Universitat Politècnica de València
Departamento de Sistemas Informáticos y Computación
Doctorado en Informática

Modelos Paralelos para la Resolución de Problemas de Ingeniería Agrícola

Tesis Doctoral

presentada por

Murilo Boratto

dirigida por

Dr. Pedro Alonso
Dr. Domingo Giménez

Valencia, Octubre de 2014

"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 modelos computacionales en arquitecturas paralelas heterogéneas para la resolución de problemas aplicados. En la tesis abordamos una serie de problemas que están relacionados con la aplicación de la tecnología en el ámbito de las explotaciones agrícolas y comprenden: la representación del relieve, el manejo de información climática como la temperatura, y la gestión de recursos hídricos. El estudio y la solución a estos problemas en el área en la que se han estudiado tienen un amplio impacto económico y medioambiental. Los problemas basan su formulación en un modelo matemático cuya solución es costosa desde el punto de vista computacional, siendo incluso a veces inviable. La tesis consiste en implementar algoritmos paralelos rápidos y eficientes que resuelven el problema matemático asociado a estos problemas en nodos multicore y multi-GPU. También se estudia, propone y aplican técnicas que permiten a las rutinas diseñadas adaptarse automáticamente a las características del sistema paralelo donde van a ser instaladas y ejecutadas con el objeto de obtener la versión más cercana posible a la óptima a un bajo coste. El objetivo es proporcionar un software a los usuarios que sea portable, pero a la vez, capaz de ejecutarse eficientemente en la ordenador donde se esté trabajando, independientemente de las características de la arquitectura y de los conocimientos que el usuario pueda tener sobre dicha arquitectura.

Palabras Clave: Computación de altas prestaciones, computación paralela, autooptimización, arquitectura paralela heterogénea, multicore, multi-GPU.

Abstract

This work is in the field of parallel computing and, in particular, in the development and use of computational models in heterogeneous parallel architectures to solve applied problems. We address a number of problems that are related to the application of technology in the field of agricultural engineering. The problems are: landform representation, processing climate information (for example, temperature), and water resources management. The study and the solution of these problems in the geographical area in question may suppose an important economic and environmental impact. The problems are formulated in a mathematical model whose solution is computationally costly, and sometimes the solution in a reasonable time is not possible. Our work deals with the implementation of fast and efficient parallel algorithms to solve the associated mathematical problems in multicore and multi-GPU nodes. In addition, some techniques are proposed, studied and implemented to automatically adapt the routines to the parallel system where they are installed, with the aim of obtaining executions close to the optimum and at a low computational cost. The objective is to provide the users with portable software able to run efficiently on the target computer, regardless of the characteristics of the architecture and the user knowledge of it.

Keywords: High performance computing, parallel computing, autooptimization, heterogeneous parallel architecture, multicore, multi-GPU.

Resum

Aquest treball s'inscriu en el camp de la computació paral.lela i, més en concret, en el desenvolupament i utilització de models computacionals en arquitectures paral·leles heterogènies per a la resolució de problemes aplicats. En la tesi abordem una sèrie de problemes que estan relacionats amb l'aplicació de la tecnologia en l'àmbit de les explotacions agrícoles i comprenen: la representació del relleu, el maneig d'informació climàtica com la temperatura, i la gestió de recursos hídrics. L'estudi i la solució a aquests problemes en l'àrea en la qual s'han estudiat tenen un ampli impacte econòmic i mediambiental. Els problemes basen la seva formulació en un model matemàtic la solució del qual és costosa des del punt de vista computacional, sent fins i tot de vegades inviable. La tesi consisteix a implementar algorismes paral·lels ràpids i eficients que resolen el problema matemàtic associat a aquests problemes en nodes multicore i multi-GPU. També s'estudia, proposa i apliquen tècniques que permeten a les rutines dissenyades adaptar-se automàticament a les característiques del sistema paral·lel on van a ser instal·lades i executades amb l'objecte d'obtenir la versió més propera possible a l'òptima a un baix cost. L'objectiu és proporcionar un programari als usuaris que sigui portable, però alhora, capaç d'executar-se eficientment en l'ordinador on s'estigui treballant, independentment de les característiques de l'arquitectura i dels coneixements que l'usuari pugui tenir sobre aquesta arquitectura.

Paraules clau: Computació d'altes prestacions, computació paral·lela. autooptimització, arquitectures paral·leles heterogènies, multicore, multi-GPU.

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 Pedro Alonso 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 investigaciones en 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: Antonio Vidal, Victor Manuel García, Miguel Óscar Bernabeu, José Juan López, Carla Ramiro, Danyel León y Marcos Barreto. 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 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, Fernando ... que me ayudaron siempre en todos los momentos.

- ◇ A mis amigos Doctores, que me ensenaron que se podía ser Doctor sin tener que estudiar medicina.
- ◇ Al Profesor Dr. Josemar Rodriguez y al grupo de investigacíon al cual pertenezco, por la inolvidable ayuda.
- ◇ A Paula "meu amor".
- ◇ A mis amigos: Manuela, Cris, Evandro, Cybele, Ana Paula, Tiburcio, Drica, Ildima, Fernanda, Gisa y Karol por estar siempre en mi vida.
- ◇ Al Profesor Brauliro Leal por su gran ayuda.
- ◇ A mi familia y al resto de personas que me han ayudado a realizar este trabajo. GRACIAS A TODOS.

Murilo Boratto
Octubre 2014

Índice General

Índice de Figuras

Abreviaturas y siglas

API	Application Programming Interface
ATLAS	Automatically Tuned Linear Algebra Software
BLAS	Basic linear algebra subprograms
CUBLAS	CUDA Basic Linear Algebra Subroutines
CUDA	Compute Unified Device Architecture
CUFFT	CUDA Fast Fourier Transform
DTM	Digital Terrain Model
FFT	Fast Fourier Transforms
FLOPS	Floating Point Operations Per Second
GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
HeteroMPI	Heterogeneous MPI
HeteroScaLAPACK	Heterogeneous ScaLAPACK
INMET	Instituto Nacional de Meteorología Brasileño
IPD	Inverso de la potencia de la distancia
IQD	Inverso del cuadrado de la distancia
LAPACK	Linear Algebra Package
LAWRA	Linear Algebra With Recursive Algorithms
MPI	Message Passing Interface
MAE	Error absoluto promedio
OpenMP	Open Multi-Processing
ScaLAPACK	Scalable Linear Algebra Package
UNIVASF	Universidade do Vale do Río São Francisco

Planteamiento y ámbito del trabajo 1

En este capítulo describimos el contexto en el que ha sido realizado este trabajo. Parte de la motivación que ha conducido a la realización del mismo radica en este contexto. A partir de ambos: contexto y motivación, explicamos el objetivo de la tesis. Repasamos también el estado del arte, resumimos las principales aportaciones y presentamos el resto del documento.

1.1 Contexto y objetivo de la tesis

A continuación mostramos, en los puntos siguientes, el contexto que enmarca a esta tesis. Tres aspectos diferentes pero que confluyen en un objetivo concreto que se especifica al final de la sección.

El área agrícola del Rio São Francisco

Entre las motivaciones existentes para realizar esta tesis queremos mencionar en primer lugar la búsqueda de respuestas a problemas reales existentes en una importante área de Brasil. El trabajo comenzó a gestarse en el Centro de Computación Agrícola de la UNIVASF (Universidade do Vale do Rio São Francisco), donde se estudian problemas de ingeniería agrícola ta-

les como el diseño de las plantaciones para optimizar los recursos hídricos, la logística de almacenamiento de la producción, o la minimización de los efectos de la erosión. La universidad está localizada en una región agrícola conocida con el nombre de *Valle del Rio São Francisco*.

El área del Valle del Rio São Francisco destaca como una de las más grandes comarcas vinícolas y frutícolas de Brasil, cuya producción se dedica casi exclusivamente a la exportación. La región bajo estudio tiene un formato de círculo de alrededor de 250 km de radio, con centro muy próximo a la ciudad de Petrolina, uno de los municipios de la provincia de Pernambuco más poblados. La cuenca del Rio São Francisco tiene una extensión total de 2863 km, una superficie media de 640000 km^2, y un caudal medio de 2943 m^3/s. Los valores de precipitación para el periodo entre 2010 y 2014 oscilaron alrededor de los 2200 mm anuales. Es un río de gran importancia económica, social y cultural para las provincias que recorre. Según la clasificación climática, el clima del área se presenta como tropical semi-árido, tipo seco y templado en la parte norte y semi-árido estepario cálido en la parte sur, caracterizado por la escasez e irregularidad de las precipitaciones con lluvias en verano y fuerte evaporación a consecuencia de las altas temperaturas. De hecho, la mayor parte de esta región pertenece al territorio brasileño conocido como *Semi-árido Nordestino*.

El trabajo realizado en esta tesis pretende contribuir en cierta medida a solucionar los problemas que se estudian en el Centro de Computación Agrícola de la UNIVASF, problemas que hoy más que nunca preocupan al gobierno brasileño, pues la deforestación y otras actividades humanas descontroladas están modificando seriamente el clima de la zona y afectando por consiguiente a la producción agrícola y su futuro económico.

El modelado matemático de problemas científicos

El modelado de problemas científicos aplicados es de gran importancia para entender, predecir y controlar los procesos físicos que tienen lugar en un determinado campo de la ciencia. La dificultad de realizar ensayos de laboratorio, así como el coste económico de llevar a cabo mediciones experimentales en campo, hacen del modelado de problemas científicos aplicados una herramienta muy útil. El modelado numérico presenta, además, la ventaja de poder estudiar el impacto que puede provocar una futura actuación de la ingeniería, y permite evaluar diferentes escenarios para compararlos. El coste temporal y económico de las soluciones numéricas puede ser muy

bajo en comparación. Pero la creciente necesidad de potencia computacional de las simulaciones hace necesario avanzar en la búsqueda de técnicas que hagan disminuir el tiempo de respuesta y hagan, por tanto, viables y útiles estas soluciones.

Los problemas aplicados estudiados en este trabajo forman parte de un conjunto de modelos computacionales que nacieron dentro del mundo de la ingeniería agrícola con el objetivo de abordar problemas que ésta, de forma aislada, no podía resolver. El desarrollo práctico y la aplicación de estas técnicas de modelado de problemas reales fue retrasada debido al problema computacional que enfrentaban. Las cantidades de datos manejadas eran demasiado grandes y las técnicas de resolución necesitaban de un nivel de computación mayor al prestado por los ordenadores disponibles en su momento. Soluciones tales como los clusters de ordenadores eran costosas y añadían la dificultad de su programación y de contar con personal especialista.

La computación de altas prestaciones

Los entornos de programación secuencial avanzan con paso sostenido ofreciendo cada vez mejores prestaciones gracias a avances tecnológicos y arquitectónicos en los diseños de los procesadores. Sin embargo, siempre se han vislumbrado en el horizonte limitaciones físicas que hacen pensar que por aquí se llegará al final tarde o temprano. La solución de reunir varios dispositivos para seguir progresando por el camino de aumentar las prestaciones siempre ha estado presente. Tampoco es difícil pensar que podemos avanzar en paralelo por ambos caminos, dado que la agregación de recursos para resolver juntos un mismo problema siempre es una opción interesante. De hecho, la computación paralela ha estado presente a lo largo de toda la historia de la computación de altas prestaciones, y ha dado lugar a una disciplina de programación en sí misma.

Lo que ha cambiado significativamente el panorama de la computación paralela en los últimos años ha sido la incorporación de aceleradores hardware en los nodos de cómputo. Esta incorporación ha traído consigo varias ventajas tales como un incremento muy significativo en la capacidad computacional teórica del nodo, una relación entre las prestaciones medidas en FLOPS y el consumo de energía muy favorable, y el acercamiento de la computación de altas prestaciones a un gran número de usuarios. Los aceleradores hardware a los que nos referimos son básicamente tarjetas

gráficas (GPU–Graphics Processing Unit) como las existentes en cualquier equipo y que, además, se pueden adquirir a precios asequibles, sobre todo si comparamos el precio con sus prestaciones.

El inconveniente viene por la parte de la programación, de la dificultad que supone extraer el máximo rendimiento de estas configuraciones. Obtener las mejores prestaciones en entornos paralelos siempre ha resultado una tarea difícil para los programadores. Ahora, el carácter heterogéneo añade una gran complejidad añadida a la propia dificultad que tiene el diseño de aplicaciones para estos aceleradores gráficos. Sin embargo, la experiencia y el uso de herramientas adecuadas abren continuamente la puerta a soluciones reales y útiles basadas en estos dispositivos a problemas computacionalmente costosos.

Objetivo de la tesis

El trabajo tiene su enfoque en el campo de la computación paralela, en el desarrollo y utilización de modelos computacionales de sistemas paralelos para la resolución de problemas aplicados utilizando entornos heterogéneos. La propuesta consiste en examinar distintas posibilidades de mejora de las prestaciones de las aplicaciones de modelado que nos fueron facilitadas desde el Centro de Computación Agrícola de la UNIVASF para resolver sus problemas agrícolas. Entre estas propuestas, está la de realizar implementaciones de la aplicación o, al menos, de parte de la misma en GPU para acelerar cálculos. En nuestra búsqueda preliminar no hemos encontrado adaptaciones de estas aplicaciones para este tipo de dispositivos. Nuestra idea va más allá porque trata de hacer funcionar la aplicación eficientemente en entornos heterogéneos, es decir, aquellos que combinan los procesadores multicore y varias GPUs (multi-GPU). La tesis también pretende aportar técnicas que permitan el desarrollo de rutinas paralelas que puedan adaptarse automáticamente a las características del sistema paralelo donde se ejecutan, de manera lo más próxima a la óptima en cuanto a tiempo de ejecución se refiere. 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 los conocimientos que sobre computación paralela tengan los mismos. Los sistemas utilizados son representativos de las configuraciones más actuales utilizadas en el mundo de la computación científica de alto rendimiento.

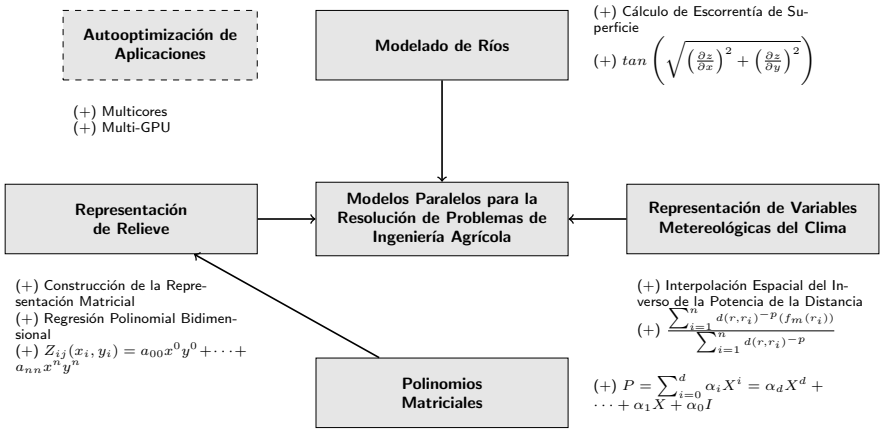


Figura 1.1: Problemas a tratar en esta tesis.

1.2 Los problemas agrícolas del Río São Francisco

Los problemas que muestra la Figura 1.1 han sido elegidos por su importancia, su coste computacional, pero también por su idoneidad al existir una posibilidad clara de realizar aportaciones significativas. Esperamos que la metodología desarrollada sirva para abordar problemas similares. Estos son los problemas abordados en la tesis:

Representación del relieve: El problema de la representación del relieve [1] se basa en una representación computacional matemática de la distribución espacial de un fenómeno que ocurre dentro de una región de la superficie de la Tierra. Según este modelo se puede representar distinta información geográfica de una superficie así como información geológica o geofísica. Mediante esta técnica se representa una superficie con un polinomio bidimensional que describe la variación de los datos de una muestra. La utilidad de esta técnica está limitada por la gran necesidad computacional requerida para realizar la regresión en un conjunto de datos muy grande. También vamos a tratar el problema inverso, que consiste en la formación del relieve de un área basándonos en el conocimiento que tengamos del mismo a través de su polinomio matricial.

Representación de variables meteorológicas del clima: La representación de variables meteorológicas del clima [2] es una técnica para conseguir simulaciones del clima de alta resolución a partir de bases de datos ya existentes. Las simulaciones climáticas que tratamos aquí se centran en la temperatura. En base a una red de estaciones meteorológicas físicas instaladas a lo largo de la región, el reto consiste en predecir la temperatura exacta en otros puntos contenidos dentro del mismo territorio que no están amparados por una estación. El alto coste computacional de las simulaciones climáticas limita su resolución espacial y obliga a buscar constantemente un equilibrio entre la longitud del periodo simulado, los recursos computacionales con los que se cuente, y el tiempo o el plazo de ejecución de que se disponga.

Modelo hídrico de ríos: El modelado hídrico de ríos [3] es una técnica de análisis que permite establecer criterios para comparar cuencas de distintas dimensiones eliminando la terminología subjetiva. Además del interés puramente geomorfológico de este tipo de trabajos, esta técnica presenta la utilidad adicional de intentar relacionar los parámetros morfométricos con el funcionamiento hidrológico de las cuencas de los ríos. Los datos morfológicos y la dinámica de fluidos que se representan y estudian con estos modelos son de vital importancia en un área tan dependiente de los recursos hídricos. La solución del problema pasa por resolver un sistema de ecuaciones sin demasiado coste, pero la construcción del propio modelo, que conduce a dicho sistema de ecuaciones, sí tiene un alto coste computacional.

1.3 Estado del arte

En este apartado tratamos de contextualizar un poco más el trabajo realizado en la tesis enumerando algunos de los trabajos que nos han parecido más significativos. El estado del arte relacionado lo clasificamos en dos temas de interés que están en la base de la tesis:

- la computación paralela heterogénea, y
- las técnicas de *autooptimización* de rutinas paralelas.

La Computación Paralela Heterogénea

La computación paralela es una solución evidente a los problemas que requieren gran capacidad computacional y que no pueden ser abordados por la tradicional máquina secuencial. Durante años se han sucedido las estrategias de configuración de entornos de computación paralela. Estas son clasificadas fundamentalmente, desde el punto de vista físico (hardware), en entornos de *memoria compartida* y de *memoria distribuida* dando lugar a sus correspondientes modelos de programación. Estos modelos difieren en la forma en la que se comunican los recursos computacionales, pueden ejecutar el mismo algoritmo o no, operar de forma síncrona o asíncrona, etc. Existen, por tanto, diversas formas de concebir la programación paralela y de construir modelos para resolver un problema mediante computación paralela.

De manera natural, tanto las soluciones basadas en máquinas paralelas como la problemática asociada a las mismas, se ha ido trasladando a los sistemas *heterogéneos*, es decir, entornos de computación formados por recursos computacionales diferentes [4]. El concepto de sistema heterogéneo se aplica a sistemas compuestos por múltiples recursos computacionales conectados. Debido a las diferencias entre los recursos que forman el sistema, es probable que las velocidades de cómputo de los recursos sean distintas y los tiempos de transferencia de datos también pueden ser diferentes en las comunicaciones entre ellos. La naturaleza de este tipo de entorno es dinámica y depende de qué recursos se utilicen en cada momento para resolver un problema y cuáles sean sus características. Mirando al trabajo que nos ocupa, vemos que incluso ya existen algunos trabajos relacionados con los que estudiamos en esta tesis, sobre arquitecturas paralelas heterogéneas [5, 6].

Como la programación es dependiente de la arquitectura subyacente tenemos una dificultad añadida en este tipo de sistemas, lo que conduce de manera habitual al desarrollo de modelos analíticos con los que predecir el rendimiento de un sistema paralelo [7, 8, 9, 10, 11, 12, 13, 14] y, de forma general, de un sistema paralelo heterogéneo. Sin embargo, se hace necesario revisar los modelos propuestos en este último caso [15, 16]. En los últimos 10 años se han publicado muchos trabajos que analizan los entornos heterogéneos y los parámetros que deben considerarse en el uso de estos sistemas. Algunos de los mejores trabajos realizados sobre sistemas heterogéneos podemos encontrarlos en [17, 18]. En [19] los autores describen diferentes tipos de heterogeneidad que pueden encontrarse en un sistema y estudian la mayo-

ría de las situaciones que surgen en plataformas heterogéneas para cómputos regulares e irregulares. Debido a los diferentes tipos de heterogeneidad los autores en [20] realizan una simplificación del problema al tener en cuenta solo la heterogeneidad producida por la diferencia entre las velocidades de procesamiento de los recursos computacionales. Sin embargo, es un hecho probado que la heterogeneidad en la velocidad de los recursos puede tener un impacto significativo en la sobrecarga de las comunicaciones [21], aún cuando las capacidades de comunicación en la red sean las mismas para todos los procesadores. En [22] los autores dan una definición formal de una plataforma heterogénea como un grafo dirigido donde cada nodo es un recurso de computación y los arcos son los enlaces de comunicación entre ellos. En [23] los autores realizan un estudio sobre estrategias de planificación teniendo en cuenta distintas plataformas heterogéneas en estrella o en árbol, considerando, además, diferentes niveles de paralelismo interno en los procesadores. En [24] se puede encontrar una heurística que proporciona la distribución de la carga óptima para los procesadores mediante el uso de programación dinámica.

Como se ha enunciado anteriormente, los sistemas paralelos heterogéneos han recibido atención por parte de los especialistas en computación paralela, aunque este interés puede considerarse marginal si lo comparamos con la cantidad de trabajos publicados para sistemas homogéneos. Sin embargo, esto ha cambiado radicalmente en los últimos años con la aparición de aceleradores gráficos como las GPUs. Por su naturaleza, los aceleradores gráficos son dispositivos de alta potencia computacional pero muy diferentes en su concepción a los procesadores de propósito general que podemos encontrar en los nodos de cómputo habituales. Además, estos aceleradores actúan como coprocesadores matemáticos y, por tanto, aparecen conectados a un sistema ordinario mediante el bus PCI. Incluso el más reciente diseño de Intel, el Xeon Phi, que puede ejecutar todo un sistema operativo en sí mismo, no puede trabajar de manera aislada sin estar conectado a un procesador de propósito general. Así pues, la computación paralela heterogénea está hoy más que nunca en el punto de mira de la computación de altas prestaciones. También para este tipo de configuraciones podemos encontrar modelos analíticos con los que predecir el rendimiento del sistema paralelo [25, 26]. La literatura es amplia en cuanto a aplicaciones que corren en este tipo de dispositivo, sin embargo, poco se ha realizado en lo que respecta a los procesos que hemos tratado en particular en la tesis. Solo en el caso de algunas librerías de propósito general que ya se encuentran

implementadas para GPUs las hemos utilizado para completar la aplicación que se ha desarrollado. Estas librerías se especifican en el capítulo siguiente.

Técnicas de autooptimización de rutinas paralelas

El tiempo de ejecución de una rutina matemática, por ejemplo, puede ser diferente según una serie de parámetros que permiten su ajuste tales como, por ejemplo, el tamaño de los datos, y que dependen del entorno. La variación del valor de estos parámetros no afecta al resultado pero sí al tiempo de ejecución, por lo que interesa conocer su valor para una determinada arquitectura. En pocas palabras, podríamos decir que la *autooptimización* es una herramienta mediante la cuál se determina el valor de estos parámetros de manera automática y, por lo tanto, un determinado software es capaz de "autoadaptarse" al entorno en el que va a ser instalado. Este objetivo científico ha motivado la elaboración de un gran número de trabajos, de los cuáles destacamos el de A. Kerr et. al [27], aunque enumeramos algunos más por considerar la existencia de GPUs en el sistema [28, 29, 30, 31, 32]. Estas técnicas se utilizan en la solución de gran cantidad de problemas aplicados, y en nuestro caso se adapta perfectamente por caracterizar problemas de alto coste computacional. Permiten 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.

Un ejemplo de software lo encontramos en el campo de las ecuaciones diferenciales para diferentes plataformas paralelas. En el trabajo de Brewer [33] 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. Otro ejemplo lo encontramos en el proyecto FFTW [34], donde el principal objetivo es la minimización automática del tiempo de resolución de transformadas rápidas de Fourier (FFT, fast Fourier transforms). La idea se basa en un generador de código que, durante la instalación, genera automáticamente trozos de código especializados en resolver la 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.

Un ejemplo muy significativo y utilizado es el de la optimización basada en *kernels* de álgebra lineal. La idea consiste en crear núcleos (*kernels*) secuenciales muy ajustados a bajo nivel. Un ejemplo es la librería BLAS [35] y una de sus implementaciones: la Math Kernel Library [36] de Intel. Estos núcleos constituyen una base sobre la que se van construyendo los sucesivos niveles superiores de rutinas de álgebra lineal. Las librerías LAPACK [37] y ScaLAPACK [38] son proyectos construidos sobre el nivel BLAS, e incluso se proyectan futuras versiones con esta idea sobre entornos heterogéneos [39]. Otro proyecto que va en la misma dirección es BLIS (new BLAS-like API) [40]. BLIS es un software ideado para crear instancias de alto rendimiento de rutinas tipo BLAS. Está escrito en C99 y permite el acceso libre al código.

En otro proyecto similar, el proyecto LAWRA (Linear Algebra With Recursive Algorithms) [41], el estudio se centra en mejorar el uso de la jerarquía de memoria de las plataformas computacionales 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. También para álgebra lineal encontramos el proyecto ATLAS [42, 43], que se caracteriza por ser una versión de BLAS con autooptimización. En tiempo de instalación se determinan experimentalmente algunos parámetros, entre ellos el tamaño de bloque y el nivel de bucles necesarios, para ofrecer las máximas prestaciones.

En el contexto de entornos heterogéneos podemos citar el proyecto mpC (The mpC parallel programming language) [4, 44], 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. Basado en mpC, la herramienta llamada HeteroMPI (Heterogeneous MPI) [45, 46, 47], per-

mite automatizar el proceso de transferencia de una aplicación paralela de un cluster homogéneo a un entorno heterogéneo. HeteroMPI ofrece un lenguaje reducido para especificar el modelo de rendimiento de la aplicación. Basándose en esta información, el runtime de HeteroMPI selecciona automáticamente un subconjunto de procesos MPI de forma óptima tal que puedan ejecutar el algoritmo homogéneo de la forma más eficiente posible en un cluster de nodos heterogéneo. El proyecto HeteroMPI habilitó la creación del proyecto HeteroScaLAPACK [48], como versión heterogénea de su homólogo ScaLAPACK. La tecnología basada en mpC/HeteroMPI proporciona la ventaja de reutilizar el software eficientemente diseñado para redes homogéneas de una manera sencilla.

Por último, también podemos hablar de una herramienta llamada Flamingo (Auto-tuning framework) [49] que presenta un marco de propósito general para ajustar automáticamente parámetros de rutinas paralelas que utilizan GPUs. La herramienta utiliza técnicas heurísticas y ciertas características de estos entornos heterogéneos para reducir significativamente el número de pruebas que se deben ejecutar para encontrar los parámetros óptimos y, por lo tanto, la cantidad de tiempo requerido para obtener la optimización.

El desarrollo de rutinas de optimización automática para álgebra lineal densa para plataformas paralelas [50] 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, la distribución del trabajo a realizar entre los procesadores de la plataforma, la selección de la librería básica de donde tomar la rutina para llevar a cabo cada operación y la elección del mejor algoritmo con el que resolver un problema de entre varios equivalentes. En un principio, estos tipos de aplicaciones 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.

Clasificación de trabajos bibliográficos

En la Tabla 1.1 se muestran de manera esquemática las características de algunos de los trabajos más significativos, anteriormente descritos y clasificados en tres grupos, cada uno de ellos caracterizado por el objetivo concreto perseguido en el proceso de reparto de tareas y/o autoajuste en

Tabla 1.1: Clasificación de trabajos bibliográficos.

Referencias	Descripción	G1	G2	G3
[5, 6]	Aplicaciones	•	—	•
[17, 18, 19, 20, 21, 22, 23]	Arquitecturas	•	—	•
[24]	Heurísticas	•	—	—
[7, 8, 11, 12]	Modelos Analíticos	•	—	•
[42, 43]	Proyecto ATLAS	—	•	•
[45, 46, 47, 51]	Proyecto HeteroMPI	—	•	•
[4, 44]	Proyecto mpC	—	•	•
[49]	Proyecto FLAMINGO	—	•	•
[9, 10, 25, 26]	Modelos de Rendimiento	•	•	•
[27, 29, 31]	Herramientas	—	•	•

plataformas heterogéneas. Podemos encontrar tres grupos:

Grupo 1 (G1)
Este grupo lo forman aquellos trabajos que persiguen repartir tareas para optimizar en arquitecturas heterogéneas.

Grupo 2 (G2)
Este grupo lo forman aquellos trabajos que tratan de obtener los mejores valores para una serie de parámetros ajustables, elegir qué recursos utilizar, balancear el trabajo, elegir el algoritmo concreto a utilizar entre varios posibles o bien escoger de entre varios recursos disponibles más apropiados para ser ejecutados.

Grupo 3 (G3)
El último grupo, en el que se inscribe nuestra propuesta, resume todas las optimizaciones perseguidas que se materializan en una metodología basada en el modelo analítico del tiempo de ejecución de la rutina a la que se le añaden las características del sistema heterogéneo.

Podríamos decir que el grupo G3 trata de ser una recopilación de los modelos presentados en la literatura actual aplicados a problemas aplicados de altas prestaciones. En esta tesis se propone un modelo de prestaciones

descrito mediante de un algoritmo híbrido con particionamiento irregular de los datos, donde existe un conjunto de tareas simétricas o no, asignadas a más de un recurso computacional. La gran mayoría de las estrategias propuestas en la literatura sugieren que el reparto del trabajo puede incluir una distribución heterogénea de los datos, que supone una recodificación total de las rutinas. Sin embargo, si el reparto del trabajo se hace mapeando heterogéneamente habrá una recodificación mínima. Luego, si únicamente se realiza una selección de recursos a utilizar de entre los de mayor capacidad de cómputo no será necesario ninguna recodificación.

