Universidad Politécnica de Valencia
Departamento de Sistemas Informáticos y Computación
Máster en Computación Paralela y Distribuida

Parametrización en Esquemas Paralelos Divide y Vencerás

Documentación presentada por **Murilo Do Carmo Boratto** como Tesis de Máster en el programa de Computación Paralela y Distribuida, en el Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia.

Valencia, Diciembre de 2007

Parametrización en Esquemas Paralelos Divide y Vencerás

Documentación presentada como Tesis de Máster en el programa de Computación Paralela y Distribuida, en el Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia. Directores de Tesis: Dr. Antonio M. Vidal Maciá del Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia y Dr. Domingo Giménez Cánovas del Departamento de Informática y Sistemas de la Universidad de Murcia.

Valencia, Diciembre de 2007

Dr. Antonio M. Vidal Maciá

Dr. Domingo Giménez Cánovas

"A vossa bondade e misericórdia hão de seguir-me por todos os dias de minha vida."

A Benedita

Resumen

El presente trabajo se inscribe en el campo de la computación paralela, y más en concreto en el desarrollo y utilización de técnicas de autooptimización en sistemas paralelos de esquemas algorítmicos Divide y Vencerás. Estos esquemas se utilizan en la solución de gran cantidad de problemas, tanto en el ámbito académico como en métodos numéricos para problemas de alto coste con aplicación de la técnica a diversos campos científicos. La tesis pretende estudiar técnicas que permitan el desarrollo de rutinas paralelas que puedan adaptarse automáticamente a las características del sistema paralelo donde se ejecutan, de manera cercana a la óptima en cuanto a tiempo de ejecución. De esta forma se proporcionaría a los usuarios rutinas capaces de ejecutarse eficientemente en el sistema donde se esté trabajando, independientemente de las características del sistema y de los conocimientos que sobre computación paralela tenga el usuario. Se trabajará en los sistemas más usados actualmente en la resolución de problemas científicos de alto coste computacional (secuenciales, multicomputadores homogéneos y heterogéneos). El trabajo que se propone está basado en desarrollar las rutinas a optimizar junto con una técnica de instalación que, basándose en el modelado y parametrización de las rutinas, realiza una selección de parámetros con los que se obtienen ejecuciones cercanas a la óptima. La técnica se ha aplicado con éxito en álgebra lineal, y se pretende estudiar y adecuar la técnica a diversos esquemas algorítmicos basados en el paradigma Divide y Vencerás.

Palabras clave: Divide y Vencerás, esquemas algorítmicos, autooptimización, programación paralela.

Abstract

The present work has been developed in the field of parallel computing, and more concretely in the development and use of techniques of autooptimization in parallel systems of Divide and Conquer algorithmic schemes. These schemes are used in the solution of a large number of problems, in the academic world and in numerical methods for problems of high cost with applications in several scientific fields. The thesis studies techniques that allow the development of parallel routines that can autooptimize for adapting to the characteristics of the parallel system where they are executed, in such a way they are executed with an execution time close to the optimum. In this way the user would be provided with routines able to execute in the system where he is working, independently of the characteristics of the system and the knowledge of the user on parallel computing. The work is done in the type of systems more used at the moment in the resolution of science problems of high computacional cost (sequential, homogeneous and heterogenous). The main idea consists on the development of the routines together with an optimization technique with which at installation time the characteristics of the system are studied. The technique is based on the modellization and parametrization of the routines, and a selection of parameters with which an execution time close to the optimum is obtained at running time. The technique has been successfully applied to linear algebra routines, and in this work it is studied and adapted to algorithmic schemes based on the Divide and Conquer paradigm.

Keywords: Divide and Conquer, algorithmic schemes, autooptimization, parallel computing.

Agradecimientos

En primer lugar, quisiera agradecer a todas las personas e instituciones que han colaborado directamente en la realización de este trabajo de investigación:

- A mis directores de investigación. A Antonio Vidal y Domingo Giménez Cánovas, por la paciencia y el entusiasmo que han depositado en este trabajo, durante todo su desarrollo, desde sus primeros pasos hasta su término.
- A los centros de investigación que he estado, entre ellos destacaría a la Universidad Politécnica de Valencia, por haber cedido su software y las máquinas para la realización de algunos de los experimentos mostrados en este trabajo. Al Departamento de Arquitectura de Computadores y Sistemas Operativos de la Universidad Autónoma de Barcelona, por el período de estancia en un proyecto de investigación.
- A otros investigadores que han aportado multitud de ideas interesantes: Victor Manuel García, Pedro Alonso, Miguel Óscar Bernabeu y Danyel León. En especial a los investigadores Alexey Lastovetsky, Ravi Reddy y el grupo de Computación Heterogénea de la Universidad de Dublin por la gran aportación de ideas durante mi corta estancia investigadora.
- A los proyectos de investigación de la Comisión Interministerial de la Ciencia y Tecnología (CICYT) (MCYT y FEDER TIC2003-08238-C02-02) y de la Fundación Séneca (02973/PI/05) por la ayuda a la realización de proyectos de investigación.

◇ A mis compañeros del Máster: Gabri, Robert, Alberto, Elizabeth, Julio, Matías, Andrés, Eloy, Miguel y Ester, por las tardes que pasamos juntos enfrentando frío, calor, clases, prácticas, autómatas, grid, cansancios y diversión.

◇ A la gente del DSIC: Rosa, Benito, Amparo, Javi, ... que me ayudaron siempre en todos los momentos.

◇ A mis amigos Doctores, que me enseñaron que se podía ser Doctor sin tener que estudiar medicina.

◇ Al Profesor Dr. Josemar Rodriguez, por la inolvidable ayuda.

◇ A mis amigos de toda la vida: Pimenta, Leandro Coelho, Manuela, Cris, Evandro, Cybele, Ana, Paulinha, Tiburcio, Cinthia, Adriana, Íldima y Fernanda por estar siempre en mi vida.

◇ A Gisa, 'por gostar de mim mais do que deveria'.

◇ A mi familia y al resto de personas que me han ayudado a realizar este trabajo. GRACIAS A TODOS.

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Capítulo 1

Parametrización en Esquemas Algorítmicos y Conceptos de Computación Paralela

En este capítulo se introducen los conceptos de la parametrización en esquemas algorítmicos, presentando las principales características y conceptos de los modelos secuenciales y paralelos utilizados y de las técnicas de optimización en sistemas de alto rendimiento, y también las herramientas usadas en la experimentación. Se finaliza este capítulo presentando los objetivos y propuestas de esta tesis.

1.1. Introducción

Durante los últimos años se han desarrollado distintas técnicas de autooptimización de rutinas paralelas con el fin de conseguir rutinas que se adapten automáticamente a las características del sistema de cómputo, reduciendo así el periodo de tiempo necesario para obtener software optimizado para un nuevo sistema, y que sean capaces de ejecutarse de manera eficiente independientemente de los conocimientos del usuario sobre programación paralela. Los principales usuarios potenciales de los sistemas paralelos de alto rendimiento son científicos e ingenieros que tienen problemas de alto coste computacional, pero que normalmente no tienen conocimientos profundos de paralelismo. Las técnicas de autooptimización se han venido aplicando en diferentes campos [1, 2], y especialmente en rutinas de álgebra lineal [3, 4, 5] que constituyen el elemento básico de cómputo en la resolución de muchos problemas científicos.

Una de las técnicas utilizadas para el desarrollo de rutinas con capacidad de autooptimización es la técnica de parametrización del modelo del tiempo de ejecución [6, 7, 8, 9]. Los modelos así obtenidos pueden ser utilizados para tomar algunas decisiones que permitan reducir el tiempo de ejecución de la rutina. La metodología seguida consiste en identificar parámetros del algoritmo y del sistema para analizar el algoritmo tanto teórica como experimentalmente, con el fin de determinar la influencia del valor de los parámetros del sistema en la selección de los mejores valores de los parámetros del algoritmo. En el grupo en el que se ha realizado este trabajo, se disponía de experiencia en la aplicación de técnicas de optimización automática en métodos de Jacobi unilaterales para el problema de valores propios [10], factorización LU, QR y el problema de mínimos cuadrados de matrices Toeplitz [11]. Actualmente se está trabajando en la aplicación de las ideas adquiridas anteriormente a esquemas algorítmicos, como esquemas de programación dinámica [12, 13, 14], y también en algoritmos de recorrido de árboles de soluciones [14, 15]. El presente trabajo se inscribe dentro de este campo de autooptimización de esquemas algorítmicos paralelos, estudiando las aplicaciones de estas técnicas de optimización a algoritmos en que se pueda hacer una parametrización del tiempo de ejecución, centrados en esquemas Divide y Vencerás.

Para este trabajo se propuso el análisis y la aplicación de técnicas automáticas de optimización. La idea básica de esta optimización consiste en obtener el tiempo de ejecución de las rutinas en la forma: *t(n) = f(n, AP, SP)* [7, 10, 12, 14], donde n representa el tamaño del problema, SP parámetros del sistema, y AP representa los parámetros algorítmicos. Se pretende estudiar técnicas que permitan el desarrollo de rutinas paralelas que puedan adaptarse automáticamente a las características del sistema paralelo donde se ejecutan, de manera que se ejecuten de una manera cercana a la óptima en cuanto a tiempo de ejecución. De esta forma se proporcionaría a los usuarios rutinas capaces de ejecutarse eficientemente en el sistema donde se esté trabajando, independientemente de las características del sistema y de los conocimientos que sobre computación paralela tenga el usuario. Se trabajó sobre los sistemas más usados actualmente en la resolución de problemas científicos de alto coste computacional (secuenciales, multicomputadores homogéneos y heterogéneos).

Primero para probar la metodología desarrollada, como ejemplo, se estudian algoritmos de ordenación por mezcla y ordenación rápida. En estos casos los parámetros del sistema son los costes de las operaciones aritméticas básicas y de la comunicación, y los parámetros algorítmicos podrían ser: el nivel de la recursión, el método directo utilizado para solucionar el problema, y el número de procesadores a utilizar. Se analizan algunos métodos de estimación de estos parámetros. En tiempo de ejecución, el valor de los parámetros algorítmicos con los que se soluciona el problema es obtenido estimando los valores a través del modelo teórico, en el cual se sustituyen los valores de los parámetros del sistema, obtenidos en un proceso previo de instalación.

Y por último se han estudiado las técnicas de autooptimización sobre un esquema Divide y Vencerás para el Cálculo de Valores Propios y su adecuación a sistemas paralelos heterogéneos, comparando las prestaciones del algoritmo desarrollado con una herramienta científica que utilizan el mismo paradigma para el mismo cálculo.

1.2. Computación Paralela

Factores como las limitaciones computacionales, y la necesidad de potencias computacionales elevadas, propician el desarrollo de nuevas tecnologías, lo que ha permitido una mayor velocidad de los procesadores. El surgimiento de computadores que realizan varias operaciones simultáneamente, haciendo que el usuario pueda dividir el trabajo en procesos independientes para reducir el tiempo de ejecución, favorecerá la implementación de aplicaciones paralelas y distribuidas. En este trabajo se destaca la disponibilidad de la computación paralela como una herramienta computacional muy potente, utilizada para solucionar problemas algorítmicos de altas prestaciones. En esta sección se presentan los modelos de programación paralela y algunos conceptos relacionados, junto con las disposiciones de redes de interconexiones utilizadas.

1.2.1. Paradigma de Memoria Distribuida

En los multiprocesadores con memoria distribuida cada elemento de proceso tiene su propia memoria local donde almacena sus datos, no existiendo una memoria global común. Una red de interconexión une los distintos elementos de proceso entre sí. La comunicación entre los procesadores se realiza mediante paso de mensajes. Es frecuente encontrar este paradigma en computadores que disponen de cierto grado de organización paralela, presentando varios niveles de paralelismo.

1.2.2. Redes de Computadores

Las redes de interconexión se utilizan tanto en los multiprocesadores de memoria compartida como en los de memoria distribuida. Una red de interconexión es el conjunto de interruptores, líneas eléctricas de enlace, software y hardware de gestión que permite conectar varios elementos de proceso y/o módulos de memoria entre sí. Podemos clasificar las redes de interconexión desde distintos puntos de vista: por la forma en que transmiten físicamente la información, por el tiempo

que emplean en transmitirla, por su funcionalidad, por la manera en que están construidas, etc. Considerando el punto de vista de su funcionalidad, podemos distinguir entre redes con latencia de comunicación uniforme y redes con latencia de comunicación no uniforme. En el primer caso, el tiempo de comunicación asociado a un determinado mensaje entre cualquier pareja de nodos de la red es constante, independientemente de su posición en la misma o de la distancia física que haya entre ellos. En el caso de las redes con latencia de comunicación no uniforme, el tiempo de comunicación de un determinado mensaje entre cualquier pareja de nodos de la red depende de la distancia entre los nodos. En la Figura 1.1 se muestran los principales tipos de enlaces físicos de redes, algunos de ellos los encontramos en las plataformas de experimentación (por ejemplo: el enlace físico Toro Bidimensional lo encontramos en la plataforma **KEFREN**).

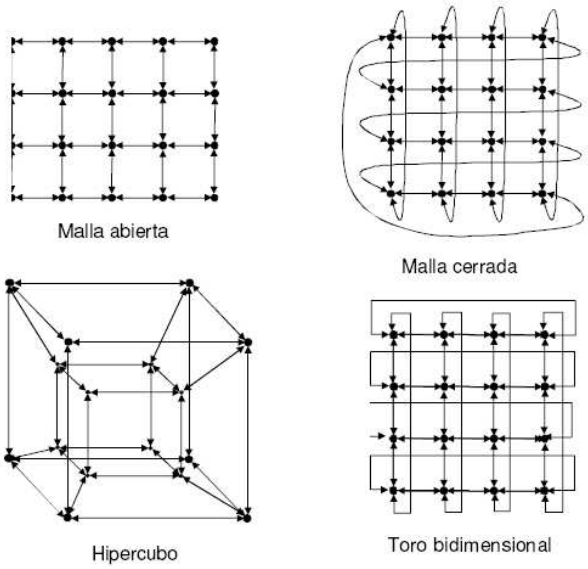

Figura 1.1: Tipos de Enlaces Físicos de Redes de Computadores.

1.3. Autooptimización en la Computación Paralela

1.3.1. Sistemas de Autooptimización en Álgebra Lineal

A lo largo de los últimos años uno de los desafíos más urgentes que enfrenta la computación científica es la tentativa de obtener algoritmos más robustos y eficientes, con un menor tiempo de programación y unas menores necesidades de computación, lo que conlleva un menor tiempo de ejecución. En este contexto surge el concepto de optimización. Se trata de tener herramientas capaces de encontrar respuestas que proporcionan el mejor resultado, en cuanto al uso más eficiente de los recursos, tales como el tiempo de ejecución. El creciente uso de este concepto generó el surgimiento de algunas nuevas técnicas de autooptimización. Tradicionalmente, la obtención de rutinas optimizadas requería de un gran tiempo de programación, y las optimizaciones eran únicas para cada plataforma, de modo que la técnica era incompatible con la evolución del hardware, e inaccesible a usuarios no expertos. Las primeras soluciones para mejorar estas técnicas se basaban en soluciones específicas para determinados tipos de problemas.

Algoritmos basados en Kernels Eficientes

La optimización basada en kernels de álgebra lineal adquirió gran interés en los últimos años. Se realiza una creación de núcleos (kernels) secuenciales de álgebra lineal, que realizan un profundo trabajo de ajuste a bajo nivel, muy cercano al nivel hardware, durante su instalación en una plataforma dada. El más conocido de estos núcleos es BLAS [16]. Estos núcleos consisten en una base sobre la que se van construyendo los sucesivos niveles superiores de rutinas de álgebra lineal. En los algoritmos por bloques la mayor parte de las implementaciones se basan en la utilización de estos núcleos, debido a su uso extensivo e intensivo en gran parte de las rutinas de álgebra lineal de nivel medio y alto.

Otro proyecto basado en kernels eficientes es ATLAS [17, 18], que se caracteriza por ser una versión de BLAS con autooptimización, la cual se adecúa al hardware de la plataforma sobre la que se lleva a cabo la ejecución. Determina experimentalmente, en tiempo de compilación, algunos parámetros, entre ellos el tamaño de bloque y el nivel de bucles necesarios de cara a ofrecer las máximas prestaciones.

Autooptimización de Rutinas Paralelas de Álgebra Lineal Densa

El desarrollo de rutinas de optimización automática para álgebra lineal densa para plataformas paralelas [19] tiene como principal característica la elección apropiada de una serie de parámetros ajustables como son el número de procesadores a utilizar, la topología lógica de éstos, el tamaño del bloque de cálculo, distribución del trabajo a realizar entre los procesadores de la plataforma, selección de la librería básica de donde tomar la rutina para llevar a cabo cada operación y elección del mejor algoritmo con el que resolver un problema de entre varios equivalentes. En un principio, estos tipos de aplicación hacen uso de un modelo analítico del tiempo de ejecución de cada rutina. De esta manera se consigue que el método de ajuste automático realice una búsqueda de la mayor optimización posible del tiempo de ejecución.

1.3.2. Desarrollo de Herramientas para Autooptimización

En esta sección describiremos las herramientas que utilizaremos para la autooptimización a lo largo de este trabajo, a fin de comparar la eficiencia que se obtiene con estas herramientas con la obtenida con la metodología propuesta para un ambiente heterogéneo.

mpC (The mpC Parallel Programming Language)

Como ejemplo podemos citar el proyecto mpC [20, 21], que es una herramienta para obtener autooptimización utilizando un lenguaje paralelo que, entre el usuario y el código final de las rutinas paralelas, permite la codificación de software sobre sistemas paralelos heterogéneos sin que el usuario tenga que manejar mucha información de cada plataforma donde se puede ejecutar. Las rutinas escritas se adaptan automáticamente a las condiciones del sistema en el momento en que se ponen a funcionar, realizando un chequeo para ver los procesadores disponibles, las prestaciones de sus enlaces con la red y la carga que cada procesador soporta.

HeteroMPI (Heterogeneous MPI)

El principal objetivo de HeteroMPI [22, 23, 24, 25] es automatizar y optimizar la selección de un grupo de procesos, de forma que puedan ejecutar un algoritmo heterogéneo de la forma más eficiente posible. Para conseguir este objetivo, HeteroMPI ofrece un lenguaje reducido para la especificación del modelo de rendimiento. Éste lenguaje es un subconjunto de mpC, el cual permite al programador definir explícitamente una red abstracta y distribuir datos, cálculos y comunicaciones en dicha red. Partiendo de este modelo, HeteroMPI automáticamente adapta la red abstracta a una red real, ajustando dinámicamente el modelo de rendimiento a parámetros específicos de la red, tales como la potencia de cálculo de los procesadores o la capacidad de los enlaces de comunicación. Se puede destacar como ventajas inmediatas de su utilización que:

- ◇ Sirve como una primera aproximación algorítmica al problema.
- ◇ Ahorro de aprendizaje de un lenguaje de programación.
- ◇ Permite partir de una aplicación homogénea.

1.3.3. Autooptimización de Rutinas usadas en la Computación Científica

En esta sección se destacan una serie de propuestas cuyo objetivo principal es aportar a la computación científica una cierta capacidad de modificarse a sí misma de manera automática, de cara a adaptarse lo mejor posible a las condiciones del entorno donde se esté utilizando, como campo de aplicación, la computación.

(1994): Resolución de algoritmos de Ordenación y Ecuaciones Diferenciales

En el trabajo de Brewer [1] se presenta una arquitectura de un sistema de gestión de una librería de alto nivel donde, automáticamente, se selecciona la mejor implementación entre varias disponibles en esta librería, para resolver un problema dado. Además, el sistema puede determinar el valor óptimo de una serie de parámetros específicos de la implementación escogida. Esta metodología ha sido aplicada a algunas rutinas de ordenación y otras de resolución de ecuaciones diferenciales sobre diferentes plataformas paralelas.

(1997): Resolución de algoritmos de Álgebra Lineal con Recursividad

En el proyecto LAWRA (Linear Algebra With Recursive Algorithms) [26] el estudio se centra en mejorar el uso de la jerarquía de memoria de las plataformas actuales cuando ejecutan rutinas de álgebra lineal. Los autores de LAWRA plantean usar ese formato por bloques, teniendo como base la librería BLAS como formato general de almacenamiento para los datos de matrices y vectores densos. De esta manera las rutinas de BLAS llamadas recibirían sus operandos en el formato que precisan.

(2001): Resolución de la Transformada de Fourier (FFTW)

En el proyecto FFTW [2] el principal objetivo es la minimización automática del tiempo de resolución de transformadas rápidas de Fourier (FFT, fast Fourier transforms). Consiste en un generador de código que, durante la instalación, genera automáticamente trozos de código especializados en resolver FFT de un determinado tamaño fijo. Estos trozos de código son teóricamente óptimos según un modelo analítico del tiempo de ejecución donde queda recogida la utilización de la memoria de la plataforma así como la cantidad de cálculo a realizar.

