–825, 1986.

[275] G. L. G. Sleijpen and H. A. Van der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM Review*, 42(2):267–293, 2000.

[276] A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing*, 23(2):517–541, 2001.

[277] G. Gambolati, F. Sartoretto, and P. Florian. An orthogonal accelerated deflation technique for large symmetrical eigenproblems. *Computer Methods in Applied Mechanics and Engineering*, 94(1):13–23, 1992.

[278] L. Bergamaschi and M. Putti. Numerical comparison of iterative eigensolvers for large sparse symmetric positive definite matrices. *Computer Methods in Applied Mechanics and Engineering*, 191(45):5233–5247, 2002.

[279] V. Hernández, J. E. Román, A. Tomás, and V. Vidal. A survey of software for sparse eigenvalue problems. Technical Report SLEPc STR-6, Universitat Politècnica de València, 2009.

[280] K. J. Maschhoff and D. C. Sorensen. PARPACK: An efficient portable large scale eigenvalue package for distributed memory parallel architectures. In *J. W. Wasniewski, J. Dongarra, K. Madsen and D. Olesen, editors. Applied Parallel Computing in Industrial Problems and Optimization, Volume 1184. Lecture Notes in Computer Science.* Springer-Verlag, 1996.

[281] O. A. Marques. BLZPACK: Description and user's guide. Technical Report TR/PA/95/30, CERFACS, Toulouse, 1995.

[282] E. Polizzi. Density-matrix-based algorithm for solving eigenvalue problems. *Physical Review B*, 79:115112–1–115112–6, 2009.

[283] M. E. Argentati A. V. Knyazev, I. Lashuk and E. Ovchinnikov. Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX) in hypre and PETSc. *SIAM Journal on Scientific Computing*, 25(5):2224–2239, 2002.

[284] A. Stathopoulos and J. R. McCombs. PRIMME: PReconditioned Iterative MultiMethod Eigensolver: Methods and software description. *ACM Transactions on Mathematical Software*, 37(2):21:1–21:30, 2010.

[285] K. Wu and H. Simon. Thick-Restart Lanczos method for large symmetric eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(2):602–616, 2000.

[286] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers Inc., 2003.

[287] F. Berman, A. J. G. Hey, and G. C. Fox. *Grid Computing. Making the Global Infrastructure a Reality.* Wiley, 2003.

[288] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[289] V. Berstis. *Fundamentals of Grid Computing.* Redbooks Paper. IBM, 2002.

[290] NEES home page. http://www.nees.org.

[291] OpenSees home page. http://opensees.berkeley.edu.

[292] EGEE home page. http://eu-egee.org.

[293] EGI home page. http://www.egi.eu.

[294] ES-GNI home page. http://www.es-ngi.es.

[295] UNICORE home page. http://www.unicore.eu.

[296] EGI Software Repository. http://repository.egi.eu/category/umd_releases.

[297] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[298] H. Kreger. *Web Services Conceptual Architecture (WSCA 1.0).* IBM Software Group, 2001.

[299] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the Grid. In *Grid computing: making the global infrastructure a reality*, pages 217–249. John Wiley & Sons, 2003.

[300] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. *Web Services Architecture.* W3C Working Group Note 11, 2004.

[301] OASIS. Advancing Open Standards for Information Society. *Web Services Resource Framework (WSRF) - Primer v1.2*, 2006.

[302] OASIS. Advancing Open Standards for Information Society. *Web Services Resource 1.3 (WS-Resource)*, 2006.

[303] OASIS. Advancing Open Standards for Information Society. *Web Services Base Notification 1.3 (WS-BaseNotification)*, 2006.

[304] M. Feller, I. Foster, and S. Martin. GT4 GRAM: A functionality and performance study. In *Proceedings of TERAGRID 2007 Conference*, 2007.

[305] B. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raiu, and I. Foster. The Globus striped GridFTP framework and server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005.

[306] W. Allcock, I. Foster, and R. Madduri. Reliable data transport: A critical service for the Grid. In *Building service based grids workshop, Global Grid Forum*, volume 11. Citeseer, 2004.

[307] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripenu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Giggle: A framework for constructing scalable replica location services. In *Proceedings of the High Performance Networking and Computing*. IEEE Computer Society, 2002.

[308] K. Czajkowskiy, S. Fitzgeraldz, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*. IEEE Press, 2001.

[309] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for Grid services. In *Proceedings of Twelfth IEEE International Symposium on High-Performance Distributed Computing (HPDC-12)*. IEEE Computer Society, 2003.