1.4 Aportaciones de la tesis

Las aportaciones hechas por este trabajo y que tienen un carácter novedoso se pueden clasificar en los siguientes puntos:

- ◇ En una primera aportación hemos realizado una implementación del método de regresión polinomial utilizado para resolver el problema de la representación del relieve para GPUs. La solución que hemos aportado es eficiente en un sistema heterogéneo que utiliza todos sus aceleradores gráficos y cores CPU en la resolución del problema.

- ◇ La segunda aportación consiste en la implementación de un algoritmo recursivo para evaluar polinomios matriciales en GPUs. A pesar de la utilidad que tiene este algoritmo en un amplio espectro de aplicaciones, no hemos encontrado nada parecido en la bibliografía hasta el momento.

- ◇ La tercera aportación ha consistido en el desarrollo de algoritmos paralelos, tanto para CPUs como para GPUs, para la representación de variables meteorológicas del clima. De hecho, solo tenemos constancia de que se hayan estudiado y desarrollado algoritmos en secuencial para modelos pequeños utilizando el método de interpolación espacial del inverso de la potencia de la distancia.

- ◇ Como cuarta aportación hemos acelerado el proceso de construcción de la matriz del sistema que describe el modelo hídrico de ríos mediante la utilización de cores CPU y GPUs.

- ◇ Consideramos la última de las aportaciones de la tesis como una de las más significativas debido al impacto real que puede tener en el futuro aprovechamiento de este trabajo. Aquí se han desarrollado modelos paralelos autooptimizados capaces de encontrar el mejor conjunto de valores para unos parámetros bajo los que se describe perfectamente una plataforma de ejecución. Con ello logramos una adaptación automática del software a la arquitectura destino que los usuarios finales puedan utilizar de manera sencilla y eficiente.

1.5 Estructura de la memoria

La tesis doctoral se estructura en 9 temas. El primer capítulo introduce al lector en el problema a abordar.

El **Capítulo 2** es meramente descriptivo y sirve para mostrar las arquitecturas heterogéneas, los compiladores, librerías y otras herramientas software utilizadas para realizar los experimentos. También se describen las métricas de rendimiento que hemos utilizado para medir y comparar.

El **Capítulo 3** presenta un modelo para representar el relieve de un determinado territorio. Se trata de una herramienta utilizada para simplificar esta representación y permite, básicamente, reducir su almacenamiento en memoria. Como consecuencia, la demanda computacional aumenta significativamente. Nosotros tratamos de optimizar este problema.

En el **Capítulo 4** se propone resolver el problema inverso al tratado en el capítulo anterior. Una vez calculados los coeficientes del polinomio que representa una superficie, se trata de reconstruir la misma de la manera más rápida posible. Como resultado, hemos diseñado una rutina rápida para evaluar polinomios de matrices con un amplio abanico de aplicaciones que van más allá de la reconstrucción de superficies.

El **Capítulo 5** presenta un modelo para la representación de variables meteorológicas del clima, utilizando el método de ponderación del inverso de la potencia de las distancias, optimizado a través de un modelo de computación paralela que utiliza multicores y GPUs.

En el **Capítulo 6** hemos tratado el tercero de los problemas agrícolas abordados en esta tesis. Trabajamos sobre un modelo computacional hídrico de ríos, mostramos cómo hemos optimizado la aplicación y cómo

la hemos adaptado para ser ejecutada eficientemente en entornos paralelos heterogéneos basados en multicore y en multi-GPU.

En el **Capítulo 7** tratamos el problema de la *autooptimización* de aplicaciones y la utilizamos en los tres problemas agrícolas. Proponemos un modelo de asignación de recursos que realiza la asignación de una aplicación paralela a recursos computacionales para obtener el mejor rendimiento posible. Según nuestra propuesta, la rutina se adapta de manera automática a las características del sistema de cómputo multicore y multi-GPU.

El **Capítulo 8** recoge las principales conclusiones obtenidas con relación a esta tesis, el entorno donde se ha trabajado, los proyectos y las aportaciones realizadas a lo largo de este trabajo. Finalmente, se relacionan algunas posibles líneas futuras de trabajo.

Herramientas computacionales 2

Este capítulo es meramente descriptivo y tiene como objetivo especificar las diferentes herramientas computacionales, tanto hardware como software, utilizadas en el desarrollo de este trabajo. También describimos de manera teórica las métricas y los diferentes estimadores de rendimiento que serán utilizados a lo largo del trabajo.

2.1 Introducción

Las configuraciones paralelas heterogéneas están formadas básicamente por elementos hardware distintos entre sí. Pero también el software puede considerarse elemento básico que caracteriza a una plataforma heterogénea. Las plataformas heterogéneas son las que se han utilizado en este trabajo, esencialmente multiprocesadores de memoria compartida que integran además varias GPUs (Graphics Processing Units).

En este capítulo describimos este tipo de dispositivos, concretamente en la Sección 2.2. La Sección 2.3 se dedica a la descripción de compiladores, entornos y librerías utilizadas para programar los dispositivos mencionados en la sección anterior. Terminamos el capítulo describiendo las métricas utilizadas en los experimentos (Sección 2.4).

2.2 Herramientas de hardware

En este trabajo se han utilizado máquinas formadas por nodos multiprocesador con un número de cores que varía entre 4 y 12, algunos con *Hyper-Threading* activado. Por lo tanto, el esquema básico de programación paralela estará basado en el modelo de *memoria compartida*, pues los procesadores o cores comparten la memoria física. Los entornos de experimentación utilizados son los siguientes:

MICROMACHIN

La plataforma MICROMACHIN (`micromachin.iteam.upv.es`) está formada por un nodo biprocesador Intel Xeon E5530 a 2,40 GHz y con 48 GB DDR3 de memoria principal. Cada procesador contiene 4 cores con 8 MB de memoria cache L3. La plataforma dispone de 4 GPUs NVIDIA Tesla C2070 con 12 Stream Multiprocessors (SM) y 20 Stream Processors (SP) lo que da un total de 240 procesadores o cores. La memoria global de cada dispositivo es de 4 GB. Cada SM posee 12 unidades de load/store y 32K de memoria *shared* configurable. La SM tiene unidades en coma flotante de simple y doble precisión. La aritmética es IEEE 754-2008. Esta plataforma pertenece al *Interdisciplinary Computation and Communication Group* [52] de la Universidad Politécnica de Valencia, España.

ELEANORRIGBY

La plataforma ELEANORRIGBY (`eleanorrigby.iteam.upv.es`) está formada por un nodo biprocesador Intel Xeon X5680 a 3,33 GHz y con 96 GB DDR3 de memoria principal. Cada procesador contiene 6 núcleos con 12 MB de memoria cache L3. La plataforma contiene 2 GPUs NVIDIA Tesla C2070 con 14 Stream Multiprocessors (SM) y 32 stream processors (SP) lo que da un total de 448 procesadores o cores. Otras características son que posee 16 unidades de load/store y 64K de memoria *shared* configurable. La memoria global de cada dispositivo es de 6 GB. Cada SM tiene unidades de ejecución en coma flotante de simple y doble precisión. Las operaciones en coma flotante siguen el estándar IEEE 754-2008. Esta plataforma pertenece también al *Interdisciplinary Computation and Communication Group* [52] de la Universidad Politécnica de Valencia, España.

GPU

La plataforma GPU (`gpu.dsic.upv.es`) tiene un procesador Intel Core(TM) i7-3820 con cuatro cores a 3,60 GHz, 16 GB DDR3 de memoria principal y 16 MB de memoria cache. La plataforma tiene 2 GPUs NVIDIA Tesla K20c. Cada GPU posee 12 Stream Multiprocessors (SM) y 24 Stream Processors (SP) por multiprocesador lo que da un total de 2496 cores. Cada SM posee 12 unidades de load/store y 128K de memoria *shared* configurable. La memoria del dispositivo es de 4,8 GB. Esta plataforma pertenece al Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia.

2.3 Herramientas de software

Las herramientas software utilizadas en el presente trabajo se están constituidas por lenguajes de programación y por librerías.

Como lenguajes, se ha utilizado ANSI C/C++. Para la programación de los procesadores (nodo) se ha utilizado la herramienta OpenMP, mientras que para la programación de GPUs se ha utilizado la herramienta CUDA.

OpenMP (Open Multi-Processing)

Para la programación en memoria compartida dentro del nodo de computación se ha utilizado OpenMP [53]. OpenMP es un estándar actual para la programación utilizando el modelo de memoria compartida, que utilizan los sistemas multicore y ordenadores de alto rendimiento. OpenMP es un lenguaje de macros mediante las que el programador expresa el paralelismo de la aplicación en función de un conjunto de hilos concurrentes. Constituye, por tanto, una capa por encima del estándar POSIX-threads. OpenMP define también una API para llamadas a funciones de biblioteca que aumentan la funcionalidad del programa y permiten, por ejemplo, la creación y gestión de semáforos, medir tiempos, establecer y/o averiguar el número de *threads*, etc.

CUDA (Compute Unified Device Architecture)

El lenguaje de programación utilizado para programar las GPUs ha sido CUDA [54] dado que todas las tarjetas son de NVIDIA y CUDA es un lenguaje de programación diseñado para estas GPUs. Se trata de una extensión de C++ que permite la definición de *kernels*, que son las unidades de proceso que ejecuta cada uno de los cores de la GPU. Este lenguaje permite seleccionar un subconjunto de *threads* y configurarlos en bloques que, a su vez, forman una malla o grid, y que serán mapeados convenientemente en la GPU para que ejecuten el kernel. El modelo de programación es completamente paralelo, y es similar al modelo SIMD (Simple Instruction, Multiple Data) [55], solo que en el caso de CUDA éste ha sido denominado como SIMT (Simple Instruction, Multiple Threads) dado que es ligeramente diferente por basarse en la programación de *threads*. Este modelo supone un cambio conceptual radical en la forma de concebir un programa paralelo y de estructurar los datos sobre los que operan los cores de la GPU, pero es, a su vez, un modelo muy potente que persigue como objetivo que la escalabilidad hacia futuras GPUs más potentes (con más SMs y/o SPs) sea automática.

BLAS: Basic Linear Algebra Subprograms

Colección de rutinas para realizar operaciones básicas con matrices y vectores [56].

- ◇ En los nodos, provistos de procesadores Intel en todos los casos, se han utilizado la versiones optimizadas en Intel, la Math Kernel Library (MKL) [36].
- ◇ En las GPUs, todas de NVIDIA, hemos utilizado CUBLAS [57], una implementación de BLAS para GPUs de NVIDIA. Esta librería permite crear matrices y vectores en memoria de la GPU, transferir los datos entre la memoria del nodo o *Host* y la memoria del dispositivo o GPU, e invocar funciones de BLAS para realizar operaciones.

LAPACK: Linear Algebra PACKage

Librería que contiene rutinas desarrolladas en lenguaje FORTRAN para resolver problemas comunes del álgebra lineal tales como sistemas de ecuaciones lineales, problemas de mínimos cuadrados lineales o problemas de valo-

res propios y problemas de valores singulares [37]. Hemos utilizado siempre versiones optimizadas para el procesador del nodo. Estas versiones están contenidas también en la libraría MKL. En caso de las GPUs no hemos necesitado utilizarla.

2.4 Métricas de rendimiento utilizadas

En esta sección se especifican algunas de las métricas empleadas para analizar el rendimiento en arquitecturas heterogéneas y que serán utilizadas a lo largo del trabajo.

Tiempo de ejecución paralelo

Se denomina tiempo de ejecución paralelo, $t_{paralelo}$, al tiempo transcurrido desde el comienzo de la ejecución paralela hasta el momento del término en el último proceso [58]:

$$t_{paralelo}(i) = max_{i=0}^{r-1}(t_{absoluto}(i)) \ ,$$

donde $t_{absoluto}(i)$ es el tiempo empleado por la ejecución paralela desde su inicio hasta el término del proceso i para un numero de recursos r.

Potencia computacional

La potencia computacional en un sistema heterogéneo puede ser definida como la cantidad de trabajo realizada por un proceso durante una unidad de tiempo [59]. La potencia computacional depende de las características físicas del recurso computacional, pero también de la tarea que se esté ejecutando en él. Esto significa que este valor puede variar para diferentes aplicaciones y, debido a las limitaciones del tamaño de memoria, caché y otros componentes hardware, la velocidad de cómputo puede cambiar también al modificar el tamaño del problema de la aplicación.

En el mundo del cálculo científico, tradicionalmente el parámetro de mayor interés es la potencia computacional de un recurso 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 recurso computacional [4].

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 recursos computacionales con características heterogéneas. La técnica desarrollada consistió en un conjunto de experimentaciones con la rutina secuencial sobre un conjunto de recursos computacionales, midiendo sus respectivos tiempos absolutos, a través de la razón entre el tiempo experimental y el comportamiento asintótico de la rutina (O):

$$t_{absoluto}(i) = \frac{t_{experimentación}(i)}{O}.$$

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 recurso y el sumatorio de todos los tiempos absolutos del grupo de cómputo.

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

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. En lugar de obtener índices absolutos de potencia para cada recurso 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 recurso computacional un valor característico expresado en porcentajes de capacidad de cómputo real para cada máquina perteneciente al conjunto de computación elegido.

Heterogeneidad

Otra de las métricas que pueden ser utilizadas en un sistema es su nivel de heterogeneidad. Este parámetro ofrece un valor de la similitud o diversidad de las máquinas del sistema. Por ejemplo, cuando un número pequeño de recursos es mucho más rápido o más lento que la mayoría se considera que es un sistema con una heterogeneidad alta. Se calcula en función de la potencia computacional de sus máquinas: escalando los valores obtenidos de las potencias computacionales para los recursos, de modo que a aquellos que sean más rápidos se les asigne un valor proporcional a la potencia computacional entre 0 y 1, donde cuanto más cerca del valor 1, más heterogéneo es el entorno [4].

Coste de las comunicaciones

Otra de las métricas importantes que debemos considerar es el coste de las comunicaciones entre los recursos. Un modelo habitual utiliza β_{ij} y α_{ij} (latencia y tiempo de transferencia respectivamente entre dos recursos i y j) para representar el coste de las comunicaciones mediante el modelo lineal clásico: $t_c(n) = \beta_{ij} + \alpha_{ij} * n$.

Para modelar estas comunicaciones se han medido los parámetros β y α mediante experimentos de tipo ping-pong [60]. Se ha utilizado la función `cudaMemcpy`, teniendo en cuenta las direciones en que se hacen las transferencias, definidas en el estándar de la librería CUDA. Al final se realiza un ajuste lineal de los tiempos de envío en función del tamaño del mensaje.

Representación del relieve 3

En este capítulo estudiamos un modelo utilizado para representar el relieve de una determinada área de terreno. La herramienta utilizada consiste en simplificar la representación del suelo utilizando un método de regresión polinomial bidimensional. Optimizamos la aplicación mediante la utilización de un modelo de computación paralela basado en multicores y GPUs.

3.1 Introducción

Avances recientes en el campo de la computación han propiciado el desarrollo de aplicaciones orientadas a representar recursos geofísicos de una manera diferente y más óptima. Entre éstas, destaca la representación del relieve a través de polinomios bidimensionales [1]. El problema de la representación del suelo a través de un polinomio ha sido previamente estudiado en diversos trabajos como, por ejemplo, en [61]. Sin embargo, el problema de esta aproximación radica en la gran carga computacional subyacente a este método ya que cuanto más grande es la información a reproducir, más cálculos son requeridos. La información necesaria es proporcional al área de suelo que se desea representar y a la precisión requerida. Una precisión muy alta requiere manejar polinomios con un alto grado. De esta manera, los

recursos computacionales disponibles o el tiempo, pueden limitar el grado del polinomio que puede ser estimado y/o también el tamaño de la zona a representar.

Son muchas las motivaciones para estudiar y representar el relieve del suelo de manera eficiente dada su amplia aplicabilidad. Tenemos como aplicaciones, por ejemplo, el diseño de plantaciones para la optimización de los recursos hídricos, la logística de almacenaje de la producción o la minimización de los efectos de erosión, entre otras. Por lo tanto, el proceso de representación del relieve se convierte en una herramienta esencial e indispensable en el funcionamiento eficiente de la agricultura en general, pero de una manera muy especial en la región agrícola localizada en el Valle del Rio São Francisco, que destaca como una de las más grandes áreas de vinicultura y producción de frutas para exportación en Brasil. Uno de los principales problemas que impiden una mayor eficiencia de los factores de la producción agrícola en este área es el uso inapropiado de técnicas de optimización en el modelado matemático de representación del relieve.

Este capítulo de la tesis aporta una optimización del método de regresión polinomial basada en el diseño de algoritmos paralelos específicos capaces de obtener el máximo rendimiento de nodos multicore con GPUs. El capítulo se estructura en cuatro secciones. La siguiente sección explica el modelo matemático utilizado para representar el relieve. La Sección 3.3 presenta el modelo computacional propuesto para la resolución del problema. La Sección 3.4 muestra los resultados experimentales obtenidos mientras que la última sección enumera las conclusiones del capítulo.

3.2 Modelado matemático de representación del relieve

Un modelo matemático del suelo es una representación computacional matemática de la distribución espacial de un fenómeno que ocurre dentro de una región de superficie de la Tierra [1]. Este modelo puede ser personalizado para representar determinadas características de la información geográfica de los distintos terrenos, tales como propiedades geológicas, geofísicas, la altitud, etc. Una de las técnicas para lograr esta representación de la superficie consiste en un modelo de *malla rectangular*, donde el mapeado de la superficie se realiza con un ajuste global a través de la técnica de regresión

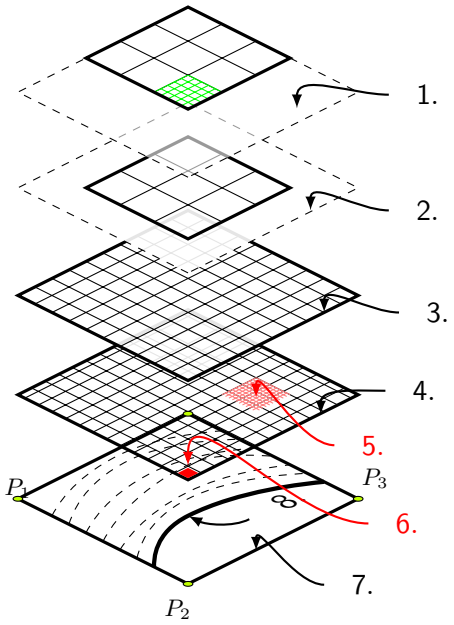


Figura 3.1: Modelo de malla rectangular.

polinomial. Esta técnica ajusta el polinomio bidimensional que mejor describe la variación de los datos de una muestra, en la malla espacial definida. La Figura 3.1 ilustra un ejemplo de una representación en malla regular generada a partir de una muestra regularmente espaciada que simboliza la información de altitud del terreno, donde se observan varias etapas que tienen el siguiente significado:

Etapas de 1-3: Las primeras etapas se caracterizan por la generación de una malla (conjunto de vértices) de tamaño predeterminado.

Etapas de 4-6: Las etapas siguientes generan la malla de tal forma que existe un número más grande de vértices concentrados cerca del centro. Cuanto más grande es la distancia al centro, más pequeño es el número de vértices presentes. Así pues, cuanto más lejos del centro, más pequeña será la necesidad de renderizar el suelo con una alta fidelidad.

Etapas de 7-8: Y por último, con el mapa de altitud originado, se reconstruye el terreno como una malla que es generada a medida que se mueve por el suelo.

La regresión polinomial es un modelado matemático que trata de describir la relación entre los fenómenos observados mediante un polinomio. De acuerdo con Rawlings et al. [62], el modelado se refiere al desarrollo de un modelo analítico matemático que describe el comportamiento de variables aleatorias de interés. Este modelo es utilizado para describir el comportamiento de las variables independientes cuya relación con la variable dependiente está mejor representada de una manera no lineal.

La relación entre las variables está descrita por funciones polinomiales bidimensionales, donde la fluctuación de la variable dependiente está relacionada con variables independientes. Específicamente, en el estudio desarrollado en este capítulo, la regresión no lineal se utiliza para describir la relación entre dos variables independientes (latitud y longitud) y una variable dependiente (altitud). El modelado matemático desarrollado permite que los coeficientes de dos dimensiones en una función polinomial de grado diferente en x y en y represente la variación en la altitud de un área.

Cuando se utilizan modelos matemáticos de regresión, el método de estimación de los parámetros más utilizado es el método de los mínimos cuadrados, que consiste en estimar una función para representar un conjunto de puntos minimizando el cuadrado de las desviaciones. Consideramos un conjunto de coordenadas geográficas (x, y, z), donde la altitud (z) es estimada en función de los puntos x e y con polinomio de grado r en x y de grado s en y conforme a la ecuación

$$\widehat{z} = f(x_i, y_j) = \sum_{k=0}^{r} \sum_{l=0}^{s} a_{kl} x_i^k y_j^l \ ,$$

con un error ε_{ij} estimado por la ecuación

$$\varepsilon_{ij} = z_{ij} - \widehat{z_{ij}} \ ,$$

con $0 \leq i \leq m$ y $0 \leq j \leq n$. De forma particularizada consideramos que, para la representación del relieve, los valores de m y n son iguales por el recorte de la muestra analizada, siendo definido como el valor de la longitud de la suma de los términos de la representación matricial que será

presentada. En los experimentos, el rango de este valor varía de 1.3 a 25.4 millones de términos.

Los coeficientes a_{kl} ($k = 0, 1, \ldots, r$ y $l = 0, 1, \ldots, s$) que minimizan el error de la función estimadora $f(x, y)$ pueden obtenerse solucionando la siguiente ecuación

$$\frac{\partial \xi}{\partial a_{cd}} = 0 \,, \tag{3.1}$$

para $c = 0, 1, \ldots, r$ y $d = 0, 1, \ldots, s$, donde

$$\xi = \sum_{i=0}^{m} \sum_{j=0}^{n} \varepsilon_{ij}^2 = \sum_{i=0}^{m} \sum_{j=0}^{n} (z_{ij} - \widehat{z_{ij}})^2 \,. \tag{3.2}$$

La Ecuación 3.1 se desarrolla en las cinco siguientes ecuaciones:

$$\varepsilon_{ij}^2 = (z_{ij} - \sum_{k=0}^{r} \sum_{l=0}^{s} a_{kl} x_i^k y_j^l)^2 \,,$$

$$\xi = \sum_{i=0}^{m} \sum_{j=0}^{n} (z_{ij} - \sum_{k=0}^{r} \sum_{l=0}^{s} a_{kl} x_i^k y_j^l)^2 \,,$$

$$\frac{\partial \xi}{\partial a_{cd}} = 2 \sum_{i=0}^{m} \sum_{j=0}^{n} [(z_{ij} - \sum_{k=0}^{r} \sum_{l=0}^{s} a_{kl} x_i^k y_j^l)) x_i^c y_j^d] \,, \tag{3.3}$$

$$2 \sum_{i=0}^{m} \sum_{j=0}^{n} [(z_{ij} - \sum_{k=0}^{r} \sum_{l=0}^{s} a_{kl} x_i^k y_j^l)) x_i^c y_j^d] = 0 \,,$$

$$\sum_{i=0}^{m} \sum_{j=0}^{n} [z_{ij} x_i^c y_j^d - (\sum_{k=0}^{r} \sum_{l=0}^{s} a_{kl} x_i^{k+c} y_j^{l+d})] = 0 \,.$$

A continuación se muestra un ejemplo del polinomio que quedaría en el caso particular $r = s$ y $n = m$:

$$\widehat{Z_{ij}}(x_i, y_i) = a_{00}x^0y^0 + a_{01}x^0y^1 + a_{02}x^0y^2 + a_{10}x^1y^0 + \cdots + a_{nn}x^ny^n \ .$$

La resolución final se resume en la representación en formato matricial $Ax = b$, donde la matriz $A = [A_{kl}]_{k,l=0,\ldots,(s+1)^2-1}$ y el vector $b = [b_k]_{k=0,\ldots,(s+1)^2-1}$ se definen como:

$$A_{kl} = \sum_{i=0}^{m}\sum_{j=0}^{n} x_i^{\alpha}y_j^{\beta} \ , \tag{3.4}$$

$$b_k = \sum_{i=0}^{m}\sum_{j=0}^{n} z_{ij}x_i^{\alpha}y_j^{\beta} \ , \tag{3.5}$$

con $\alpha = k \operatorname{div}(s+1)+l \operatorname{div}(s+1)$ y $\beta = k \bmod(s+1)+l \bmod(s+1)$. El vector $x = [x_l]_{l=0,\ldots,(s+1)^2-1}$ contiene los elementos del polinomio $[a_{cd}]_{c,d=0,\ldots,s}$ de tal forma que $x_l = a_{cd}$, siendo $c = l \operatorname{div}(s+1)$ y $d = l \bmod(s+1)$. La estimación consta de dos partes principales:

1. La construcción de la matriz A y del vector b por medio de la Ecuación 3.4 y la Ecuación 3.5, y

2. la solución del sistema lineal de ecuaciones $Ax = b$.

3.3 Modelo computacional de representación del relieve

Lo primero que cabe destacar es que esta técnica de modelado de una superficie está limitada por la gran demanda computacional exigida para realizar la regresión en un conjunto de datos muy grande. En realidad, el coste computacional más grande del algoritmo de representación del relieve está

Algoritmo 3.1 Rutina `matrixComputation` para la construcción de la matriz A.

```
 1: int N = (s+1) * (s+1);
 2: for( int k = 0; k < N; k++ ) {
 3:  for( int l = 0; l < N; l++ ) {
 4:   int exp1 = k/s + l/s;
 5:   int exp2 = k%s + l%s;
 6:   int ind = k + l*N;
 7:    for( int i = 0; i < q; ++i ) {
 8:     A[ind] += pow(x[i],exp1) * pow(y[i],exp2);
 9:    }
10:  }
11: }
```

en la construcción de la estructura matricial A. El esquema de construcción de esta matriz se muestra en el Algoritmo 3.1. Vemos que la relación entre el tamaño de la matriz `N`×`N` y el grado del polinomio s es `N`$= (s+1)^2$. Los vectores `x` y `y` han sido previamente cargados desde un archivo y almacenados de manera que se hace una suma de orden cuadrático de tamaño `n`. La rutina `matrixComputation` recibe los vectores como argumentos (`x` y `y`), el tamaño de la suma (`q`) y el orden polinomial (`s`), devolviendo a la salida la matriz A. Como la construcción de la matriz A (Ecuación 3.4) es la parte más costosa del proceso global[1], se puede observar que existe una gran oportunidad de paralelismo en este cálculo. No es difícil ver que todos los elementos de la matriz se pueden calcular de forma simultánea. Además, cada término de la suma puede ser calculado de forma independiente. Nuestro algoritmo paralelo se basa en estas observaciones y en que el valor del orden del polinomio s posee valores experimentales que van desde 2 hasta 40, lo que produce matrices cuyo orden va desde 9 hasta 441 (variable `N`). Además, la longitud de la suma (`q`) se encuentra en un rango que va desde los 1.3 hasta los 25.4 millones de términos, dependiendo de las características de la malla deseadas [1].

La formación de la matriz A se basa en el particionado de la suma representada en la Ecuación 3.5, cada trozo o parte de esta suma puede tener

[1]Consideramos implícitamente que la carga de trabajo consiste tanto en la construcción de la matriz A como en la construcción del vector b. En la implementación real, ambos cálculos se han separado en dos partes con el fin de distribuir su cómputo entre los cores CPU.

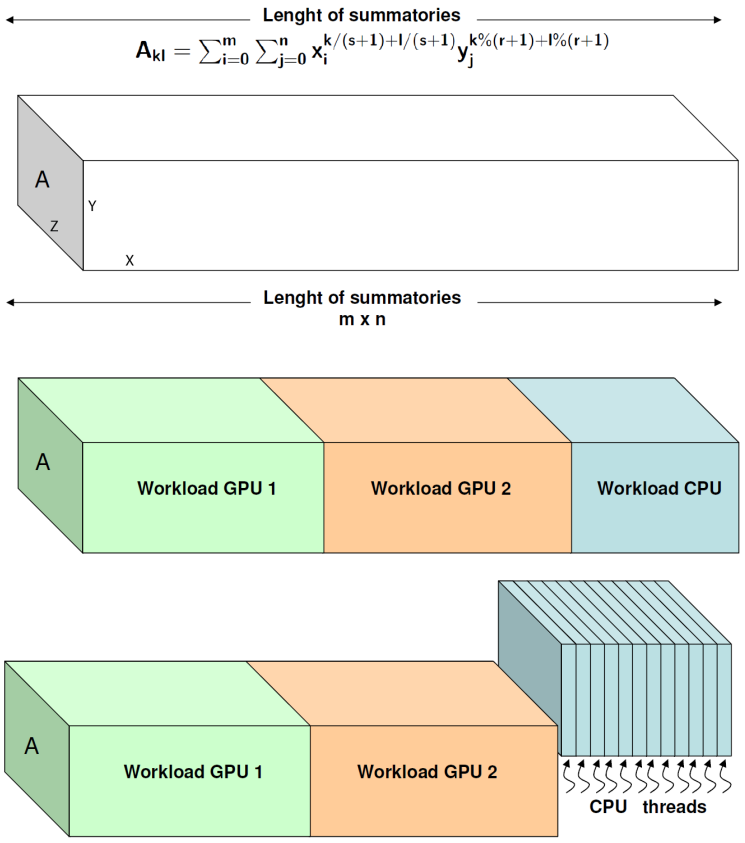


Figura 3.2: Estructuras matriciales particulares que presentan los cálculos en trozos de las matrices, siendo representado por la suma de los términos de A, para la partición de carga de trabajo entre los multicores y multi-GPU.

un tamaño dado diferente del resto de los trozos. Esto es lo que representa la Figura 3.2.