(2007): Actualización para las Librerías LAPACK y ScaLAPACK

Este proyecto tiene como principal objetivo hacer que las librerías LAPACK y ScaLAPACK funcionen de forma más eficiente en una gama de arquitecturas mucho más amplias basadas en supercomputadores y multicores con características heterogéneas [27].

(2007): HeteroScaLAPACK

El proyecto HeteroScaLAPACK [28] presenta una metodología que hace posible el diseño de rutinas ScaLAPACK a procesadores paralelos, adaptándolos a un ambiente heterogéneo de computación. En su gran mayoría las rutinas desarrolladas en ScaLAPACK, no extraen el funcionamiento máximo de las redes heterogéneas, pero la metodología proporciona una manera fácil y simple de ejecutar los programas de ScaLAPACK sobre tales redes con buenas mejoras del funcionamiento.

1.3.4. Computación Paralela Heterogénea

Dentro de la Computación Paralela el campo de la Computación Heterogénea de Altas Prestaciones y sus aplicaciones se ha convertido en uno de los más dinámicos en el mundo de la Informática. La implementación de algoritmos eficientes en entornos heterogéneos de computadores constituye un reto de mayor dificultad que la programación paralela en entornos homogéneos. Las redes de computadores se han abaratado y su utilización se ha extendido recientemente, con lo que un objetivo crucial tanto para el mundo de la investigación como en cualquier campo de la industria y de la sociedad, es obtener el máximo rendimiento de la potencia de computadores de distinta naturaleza trabajando juntos en la solución de un determinado problema científico.

La heterogeneidad se caracteriza por la forma de su funcionamiento y por las velocidades de la transferencia de datos entre los procesadores [29]. En general, la utilización del software numérico se hace originalmente para máquinas homogéneas, pero la computación científica de alto rendimiento debe concentrarse también en el desarrollo de librerías numéricas paralelas dirigidas el logro del mejor funcionamiento para plataformas heterogéneas, ya que aplicaciones de esta disciplina son muy numerosas y pueden encontrarse prácticamente en cualquier campo de la Ingeniería y de la Industria. En esta tesis se ha estudiado una solución Divide y Vencerás para un determinado tipo de problema (Cálculo de Valores Propios), adaptándolo a un ambiente heterogéneo de computación.

1.4. Paradigma Divide y Vencerás

El Esquema Divide y Vencerás consiste en un esquema algorítmico para la resolución de problemas, cuya metodología se basa en descomponer un problema en subproblemas más simples del mismo tipo, de forma que las soluciones obtenidas sean independientes y, una vez obtenidas las soluciones parciales, combinarlas para obtener la solución del problema original [30, 31, 32]. El pseudo código del Algoritmo 1 refleja la estructura general de un algoritmo recursivo propuesto.

Algoritmo 1 Esquema General de un Algoritmo Divide y Vencerás.

DivideVencerás (p: problema, n: tamaño): solución
if (n es suficientemente pequeño) **then**
  devolver solución de p por método directo
else
  dividir p en subproblemas más pequeños p_1, p_2, ..., p_k
  for (i=1 to k) **do**
    s_i = **DivideVencerás** (p_i, n_i)
  end for
  devolver combinar (s_1, s_2, ..., s_k)
end if
end DivideVencerás

Para que pueda aplicarse la técnica Divide y Vencerás se debe cumplir:

- ◇ El problema original debe poder dividirse fácilmente en un conjunto de subproblemas, del mismo tipo que el problema original pero con una resolución menos costosa.
- ◇ La solución de un subproblema debe obtenerse de forma independiente de la de los demás.
- ◇ Se necesita de un método directo para la resolución de problemas de tamaños pequeños.
- ◇ Es necesario tener un método que combine las soluciones parciales hasta obtener la solución del problema original.

En el esquema tenemos que, dada una función para computar una determinada solución sobre una entrada de tamaño n, la estrategia Divide y Vencerás divide la entrada en p_1, p_2, ..., p_k subproblemas distintos. Una vez resueltos los subproblemas, se elige un método que permita combinar las soluciones en una solución del problema total.

En la primera fase se realiza la división en subproblemas que se van generando mediante las llamadas de la función *DivideVencerás*. Si el subproblema se considera grande, entonces volvemos a aplicar la técnica. Superada la fase de división

se pasa a la fase de combinación. Al llegar a este punto se combinan las soluciones parciales hasta obtener la solución del problema original. Otra consideración importante a la hora de diseñar algoritmos Divide y Vencerás es el reparto de la carga entre los subproblemas, puesto que suele ser importante que la división en subproblemas se haga de la forma más equilibrada posible. También es interesante tener presente la dificultad y el esfuerzo requerido en cada una de estas fases, lo que va a depender del planteamiento del algoritmo concreto. Por ejemplo, los métodos de ordenación por mezcla *Mergesort* y el método de ordenación rápida *Quicksort* son dos representantes claros de esta técnica pues ambos están diseñados siguiendo el esquema presentado: dividir, resolver y combinar. El método Divide y Vencerás se destaca por ofrecer prestaciones competitivas proporcionando resultados más rápidos y precisos, caracterizándose también por el alto grado de paralelismo intrínseco, pudiendo ser trasladado fácilmente a multicomputadores. Por otro lado ha sido aplicado por diversos autores a la resolución de problemas matriciales numéricos [33], siendo uno de los focos de interés de estudio de esta tesis.

1.5. Visión General del Sistema de Autooptimización Propuesto

La propuesta consiste en aplicar las técnicas de autooptimización teniendo como base la referencia [19], extendiéndola a rutinas que siguen un esquema Divide y Vencerás. La idea consiste en construir un modelo matemático del tiempo de ejecución de una rutina, de manera que este modelo sea una herramienta útil para decidir los valores de unos parámetros con los que se obtenga una ejecución eficiente, y con ello minimizar su tiempo de ejecución. Para esto, el modelo debe reflejar las características de cómputo y de comunicaciones de los algoritmos y del sistema sobre el que se ejecutará. Tendremos, de este modo, el modelo matemático del tiempo de ejecución de la siguiente forma:

$$t(s) = f(s, AP, SP), \tag{1.1}$$

donde s representa el tamaño de la entrada, SP (Parámetros del Sistema) representa los parámetros que reflejan las características del sistema, y AP (Parámetros Algorítmicos) los que intervienen en el algoritmo y cuyos valores deben ser seleccionados adecuadamente para obtener un tiempo de ejecución reducido. Parámetros SP típicos son: el coste de una operación aritmética y los tiempos de inicio de las comunicaciones y de envío de un dato en operaciones de comunicación. Con la utilización de estos parámetros se reflejan las características del sistema de cómputo y de comunicaciones. Algunos parámetros AP típicos son: el número de procesadores a utilizar, y el número de procesos a poner en marcha y su mapeo en el sistema físico, parámetros que identifiquen la topología lógica de los procesos, el tamaño de los bloques de comunicación o de particionado de los datos entre los procesos, o el tamaño de los bloques de computación en algoritmos que trabajan por bloques. El valor de todos estos parámetros debe ser seleccionado para obtener tiempos de ejecución reducidos, y no podemos pretender que usuarios no expertos tengan suficiente conocimiento del problema como para realizar una selección satisfactoria, por lo que creemos que es conveniente utilizar técnicas de autooptimización. De cara a obtener un modelo más realista, se puede tener en cuenta que los valores SP están influenciados por los de los AP [10]. Los parámetros algorítmicos podrían ser, por ejemplo, en el método Divide y Vencerás: el nivel de recursión, el método directo y el número de procesadores; y se obtienen sustituyendo en el modelo del tiempo los valores de los parámetros del sistema (los tiempos de inicio de comunicación y de envío de un dato, y los costes de computación), obteniendo los valores óptimos de los parámetros algorítmicos. Los SP se pueden expresar como una función del tamaño de la entrada y de los parámetros AP ($SP = g(s, AP)$), con lo que el tiempo queda de la siguiente forma:

$$t(s) = f(s, AP, g(s, AP)). \tag{1.2}$$

Los valores SP se obtendrán en el momento de instalar la rutina en un nuevo sistema. Para esto, el diseñador de la rutina debe haber desarrollado el modelo del tiempo de ejecución, haber identificado los parámetros SP que intervienen en el modelo, y haber diseñado una estrategia de instalación, que incluye para

cada *SP* los experimentos a realizar para su estimación, y los parámetros *AP* y sus valores con los que hay que experimentar. Los valores obtenidos para los *SP* se incluyen junto con el modelo del tiempo de ejecución en la rutina que se está optimizando, que se instala de esta forma con información del sistema para el que se está optimizando. En tiempo de ejecución, para un tamaño de la entrada concreto, la rutina obtiene de manera automática valores de los *AP* con los que se obtiene una ejecución óptima según el modelo de ejecución de la rutina y los valores de los *SP* obtenidos en la instalación. El tiempo de obtención de estos valores debe ser reducido debido a que incrementa el tiempo de ejecución de la rutina. Para los parámetros del sistema se estiman los costes de las operaciones aritméticas básicas que aparecen en la rutina. Se obtienen los parámetros por medio de una serie de ejecuciones, y consiguiendo mayor precisión a través de la realización de la mayor cantidad de experimentaciones posibles.

1.6. Diseño del Sistema Autooptimizado

En este apartado presentamos el diseño de la arquitectura de la rutina autooptimizada para la implementación de la metodología de autooptimización basada en la fundamentación teórica anterior. La idea general consistiría en crear una rutina de resolución del problema especifico, que antes de ser ejecutada, recibe información de parámetros de autooptimización obtenidos en la plataforma durante la fase de instalación, haciendo uso de esta información en la etapa posterior de ejecución para conseguir mayor eficiencia para el algoritmo. En el sistema desarrollado se distinguen las siguientes etapas: instalación y ejecución.

Se hace necesaria una etapa de instalación, durante la que se lleva a cabo un estudio de las características de la plataforma sobre la que se está instalando la rutina. Las características se reflejarán por medio de los parámetros del sistema. Los parámetros del sistema a determinar pueden ser el coste de las operaciones aritméticas para cada estrategia desarrollada. Una vez obtenidos los valores de estos parámetros, se incluyen junto con el modelo del tiempo de ejecución en la rutina que se está instalando, y se instala ésta, posiblemente compilándola

con el código que se le ha añadido. En la etapa de ejecución tenemos que, dada una entrada, se decide el valor de los parámetros algorítmicos para resolver el problema con un tiempo próximo al óptimo. Para esto, la rutina sustituye en el modelo teórico del tiempo posibles valores como: el tamaño del problema, el tamaño de bloque o el comportamiento de la librería y se ejecuta con el valor con que se obtiene el menor tiempo teórico de ejecución. Se puede detallar el sistema de autooptimización en los siguientes pasos:

Inicialización de la Interface de Configuración

Se ha desarrollado una interface de configuración para ayudar al usuario a interaccionar con los módulos del sistema, dando la opción de experimentar libremente con las rutinas optimizadas. La interface fue implementada como un servicio web de fácil diseño, teniendo como principales características: la simplicidad y la interactividad con el usuario a partir de un módulo gráfico. Se explica con detalles las funcionalidades de la herramienta gráfica desarrollada en la sección 1.8.

Estimación de los Parámetros del Sistema (SP)

Se estiman los costes de las operaciones aritméticas básicas del algoritmo para cada uno de los métodos propuestos, a través de experimentaciones con los tiempos de ejecución de las estrategias para tamaños del problema y de bloques pequeños, aproximando el comportamiento de la líbreria a una expresión analítica.

Almacenamiento de los datos de los Parámetros del Sistema (SP) calculados en la estructura SP.h

La información de los parámetros *SP*, se almacena en una estructura llamada SP.h, la cual determina las caraterísticas del sistema en que la rutina fue instalada. Esta estructura servirá para sustituir los valores de los parámetros calculados en el modelo teórico en la fase de ejecución.

Sustitución de los Parámetros del Sistema (SP) en el modelo teórico prediciendo los parámetros algorítmicos (AP) óptimos

Se sustituyen los valores de los parámetros *SP* en el modelo teórico implementado para predecir los parámetros algorítmicos.

1.7. Herramientas Hardware y Software de Entornos Paralelos y Secuenciales Utilizados

En está sección se describe el entorno y las herramientas de trabajo utilizadas para esta tesis. El ambiente de trabajo está formado básicamente por elementos de software y hardware, y se ha trabajado con problemas de alto coste (cálculo de los valores propios de una matriz tridiagonal) y de menor coste (métodos de ordenación), pero se pretende siempre desarrollar algoritmos que hagan un uso eficiente de los recursos computacionales, de manera que se obtenga un tiempo de ejecución reducido. Entornos de programación secuencial pueden ofrecer estas garantías siempre que se cuente con los recursos suficientes para acceder a entornos con alta productividad, y que incorporen los últimos avances tecnológicos. Sin embargo, siempre se encontrarán limitaciones físicas que impiden obtener el mayor beneficio de los recursos disponibles. Una de las soluciones sería utilizar un entorno de programación alternativo que integra computadores de bajo coste unidos por redes de interconexión que utilizan multiprocesadores. Con este tipo de entornos se logran resoluciones sin el problema de las limitaciones físicas, ni de los altos costos de un supercomputador al tener la posibilidad de aumentar el número de procesadores que se integran. En este trabajo se han utilizado esencialmente multiprocesadores de memoria distribuida, cuyos componentes son estaciones de trabajo conectadas mediante una red de interconexión de paso de mensajes. El uso de este tipo de sistema ha dado la posibilidad de diseñar algoritmos donde el control de la distribución y manipulación de los datos es manejada por el programador, utilizando mecanismos de paso de mensajes. Una breve descripción de los clusters y de los softwares y librerías utilizadas se da a continuación.

1.7.1. Descripción de los Clusters Utilizados

Los entornos de experimentación caracterizados a continuación, nos servirán para experimentar y obtener resultados con la mayoría de las soluciones desarrolladas. La elección de los entornos se ha dado debido las necesidades de cómputo, siendo utilizado en algunos casos solo parte de las plataformas citadas.

Cluster KEFREN

La plataforma **KEFREN** (kefren.dsic.upv.es) está formada por 20 nodos biprocesadores Pentium Xeon 2 Ghz y interconexión SCI con una topología de Toro 2D en una matriz de 4×5 nodos. Esta plataforma pertenece al Grupo de Redes y Computación de Altas Prestaciones [35] de la Universidad Politécnica de Valencia, España

Cluster ROSEBUD

La plataforma ROSEBUD (rosebud.dsic.upv.es) es una red heterogénea que está formada por 7 nodos. Los 2 biprocesadores Xeon, poseen reloj de 2.2 GHz, memoria principal de 4 MB y memoria cache de 1024 KB. Los 2 tetraprocesadores Itanium II dualcore poseen reloj de 1.4 GHz, memoria principal de 8 MB y memoria cache de 1024 KB, y los 3 pentium IV poseen reloj de 1.5, 1.7 y 3 Ghz respectivamente, memoria principal de 1024 MB y memoria cache de 512 KB. Dichos procesadores están conectados por medio de 2 redes de comunicaciones: 1 Ethernet de 1Gbs y, otra FastEthernet de 100 Mbs. Esta plataforma fue adquirida en el marco del proyecto TIC2003-08238-C02.

Cluster ALDEBARAN

La plataforma ALDEBARAN (aldebaran.upv.es) está formada por 48 nodos procesadores Itanium II con memoria distribuida. Esta plataforma pertenece al Centro de Cálculo [36] de la Universidad Politécnica de Valencia, España.

Cluster PGCLUSTER-CSULTRA

La plataforma PGCLUSTER-CSULTRA (csserver.ucd.ie) es una red heterogénea de maquinas Linux/SunOs con 15 procesadores. Los sistemas PGCLUSTER poseen 2 procesadores, con reloj de 1977 MHz, memoria principal de 1024 MB y memoria cache de 512 KB. Por su parte, los sistemas CSULTRA son monoprocesador, con reloj de 440 MHz, memoria principal del 512 MB y memoria cache de 2048 KB. Dichos procesadores están conectados por medio de una red de comunicación Ethernet de 100 Mbits, con un switch que permite comunicar procesadores entre sí. Esta plataforma pertenece al Grupo de Computación Heterogénea [37] de la Universidad de Dublin, Irlanda.

1.7.2. Descripción de los Softwares y Librerías Utilizadas

Para el tipo de hardware utilizado existe software adecuado que explota las características de la arquitectura y permite obtener el máximo rendimiento; por eso se ha empleado, para los programas codificados en lenguaje de programación Fortran, C y C++, las siguientes softwares y librerías:

BLAS (Basic Linear Álgebra Subprograms)

Colección de rutinas para realizar operaciones básicas con matrices y vectores [16].

LAPACK (Linear Álgebra Package)

Librería que contiene rutinas desarrolladas en lenguaje Fortran para resolver problemas comunes del álgebra lineal (sistemas de ecuaciones lineales, problemas de mínimos cuadrados lineales, problemas de valores propios y problemas de valores singulares) [38].

MPI (Message Passing Interface)

También se han utilizado las líbrerias de paso de mensajes: MPI [39, 40, 41]. Es una biblioteca de comunicaciones que implementa el mecanismo de paso de mensajes entre nodos de cómputo que se comunican mediante una red de interconexión. Este estándar define el interfaz con el usuario y la funcionalidad de un amplio rango de capacidades de paso de mensajes. El principal objetivo de MPI es la portabilidad del código de paso de mensajes entre plataformas de muy diferente naturaleza (multiprocesadores de memoria distribuida, multiprocesadores de memoria compartida, redes de computadores, ...), manteniendo un alto grado de eficiencia en todas ellas. Se pueden encontrar multitud de implementaciones de MPI, siendo la versión MPICH [42] las más difundida. MPI realiza las diferentes operaciones de comunicación, que en el programa se especifican de manera lógica. El entorno incluye, además de las operaciones de comunicación punto a punto, un conjunto de operaciones colectivas: barreras de sincronización, difusión (*broadcast*), distribución (*scatter*), recolección (*gather*), así como operaciones de multidifusión (*multicast*). Algunas de las principales características MPI son:

• **Control de Procesos:** MPI tiene la capacidad de iniciar y finalizar procesos en cualquier momento de la ejecución de un programa (versión MPI-2).

• **Topología:** En MPI se pueden agrupar tareas en una topología lógica de interconexión específica, realizándose la comunicación entre estas tareas.

• **Tolerancia a fallos:** Es un esquema básico de notificación de fallos a las tareas. No existe ningún tipo de mecanismo de este tipo, dejándose la decisión de incluirlos a los implementadores de MPI para cada plataforma.

• **Entrada y Salida en Aplicaciones Paralelas:** MPI-2 permite de manera sencilla realizar una distribución de datos entre diferentes procesadores. Permite, por ejemplo, que un número de procesadores escriba un fichero que luego sea procesado por un número diferente de ellos, realizando automáticamente la distribución de los datos.

BLACS (Basic Linear Álgebra Communication Subprograms)

Es una líbreria que ofrece rutinas de comunicación de paso de mensajes que aparecen con frecuencia en cómputos paralelos de álgebra lineal [43].

PBLAS (Parallel Basic Linear Álgebra Subprograms)

Es una versión de BLAS para computadoras de memoria distribuida [44]. Tanto PBLAS como BLACS proporcionan rutinas necesarias para el funcionamiento de ScaLAPACK.

ScaLAPACK (Scalable LAPACK)

Librería diseñada para resolver un conjunto de rutinas LAPACK en multiprocesadores de memoria distribuida con mecanismos de comunicación de paso de mensajes [45]. Asume que las matrices que maneja se descomponen en bloques cíclicos bidimensionales de tamaño variable sobre una malla lógica de procesadores.

1.8. Interface Web de Configuración

Con la idea de simplificar el uso de los sistemas autooptimizados, se ha desarrollado una herramienta gráfica web para que el usuario interactúe y establezca un contacto más fácil e intuitivo. Esta interface utiliza un conjunto de imágenes y

objetos gráficos que representan la información y acciones disponibles, proporcionando al usuario un conjunto de posibilidades que podrá seguir durante todo el tiempo que se relacione con el programa, así como las respuestas que puede ofrecer el sistema. Las características básicas de la interface web podrían sintetizarse en:

- ◇ Facilidad de comprensión, aprendizaje y uso.
- ◇ Sistema gráfico web interactivo.
- ◇ Las interacciones proporcionarán operaciones rápidas y reversibles, con efectos inmediatos.
- ◇ Diseño mediante el establecimiento de menús, barras de acciones e iconos de fácil acceso.
- ◇ Las interacciones se basarán en acciones sobre elementos de código visual (botones, imágenes, mensajes de texto, barras de desplazamiento y navegación...).