[310] OASIS. Advancing Open Standards for Information Society. *OASIS Web Services Security (WSS) TC*, 2006.

[311] D.Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[312] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[313] S. Venugopal, R. Buyya, and Lyle Winton. A Grid Service Broker for Scheduling e-science Applications on Global Data Grids. *Concurrency and Computation: Practice and Experience*, 18(6):685–699, 2006.

[314] E. Huedo, R. S. Montero, and I. M. Llorente. A framework for adaptive execution on Grids. *Journal Software - Practice and Experience*, 37(7):631–651, 2004.

[315] Open Grid Forum. *Distributed Resource Management Application API Version 2*, 2012.

[316] R. Buyya, J. Broberg, and A. Goscinski. *Cloud Computing. Principles and Paradigms.* John Wiley & Sons, 2011.

[317] K. Hwang, G. C. Fox, and J. Dongarra. *Distributed and Cloud Computing. From Parallel Processing to the Internet of Things.* Morgan Kaufmann, 2012.

[318] P. Mell and T. Grance. Special Publication 800-145. The NIST Definition of Cloud Computing. Technical report, Gaithersburg, MD, United States, 2011.

[319] L. J. Aguilar. *Computación en la nube. Estrategias de Cloud Computing en las empresas.* Marcombo, 2013.

[320] Amazon Web Services. *Amazon Elastic Compute Cloud: Guía de introducción*, 2015.

[321] D. Betts, S. Densmore, R. Dunn, M. Narumoto, E. Pace, and M. Woloski. *Moving Applications to the Cloud on the Microsoft Windows Azure Platform.* Microsoft, 2010.

[322] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, 2002.

[323] VENUS-C project website. http://www.venus-c.eu.

[324] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, and M. Theimer. *OGSA Basic Execution Service. Version 1.0.* Open Grid Forum, 2008.

[325] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. *Job Submission Description Language (JSDL) Specification, Version 1.0.* Open Grid Forum, 2008.

[326] D. Lezzi, G. Brasche, and H. Soncu. *D6.4 - Programming Models - Prototypes.* VENUS-C. Virtual multidisciplinary EnviroNments USing Cloud Infraestructures, 2010.

[327] SNIA. *Cloud Data Management Interface (CDMI TM). Versión 1.1.0*, 2014.

[328] D. Lezzi, R. Rafanell, R .M. Badia, J. Lordan, and E. Tejedor. COMPSs in the VENUS-C Platform: Enabling e-Science Applications on the Cloud. In *Proceedings of 5th Iberian Grid Infrastructure Conference (IBERGRID 2011)*, 2011.

[329] C. Geuer-Pollmann. *The VENUS-C Generic Worker.* 1st anual SICS Cloud Day, 2011.

[330] Y. Simmhan, C. van Ingen, G. Subramanian, and J. Li. Bridging the gap between desktop and the cloud for eScience applications. In *IEEE Cloud*, 2010.

[331] E. Tejedor and R. M. Badia. COMP Superscalar: Bringing GRID superscalar and GCM together. In *Proceedings of 8th IEEE International Symposium on Cluster Computing and the Grid*, 2008.

[332] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. IaaS Cloud Architecture: From virtualized datacenters to federated Cloud infrastructures. *IEEE Computer*, 45:65–72, 2012.

[333] I. Goiri, J. Guitart, and J. Torres. Elastic management of tasks in virtualized environments. In *XX Jornadas de Paralelismo (JP 2009)*, pages 671–676, 2009.

[334] D. R. Mackay and K. H. Law. A parallel implementation of a generalized Lanczos procedure for structural dynamic analysis. *International Journal of High Speed Computing*, 8(2):171–204, 1996.

[335] Página web de Architrave. http://www.architrave.es.

[336] Mono Project home page. http://www.mono-project.com.

[337] J. W. Ruge and K. Stüben. Algebraic multigrid (amg). In S.F. McCormick, editor, *Multigrid Methods, Frontiers in Applied Mathematics, Vol. 5*, Philadelphia, 1986. SIAM. pages 73-130.

[338] R. Latham, R. Ross, and R. Thakur. The impact of file systems on MPI-IO scalability. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 3241 of *Lecture Notes in Computer Science*, pages 87–96. 2004.

[339] SAP2000. http://www.csiamerica.com/products/sap2000.

[340] Grid4build como aplicación perteneciente a EGI. https://appdb.egi.eu/#/store/software/grid4build.

[341] Página web de PrefSuite. http://www.prefsuite.com.