La aplicación diseñada es un programa escrito en C que ejecuta la CPU y que puede observarse en el Algoritmo 3.2. El programa genera primero un número de *hilos* o *threads* (`nthreads`) en las líneas 3 y 4 mediante la utilización de la directiva `parallel` de OpenMP. Cada hilo trabajará con diferentes sumas, es decir, calculará una de las partes en las que se ha particionado la suma total para producir la matriz `A_[thread_id]` como resultado (esta

Algoritmo 3.2 Rutina de computación utilizando multicore y multi-GPU para la representación del relieve.

```
 ENTRADA
  numberCPUcores /* numero de cores de CPU utilizados     */
  numberGPUs     /* numero de GPUs utilizados             */
  sizeGPU        /* tamanos de problema asignados a cada GPU */
  problemSize    /* tamano de problema total, para ser divido
                    entre CPU y GPU                        */
 SALIDA
 /* Superficie que mejor ajusta los puntos de la malla regular. */
1: nthreads = numberCPUcores + numberGPUs;
2: sizeCPU  = (problemSize-numberGPUs*sizeGPU)/numberCPUcores;
3: omp_set_num_threads(nthreads);
4: #pragma omp parallel
5: {
6:  thread_id = omp_get_thread_num();
7   if(thread_id < numberGPUs) {  /* computacion con la GPU */
8:    first = thread_id * sizeGPU;
9:    cudaSetDevice(thread_id);
10:   matrixGPU(thread_id, first, sizeGPU);
11: } else {                     /* computacion con la CPU */
12:   first = numberGPU*sizeGPU + (thread_id-numberGPU)*sizeCPU;
13:   matrixCPU(thread_id, first, sizeCPU);
14:   }
15:}
```

matriz y otras están ocultos en otras partes del código y no aparecen en el Algoritmo 3.2). Consideramos `A_[thread_id]` como un puntero a una matriz de longitud `N`×`N`. El resultado, la matriz `A`, es la suma de todas estas matrices de tal forma que `A` = `A_[0]` + `A_[1]` + ... + `A_[nthreads-1]`[2]. La rutina `matrixComputation` puede ser fácilmente modificada para trabajar sobre estructuras matriciales particulares que presentan los cálculos en trozos (Figura 3.2).

El desarrollo de una versión de las rutinas para varios *threads* es muy simple ya que en los sistemas actuales tenemos cores distribuidos entre los procesadores de la CPU y las GPUs, pero la explotación eficiente de todo

[2]Toda discusión para el cálculo de la matriz A es extrapolable para el cálculo del vector b, incluyendo su cómputo en la misma rutina.

el sistema no es una tarea trivial. Para los experimentos consideramos un sistema heterogéneo con multicore y multi-GPU idénticas.

Tanto el Algoritmo 3.2 como la Figura 3.2 muestran el esquema utilizado para la distribución de la carga de trabajo entre CPU y GPU por medio de la estructura condicional `if` que comienza en la línea 7. La idea es gestionar cada uno de los *threads* vinculados, bien a un core CPU o bien a una de las GPUs existentes en el nodo, dando como número total `nthreads`. Los primeros *threads* se asocian a los dispositivos GPUs, mientras que el resto a los cores. A veces, se obtienen mejores resultados cuando el número de cores se sobreescribe, es decir, cuando utilizamos una cantidad de *threads* mayor que el número de cores CPU existentes en el sistema debido a que el *Intel Hyper-Threading* [63] está activado. Para enviar los datos y las tareas a los cores CPU y a las GPUs hemos empleado una estrategia estática. Esto quiere decir que el porcentaje de carga de trabajo asignado a cada sistema es una entrada para el algoritmo, es decir, es equilibrado por el usuario en la llamada al programa. El usuario de la rutina especifica el porcentaje deseado de computación que será asignado a cada tipo de dispositivo. A través de este dato se calcula el balanceo de las tareas, teniendo como base el tamaño de los datos, antes de llamar al Algoritmo 3.2. La variable `sizeGPU` almacena el número de términos de la suma que será asignada a cada GPU mientras que la variable `sizeCPU` almacena la cantidad total de términos a calcular por cada core CPU del sistema multicore.

Cada *thread* trabaja en una parte diferente de los vectores `x`, `y` y `z`. Las rutinas `matrixCPU` y `matrixGPU` son adaptaciones del Algoritmo 3.1 que reciben punteros a la ubicación adecuada en la matriz para los vectores `x`, `y` y `z` (almacenados en la variable `first` en las líneas 8 y 12) y la longitud de la suma. La matriz A y el vector b son la salida del Algoritmo 3.2. Contienen la suma total computada (Ecuación 3.4 y Ecuación 3.5). Se utilizan vectores para el almacenamiento de las matrices que representan las sumas parciales y que se han obtenido de manera independiente por cada uno de los subsistemas del nodo. Una vez que los *threads* terminan en el punto de sincronismo implícito de la línea 15, el *thread* maestro lleva a cabo la suma total de estos resultados parciales.

Kernel CUDA para la representación del relieve

El cálculo en cada GPU es llevado a cabo por un kernel (`matrixGPU`) al que se llama en la línea 10 del Algoritmo 3.2. Esta rutina lleva a cabo, en

primer lugar, operaciones habituales de asignación de tareas en la GPU y la carga de los datos de la memoria del nodo a la GPU. En la descripción de la implementación del kernel, las estructuras matriciales A, `x` e `y` han sido previamente cargadas con los datos, es decir, los datos están en la memoria global de las GPUs.

El grado de paralelismo intrínseco a la operación de construcción de la matriz A es utilizado para implementar el kernel en CUDA. Por un lado, todos los elementos de la matriz A pueden ser computados concurrentemente. Por otro lado, los términos de la suma pueden ser calculados de manera independiente. A fin de explotar toda esta concurrencia hemos utilizado un grid de bloques de *threads*. Cada bloque de *threads* tiene tres dimensiones: `BLKSZ_X`×`BLKSZ_Y`×`BLKSZ_Z`. Los valores concretos son constantes definidas como macros de C en las tres primeras líneas del Algoritmo 3.3. Cada *thread* se encuentra en un bloque a través de las 3 coordenadas representadas por tres variables, respectivamente (líneas 5–7). El grid o malla de bloques de *threads* se ha gestionado como tridimensional pero con la primera dimensión establecida a 1, por lo que es en realidad de dos dimensiones. Las otras dos dimensiones son $\lceil$`N/BLKSZ_Y`$\rceil$ y $\lceil$`N/BLKSZ_Z`$\rceil$, siendo `N` la dimensión de la matriz A. El siguiente código está dentro de la rutina `matrixGPU` y muestra esta distribución y la llamada al kernel:

```
1: dim3 dimGrid(1,idiv(N,BLKSZ_Y),idiv(N,BLKSZ_Z));
2: dim3 dimBlock(BLKSZ_X,BLKSZ_Y,BLKSZ_Z);
3: kernel<<<dimGrid,dimBlock>>>(...);
```

La rutina `idiv` devuelve la longitud entera de las dos últimas dimensiones. El objetivo es que todos los *threads* dentro de un bloque puedan calcular simultáneamente las instrucciones asignadas (línea 22). Al *thread* con las coordenadas correspondientes se le asigna el cálculo de los términos de la suma `X`+i, con $i = 0$:`BLKSZ_X`: n. Esta operación se realiza en un bucle que se cierra en la línea 24. Los exponentes `exp1` y `exp2` dependen de la línea `k` y la columna `l` indexadas en la matriz A. Estos índices se calculan en las líneas 5 y 6, basándose en las coordenadas `Y` y `Z` del *thread*. Todos los bloques de *threads* almacenan las informaciones en los vectores `x` e `y`. La memoria compartida es un espacio de rápido acceso desde todos los *threads* dentro del bloque. Cada *thread* en la dimensión `Y=0` y `Z=0` realiza la carga de cada elemento de los vectores `x` e `y` en los vectores `sh_x` e `sh_y` (líneas 17–20) en la memoria compartida. Más tarde, en el bucle de las líneas 29–30 todos los términos de la suma asignados al *thread* se calculan y almacenan en la varia-

Algoritmo 3.3 Rutina con el kernel CUDA para la representación del relieve.

```
1: #define BLKSZ_X    128
2: #define BLKSZ_Y    2
3: #define BLKSZ_Z    2
4: __global__ void kernel( double *A, *x, *y, int s, n, N ) {
5:  int k = blockIdx.y * blockDim.y + threadIdx.y;
6:  int l = blockIdx.z * blockDim.z + threadIdx.z;
7:  int X = threadIdx.x, Y = threadIdx.y, Z = threadIdx.z;
8:  __shared__ double sh_x[BLKSZ_X], sh_y[BLKSZ_X];
9:  __shared__ double sh_A[BLKSZ_X][BLKSZ_Y][BLKSZ_Z];
10:
11: if( k < N && l < N ) {
12:    int exp1 = (k/s) + (l/s);
13:    int exp2 = (k%s) + (l%s);
14:    for( int K = 0; K < n; K += BLKSZ_X ) {
15:      int i = X+K;
16:      if( i < n ) {
17:        if( Y == 0 && Z == 0 ) {
18:          sh_x[X] = x[i];
19           sh_y[X] = y[i];
20:        }
21:      __syncthreads();
22:      a += pow(sh_x[X],exp1) * pow(sh_y[X],exp2);
23:      }
24:    }
25:    sh_A[X][Y][Z] = a;
26:    __syncthreads();
27:    if( X == 0 ) {
28:      a = 0.0;
29:      for( int i = 0; i < BLKSZ_X; i++)
30:        a += sh_A[i][Y][Z];
31:      A[k + N*l] = a;
32:    }
33:  }
34: }
```

ble `a`. Este valor se almacena, a su vez, en memoria compartida. Por tanto, todo el vector tridimensional `sh_A` de tamaño `BLKSZ_X`×`BLKSZ_Y`×`BLKSZ_Z` acabará almacenado en la memoria compartida en este punto.

Tenemos que imaginar la información almacenada en la memoria com-

partida, `sh_A`, como un cubo tridimensional donde cada elemento contiene la posición de cada suma parcial de la suma total. Existe una suma para cada elemento en la línea y columna `r`×`c` de la matriz `A`. En otras palabras, un elemento `sh_A[i][Y][Z]`, para todo `i`, contiene la suma parcial que corresponde a un elemento dado `r`×`c`, teniendo en cuenta la correspondencia entre la matriz y las coordenadas de los *threads* en la línea 25. Seguidamente, es necesario añadir todas las sumas parciales en la dimensión `X`. Esta operación (líneas 27–32) es llevada a cabo solo por los *threads* en los cuales `X=0`. Una vez que la suma se ha calculado, el resultado es guardado en la memoria global (línea 31).

Los puntos de sincronización dentro de la rutina se indican con la palabra reservada `syncthreads()`, que sirve para asegurarnos de que los datos están en la memoria compartida antes de su lectura [54]. El uso de la memoria compartida se restringe a tamaños pequeños que necesitan ser asignados estáticamente. El tamaño de memoria compartida de las GPUs utilizadas es de 48 KB. Esto limita el número máximo de *threads* a 1024 en el entorno de experimentación utilizado. La cantidad total de memoria compartida está realmente determinada por el tamaño del bloque. De todos modos, la limitación del número de *threads* por bloque se puede superar fácilmente aumentando el número de los bloques. Los valores más típicos en el ejemplo y entorno utilizados son los que se muestran en la línea 1. Experimentalmente hemos comprobado que las dimensiones `Y` y `Z` podrían ser iguales, y que éstas a su vez están relacionadas directamente con `N`, el tamaño de la matriz `A`. La dimensión `X` está relacionada con el valor de `n`, que es el tamaño de los vectores `x` e `y`. Los valores de `N` varían de 9 hasta 441, mientras que los valores de `n` están en un rango que va desde los 1.3 a los 25.4 millones de términos en nuestros experimentos. Dada la desproporción que existe entre `N` y `n`, está claro que existe más oportunidad de concurrencia en el valor de la dimensión del tamaño de la variable `X` que en las dimensiones correspondientes al tamaño de la matriz A. Hemos elegido la primera dimensión "más grande" debido a los límites que impone la GPU en cuanto al tamaño máximo de las dimensiones, y hemos variado el número de bloques de *threads* entre 64 y 1024.

3.4 Resultados experimentales

En esta sección se van a describir diversos experimentos mediante los cuales se estudian los algoritmos propuestos anteriormente. Se pretende estudiar los costes de las diferentes partes de dichos algoritmos, así como realizar comparaciones entre ellos, variando el tamaño del problema. Para ello es necesario aplicar los esquemas propuestos de representación del relieve creados previamente. Se incluyen estudios experimentales, explicaciones y comentarios detallados de la optimización para el entorno de ejecución ELEANORRIGBY descrita en la Sección 2.2.

Análisis de la representación del relieve

Con el fin de validar la metodología presentada mostramos la optimización del modelo de regresión polinómica bidimensional aplicada a la representación del relieve de una región del Valle del Rio São Francisco. La fuente con los datos de la zona elegida viene de un Modelo Digital del Terreno (DTM[3]) [64], en la forma de una matriz regular, con espaciamiento de aproximadamente 900 metros en las coordenadas geográficas. Los datos de las elevaciones indican una dispersión de 1 a 2863 metros. El DTM con 1400 puntos tiene solamente 722 puntos dentro de la región, los demás puntos están fuera del área.

Utilizando todos los puntos que representan los datos de la forma de la representación del relieve de la zona y la Ecuación 3.3, calculamos los coeficientes del polinomio para representar la variación de altitud del terreno. La estimación de un polinomio de grado alto necesita gran potencia de cálculo y puede necesitar, por tanto, un largo tiempo de procesamiento pero también se obtiene más exactitud en la descripción de la forma del suelo logrando, de esta manera, un nivel más satisfactorio de detalles. A partir de las ecuaciones que se presentan en la Sección 3.2 y del índice de correlación R^2 se ha determinado que el polinomio para un nivel considerado óptimo tendría un grado del orden de 500, y la representación del relieve para el Valle del Rio São Francisco se ha ajustado con un índice de correlación $R^2 = 0{,}99$. Un cálculo estimativo del coste de obtener este polinomio arroja un valor de aproximadamente 45 años como tiempo necesario para generar todos los coeficientes de los polinomios del sistema de ecuaciones

[3]DTM = Digital Terrain Model.

Tabla 3.1: Estimación del tiempo necesario en meses y/o años para obtener el resultado de la representación del relieve frente a la correlación R^2 y al grado del polinomio deseado.

Grado	Tiempo	R^2	Grado	Tiempo	R^2
30	10.7 meses	0.70	225	14.5 años	0.90
40	11.3 meses	0.73	250	18.8 años	0.91
50	1.2 años	0.77	300	20.9 años	0.93
60	3.1 años	0.80	350	28.4 años	0.95
70	5.7 años	0.81	375	37.5 años	0.96
80	7.2 años	0.82	400	39.2 años	0.97
90	8.3 años	0.86	450	40.3 años	0.98
100	12.6 años	0.89	500	45.9 años	0.99

lineales mediante un cálculo secuencial (Tabla 3.1). Esta estimación fue realizada considerando una plataforma formada por un nodo con 1 procesador Intel Pentium M 755 Dothan con 512 Mb DDR. Analizando la Figura 3.3 observamos que existe una gran diferencia entre las superficies cuanto más alto es el grado del polinomio y que, por tanto, para generar una superficie con una fidelidad suficientemente representativa, es necesario estimar con un grado alto.

Utilizando las informaciones del DTM, obtuvimos la solución para el modelo desarrollado, mostrado en los mapas de elevación 3D (Figura 3.3) y en la proyección 2D de tonos de grises (Figura 3.4). También se puede observar que la región estudiada tiene una topografía muy desigual, lo que hace más necesario el grado alto del polinomio para alcanzar una representación exacta de la superficie.

Estudio experimental

En esta sección se muestran los resultados experimentales obtenidos con la implementación del algoritmo paralelo cuyo esquema se ha propuesto en la Sección 3.3 para construir el mapa de la superficie analizada con un ajuste global mediante la técnica de regresión polinomial. El algoritmo paralelo es heterogéneo ya que utiliza tanto los cores CPU como las GPUs disponibles en el nodo. Las rutinas se han compilado con el compilador

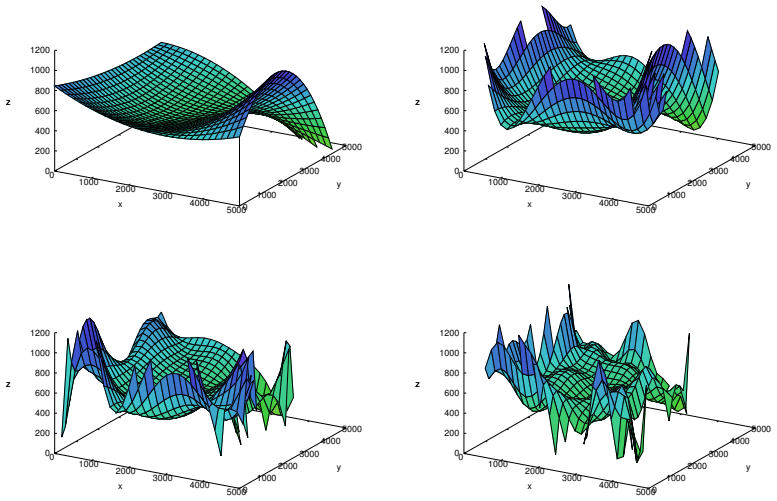


Figura 3.3: Visión 3D de la representación del relieve del Valle del Rio São Francisco para polinomios de grados 2, 4, 6 y 20.

de CUDA `nvcc`. En los experimentos, primero incrementamos el número de CPU *threads* desde 1 hasta 24 para obtener el número que minimiza el tiempo de ejecución en el caso de utilización de la CPU únicamente. Luego añadimos 1 y 2-GPUs, respectivamente, al número de *threads* obtenidos en la prueba anterior. El grado del polinomio es requerido como entrada y se han utilizado en los experimentos los valores: $8, 12, \ldots, 40$. El rendimiento de los algoritmos se analiza en la Tabla 3.2 y en la Figura 3.5.

La ejecución utilizando solamente un *thread* se ha denotado como "Secuencial (S)", mientras que la denominación "CPU cores (OMP)" denota la computación con el uso de múltiples *threads* en los cores CPU, donde los hilos han sido generados mediante la directiva OpenMP correspondiente, es decir, la versión OMP distribuye el cálculo matricial entre los cores CPU únicamente. Por el contrario, las versiones 1-GPU y 2-GPUs representan ejecuciones en una y dos GPUs, respectivamente, sin utilizar cores CPU. El modelo paralelo híbrido "multicore+multi-GPU" muestra los valores máximos de *speedup* obtenidos experimentalmente utilizando los parámetros

Figura 3.4: Tonos de grises de la representación del relieve del Valle del Rio São Francisco para polinomios de grados 2, 4, 6 y 20.

óptimos para el sistema de ejecución. Los parámetros óptimos obtenidos para el entorno de ejecución utilizado son: 2-GPUs + 2-Procesadores Hexacore (número de procesadores utilizados = 24), y con una carga de trabajo asignada de la siguiente manera: (GPU, GPU, CPU) = (45 %, 45 %, 10 %). En este modelo, los *threads* son ejecutados por todos los cores de la máquina, teniendo el número adecuado de procesadores de los multicore y las dos GPUs como parámetro de ejecución. Los resultados de los experimentos muestran que el algoritmo en multicore (OMP) reduce significativamente el tiempo de ejecución con respecto a la versión secuencial. Como puede verse en la Tabla 3.2, la aceleración máxima es de alrededor de 12, que coincide con el número de cores CPU.

La Figura 3.5 muestra los rendimientos obtenidos a través del *speedup*. Debe tenerse en cuenta que, en general para polinomios de grado pequeño, el rendimiento de OMP es mayor que el rendimiento con 1-GPU (grado <= 10). Esto es debido al sobrecoste introducido por la transferencia de datos entre la CPU y la GPU. Del mismo modo, el rendimiento de 1-GPU es más grande que el rendimiento con 2-GPUs también para tamaños muy pequeños. Aunque en este caso se debe al tiempo de configuración necesario en la selección de los dispositivos que, en particular en nuestra máquina, es muy alto $\approx$ 4,5 seg. El mejor resultado se ha obtenido con todos los

Tabla 3.2: Tabla con los tiempos de ejecución (en segundos) variando el tamaño del problema (grado del polinomio).

Grado	S	OMP	1-GPU	2-GPUs	multicore+multi-GPU
8	84.49	12.32	12.44	14.19	13.61
12	386.17	41.85	21.36	19.39	18.04
16	1166.88	114.55	43.31	31.48	25.53
20	2842.52	268.32	90.57	57.03	89.29
24	5916.06	544.93	172.71	101.08	118.72
28	11064.96	1011.42	310.53	276.88	219.67
32	24397.66	1777.25	521.63	385.07	322.82
36	30926.82	2700.00	828.50	450.67	448.16
40	46812.70	4252.69	1261.09	666.77	600.90

recursos disponibles en el sistema heterogéneo. El incremento de velocidad aumenta con el grado del polinomio llegando a un máximo de 78, un número que se ha obtenido mediante la comparación de la aceleración de 1-GPU con respecto a 1 core de la CPU. El uso de la GPU como una herramienta independiente proporciona beneficios, pero no permite alcanzar el rendimiento potencial que podría obtenerse mediante la adición de más GPUs y el sistema multicore.

3.5 Conclusiones

Entre las herramientas de desarrollo más importantes utilizadas para obtener el máximo rendimiento de los sistemas computacionales utilizados, se puede destacar OpenMP y CUDA, y haciendo un análisis comparativo entre ellos y teniendo como base el problema de la representación del relieve, es evidente que las aplicaciones heterogéneas utilizando cores CPU y GPUs tienen un enorme potencial. El desarrollo de aplicaciones computacionales para estos sistemas, aunque más complejo, da al programador más control sobre el algoritmo diseñado y más potencial para aprovechar al máximo los recursos disponibles. Los resultados experimentales obtenidos en este estu-

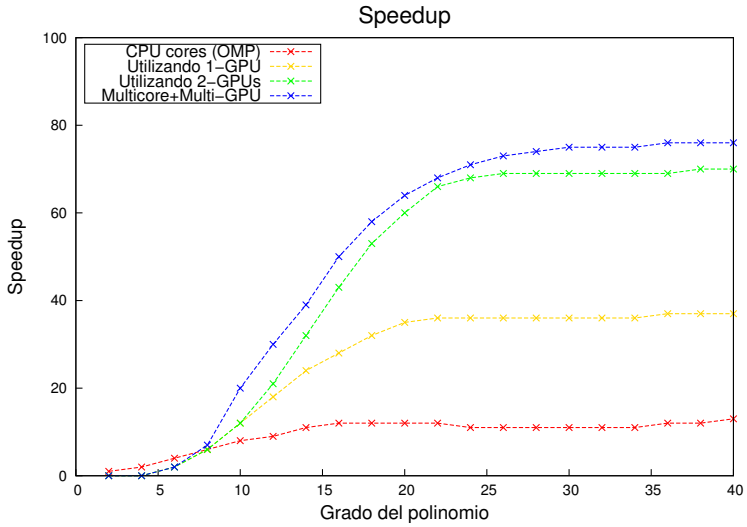


Figura 3.5: Gráfico con el *speedup* variando el grado del polinomio.

dio demuestran que es posible utilizar este método de modelado para la representación del relieve y obtener representaciones del terreno bastante precisas en un tiempo razonable. Para ello, ha sido necesario diseñar una aplicación heterogénea que realiza una clasificación de tareas eficiente y un kernel apropiado para la resolución del problema en GPUs. Se ha utilizado como ejemplo el área del Valle del Rio São Francisco pero es evidente que esta solución es válida para representar el relieve de cualquier área de terreno.

Evaluación de polinomios matriciales 4

Motivados por la experiencia en el trabajo realizado en el capítulo anterior, hemos decidido abordar, en esta parte de la tesis, el problema de la reconstrucción de una superficie de la que se posee almacenada la información en forma de polinomio bidimensional. A través de polinomios matriciales se pueden describir las superficies espaciales de manera más reducida y, por lo tanto, ahorrando memoria. Sin embargo, la reconstrucción de la superficie en base a los coeficientes del polinomio requiere un alto coste computacional. A pesar de tratarse de un problema fácilmente definible y de amplia aplicabilidad, no hemos encontrado trabajos que traten este problema en el caso matricial. Parte de la investigación en este campo se quedó en trabajos antiguos para polinomios escalares en los que, por ejemplo, se reduce el número de operaciones para evaluar un polinomio [65]. También antiguo y célebre es el trabajo realizado en [66] en el que se trata la evaluación eficiente y paralela de polinomios escalares. Nosotros hemos revisado estos trabajos y los hemos ampliado al caso matricial utilizando entornos heterogéneos. Basado en la idea de Paterson–Stockmeyer [65], proponemos en este capítulo un algoritmo recursivo y una implementación eficiente para GPUs utilizando OpenMP para la gestión de los threads CPU que manejan las GPUs.

4.1 Introducción

El cálculo de polinomios matriciales ha recibido un amplio impulso en el pasado ya que ésta es una operación de base para calcular las funciones de la matriz [67]. La evaluación de un polinomio matricial es un proceso básico para el cálculo de funciones de matrices por el método de Taylor. El primer contacto de nuestro grupo de investigación con el tema surgió al desarrollar un modelo para la representación del relieve mediante polinomios bidimensionales (Capítulo 3). Pero también muchos otros fenómenos de la ingeniería y la física se rigen por sistemas de ecuaciones diferenciales ordinarias lineales de primer orden de ecuaciones con coeficientes constantes cuya solución se da en términos de una exponencial de matriz. La exponencial de una matriz juega un papel muy significativo en muchas áreas de la ciencia y la tecnología como la teoría de control, electrodinámica sobre medios estratificados, líneas de energía eléctrica multimodo, etc [68, 69, 70]. Otras funciones de matriz como la función del signo o la función logaritmo aparecen, por ejemplo, en la teoría de control [67, pp 39]. Además, algunos otros procesos de ingeniería son descritos por ecuaciones diferenciales de segundo orden, cuya solución exacta se da en términos de las funciones trigonométricas matriciales del seno y el coseno. Todo lo anterior sirve de motivación añadida para trabajar en métodos rápidos que permitan evaluar polinomios matriciales.

Hay diferentes técnicas para calcular o aproximar funciones de matrices. Algunas de ellas son métodos generales pero otras son aplicables solo a funciones particulares. Existen dos técnicas ampliamente utilizadas para aproximar una función matricial. Una se basa en aproximaciones polinómicas y la otra se basa en aproximaciones racionales. Al contrario de lo que se pensaba en el pasado, es posible obtener mayor precisión con aproximaciones polinómicas que con aproximaciones racionales, incluso con un coste computacional similar o inferior [71, 72, 73, 74].

Definimos un polinomio matricial P de grado d como

$$P = \sum_{i=0}^{d} \alpha_{d-i} X^{d-i} = \alpha_d X^d + \alpha_{d-1} X^{d-1} + \cdots + \alpha_1 X + \alpha_0 I \ , \qquad (4.1)$$

donde $X, I \in \mathbb{R}^{n \times n}$, siendo I la matriz identidad.

También definimos el vector $\bar{\alpha}$ como $\bar{\alpha} = [\alpha_i]_{i=0,\ldots,d}$ por conveniencia para futuras descripciones. La idea principal es mostrar cómo evaluar polinomios de matrices en problemas de gran escala. Consideramos aquí que un problema es de gran escala cuando el tamaño de la matriz, el grado del polinomio, o bien ambas cosas son grandes. Por ejemplo, los problemas de representación del relieve vistos en el capítulo anterior son problemas de gran escala dado que las matrices representan toda el área del Rio São Francisco, y para obtener un nivel aceptable de precisión en la representación necesitamos que el grado del polinomio sea alto.

Hemos diseñado y evaluado varios algoritmos para evaluar polinomios utilizando GPUs. El método propuesto en esta tesis está basado en POSIX *threads*, cada uno de ellos vinculado a una de las GPUs del nodo. Los hilos son lanzados utilizando directivas OpenMP.

En primer lugar, proponemos y evaluamos experimentalmente una paralelización simple con el objeto de comparar con otras opciones más sofisticadas. La siguiente sección describe esta paralelización utilizando tanto una como dos GPUs, para mostrar después los resultados en la Sección 4.3. En la Sección 4.4 estudiamos e incorporamos un método para reducir el número de operaciones necesario para evaluar polinomios: el método de Paterson–Stockmeyer method [65]. Hemos adaptado dicho método al caso de polinomios matriciales dando como resultado un algoritmo bastante simple en su concepción. Su simplicidad está basada, en parte, en su naturaleza recursiva, derivada de la regla de Horner, una regla que permite reducir el número de multiplicaciones de matrices en la evaluación del polinomio [75]. En la misma sección explicamos cómo aplicar este método al diseño de un algoritmo paralelo que utilice dos GPUs. Los resultados experimentales de esta aproximación están explicados en la Sección 4.5.

4.2 Un algoritmo sencillo para evaluar polinomios matriciales

Un algoritmo para evaluar la Ecuación 4.1 puede ser expresado de muchas maneras. El diseño mostrado en la función EVALUATE (Algoritmo 4.1) ayuda al usuario a implementarlo fácilmente en CPU utilizando BLAS, dado que los espacios de trabajo A, B y P han sido expresamente introducidos para este fin. Un algoritmo para una GPU (1-GPU) también es fácil de

Algoritmo 4.1 Algoritmo para la evaluación de un polinomio matricial.

```
1: function EVALUATE( n, X, d, ᾱ ) return P
2:     P ← α_0 I
3:     P ← P + α_1 X
4:     B ← X
5:     for i ← 2, d do
6:         A ← B
7:         B ← A · X
8:         P ← P + α_i B
9:     end for
10: end function
```

implementar utilizando las rutinas del paquete CUBLAS, una vez la matriz X ha sido cargada en la memoria global de la GPU.