El espacio de navegación web se organiza de forma que ayude al usuario a recorrer los contenidos ofreciéndole operaciones que guíen o faciliten los recorridos y la recuperación de información y que le ofrezcan la posibilidad de realizar una serie de acciones estableciendo mecanismos de vuelta atrás, y la vista previa de enlaces y otros procesos automatizados a modo de servicios interactivos.

1.8.1. Ejemplos de Interfaces Web Desarrolladas

Interface Web de Configuración del Sistema de Ordenación Autooptimizado

El desarrollo de esta Interface Web de Configuración fue utilizada como parte del módulo del Sistema de Ordenación Autooptimizado se describe en el capítulo

2 de esta tesis. Esta interface tiene como funcionalidad, ayudar la experimentación de la etapa de instalación, dando al usuario la posibilidad de experimentar libremente. Se pueden destacar las siguientes partes funcionales de su diseño:

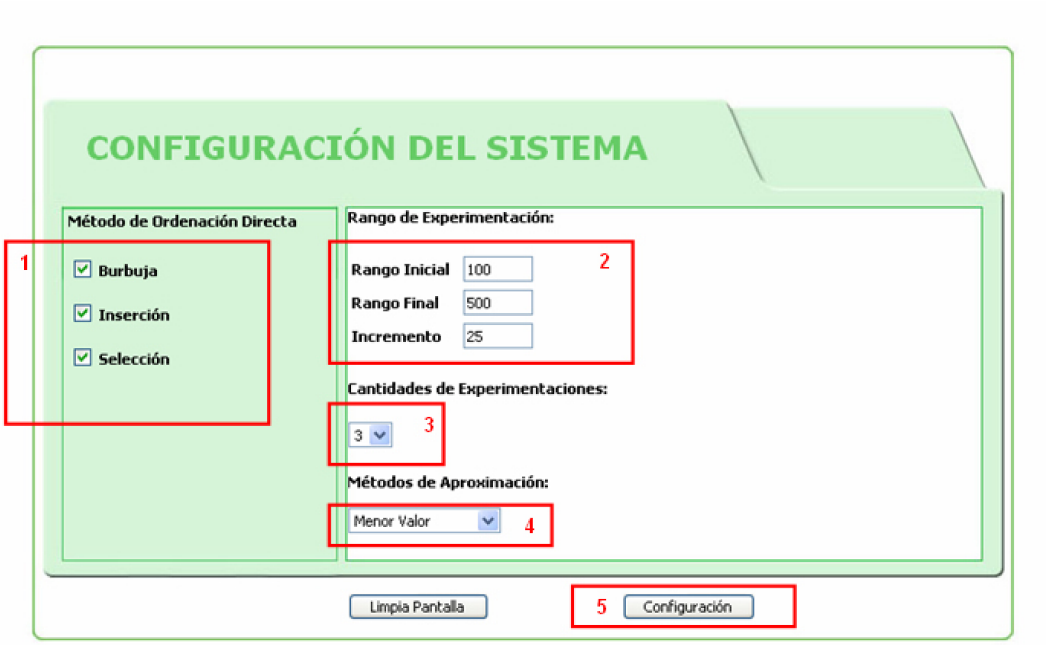

Figura 1.2: Interface Web de Configuración del Sistema de Ordenación.

1. Definición de los métodos de ordenación directos usados para el estudio (métodos directos de ordenación usados: burbuja, inserción y selección).

2. Definición de los rangos de la experimentación (rango máximo, rango mínimo y el incremento de intervalo).

3. Cantidades de experimentaciones, que pueden ser 1, 3 ó 5.

4. Los métodos de aproximación usados para la experimentación, los cuales pueden ser: el mayor o menor valor o el promedio de los valores.

5. Envío de los datos de la configuración del sistema para el inicio de los tests en la instalación.

Interface Web de Configuración de las Rutinas de Resolución de Sistemas Tridiagonales por Bloques

La interfaz desarrollada puede ser utilizada en cualquier tipo de problema relacionado en las técnicas de autooptimización. Como ejemplo, mostramos también la aplicación de una Interface Web de Configuración desarrollada en el marco de un proyecto interdisciplinar entre el grupo de Redes y Computación de Altas Prestaciones [35] y el grupo de Aplicaciones de las Microondas [46] en el campo del desarrollo de nuevos algoritmos matriciales para su aplicación en problemas electromagnéticos. El problema físico se basa en la mejora de las soluciones presentadas en la resolución de problemas matriciales, para el desarrollo de rutinas autooptimizables en la resolución de Ecuaciones Tridiagonales por Bloques [47], originados a partir de un problema numérico estudiado en la área de Telecomunicaciones.

La Interface Web de Configuración desarrollada tiene como principales funcionalidades:

- ◇ Definición de los métodos de resolución usados para el estudio.
- ◇ Definición de los rangos de la experimentación (por ejemplo: rangos máximo y mínimo, dimensión y tamaño de bloque de la matriz).
- ◇ Elección de las cantidades de experimentaciones de la herramienta desarrollada.
- ◇ Elección de los modelos de ajuste usados para la experimentación.
- ◇ Envío de los datos de la configuración del sistema para el inicio de los tests en la instalación.

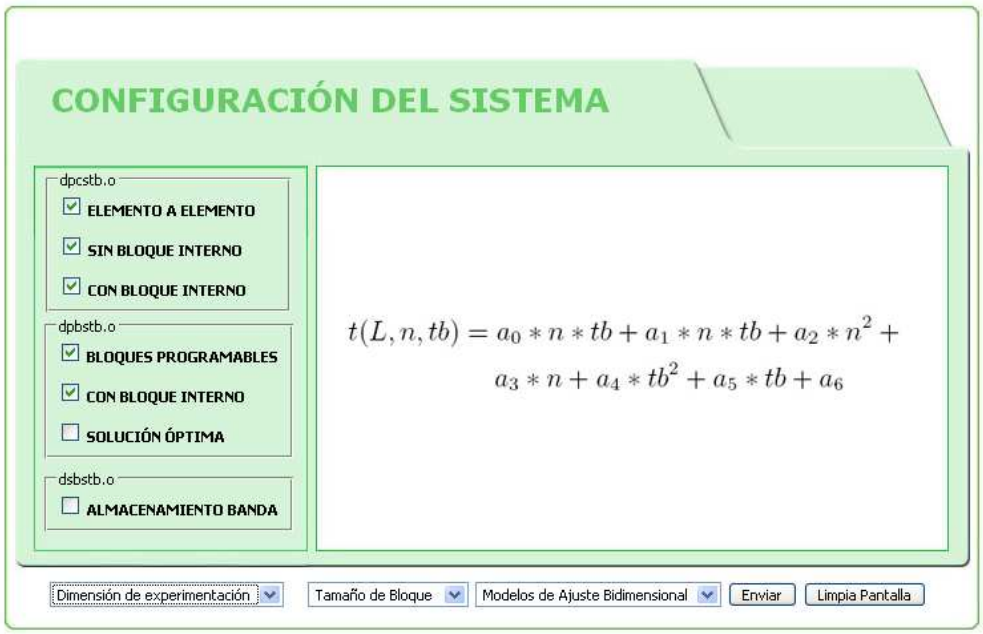

Figura 1.3: Interface Web de los Sistemas Tridiagonales por Bloques.

1.9. Evaluación de las Prestaciones Algorítmicas

Para llevar a cabo el análisis de las experimentaciones, se utilizan una serie de parámetros, teniendo como objetivo fundamental la paralelización de los algoritmos, en nuestro caso esquemas basados en el paradigma Divide y Vencerás, consiguiendo que nuestro algoritmo se ejecute lo más rápido posible, explotando al máximo todas las capacidades de máquinas paralelas donde se ejecuten. A continuación se definen algunos conceptos que permitan medir y evaluar los algoritmos implementados, teniendo como base teórica la referencia bibliográfica [48].

1.9.1. Tiempo Secuencial y Paralelo

El tiempo secuencial (T_s) es el tiempo necesario para resolver el problema secuencialmente, utilizando el algoritmo más rápido conocido. Al tiempo de ejecución del algoritmo paralelo con p procesadores le llamamos (T_p), y dependerá del tiempo aritmético (T_a) y, en el caso de paso de mensajes, del tiempo de las comunicaciones relativo a los mensajes (T_c), de la siguiente manera:

$$T_p = T_a + T_c. \tag{1.3}$$

En los estudios experimentales para medir el tiempo que tardan en ejecutarse los algoritmos, se utilizó un temporizador para medir los intervalos de tiempo empleados, tanto en las ejecuciones secuenciales como en las paralelas.

1.9.2. Parámetros de eficiencia

Para evaluar un sistema el primer paso es saber la ganancia en prestaciones obtenidas por paralelizar una determinada aplicación en relación con la misma en el caso secuencial. Una de las medidas utilizadas para medir el beneficio relativo de las prestaciones es el *speedup*.

Speedup y Eficiencia

El *speedup* (S) se define como la razón entre el tiempo de ejecución del mejor algoritmo secuencial para resolver determinado problema, por el tiempo de ejecución para resolver el mismo problema utilizando p procesadores en paralelo [48]:

$$S = \frac{T_s}{T_p}, \tag{1.4}$$

la eficiencia (E) es una medida de la fracción del tiempo en el que un procesador

está siendo utilizado. Se define como la razón entre el *speedup* y el número de procesadores [48]:

$$E = \frac{S}{p}. \tag{1.5}$$

En un sistema ideal el *speedup* debería de ser igual al valor de p y la eficiencia igual a 1. En la práctica el *speedup* suele ser menor que p y la eficiencia está entre 0 y 1, dependiendo del grado de paralelismo alcanzado.

1.10. Objetivos de la Tesis

En este trabajo se tiene por objetivo general el estudio de esquemas paralelos Divide y Vencerás, con estudio del modelo teórico del tiempo e identificación de parámetros que intervienen en el modelo, desarrollando técnicas de instalación y de toma de decisiones en tiempo de ejecución. Para alcanzar el objetivo general de esta tesis, se pretenden alcanzar los siguientes objetivos específicos:

- ◇ Diseñar una arquitectura que sirva para el desarrollo de aplicaciones y sistemas autooptimizables.
- ◇ Proporcionar un modelo de programación sencillo y potente.
- ◇ Implementar rutinas de instalación para una librería de rutinas de cálculo numérico con autooptimización.
- ◇ Estudio de modelos de tiempo de ejecución y de técnicas de instalación de librerías para sistemas heterogéneos, lo que permitiría aplicar a estos sistemas las ideas desarrolladas para sistemas homogéneos.
- ◇ En cuanto a aplicaciones, como se ha indicado en los puntos anteriores, se pretende aplicar las técnicas anteriores a problemas de alto nivel, con la finalidad de validar las técnicas desarrolladas.

1.11. Organización de la Tesis

La estructura básica de los capítulos está compuesta por la descripción del problema, el diseño de los algoritmos secuenciales y paralelos propuestos, los resultados obtenidos en los experimentos y un conjunto de conclusiones. Habiendo definido el problema a tratar y especificando las condiciones de resolución en este capítulo, el trabajo se organiza como sigue:

- ◇ **Capítulo 2**. Se propone un estudio de modelos de autooptimización en esquemas algoritmos de ordenación. Se eligieron para la implementación el algoritmo de ordenación por mezcla *Mergesort* y el algoritmo de ordenación rápida *Quicksort* ya que se basan en un esquema Divide y Vencerá. Se intenta mejorar las prestaciones de los algoritmos a través de la predicción de los mejores párametros en un modelo matemático.
- ◇ **Capítulo 3**. Se estudia la adecuación a sistemas heterogéneos de las técnicas de autooptimización de problemas típicos que se resuelven por Divide y Vencerás, en particular el problema típico del cálculo de los valores propios de matrices tridiagonales.
- ◇ **Capítulo 4**. Se resumen las conclusiones obtenidas y se proponen líneas futuras de investigación.

Capítulo 2

Propuesta de Autooptimización en Librerías de Ordenación

En este capítulo se estudia la aplicación de técnicas de parametrización a algoritmos de ámbito académico basados en el paradigma Divide y Vencerás.

2.1. Introducción

En este estudio se intenta mejorar las prestaciones de los algoritmos a través de la aplicación de las técnicas de autooptimización prediciendo los mejores párametros en un modelo matemático. Los algoritmos propuestos resuelven el problema dividiéndolo en varios subproblemas del mismo tipo para que se resuelvan más fácilmente, combinando la solución de los resultados de los subproblemas para obtener la solución del problema original. Se propone un estudio de modelos de autooptimización en esquemas algoritmos de ordenación que se basan en el paradigma Divide y Vencerás.

2.2. Algoritmo de Ordenación por Mezcla

Elegimos para la implementación el algoritmo de ordenación por mezcla *Mergesort* [30, 49, 50], para estudiar la metodología propuesta, ya que se basa en un esquema Divide y Vencerás. El algoritmo lo que hace es, a partir de una lista de elementos desordenados, repartir sus elementos en trozos, ordenando de forma recursiva los elementos. Una vez que se dividió la lista hasta llegar a un tamaño base, el algoritmo mezcla los elementos, poniendo juntos los elementos ordenados. Consideramos la ordenación de n datos almacenados en un *array* con índices entre 1 y n. Los datos se ordenan dividiendo el problema en 2 subproblemas de igual tamaño hasta llegar al caso base que será tener un número de elementos menor o igual que un valor del caso base, que se ordenan por medio de un método directo. Una vez que tenemos dos *subarrays* ordenados los mezclamos para obtener ordenado el *array* inicial, con lo que el procedimiento de combinación consiste en mezclar dos *arrays* ordenados. El Algoritmo 2 representa el esquema implementado.

Algoritmo 2 Algoritmo de Ordenación por Mezcla.

```
mergesort (a: array[1..n]; p, q: índices, casoBase: var tipo)
if ( q-p <casoBase) then
  MetodoDirecto(a, p, q)
else
  m=(p+q)/2
  mergesort(a, p, m, casoBase)
  mergesort(a, m+1, q, casoBase)
  merge(a, p, m, q)
end if
end mergesort

merge (a: array[1..n]: tipo; p, m, q: índices)
var b: array[p..q]
var h, i, j, k: índices
h = p
i = p
j = m + 1
while ((h <= m) and (j <= q)) do
  if (a[h] <= a[j]) then
    b[i] = a[h]
    h = h + 1
  else
    b[i] = a[j]
    j = j + 1
  end if
  i = i + 1
end while
if (h >m) then
  for k = j..q do
    b[i] = a[k]
    i = i + 1
  end for
else
  for k = h..m do
    b[i] = a[k]
    i = i + 1
  end for
end if
for k = p..q do
  a[k] = b[k]
end for
end merge
```

2.2.1. Estudio del tiempo de Ejecución

Una vez que se han determinado los parámetros del sistema, esto es, los costes de las operaciones aritméticas básicas del algoritmo para cada uno de los métodos directos de ordenación propuestos (k_{MD}) y el coste del algoritmo de mezcla, se desarrolla un modelo matemático del *Mergesort* para predecir el tiempo de ejecución en función de los parámetros calculados. El modelo del tiempo de ejecución se basa en la siguiente ecuación de recurrencia:

$$t(n, casoBase, MD) = \begin{cases} k_{MD}\ n^2 & n \leq casoBase \\ 2t\left(\frac{n}{2}\right) + (b_{merge}\ n + c_{merge}) & n > casoBase \end{cases} \tag{2.1}$$

donde se obtiene el tiempo en función de los valores del tamaño del problema y del caso base, de tal forma que el término $k_{MD}\ n^2$ corresponde a solución por un método directo, y $b_{merge}\ n + c_{merge}$ a la mezcla. La ecuación de recurrencia tiene como principal objetivo obtener un término general para el tiempo de ejecución del algoritmo, en función del tamaño del problema (n), del nivel de recursión utilizado, representado por el *casoBase*, y del método directo (MD) elegido. A partir de la fórmula conseguiremos obtener el mínimo valor del tiempo de ejecución t dependiendo de los valores de los parámetros del sistema ($k_{MD},\ b_{merge},\ c_{merge}$) calculados. Si expandimos la recursión hasta llegar al caso base tenemos:

$$\begin{aligned} t(n, casoBase, MD) &= 2\left(2t\left(\frac{n}{2^2}\right) + b_{merge}\frac{n}{2} + c_{merge}\right) + b_{merge}\ n + c_{merge} \\ &= 2^2\ t\left(\frac{n}{2^2}\right) + 2\ b_{merge}\ n + c_{merge}(1+2) \\ &= \cdots \\ &= 2^k\ t\left(\frac{n}{2^k}\right) + k\ b_{merge}\ n + c_{merge}(1+2+...+2^{k-1}). \end{aligned}$$

y como sabemos que $\frac{n}{2^k} = casoBase$, y $t(casoBase) = k_{MD}\ casoBase^2$, si sustituimos los valores de los parámetros del sistema y desarrollamos la ecuación de recurrencia llegaremos a una ecuación que representa la aproximación del comportamiento del tiempo de ejecución, $t(n, casoBase, MD)$, en función del tamaño del problema y del caso base óptimo, en la siguiente forma:

$$t(n, casoBase, MD) = k_{MD}\ n\ casoBase + b_{merge}\ n\ log\left(\frac{n}{casoBase}\right) + c_{merge}\left(\frac{n}{casoBase} - 1\right). \quad (2.2)$$

Valor Óptimo del Caso Base

Para calcular el valor óptimo del caso base dado un tamaño de problema fijo, n, se deriva la función del tiempo respecto de la variable *casoBase* y se calcula el valor que anula la derivada:

$$\frac{\partial t}{\partial casoBase} = 0,$$

$$\frac{\partial\left(k_{MD}\ n\ casoBase + b_{merge}\ n\ log\left(\frac{n}{casoBase}\right) + c_{merge}\left(\frac{n}{casoBase} - 1\right)\right)}{\partial casoBase} = 0,$$

$$k_{MD}\ n - \left(\frac{b_{merge}\ n}{ln(2)\ casoBase}\right) - \frac{c_{merge}\ n}{casoBase^2} = 0,$$

$$k_{MD}\ casoBase^2 - \frac{b_{merge}}{ln(2)}\ casoBase - c_{merge} = 0,$$

$$casoBase \;=\; \frac{\frac{b_{merge}}{ln(2)} + \sqrt{\left(\frac{b_{merge}}{ln(2)}\right)^2 + 4\ k_{MD}\ c_{merge}}}{2\ k_{MD}}. \qquad (2.3)$$

Sustituyendo los valores de los parámetros del sistema (k_{MD}, b_{merge}, c_{merge}) calculados, podremos sacar el valor del caso base para el que se obtiene el menor tiempo de ejecución, para determinado tamaño del problema de forma directa.

2.3. Algoritmo de Ordenación Rápida

Elegimos también para estudiar la metodología propuesta la implementación del algoritmo de ordenación rápida *Quicksort* [30, 49, 50], ya que como el anterior también se basa en un esquema Divide y Vencerás. En este caso el problema se divide en dos subproblemas, pero estos no tienen por qué ser de igual tamaño. Dado el array de índices p a q, se toma un elemento pivote y se obtienen los elementos menores o iguales que él almacenándolos en a, en las posiciones de la p a la m, y los mayores almacenándolos en las posiciones de la $m+1$ a la q. En el Algoritmo 3 se puede ver el Esquema Divide y Vencerás para la ordenación rápida:

Algoritmo 3 Algoritmo de Ordenación Rápida.

```
quicksort (a: array[1..n]: tipo; p, q: índices, casoBase: var tipo)
var j: entero
if ( q-p < casoBase) then
  ordenaciónDirecta(a, p, q)
else
  [a, j]=particion(a, p, q)
  quicksort(a, p, j-1, casoBase)
  quicksort(a, j+1, q, casoBase)
end if
end quicksort
```

```
particion (a: array[1..n]: tipo; izq, der: índices): entero
var temp: tipo
var i, d: índices

i = izq;
d = der;

while (i <d)  do
   while (a[d] >a[izq])  do
      d=d-1
   end while

   while ((i <d) and (a[i] <= a[izq]))  do
      i=i+1
   end while

   if (i <d)  then
      temp = a[i]
      a[i] = a[d];
      a[d] = temp;
   end if
end while

temp = a[d]
a[d] = a[izq];
a[izq] = temp;

return d

end particion
```

2.3.1. Estudio del tiempo de Ejecución

Se desarrolla un modelo matemático del *Quicksort* para predecir el tiempo de ejecución en función de los parámetros del sistema calculados, estimando los costes de las operaciones aritméticas básicas del algoritmo para cada uno de los métodos directos de ordenación propuestos (k_{MD}) y el coste del algoritmo de particionar, igual que anteriormente. El modelo del tiempo de ejecución se expresó a través de ecuaciones de recurrencias utilizando una aproximación considerado un caso promedio.