La evaluación de polinomios matriciales en 2-GPUs en más complicada. Las dos GPUs trabajan concurrentemente y esto se ha de expresar en el código de alguna manera. La naturaleza asíncrona de la ejecución de *kernels* CUDA con respecto al nodo o *host* permite implementar una solución basada, por ejemplo, en llamadas replicadas, es decir, una llamada a cada kernel o rutina por GPU. En esta tesis proponemos un modelo diferente de lo usual y sencillo con objeto de gestionar esta concurrencia. Hemos optado por un modelo de programacion SPMD (*Single Program, Multiple Data*) para implementar el programa en GPU. Este modelo es fácil de entender y muy utilizado en contextos de memoria distribuida donde los programadores utilizan el modelo de Paso de Mensajes. Bajo este paradigma, las dos GPUs comparten exactamente el mismo código, salvo pequeñas partes que son específicas para cada GPU. La estrategia consiste en lanzar dos POSIX *threads* utilizando OpenMP, en particular, por medio de la directiva `parallel for`. De esta manera, cada GPU ejecuta una iteración del bucle sobre sus propios datos. Dado que el código puede ser largo, todo el proceso se ha dividido en varios bucles paralelos. La creación/destrucción de hilos concurrentes conlleva la creación/destrucción de variables locales a dichos hilos. Sin embargo, la información que almacenan algunas de estas variables debe persistir entre varias regiones paralelas. Para conservar esta información hemos utilizado la característica de OpenMP `threadprivate`, la cuál fue introducida en la especificación V3.0 de esta herramienta con el objeto de calificar variables cuyo contenido debía persistir de manera estática a lo

Algoritmo 4.2 Algoritmo para el cálculo de las potencias de X utilizando 2-GPUs.

```
1: function COMPUTE_POWERS( d, X ) return (A, m)
2:     #pragma omp parallel for
3:     for g ← 0, 1 do
4:         m = d/2 + g ·  mód (d, 2)
5:         A(0) ← X
6:         A(1) ← X · A(0)                                   ▷ A(1) = X^2
7:         if g = 0 then
8:             A(0) ← A(1)                                   ▷ A(0) = X^2 si g = 0
9:         end if
10:        A(2) ← A(0) · A(1)                                ▷ A(2) = { X^4 if g = 0
                                                                      { X^3 if g = 1
11:        for i ← 3, m do
12:            A(i) ← A(1) · A(i − 1)                        ▷ A(i) = X^2 · A(i − 1)
13:        end for
14:    end for
15: end function
```

largo de toda la ejecución del programa. Una vez creadas las variables al inicio de una región paralela, ésta es replicada por cada uno de los hilos y pertenece a dicho hilo durante toda la ejecución. Además, el valor de esta variable se conserva entre la destrucción y creación de los *threads*, esto es, entre regiones paralelas consecutivas. Lógicamente, el modelo es fácilmente extensible al caso de tener más de dos GPUs, incluso a otros aceleradores hardware diferentes que puedan existir en un contexto heterogéneo.

Hemos dividido el proceso de evaluar un polinomio en dos partes: *a)* el cálculo de las potencias de la matriz X (Ecuación 4.1), y *b)* la propia evaluación del polinomio. Este método es más difícil de ser programado que el método descrito en el Algoritmo 4.1, donde no se distinguen ambos cálculos. Sin embargo, estructurando el proceso de esta manera obtenemos la ventaja de poder evaluar más de un polinomio del mismo orden una vez estén calculadas las potencias de X. Otro inconveniente de esta opción está en el espacio de memoria que se necesita para almacenar explícitamente estas potencias. Sin embargo, este segundo problema se puede aliviar utilizando paralelismo.

El Algoritmo 4.2 describe el primer paso, el cálculo de las potencias de

matriz utilizando la mencionada directiva de OpenMP para crear un bucle paralelo. Las dos GPUs ejecutan el mismo código, solo que cada GPU ejecuta una iteración distinta de las dos existentes. En concreto, cada GPU está al cargo de calcular un conjunto diferente de potencias de X. Denotamos en el texto como GPU0 la primera GPU y mediante GPU1 a la segunda, de manera que la GPU0 (con índice $g = 0$ en el bucle del algoritmo) calcula las potencias pares de X mientras que la GPU1 (con índice $g = 1$) calcula las impares.

Las potencias de la matriz X están almacenadas en un vector o array de matrices A,

$$A = [A(i)]_{i=0,\ldots,m}, \quad A(i) \in \mathbb{R}^{n \times n} ,$$

siendo $m = \frac{d}{2} + g \cdot \mod (d, 2)$, donde $g \in \{0, 1\}$ representa el número de dispositivo. El valor de este array para cada GPU después de la ejecución del Algoritmo 4.2 es

$$A = \begin{cases} \begin{pmatrix} X^2 & X^2 & X^4 & X^6 & X^8 & \ldots \end{pmatrix} & \text{if GPU0} \\ \begin{pmatrix} X & X^2 & X^3 & X^5 & X^7 & \ldots \end{pmatrix} & \text{if GPU1} \end{cases} .$$

Las matrices $A(0)$ y $A(1)$ han sido utilizadas de una manera especial en ambos dispositivos tal que los productos matriciales pueden ser llevados a cabo mediante la rutina correspondiente de CUBLAS, permitiendo trasladar el Algoritmo 4.2 directamente a CUDA. Esto implica que la GPU1 tiene que calcular X^2 (línea 6) aunque no es una potencia impar porque se necesita para calcular el resto de potencias. También, la GPU0 tiene que almacenar X^2 en $A(0)$ (línea 8) aunque este término ya está en $A(1)$ dado que los argumentos que son punteros a matrices en la multiplicación de CUBLAS no pueden referenciar a la misma posición de memoria.

La segunda etapa es la evaluación del polinomio, y puede verse en el Algoritmo 4.3. El algoritmo se basa en la misma idea utilizada anteriormente en el Algoritmo 4.2, que consiste en un bucle paralelo, donde cada GPU ejecuta una iteración. Cada GPU utiliza la matriz A calculada en la etapa anterior. El algoritmo recibe un grado de polinomio q como argumento que será d como argumento actual para la solución de la Ecuación 4.1. La GPU0 calcula los términos pares de la Ecuación 4.1 en la matriz B, mientras que la GPU1 calcula los impares en su propia matriz B. La matriz hB es un vector de dos matrices alojada en el *host*. La matriz $hB(0)$ es para la GPU0

Algoritmo 4.3 Algoritmo para la evaluación de polinomios matriciales en 2-GPUs.

```
1: function EVALUATE2( n, q, ᾱ, A ) return P
2:     #pragma omp parallel for
3:     for g ← 0, 1 do
4:         m = q/2 + g ·  mód (q, 2)
5:         if g = 0 then
6:             B ← α_0 I
7:         else
8:             B ← α_1 A(0)
9:         end if
10:        for i ← 1 + g, m do
11:            B ← B + α_{2i−g} A(i)
12:        end for
13:        hB(g) ← B
14:    end for
15:    (host) P ← hB(0) + hB(1)
16: end function
```

y la $hB(1)$ es para la GPU1. Cada GPU carga su propia matriz B en la componente correspondiente de la matriz hB (línea 15). El último paso lo lleva a cabo el *host* para formar el resultado final, esto es, la evaluación del polinomio de matrices P (Ecuación 4.1). Por simplicidad en la descripción de los algoritmos hemos omitido las transferencias *host*-GPU que son necesarias antes de los cálculos.

4.3 Resultados de la evaluación de polinomios matriciales

Los resultados experimentales que se muestran en este apartado han sido obtenidos en el entorno de ejecución GPU (Sección 2.2). En primer lugar mostramos en la Figura 4.1 el tiempo de ejecución (arriba) y *speedup* (abajo) para la evaluación de un polinomio, teniendo como base la Ecuación 4.1, y para diferentes grados entre 4 y 20. La matriz X posee un tamaño fijo de $n = 4000$ en este ejemplo. Ambos gráficos muestran cómo la GPU supera claramente las prestaciones de la CPU en nuestro sistema por lo que

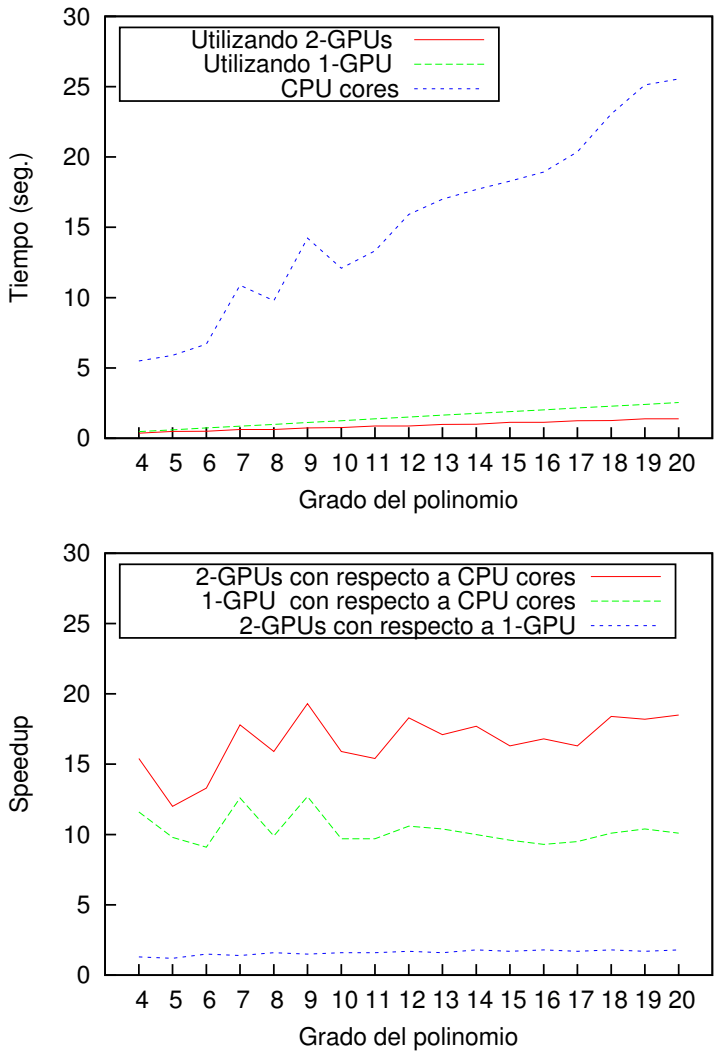


Figura 4.1: Tiempo de ejecución y *speedup* para evaluar un polinomio matricial con matriz de tamaño $n = 4000$ y variando el grado.

omitiremos los resultados en la CPU en el resto del análisis.

La Figura 4.2 muestra el incremento de velocidad logrado usando dos

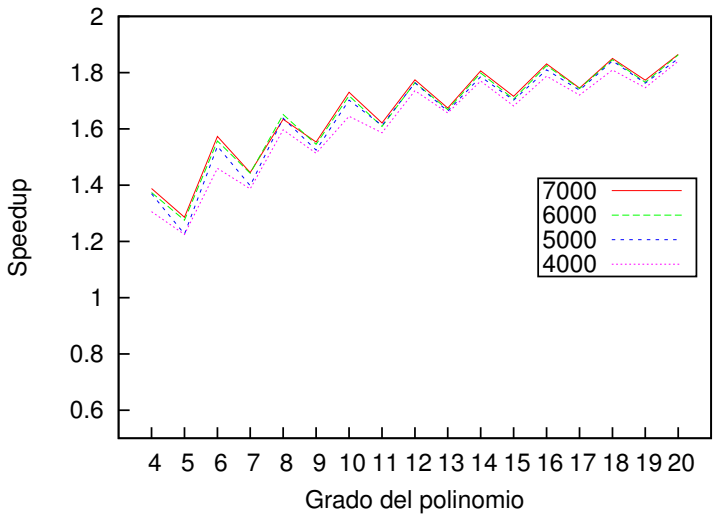


Figura 4.2: *speedup* para la evaluación de polinomios matriciales en 2-GPUs con respecto a 1-GPU.

GPUs con respecto a una sola. El incremento de velocidad aumenta con el grado del polinomio y también con respecto a los tamaños de la matriz. Sin embargo, el incremento con el aumento del tamaño de la matriz es menor que el obtenido con el aumento del grado del polinomio. También se observa una forma de diente de sierra en las gráficas. Esto es debido a que la carga de trabajo está desequilibrada, el tiempo de ejecución depende de la paridad del grado. Si el grado es impar una de las dos GPUs realiza una multiplicación de más que la otra.

También presentamos la relación entre el tiempo para calcular las potencias de matrices en el polinomio y el tiempo de evaluación (Figura 4.3). El primer paso posee un coste cúbico mientras que el segundo es cuadrático. La figura muestra claramente la diferencia de tiempos entre ambos pasos. Por último, hemos forzado al sistema a realizar un cálculo maximizando el tamaño de problema y el grado. El tiempo obtenido para evaluar un polinomio de grado 6 con un tamaño de matriz $n = 11000$ fue de 13,50 segundos con una GPU y de 8,43 segundos con dos GPUs.

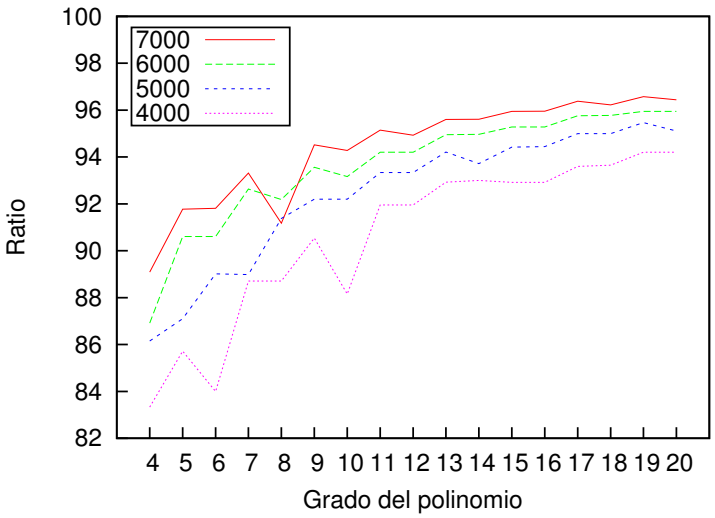


Figura 4.3: Relación entre el cálculo de las potencias de matrices y la evaluación de polinomios matriciales (en porcentaje) en 2-GPUs.

4.4 Un algoritmo paralelo eficiente

Existe una conocida técnica que permite reducir el número de cálculos necesarios (productos de matrices) para evaluar un polinomio como el mostrado en la Ecuación 4.1 [76]. Por concisión, en lo sucesivo llamaremos a esta técnica *encajonamiento*.Demostraremos esta técnica a través de un ejemplo. Supongamos que el grado d del polinomio para evaluar es 14. La Ecuación 4.1 puede expresarse de la siguiente manera usando la propiedad asociativa.

$$
\begin{aligned}
P &= \sum_{i=0}^{14} \alpha_{14-i} X^{14-i} \qquad (4.2)\\
&= \alpha_{14}X^{14} + \alpha_{13}X^{13} + \alpha_{12}X^{12} + \alpha_{11}X^{11} + \alpha_{10}X^{10} + \alpha_9 X^9 + \alpha_8 X^8\\
&+ \alpha_7 X^7 + \alpha_6 X^6 + \alpha_5 X^5 + \alpha_4 X^4 + \alpha_3 X^3 + \alpha_2 X^2 + \alpha_1 X + \alpha_0 I\\
&= X^{12}\left(\alpha_{14}X^2 + \alpha_{13}X + \alpha_{12}I\right) + X^8\left(\alpha_{11}X^3 + \alpha_{10}X^2 + \alpha_9 X + \alpha_8 I\right)\\
&+ X^4\left(\alpha_7 X^3 + \alpha_6 X^2 + \alpha_5 X^1 + \alpha_4 I\right) + \left(\alpha_3 X^3 + \alpha_2 X^2 + \alpha_1 X + \alpha_0 I\right) .
\end{aligned}
$$

Sea $Q^q(\bar{\alpha}, X)$ el polinomio de grado q en X con coeficientes dados por el vector $\bar{\alpha} = \{\alpha_q, \alpha_{q-1}, \ldots, \alpha_1, \alpha_0\}$,

$$Q^q(\bar{\alpha}) = Q^q(\bar{\alpha}, X) = \alpha_q X^q + \alpha_{q-1} X^{q-1} + \cdots + \alpha_1 X + \alpha_0 I \ , \qquad (4.3)$$

entonces el polinomio de la Ecuación 4.2 puede ser reescrito así:

$$\begin{aligned} P &= X^{12} Q^2(\bar{\alpha}_{14:12}) + X^8 Q^3(\bar{\alpha}_{11:8}) + X^4 Q^3(\bar{\alpha}_{7:4}) + Q^3(\bar{\alpha}_{3:0}) \quad (4.4) \\ &= X^4(X^4(X^4(Q^2(\bar{\alpha}_{14:12})) + Q^3(\bar{\alpha}_{11:8})) + Q^3(\bar{\alpha}_{7:4})) + Q^3(\bar{\alpha}_{3:0}) \ . \end{aligned}$$

El ejemplo utiliza un tamaño de "encajonamiento" de $b = 3$, lo que quiere decir que el tamaño del polinomio más grande en la expresión de la Ecuación 4.4 es de 3. El tamaño de encajonamiento b también quiere decir que la potencia $b+1$ de la matriz X (X^4 en el ejemplo) es utilizada como factor común para derivar la Ecuación 4.4. Es fácil observar que el número de operaciones en la Ecuación 4.4 is menor que en la Ecuación 4.1.

El Algoritmo 4.4 muestra el proceso de evaluación de un polinomio utilizando encajonamiento. En las líneas 2-5 se rellena el vector A de tamaño $b+1$ con las potencias de X de manera que

$$A = [X^{i+1}]_{i=0,\ldots,b} = \begin{pmatrix} X & X^2 & X^3 & \ldots & X^{b-1} & X^b & X^{b+1} \end{pmatrix} \ . \quad (4.5)$$

El vector A será utilizado para evaluar polinomios del tipo $Q^q(\bar{\alpha}, X)$ en la expresión de la Ecuación 4.3. Este algoritmo es equivalente al Algoritmo 4.1 dado que ambos resuelven el mismo problema. Una vez que el vector A ha sido calculado, el Algoritmo 4.4 (línea 6) llama a la rutina ENCAJONAMIENTO (Algoritmo 4.5).

El Algoritmo 4.5 es recursivo y evalúa polinomios usando el método que presenta la Ecuación 4.4. Suponiendo que el factor de *encajonamiento* es b, para el caso general, calcula una matriz P_i tal que

$$P_i \leftarrow X^{b+1} P_{b+i+1} + Q^b(\bar{\alpha}_{b+i:i}) = X^{b+1} Q_1 + Q_2 \ , \qquad (4.6)$$

para $i = 0, b+1, 2(b+1), 3(b+1), \ldots$ El caso base de esta recursión es alcanzado cuando $d - i \leq b$, lo que quiere decir que ya no hay posibilidad de realizar más "encajonamiento". En este caso, el algoritmo evalúa el polinomio $Q^q(\bar{\alpha}_{q+i:i})$, siendo $q = d - i$. Se puede demostrar fácilmente que la recursión

Algoritmo 4.4 Algoritmo para la evaluación de un polinomio de matrices utilizando *encajonamiento*.

```
1: function BOX_EVAL( n, X, d, ᾱ, b ) return P
2:     A(0) = X
3:     for i ← 1, b do
4:         A(i) ← A(i − 1) · X
5:     end for
6:     P ← ENCAJONAMIENTO( n, d, b, 0, ᾱ, A )
7: end function
```

Algoritmo 4.5 Algoritmo recursivo para evaluar polinomios de matrices utilizando *encajonamiento*.

```
1: function ENCAJONAMIENTO( n, d, b, i, ᾱ, A ) return P
2:     q ← d − i
3:     if q ≤ b + 1 then
4:         P ← EVAL( n, q, ᾱ_{q+i:i}, A )
5:     else
6:         Q_1 ← ENCAJONAMIENTO( n, d, b, i + b + 1, ᾱ, A )
7:         Q_2 ← EVAL( n, b, ᾱ_{b+i:i}, A )
8:         P ← A(b) · Q_1 + Q_2
9:     end if
10: end function
```

de la Ecuación 4.6 aplicada al ejemplo de la Ecuación 4.4 da

$$\begin{aligned} P = P_0 &= X^4 P_4 + Q^3(\bar{\alpha}_{3:0}) \\ P_4 &= X^4 P_8 + Q^3(\bar{\alpha}_{7:4}) \\ P_8 &= X^4 P_{12} + Q^3(\bar{\alpha}_{11:8}) \\ P_{12} &= Q^2(\bar{\alpha}_{14:12}) \, . \end{aligned}$$

La función recursiva ENCAJONAMIENTO hace uso de otra función llamada EVAL. La función EVAL(n, q, $\bar{\alpha}$, A), mostrada en el Algoritmo 4.6, calcula $Q^q(\bar{\alpha}_{q+i:i})$ siendo que $q \leq b + 1$.

Ahora vamos a desarrollar el coste teórico de los algoritmos en términos de productos de matrices dado que esta es de lejos la operación más

Algoritmo 4.6 Dados las $b+1$ potencias de una matriz X almacenadas en el vector A tal como muestra la Ecuación 4.5, este algoritmo evalúa un polinomio de matrices.

```
1: function EVAL( n, q, ᾱ, A ) return R
2:     R ← α_0 I
3:     for i ← 0, q − 1 do
4:         R ← R + α_{i+1} A(i)
5:     end for
6: end function
```

costosa para evaluar el polinomio. Sea d el grado del polinomio, entonces el Algoritmo 4.1 realiza $d-1$ productos matriciales. Supongamos que b es el factor de *encajonamiento*, esto es, el grado del polinomio de *encajonamiento*, entonces el número de productos necesarios para obtener las potencias de X (Ecuación 4.5) es $b-1$, a las cuáles añadimos una más para obtener X^{b+1}, dando lugar a un total de b productos de matrices (Algoritmo 4.4). Por simplicidad asumimos que $d+1 = k \cdot (b+1)$, siendo k un entero positivo. Entonces, el Algoritmo 4.5 realiza $k-1$ productos matriciales en la recursión (Ecuación 4.6). El número total de productos de matriz realizados por el Algoritmo 4.4 es, entonces,

$$b + k - 1 = b + \frac{d+1}{b+1} - 1 \; . \tag{4.7}$$

Un número mínimo de productos de matriz realizados por el método del *encajonamiento* se deriva de la expresión de la Ecuación 4.7, dando lugar a un valor de b como el entero más cercano a $\sqrt{d+1} - 1$. Utilizando este valor, el *speedup* alcanzable puede ser calculado como

$$\text{speedup} = \frac{d-1}{2\left(\sqrt{d+1}-1\right)} \; ,$$

y no es difícil demostrar que $\frac{\sqrt{d+1}}{2}$ es una cota inferior para el *speedup* teórico, para $d \geq 3$.

El *speedup* del Algoritmo 4.2 para 2-GPUs crece con el grado del polinomio (Figura 4.2). Cuando se utiliza el *encajonamiento*, el grado b del polinomio de *encajonamiento* suele ser pequeño, limitando así la oportunidad de acelerar la aplicación. Por este motivo, proponemos una aproximación

diferente al problema de incrementar las prestaciones de la evaluación de polinomios en dos GPUs. Nuestra solución está basada en el particionado de las matrices con la intención de que las dos GPUs participen en la multiplicación de cada par de matrices.

Considérense las siguiente particiones:

$$\begin{aligned} X^{p-1} &= \begin{pmatrix} X_1 & X_2 \end{pmatrix}, \\ X^{p} &= \begin{pmatrix} Y_1 & Y_2 \end{pmatrix}, \end{aligned}$$

donde $X_1, Y_1, X_2, Y_2 \in \mathbb{R}^{n\times(n/2)}$ (se asume por simplicidad que n es par), entonces

$$X^p = X \cdot X^{p-1} = X \cdot \begin{pmatrix} X_1 & X_2 \end{pmatrix} = \begin{pmatrix} X \cdot X_1 & X \cdot X_2 \end{pmatrix} = \begin{pmatrix} Y_1 & Y_2 \end{pmatrix},$$

probando que el cálculo de cada potencia p de X puede ser llevado a cabo realizando dos productos independientes de matrices, siempre y cuando cada GPU tenga toda la matriz X y las dos mitades de la partición de la potencia $p-1$ de X. El Algoritmo 4.7 realiza el cálculo de las potencias de X utilizando esta partición de las matrices. Al principio (línea 5), las dos GPUs reciben toda la matriz X del *host* (esta matriz se denota por hX). El símbolo $\Leftarrow$ representa transferencias *host*-GPU, GPU-*host* o entre GPUs distintas. Una mitad del factor X es copiada en la primera matriz del vector, esto es, $A(0)$ (línea 6). A continuación, por medio del bucle de las líneas 7 a 9, el algoritmo calcula una mitad de cada potencia de X. La GPU0 calcula las primeras $n/2$ columnas de estos factores y las almacena en las matrices $A(i) \in \mathbb{R}^{n\times n/2}$, $i = 0, \ldots, b-1$. De manera similar, la GPU1 almacena las $n/2 + mod(n,2)$ columnas desde la $n/2$ a la $n-1$ de las potencias de matrices correspondientes $A(i) \in \mathbb{R}^{n\times n/2+mod(n,2)}$.

Para evaluar los polinomios de *encajonamiento* de grado b es necesario también que las dos GPUs tengan todo el factor X^{b+1}. Cada GPU ha calculado solo una mitad de este factor que ha sido almacenado en la correpondiente mitad de B (línea 10). Ahora se necesitan intercambiar estas partes del factor B de manera que, al final, las dos GPUs tengan el factor X^{b+1} almacenado en B. En la línea 11, la GPU g envía las columnas $i_g : j_g$ a la otra GPU. El índice de la otra GPU es identificado por $\bar{g}$, donde $\bar{g} = 1 - g$. La matriz $\bar{B}$ se refiere a la matriz B en la otra GPU. Hemos utilizado la rutina CUDA `cudaMemcpyPeerAsync` para realizar esta comuni-

Algoritmo 4.7 Algoritmo para calcular las $b+1$ potencias de la matriz X en dos GPUs.

1: **function** COMPUTE_POWERS2(n, b, hX) **return** (A, X, B)
2: $i_0 = 0$, $j_0 = n/2 - 1$, $i_1 = n/2$, $j_1 = n - 1$
3: `#pragma omp parallel for`
4: **for** $g \leftarrow 0, 1$ **do**
5: $X \Leftarrow hX$
6: $A(0) \leftarrow X(:, i_g : j_g)$
7: **for** $k \leftarrow 1, b-1$ **do**
8: $A(k) \leftarrow X \cdot A(k-1)$
9: **end for**
10: $B(:, i_g : j_g) \leftarrow X \cdot A(b-1)$
11: $B(:, i_{\bar{g}} : j_{\bar{g}}) \Leftarrow \bar{B}(:, i_{\bar{g}} : j_{\bar{g}})$
12: **end for**
13: **end function**

cación entre iguales (peer-to-peer) entre las dos GPUs. Esta función permite realizar copias de memoria entre dos dispositivos diferentes evitando el almacenamiento intermedio en la memoria del *host*. La rutina no comienza hasta que todas las instrucciones emitidas previamente a cada dispositivo se hayan completado, siendo a su vez asíncrona con respecto al *host*.

El algoritmo para evaluar el polinomio (Ecuación 4.1) utilizando *encajonamiento* en dos GPUs está descrito en el Algoritmo 4.8. Antes de la utilización de este algoritmo se supone que ya se ha ejecutado el Algoritmo 4.7, cuya función COMPUTE_POWERS2 ha permitido calcular las potencias 1 a $b+1$ de X, y éstas están almacenadas en ambas GPUs. Los argumentos A y B son factores previamente almacenados en la memoria del dispositivo. El argumento i del Algoritmo 4.8 es utilizado para indexar el vector $\bar{\alpha}$ y también para controlar la recursión. La matriz resultado P, que contiene la solución del polinomio, es una matriz local a cada GPU de $n/2$ columnas (o $n/2+1$ en el caso de la GPU1 si n es impar). Asimismo, las matrices auxiliares Q_1 y Q_2 poseen el mismo tamaño que P. Éstas son utilizadas para realizar el producto de la línea 11, que corresponde a la operación mostrada en la recursión descrita por la Ecuación 4.6. La Figura 4.4 muestra gráficamente la disposición de este producto entre las dos GPUs. Todo el proceso de evaluación es llevado a cabo en paralelo entre ambos dispositivos sin que medie ninguna comunicación. Solo al terminar, el sistema CPU recibe

Algoritmo 4.8 Algoritmo recursivo para evaluar un polinomio de matrices utilizando *encajonamiento* en dos GPUs.

```
1: function ENCAJONAMIENTO2( n, d, b, i, ᾱ, A, B ) return P
2:     #pragma omp parallel for
3:     for g ← 0, 1 do
4:         q ← d − i
5:         if q ≤ b + 1 then
6:             P ← EVAL( q, ᾱ_{q+i:i}, A )                    ▷ Algoritmo 4.6
7:         else
8:             Q_1 ← ENCAJONAMIENTO2( n, d, b, i + b + 1, ᾱ, A, B )
9:             Q_2 ← EVAL( n, b, ᾱ_{b+i:i}, A )               ▷ Algoritmo 4.6
10:            P ← B · Q_1 + Q_2
11:        end if
12:    end for
13: end function
```

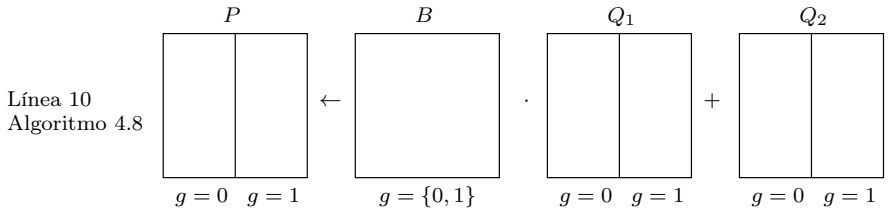


Figura 4.4: Multiplicación de 2 matrices en 2 GPUs.

los fatores P de ambas GPUs para contruir la matriz cuadrada final que representa la solución al polinomio. Esto se efectúa después de la ejecución del Algoritmo 4.8.

4.5 Resultados de la evaluación utilizando *encajonamiento*

A continuación se muestran los diferentes aspectos del comportamiento del algoritmo utilizando *encajonamiento*. Primero, en la Figura 4.5 se traza la evolución del tiempo en relación con el factor de *encajonamiento* b utilizado.