Caso Promedio

En el algoritmo *Quicksort*, el problema no siempre se divide por la mitad: el punto de división depende del pivote elegido, por lo que el procedimiento *Particion* no siempre devolverá un elemento pivote que deja la misma cantidad de elementos a la izquierda y a la derecha del mismo. Como simplificación tomamos un valor i que está entre 1 y n, correspondientes a las n posiciones en que puede quedar el elemento pivote.

$$t(n, casobase, MD) = \begin{cases} k_{MD}\ n^2 & n \leq casoBase \\ t(n-i) + t(i-1) + (b_{part}\ n + c_{part}) & n > casoBase \end{cases} \tag{2.4}$$

En este caso también obtenemos el tiempo de ejecución en función del tamaño del problema y del caso base, de tal forma que el término $k_{MD}\ n^2$ corresponde a una solución por un método directo y $b_{part}\ n + c_{part}$ a la partición. A partir de la fórmula conseguiremos obtener el valor del tiempo de ejecución t dependiendo de los valores de los parámetros del sistema (k_{MD}, b_{part}, c_{part}) calculados.

Si consideramos que el elemento pivote puede ir a parar con la misma probabilidad a cada una de las de las n posiciones del *array* y aplicamos algunas modificaciones álgebraicas nos queda que:

$$
\begin{aligned}
t(n) &= b_{part}\ n + c_{part} + \frac{1}{n}\sum_{i=1}^{n} t(n-1) + \frac{1}{n}\sum_{i=1}^{n} t(i-1) \\
t(n) &= b_{part}\ n + c_{part} + \frac{2}{n}\ (t(n-1) + t(n-2) + \cdots + t(0)) \\
t(n) &= b_{part}\ n + c_{part} + \sum_{i=1}^{n} \frac{2}{n}\ t(i-1) \\
n\ t(n) &= b_{part}\ n^2 + c_{part}\ n + 2\sum_{i=1}^{n} t(i-1) \qquad (2.5)
\end{aligned}
$$

considerando en 2.5 el caso que n se sustituye por $n-1$:

$$
(n-1)\ t(n-1) = b_{part}(n-1)^2 + c_{part}\ (n-1) + 2\sum_{i=1}^{n-1} t(i-1) \qquad (2.6)
$$

y restando a 2.5 la Ecuación 2.6, agrupando y dividiendo $n(n+1)$:

$$
\frac{t(n)}{n+1} - \frac{t(n-1)}{n} = b_{part}\frac{n^2-(n-1)^2}{n(n+1)} + c_{part}\frac{1}{n(n+1)} + 2\frac{t(n-1)}{n(n+1)}.
$$

Si consideramos que los términos en $\frac{1}{n(n+1)}$ son muchos menores que los términos en $\frac{1}{n}$ podemos aproximar:

$$
\begin{aligned}
\frac{t(n)}{n+1} &\cong \frac{2\ b_{part}}{n+1} + \frac{t(n-1)}{n} \\
\frac{t(n)}{n+1} &= \left(\frac{2}{n+1} + \frac{2}{n}\right) b_{part} + \frac{t(n-2)}{n-1} \\
\frac{t(n)}{n+1} &= \left(\frac{2}{n+1} + \frac{2}{n} + \frac{2}{n-1}\right) b_{part} + \frac{t(n-3)}{n-2} \\
\frac{t(n)}{n+1} &= \left(\frac{2}{n+1} + \frac{2}{n} + \frac{2}{n-1} + \cdots + \frac{2}{casoBase+2}\right) b_{part} + \frac{t(casoBase)}{casoBase+1} \\
\frac{t(n)}{n+1} &= \left(\frac{2}{n+1} + \frac{2}{n} + \cdots + \frac{2}{casoBase+2}\right) b_{part} + \frac{casoBase^2\ k_{MD}}{casoBase+1}.
\end{aligned}
$$

Llegaremos a una ecuación que representa la aproximación del comportamiento del tiempo de ejecución, $t(n, casoBase, MD)$, en función del tamaño del problema y del caso base óptimo, en la siguiente forma:

$$t(n, casoBase, MD) = 2\ b_{part} \left(ln \frac{n+1}{casoBase+1} \right) + \frac{casoBase^2\ k_{MD}}{casoBase+1}. \tag{2.7}$$

Valor Óptimo del Caso Base

En el modelo secuencial se calcula el valor óptimo del caso base dado un tamaño de problema fijo, n, derivando la función del tiempo respecto de la variable $casoBase$ y se calcula el valor que anula la derivada:

$$\frac{\partial t}{\partial casoBase} = 0,$$

$$\frac{-2\ b_{part}}{casoBase+1} + \frac{k_{MD}\ casoBase\ (casoBase+2)}{(casoBase+1)^2} = 0,$$

$$k_{MD}\ casoBase^2 + (-2\ b_{part} + 2\ k_{MD})\ casoBase - 2\ b_{part} = 0,$$

$$casoBase = \frac{(2\ b_{part} - 2\ k_{MD}) + \sqrt{(-2\ b_{part} + 2\ k_{MD})^2 - 4\ (k_{MD})\ (-2\ b_{part})}}{2\ k_{MD}}. \tag{2.8}$$

Sustituyendo los valores de los parámetros del sistema (k_{MD}, b_{part}, c_{part}) calculados, podremos sacar el valor del caso base para el que se obtiene el menor tiempo de ejecución, para determinado tamaño del problema de forma directa.

2.4. Modelo Paralelo

En esta sección presentamos el esquema algorítmico y los modelos teóricos paralelos utilizados para el estudio del problemas analizado.

2.4.1. Esquema Algorítmico Paralelo Maestro−Esclavo

En el esquema paralelo Maestro−Esclavo tenemos que un proceso principal (Maestro) controla la actividad de un grupo de procesos (Esclavos), asignándoles trabajos que se han de realizar en paralelo. En este esquema se trabajó con los datos en un procesador inicial P_0, dividiendo el problema hasta llegar a obtener problemas para el número de procesadores que va a intervenir en la computación. Cada problema que no se sigue dividiendo en P_0 se envía a un procesador. En la fase de combinación, los procesadores que actúan como Esclavos envían a P_0 la solución de sus subproblemas, y éste combina las subsoluciones.

La forma típica de trabajo de este esquema es inicialmente trabajar con un único proceso Maestro, y poner en marcha una serie de procesos Esclavos, teniendo en cuenta que el proceso inicial accede a todos los datos y en la creación de los Esclavos se determina la forma en que accede a ellos todo el conjunto de procesos.

Consideraciones Generales del Esquema

En el esquema del Algoritmo 4 tenemos los pasos de la implementación de la solución del Divide y Vencerás Paralelo utilizado para estudiar la metodología propuesta.

Para el esquema paralelo podemos considerar que:

Algoritmo 4 Esquema Divide y Vencerás Paralelo Maestro-Esclavo.

DVParaleloME (p: problema, n: tamaño, i: id de proceso): solución
if MAESTRO then
  dividir p en bloques de problemas más pequeños p_1, p_2,..., p_k
  distribuir bloques a los procesos **ESCLAVOS**
  s_1 = **DivideVencerás** (p_1, n_1)
  recibir y combinar todos los resultados parciales
  generación del resultado final
else
  recibir del **MAESTRO** bloque en p_i
  s_i = **DivideVencerás** (p_i, n_i)
  enviar resultado al **MAESTRO**
end if
end DVParaleloME

- ◇ Se utiliza la programación por paso de mensajes para distribuir los bloques a los procesos.

- ◇ El proceso Maestro será el que genere el problema a resolver, posiblemente leyendo de la entrada estándar o de fichero o realizando una serie de cálculos iniciales para preparar los datos con que trabajar. El proceso Maestro será siempre el primer proceso inicializado con identificador igual a 0.

- ◇ Después de preparar los datos con que trabajar se distribuyen éstos desde el Maestro a los Esclavos, asignando bloques distintos de datos a procesos distintos.

- ◇ Una vez los datos han sido distribuidos por todo el sistema empieza la computación, pudiendo intervenir en ella el Maestro o no, y pudiendo haber partes de comunicación intercaladas en la computación.

- ◇ Al final de la computación habrá una fase de salida de datos, para lo que el Maestro recibe de los Esclavos sus resultados parciales y forma el resultado final produciendo la salida por pantalla o fichero, o dejando el resultado en memoria para ser tratado a continuación en otra parte del programa.

2.4.2. Modelo Teórico del Tiempo de Ejecución Paralelo

En la estimación del tiempo real de ejecución de un programa en paralelo a través de un modelo, se tiene en cuenta factores como: el tipo de los datos transmitidos, el esquema de almacenamiento en memoria de estos datos y las estrategias disponibles en el paradigma de paso de mensajes, para llevar a cabo las comunicaciones entre procesadores. El modelo del tiempo de ejecución se puede expresar como el sumatorio entre los tiempos de comunicar mensajes entre procesadores, en una red de interconexión homogénea con comunicaciones punto a punto, más el tiempo de cómputo aritmético.

Nuestro modelo del sistema completo estará formado por la suma de los tiempos de cómputo y de comunicaciones planteados. El conjunto de todos los parámetros del modelo caracterizarán, por tanto, la capacidad de cómputo y de comunicaciones del sistema completo, entendiendo al sistema como la unión de la plataforma hardware y de las rutinas de cómputo aritmético y de comunicaciones instaladas.

1) Modelado del Tiempo de Ejecución Paralelo para la Ordenación por Mezcla

A partir del Algoritmo Paralelo Maestro−Esclavo descrito anteriormente, podemos descomponer el tiempo de ejecución paralelo total del algoritmo de ordenación por mezcla en las siguientes partes:

Tiempo de Comunicación

Se considera t_0 el coste para transferir n datos entre el proceso Maestro y los procesos Esclavos, teniendo en cuenta las comunicaciones de envío y recepción de datos. Puesto que el proceso Maestro envía a los $p-1$ esclavos mensajes de tamaño $\frac{n}{p}$ y posteriormente recibe también, de éstos, mensajes de mismo tamaño, el tiempo de comunicación viene dado por:

$$t_0 = 2\ (p-1)\left(t_s + t_w \frac{n}{p}\right), \tag{2.9}$$

de manera que el coste del tiempo de comunicación del algoritmo varía en función del tamaño del problema n, pudiendo estimarse en función de los siguientes parámetros: $\boldsymbol{t_s}$ (el tiempo de inicio de una comunicación), $\boldsymbol{t_w}$ (el tiempo de transmisión de un dato) y $\boldsymbol{p}$ (número de procesadores). Los parámetros t_s y t_w se calculan a partir de una toma de tiempos experimental entre dos procesadores, realizando ejecuciones para varios tamaños de cantidades enviadas y recibidas, y haciendo un ajuste por mínimos cuadrados por una recta. Se estiman los valores de los parámetros de comunicaciones (t_s, t_w), a través de una técnica de estimación de los costes de envío y recepción de mensajes en una red, a través de un algoritmo denominado *ping pong* (veáse Algoritmo 5).

Algoritmo 5 Esquema del Algoritmo Ping Pong.

PingPong (*ProcA*, *ProcB*: procesos asignados, n: tamaño)
if ($myid == ProcA$) **then**
  Ping ($array$, n)
else
  Pong ($array$, n)
end if
end PingPong
Ping ($array$: lista de elementos vacia, n: tamaño)
while ($i < n$) **do**
  $array[i]$ = genera los datos de forma aleatoria
  $i = i + 1$
end while
$inicio$ = se mide el tiempo inicial
  Envia ($array$, $ProcB$, $LabelPing$, $comunicador$)
  Recibe ($array$, $ProcB$, $LabelPong$, $comunicador$, $status$)
fin = se mide el tiempo final
$tiempo = fin$ - $inicio$
end Ping
Pong ($array$: lista de elementos vacia, n: tamaño)
  Recibe ($array$, $ProcA$, $LabelPing$, $comunicador$, $status$)
  Envia ($array$, $ProcA$, $LabelPing$, $comunicador$)
end Pong

Tiempo de la Mezcla

El tiempo de la mezcla corresponde al tiempo, una vez ordenados los datos que se envían al proceso Maestro, de combinar los datos recibidos de todos los procesos. Esta combinación se basa en obtener una mejor eficiencia en la combinación de los bloques recibidos, teniendo como base la combinación por pares de elementos, siguiendo el orden del vector de longitudes. Esta combinación se denomina **Mezcla Especial** (véase Algoritmo 6).

Algoritmo 6 Esquema del Algoritmo Mezcla Especial.

```
MezclaEspecial(vectorLongitudes: lista de posiciones, array: lista de elementos, numproces: procesos asignados): solución
incremento = 2
while (vectorLongitudes[0] != tamaño de la lista) do
  for(i = 0; i <numproces; i = i + incremento)
  if ((posicion + vectorLongitudes[i]) <tamaño vector longitudes) then
    Mezcla(array[posicion],vectorLongitudes[i],array[posicion
    +vectorLongitudes[i]],vectorLongitudes[i + 1])
    posicion=posicion+vectorLongitudes[i]+vectorLongitudes[i+1]
  end if
  endfor
  incremento = incremento * 2
end while
end MezclaEspecial
```

En este algoritmo el proceso Maestro hace la mezcla de la misma forma que en la ordenación por mezcla. Se puede obtener el tiempo de la mezcla (t_1) teniendo como base el Algoritmo de **Mezcla Especial** de la siguiente forma:

$$\begin{aligned} t_1 &= \left(b_{merge}\frac{2n}{p} + c_{merge}\right)\frac{p}{2} + \cdots + (b_{merge}\ n + c_{merge}) \\ t_1 &= b_{merge}\ n\ log(p) + c_{merge}\ p\left(\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{p}\right) \\ &\vdots \\ t_1 &= b_{merge}\ n\ log(p) + c_{merge}(p-1). \end{aligned} \tag{2.10}$$

Tiempo de Ordenación

La ordenación se realiza en cada procesador de forma secuencial con bloques de tamaño $\frac{n}{p}$, por lo que el tiempo de ordenación se obtiene sustituyendo en la Ecuación 2.2, n por $\frac{n}{p}$. Como en el modelo secuencial tenemos que el coste de cómputo del algoritmo será el mismo, pero considerando que el tamaño del problema se reduce a $\frac{n}{p}$ en función de p procesadores computados, luego sustituyendo obtenemos la expresión:

$$t_2 = k_{MD}\left(\frac{n}{p}\right) casoBase + b_{merge}\left(\frac{n}{p}\right) log\left(\frac{\frac{n}{p}}{casoBase}\right) + c_{merge}\left(\frac{\frac{n}{p}}{casoBase} - 1\right). \tag{2.11}$$

Tiempo de Ejecución Total

Al final tendremos que el modelo paralelo del sistema completo estará formado por la suma de los tiempos de cómputo y de comunicaciones planteados. Luego el modelo analítico del tiempo de ejecución del algoritmo paralelo será la suma de las Ecuaciones 2.9, 2.10 y 2.11.

Valor Óptimo del Caso Base

De la misma forma como en el modelo secuencial se calcula el valor óptimo del caso base dado un tamaño de problema fijo, n, derivando la función del tiempo respecto de la variable *casoBase* y se calcula el valor que anula la derivada:

$$\frac{\partial t}{\partial casoBase} = 0,$$

$$k_{MD}\frac{n}{p} - \frac{b_{merge}\ n}{ln(2)\ casoBase\ p} - \frac{c_{merge}\ n}{casoBase^2\ p} = 0,$$

$$\frac{k_{MD}}{p}\ casoBase^2 - \frac{b_{merge}}{ln(2)\ p}\ casoBase - \frac{c_{merge}}{p} = 0,$$

$$casoBase = \frac{\frac{b_{merge}}{ln(2)\ p} + \sqrt{\left(\frac{b_{merge}}{ln(2)\ p}\right)^2 + \left(\frac{4\ k_{MD}\ c_{merge}}{p^2}\right)}}{\frac{2\ k_{MD}}{p}}.$$

Simplificando la ecuación se obtiene

$$casoBase = \frac{\frac{b_{merge}}{ln(2)} + \sqrt{\left(\frac{b_{merge}}{ln(2)}\right)^2 + 4\ k_{MD}\ c_{merge}}}{2\ k_{MD}}. \tag{2.12}$$

La ecuación se queda igual que la Ecuación 2.3, ya que el número de procesadores no modifica el valor del caso base. De este modo sustituyendo los valores de los parámetros del sistema (k_{MD}, b_{merge}, c_{merge}) calculados, de la misma forma podremos sacar el valor del caso base para el que se obtiene el menor tiempo de ejecución, para determinado tamaño del problema de forma directa.

2) Modelado del Tiempo de Ejecución Paralelo para la Ordenación Rápida

De la misma forma que en el modelo anterior conseguimos a partir del Algoritmo Paralelo Maestro−Esclavo descrito anteriormente, descomponer el tiempo de ejecución paralelo total del algoritmo de ordenación rápida en las siguientes partes:

Tiempo de Comunicación

El tiempo de comunicación será lo mismo que en la ordenación por mezcla, siendo estimado de la misma manera. Se considera $\bar{t_0}$ el coste para transferir n datos entre el proceso Maestro y los procesos Esclavos:

$$\bar{t_0} = 2\ (p-1)\left(t_s + t_w \frac{n}{p}\right). \tag{2.13}$$

Tiempo de Ordenación

La ordenación se realiza en cada procesador de forma secuencial con bloques de tamaño $\frac{n}{p}$, por lo que el tiempo de ordenación se obtiene sustituyendo en la Ecuación 2.7, n por $\frac{n}{p}$. Como en el modelo secuencial tenemos que el coste de cómputo del algoritmo será el mismo, pero considerando que el tamaño del problema se reduce a $\frac{n}{p}$ en función de p procesadores computados, luego sustituyendo obtenemos la expresión:

$$\bar{t_1}(n, casoBase, MD) = 2\ b_{part}\left(ln\frac{\frac{n}{p}+1}{casoBase+1}\right) + \frac{casoBase^2\ k_{MD}}{casoBase+1}. \tag{2.14}$$

Tiempo de Ejecución Total

Al final tendremos que el modelo paralelo del sistema completo estará formado por la suma de los tiempos de cómputo y de comunicaciones planteados. Luego el modelo analítico del tiempo de ejecución del algoritmo paralelo será la suma de las Ecuaciones 2.13 y 2.14.

Valor Óptimo del Caso Base

De la misma forma como en el modelo secuencial se calcula el valor óptimo del caso base dado un tamaño de problema fijo, n, derivando la función del tiempo respecto de la variable *casoBase* y se calcula el valor que anula la derivada:

$$\frac{\partial t}{\partial casoBase} = 0,$$

$$\frac{\partial \left(2\ b_{part} \left(ln \frac{\frac{n}{p}+1}{casoBase+1} \right) + \frac{casoBase^2\ k_{MD}}{casoBase+1} \right)}{\partial casoBase} = 0,$$

$$casoBase = \frac{(2\ b_{part} - 2\ k_{MD}) + \sqrt{(-2\ b_{part} + 2\ k_{MD})^2 - 4\ (k_{MD})\ (-2\ b_{part})}}{2\ k_{MD}}.$$

La ecuación se queda igual que la Ecuación 2.8, ya que el número de procesadores no modifica el valor del caso base. De este modo sustituyendo los valores de los parámetros del sistema (k_{MD}, b_{merge}, c_{merge}) calculados, de la misma forma podremos sacar el valor del caso base para el que se obtiene el menor tiempo de ejecución, para determinado tamaño del problema de forma directa.

2.5. Sistema Paralelo Autooptimizado

2.5.1. Diseño del Sistema Autooptimizado

En esta sección se describe la arquitectura de la rutina autooptimizada de un esquema Divide y Vencerás, y se distinguen los siguientes tipos de perfiles de usuarios: un perfil de *manager*, que se encarga de instalar la rutina en la plataforma de ejecución, grabando las informaciones de configuración del sistema, y un segundo perfil de *cliente*, que tiene un problema a resolver y sólo quiere obtener la solución, sin la necesidad de saber o cambiar detalles del sistema. La Figura 2.1 muestra el ciclo de vida de los componentes de cada etapa para los 2 perfiles.