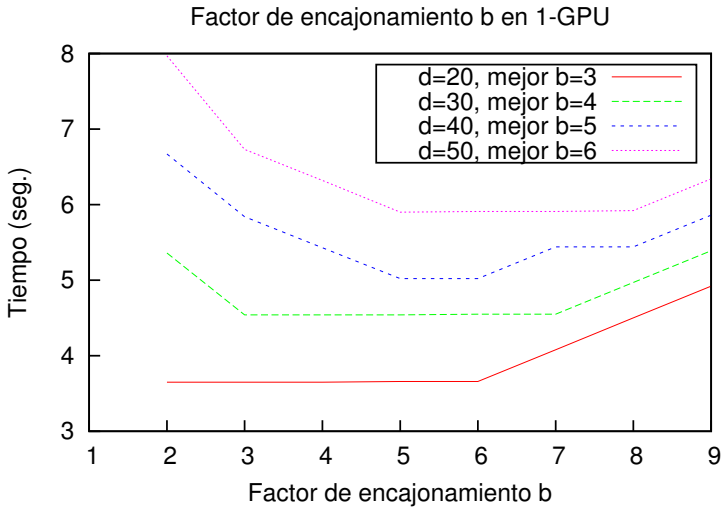


Figura 4.5: Gráficas de tiempo de ejecución para la evaluación de un polinomio con tamaño de matriz $n = 6000$ para distintos grados de polinomio en función del factor de *encajonamiento* b en 1-GPU.

Según los experimentos se comprueba que el valor $b = \lfloor\sqrt{d+1}\rfloor - 1$ obtenido a través de la derivación teórica es la mejor opción en caso de 1-GPU. Analizamos el comportamiento de las dos GPUs para ver si el producto matricial puede dar lugar a un resultado diferente. El tiempo para obtener la matriz resultado P se divide en dos pasos: el cálculo de las potencias de la matriz X (términos del polinomio $Q^b(\bar{\alpha})$ – Ecuación 4.3), y la evaluación del polinomio utilizando la función recursiva ENCAJONAMIENTO2 descrita en el Algoritmo 4.8. Este tiempo se muestra en la Figura 4.6. El cómputo de las potencias crece linealmente con el factor de *encajonamiento* b y es exactamente el mismo para todos los grados de polinomio puesto que solo depende de b. La carga de trabajo a realizar para evaluar el polinomio mediante el Algoritmo 4.8 se reduce a medida que b crece dado que el número de términos P_i en la recursión (Ecuación 4.6) decrece. El mejor factor de *encajonamiento* está alrededor del cruce entre las líneas que muestra la Figura 4.6. La Figura 4.7, que muestra el tiempo total para la evaluación del polinomio en dos GPUs, tiene el mismo aspecto que la Figura 4.5 en una GPU.

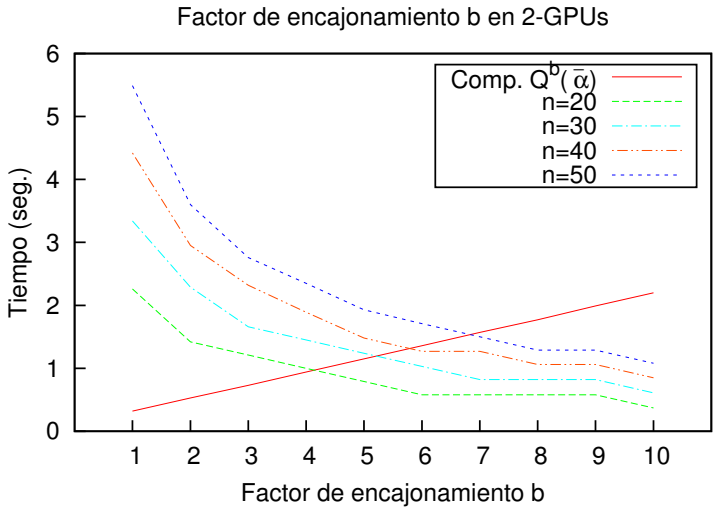


Figura 4.6: Gráficas de tiempo de cálculo de potencias de X y evaluación de un polinomio con tamaño de matriz $n = 6000$ para distintos grados de polinomio en función del factor de *encajonamiento* b en 2-GPUs.

El siguiente experimento muestra el *speedup* del algoritmo utilizando *encajonamiento*. La Figura 4.8 muestra el incremento en velocidad alcanzado por el uso de las dos GPUs y cómo esta mejora crece a su vez con el tamaño del problema gracias a la paralelización de la multiplicación de matrices. El grado del polinomio no infiere una gran diferencia en el incremento de velocidad debido al pequeño peso que tienen las comunicaciones *host*-GPU con respecto al peso de la computación.

4.6 Conclusiones

En este capítulo hemos estudiado el problema de evaluar polinomios de matrices en computadores modernos de cálculo intensivo que disponen de aceleradores hardware. Nuestra motivación surge de la ausencia de ningún software y/o contribución que trate este problema y de su utilidad en el contexto descrito en el capítulo anterior, donde se generaba un polinomio

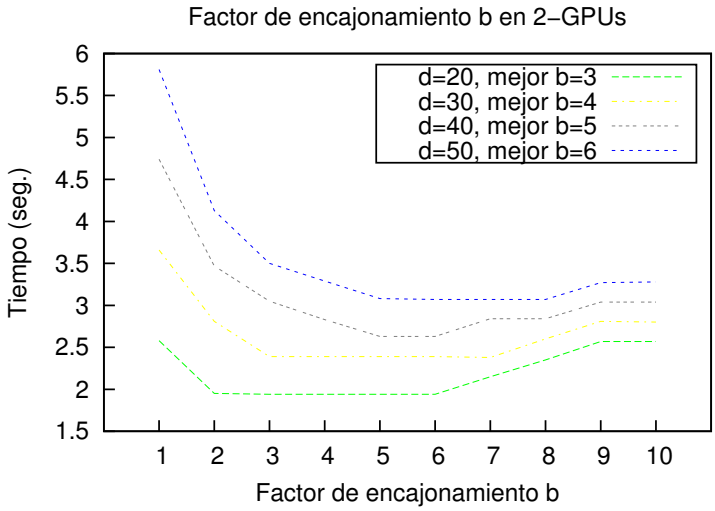


Figura 4.7: Gráficas de tiempo de ejecución para la evaluación de un polinomio con tamaño de matriz $n = 6000$ para distintos grados de polinomio en función del factor de *encajonamiento* b en 2-GPU.

matricial para representar el relieve de una superficie. En nuestro caso hemos trabajado con GPUs de NVIDIA pero los resultados también son aplicables a otros tipos de coprocesadores. En esencia, el algoritmo propuesto es sencillo y se basa en la existencia de una rutina de multiplicación de matrices eficiente para ese hardware. El resto de propuestas realizadas en el capítulo no necesitan ser cambiadas.

En primer lugar hemos planteado una posible solución sobre la que trabajar y poder realizar comparaciones. Esta solución se basa en distribuir el cómputo de las potencias de la matriz entre las GPUs. La propuesta funciona bien en el caso de que el grado del polinomio sea alto y no se utilice *encajonamiento*. Sin embargo, la solución más útil es la segunda, donde el paralelismo se obtiene del particionado de la matriz para dividir este producto entre las GPUs. Esta solución funciona bien cuanto mayor es el tamaño de la matriz y es independiente del grado del polinomio. El algoritmo propuesto para la evaluación de polinomios matriciales es sencillo gracias a su carácter recursivo y utiliza lo que hemos decidido llamar co-

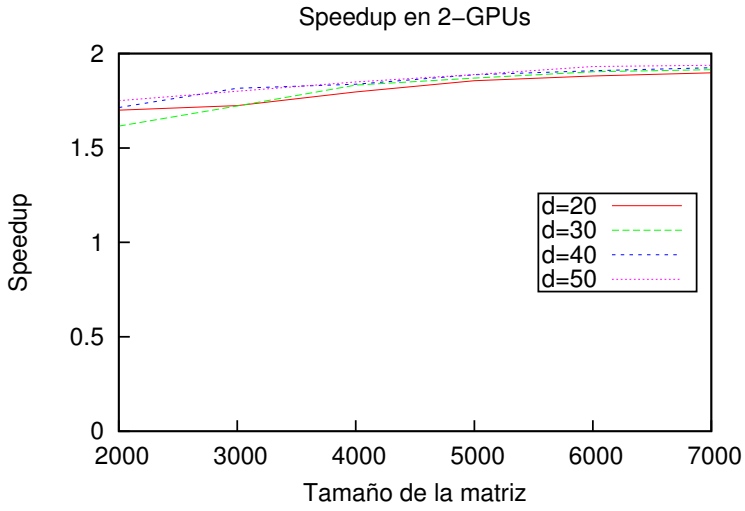


Figura 4.8: Gráficas con el *speedup* obtenido por 2-GPU con respecto a 1-GPU para la evaluación de un polinomio con tamaño fijo de matriz $n = 6000$ variando el tamaño de la matriz y en función del grado del polinomio, utilizando un factor de *encajonamiento* óptimo.

mo *encajonamiento*, una técnica que reduce el número de multiplicaciones necesarias para la evaluación.

Destacamos el método propuesto para implementar los algoritmos por su sencillez y efectividad. El método se resume en la utilización de hilos OpenMP generados en bucles paralelos donde cada iteración es ejecutada por una GPU. El código escrito sirve para las dos GPUs (o más GPUs si existiesen en el sistema). Esta solución se puede utilizar bajo la especificación v3.0 de OpenMP o superior para poder utilizar la directiva `threadprivate`.

Variables meteorológicas del clima 5

En este capítulo trabajamos con otro problema relacionado con la producción agrícola en el área del Rio São Francisco, aunque las soluciones y resultados estudiados en esta tesis son extrapolables a otras zonas agrícolas. Trabajamos con una metodología diseñada en particular para representar variables meteorológicas del clima en esta zona. Estas variables, entre las que se encuentra por ejemplo la temperatura, son conocidas gracias a sensores estratégicamente situados a lo largo y ancho del área bajo estudio. Sin embargo, no es posible conocer el valor de estas variables en cualquier punto por este medio debido a la poca densidad del número de sensores. Mediante un método de interpolación espacial basado en el inverso de la potencia de la distancia es posible inferir potencialmente el valor de una de estas variables en cualquier punto del terreno. Este método puede necesitar de una gran capacidad computacional, siendo además crítico el tiempo de respuesta. Nosotros hemos optimizado el procedimiento de cálculo utilizando computación paralela heterogénea, es decir, utilizando multicores y GPUs.

5.1 Introducción

El alto coste computacional de una simulación climática, realizada con un modelo numérico que reproduzca el comportamiento, supone un importante reto científico. Este coste puede limitar la resolución espacial de las simulaciones y obligar a la búsqueda constante de un equilibrio entre la longitud del período simulado, los recursos computacionales con los que se cuenta, y el tiempo de ejecución de que se disponga [77]. Estas limitaciones computacionales se van superando a medida que avanza la potencia computacional de los nodos de cómputo. Sin embargo, desde el punto de vista del programador de HPC, sigue presente el reto de extraer el máximo de las máquinas modernas que frecuentemente basan su potencia computacional en la heterogeneidad de los aceleradores para resolver este problema.

El Instituto Nacional de Meteorología Brasileño (INMET) [78] mantiene una red de estaciones meteorológicas en todo el país. De todas ellas, 24 concretamente están en situadas en la región agrícola del Valle del Rio São Francisco. Los datos obtenidos por las mismas son de alta calidad, pero su tratamiento no es trivial, es necesario procesarlos adecuadamente. En particular, el clima tropical existente en la zona es muy variable y requiere de un cuidadoso tratamiento. Además, el mantenimiento de las estaciones meteorológicas es muy caro. Esto hace difícil que se puedan instalar nuevas estaciones meteorológicas en todos lo sitios. La solución pasa por inferir datos meteorológicos en puntos donde no hay estaciones. Así pues, las simulaciones climáticas de alta resolución se han convertido en una herramienta muy importante y de gran impacto, entre otras cosas, por el impacto que tiene su estudio en temas como el cambio climático global.

El problema que nos ocupa en este caso de estudio es el alto coste computacional al cálculo necesario para estimar las variables meteorológicas que representan el clima. El método de interpolación espacial [79] utilizado requiere una potencia de cálculo considerable para tratar con un conjunto muy grande de datos. La aparición de nuevas tecnologías para la computación de alto rendimiento, junto con la llegada de arquitecturas multicore y multi-GPU, aceleran la solución de problemas complejos de alto coste computacional. En este escenario, el objetivo general de este capítulo consiste en acelerar los cálculos necesarios para realizar estas simulaciones climáticas basadas en la representación de variables meteorológicas por el método de interpolación espacial del inverso de la potencia de la distancia.

En la sección siguiente se explica el modelo matemático en el que se basa la representación de las variables meteorológicas. En particular, en la Sección 5.3 se analiza el comportamiento del algoritmo secuencial que trabaja con la temperatura como variable meteorológica de ejemplo. En base a este análisis se plantea una solución paralela para GPUs (kernel CUDA) y una paralelización completa heterogénea en el mismo apartado. Los resultados experimentales de nuestra solución se muestran en la Sección 5.4. El capítulo termina con una sección de resultados.

5.2 Representación de variables meteorológicas del clima

El modelo matemático para la representación de variables meteorológicas es un procedimiento para calcular el valor de un atributo en zonas donde no hay muestras de variables meteorológicas, estando estos atributos situados en la misma región. El método aplicado de ponderación espacial del Inverso de la Potencia de la Distancia (IPD) es un estimador de carácter determinista, con el cual se obtiene un valor interpolado a partir de una combinación lineal de los valores de los puntos de muestreo, ponderados por el método de interpolación espacial del inverso de la potencia de la distancia (véase Figura 5.1). El método considera, pues, que los puntos próximos a los lugares no incluidos en la muestra son más representativos que los más lejanos. Los pesos se asignan en proporción al inverso de la distancia a los valores de potencia de los puntos estimados y reales [80, 81].

Método de interpolación espacial del Inverso de la Potencia de la Distancia

El IPD es un método de interpolación espacial. La intención del método no es la de caracterizar completamente un fenómeno físico por medio del conjunto de factores que están en su origen. Su objetivo es la interpolación de los valores observados [79]. La interpolación espacial convierte los datos de observaciones puntuales en campos continuos, produciendo patrones espaciales que se pueden comparar con otras entidades espaciales.

La forma general de encontrar un valor interpolado para un punto dado

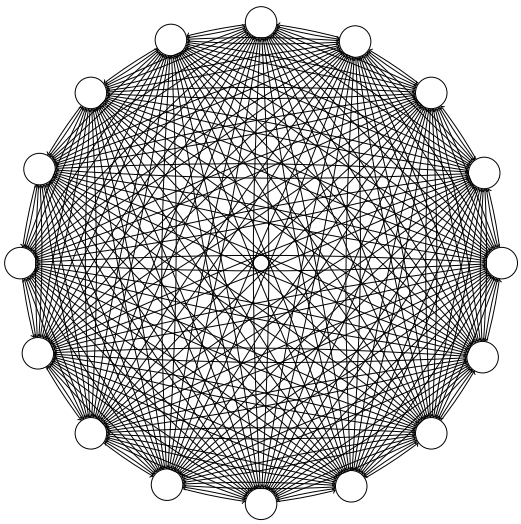


Figura 5.1: Esquema de la representación de variables meteorológicas del clima, considerando los círculos las estaciones meteorológicas y los cruces de las rectas las combinaciones lineales de los valores de los puntos de muestreo.

r utilizando el IPD está representada en la siguiente ecuación:

$$f_e(r) = \frac{\sum_{i=1}^{n} d(r, r_i)^{-p}(f_m(r_i))}{\sum_{i=1}^{n} d(r, r_i)^{-p}} \ , \tag{5.1}$$

donde la descripción de los parámetros puede encontrarse en la Tabla 5.1.

El principio en que se basa el IPD es simple y consiste en que, en media, los valores de un atributo tienden a ser similares en los lugares más cercanos que en los más distantes. Para predecir un valor en cualquier lugar, el IPD debe utilizar los valores medidos en su entorno, que tendrán un peso mayor que los valores más distantes, o sea, cada punto tiene una influencia en el nuevo punto que disminuye a medida que aumenta la distancia. Este concepto también tiene como base las relaciones espaciales entre los fenómenos geográficos y la utilización de correlación espacial como un promedio de la diferencia entre los atributos estimados [79].

Se ha realizado una revisión de la literatura relacionada más actual pa-

Tabla 5.1: Descripción de los parámetros de la Ecuación 5.1 de la interpolación espacial del Inverso de la Potencia de la Distancia (IPD).

Símbolo	**Descripción**
$f_e(r)$	Valor estimado de f_e en el vector posición r
$f_m(r_i)$	Valor medido de f_m en el vector posición r_i
n	Número total de puntos conocidos y utilizados
$d(r, r_i)$	Distancia euclidiana entre los vectores r y r_i
p	Parámetro de potencia

ra contextualizar los trabajos realizados en el modelado de problemas de representación de variables meteorológicas del clima. Por ejemplo en [82], la interpolación IPD se aplica a las zonas de fuerte declive en un río, demostrando la eficacia del método para las variables espaciales de temperatura y humedad relativa. Los resultados también muestran que la representación de estas variables es independiente de la influencia de la estacionalidad, de los efectos de la latitud y la "continentalidad", y de las escalas espaciales y temporales consideradas. En [83] los autores presentan un método de evaluación de rendimiento aplicados a la precipitación de datos espaciales en un área específica situada en Brasil. En este trabajo se trabaja con 25 millones de mediciones obtenidas de las estaciones, siendo recomendado el uso de la interpolación ya que con este método se mostraba una varianza más baja y un menor margen de error.

Uno de los métodos más estudiados en la literatura para evaluar la representación es el proceso de *validación cruzada* [84]. Este método se basa en eliminar un conjunto de datos de la muestra, utilizando un estimador ponderado en función de la distancia. Las estimaciones del valor surgen con la cantidad retirada de las muestras restantes. Ellos son, por lo tanto, dos valores al mismo punto, el real y el estimado. Este procedimiento se repite para todas las muestras disponibles. La validación cruzada es un procedimiento estadístico libre de distribución teórica, que tiene un alto rendimiento en la aplicación de los métodos de representación. Este método nos permite comparar teóricamente varios modelos y valores empíricos obtenidos en el muestreo. Con base en el análisis de los errores de estimación podemos seleccionar el mejor modelo [85].

A partir de los trabajos anteriores se fue mejorando la técnica de tal forma que se encontraron cantidades óptimas de variables meteorológicas

del clima, reduciendo la cantidad de la muestra a partir del conjunto de datos temporal. Luego se estimó a partir de la muestra restante. Una vez que se realiza la estimación se puede comparar con el valor de la muestra que se eliminó inicialmente a partir del conjunto de datos. Este procedimiento se repite para todas las muestras disponibles. Esto puede ser visto como un experimento en el que se imita el proceso de estimación por el supuesto de que nunca se toma una muestra en esa ubicación [84]. En [86] se estudian errores en las temperaturas interpoladas por el Inverso del Cuadrado de la Distancia (IQD) obtenido a partir de 10 años de observaciones mensuales en 1807 estaciones meteorológicas. A partir del histórico mundial, se ha observado que el Error Absoluto Promedio (EAP) es la medida más natural de la media de los errores de una interpolación espacial:

$$EAP = \frac{1}{n} \sum_{i=1}^{n} |f_m(r_j) - f_e(r_j)| \,. \tag{5.2}$$

Por lo tanto, la evaluación dimensional de los valores de la interpolación se basan en el índice EAP (Ecuación 5.2). A través de estos estudios encontrados en la literatura, nosotros utilizamos este índice para medir la fiabilidad de los resultados a partir de la muestra de experimentación en nuestro problema.

Análisis del orden de magnitud del problema

Sin entrar demasiado en los detalles de la elección de la configuración física del modelo ni en la base de datos empleada como condición inicial y de contorno (ya que esto queda fuera del ámbito de este trabajo), la representación de variables meteorológicas del clima que hemos tratado posee las siguientes características para el dominio de experimentación:

- ◇ Tiene un formato de círculo de alrededor de 250 km de radio con proximidad a Petrolina, que es la ciudad de referencia, y está situado en la provincia brasileña de Pernambuco. Esta región es conocida como *Semi-árido Nordestino* e incluye la región agrícola del Valle del Rio São Francisco (Figura 5.2).

- ◇ Fueron utilizados valores de datos diarios de variables meteorológicas del clima en 18 estaciones meteorológicas del INMET que operan en este área, tomados a partir del segundo semestre del año de 2010 y

Figura 5.2: Representación de algunas de las estaciones meteorológicas en relación a la estación de referencia (Petrolina).

primer semestre de 2011.

- ◇ En la obtención del tiempo de ejecución no se ha contemplado el tiempo empleado en el pre y posprocesado de los datos. Para aproximar el problema con los dominios de la versión secuencial del modelo, se ha necesitado el tiempo aproximado de un día entero de ejecución para simular 1 año empleando 1 solo procesador (Intel Pentium M 755 Dothan, con 512 Mb DDR y Ati Mobility Radeon 9700 128 Mb). Estos datos fueron utilizados para validar el parámetro de potencia p (Ecuación 5.1) obtenido para un período de 1 año.

- ◇ Los valores de variables meteorológicas del clima fueron obtenidos por medio del IPD (Ecuación 5.1) variando el parámetro de potencia p de 25 hasta 0, decreciendo en pasos de valor 0.1. El IPD fue determinado utilizando los valores de EAP (Ecuación 5.2) calculados mediante la validación cruzada aplicada a los datos de los períodos. En esta etapa fueron obtenidos los valores de p de las épocas del año: invierno, primavera, verano y otoño, para las estaciones meteorológicas utilizadas y localizadas en el Valle del Rio São Francisco que se pueden ver en la Tabla 5.2.

Tabla 5.2: Tabla con los datos meteorológicos: Estaciones meteorológicas, provincias brasileñas (UF), longitudes, latitudes, altitudes y distancias en relación a la estación meteorológica de referencia (Petrolina).

Estaciones	UF	Longitudes (°)	Latitudes (°)	Altitudes (m)	Distancias (km)
Arcoverde	PE	-37.0314	-8.2508	522.0	459.0
Afranio	PE	-41.0018	-8.3054	522.0	107.0
Cabrobó	PE	-39.3144	-8.5036	342.0	191.6
Canudos	BA	-39.0135	-9.5348	402.0	198.2
Delfino	BA	-41.2072	-10.4553	637.0	127.3
Euclides da Cunha	BA	-38.9978	-10.5363	432.0	237.5
Floresta	PE	-38.5922	-8.6103	329.0	259.7
Jacobina	BA	-11.2050	-40.4653	453.0	237.5
Oeiras	PI	-6.9742	-42.1469	156.0	205.6
Paulistana	PI	-41.1428	-8.1325	374.0	144.2
Paulo Afonso	BA	-38.1319	-9.2428	374.0	243.0
Petrolina	PE	-40.8000	-9.3833	370.5	0
Picos	PI	-7.0708	-41.4042	233.0	221.0
Remanso	BA	-42.0831	-9.6189	401.0	144.8
São João do Piauí	PI	-42.1449	-8.2128	222.0	279.8
São Raimundo Nonato	PI	-42.4156	-9.0054	222.0	124.0
Senhor do Bonfim	BA	-10.4442	-40.1469	548.0	138.3
Uauá	BA	-39.4956	-9.8336	453.0	153.2

5.3 Análisis de rendimiento y optimización

Antes de iniciar la optimización del algoritmo de representación de variables meteorológicas con los modelos paralelos de computación se hace necesario conocer el comportamiento del código secuencial, identificando las partes que contienen los costes computacionales más altos para aplicar ahí las optimizaciones. El Algoritmo 5.1 muestra la parte de código extraída del total sobre la que se realizarán las optimizaciones.

El algoritmo secuencial aplica interpolación espacial a las variables del clima (temperatura, humedad relativa del aire, velocidad del viento, radiación solar, precipitación y la presión atmosférica) para valores diarios, mensuales y anuales de datos recogidos en las estaciones meteorológicas.

Hemos recopilado tiempos de ejecución de la aplicación y, mediante *profiling* llevado a cabo con la herramienta `gprof` [87], hemos sacado conclusiones apropiadas acerca del lugar en el que optimizar. Como muestra se da parte de la ejecución con `gprof`:

```
Flat profile:
  %   cumulative   self
 time   seconds   seconds  name
 30.46   2814.31  2814.31  clCombina::interpola()
 10.56   5510.12  2695.81  std::vector<clEstacao>
 12.23   7114.38  1604.26  sqlite3VdbeExec
  6.24   7932.07   817.69  std::dates<clEstacao>
  4.17   8478.42   546.34  sqlite3BtreeMovetoUnpacked
 ...
```

Se puede comprobar que el punto con el coste computacional más grande está en el método `clCombina::interpola`, que tiene como principal funcionalidad aplicar el método de interpolación espacial del inverso de la potencia de la distancia, representando hasta el 30 % del total del tiempo de ejecución.

El modelo algorítmico desarrollado permite que el método IPD interpole los valores a partir de una combinación lineal para los puntos del muestreo. El algoritmo consiste en dividir la computación entre varios recursos computacionales. Estos puntos se refieren a variables del clima contextualizados anteriormente para la construcción del modelo computacional propuesto.

Algoritmo 5.1 Rutina secuencial para la representación de variables meteorológicas del clima.

```
1:  double clCombina::interpola( int e, double p ) {
2:    double d, den = 0.0, num = 0.0;
3:
4:   for( unsigned j = 0; j < Estacao.size(); ++j ) {
5:    if(( j != e )
6:     && (Estacao[e].dia == Estacao[j].dia)
7:     && (Estacao[e].mes == Estacao[j].mes)
8:     && (Estacao[e].ano == Estacao[j].ano)) {
10:    d = de(Estacao[e].x,Estacao[e].y,Estacao[j].x,Estacao[j].y,p);
11:    if( d > 0.0 ) {
12:     num += Estacao[j].Vm/d;
13:     den += 1.0/d;
14:    }
15:   }
16:  }
17:  if( den > 0.0 ) return num/den;
18:   else return 0.0;
19: }
20: bool clCombina::processa( double p, int nthreads ) {
21:  while( p < 25 ) {
22:   int    n   = 0, e;
23:   double sum = 0.0, EAP;
24:
25:   for( e = 0; e < Estacao.size(); ++e ) {
26:    sum += fabs( Estacao[e].Vm-interpola(e) );
27:    ++n;
28:   }
29:   EAP  = sum/n; avg += EAP; ++m;
30:   if( MAEn > EAP ){ MAEn = EAP; pmin = p; }
31:   if( MAEx < EAP ){ MAEx = EAP; pmax = p; }
32:   p += 0.25;
33:  }
34: }
```

La construcción de la combinación lineal de los puntos de muestreo es la parte más costosa del proceso global lo que ofrece una buena oportunidad para aplicar paralelismo. El método IPD está representado por una matriz `A`, y todos los elementos de la matriz se pueden calcular de forma independiente y simultánea. Esto se puede realizar mediante la partición de la suma en trozos de diferentes tamaños. Nuestro programa paralelo se basa en este enfoque ya que el valor habitual para el orden de la suma r varía entre 1 y

n, donde n en nuestros experimentos posee un valor de aproximadamente 13 millones de términos por período de experimentación. Dividimos, pues, la suma en trozos, donde cada trozo posee un tamaño determinado que se calcula en función de la potencia de cómputo del recurso computacional elegido para ejecutar ese trozo. El algoritmo básico es muy parecido al mostrado en el Capítulo 3 (Algoritmo 3.2), habiendo sido convenientemente adaptado al problema actual. De la misma forma que antes, el algoritmo muestra el esquema utilizado para la partición de carga de trabajo entre CPU y GPU.

La construcción de la matriz a través de un kernel CUDA resulta bastante natural gracias al paralelismo intrínseco al cálculo. Se aprovecha tanto el hecho de que todos los elementos de matriz se pueden calcular al mismo tiempo como que cada término de la suma es independiente de cualquier otro. Con el fin de utilizar toda esta concurrencia se ha creado un *grid* bidimensional para los bloques de *threads*. El bloque de tamaño `BLOCK_SIZE_X`×`BLOCK_SIZE_Y` está definido por la macros de las líneas 1 y 2 del Algoritmo 5.2. Cada hilo se encuentra en el bloque a través de 2 coordenadas que están representados por las variables `X` y `Y` (líneas 4 y 5). Los bloques de hilos están dispuestos en un conjunto de dos dimensiones. La dimensión es de $\lceil$`n/BLOCK_SIZE_X`$\rceil$ y $\lceil$`n/BLOCK_SIZE_Y`$\rceil$, respectivamente, siendo `n` la dimensión de la matriz A. El siguiente código está dentro de la rutina `matrixGPU` y muestra la disposición y la llamada al kernel:

```
1: dim3 dimGrid( idiv( n, BLOCK_SIZE_X ), 1 );
2: dim3 dimBlock( BLOCK_SIZE_X, BLOCK_SIZE_Y );
3: compute_kernel<<< dimGrid, dimBlock >>>( dEstation, dv, p, n );
```

donde `idiv` es una rutina que devuelve la longitud entera de las dos últimas dimensiones. El objetivo es que todos los *threads* puedan calcular simultáneamente las instrucciones asignadas. El *thread* con las coordenadas apropiadas es designado para calcular los términos de la suma `X`+j, con $j = 0$:`BLOCK_SIZE_Y`: n. Esta operación se especifica por el bucle que cierra en la línea 46. Los exponentes `sh_e_day`, `sh_e_year` y `e_y` dependen de la fila (`Estation`) y columna (`j`) indexadas en la matriz.