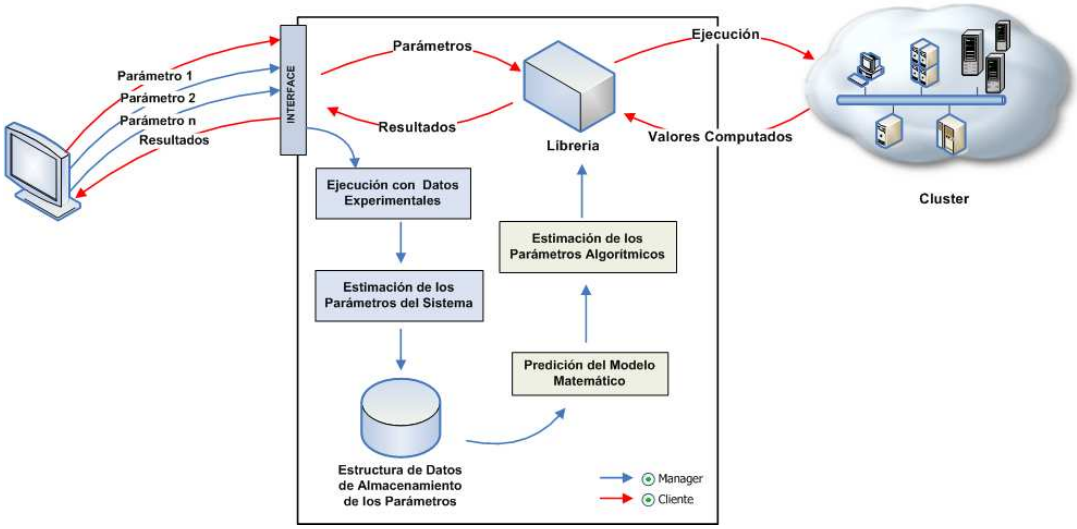

Figura 2.1: Ciclo de Vida de la Librería de Ordenación.

Se distinguen las siguientes etapas: Diseño, Instalación y Ejecución. En la etapa de **Diseño**, además de desarrollar la rutina, se obtiene un modelo analítico del tiempo de ejecución en función del tamaño del problema, de los parámetros del sistema y de los parámetros algorítmicos, en la forma: *t(n, SP, AP)*. La rutina de Instalación se desarrolla de manera que se ejecute y después sea posible la obtención de los parámetros a utilizar en la resolución del problema. Los parámetros AP a determinar en este caso son: tamaño del caso base, *casoBase*, el número de

procesadores, p, y el método directo de ordenación, *MD*. Teniendo como condición que la rutina de instalación se pueda volver a ejecutar cada vez que cambien las condiciones del sistema.

En el proceso de **Instalación** se realizan las siguientes tareas:

Ejecución con Datos Experimentales

Para cada método estudiado se generan cantidades experimentales siendo definidas previamente las características de las pruebas en la configuración del sistema. Esto es, se genera una cantidad de elementos de forma aleatoria, y se realiza una ordenación de estos datos para calcular los parámetros del sistema. En este caso la distribución de los datos se mantiene de manera estática, en un proceso principal, y su distribución se hace por bloques en el sistema de procesos utilizando cantidades equilibradas, ya que el sistema es homogéneo.

Estimación de los Parámetros del Sistema (SP)

Para los parámetros del sistema se estiman los costes de las operaciones aritméticas básicas que aparecen en la rutina. Se obtiene los parámetros por medio de una serie de ejecuciones, consiguiendo una mayor precisión a través de la realización de la mayor cantidad de experimentaciones posibles.

En tiempo de **Ejecución** se realizan las siguientes tareas:

Estimación de los Parámetros Algorítmicos (AP)

Para los parámetros algorítmicos estimamos el nivel de la recursión, el método directo utilizado para solucionar el problema, y el número de procesadores a utilizar. Los valores de *casoBase* y *MD* pueden ser obtenidos utilizando el modelo teórico del tiempo de ejecución, y los parámetros del coste de las operaciones aritméticas son estimados a través de una aproximación del coste teórico.

Modelo teórico Experimental

Se usa el modelo matemático del tiempo de ejecución desarrollado en función del tamaño del problema y de los parámetros algorítmicos y del sistema. El modelo ya estaba hecho en el diseño, por lo que basta con sustituir los valores de los parámetros, obteniéndose valores óptimos para la solución del problema.

2.6. Resultados Experimentales

Dentro del conjunto de experimentaciones planteadas para comprobar la validez de la propuesta de este trabajo, se utilizó una plataforma paralela, donde se implementaron y ejecutaron los algoritmos propuestos. Los experimentos consistirán en aplicar distintos tipos de ajuste a los algoritmos, variando parámetros como: tamaños del problema, tamaños del caso base, métodos de ordenación, etc, de cara a validar la propuesta de modelado. Todos los datos experimentados son del tipo entero de simple precisión generados aleatoriamente, los programas se han hecho en el lenguaje C, y los tiempos que se mostrarán serán en todos los casos tiempos en segundos, utilizando para las experimentaciones la plataforma **KEFREN**. En esta sección se evaluarán los algoritmos propuestos, y presentaremos los resultados más significativos.

2.6.1. Experimentación Secuencial

De cara a validar la propuesta de modelado, en esta sección se muestran diversas comparativas de los tiempos de ejecución de las rutinas secuenciales modeladas, en la plataforma descrita, con las rutinas de ordenación. Para las rutina se compararán el tiempo teórico según el modelo matemático secuencial propuesto en las Secciónes 2.2.1 y 2.3.1, y el tiempo de ejecución que realmente se obtiene.

Tabla 2.1: Desviaciones respecto al tiempo de ejecución experimental del tiempo teórico según el modelo secuencial, para la rutina secuencial de Ordenación por Mezcla (*Mergesort*) en la plataforma **KEFREN**, para tamaños del problema de 1, 5 y 10 millones.

1 millón	tiempo experimental	tiempo modelo	desviación %
Burbuja	0.41	0.52	20.00
Inserción	0.37	0.48	21.87
Selección	0.40	0.49	16.73

5 millones	tiempo experimental	tiempo modelo	desviación %
Burbuja	2.26	2.68	15.5
Inserción	2.18	2.46	12.00
Selección	2.27	2.53	10.67

10 millones	tiempo experimental	tiempo modelo	desviación %
Burbuja	4.97	5.40	7.90
Inserción	4.54	4.95	8.16
Selección	4.74	5.11	7.04

Tabla 2.2: Desviaciones respecto al tiempo de ejecución experimental del tiempo teórico según el modelo secuencial, para la rutina secuencial de Ordenación Rápida (*Quicksort*) en la plataforma **KEFREN**, para tamaños del problema de 1, 5 y 10 millones.

1 millón	tiempo experimental	tiempo modelo	desviación %
Burbuja	0.09	0.10	10.20
Inserción	0.08	0.10	19.00
Selección	0.09	0.16	16.00

5 millones	tiempo experimental	tiempo modelo	desviación %
Burbuja	1.97	1.89	4.22
Inserción	1.95	1.86	4.82
Selección	1.96	1.87	4.81

10 millones	tiempo experimental	tiempo modelo	desviación %
Burbuja	8.14	7.94	2.55
Inserción	7.71	7.87	2.07
Selección	7.73	7.89	2.06

Tabla 2.3: Parámetros del Sistema Calculados en la Plataforma **KEFREN** para el Modelo Secuencial.

Parámetros del Sistema	
$k_{burbuja}$	0.016
$k_{insercion}$	0.014
$k_{seleccion}$	0.015
b_{merge}	0.012
c_{merge}	2.806
b_{part}	0.019
c_{part}	0.0034

Tabla 2.4: Parámetros Algorítmicos Óptimos Seleccionados por el Modelo Secuencial para a Plataforma **KEFREN**.

	Parámetros Algorítmicos			
	Mergesort		Quicksort	
	casoBase	MD	casoBase	MD
1000000	18	Inserción	20	Inserción
5000000	18	Inserción	20	Inserción
10000000	18	Inserción	20	Inserción

Los valores de los Párametros del Sistemas fueron obtenidos a través de la experimentación con la herramienta gráfica web desarrollada ajustando los datos por mínimos cuadrados y generando los valores de forma aleatoria.

Los resultados muestran cómo el modelo analítico de una rutina paralela puede predecir el comportamiento del algoritmo gracias al acercamiento a la realidad que se introduce en la fórmula teórica cuando se usan los valores calculados experimentalmente de los Parámetros del Sistema. Estos parámetros toman valores diferentes dependiendo de la plataforma hardware donde las librerías son utilizadas, del tamaño de problema a resolver y de algunas características del algoritmo, como, por ejemplo, el tamaño del caso base utilizado.

Constatamos también que los valores experimentados fueron razonablemente satisfactorios, y especialmente óptimos para tamaños grandes del problema. Por lo tanto, la implementación de la optimización, a través del desarrollo del modelo matemático secuencial es capaz de predecir el comportamiento de los algoritmos, para la ordenación por mezcla y rápida.

2.6.2. Experimentación Paralela

Por último, en esta sección se va a mostrar el comportamiento de las rutinas de ordenación paralelas, a través de diversas comparativas de los tiempos de ejecución, para la plataforma **KEFREN**. Para las rutinas se compararán el tiempo teórico según los modelos matemático paralelos propuestos en la Sección 2.4.2, y el tiempo de ejecución paralelo que realmente se obtiene.

El objetivo principal sigue siendo, obviamente, realizar una buena toma de decisiones para los valores de los parámetros algorítmicos, como se hacía en la primera propuesta secuencial, pero, ahora, con la necesidad de tener en cuenta la carga de trabajo de la plataforma repartida entre los procesadores en el momento de la ejecución de la rutina paralela.

Tabla 2.5: Desviaciones respecto al tiempo de ejecución experimental del tiempo teórico según el modelo paralelo, para la rutina paralela de Ordenación por Mezcla (*Mergesort*) en la plataforma **KEFREN**, para tamaños del problema entre 100 mil y 10 millones, utilizando un número óptimo de procesadores.

tamaño	procesadores	t. experimental	t. modelo	desviación %
100000	1	0.02	0.04	37.77
500000	1	0.15	0.23	34.19
1000000	2	0.36	0.46	20.90
5000000	4	1.15	1.32	13.01
10000000	6	1.81	2.05	12.05

Tabla 2.6: Desviaciones respecto al tiempo de ejecución experimental del tiempo teórico según el modelo paralelo, para la rutina paralela de Ordenación Rápida (*Quicksort*) en la plataforma **KEFREN**, para tamaños del problema entre 100 mil y 10 millones, utilizando un número óptimo de procesadores.

tamaño	procesadores	t. experimental	t. modelo	desviación %
100000	1	0.03	0.02	33.33
500000	2	0.22	0.21	4.78
1000000	2	0.38	0.68	44.11
5000000	8	1.25	1.46	14.38
10000000	10	2.05	1.58	29.74

La experimentación para recoger las características de la plataforma se realiza durante la etapa instalación, consigue en el momento de la ejecución un menor coste computacional al aplicar los valores de los parámetros del sistema cálculados en función de esta carga. Tras ello, ya se pueden escoger los valores de los parámetros algorítmicos que minimizan el tiempo de ejecución teórico.

Tabla 2.7: Parámetros del Sistema Calculados en la Plataforma **KEFREN** para el Modelo Paralelo.

Parámetros del Sistema	
$k_{insercion}$	0.0140
b_{merge}	0.012
c_{merge}	2.806
b_{part}	0.019
c_{part}	0.0034
t_w	0.003
t_s	113353

Tabla 2.8: Parámetros Algorítmicos Óptimos Seleccionados por el Modelo Paralelo para a Plataforma **KEFREN**.

	Parámetros Algorítmicos					
	Mergesort			Quicksort		
	$casoBase$	MD	$proces.$	$casoBase$	MD	$proces.$
100000	18	Inserción	1	20	Inserción	1
500000	18	Inserción	1	20	Inserción	2
1000000	18	Inserción	2	20	Inserción	2
5000000	18	Inserción	4	20	Inserción	8
10000000	18	Inserción	6	20	Inserción	10

Los resultados anteriores muestran que la diferencia entre los dos tiempos, del modelo y el experimental para ambos los casos, fueron satisfactorios. En el estudio del *Mergesort* y del *Quicksort* obtenemos buenos resultados, con bajas discrepancias entre los valores teóricos y experimentales. Se resalta apenas una discontinuidad un poco mayor en el segundo caso en relación al primero, debido a la influencia del factor de la componente probabilística del algoritmo.

Tabla 2.9: Tiempo de Ejecución Paralelo para la Ordenación por Mezcla (*Mergesort*) que **realmente** se obtiene en la Plataforma **KEFREN**. En negrito se marcan el valor óptimo a los parámetros elegidos.

Procesadores	100000	500000	1000000	5000000	10000000
1	**0.18**	**0.25**	0.37	2.18	4.54
2	0.20	0.27	**0.36**	1.25	2.39
4	0.51	0.54	0.64	**1.15**	2.02
6	1.05	1.50	1.31	1.60	**1.87**
8	1.05	1.51	1.32	1.61	2.03
10	1.36	1.59	1.47	1.77	2.05
12	1.83	1.66	1.67	1.93	2.31
14	2.05	2.71	2.00	2.16	2.53
16	2.28	2.05	2.12	2.41	2.84

Tabla 2.10: Tiempo de Ejecución Paralelo para la Ordenación Rápida (*Quicksort*) que **realmente** se obtiene en la Plataforma **KEFREN**. En negrito se marcan el valor óptimo a los parámetros elegidos.

Procesadores	100000	500000	1000000	5000000	10000000
1	**0.03**	0.31	0.89	1.95	7.71
2	0.12	**0.22**	**0.38**	5.28	9.85
4	0.35	0.38	0.45	1.76	5.50
6	0.60	0.63	0.65	1.30	3.04
8	0.82	0.81	0.85	**1.25**	2.31
10	1.04	1.05	1.11	1.41	**2.05**
12	1.28	1.28	1.29	1.69	2.11
14	1.56	1.51	1.52	1.72	2.13
16	1.82	1.77	1.79	1.93	2.44

En el estudio hecho para un tamaño variable del problema se obtiene un valor óptimo de procesadores (tabla 2.8). Por otro lado en las tablas 2.9 y 2.10 vemos que subiendo el número de procesadores, bajamos el tiempo de ejecución hasta un cierto punto, y que aumentando esta cantidad, el tiempo vuelve a subir debido el *overhead* de las comunicaciones. De la misma forma comparando los tiempos de ejecución experimental y del modelo, en las tablas 2.8, 2.9 y 2.10 vemos que los valores coinciden, por lo cual concluimos que el comportamiento de algoritmo se predice por el modelo matemático.

A partir de estés resultados se puede concluir que la técnica de parametrización da buenos resultados, y que la implementación de la optimización, a través del desarrollo del modelo matemático paralelo, es capaz de predecir el comportamiento del algoritmo, siendo posible elegir unos parámetros cercanos a los óptimos.

2.7. Resumen y conclusiones

Este capítulo resume un conjunto de técnicas de autooptimización aplicadas a esquemas algorítmicos basados en el paradigma Divide y Vencerás. Se han utilizado para estudiar la metodología propuesta en problemas de ordenación. Los resultados experimentales muestran cómo el modelo teórico de una rutina puede predecir su comportamiento gracias al acercamiento a la realidad que se introduce en la fórmula teórica cuando se usan los valores medidos experimentalmente de los parámetros del sistema. Estos parámetros toman valores diferentes dependiendo de la plataforma hardware, del tamaño de problema a resolver y de algunas características del algoritmo. Esta parte del trabajo sirvió para adecuar la técnica utilizada para el desarrollo de librerías con capacidad de optimización automática a sistemas homogéneos. Se pretende adecuar el estudio a estrategias de asignación de trabajos a procesadores en un ambiente heterogéneo [9], con el fin de que el diseño de la rutina fuera capaz de obtener ejecuciones cercanas a las óptimas sin intervención del usuario.

Capítulo 3

Adecuación de las Técnicas de Autooptimización de Esquemas Divide y Vencerás para el Cálculo de Valores Propios a Sistemas Heterogéneos

En este capítulo se estudian las técnicas de autooptimización aplicadas a un esquema Divide y Vencerás para el Cálculo de Valores Propios y su adecuación a sistemas paralelos heterogéneos.

3.1. Introducción

En este capítulo se presentan y analizan diferentes estrategias de computación de un problema típico que se resuelve por Divide y Vencerás: el cálculo de valores propios de matrices tridiagonales simétricas, utilizando técnicas de computación paralela, adaptadas a un ambiente heterogéneo de computación. La primera estrategia se basa en una distribucción homogénea de los datos asignando partes iguales a los procesadores con capacidades distintas de procesamiento. La segunda estrategia se basa en una distribucción heterogénea de datos dependiendo de la capacidad de cómputo de las máquinas. La implementación del cálculo de los valores propios sirvió para mostrar el logro del mejor funcionamiento para plataformas heterogéneas, obteniendo el máximo rendimiento de la potencias de cómputo, en máquinas de distintas naturaleza trabajando juntas en la solución de un determinado problema científico.

3.2. Formulación Matemática del Algoritmo

El primer paso para estudiar el algoritmo para el cálculo de valores propios es entender matemáticamente cómo trabaja el algoritmo. El problema de valores propios simétricos consiste en determinar los valores de $\lambda \in \mathcal{R}$, tal que el sistema

$$Tx = \lambda x \tag{3.1}$$

tiene solución no trivial, siendo $T \in \mathcal{R}^{n \times n}$ simétrica y $x \in \mathcal{R}^n$, con $x \neq 0$. El escalar λ y el vector x se denominan valor y vector propios de T, respectivamente. La ecuación 3.1 se puede escribir como $(T - \lambda I)x = 0$, por lo que existe una solución no trivial si la matriz $(T - \lambda I)$ es singular, es decir, si $det(T - \lambda I) = 0$. Expandiendo este determinante que denominamos polinomio característico, se tiene que

$$(-1)^n \lambda^n + \alpha_{n-1}\lambda^{n-1} + \cdots + \alpha_1 \lambda + \alpha_0 = 0, \tag{3.2}$$

de lo que se deduce que existen exactamente n valores propios. No es una buena idea obtener los valores propios mediante el cálculo de las raíces del polinomio característico por ser un problema mal condicionado, por lo que se recurre a otros métodos. El método Divide y Vencerás destaca como uno de los métodos más rápidos, para el caso de matrices tridiagonales simétricas. El algoritmo tiene como punto de partida una matriz simétrica real tridiagonal T que puede ser descompuesta en una suma de 2 matrices de la siguiente manera:

$$T = \left[\begin{array}{ccc|ccc} \ddots & \ddots & & & & \\ \ddots & a_{m-1} & b_{m-1} & & & \\ & b_{m-1} & a_m & b_m & & \\ \hline & & b_m & a_{m+1} & b_{m+1} & \\ & & & b_{m+1} & a_{m+2} & \ddots \\ & & & & \ddots & \ddots \end{array}\right] =$$

$$\left[\begin{array}{ccc|ccc} \ddots & \ddots & & & & \\ \ddots & a_{m-1} & b_{m-1} & & & \\ & b_{m-1} & a_m - b_m & & & \\ \hline & & & a_{m+1} - b_m & b_{m+1} & \\ & & & b_{m+1} & a_{m+2} & \ddots \\ & & & & \ddots & \ddots \end{array}\right] + \left[\begin{array}{cc|c} & & \\ & b_m & b_m \\ \hline & b_m & b_m \\ & & \\ & & \end{array}\right].$$

Consideramos que $m{=}\frac{n}{2}$ como primera simplificación. Entonces tenemos que:

$$T = \left[\begin{array}{cc} T_1 & \\ & T_2 \end{array}\right] + \rho u u^T, \quad donde \quad \rho = b_m,$$

de tal forma que T_1 y T_2 son submatrices con la misma estructura de T y $u = [0, ..., 1, 1, 0..., 0]^T$. A su vez, si consideramos la descomposición en valores propios de $T_i = Q_i D_i Q_i^T$, con Q_i ortogonal y D_i diagonal, tenemos que

$$T = \begin{bmatrix} Q_1 D_1 Q_1^T & \\ & Q_2 D_2 Q_2^T \end{bmatrix} + \rho u u^T.$$

Por tanto

$$T = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \left\{ \begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} + \rho z z^T \right\} \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix},$$

por lo que los valores de z corresponden a un vector formado por la última y primera columna de las transpuestas de Q_i, dado que las matrices Q_1 y Q_2 son ortogonales,

$$u = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} z \ \Rightarrow z = \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix} u,$$

Como la matriz T es semejante a la matriz $D + \rho z z^T$, sus valores propios coinciden. Sí tenemos que λ es un valor propio de $D + \rho z z^T$ y q el vector propio correspondiente:

$$\begin{aligned} det(D + \rho z z^T - \lambda I) &= 0 \\ det[(D - \lambda I)(I + \rho (D - \lambda I)^{-1} z z^T)] &= 0 \\ (det(D - \lambda I))(det(I + \rho (D - \lambda I)^{-1} z z^T)) &= 0. \end{aligned}$$

Supongamos que $det(D - \lambda I) \neq 0$, esto implica que $det(I + \rho (D - \lambda I)^{-1} z z^T) = 0$.