Algoritmo 5.2 Kernel CUDA para el cálculo del IPD.

```
1:   #define BLOCK_SIZE_X 32
2:   #define BLOCK_SIZE_Y 32
3:   __global__ void kernel(clEstation *Estation, double *v, p, int n) {
4:   int X = threadIdx.x;
5:   int Y = threadIdx.y;
6:   int e = blockIdx.x * BLOCK_SIZE_X + X;
7:   if( e < n ) {
8:     if( Y == 0 ) {
9:       sh_e_day[X]   = Estation[e].day;
10:      sh_e_month[X] = Estation[e].month;
11:      sh_e_year[X]  = Estation[e].year;
12       sh_e_x[X]     = Estation[e].x;
13:      sh_e_y[X]     = Estation[e].y;
14:      sh_e_Vm[X]    = Estation[e].Vm;
15:    }
16:    __syncthreads();
17:    day_e    = sh_e_day[X];
18:    month_e  = sh_e_month[X];
19:    year_e   = sh_e_year[X];
20:    x_e      = sh_e_x[X];
21:    y_e      = sh_e_y[X];
22:    Vm_e     = sh_e_Vm[X];
23:    for( int j = 0; j < n; j += BLOCK_SIZE_Y ) {
24:      if( j+Y < n ) {
25:        if( X == 0 ) {
26:          sh_j_day[Y]   = Estation[j+Y].day;
27:          sh_j_month[Y] = Estation[j+Y].month;
28:          sh_j_year[Y]  = Estation[j+Y].year;
29:          sh_j_x[Y]     = Estation[j+Y].x;
30:          sh_j_y[Y]     = Estation[j+Y].y;
31:          sh_j_Vm[Y]    = Estation[j+Y].Vm;
32:        }
33:        __syncthreads();
34:        day_j   = sh_j_day[Y];
35:        month_j = sh_j_month[Y];
36:        year_j  = sh_j_year[Y];
37:        x_j     = sh_j_x[Y];
38:        y_j     = sh_j_y[Y];
39:        Vm_j    = sh_j_Vm[Y];
40:        if((day_e==day_j)&&(month_e==month_j)&&(year_e==year_j)) {
41:          d = device_de( x_e, y_e, x_j, y_j, p );
42:          if( d > 0.0 )
43:            num += ( Vm_j / d ); den += ( 1.0 / d );
44:        }
45:      }
46:    }
47:  }
48:}
```

5.4 Resultados experimentales

En esta sección se van a mostrar diversos experimentos mediante los cuales se estudia el comportamiento del algoritmo propuesto en este capítulo. Se pretende analizar resultados obtenidos y estudiar los costes de las diferentes partes de dichos algoritmos así como sus optimizaciones. Se incluyen

Tabla 5.3: Tabla con el EAP y la p para la representación de la temperatura.

Períodos	EAP	p
Invierno	1.16	4.6
Primavera	0.94	4.8
Verano	0.97	3.3
Otoño	0.94	3.6

Tabla 5.4: Tabla con los datos de las estaciones meteorológicas: Períodos del año, temperaturas en grados Celsius (°C): mínima (T_{Min}), promedio (T_{Pro}), máxima (T_{Max}), varianza (σ^2) y la cantidad de datos (n).

Períodos	T_{Min}	T_{Pro}	T_{Max}	σ^2	n
Invierno	17.6	23.6	30.7	2.47	13 millones
Primavera	21.6	27.4	33.1	2.25	13 millones
Verano	21.4	26.7	30.9	1.65	13 millones
Otoño	19.9	24.9	31.0	1.91	13 millones

estudios experimentales, explicaciones y comentarios detallados de la optimización para el entorno de ejecución MICROMACHIN (Sección 2.2).

Análisis de la representación de la temperatura

Con el fin de presentar los resultados obtenidos de la optimización del modelo de representación de variables meteorológicas del clima, se muestran los valores simulados para la variable *temperatura* en esta sección. Se utilizaron valores periódicos medidos en las estaciones meteorológicas. Los índices: EAP y p calculados se pueden ver en la Tabla 5.3. Mientras que los valores de temperatura en grados Celsius (°C): mínima (T_{Min}), promedio (T_{Prom}), máxima (T_{Max}) y cantidad de términos experimentados (n) se presentan en la Tabla 5.4 clasificados por períodos del año y en la Tabla 5.5 por estaciones meteorológicas.

A través de las Ecuaciones 5.1 y 5.2 calculamos los índices que representan la variaciones de la temperatura en el área de estudio. En la Tabla 5.3 se muestran las variaciones relativas calculadas cuando se utilizan los datos

Tabla 5.5: Tabla con los datos de las estaciones meteorológicas utilizadas: Estaciones meteorológicas, temperaturas en grados Celsius (°C): mínima (T_{Min}), promedio (T_{Pro}) y máxima (T_{Max}).

Estaciones	Temperaturas (°C)	T_{Min}	T_{Pro}	T_{Max}
Arcoverde		21.05	26.00	29.32
Afranio		19.72	28.14	31.89
Canudos		16.49	25.09	31.06
Cabrobó		22.12	26.57	31.78
Delfino		22.90	26.66	31.61
Euclides Da Cunha		19.13	24.44	30.40
Floresta		19.03	23.65	28.84
Jacobina		17.93	23.98	28.89
Oeiras		19.76	25.75	30.98
Paulistana		23.04	27.09	33.14
Paulo Afonso		20.12	29.48	31.90
Petrolina		22.30	26.84	31.96
Picos		21.31	26.70	30.44
Remanso		18.71	29.16	30.14
São João do Piauí		23.62	27.52	32.47
São Raimundo Nonato		22.50	29.88	33.03
Senhor Do Bonfim		18.61	23.96	29.32
Uauá		20.43	25.27	30.52

para los períodos estimados. Los valores del índice EAP obtenidos fueron pequeños, y fluctuaron entre 0.91 % y 1.20 %. Este índice sirve para medir la fiabilidad de los resultados a partir de la muestra de experimentación, presentando un alto grado de seguridad y precisión en los valores encontrados en nuestro problema.

Fueron obtenidos valores más grandes para el parámetro p, que indica una mayor influencia de los valores más cercanos del punto interpolado. Para valores entre $0 < p < 1$ tenemos que los puntos x_k se tornan menos acentuados, mientras que para $p > 1$ se ven aún más acentuados los puntos [88]. Luego, los valores de p mayores que 1 en nuestro caso de estudio indican una mayor influencia de la superficie del suelo en los valores de temperatura más cercanos a los puntos interpolados [89]. Aunque las estaciones

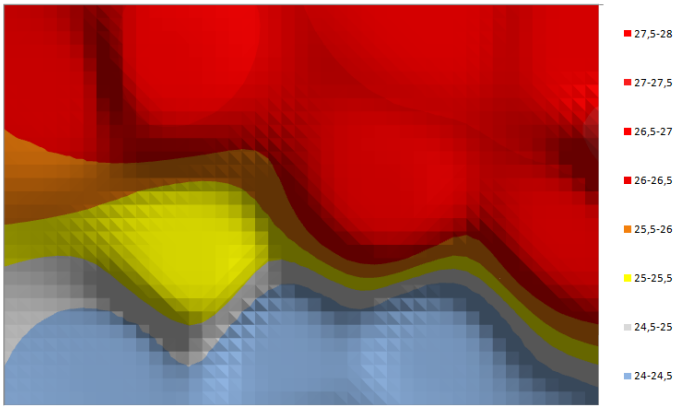


Figura 5.3: Mapa de calor de datos anuales para la representación de la temperatura.

meteorológicas están distantes entre sí (Figura 5.2), se puede observar una buena concordancia entre los valores reales y estimados de temperatura con los datos de los períodos. Los resultados de fiabilidad obtenidos se pueden comparar con los obtenidos por los autores Amorim et al. en [83], ya que estos autores realizan el mismo procedimiento propuesto. Estos autores interpolan las temperaturas en la región del Valle del Rio Doce teniendo en sus resultados un bajo valor para el EAP, mostrando una alta fiabilidad de la misma forma que en nuestros experimentos.

Con la ayuda del código de representación de variables meteorológicas del clima y la ecuación para el IPD, se generaron mapas de calor. Los mapas de calor generados representan la temperatura en cada período del año (véase la Figura 5.3), y también se puede representar la temperatura anual por períodos (véase Figura 5.4). En las figuras, el color azul representa las temperaturas más bajas, mientras que el color rojo representa las temperaturas más altas. En estos mapas de calor es posible observar el efecto de las desviaciones de temperatura cerca de las estaciones meteorológicas donde los colores son más intensos y tendrán interferencias de otras estaciones a

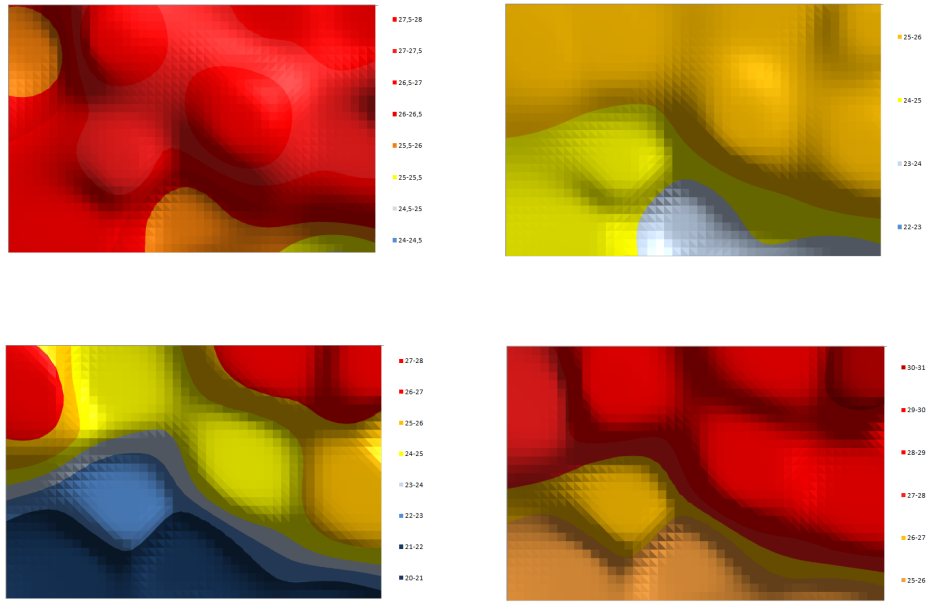


Figura 5.4: Mapa de calor para datos del verano (arriba-izquierda), otoño (arriba-derecha), invierno (abajo-izquierda), y primavera (abajo-derecha) con p igual a 3.3 y 3.6 para la representación de la temperatura.

lo largo de las distancias.

Así pues, se ha predicho la variable temperatura para un nivel considerado óptimo de fiabilidad, y el modelo de representación del clima para el Valle del Rio São Francisco se ha ajustado bien a los datos de los experimentos. Analizando los resultados, es posible observar también que, para generar valores de temperatura cuya fidelidad sea representativa, es necesario estimar los cálculos con un gran número de estaciones, lo que implica un tiempo de procesamiento muy alto. Es decir, la fidelidad de la representación es directamente proporcional al número de estaciones e inversamente proporcional al tiempo de procesamiento necesario para su estimación, por lo que es crucial aplicar los modelos de computación de alto rendimiento en la representación del clima.

Estudio experimental del algoritmo

Con el fin de validar la metodología presentada se ha utilizado el esquema paralelo propuesto para representar el modelo computacional de variables meteorológicas del clima. Al igual que para el estudio del relieve mostrado en el Capítulo 3, se ha partido de un algoritmo paralelo basado en OpenMP+CUDA. Hemos experimentado con un número de estaciones de 4 hasta 18 debido al grado de fiabilidad de las muestras en este rango para obtener los tiempos de ejecución.

La ejecución del código original secuencial se presenta en la Tabla 5.6 donde

Automático: representa el tiempo de ejecución necesario para acceder a la base de datos,

Bucle: representa el tiempo de ejecución relativo al bucle interno, y

Computación: representa el tiempo de ejecución correspondiente a la suma de los otros dos tiempos (Automático+Bucle).

La Tabla 5.6 muestra, para distintos tamaños de problema, el tiempo de ejecución (en segundos) obtenido con los valores experimental y con la selección realizada en el algoritmo secuencial. Los porcentajes con respecto al tiempo de ejecución total se muestran en las respectivas columnas en la parte de bajo de la tabla. Estos porcentajes muestran que, para tamaños pequeños, en MICROMACHIN el tiempo **Automático** es más grande que el tiempo de **Bucle**. Para tamaños de problemas medianos y grandes sucede al revés. A medida que el número de estaciones requerido es más grande, la forma óptima de realizar la simulación tiende a dividir las tareas para el tiempo de **Bucle** entre los recursos computacionales existentes. Es decir, es preferible emplear configuraciones en las que varios cores trabajen en paralelo en una misma simulación, aunque su rendimiento se vea desgastado por las comunicaciones que tienen que establecer entre ellos, ya que esto se ve compensado por el ahorro en el tiempo de ejecución.

La Figura 5.5 muestra gráficamente el detalle de los tiempos de ejecución de cada parte para 12, 14, 16, y 18 estaciones.

El problema del clima tiene como centro la combinación lineal de los puntos de muestreo, ponderado por el modelo matemático propuesto. La

Tabla 5.6: Análisis comparativo de los tiempos de ejecución secuencial (en segundos).

Estaciones	**Automático**	**Bucle**	**Computación**
4	3.53	1.24	4.78
6	18.07	9.23	27.30
8	90.54	62.22	152.76
10	434.60	360.63	795.27
12	988.12	1996.19	2984.31
14	4472.69	21243.02	25715.71
16	12004.17	63424.38	75428.55
18	81498.14	122191.27	203689.40
4	73.99 %	26.00 %	100.00 %
6	66.19 %	33.81 %	100.00 %
8	59.27 %	40.73 %	100.00 %
10	54.65 %	45.35 %	100.00 %
12	33.11 %	66.89 %	100.00 %
14	17.39 %	82.61 %	100.00 %
16	15.91 %	84.08 %	100.00 %
18	40.01 %	59.99 %	100.00 %

solución consiste en dos tareas principales: la construcción de la representación matricial y el cálculo de las variables del clima. Esta segunda parte es llevada a cabo por el kernel (Algoritmo 5.2) en caso de que se utilicen GPUs. El tamaño de la matriz resultado no es grande, mientras que la construcción de la matriz en sí requiere cálculos pesados que implican sumas de millones de iteraciones. Esto da lugar a diversas posibilidades en cuanto a la forma de calcular la suma de manera eficiente, es decir, a cómo realizar la división y la distribución de la suma en distintos recursos computacionales. La versión paralela "híbrida" desarrollada (denotada como multicore+multi-GPU) representa las ejecuciones utilizando los cores de la CPU y 2-GPUs simultáneamente. Se ha utilizado la cantidad óptima de los recursos computacionales del entorno de experimentación, esto es, la cantidad óptima de cores de CPU y de las GPUs.

La Tabla 5.7 muestra, para distintos tamaños de problema, el tiempo de ejecución (en segundos) obtenido con los valores experimentales y con la

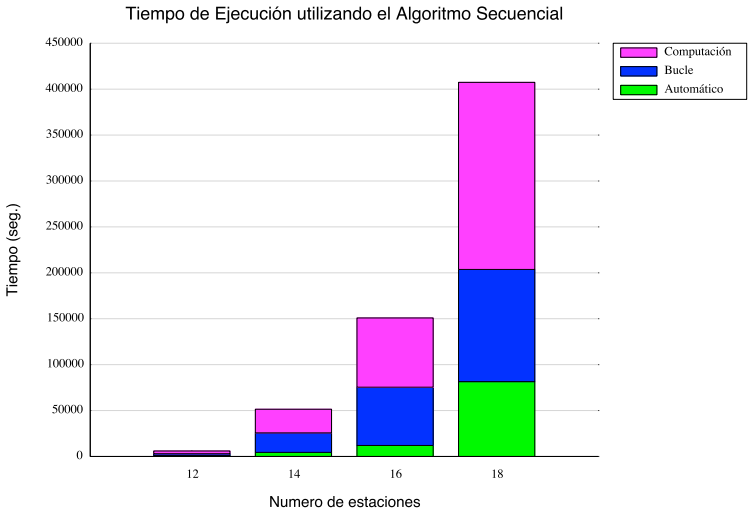


Figura 5.5: Análisis comparativo de los tiempos de ejecución secuencial (en segundos).

selección realizada por el algoritmo paralelo. Los porcentajes con respecto al tiempo de ejecución total, de la misma forma que anteriormente, son mostrados en las respectivas columnas de la parte de debajo de la tabla. Los porcentajes muestran que para tamaños pequeños en MICROMACHIN el tiempo **Automático** es menor que el tiempo de **Bucle**. Ahora sin embargo, para tamaños de problemas medianos y grandes los valores continúan siendo más grandes, debido a la optimización aplicada. La estrategia paralela ha conseguido dividir las tareas para el tiempo de **Bucle** entre los recursos computacionales existentes de forma optimizada.

Se ha extraído también la información acerca de la escalabilidad del código paralelo bajo el entorno de ejecución heterogéneo. Para ello se han realizado un conjunto de pruebas circunscritas a un dominio de tamaños y formas, variando las particiones de este dominio al ejecutarlo en paralelo con varios recursos. Con todo esto se ha conseguido obtener la mejor combinación de parámetros de ejecución bajo el entorno de simulación utilizado para los experimentos. Los parámetros óptimos obtenidos para el entorno de ejecución han sido: 2-GPUs + 2-Procesadores Quadcore utilizando 8

Tabla 5.7: Análisis comparativo de los tiempos de ejecución para la versión multicore+multi-GPU (en segundos).

Estaciones	**Automático**	**Bucle**	**Computación**
4	2.47	5.44	7.91
6	8.35	5.25	13.60
8	38.33	20.12	59.87
10	187.34	73.33	263.49
12	284.83	157.81	442.58
14	912.75	1250.66	2154.24
16	4283.81	2458.08	6742.75
18	14498.81	7723.21	22222.84
4	46.27 %	54.00 %	100.00 %
6	58.78 %	42.09 %	100.00 %
8	50.97 %	48.81 %	100.00 %
10	66.63 %	33.36 %	100.00 %
12	61.23 %	34.31 %	100.00 %
14	58.50 %	41.50 %	100.00 %
16	66.72 %	33.02 %	100.00 %
18	65.36 %	34.64 %	100.00 %

cores. Bajo este entorno, la carga de trabajo asignada es = (GPU, GPU, CPU) = (40 %, 40 %, 20 %). Los resultados muestran que el modelo paralelo multicore+multi-GPU reduce el tiempo de ejecución significativamente. Como también se puede ver, el máximo *speedup* obtenido, según la Figura 5.7, es 6,62 para la parte que hemos denotado como **Automático**, y 15,82 para la parte **Bucle**, coincidiendo este último valor con el número de cores del entorno de ejecución. El rendimiento obtenido con la combinación de herramientas OpenMP y CUDA aventaja notablemente a la versión paralela que aprovecha solo los cores CPU.

5.5 Conclusiones

En este capítulo se ha trabajado en el problema de la representación de variables meteorológicas del clima. El problema en particular se ha centrado

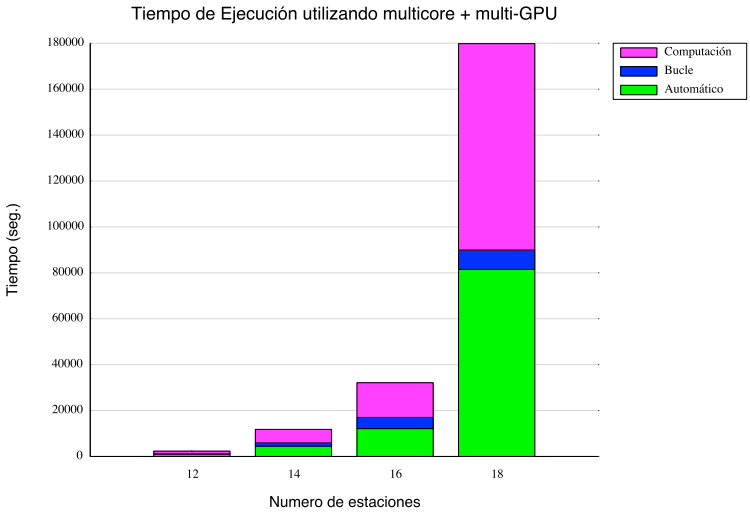


Figura 5.6: Análisis comparativo de rendimiento de los tiempos de ejecución para el modelo multicore+multi-GPU (en segundos).

en inferir valores de estas variables en puntos del área donde no se tienen mediciones, con base en mediciones realizadas en otros puntos de alrededor. Se ha descrito la metodología matemática en la que se basa este método y se han realizado los experimentos pertinentes que han demostrado que la estimación de las variables meteorológicas necesita de una gran potencia de cálculo. El tiempo de procesamiento es más grande cuanto mayor es el número de estaciones y más exactitud necesitamos para su descripción. Sin embargo, hemos observado que el problema tiene un grado de paralelismo intrínseco importante, cosa que se ha aprovechado para desarrollar un algoritmo paralelo capaz de aprovechar los cores existentes en el Host. Este paralelismo se ha podido aprovechar también con éxito para implementar un kernel CUDA que acelera el cálculo de forma muy significativa. El diseño completo de la aplicación es híbrido, es decir, utiliza todos los recursos computacionales al alcance: cores CPU y GPUs. El diseño utilizado está basado en el esquema planteado en el Capítulo 3 y el peso adecuado que se le otorga a cada recurso se ha obtenido aquí de manera experimental.

La optimización de la parte computacional del problema ha dejado al

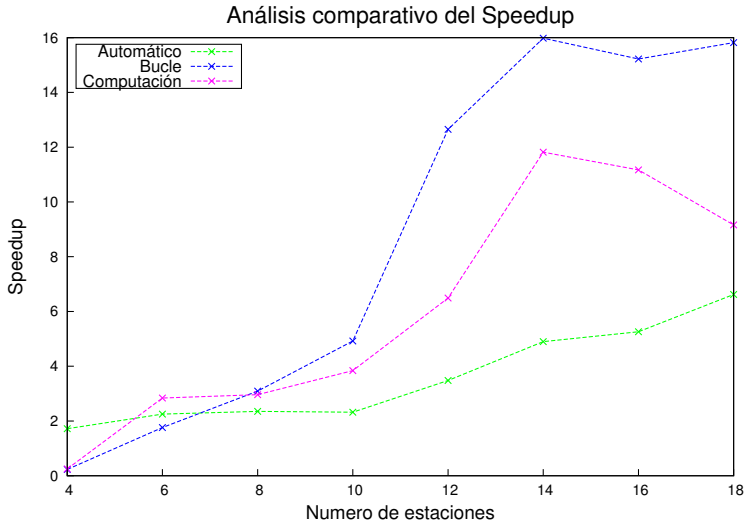


Figura 5.7: Análisis comparativo del *speedup* para las partes: Automático, Bucle y Computación.

descubierto, sin embargo, otro problema subyacente que no se ha podido resolver convenientemente en esta tesis y concierne al tiempo necesario de acceso a la base de datos (tiempo **Automático**). Queda así como trabajo futuro la paralelización de esta parte del cálculo.

Modelado de ríos 6

En este capítulo se estudia un modelo computacional hídrico de ríos. La alta variabilidad de los cauces hídricos de Brasil redunda frecuentemente en fuertes inundaciones o sequías persistentes. Estos fenómenos provocan un impacto socio-económico importante que afecta tanto a la agricultura como al resto de formas de aprovechamiento humano de los recursos hídricos. Esto ha motivado el desarrollo de modelos hídricos para estudiar el comportamiento de los acuíferos. La simulación de estos modelos es computacionalmente costosa, y su coste aumenta con la longitud del río bajo estudio y la precisión requerida. En este capítulo aplicamos técnicas de programación paralela heterogénea para abordar y reducir el coste computacional de la simulación de estos sistemas naturales.

6.1 Introducción

Brasil es un país rico en recursos hídricos, pero estos recursos no están bien distribuidos por diversas cuestiones que atienden a razones climáticas, geológicas y/o topográficas. Se puede poner a la región del *Semi-árido Nordestino* como ejemplo de esta mala distribución hídrica. La región del *Semi-árido Nordestino* posee altas tasas de evaporación, escasez de precipi-

taciones y, consecuentemente, sequías frecuentes y continuas. El problema se agrava en la medida en que los recursos hídricos se hacen cada vez más necesarios. Las estimaciones de crecimiento de la población hacen prever unas necesidades de almacenamiento y distribución de agua muy grandes que permitan sostener ese mayor consumo previsible y controlar a la vez la mayor susceptibilidad existente a la contaminación de los acuíferos. Dado que el agua es algo vital para el desarrollo de la agricultura, los científicos actualmente han dado mayor atención al desarrollo de técnicas para su explotación y mantenimiento de su calidad.

En algunas regiones de Brasil el agua es la única fuente de vida. Es por esto que el gobierno brasileño promueve estudios previos de los recursos hídricos disponibles con el objeto de disminuir el impacto de las sequías que llegan a constituir verdaderas situaciones de emergencia en algunas regiones. Estos estudios incluyen los aspectos de gestión económica, social y científica. La explotación racional y sostenible de los recursos hídricos exige que se lleven a cabo simulaciones que puedan indicar el comportamiento de los acuíferos. Estas simulaciones, actualmente, se realizan a través de modelos computacionales que requieren un conocimiento preciso de las características hidrológicas.

El estudio de modelos computacionales hídricos de ríos, por tanto, es de gran importancia para entender, predecir y controlar los procesos físicos que tienen lugar en ellos, además de servir como base para estudios de transporte de contaminantes y de procesos de erosión [90]. La dificultad de realizar ensayos de laboratorio, así como el coste económico de llevar a cabo mediciones experimentales en campo, hacen de los modelos computacionales una herramienta muy útil para el estudio de este tipo de problemas. El modelado computacional presenta, además, la ventaja de poder estudiar el impacto que puede provocar una futura actuación de la ingeniería, permitiendo la evaluación de diferentes escenarios. Todo ello a un coste económico relativamente bajo, pero con una desventaja muy grande: el alto coste computacional necesario para ejecutar las simulaciones. Esto nos ha motivado a aplicar técnicas de computación paralela que permitan simular los modelos hídricos y acelerar su ejecución. Como en capítulos anteriores de esta tesis, hemos trabajado en un entorno heterogéneo de computación. La estrategia utilizada también es común a la utilizada en capítulos anteriores: implementación de un kernel CUDA para realizar computación en GPUs y realizar una distribución heterogénea de datos teniendo en cuenta

la capacidad de cómputo de los recursos computacionales disponibles.

La sección siguiente muestra el trasfondo matemático en el que se basa el modelado hídrico de ríos utilizado. La Sección 6.3 presenta la implementación realizada, mientras que los resultados experimentales están descritos en la Sección 6.4. El capítulo termina con unas conclusiones esbozadas en la última sección.

6.2 Modelo matemático hídrico de ríos

El modelo matemático hídrico surge a partir de las ecuaciones de flujo unidimensional en función de la velocidad del agua (v) y de la profundidad del río (y), cuyo desarrollo [3] se resume para llegar a la ecuación matricial siguiente:

$$\begin{aligned} & b\frac{\partial y}{\partial t} + A\frac{\partial v}{\partial x} + v \cdot b\frac{\partial y}{\partial x} = 0 \ , \\ & \frac{1}{g}\frac{\partial v}{\partial t} + \frac{v}{g}\frac{\partial v}{\partial x} + \frac{\partial y}{\partial x} = (S_o - S_f) \ , \end{aligned} \tag{6.1}$$

donde g es la aceleración de la gravedad, x es la distancia longitudinal, t es el tiempo, S es la escorrentía de la línea de roce, y b el ancho del área estudiada.