Si suponemos las siguientes restricciones: $d_i \neq d_j$ si $i \neq j$ y que $z_i \neq 0 \ \forall_{i,j}$, podemos escribir la ecuación de la siguiente manera:

$$f(\lambda) = 1 + \rho \sum_{i=1}^{n} \frac{z_i^2}{d_i - \lambda} = 0, \tag{3.3}$$

donde se ha utilizado la propiedad $det(I + ab^T) = 1 + b^T a$, reduciéndose así el problema del cálculo de los valores propios de una matriz simétrica real tridiagonal T, al cálculo de las raíces en una función racional. El cálculo de estas raíces puede llevarse a cabo mediante el método numérico de Newton [51], que converge de forma rápida y consistente, o mediante cualquier otro método de cálculo de raíces de ecuaciones no lineales. Además, si λ_i es un valor propio de $D + \rho zz^T$, entonces $(D - \lambda_i I)^{-1} z$ es su correspondiente vector propio ya que

$$\begin{aligned} & (D + \rho zz^T)(D - \lambda_i I)^{-1} z \\ = \; & D(D - \lambda_i I)^{-1} z + \rho zz^T (D - \lambda_i I)^{-1} z \\ = \; & (D - \lambda_i I)^{-1} Dz - z \\ = \; & (D - \lambda_i I)^{-1} (Dz - (D - \lambda_i I)z) \\ = \; & \lambda_i (D - \lambda_i I)^{-1} z. \end{aligned}$$

3.2.1. Deflación

La deflación es el proceso por el cual podemos reducir el tamaño de un problema cuando conocemos parte de su solución, bien porque la hemos calculado previamente, o bien porque ésta puede obtenerse de un modo más o menos trivial [52]. En nuestro caso, si $det(D - \lambda I) = 0 \Rightarrow d_i$ es valor propio de $D + \rho zz^T$, para algún i:

$$det(D - \lambda I) = 0 \Rightarrow det \begin{bmatrix} d_1 - \lambda & & \\ & \ddots & \\ & & d_n - \lambda \end{bmatrix} = 0 \Rightarrow d_i - \lambda = 0 \Rightarrow \lambda = d_i.$$

Si d_i es valor propio de $D + \rho zz^T$

$$
\begin{aligned}
(D + \rho z z^T)q &= d_i q \\
Dq + \rho z(z^T q) &= d_i q \\
d_i q_i + \rho z_i (z^T q) &= d_i q_i,
\end{aligned}
$$

de tal forma que tenemos $z_i = 0$, o bien $d_i = d_j, i \neq j$, ya que $z^T q = 0 \Rightarrow Dq = d_i q \Rightarrow$

$$
\begin{aligned}
d_1 q_1 &= d_i q_1 \\
d_2 q_2 &= d_i q_2 \\
&\vdots \\
d_n q_n &= d_i q_n.
\end{aligned}
$$

Caso 1: $z_i = 0$**, para algún** i

Una primera clase de deflación se produce cuando $z_i = 0$, para algún i. En este caso los valores propios de $D + \rho z z^T$ son los valores propios la matriz deflactada $\bar{D} + \rho \bar{z} \bar{z}^T$ (definida después) junto con d_i. Se puede calcular los valores propios de la siguiente forma:

Si $z_i = 0$, aplicando una permutación P_{in}:

$$
P_{in} \left(D + \rho \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \begin{bmatrix} z_1 & z_2 & \cdots & z_n \end{bmatrix} \right) P_{in}^T,
$$

se llegaría a una matriz de la forma

$$\left(\begin{array}{c|c} \bar{D} + \rho\bar{z}\bar{z}^T & 0 \\ \hline 0 & d_i \end{array}\right),$$

cuyos los valores propios son $\{\lambda(\bar{D} + \bar{\rho}\bar{z}\bar{z}^T),\ d_i\}$.

Caso 2: $d_i = d_j$**,** $i \neq j$

Una segunda clase de deflación puede producirse cuando los elementos de la diagonal de D, cumplen que $d_i = d_j$, $i \neq j$. En este caso podemos aplicar una rotación de Givens [47] que anule la componente z_j. Se aplica la rotación de Givens para calcular los valores propios de la siguiente forma:

Si $d_i = d_j$, $i \neq j$, y calculamos una rotación de Givens (G_{ij}), 2×2, tal que

$$G_{ij} \left[\begin{array}{c} z_i \\ z_j \end{array}\right] = \left[\begin{array}{c} \bar{z}_i \\ 0 \end{array}\right],$$

se puede aplicar ahora a la matriz $D + \rho z z^T$ en la forma $\bar{G}_{ij}(D + \rho z z^T)\bar{G}_{ij}^T$, donde $\bar{G}_{ij}$ es la correspondiente matriz de Gives, $n \times n$. Se obtiene una matriz $\widetilde{D} + \rho\widetilde{z}\widetilde{z}^T$ semejante, con $z_j = 0$, por lo que se puede permutar convenientemente para reducir el tamaño del problema de la misma forma que en el caso anterior

$$P_{jn}\bar{G}_{ij}T\bar{G}_{ij}^T P_{jn}^T = \left[\begin{array}{c|c} \widetilde{D} + \rho\widetilde{z}\widetilde{z}^T & 0 \\ \hline 0 & d_i \end{array}\right],$$

cuyos los valores propios son $\{\lambda(\widetilde{D} + \rho\widetilde{z}\widetilde{z}^T),\ d_i\}$.

3.3. Diseño e Implementación de un Algoritmo Secuencial para Cálculo de Valores Propios

El algoritmo Divide y Vencerás para calcular valores propios está dividido en las siguientes etapas:

- ◇ **Determinación del Tamaño Mínimo**. Comprobación que $orden(T)$ es igual al valor del caso base elegido.
- ◇ **Solución**. Devuelve los valores y vectores propios de una matriz.
- ◇ **Dividir**. División de la matriz original, en dos submatrices del mismo tamaño.
- ◇ **Combinar**. Se calculan los valores y vectores propios de la matriz original a partir de los valores y vectores propios de sus submatrices, calculando las raíces de la función racional 3.3.

Se aumenta el potencial del algoritmo aplicando el cálculo de los valores propios de las submatrices de manera recursiva de tal manera que este proceso se repite hasta obtener matrices de tamaño 2×2 (véase Algoritmo 7).

Algoritmo 7 Cálculo de los Valores Propios basado en Divide y Vencerás.

[D,Q]=DVSecuencial(T: matriz tridiagonal, n: tamaño)
if (orden(T)==2) **then**
  return($valoresPropios$, $vectoresPropios$ de T)
else
  Dividir T **en** T_1 **y** T_2
  $[D_i, Q_i]$**=DVSecuencial(**T_i**,** n_i**)**
  Resolución $f(\lambda) = 1 + \rho \sum_{i=1}^{n} \frac{z_i^2}{d_i - \lambda} = 0$
  Cálculo Vectores Propios
end if
end DVSecuencial

3.4. Diseño e Implementación de un Algoritmo Paralelo Homogéneo para el Cálculo de Valores Propios

Para la estrategia homogénea se pensó en aplicar el paralelismo al algoritmo Divide y Vencerás intentando distribuir cómputos entre los procesadores uniformemente. Esta estrategia se basa en una distribucción homogénea HoHo [53], que consiste en una distribución homogénea de los procesos del programa paralelo, y distribución homogénea del volumen de datos. Según la estrategia desarrollada, los datos se distribuyen uniformemente entre los procesadores de distintas características de cómputo. Se pueden destacar algunos puntos principales de la estrategia HoHo implementada teniendo como base el algoritmo Divide y Vencerás Paralelo a seguir:

Algoritmo 8 Esquema Divide y Vencerás Paralelo Maestro-Esclavo para Cálculo de Valores Propios.

DVParaleloME (T: problema, n: tamaño, i: id de proceso): solución
if MAESTRO then
  dividir T en 2 bloques de problemas más pequeños T_i y T_{i+1}
  while ($i <$*num_procesadores*) **do**
    dividir T_i en T_l y T_{l+1}
    dividir T_{i+1} en T_k y T_{k+1}
  end while
  distribuir bloques a los procesos **ESCLAVOS**
  $[D_1, Q_1]$ = **DVSecuencial** (T_1, n_1)
  recibir y combinar todos los resultados parciales
  generación del resultado final
else
  recibir del **MAESTRO** bloque en P_j
  calcular en P_j: $[D_j, Q_j]$ = **DVSecuencial** (T_j, n_j)
  enviar resultado al **MAESTRO**
end if
end DVParaleloME

◇ **Asignación de Procesos:** Se asigna 1 proceso por nodo.

◇ **División de los Datos:** El algoritmo paralelo descompone el problema

inicial en 2 partes, y de forma sucesiva divide el problema siempre en partes fijas y de igual tamaño, siendo que en este caso el número de divisiones siempre será proporcional a la cantidad de nodos existentes.

- ◇ **Modelo Paralelo:** El modelo paralelo elegido fue el Maestro−Esclavo, es decir un proceso principal (Maestro) se encarga de distribuir y organizar los datos a los demás procesos (Esclavos).

A pesar de que la implementación paralela homogénea desarrollada consigue ejecutarse en una plataforma paralela heterogénea, una de las desventajas de esta estrategia es que, cuando funciona en este tipo de plataforma, el tiempo total de cómputo está limitado por la velocidad del procesador más lento, ya que los procesadores más rápidos esperan a los procesadores más lentos para hacer la sincronización al final del cómputo. Por eso se hizo necesario el desarrollo de una implementación paralela heterogénea capaz de obtener el máximo rendimiento de la potencia de computadores de distinta naturaleza, y que trabajando juntos fuera posible obtener la solución del cálculo de valores propios con mejores prestaciones.

3.5. Diseño e Implementación de un Algoritmo Paralelo Heterogéneo para el Cálculo de Valores Propios

En esta sección se adapta la técnica de autooptimización Divide y Vencerás a un ambiente heteregéneo, donde los recursos de cómputo poseen características diferentes. La estrategia de cómputo consiste en asignar más datos a los procesos con mayor capacidad de cómputo. Se eligió un modelo de cómputo heterogéneo: un modelo estático de asignación de datos, que consiste en que el modelo de las prestaciones es calculado en un paso anterior previo, teniendo definidas las asignaciones a partir de la potencias de las maquinas de cómputo. Esta estrategia consiste en la utilización de una herramienta que es responsable del mecanismo automático del mapeo de los procesos y del Modelo de Prestaciones.

3.5.1. Modelo Estático de Asignación

Para obtener el máximo rendimiento de las plataformas paralelas heterogéneas, teniendo como base el modelo estático de asignación, es necesario hacer una distribución heterogénea del trabajo que se haga entre los procesadores, según sus velocidades de cómputo. Dos estrategias principales para tal distribución del cómputo pueden ser consideradas [54]:

- ◇ **HeHo:** Distribución heterogénea de los procesos del programa paralelo, y distribución homogénea del volumen de datos.
- ◇ **HoHe:** Distribución homogénea de los procesos del programa paralelo, y distribución heterogénea del volumen de datos.

En el primer caso, la estrategia HeHo no requiere cambios en las rutinas paralelas existentes, puestas en ejecución previamente de una manera homogénea. La meta principal consiste en mapear los procesos sobre los procesadores según sus prestaciones, distribuyendo la carga según su capacidad de computación calculada anteriormente. En el segundo caso, la aplicación requiere una redefinición del mapeo de los datos, de tal forma que la asignación de los datos ocurre también en función de la capacidad de computación de las máquinas, pero ahora considerando la distribucción heterogénea de los datos.

En nuestro estudio de caso comparamos la implementación de la estrategia HeHo de una aplicación utilizando paso de mensajes mediante la herramienta HeteroMPI [20] con rutinas LAPACK y ScaLAPACK para el cálculo de valores propios.

1) Metodología para el Cálculo de las Potencias Relativas de Cómputo

En el mundo del cálculo científico, tradicionalmente el parámetro de mayor interés es la potencia de cálculo del procesador en operaciones de coma flotante, estimada en flops. Se puede definir esta magnitud como el número de operaciones en coma flotante por segundo que es capaz de ejecutar un procesador [20].

A partir de esto, el primer paso para la implementación paralela heterogénea consistió en desarrollar una metodología simple que permitiera medir las potencias relativas de cómputo de las máquinas pertenecientes a un grupo de procesadores con características heterogéneas. La técnica desarrollada consistió en un conjunto de experimentaciones con la rutina secuencial para cálculo de los valores propios sobre un conjunto de procesadores, midiendo sus respectivos tiempos absolutos, a través de la razón entre el tiempo experimental y el comportamiento asintótico de la rutina, en nuestro caso cuadrático:

$$t_{absoluto}(i) = \frac{t_{experimentación}}{O(n^2)}. \tag{3.4}$$

Después de estimados los tiempos absolutos de cada máquina se calculan los tiempos relativos, estos tiempos se expresan en porcentajes, y se denominan potencias relativas de cómputo. La potencia relativa de cómputo se define como la razón entre el respectivo tiempo absoluto de cada máquina y el sumatorio de todos los tiempos absolutos del grupo de cómputo.

$$Potencia(i) = \frac{t_{absoluto}(i)}{\sum_{i=1}^{n} t_{absoluto}(i)} * 100\,\%. \tag{3.5}$$

Esta medida nos sirve como una herramienta valiosa para la asignación de trabajos de manera eficiente, ya que en función de las características heterogéneas de las máquinas se consigue obtener un mayor balanceo de las cargas a ser computadas. La Figura 3.1 representa el esquema de los pasos de la metodología implementada.

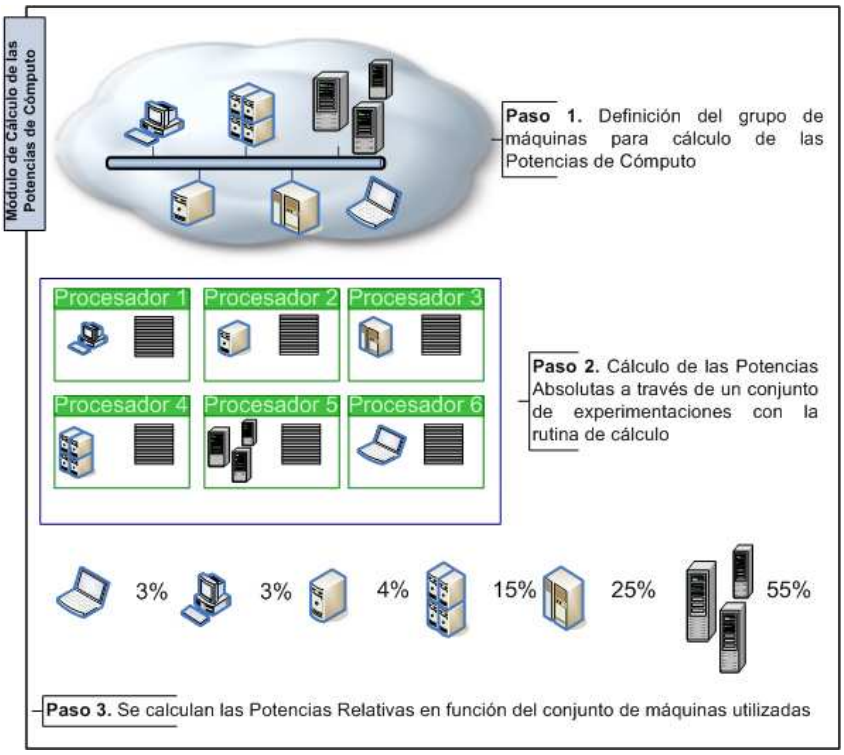

Figura 3.1: Cálculo de las Potencias Relativas de Cómputo.

En lugar de obtener índices absolutos de potencia para cada procesador basados en referencias de fabricación, trata de obtener una medida de la ejecución en tiempo real con la aplicación, y así obtener la capacidad de cómputo relativa. Al final, estas pruebas ofrecerán para cada procesador un valor característico expresado en porcentajes de capacidad de cómputo real para cada máquina perteneciente al conjunto de computación elegido.

2) Estrategia de Mapeo de los Procesos

La estrategia de mapeo se basa en una distribución homogénea del volumen de datos, y una distribución heterogénea de los procesos en un ambiente heterogéneo, en función de las potencias de las máquinas. La Tabla 3.1 muestra la aplicación de la Estrategia de Mapeo de los Procesos con el grupo de cómputo formado por 6 procesadores, caracterizado con sus respectivas potencias relativas de cómputo.

Tabla 3.1: Estrategia de Mapeo de los Procesos.

Procesadores	Potencia de las Máquinas	Mapeo de los Procesos
P_0	10 %	2
P_1	10 %	2
P_2	15 %	3
P_3	15 %	3
P_4	25 %	5
P_5	25 %	5
	100 %	**20**

La inicialización de los grupos de procesos en HeteroMPI se define a partir de una estructura, denominada máquina paralela virtual (VMP). Esta estructura está representada por un archivo que define el número de procesadores conocidos y el número de procesos inicializados, junto al nombre de las máquinas correspondientes, de tal forma que la cantidad de procesos inicializados es proporcional a los valores de Mapeo de los Procesos. A través del comando *mpccreate* [25] se inicializa este fichero de máquinas, sirviendo como base en la ejecución para todas las máquinas que son utilizadas. Se puede ejemplificar un fichero de máquinas de la siguiente forma:

```
# <name_of_machine> <number_of_processes> [number_of_processors]
# Hostname of the machines
# Number of processes to run on machine
# Number in square brackets indicate the number of processors

   rosebud01 2  [1]
   rosebud02 2  [1]
   rosebud03 6  [2]
   rosebud04 6  [2]
   rosebud05 40 [8]
   rosebud06 40 [8]
```

El fichero de máquinas utilizado por la implementación paralela heterogénea representará el numero ideal de procesos a ser ejecutado en cada nodo, este valor es obtenido como el producto del número de procesos mapeados por el número de procesadores existentes en cada maquina. La idea fue optimizar este fichero de máquinas haciendo el mapeo de los procesos en función de las potencias de cómputo de las máquinas calculadas, aprovechando al máximo la heterogeneidad de cómputo por el grupo elegido.

3) Estrategia de Asignación de Datos a los Procesos

La estrategia de asignación de submatrices a los procesos se basa en una decisión estática tomada antes de la ejecución. Los datos se asignarán a cada proceso conociendo de antemano su orden de ejecución. La asignación de los procesos consiste en descomponer los cálculos en un número determinado de datos conectadas según una estructura de comunicación particular de paso de mensajes. Los procesos inicializados tienen como principales características: el agrupamiento previo de un conjunto de datos en un único proceso y la selección de un orden de ejecución de los datos. La asignación de los procesos ocurre de forma cíclica [55], siendo balanceada de la mejor manera posible, teniendo como base el mapeo de los procesos. El siguiente esquema ilustra un ejemplo del la estrategia de asignación de datos a un grupo de 4 procesadores heterogéneos (veáse Figura 3.2).

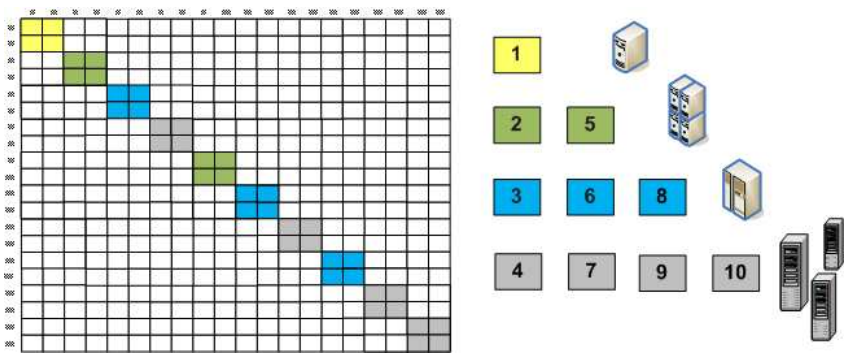

Figura 3.2: Estrategia de Asignación de los Datos a Procesadores Heterogéneos.

3.5.2. Modelo HeteroMPI

HeteroMPI es una extensión de MPI que sirve para implementar algoritmos de alto rendimiento en redes heterogéneas de computadores. Se basa en que el programador describa el Modelo de Prestaciones de funcionamiento del algoritmo puesto en ejecución en una forma genérica. Su principal objetivo es automatizar y optimizar la selección de un grupo de procesos, de forma que puedan ejecutar un algoritmo heterogéneo de la forma más eficiente posible. Para conseguir este objetivo, HeteroMPI utiliza un lenguaje para la especificación del modelo de rendimiento. Éste lenguaje es un subconjunto de mpC, el cual permite al programador definir explícitamente una red abstracta y distribuir datos, cálculos y comunicaciones en dicha red. Partiendo de este modelo, HeteroMPI automáticamente adapta la red abstracta a una red real, ajustando dinámicamente el modelo de rendimiento a parámetros específicos de la red, tales como la potencia de cálculo de los procesadores o la capacidad de los enlaces de comunicación (veáse Algoritmo 9).