La derivada total de las variables dependientes es:

$$\begin{aligned} dy &= \frac{\partial y}{\partial x}dx + \frac{\partial y}{\partial t}dt \ , \\ dv &= \frac{\partial v}{\partial x}dx + \frac{\partial v}{\partial t}dt \ . \end{aligned} \tag{6.2}$$

Los sistemas de la Ecuación 6.1 y la Ecuación 6.2 pueden ser puestos en formato matricial, teniendo como variables las derivadas parciales

dependientes:

$$\begin{bmatrix} v \cdot b & b & A & 0 \\ 1 & 0 & \frac{v}{g} & \frac{1}{g} \\ dx & dt & 0 & 0 \\ 0 & 0 & dx & dt \end{bmatrix} \begin{bmatrix} \frac{\partial y}{\partial x} \\ \frac{\partial y}{\partial t} \\ \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial t} \end{bmatrix} = \begin{bmatrix} 0 \\ S_o - S_f \\ dy \\ dv \end{bmatrix} . \tag{6.3}$$

A partir del concepto de *líneas características* [3], como aquellas perturbaciones que se desplazan a lo largo del río, tenemos que estas líneas presentan una discontinuidad en la primera derivada y de más alto orden en las funciones dependientes, para este caso y y v. La discontinuidad puede ser caracterizada por la indeterminación de la función derivada $\frac{\partial y}{\partial x}$, o sea $\frac{\partial y}{\partial x} = 0$. Solucionando el sistema de ecuaciones matriciales de la Ecuación 6.3, tenemos la siguiente ecuación:

$$v \cdot b \left(\frac{1}{g} dx \cdot dt - dt^2 \frac{v}{g} \right) - b \left(\frac{1}{g} dx^2 - \frac{v}{g} dt \cdot dx \right) + A \cdot dt^2 = 0 .$$

Solucionando la ecuación anterior tenemos

$$\frac{dx}{dt} = v \pm \sqrt{\frac{A \cdot g}{b}} . \tag{6.4}$$

Considerando que $\frac{\partial y}{\partial x}$ debe ser igual a cero, tenemos la solución:

$$-b \left(\frac{dy \cdot dx}{g} - \frac{dt \cdot dy \cdot v}{g} \right) + A \left[(S_o - S_f) dt^2 - \frac{dt \cdot dv}{g} \right] = 0 . \tag{6.5}$$

Con la reorganización de la Ecuación 6.4 y sustituyendo en la Ecuación 6.5, obtenemos el siguiente par de ecuaciones:

$$\frac{dx}{dt} = v \pm c ,$$

$$\frac{dv}{dt} \pm \frac{g}{c} \frac{dy}{dt} = g(S_o - S_f) .$$

Aplicamos ahora el esquema implícito lineal propuesto en [3] para la resolución de las ecuaciones anteriores, obteniendo un esquema basado en la linearización de las ecuaciones diferenciales. Considerando el dominio del tiempo $t+1$ para el término de roce, tenemos la siguiente ecuación de continuidad:

$$\frac{Q_{i+1}^{t+1} - Q_i^t}{\Delta x_i} + \frac{b_o^t}{2\Delta t}\left(y_i^{t+1} - y_i^t + y_{i+1}^{t+1} - y_{i+1}^t\right) = \frac{1}{2}\left(q_i^t + q_i^{t+1}\right) \ . \quad (6.6)$$

Reorganizando la Ecuación 6.6 y teniendo como base el área de inundación, la ecuación de continuidad resulta:

$$A_i Q_i^{t+1} + B_{i+1}^{t+1} y_i^{t+1} + C_i Q_{i+1}^{t+1} + D_i y_{i+1}^{t+1} = E_i \ ,$$

donde

$$\begin{aligned}
A_i &= -4\frac{\Delta t}{\Delta x_i}, \\
B_i &= b_i^t + b_{i+1}^t + \frac{A \cdot F_i^t}{\Delta x_i}, \\
C_i &= 4\frac{\Delta t}{\Delta x_i}, \\
D_i &= B_i, \\
E_i &= 2\Delta t\left(q_i^t + q_i^{t+1}\right) + \left(b_i^t + b_{i+1}^t + \frac{A \cdot F_i^t}{\Delta x_i}\right)\left(y_i^t + y_{i+1}^t\right).
\end{aligned}$$

Desarrollando la ecuaciones anteriores, obtenemos la siguiente ecuación:

$$\frac{Q_i^{t+1} - Q_i^t - Q_{i+1}^t}{2\Delta t} + \frac{2v_i^t\left(Q_{i+1}^{t+1} - Q_i^{t+1}\right)}{\Delta x_i} + g \cdot A_i^t\left(1 - F^2\right)_i^t \ ,$$

$$\frac{\left(y_{i+1}^{t+1} - y_i^{t+1}\right)}{\Delta x_i} =$$

$$g \cdot A_i^t \frac{(zo_{i+1} - zo_i)}{\Delta x_i} + \frac{g}{2}\left[A_i^t S_{f(i)}^{t+1} + A_{i+1}^t S_{f(i+1)}^{t+1}\right] + \left[v^2\left(\frac{\partial A}{\partial x}\right)_{y=cte}\right]_i^t \ ,$$

Utilizando ahora la expansión de Taylor de primer orden obtenemos:

$$A^t S_f^{t+1} \cong A^t \left[S_f^t + \left(\frac{\partial S_f}{\partial Q} \right)^t \left(Q^{t+1} - Q^t \right) + \left(\frac{\partial S_f}{\partial y} \right)_t \left(y^{t+1} + y^t \right) \right] ,$$

donde

$$\frac{\partial S_f}{\partial Q} = 2 \frac{S_f}{Q} ,$$

$$\frac{\partial S_f}{\partial y} = \frac{-2 S_f}{K} \frac{\partial K}{\partial y} ,$$

Y mediante una reordenación de los términos se tiene:

$$A_i Q_i^{t+1} + B_i^{t+1} y_i^{t+1} + C_i Q_{i+1}^{t+1} + D_i y_{i+1}^{t+1} = E_i , \tag{6.7}$$

teniendo en cuenta las siguientes igualdades:

$$\begin{aligned}
A_i &= (1 + 2CS2_i) - CS1_i, \\
B_i &= -2 \frac{\Delta t}{\Delta x_i} \left[gA \left(1 - F^2 \right) \right]_i^t + CS3_i, \\
C_i &= (1 + 2CS2_{i+1}) + CS1_{i+1}, \\
D_i &= 2 \frac{\Delta t}{\Delta x_i} \left[gA \left(1 - F^2 \right) \right]_i^t + CS3_{i+1}, \\
E_i &= Q_i^t \left(1 + 2CS2_i \right) + Q_{i+1}^t \left(1 + CS2_{i+1} \right) + 2 \frac{\Delta t}{\Delta x_i} g A_i^t \left(zo_{i+1} - zo_i \right) + \\
&\quad y_i^t CS3_i + y_{i+1}^t CS3 + y_{i+1}^t CS3_{i+1} + 2\Delta t \left[v^2 \left(\frac{\partial A}{\partial x} \right)_{y=cte} \right]_i^t , \\
CS1 &= 4 \frac{\Delta t}{\Delta x_i} v_i^t, \\
CS2 &= g \Delta t \left(\frac{S_f}{v} \right)_i^t , \\
CS3 &= g \Delta t \left[(S_f b)_i^t - 2 \left(A \frac{S_f}{K} \right)_i^t \left(\frac{\partial K}{\partial y} \right)_i^t \right] .
\end{aligned}$$

Las Ecuaciones 6.7 son aplicadas en cada tramo del río, para un río particionado en n secciones, $n-1$ tramos y $2(n-1)$ ecuaciones. Las variables son los valores de Q e y en el dominio del tiempo $t+1$ en cada sección. El sistema de ecuaciones resultante es $Qx = y$ donde,

$$Q = \begin{pmatrix} A_1 & B_1 & C_1 & D_1 & & & & & & \\ A_2 & B_2 & C_2 & D_2 & & & & & & \\ & & A_3 & B_3 & C_3 & D_3 & & & & \\ & & \cdots & \cdots & \cdots & \cdots & \cdots & & & \\ & & & & & & A_{n-1} & B_{n-1} & C_{n-1} & D_{n-1} \\ & & & & & & A_{n-2} & B_{n-2} & C_{n-2} & D_{n-2} \end{pmatrix}, \quad (6.8)$$

$$x = \begin{pmatrix} Q_1 \\ y_1 \\ y_2 \\ \cdots \\ Q_n \\ y_n \end{pmatrix}, \text{ e } y = \begin{pmatrix} 0 \\ E_1 \\ E_2 \\ \cdots \\ E_n \\ 0 \end{pmatrix}.$$

La estimación del modelo hídrico consta de las siguientes partes principales de cómputo:

1. la construcción de la matriz Q y del vector y,
2. el cálculo de la escorrentía total que se expresa por la matriz Q, y
3. la solución del sistema lineal de ecuaciones $Qx = y$.

Análisis del orden de magnitud del problema

Sin ánimo de entrar demasiado en los detalles de la elección de la configuración física del modelo hídrico ni en la base de datos empleada como condición inicial y de contorno, ya que esto queda fuera del ámbito de este trabajo, diremos que el modelo computacional hídrico que se plantea posee las siguientes características para el dominio de experimentación:

- ◇ Los datos han sido obtenidos, al igual que en capítulos anteriores, a partir de muestras del Río São Francisco que está situado en la comarca del *Semi-árido Nordestino* y que incluye la región agrícola conocida como Valle del Río São Francisco.

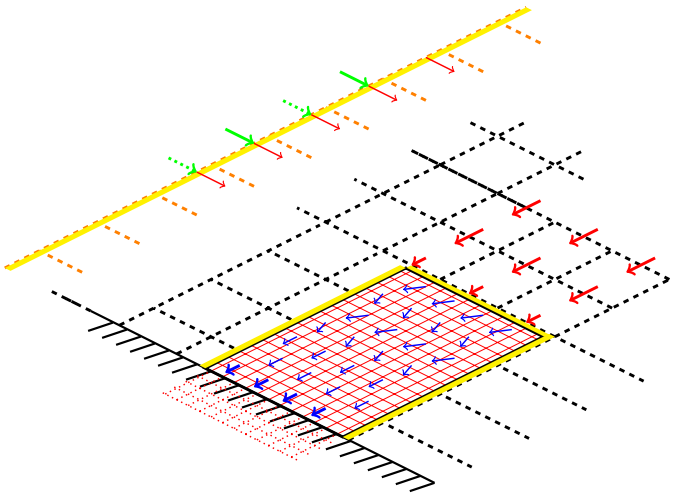


Figura 6.1: Flujo unidimensional en función de la velocidad del agua y de la profundidad del Río São Francisco. Variable hídrica: escorrentía total.

◇ Se han utilizado valores de datos diarios de la variable hídrica bajo estudio, esto es, de la escorrentía total de la superficie del río (Figura 6.1), mediante mediciones experimentales en campo tomadas por estaciones que operan en esta área y que son propiedad de la Universidade do Vale do Río São Francisco (UNIVASF) [91]

6.3 Modelo computacional hídrico de ríos

El algoritmo secuencial del modelo hídrico de ríos posee una parte con un alto coste computacional con un alto número de operaciones trigonométicas en su fundamento. El Algoritmo 6.1 esboza esa parte, que se corresponde con la formación de la matriz Q (Ecuación 6.8) y el cálculo de la escorrentía total de la superficie del río. Es sobre esa parte sobre la que hemos trabajado para paralelizar. La paralelización de esta parte de código resulta muy útil debido al alto coste computacional que alcanza por tratarse de problemas muy grandes. También resulta factible debido a la existencia de un alto grado de

Algoritmo 6.1 Rutina secuencial para la construcción de la matriz `mde.a`.

```
 1: #define XO -40.4759
 2: #define YO -9.3887
 3: for( int r = 1; r < My - 1; r++ ) {
 4:   for( int c = 1; c < Mx - 1; c++ ) {
 5:    d[0]= d[4] = dx;
 6:    d[1]= d[3] = d[5] = d[7] = sqrt(dx*dx+dy*dy);
 7:    d[2]= d[6] = dy;
 8:
 9:   a[0]= atan((mde[(r+1)*Mx +  c   ].z - mde[r*My + c].z)/d[0]);
10:   a[1]= atan((mde[(r+1)*Mx + (c+1)].z - mde[r*My + c].z)/d[1]);
11:   a[2]= atan((mde[ r   *Mx + (c+1)].z - mde[r*My + c].z)/d[2]);
12:   a[3]= atan((mde[(r-1)*Mx + (c+1)].z - mde[r*My + c].z)/d[3]);
13:   a[4]= atan((mde[(r-1)*Mx +  c   ].z - mde[r*My + c].z)/d[4]);
14:   a[5]= atan((mde[(r-1)*Mx + (c-1)].z - mde[r*My + c].z)/d[5]);
15:   a[6]= atan((mde[ r   *Mx + (c-1)].z - mde[r*My + c].z)/d[6]);
16:   a[7]= atan((mde[(r+1)*Mx + (c-1)].z - mde[r*My + c].z)/d[7]);
17:
18:   ap = a[0];
19:   for( int i = 1; i < 8; i++ ) {
20:    if( ap > a[i] )
21:    ap = a[i];
22:   }
23:   ap = fabs(ap)*180.0/PI;
24:   mde[r*Mx + c].a  = ap;
25:  }
26: }
```

paralelismo intrínseco que se deriva del hecho de que muchas operaciones se llevan a cabo de manera independiente sobre diversos datos.

El desarrollo de una versión paralela de la rutina secuencial utilizando varios cores CPU no resulta muy difícil. Sin embargo, un aprovechamiento eficiente de todo el sistema no es una tarea trivial. Para los experimentos hemos considerado un sistema heterogéneo compuesto por un multicore y un subsistema multi-GPU, tal como ha sido habitual a lo largo de todo el trabajo de esta tesis. El esquema paralelo utilizado para particionar y equilibrar la carga de trabajo entre los cores CPU y las GPUs ha sido el mismo que el utilizado en capítulos anteriores (veáse Algoritmo 3.2). Este esquema de diseño del algoritmo principal, que hemos utilizado a lo largo

de todo el trabajo en distintos problemas, ha resultado ser sencillo y, a su vez, muy útil.

La matriz denotada en el código como `mde.a` constituye la salida del algoritmo y contiene las sumas parciales calculadas por cada subsistema. Se utilizan vectores para el almacenamiento de la solución, obteniendo resultados parciales de forma independiente que has sido obtenidos por los cores CPU y/o por las GPUs. Cada *thread*, pues, trabaja en una parte diferente de los vectores. Una vez que los *threads* se sincronizan para terminar, el *threads* maestro lleva a cabo la suma total de estos resultados parciales.

Kernel CUDA para modelo hídrico de ríos

El código mostrado en el Algoritmo 6.1 se caracteriza por trabajar sobre una matriz de 3×3 puntos que pueden ser actualizados concurrentemente. Esto es aprovechable para diseñar un kernel CUDA que utilice este paralelismo a nivel de operación sobre distintos datos para alimentar los *threads*. La implementación de este kernel puede verse en el Algoritmo 6.2. En primer lugar, se llevan a cabo las operaciones habituales de asignación de tareas a la GPU y la carga de datos en las mismas. Por lo tanto, se asume que las estructuras matriciales `mde`, y las dimensiones `Mx` y `My` ya han sido previamente cargadas en la memoria global de las tarjetas gráficas para poder interpretar el kernel del Algoritmo 6.2 correctamente. La construcción de la matriz `mde` proporciona una gran oportunidad para aplicar el paralelismo. Con fin de aprovechar la concurrencia, utilizamos un grid bidimensional de bloques de *threads*. Este bloque posee una dimensión `Mx`×`My`, cuyos valores son definiciones de macros en las primeras líneas del algoritmo. Cada *thread* se encuentra en un bloque a través de las dos coordenadas calculadas en las líneas 25 y 26. El bloque de *threads* se organiza en un grid bidimensional. La dimensiones son $\lceil$`Mx/bx`$\rceil$ y $\lceil$`My/by`$\rceil$, definiendo `bx` y `by` como la dimensión del tamaño de bloque de `mde`. El código que se muestra a continuación figura dentro de la rutina `matrixGPU` (que se encuentra en la versión del Algoritmo 3.2 construida para resolver este problema) y muestra esta distribución y la llamada al kernel:

```
1: dim3 dimBlock(bx , by);
2: dim3 dimGrid( (int) ceil(Mx/bx), (int) ceil(My/by) );
3: kernel<<<dimGrid, dimBlock>>>( Mx, My, mde);
```

Algoritmo 6.2 Rutina del kernel CUDA para el modelo computacional hídrico de ríos.

```
 1:__device__ void angulo( int l, c, Mx, clDado *mde ) {
 2:  double a[8], d[8], ap, dx = dy = 10;
 3:  d[0]= d[4] = dx;
 4:  d[1]= d[3] = d[5] = d[7] = sqrt(dx*dx+dy*dy);
 5:  d[2]= d[6] = dy;
 6:
 7:  a[0]= atan((mde[(l+1)*Mx  + c ].z    - mde[l*Mx + c].z)/d[0]);
 8:  a[1]= atan((mde[(l+1)*Mx  + (c+1)].z - mde[l*Mx + c].z)/d[1]);
 9:  a[2]= atan((mde[l*Mx      + (c+1)].z - mde[l*Mx + c].z)/d[2]);
10:  a[3]= atan((mde[(l-1)*Mx  + (c+1)].z - mde[l*Mx + c].z)/d[3]);
11:  a[4]= atan((mde[(l-1)*Mx  + c ].z    - mde[l*Mx + c].z)/d[4]);
12:  a[5]= atan((mde[(l-1)*Mx  + (c-1)].z - mde[l*Mx + c].z)/d[5]);
13:  a[6]= atan((mde[l*Mx      + (c-1)].z - mde[l*Mx + c].z)/d[6]);
14:  a[7]= atan((mde[(l+1)*Mx  + (c-1)].z - mde[l*Mx + c].z)/d[7]);
15:
16:  ap = a[0];
17:  for( int i = 1; i < 8; i++ ) {
18:   if( ap > a[i] )
19:      ap = a[i];
20:  }
21:  ap = fabs(ap)*180.0/M_PI;
22:  mde[l*Mx + c].a  = ap;
23:} /*angulo*/
24:
25:__global__ void kernel(int Mx, int My, clDado *mde) {
26:  int c = threadIdx.x + blockIdx.x*blockDim.x;
27:  int r = threadIdx.y + blockIdx.y*blockDim.y;
28:
29:  if( c < Mx && r < My ) {
30:   d = sqrt((mde[r*Mx+c].x-xo)*(mde[r*Mx+c].x-xo) +
31:            (mde[r*Mx+c].y-yo)*(mde[r*Mx+c].y-yo));
32:   angulo(r, c, mde, Mx);
33:  }
34:} /*kernel*/
```

6.4 Resultados experimentales

A continuación se describen diversos experimentos realizados para estudiar los algoritmos propuestos anteriormente. Se estudian los costes de las dife-

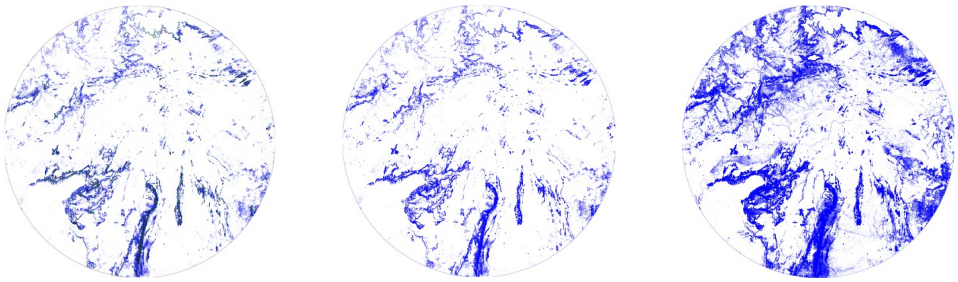


Figura 6.2: Mapas hídricos del Río São Francisco para escorrentía con 55 %, 60 % y 90 % de resolución.

rentes partes del algoritmo secuencial para compararlas en función del tamaño del problema (el tamaño corresponde a la longitud del recorte del río). Todos estos resultados se han obtenido en el entorno ELEANORRIGBY (Sección 2.2).

Los resultados de la simulación producen los mapas hídricos que muestran las variables de salida del ciclo hidrológico. En la Figura 6.2 se muestra la variable escorrentía total. Los caudales mensuales en cada intervalo de tiempo se obtienen integrando la escorrentía total en las cuencas vertientes a los puntos de simulación.

La cuenca del Río São Francisco tiene una extensión total de 2863 km, una superficie media de 640000 km^2, con un caudal medio de 2943 m^3/s. Los valores de la precipitación para el periodo entre 2010 y 2014 oscilan alrededor de los 2200 mm anuales en esta cuenca. Los valores simulados se contrastan muy bien con los datos históricos existentes, siendo que las principales divergencias entre los valores observados y simulados se deben a la falta de lluvia en las áreas estudiadas, así como a procesos mas complejos que interfieren en la simulación [82].

En todos los experimentos se ha ejecutado el algoritmo paralelo heterogéneo que se ha diseñado para generar los mapas hídricos anteriores para diversas resoluciones de escorrentía total. En los experimentos incrementamos el número de *threads* CPU desde 1 hasta 24 para obtener el número que minimiza el tiempo. Después, añadimos 1 y 2-GPUs al número de *threads* obtenidos en la prueba anterior. Los tamaños de entrada del problema (tamaño del recorte del río) para los experimentos fueron 20000, 40000, ...,

Tabla 6.1: Tiempos de ejecución de las distintas versiones del algoritmo paralelo (en segundos) variando el tamaño del problema (recorte del río).

Río	S	OMP	1-GPU	2-GPUs	multicore+multi-GPU
20000	29,07	14,46	4,94	12,67	10,13
40000	113,15	47,24	25,53	14,56	12,00
60000	252,20	59,27	58,31	33,81	29,52
80000	446,20	87,66	107,10	56,42	52,69
100000	695,16	105,29	169,03	75,38	66,50
120000	999,08	151,21	245,05	95,71	85,96
140000	1357,96	205,42	335,17	130,40	118,07
160000	1771,81	267,91	439,28	170,44	141,83
180000	2240,61	338,69	557,69	215,85	167,23
200000	2764,37	417,75	690,10	266,61	194,29

200000 km^2. Al igual que en el estudio secuencial, y con objeto de simplificar los experimentos, se han generado los modelos con un valor fijo de resolución del 90 %, para variar el tamaño del problema y comparar así los tiempos resultantes y los *speedups*. Con todo ello, se pretende estudiar experimentalmente los siguientes puntos:

- ◇ la influencia de los parámetros del sistema y del algoritmo en el tiempo de ejecución, y
- ◇ la reducción del tiempo de ejecución cuando se usa un algoritmo híbrido con varios recursos computacionales.

La Tabla 6.1 muestra el tiempo de ejecución obtenido con el algoritmo secuencial en el entorno ELEANORRIGBY. La denominación "OMP" denota la computación con el uso de múltiples *threads* en los cores CPU. Las versiones denotadas por 1-GPU y 2-GPUs representan ejecuciones utilizando exclusivamente una o dos GPUs, respectivamente. El modelo paralelo híbrido, nombrado como "multicore+multi-GPU", muestra los valores máximos de *speedup* obtenidos experimentalmente, utilizando los parámetros óptimos para el sistema de ejecución. Los parámetros óptimos obtenidos para el entorno de ejecución son de 2-GPUs + 2-Procesadores CPU empleando 24 hilos de ejecución. La distribución de carga de trabajo entre los recursos

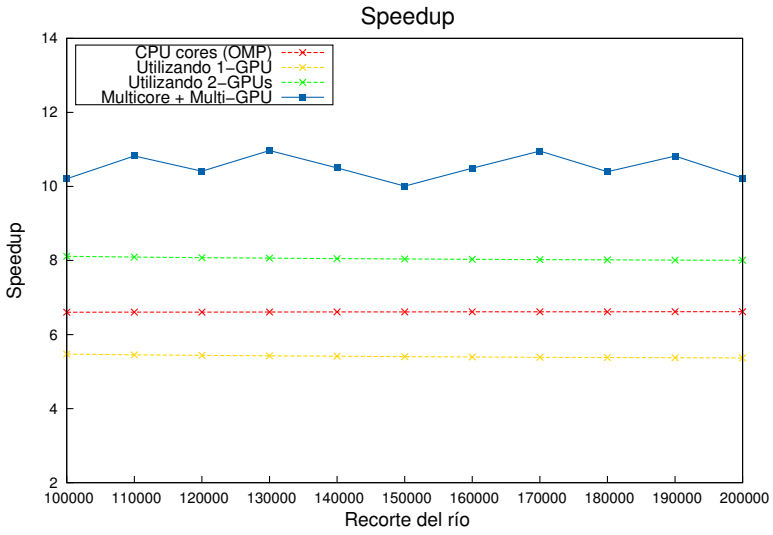


Figura 6.3: Valores de *speedup* conseguido para cada configuración algorítmica respecto del tamaño del problema (recorte del río).

utilizados es: (GPU, GPU, CPU) = (40 %, 40 %, 20 %). La comparación entre los incrementos de velocidad de procesamiento de cada uno de los algoritmos se ha representado en la Figura 6.3. Como se observa en la figura, el *speedup* aumenta significativamente con el tamaño del problema en todos los casos. Los resultados de los experimentos muestran que incluso para el peor caso, es decir, utilizando una sola GPU, se reduce significativamente el tiempo de ejecución con respecto al secuencial.

6.5 Conclusiones

El modelado hídrico de ríos nos ha permitido establecer criterios para comparar cuencas de distintas dimensiones. Esta técnica presenta la utilidad adicional de intentar relacionar los parámetros morfométricos con el funcionamiento hidrológico de las cuencas de ríos. De la misma forma que en los problemas anteriores, el objetivo final consiste es solucionar problemas más grandes. Se ha aportado al problema el modelado computacional paralelo

que permite la reducción del tiempo de ejecución requerido por una simulación con características hidrológicas. En este capítulo se ha aportado un problema diferente, pero la solución global sigue siendo la misma en esencia que la utilizada para resolver los otros problemas en un nodo heterogéneo.

7 Autooptimización de aplicaciones

En nuestro afán por procurar soluciones eficientes y útiles a los problemas computacionales abordados a lo largo de todo este trabajo, hemos tratado el problema que pueden enfrentar los usuarios no expertos para utilizar adecuadamente nuestras soluciones. Hemos tratado de evitar, así pues, que este trabajo caiga en desuso tal como suele suceder frecuentemente, debido a la lejanía entre la solución y el usuario final. Esto nos ha motivado a estudiar, a modo de cierre de todo el trabajo realizado, técnicas llamadas de *auto-optimización* que adaptamos y aplicamos de manera particular a entornos de ejecución heterogéneos compuestos por un subsistema multicore y un subsistema multi-GPU. Con el objetivo de aprovechar los recursos proporcionados por estas arquitecturas heterogéneas, proponemos un modelo de asignación de recursos basado en un modelo de programación paralela que se puede optimizar a sí mismo. El modelo realiza la asignación de recursos de distintas características adaptándose automáticamente a las características del sistema de cómputo con el objetivo fundamental de obtener el mejor rendimiento de la aplicación paralela.

Durante los últimos años se han desarrollado distintas técnicas de optimización de rutinas paralelas que se adapten a las características del sistema de cómputo y reduzcan así el periodo de tiempo necesario para obtener el software optimizado para un nuevo sistema. Para este capítulo de la tesis se

propuso el análisis y la aplicación de técnicas existentes de optimización a los problemas de alto coste computacional presentados. Sin embargo, nuestro objetivo va más allá y modelamos también el tipo de usuario potencial de nuestras soluciones. Tratando a los usuarios de hasta tres maneras diferentes podemos adecuar nuestras herramientas de autooptimización y predecir la optimización que cabe esperar según cada uno de estos tres tipos de usuario.

7.1 Introducción

Las rutinas paralelas que resuelven problemas tales como los tratados en esta tesis son utilizadas principalmente por científicos e ingenieros que no dominan el campo de los sistemas computacionales de alto rendimiento lo suficiente como para adaptar este software al sistema subyacente particular sobre el que trabajan. Por lo tanto, el producto final debe consistir, no solo en rutinas paralelas capaces de aprovechar sistemas paralelos heterogéneos del tipo de los que hemos utilizado en esta tesis, sino que también debe incorporar en sí mismo la capacidad de "autoadaptarse" al entorno en el que trabaja. Es fácil imaginar que existen una serie de parámetros concretos bajo los cuáles una rutina paralela puede obtener mejores prestaciones que con otros. Estos parámetros pueden ser obtenidos mediante avanzadas técnicas de optimización. Las técnicas de optimización se han venido aplicando desde hace tiempo y de manera particular a diferentes disciplinas [33, 34, 92], y muy especialmente a rutinas de álgebra lineal numérica [43, 93], debido a que éstas constituyen el elemento básico de cómputo en la resolución de muchos problemas científicos.

En particular y para llevar a cabo esta parte del trabajo de la tesis, hemos trabajo con un grupo de investigación que disponía de experiencia en la aplicación de técnicas de *optimización automática* u *autooptimización* a métodos de Jacobi unilaterales para resolver el problema de los valores propios [94], a las factorizaciones LU y QR, y a la resolución del problema de mínimos cuadrados de matrices Toeplitz [95]. También disponían de experiencia en aplicar estas ideas a distintos tipos de esquemas algorítmicos, tales como la *programación dinámica* [96, 97] o el *recorrido de árboles de soluciones* [98].

Una de las técnicas más utilizada para el desarrollo de rutinas con capacidad de optimización es la técnica de *parametrización* del modelo del

tiempo de ejecución [24, 99, 100, 101]. Los modelos así obtenidos pueden ser utilizados para tomar decisiones que permitan reducir el tiempo de ejecución de la rutina. La metodología seguida consiste en identificar dos tipos de parámetros: parámetros *algorítmicos* y parámetros del *sistema*. Mediante la identificación de estos parámetros se analiza 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 algorítmicos*. La idea básica de esta optimización consiste en modelizar el tiempo de ejecución de las rutinas en la forma:

$$t(s) = f\left(\ s, \quad AP, \quad SP \ \right) \ , \tag{7.1}$$

donde s representa el tamaño del problema, SP (System Parameters) representa los parámetros que reflejan las características del sistema, y AP (Application Parameters) representa a los parámetros que influyen en el algoritmo y cuyos valores deben ser seleccionados adecuadamente para obtener un tiempo de ejecución reducido [94, 96, 99]. Se han estudiado técnicas mediante las cuales las rutinas paralelas pueden adaptarse automáticamente a las características del sistema paralelo donde se ejecutan, pudiéndose ejecutar en un tiempo muy próximo al óptimo. El objetivo es claro: pretendemos proporcionar 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 se tenga sobre computación paralela. Los entornos de trabajo de este capítulo son los mismos que los utilizados hasta ahora (Sección 2.2): entornos heterogéneos de gran capacidad computacional y muy comúnmente utilizados hoy en día por los potenciales usuarios de estas rutinas.

Ofrecemos una visión general del sistema de autooptimización propuesto en la sección siguiente. La Sección 7.3 muestra cómo aplicar esta metodología para obtener una rutina eficiente basada en los parámetros del modelo que aproveche eficientemente la heterogeneidad de los nodos de cómputo. Los resultados experimentales que permiten validar la propuesta figuran en la Sección 7.4. Terminamos con algunas conclusiones en la Sección 7.5.

7.2 Sistema de autooptimización propuesto

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. Esta propuesta se inspira en las técnicas de autooptimización del trabajo [50], que en esta tesis extendemos a rutinas científicas que siguen un esquema heterogéneo multicore+multi-GPU. Es así como surge el modelo matemático del tiempo de ejecución mostrado en la Ecuación 7.1. Parámetros *SP* típicos con los que se reflejan las características del sistema computacional y de comunicaciones 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 y de transferencia de datos entre CPU y GPU. Algunos parámetros *AP* típicos son: el número de cores de un procesador multicore, el número de procesos a poner en marcha y su mapeo en el sistema físico, 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, o el porcentaje de carga de trabajo asignada a una GPU. De cara a obtener un modelo más realista, se puede tener en cuenta que los valores *SP* están influenciados por los valores de *AP*. 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 la Ecuación 7.1 queda de la siguiente forma:

$$t(s) = f\left(\ s, \quad AP, \quad g(s, AP) \ \right) \ . \tag{7.2}$$

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, identificando los parámetros *SP* que influye 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, teniendo una mayor precisión a través de la realización de una mayor cantidad de experimentaciones.

Diseño del sistema autooptimizado

La idea general del sistema autooptimizado consistiría en crear una rutina de resolución del problema específico, que antes de ser ejecutada, recibe información de los parámetros de autooptimización obtenidos en la plataforma durante la fase de *instalación*. Después, se hace uso de esa información en la etapa posterior de *ejecución* para conseguir mayor eficiencia para el algoritmo. Así pues, 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* (SP). Entre los parámetros del sistema a determinar pueden estar, por ejemplo, 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* (AP) para resolver el problema con un tiempo próximo al óptimo. Para esto, la rutina sustituye en el modelo teórico del tiempo posibles valores para parámetros tales 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.