Algoritmo 9 Modelo de Prestaciones en mpC.

```
nettype eigenvalue(int fa, int p, int b, int powers[p])
{
 coord I=p;

 node{
  I>=0: bench*(powers[I]);
 };
 link (K=p, L=p){
  I==0 && I!=K: length*(powers[K]*b*b*sizeof(double))[0]->[K];
  I==0 && I!=L: length*(powers[L]*b*b*sizeof(double))[L]->[0];
 };
 parent [0];
 scheme{
  int num_of_blocks = (fa/b);
  PARALLEL_ROUTINE_macro(fa, p, num_of_blocks, b, powers);
 };
};
```

Los parámetros de entrada `fa`, `p`, `b`, y `powers` representan: el tamaño del problema, el número procesadores, el tamaño de bloque y la lista de las potencias de los procesadores, respectivamente. En el apartado `link` se representa los costes de comunicación de la aplicación referente al esquema paralelo implementado. En nuestro caso se representan los costes de la comunicaciones punto a punto desde el Maestro a los Esclavos, considerando que estas comunicaciones son iguales y constantes a lo largo del algoritmo.

La aplicación desarrollada en HeteroMPI (Algoritmo 10) comienza con la directiva de inicialización del sistema, utilizando la función:

```
HMPI_Init(int argc, char **argv)
```

Tras la inicialización se invoca la rutina para obtener el identificador para el grupo de procesos MPI.

```
HMPI_Group_create(HMPI_Group gid, const HMPI_Model perf_model,
const mparam)
```

El siguiente paso consiste en obtener las características de rendimiento de los distintos procesadores heterogéneos. Para realizar dicha tarea HeteroMPI permite utilizar una función para predecir el tiempo total del algoritmo en ejecución:

```
HMPI_Timeof(const HMPI_Model perf_model, const mparam)
```

Esta característica permite que el programador escriba un programa paralelo que puede tomar diversas decisiones dependiendo del problema a solucionar. Al finalizar la ejecución se utiliza la siguiente función para terminar los grupos de procesos creados:

```
HMPI_Finalize()
```

El Algoritmo 10 muestra la estructura del programa principal que implementa el algoritmo paralelo heterogéneo para el cálculo de los valores propios.

Algoritmo 10 Esquema HeteroMPI para Cálculo de Valores Propios.

```
HMPI_Init(&argc,&argv);
HMPI_Group gid;
        .
        .
MPC_Update_Processors_info(dpowers);

if (HMPI_Is_host()){
   for (i = start; i <= fa; i *= mult){
        .
        .
     HMPI_Group_create(
            &gid,
            &MPC_NetType_eigenvalue,
             model_params,
             model_count
     );
        .
        .
        time = HMPI_Timeof(
                 &MPC_NetType_eigenvalue,
                  model_params,
                  model_count
       );

       if (time < mintime){
          optimal_b = i;
          mintime = time;
       }
   }
}
        .
        .

PARALLEL_ROUTINE(fa,num_procesadores,myid,namelen,
nombre_procesador, fa/optimal_b, optimal_b, powers));
        .
        .
HMPI_Group_free(&gid);
}

HMPI_Finalize(0);
```

3.6. Análisis Experimental

En esta sección evaluamos los algoritmos secuenciales y paralelos propuestos. En primer lugar describimos brevemente las características de los algoritmos probados, y a continuación describimos las matrices tridiagonales simétricas utilizadas para las pruebas.

3.6.1. Descripción de los Algoritmos Probados

En esta sección comparamos las prestaciones de la implementación desarrollada con 4 algoritmos para el cálculo de los valores propios de una matriz tridiagonal simétrica. Estos algoritmos utilizan estrategias de tipo Divide y Vencerás para rutinas LAPACK y ScaLAPACK con distintas características.

LAPACK

- ◇ ***dsyevd*** esta rutina utiliza para el cálculo de las raíces el algoritmo iterativo QR.
- ◇ ***dstevd*** esta rutina utiliza modificaciones de rango uno durante la fase de división y aplica la iteración de Laguerre para calcular los ceros del polinomio.
- ◇ ***dstegr*** esta rutina utiliza modificaciones de rango uno con el uso de interpolación racional para la aproximación de los valores propios.

ScaLAPACK

- ◇ ***pdsyevd*** esta rutina calcula todos los valores propios haciendo uso de un esquema paralelo Divide y Vencerás.

3.6.2. Descripción de las Matrices de Pruebas

Una de las características del algoritmo Divide y Vencerás desarrollado es que es dependiente del problema [56], esto es, su comportamiento depende de las matrices cuyos valores propios se quiere calcular. Los resultados experimentales obtenidos demuestran que los algoritmos probados son directamente dependientes de la distribución de los valores propios a lo largo del espectro. El análisis experimental de los algoritmos se basó en la elección de un conjunto de matrices de prueba con diferentes distribuciones de los valores propios, que se muestra en la Tabla 3.2:

Tabla 3.2: Experimentación con diferentes Distribuciones de Valores Propios.

Tipo	Valores Propios
1	Generados Aleatoriamente
2	Distribuidos Formando *Clusters* de Valores Propios

3.6.3. Resultados Experimentales

En esta sección mostramos los resultados experimentales obtenidos utilizando la herramienta HeteroMPI. El cluster de prueba utilizado para esta experimentación fue **ROSEBUD**, descrito anteriormente, un cluster formado por un conjunto de máquinas de características distintas, siendo posible explotar al máximo la heterogeneidad de la plataforma a través de la implementación desarrollada.

Pruebas para el Cálculo del Tamaño de Bloque Óptimo

El experimento consistió en utilizar la rutina *HMPI_Timeof* [25] para predecir el tamaño de bloque óptimo de forma automática, para un valor determinado de tamaño del problema. Lo que se hace es seleccionar el valor más pequeño del tiempo de ejecución en relación al tamaño de bloque hallado. La Figura 3.3 nos muestra el comportamiento de la aplicación en la búsqueda del tamaño de bloque

ideal para un tamaño del problema de 1600. Se utilizó una topología formada por 4 nodos (**rosebud01** a **rosebud04**) pertenecientes al cluster **ROSEBUD**.

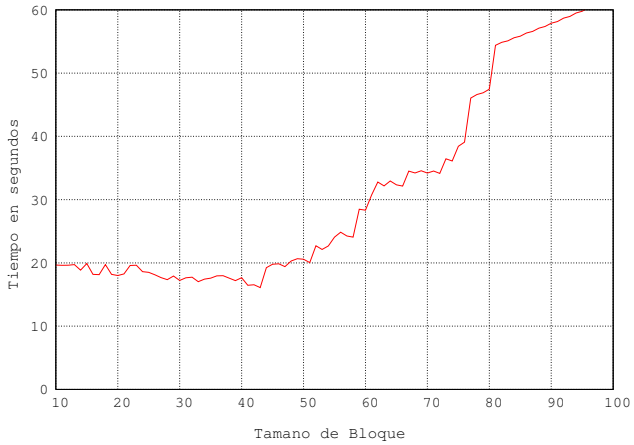

Figura 3.3: Cálculo del Tamaño de Bloque Óptimo.

Comparación entre la Estrategia Paralela Heterogénea DV para el Cálculo de Valores Propios y las Rutinas Secuenciales LAPACK

En este apartado mostramos el comportamiento del tiempo de ejecución en segundos de los algoritmos DV desarrollados con relación a las matrices de pruebas utilizadas. El experimento se fundamenta en comparar la Estrategia Paralela DV utilizando HeteroMPI con las rutinas LAPACK que se basaban en el mismo paradigma.

La experimentación consistió en variar el tamaño del problema para cada estrategia, viendo el comportamiento del tiempo de ejecución utilizando los parámetros óptimos seleccionados para cada caso, utilizando los nodos de la plataforma **ROSEBUD** con características heterogéneas. Para las implementaciones secuenciales se utilizó el nodo **rosebud06**, debido su gran capacidad de cómputo en relación a los otros nodos. En cuanto a la implementación paralela heterogénea se utilizaron los 6 nodos, con 22 procesadores en su total. Los resultados mostrados en las Tablas 3.3 y 3.4 representan los tiempos de ejecución en segundos para los algoritmos probados, para los casos que se calculan los valores propios variando las

dimensiones entre los tipos de matrices de pruebas. Tal y como se ha comentado anteriormente los resultados de todos los algoritmos utilizan métodos Divide y Vencerás para el cálculo de los valores propios, y su comportamiento experimental se relaciona directamente con el tipo de matriz tratada. Comparando con las rutinas secuenciales LAPACK se ha constatado que la rutina *dstevd* tiene mejores prestaciones que *dstegr*, y que, a su vez, son mejores que *dsyevd* para los tipos de matrices experimentados. Pero si comparamos con la implementación utilizando HeteroMPI, vemos que el balanceo de la carga utilizando la asignación en función de las capacidades de procesamiento de las máquinas mejora sensiblemente las prestaciones obtenidas en rangos de experimentación de grandes dimensiones en relación a LAPACK. En la Figura 3.4 se puede observar el comportamiento de las rutinas frente sus respectivos tiempos de ejecución expresados por la Tabla 3.3 para el tipo de matriz de prueba 1.

Tabla 3.3: Comparación entre los Tiempos de Ejecución en segundos de las Estrategias DV para matrices de prueba tipo 1, en un conjunto de máquinas heterogéneas con 22 procesadores en total, en la plataforma **ROSEBUD**.

dimensión	dsyevd	dstevd	dstegr	HeteroMPI
3000×3000	13.67	4.76	9.95	3.41
3500×3500	21.16	4.76	14.64	4.16
4000×4000	31.10	5.76	15.61	4.87
4500×4500	43.67	6.61	20.61	5.87
5000×5000	60.02	9.60	26.54	6.56
5500×5500	78.13	9.65	34.44	7.35
6000×6000	97.78	18.42	35.81	8.47
6500×6500	126.04	19.52	47.87	8.58
7000×7000	155.98	19.89	52.21	9.14
7500×7500	202.53	25.21	60.60	9.51
8000×8000	234.00	29.52	64.70	10.58
8500×8500	291.01	29.58	78.08	11.61
9000×9000	350.29	29.78	84.81	12.32
9500×9500	394.53	31.04	101.99	14.23
10000×10000	441.67	31.18	118.54	14.94

Tabla 3.4: Comparación entre los Tiempos de Ejecución en segundos de las Estrategias DV para matrices de prueba tipo 2, en un conjunto de máquinas heterogéneas con 22 procesadores en total, en la plataforma **ROSEBUD**.

dimensión	dsyevd	dstevd	dstegr	HeteroMPI
3000 × 3000	13.63	9.76	9.85	3.80
3500 × 3500	21.12	9.76	14.64	4.34
4000 × 4000	31.70	9.76	15.13	5.10
4500 × 4500	43.85	9.76	22.19	5.41
5000 × 5000	59.34	9.76	26.47	6.01
5500 × 5500	83.26	19.52	34.50	7.17
6000 × 6000	112.38	19.52	35.19	7.81
6500 × 6500	141.58	19.52	46.04	8.26
7000 × 7000	187.32	19.52	52.16	9.21
7500 × 7500	230.87	29.28	60.09	10.19
8000 × 8000	291.27	29.28	64.08	10.32
8500 × 8500	357.86	29.28	76.81	12.32
9000 × 9000	438.24	29.28	84.14	14.43
9500 × 9500	493.51	39.04	100.19	15.24
10000 × 10000	552.38	39.04	120.48	15.93

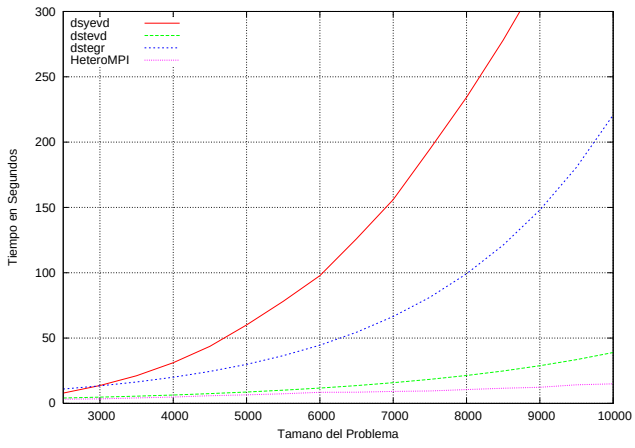

Figura 3.4: Resultados de los Tiempos de Ejecución para la Estrategia DV para Matrices de Pruebas Tipo 1, utilizando 22 procesadores en la Plataforma ROSEBUD.

Comparación entre la Estrategia Paralela Heterogénea DV para el Cálculo de Valores Propios y ScaLAPACK

En este apartado comparamos el comportamiento del tiempo de ejecución en segundos entre los algoritmos: paralelo heterogéneo y la rutina *pdsyevd* ScaLAPACK, que utilizan el paradigma Divide y Vencerás para el cálculo de los valores propios. La experimentación consistió en variar el tamaño del problema para las estrategias DV, viendo el comportamiento del tiempo de ejecución.

Se utilizó una topología con características homogéneas, formada por 4 nodos (**rosebud01** a **rosebud04**), con 6 procesadores en su total, pertenecientes al cluster **ROSEBUD**, formando un grid de procesos 2×3, para el caso del ScaLAPACK, ya que con este formato se consigue las mejores distribuciones de carga con las más bajas prestaciones para el problema a ser resuelto. Una de las limitaciones encontradas para esta experimentación, consistió que la rutina *pdsyevd* asume que la inicialización de los procesos está pensada para sistemas homogéneos, emitiendo una mensaje de error al ser ejecutado en sistemas con características heterogéneas [57].

La inicialización de los grupos de procesos en la implementación paralela heterogénea utilizando HeteroMPI es definida a partir los valores de Mapeo de los Procesos en función de las potencias de cómputo de las máquinas calculadas, aprovechando al máximo la heterogeneidad de cómputo para el grupo de computación. La constitucción del fichero de máquinas para esta experimentación se basa en las potencias relativas calculadas en la Tabla 3.1, expresando el número ideal de procesos a ser ejecutado en cada nodo.

A partir de la Tabla 3.5 y impresos por la Figura 3.5 se puede observar que los valores fueron satisfactorios para la implementación paralela heterogénea utilizando HeteroMPI frente la rutina ScaLAPACK. En la mayor parte de los casos estudiados, variando el tamaño del problema la herramienta consigue obtener valores con menores tiempos de computación.

Tabla 3.5: Comparación entre los Tiempos de Ejecución en segundos de la Estrategia Paralela Heterogénea DV y ScaLAPACK para matrices de prueba tipo 2, en un conjunto de máquinas homogéneas con 6 procesadores en total, en la plataforma **ROSEBUD**.

dimensión	pdsyevd	HeteroMPI	Speedup
3000×3000	8.12	10.12	0.80
3500×3500	12.14	12.70	0.95
4000×4000	13.53	16.27	0.83
4500×4500	15.18	20.72	0.73
5000×5000	20.11	25.60	0.78
5500×5500	30.72	32.54	0.94
6000×6000	52.47	40.71	1.28
6500×6500	77.72	50.75	1.53
7000×7000	106.47	63.45	1.67
7500×7500	138.72	80.32	1.72
8000×8000	174.51	100.87	1.73
8500×8500	213.72	126.31	1.69
9000×9000	256.47	158.15	1.62
9500×9500	302.49	199.12	1.51
10000×10000	352.05	249.36	1.41

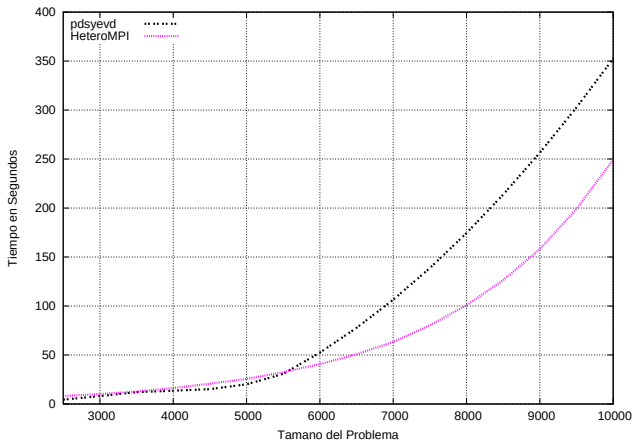

Figura 3.5: Resultados de los Tiempos de Ejecución para las Estrategias DV para Matrices de Pruebas Tipo 2, utilizando 6 procesadores en la Plataforma ROSEBUD.

Comparación entre las Estrategias Paralelas DV para el Cálculo de Valores Propios

En este apartado comparamos el comportamiento del tiempo de ejecución en segundos entre los algoritmos paralelos DV para cálculo de los valores propios. El experimento se basa en analizar el algoritmo paralelo heterogéneo utilizando HeteroMPI siendo inicializando con las cantidades óptimas de procesos por nodo, teniendo como referencia sus respectivas potencias relativas, comparando con la aplicación *standard* homogénea en MPI. Como en la experimentación anterior se varió el tamaño del problema para las estrategias paralelas, viendo el comportamiento del tiempo de ejecución para las matrices de pruebas de tipo 2. Se utilizó en esta experimentación la plataforma **ROSEBUD** con 6 nodos de características heterogéneas con 22 procesadores en total. Los valores experimentales están en la Tabla 3.6 y su representación gráfica en la Figura 3.6. El análisis de los resultados muestra una ganancia de la implementación paralela heterogénea utilizando HeteroMPI con la cantidad óptima de procesos por nodo frente la *standard* homogénea en MPI.

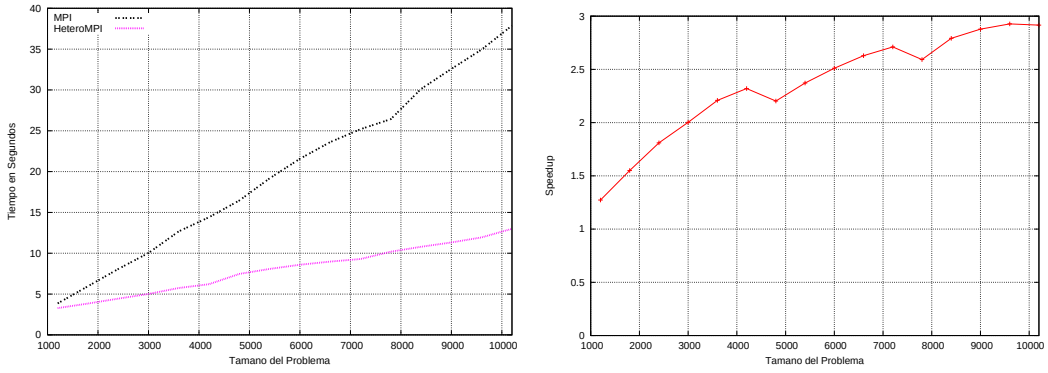

Figura 3.6: Comparación de los Tiempos de Ejecución utilizando HeteroMPI y MPI Homogéneo y Speedup, utilizando 22 procesadores en la Plataforma ROSEBUD.

Tabla 3.6: Comparación entre los Tiempos de Ejecución en segundos de las Estrategias Paralelas DV para matrices de prueba tipo 2, en un conjunto de máquinas heterogéneas con 22 procesadores en total, en la plataforma **ROSEBUD**.

dimensión	Homogéneo	HeteroMPI	Speedup
3000×3000	5.96	3.80	1.56
3500×3500	8.07	4.34	1.85
4000×4000	10.04	5.10	1.96
4500×4500	12.68	5.41	2.34
5000×5000	14.68	6.01	2.44
5500×5500	16.46	7.17	2.29
6000×6000	19.17	7.81	2.45
6500×6500	21.57	8.26	2.61
7000×7000	23.61	9.21	2.56
7500×7500	25.19	10.19	2.47
8000×8000	26.42	10.32	2.56
8500×8500	30.13	12.32	2.44
9000×9000	32.58	14.43	2.25
9500×9500	34.95	15.24	2.29
10000×10000	37.88	15.93	2.37

3.7. Resumen y conclusiones

En este capítulo hemos descrito esquemas paralelos adaptados a plataformas heterogéneas. La herramienta HeteroMPI fue utilizada para el análisis de un algoritmo para el cálculo de valores propios. Los resultados experimentales obtenidos muestran que los algoritmos propuestos son altamente eficientes en tiempo de respuesta y en distribucción de la carga. La comparación de todos los algoritmos implementados demuestra que la utilización de la herramienta ofrece mejores resultados que los algoritmos Divide y Vencerás de las rutinas LAPACK y ScaLAPACK para grandes dimensiones para los tipos de matrices probados. El balanceo de la carga utilizando la asignación en función de las capacidades de procesamiento de las máquinas, mejora sensiblemente las prestaciones obtenidas. Se pretende comparar los algoritmos propuestos con diferentes estrategias de asignación y mapeo de los datos, a fin de modelar la solución óptima para distintos tipos de problema.

Capítulo 4

Conclusiones Finales

En este capítulo se presentan las conclusiones generales obtenidas de esta tesis y se proponen líneas futuras de investigación.

4.1. Conclusiones

El uso de las técnicas de la autooptimización para un esquema Divide y Vencerás se ha estudiado utilizando como ejemplo algoritmos de ordenación por mezcla, ordenación rápida y algoritmos para cálculo de valores propios de matrices tridiagonales simétricas. A corto plazo se trabajará con otros problemas típicos de Divide y Vencerás: multiplicación rápida de enteros largos, multiplicación rápida de matrices, etc, y con otros más elaborados como algoritmos para resolución de sistemas de ecuaciones tridiagonales por bloques. Se pretende validar la metodología de autooptimización para un rango amplio de esquemas paralelos de Divide y Vencerás.

A largo plazo, se podría adecuar la técnica utilizada para el desarrollo de librerías con capacidad de optimización automática a otros esquemas heterogéneos, variando las estrategias de asignación de trabajos a procesadores.

Por otra parte, puede ser de interés aplicar la experiencia adquirida en esquemas de desarrollo de esqueletos paralelos, de manera que el usuario que necesite resolver un problema en paralelo pueda programar en secuencial funciones particulares del esqueleto, y la ejecución paralela será transparente al usuario, al no serle necesario programar en paralelo y no ser necesario tener conocimientos de paralelismo para poder obtener tiempos de ejecución reducidos.

Las aportaciones principales de esta tesis se inscriben en el campo de la computación paralela, y más en concreto en el desarrollo y utilización de técnicas de autooptimización en sistemas paralelos de esquemas algorítmicos paralelos Divide y Vencerás. Se puede destacar que:

- ◇ Se propone un modelo para sistemas paralelos de computación numérica. Estos sistemas están formados por la unión de la arquitectura hardware junto al software básico de resolución numérica instalado.
- ◇ Tomando como arquitectura base el modelo del sistema propuesto anteriormente, se plantea un modelo teórico experimental del tiempo de ejecución de las rutina desarrolladas.

- ◇ Se plantea una arquitectura software que confiere a cada rutina de resolución numérica la capacidad de adaptarse automáticamente a las condiciones de trabajo, siendo el proceso de ajuste transparente al usuario.
- ◇ Se propone el desarrollo de esquemas de resolución numérica de altas prestaciones, que se utilizan en la resolución de problemas científicos reales.

Como resultado de este trabajo se han realizados las siguientes contribuciones:

- ▷ M. Boratto, D. Giménez and A. M. Vidal, ***Automatic Parametrization on Divide and Conquer Algorithms***, International Congress of Mathematicians, Madrid, España, Agosto, 2006, pp 495-496.
- ▷ D. Giménez, J. Cuenca, J. P. Martínez, J. M. Beltrán, M. Boratto and A. M. Vidal, ***Parametrización de esquemas algorítmicos paralelos para autooptimización***, XVII Jornadas de Paralelismo, Albacete, España, Septiembre, 2006.
- ▷ M. Boratto, D. Giménez and A. M. Vidal, ***Autooptimización en algoritmos numéricos Divide y Vencerás***, Reunión sobre Optimización de Rutinas Paralelas y Aplicaciones, Murcia, España, 12-13 junio, 2007.

4.2. Propuestas Futuras

La continuación de este trabajo podría dar lugar a la realización de la tesis doctoral sobre "Autooptimización en Esquemas Paralelos Divide y Vencerás". La tesis está previsto realizarla bajo la supervisión de los Doctores D. Antonio M. Vidal Maciá, profesor del Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia y D. Domingo Giménez Cánovas, profesor del Departamento de Informática y Sistemas de la Universidad de Murcia. El organigrama de la secuencia los trabajos futuros pueden organizarse según la Tabla 4.1:

Esquemas Algorítmicos	Secuencial	Homogéneo	Heterogéneo	Híbrido	Grid
Métodos de Ordenación	✓	✓			
Sist. Ecuaciones	✓	✓			
Valores Propios	✓	✓	✓		
Multip. Matrices					
⋮					
Esquema General DV	✓	✓	✓	✓	✓

Tabla 4.1: Líneas de Investigación Futuras.

La tabla anterior representa el estudios de diversos esquemas algorítmicos basados en Divide y Vencerás estudiados y propuestos, contrastados con los diferentes paradigmas de programación. La idea es recorrer esta tabla abarcando el mayor número de casos posibles, con la finalidad de obtener un modelo de carácter general para la parametrización de esquemas paralelos basados en el paradigma Divides y Vencerás.

Bibliografía

[1] E. Brewer. *Portable High Performance Supercomputing: High Level Platform Dependent Optimization*. PhD thesis, Massachusetts Institute of Technology, 1994.

[2] M. Frigo and S. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, volume 3, pages 1381–1384, Seattle, Washington, May 1998.

[3] Z. Chen, J. Dongarra, P. Luszczek, and K. Roche. Self adapting software for numerical linear algebra and LAPACK for clusters. *Parallel Computing*, 29(11-12):1723–1743, 2003.

[4] M. Kandemir, A. Choudhary, N. Shenoy, P. Banerjee, and J. Ramanujam. A linear algebra framework for automatic determination of optimal data layouts. *IEEE Transactions on Parallel and Distributed Systems*, 10(2):115–123, 1999.

[5] Jim Demmel and Jack Dongarra. LAPACK 2005 prospectus: Reliable and scalable software for linear algebra computations on high end computers. LAPACK Working Note 164, February 2005. UT-CS-05-546, February 2005.

[6] A. Faraj and X. Yuan. Automatic generation and tuning of MPI collective communication routines. In *ICS '05: Proceedings of the 19th annual international conference on supercomputing*, pages 393–402, New York, NY, USA, 2005. ACM Press.

[7] J. Cuenca, D. Giménez, and J. González. Architecture of an automatically tuned linear algebra library. *Parallel Computing*, 30(2):187–210, 2004.

[8] K. Kise T. Katagiri and H. Honda. RAO-SS: A prototype of run-time autotuning facility for sparse direct solvers. In *International Symposium on Parallel Architectures*, pages 1–10, June 2005.

[9] J. Cuenca, D. Giménez, and J. P. Martínez. Heuristics for work distribution of a homogeneous parallel dynamic programming scheme on heterogeneous systems. *Parallel Comput.*, 31(7):711–735, 2005.

[10] J. Cuenca, D. Giménez, and J. González. Modelling the behaviour of linear algebra algorithms with message passing. pages 282–289. IEEE Computer Society, February 2001.

[11] P. Alberti, P. Alonso, A. Vidal, J. Cuenca, and D. Giménez. Designing polylibraries to speed up parallel computations. *International Journal of High Performance Computing Applications*, 1(1/2/3):75–84, 2004.

[12] D. Giménez and J. P. Martínez. Automatic optimization in parallel dynamic programming schemes. In *Proceedings of VECPAR2004*, pages 639–649, 2004.

[13] J. P. Martínez, F. Almeida, and D. Giménez. Mapping in heterogeneous systems with heuristical methods. Workshop on state-of-the-art in Scientific and Parallel Computing, Sweden, June 2006. Lecture Notes in Computing Science (LNCS).

[14] D. Giménez, J. Cuenca, J. P. Martínez, J. M. Beltrán, M. Boratto, and A. M. Vidal. Parametrización de esquemas algorítmicos paralelos para autooptimización. In *XVII Jornadas de Paralelismo*, September 2006.

[15] J. M. Beltrán. Autooptimización en esquemas paralelos de backtracking y branch and bound: Esquema del problema de la mochila 0/1. Trabajo de Inicialización a la Investigación - Universidad de Murcia, 2006.

[16] J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN basic linear algebra subroutines. *ACM Transactions on Mathematical Software*, 14:1–17, 1988.

[17] ATLAS. Automatically Tuned Linear Algebra Software. Available in: http://math-atlas.sourceforge.net/.

[18] R. C. Whaley, A. Petitet, and J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.

[19] J. Cuenca. *Optimización Automática de Software Paralelo de Álgebra Lineal*. PhD thesis, Universidad de Murcia, 2004.

[20] A. Lastovetsky. *Parallel Computing on Heterogeneous Networks*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[21] mpC. The mpC Parallel Programming Language. Available in: http://hcl.ucd.ie/Projects/mpC/.

[22] HeteroMPI. Heterogeneous MPI. Available in: http://hcl.ucd.ie/Projects/HeteroMPI/.

[23] D. Valencia, A. Lastovetsky, and A. Plaza. Design and implementation of a parallel heterogeneous algorithm for hyperspectral image analysis using HeteroMPI. pages 301–308, Timisoara, Romania, July 2006. IEEE Computer Society Press.

[24] A. Lastovetsky and R. Reddy. HeteroMPI: Towards a message-passing library for heterogeneous networks of computers. *Journal of Parallel and Distributed Computing*, 66:197–220, 2006.

[25] R. Reddy. *HMPI: A Message-Passing Library for Heterogeneous Networks of Computers*. PhD thesis, Computer Science Department, University College Dublin, June 2005.

[26] B. S. Andersen, F. G. Gustavson, A. Karaivanov, M. Marinova, J. Wasniewski, and P. Y. Yalamov. LAWRA: Linear algebra with recursive algorithms. In *PARA*, pages 38–51, 2000.

[27] J. Demmel, J. Dongarra, B. Parlett, W. Kahan, M. Gu, D. Bindel, Y. Hida, X. Li, O.i Marques, E. J. Riedy, C. Voemel, J. Langou, P. Luszczek, J. Kurzak, A. Buttari, J. Langou, and S. Tomov. Prospectus for the next LAPACK and ScaLAPACK libraries. LAPACK Working Note 181, February 2007.

[28] R. Reddy and A. Lastovetsky. HeteroMPI + ScaLAPACK: Towards a ScaLAPACK (Dense Linear Solvers) on Heterogeneous Networks of Computers. volume 4297, pages 242–253, Bangalore, India, 18-21 Dec 2006 2006. Springer.

[29] A. Lastovetsky and R. Reddy. Data partitioning with a functional performance model of heterogeneous processors. *International Journal of High Performance Computing Applications*, 21:76–90, 2007.

[30] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 2001.

[31] M. Boratto, D. Giménez, and A.Vidal. Automatic parametrization on Divide-and-Conquer Algorithms. In *International Congress of Mathematicians*, pages 495–496, 2006.

[32] A. Rojas and C. León. MALLBA: Paradigma Divide y Vencerás. Documento de trabajo interno (DT-01-1), Universidad de La Laguna, Canarias, Spain, 2001.

[33] J. Rutter. A serial implementation of Cuppen's divide and conquer algorithm for the symmetric eigenvalue problem. LAPACK Working Note 69, March 1994. UT-CS-94-225, March 1994.

[34] Q. Du, M. Jin, T. Li, and Z. Zeng. The quasi-Laguerre interation. *Math. Computing*, 66:345–361, 1997.

[35] GRYCAP. Grupo de Redes y Computación de Altas Prestaciones (Universidad Politénica de Valencia). Available in: http://www.grycap.upv.es/.

[36] ASIC. Área de Sistemas de Información y Comunicaciones (Universidad Politénica de Valencia). Available in: http://www.asic.upv.es/.

[37] HCL. Heterogeneous Computing Laboratory (University College Dublin). Available in: http://hcl.ucd.ie/.

[38] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. J. Dongarra, J. Du Croz, A. Grenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1995.

[39] Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*. Univ. of Tennessee, Knoxville, Tennessee, 1995.

[40] P. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, 1997.

[41] A. Vidal and J. L. Gómez. *Introdución a la programación en MPI*. Universidad Politécnica de Valencia (UPV), 2000.

[42] MPICH. Implementation Message Passing Interface. Available in: http://www.mpi.org/mpich/.

[43] J. Dongarra and R. Clint Whaley. A user's guide to the BLACS v1.0. Technical Report CS-95-292, Computer Science Dept. University of Tennesee, 1995.

[44] J. Choi, J. J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. Clint Whaley. A proposal for a set of parallel basic linear algebra subprograms. Technical Report CS-95-292, Computer Science Dept. University of Tennesee, May. 1995.

[45] 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 User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[46] GAM. Grupo de Aplicaciones de las Microondas (Universidad Politénica de Valencia). Available in: http://www.iteam.es/.

[47] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Baltimore, MD, USA, second edition, 1989.

[48] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing: design and analysis of algorithms*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.

[49] R. Xiong. *Design of parallel mergesort and quicksort algorithms*. PhD thesis, New York, NY, USA, 1990.

[50] D. Giménez, J. Cervera, G. García, and N. Marín. *Algoritmos y Estructuras de Datos*, volume II. Colección Texto-Guía de la Universidad de Murcia, 2003.

[51] R. Li. Solving secular equations stably and efficiently. Technical Report UCB/CSD-94-851, EECS Department, University of California, Berkeley, 1994.

[52] J. M. Badía. *Algoritmos Paralelos para Cálculo de Valores Propios de Matrices Estructuradas*. PhD thesis, Universidad Politécnica de Valencia, 1996.

[53] A. Kalinov and S. Klimov. Optimal mapping of a parallel application processes onto heterogeneous platform. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, page 123.2, Washington, DC, USA, 2005. IEEE Computer Society.

[54] A. Kalinov and A. Lastovetsky. Heterogeneous distribution of computations while solving linear algebra problems on networks of heterogeneous computers. In *HPCN Europe '99: Proceedings of the 7th International Conference on High-Performance Computing and Networking*, pages 191–200, London, UK, 1999. Springer-Verlag.

[55] H. Siegel and S. Ali. Techniques for mapping tasks to machines in heterogeneous computing systems. 46(8):627–639, may 2000.

[56] J. M. Badía and A. M. Vidal. *Cálculo de los valores propios de matrices tridiagonales simétricas mediante la interación de Laguerre*, volume 16, pages 44–48. Revista Internacional de Métodos Numericos para Cálculo y Diseño en Ingenieria, 2000.

[57] ScaLAPACK. Routine ScaLAPACK for Eigenvalues Problem (pdsyevd). Available in: www.netlib.org/scalapack/double/pdsyevd.f.

[58] J. H. Wilkinson, editor. *The algebraic eigenvalue problem.* Oxford University Press, Inc., New York, NY, USA, 1988.

Apéndice A

Código Fuente de Parte de la Rutina HeteroMPI

El principal objetivo de HeteroMPI es automatizar y optimizar la selección de un grupo de procesos, de forma que puedan ejecutar un algoritmo heterogéneo de la forma más eficiente posible. El modelo propuesto por la herramienta consigue automáticamente adaptar la red abstracta a una red real, ajustando dinámicamente el modelo de rendimiento a parámetros específicos de la red, tales como la potencia de cálculo de los procesadores o la capacidad de los enlaces de comunicación. En esta sección presentamos una pequeña parte de la implementación de la rutina paralela desarrollada para un ambiente heterogéneo mostrando didácticamente la estructuración y planteamiento del código fuente desarrollado.

```
/*****************************************************************
* TITULO:     HETEROMPI                                          *
* FECHA:      27/09/2007 (V5.0)                                  *
* PROGRAMADOR:Murilo Boratto [mdocarmo@dsic.upv.es]              *
* COMENTARIO: Código para el Cálculo de Valores Propios          *
* COMPILAR:   make -f makefile.rosebud                           *
* EJECUTAR:   hmpirun principal.exe -- [size of problem][powers]*
*****************************************************************/

main(int argc, char **argv){

/*Inicializa HeteroMPI*/
  HMPI_Init(&argc,&argv);
  HMPI_Group gid;
  MPI_Comm_size(HMPI_COMM_WORLD,&num_procesadores);
  MPI_Comm_rank(HMPI_COMM_WORLD,&myid);
  MPI_Get_processor_name(nombre_procesador, &namelen);
/*Parámetros por linea de comando*/
  fa = atoi(argv[1]);

  P0 = atoi(argv[2]);
  P1 = atoi(argv[3]);
  P2 = atoi(argv[4]);
  P3 = atoi(argv[5]);
  P4 = atoi(argv[6]);
  P5 = atoi(argv[7]);

  powers[0] = dpowers[0] = P0;
  powers[1] = dpowers[1] = P1;
  powers[2] = dpowers[2] = P2;
  powers[3] = dpowers[3] = P3;
  powers[4] = dpowers[3] = P4;
  powers[5] = dpowers[3] = P5;

  MPC_Update_Processors_info(1.0/dpowers[myid]);
```

```
if (HMPI_Is_host()){
    for (i = 16; i <= fa; i *= 2){
      int model_count = 7, model_params[7], d[num_procesadores];

 model_params[0] = fa;
 model_params[1] = num_procesadores;
 model_params[2] = i;

 HMPI_Partition_set(num_procesadores, 1, dpowers, NULL,
 NULL, fa/i, NULL, 0, 0, -1, NULL, NULL, d);

 model_params[3] = d[0]; model_params[4] = d[1];
 model_params[5] = d[2]; model_params[6] = d[3];

 time = HMPI_Timeof(&MPC_NetType_eigenvalue, model_params,
        model_count);

     if (time < mintime){
      optimal_b = i;
      mintime = time;
      }
    }
 }

 MPI_Bcast(&optimal_b, 1, MPI_INT, MASTER, HMPI_COMM_WORLD);

 if (HMPI_Is_host()){
  int model_count = 7;
  int model_params[7];
  int d[num_procesadores];

  model_params[0] = fa;
  model_params[1] = num_procesadores;
  model_params[2] = optimal_b;
```

```
 HMPI_Partition_set(num_procesadores, 1, dpowers,
 NULL, NULL, fa/optimal_b, NULL, 0, 0, -1, NULL, NULL, d);
 model_params[3] = d[0]; model_params[4] = d[1];
 model_params[5] = d[2]; model_params[6] = d[3];
 HMPI_Group_create(&gid, &MPC_NetType_eigenvalue,
 model_params, model_count);
}
if (HMPI_Is_free(){
  HMPI_Group_create(
      &gid,
      &MPC_NetType_eigenvalue,
      NULL,
      0
  );
}
if (HMPI_Is_member(&gid)){
  algocomm  = *(MPI_Comm*)HMPI_Get_comm(&gid);

  MPI_Comm_rank(algocomm, &myid);
  {
     int d[num_procesadores];

     HMPI_Partition_set(num_procesadores, 1, dpowers,
         NULL, NULL, fa/optimal_b, NULL, 0, 0, -1, NULL,
         NULL, d);

     PARALLEL_ROUTINE( fa, num_procesadores,myid, namelen,
         nombre_procesador, fa/optimal_b, optimal_b, d[0],
         d[1], d[2], d[3]);
  }
  HMPI_Group_free(&gid);
 }
 HMPI_Finalize(0);
}
```

Apéndice B

Generación de las Matrices de Pruebas para el Cálculo de los Valores Propios

Teniendo en cuenta que uno de los factores que pueden influir en las prestaciones de los métodos para cálculo de los valores propios es la distribucción de sus valores, utilizamos un conjunto de matrices de pruebas que presenten distintas distribuciones.

Se ha utilizado matrices en el formato simétrico tridiagonal con valores reales de doble precisión. En primero lugar utilizamos matrices con valores generado aleatoriamente y en segundo lugar matrices cuyos se han generados formando clusters de valores propios (matrices de Wilkinson [58]).

Para generar las matrices tridiagonales de prueba con los valores propios distribuidos según los 3 tipos determinados anteriormente, se utilizó una operación algebraica matricial, a una matriz diagonal, denominada Rotación de Givens [47]. Esta operación no modifica los valores absolutos de los autovalores de la matriz. Se puede ejemplificar la generación de las matrices de pruebas a partir de los siguientes pasos en el ejemplo a seguir:

Paso 1: Definición de la Matriz Diagonal

Se construye la matriz diagonal D, donde se define la districción de los valores propios de la matriz de prueba (para este caso se tería una distribución uniforme).

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

Paso 2: Cálculo de los Valores Trigonométricos

Consideramos α un valor aleatorio para los cálculos de c y s (para el ejemplo consideramos su valor igual a $\frac{\Pi}{7}$).

$$\begin{cases} c = cos(\alpha) = cos(\frac{\Pi}{7}) = 0{,}901 \\ s = sin(\alpha) = sin(\frac{\Pi}{7}) = 0{,}434 \end{cases}$$

Paso 3: Definición de la Matriz Ortogonal

Se construye una matriz ortogonal Q a partir de los valores trigonométricos calculados anteriormente.

$$Q = \begin{bmatrix} c & s & 0 & 0 & 0 \\ -s & c & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0{,}901 & 0{,}434 & 0 & 0 & 0 \\ -0{,}434 & 0{,}901 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Paso 4: Se Aplica la Operación Matricial

Esta operación matricial no cambia los valores propios de la matriz diagonal D.

$$A = Q * D * Q^T$$

$$A = \begin{bmatrix} 1{,}188 & 0{,}391 & 0 & 0 & 0 \\ 0{,}391 & 1{,}812 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

Se puede comprobar que los valores propios de la matriz tridiagonal A de prueba son los mismos de la matriz diagonal D.

$$eigen(D) = eigen(A) = \{1, 2, 3, 4, 5\}$$