El sistema de autooptimización se puede articular como sigue:

Estimación de los Parámetros del Sistema (SP): Se estiman los cos-

tes de las operaciones aritméticas básicas del algoritmo para cada uno de los métodos propuestos. Esto se realiza a través de experimentos que devuelven tiempos de ejecución para cada una de las diferentes estrategias, tamaños del problema y de bloques pequeños, con objeto de aproximar el comportamiento de la rutina a una expresión analítica.

Almacenamiento de los datos de los Parámetros (SP): La información relativa a los parámetros SP se almacena en una estructura de datos que describe y determina las características del sistema donde fue instalada la rutina. Esta estructura servirá para sustituir los valores de los parámetros calculados en el modelo teórico en la fase de ejecución.

Predicción de los Parámetros (AP) óptimos: Se sustituyen los valores de los parámetros SP en el modelo teórico implementado para predecir los parámetros algorítmicos.

7.3 Aplicación de la metodología

A continuación comentamos algunos aspectos generales de la metodología de autooptimización y de cómo llevarla a cabo en la práctica. Para mostrar la utilidad del modelo propuesto utilizamos como ejemplos los tres casos de estudio que hemos tratado en esta tesis: representación del relieve (Capítulo 3), representación de variables meteorológicas del clima–temperatura (Capítulo 5), y modelo hídrico de ríos (Capítulo 6).

Los parámetros algorítmicos para la autooptimización presentada en nuestro modelo son: el número de cores de la CPU (c) y el porcentaje de carga de trabajo asignada a GPU, definida como *workload* (w). Estos dos parámetros recogen las características del rendimiento de la aplicación y del entorno de ejecución en el que será ejecutada la aplicación. Los cores CPU y la carga de trabajo se utilizan para mostrar la escalabilidad de la aplicación. Estos parámetros también son reflejo de las razones por las cuales el rendimiento de la aplicación disminuye cuando utilizamos mal sus valores. Así pues, en nuestros problemas, el conjunto de parámetros algorítmicos son

$$AP = \{c, w\} \ ,$$

luego, según la Ecuación 7.2, el tiempo total de ejecución es

$$t(s) = f\left(\ s,\quad AP,\quad SP\ \right) = f\left(\ s,\quad c,\quad w,\quad g(s,c,w)\ \right)\ .$$

El número óptimo de cores CPU y la carga de trabajo no son valores constantes, sino que dependen del entorno de ejecución y del tamaño del problema. Una buena selección de estos valores es de gran importancia. El desarrollo de rutinas "autooptimizables" hace fácil una eficiente utilización de estas rutinas por usuarios no expertos. En la metodología de autooptimización los mejores valores de los parámetros AP se buscan cuando la rutina está siendo instalada en el sistema. Se resuelven problemas de distintos tamaños y para diferentes AP, y sus mejores valores son incorporados a la rutina antes de su utilización. Se guardan el modelo y los parámetros SP calculados, y luego se obtienen los parámetros AP en tiempo de ejecución.

En este trabajo se consideran dos modelos de autooptimización para conseguir una distribución balanceada de las cargas de trabajo entre CPU y GPU, teniendo en cuenta las características del sistema computacional: un modelo que combina el coste teórico del tiempo de ejecución con experimentos para adaptar el modelo al sistema, y que llamaremos *modelo empírico* [102], y un *modelo experimental* que utiliza una búsqueda guiada [103].

Modelo empírico

Como se ha dicho antes, los algoritmos son estudiados de forma teórica y experimental a fin de determinar la influencia de los parámetros algorítmicos y del sistema. La parte más importante de este estudio es la incorporación del modelo analítico en función del tiempo de ejecución, teniendo como parámetros de entrada el tamaño del problema, los parámetros del sistema, y los parámetros algorítmicos. Se puede utilizar este modelo teórico del tiempo de ejecución para obtener la división óptima del problema en el sistema heterogéneo. Hay que modelizar el tiempo de ejecución de las aplicaciones. Para ello se propone un modelo simplificado del tiempo de ejecución paralelo dado por:

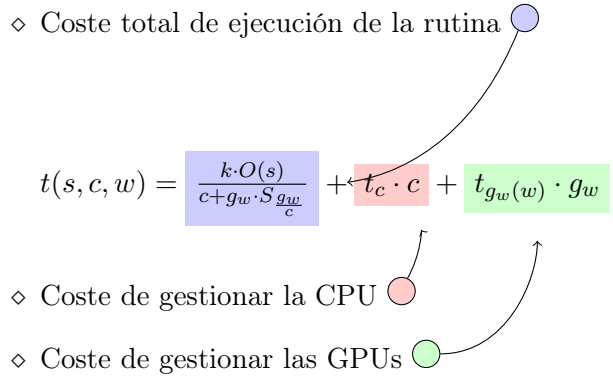
$$t_{par} = t_{rt} + CPU_{mt} + GPU_{mt}\ ,$$

Tabla 7.1: Parámetros del modelo de tiempo de ejecución (Ecuación 7.3).

Símbolo	Descripción
k	Parámetro de computación para cada entorno de ejecución
O	Complejidad algorítmica
s	Número de operaciones
c	Número de cores de la CPU
w	Porcentaje de la carga de trabajo asignada a una GPU
g_w	Número de GPUs
t_c	Coste de gestión de los *threads* en la CPU
t_{g_w}	Coste de gestión del kernel en la GPU
$S_{\frac{g_w}{c}}$	*speedup* de la GPU con respecto a 1 core de la CPU

donde t_{rt} es el coste total de ejecución de la rutina, y CPU_{mt} y GPU_{mt} son los costes de gestionar la CPU y la GPU, respectivamente.

Este modelo analítico de predicción del tiempo de ejecución es una función de parámetros de entrada, y requiere una información directa sobre la arquitectura de la máquina objetivo. Para determinar con precisión el rendimiento de la aplicación se han utilizado técnicas basadas en el concepto de *evaluación comparativa* [104]. Los parámetros del modelo del tiempo de ejecución de la rutina se pueden modelizar mediante la Ecuación 7.3.



El modelo para el tiempo de ejecución se ha estimado a partir del coste de la complejidad secuencial del algoritmo $t(s) = k \cdot O(s)$, dividiendo por el número de cores CPU (c), sumando el número de GPUs (g_w), multiplicando por la aceleración entre la GPU con respecto a la CPU ($S_{\frac{g_w}{c}}$).

Este coste es influenciado por una constante k, que representa el coste de una operación aritmética básica. El tiempo de computación obtenido de la fórmula es el tiempo necesario para balancear el trabajo entre los recursos computacionales, de tal forma que el valor de w necesita estar muy bien seleccionado.

El modelo de tiempo de ejecución representado en la Ecuación 7.3 depende de los tiempos de inicialización y de gestión de la CPU (t_c) y de la GPU ($t_{g_w(w)}$), respectivamente.La Tabla 7.1 resume el significado de cada término utilizado en dicha ecuación. La gestión de los cores CPU será significativa cuando tengamos tamaños de problema muy grandes. Esta decisión se justifica a través de los experimentos, en los cuáles se ha comprobado que el modelo desarrollado se ajusta mejor con tamaños grandes del problema. La gestión de la GPU incluye el coste de la transferencia de datos entre la CPU (*host*) y la(s) GPU(s) (*device*), y se representa en la fórmula como un término agregado al coste de computación. Podríamos trabajar con la superposición de la computación y transferencia de los datos, pero los resultados obtenidos para este sencillo modelo son satisfactorios para los tipos de algoritmos con los que estamos trabajando. El modelo empírico es muy simple y no tiene en cuenta los factores de la arquitectura del entorno de ejecución. Por el contrario, la ecuación refleja el coste computacional a través de los parámetros algorítmicos y del sistema.

Como el valor de los parámetros influye en el modelo desarrollado, su cálculo es uno de los factores clave en el desarrollo de la metodología de autooptimización. Tiene que tenerse en cuenta que las aplicaciones de alto rendimiento ejecutadas en diferentes plataformas pueden ser utilizadas por usuarios sin ninguna experiencia. Durante el proceso de instalación, llevamos a cabo experimentos con diferentes combinaciones de parámetros: s, c y w, para los cuales se ha tratado de minimizar la diferencia de tiempo entre los valores teóricos y experimentales mediante el uso de un procedimiento de mínimos cuadrados. Se hace un ajuste por mínimos cuadrados para estimar los valores de los parámetros del sistema para nuestro programa en el sistema donde se está instalando. Un ejemplo lo constituye el cálculo de la constante k, para la cual se obtienen diferentes valores ajustando el coeficiente a partir de la propia computación de la aplicación utilizando solamente la CPU [105]. Esta metodología requiere muchos experimentos con el fin de tener una buena estimación de los parámetros. Es un proceso iterativo, y una vez que el tiempo de ejecución se ha estimado con nuestro modelo, se conseguirá

hallar los valores de c y w, que serán sustituidos en la fórmula para luego recompilar la rutina definitiva. Esto es, los valores finales de los parámetros se incluyen en la rutina dando al usuario el valor del tiempo teórico que será utilizado para resolver el problema. Hay que añadir que el formato de nuestro modelo está simplificado dado que ha sido suficientemente útil en las aplicaciones abordadas en esta tesis. Esta simplificación puede dar lugar a alguna discrepancia puntual entre el tiempo estimado y el real, solo en el caso de algunos tamaños de problema muy grandes. Esto es debido a la omisión de algunos aspectos arquitectónicos en nuestro modelo.

Modelo experimental

A partir de un modelo experimental basado en búsqueda guiada se persigue la distribución balanceada de las cargas de trabajo entre CPU y GPU. El primer paso para la implantación del modelo consiste en desarrollar una metodología simple que permita medir las potencias relativas de cómputo de las máquinas pertenecientes a un grupo de recursos con características heterogéneas. La técnica desarrollada consiste en lanzar un conjunto de ejecuciones con la rutina híbrida que representa a la propia aplicación sobre un conjunto de recursos. Estas ejecuciones se realizan bajo la definición de lo que llamamos *Conjunto de Instalación*, que reúne distintos tamaños de problema que son significativos, desde tamaños reducidos hasta suficientemente grandes. Las ejecuciones toman valores de dicho conjunto y varían, a su vez, las porciones de reparto de trabajo que se asignan al subsistema CPU y GPU.

La estrategia de asignación de datos a los *threads* (serán a su vez mapeados en los recursos) se basa en una decisión tomada antes de la ejecución. Los datos se asignan a cada proceso conociendo de antemano su orden de ejecución. La asignación consiste en descomponer los cálculos en un número determinado de tareas según una estructura de memoria compartida. Los *threads* inicializados tienen como principales características: el agrupamiento previo de un conjunto de datos en un único *thread*, y la selección de un orden de ejecución de los datos. La meta final consiste en mapear las tareas sobre los recursos según sus prestaciones, distribuyendo la carga según la capacidad de cómputo de cada recurso calculada anteriormente. El esquema mostrado en la Figura 7.1 ilustra mediante un ejemplo la estrategia de partición de la carga de trabajo entre un grupo de n recursos en una arquitectura heterogénea.

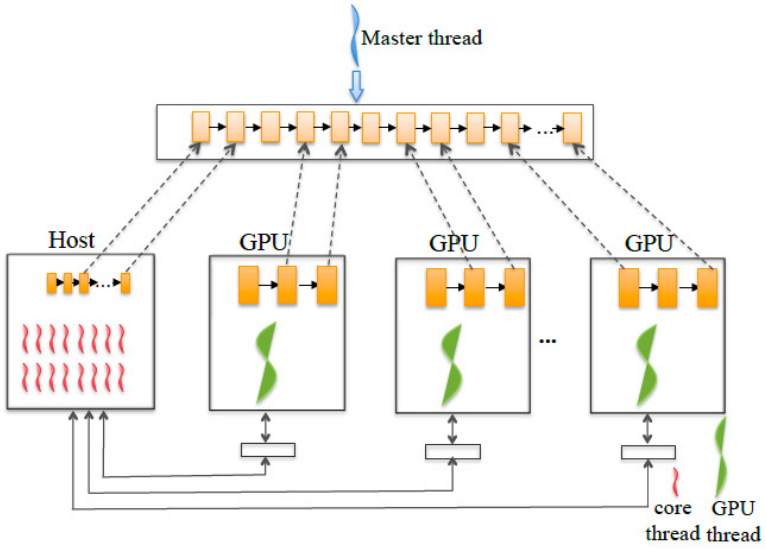


Figura 7.1: Esquema de partición de la carga de trabajo en una arquitectura heterogénea.

Este modelo, en lugar de obtener índices absolutos de potencia para cada recurso basados en referencias de fabricación, trata de obtener una medida de la ejecución de la aplicación real, para obtener así la capacidad de cómputo relativa de ese recurso para la aplicación objetivo. Al final, estas pruebas ofrecen un valor característico, expresado en porcentaje de capacidad de cómputo, para cada recurso perteneciente al conjunto de computación elegido. Este modelo sirve como herramienta para asignar tareas de manera eficiente, ya que, en función de las características heterogéneas de las máquinas, se consigue obtener un mayor balanceo de la carga.

7.4 Resultados experimentales

En esta sección se aplican los esquemas propuestos de autooptimización con el fin de validar la metodología. Se incluyen estudios experimentales, explicaciones y comentarios detallados de la optimización en el primer caso

Tabla 7.2: Parámetros de computación para cada entorno de ejecución (k) para la representación del relieve.

Parámetros k estimados	
ELEANORRIGBY	0.0059
MICROMACHIN	0.0271

de estudio referente a la representación del suelo en los entornos de ejecución ELEANORRIGBY (denotado en las tablas con la letra **E**) y MICROMACHIN (denotado en las tablas con la letra **M**). Sin embargo, también se muestra algún resultado para los otros dos casos de estudio: la representación de variables meteorológicas del clima y el modelo computacional hídrico de ríos. Para facilitar las experimentaciones se ha decidido crear una librería que permita aplicar la autooptimización a los problemas aplicados. Esta librería la hemos denominado AUTO y su manual y explicación se adjuntan en un Apéndice al final de esta tesis.

Modelo empírico: Proceso de instalación

Cuando se instala la rutina en una plataforma especifica, se deben obtener experimentalmente los diferentes valores de t_c, t_{gw} y k. Los valores de los costes de comunicación se pueden estimar con un *benchmark* simple para diferentes tamaños de datos a transferir. Los valores del coste de comunicación se estiman por regresión lineal, y los del coeficiente k por mínimos cuadrados con los resultados experimentales obtenidos con la rutina secuencial aplicada a unos tamaños de problema especificados en el Conjunto de Instalación. La Tabla 7.2 muestra los parámetros k estimados.

Los experimentos son conducidos de forma similar en los dos modelos. Para el tamaño de problema más pequeño del Conjunto de Instalación se empieza con unos valores para c y w, y se van incrementando en una cierta cantidad hasta que el tiempo de ejecución exceda de un umbral. De esta forma, además de reducir el tiempo de testeo, se consigue que los experimentos para estimar los valores de los parámetros se realicen con valores más cercanos a los óptimos, y de esta forma los valores obtenidos reflejarán mejor el comportamiento de la rutina para los valores con los que se usará.

Tabla 7.3: Tiempo de ejecución obtenido en la instalación para diferentes valores de AP (en segundos) para el modelo empírico en el caso de la representación del relieve.

E	$w = 20$		$w = 30$		$w = 40$		$w = 45$	
N	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$
10	16	18.13	18	19.59	**18**	**16.78**	18	19.87
12	16	9.73	22	12.94	**22**	**5.97**	22	15.36
14	18	8.58	22	13.36	**24**	**2.64**	24	13.50
16	18	14.67	24	20.83	**24**	**6.76**	24	15.73
18	18	28.01	24	35.36	**24**	**18.35**	24	36.97
20	18	48.59	24	56.94	**24**	**37.40**	24	48.77
M	$w = 10$		$w = 15$		$w = 20$		$w = 22$	
N	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$
10	12	83.66	14	88.57	**14**	**77.64**	14	78.83
12	12	241.17	16	295.78	**14**	**74.48**	16	262.47
14	14	395.79	16	519.29	**16**	**107.61**	16	432.09
16	14	537.54	18	739.09	**16**	**177.03**	20	587.70
18	14	666.41	18	955.18	**16**	**282.73**	20	729.30
20	14	782.40	18	1167.57	**16**	**424.72**	20	856.89

En este proceso se instala con información del sistema para el que se está instalando. Lo primero es experimentar variando el número de cores utilizados (c). Una vez obtenido un número adecuado, se varía la carga de trabajo asignada (w) a las GPUs para grados de polinomio pequeños que varían de 10 hasta 20. Después, se aplica un ajuste por mínimos cuadrados. Se realiza la instalación con una serie de experimentos reducidos, y después se llevan a cabo experimentos con otros tamaños distintos para validar el modelo (Por ejemplo: Se valida con los valores 22, 24, 26, 28, 30, 32, 34, 36, 38, 40). Así se puede comprobar que las decisiones tomadas son conformes a la información que se obtuvo en la instalación de la rutina. El *benchmark* obtiene así los tiempos de ejecución con las mismas operaciones básicas y almacenamiento de los datos que se utilizarán en la rutina que se está modelizando. La Tabla 7.3 muestra los tiempos de ejecución para algunos valores de los parámetros algoritmos, para hallar el Conjunto de Instalación formado por los parámetros AP en los dos entornos de ejecución estudiados.

Los valores de la entrada en el proceso de instalación del Conjunto de Instalación pueden ser muy numerosos por lo que se prueba a partir

del tamaño más pequeño como con valor inicial y se continúa con valores del cálculo de la potencia de cómputo realizado anteriormente para c para reducir el número de posibles valores. Los valores de w son distintos en los dos sistemas, debido a que el número de GPUs es también distinto en los dos entornos probados. Es por eso que a veces hay variaciones grandes en el valor de w. El sistema ELEANORRIGBY posee 2 GPUs iguales, luego los valores de reparto son hallados en función de la similitud del cálculo. La misma idea es utilizada para el sistema MICROMACHIN que dispone de 4 GPUs. Los parámetros AP óptimos obtenidos en el proceso de autooptimización son:

- ◇ para ELEANORRIGBY, utilizando 2-GPUs + 2-Procesadores Hexacore: número de *threads* utilizados (c) = 24, y carga de trabajo asignada (w) = (GPU, GPU, CPU) = (40 %, 40 %, 20 %); y
- ◇ para MICROMACHIN, utilizando 4-GPUs + 2-Procesadores Quadcore: número de *threads* utilizados (c) = 16, y carga de trabajo asignada (w) = (GPU, GPU, GPU, GPU, CPU)=(20 %, 20 %, 20 %, 20 %, 20 %).

Experimentalmente se ha comprobado que el modelo de la Ecuación 7.3 predice bien el tiempo de ejecución en los dos sistemas probados. Esto a pesar de la irregularidad en el proceso de transferencia de datos entre CPU a GPU y que el tipo, forma y disposición del bus de comunicación es un factor que limita la predicción de los modelos. El tiempo de instalación en MICROMACHIN, con el modelo empírico, es de 300 segundos mientras que en ELEANORRIGBY es de 500 segundos. Por tanto, comparado con los tiempos de ejecución reales, los tiempos de instalación no son altos, no existiendo, además, diferencias significativas entre los dos entornos.

A partir de los experimentos se pueden sacar algunas observaciones: i) El valor del parámetro c depende del tamaño de problema utilizado en el entorno de ejecución, y ii) para cada valor del tamaño del problema y diferentes valores de c y w obtenemos un valor diferente para k en cada entorno. Considerar esta variabilidad es fundamental para tomar buenas decisiones en la selección de los AP óptimos.

Modelo experimental: Proceso de instalación

En el modelo experimental se utilizan varios valores de parámetros algorítmicos. Para el número de cores en la CPU, el valor fue de 1 a 24 para

Tabla 7.4: Tiempo de ejecución obtenido en la instalación para diferentes valores de AP (en segundos) para el modelo experimental en el caso de la representación del relieve.

E	$w = 20$		$w = 30$		$w = 40$		$w = 45$	
N	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$
10	16	19.87	18	19.58	18	8.13	**18**	**3.13**
12	16	14.36	18	12.94	22	9.73	**22**	**4.73**
14	16	35.04	22	13.35	22	8.57	**24**	**3.57**
16	18	37.26	22	20.83	24	14.66	**24**	**9.66**
18	18	36.97	24	35.35	24	28.00	**24**	**23.00**
20	18	68.76	24	56.94	24	48.58	**24**	**43.58**
M	$w = 10$		$w = 15$		$w = 20$		$w = 22$	
N	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$	c	$t(N, n, c, w)$
10	12	68.57	14	78.83	14	164.01	**14**	**58.00**
12	12	295.78	14	262.47	16	167.12	**14**	**189.25**
14	12	519.29	16	432.09	16	197.45	**14**	**208.34**
16	14	739.09	18	587.70	18	267.30	**16**	**214.12**
18	14	955.18	20	729.30	18	372.90	**16**	**328.00**
20	14	1167.57	20	856.89	18	514.05	**16**	**448.21**

ELEANORRIGBY y de 1 a 16 para MICROMACHIN (con *Hyper-Threading* activado en los dos sistemas), para obtener el mejor tiempo de ejecución. Luego, se comprueba el porcentaje de carga de trabajo asignado a la GPU, variando el valor de 10 hasta 45. Las entradas de los tamaños de problema (grados del polinomio) para nuestra experimentación fueron $10, 12, \ldots, 20$. La Tabla 7.4 muestra el Conjunto de Instalación con los parámetros utilizados para estimar la instalación a partir de los parámetros AP en los dos entornos de ejecución estudiados. De la misma forma que para el modelo anterior, se lleva a cabo una serie reducida de experimentos para después experimentar con otros tamaños distintos que permitan validar el modelo.

En el momento de la ejecución, cuando se va a aplicar la rutina para un cierto tamaño de problema, se decide la partición utilizando la información almacenada en la instalación, seleccionando el parámetro almacenado para el tamaño de problema del Conjunto de Instalación más cercano al tamaño actual. Los parámetros AP óptimos obtenidos en el proceso de autooptimización son:

- ◇ para ELEANORRIGBY, utilizando 2-GPUs + 2-Procesadores Hexacore: número de *threads* utilizados (c) = 24, y carga de trabajo asignada (w) = (GPU, GPU, CPU) = (45 %, 45 %, 10 %); y
- ◇ para MICROMACHIN, utilizando 4-GPUs + 2-Procesadores Quadcore: número de *threads* utilizados (c) = 16, y carga de trabajo asignada (w) = (GPU, GPU, GPU, GPU, CPU)=(22 %, 22 %, 22 %, 22 %, 12 %).

El tiempo de instalación con el modelo experimental es de 14980 segundos en MICROMACHIN, y de 840 segundos en ELEANORRIGBY. Estos tiempos de instalación son un poco altos debido a la gran cantidad de ejecuciones propuestas para obtener valores cercanos al óptimo. La diferencia de los tiempos de instalación se justifica por que el poder de cómputo de ELEANORRIGBY, que es muy superior al de MICROMACHIN. A pesar de ello, la diferencia entre a la decisión que se toma utilizando el modelo experimental y el empírico es muy pequeña.

Validación de los modelos Experimental y Empírico

Para validar las decisiones que se tomarían durante la ejecución utilizando la información guardada durante la instalación, usamos un conjunto de tamaños de problema distintos al utilizado para la instalación. El modelo de la rutina paralela debe proporcionar información útil sobre el valor de c y w a usar independientemente del tamaño de problema que se quiera resolver y del tamaño y de la velocidad relativa (CPU/GPU) del sistema donde se trabaje. La validación de los modelos para la representación del relieve se analiza en la Figura 7.2. Tenemos los tiempos de ejecución en la parte de arriba y el *speedup* en la parte de abajo. El *speedup* se calcula teniendo como referencia el tiempo de ejecución secuencial en un core. La parte izquierda muestra las gráficas en ELEANORRIGBY y la derecha en MICROMACHIN. También hemos sacado tiempos y *speedup* para el caso de la representación de variables meteorológicas del clima (temperatura) en la Figura 7.3 y para el caso del modelo hídrico de ríos en la Figura 7.4.

En los dos sistemas se muestran los tiempos solo para los cores CPU o utilizando, además, 1, 2, 3 ó 4 GPUs, número que depende del sistema utilizado. En la curva etiquetada como ÓPTIMO se muestran el mínimo tiempo de ejecución obtenidos experimentalmente y el valor máximo de *speedup*. Las figuras del *speedup* muestran la discrepancia entre los dos sistemas cuando se aplica la autooptimización. Esta diferencia está justificada por la dificul-

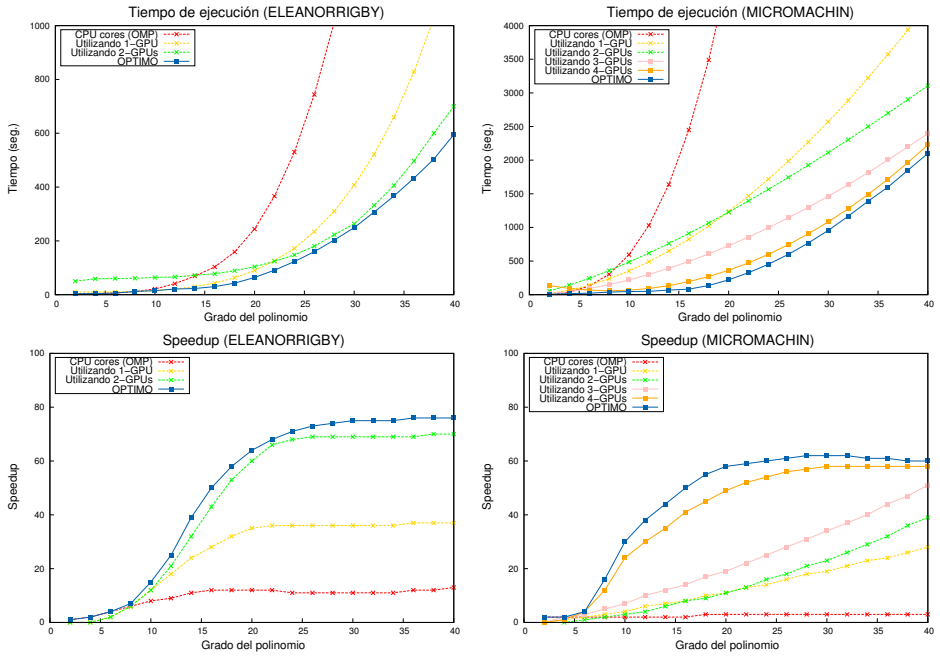


Figura 7.2: Tiempo y *speedup* obtenidos con la autooptimización aplicada a la representación del relieve.

tad de medir el tiempo de inicialización de los *threads* en los cores (t_c) y el tiempo de procesamiento en las GPUs (t_{g_w}). En cualquier caso, pensamos que la precisión alcanzada es importante, más si cabe teniendo en cuenta la simplicidad del modelo utilizado.

La Figura 7.5 muestra la desviación entre los distintos modelos utilizados en diferentes configuraciones de hardware y parámetros algorítmicos. La figura muestra los tres casos bajo estudio. Se compara el tiempo obtenido con cada uno de los modelos de autooptimización, esto es, el EMPÍRICO y el EXPERIMENTAL, con el ÓPTIMO. Con EMPÍRICO se muestran los valores óptimos de tiempo de ejecución dados por la Ecuación 7.3. En el modelo EXPERIMENTAL se muestran los datos obtenidos usando los valores de los parámetros calculados de forma experimental. La desviación media respecto al óptimo tiene un valor aceptable en los dos entornos y para los dos modelos. La desviación media entre el tiempo de ejecución obtenido usando

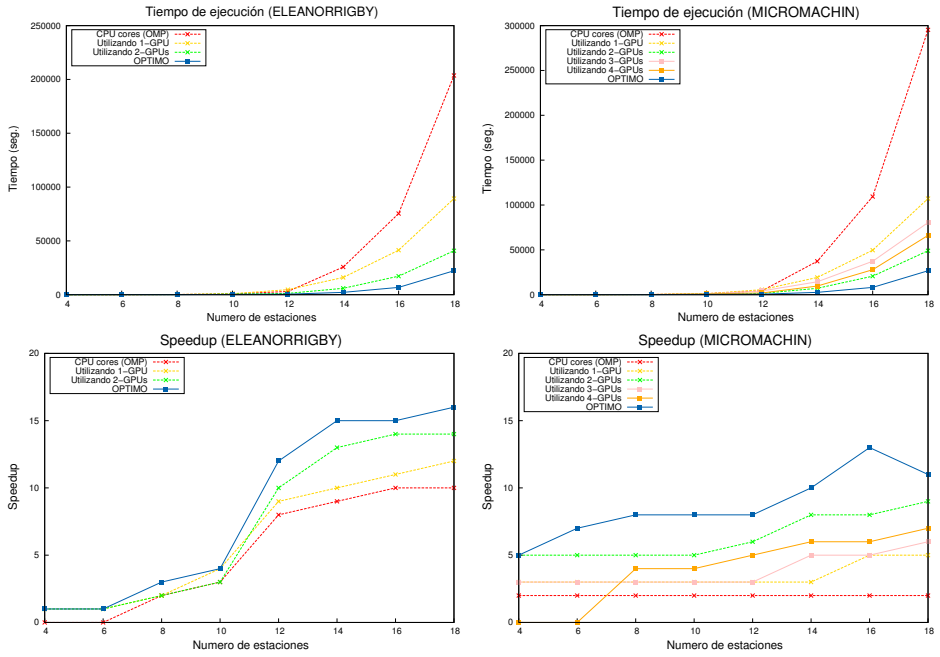


Figura 7.3: Tiempo y *speedup* obtenidos con la autooptimización aplicada a la representación de variables meteorológicas del clima.

el modelo EMPÍRICO y el ÓPTIMO tiene un promedio del 13 % para tamaños de problemas medianos y grandes, y del 5 % para tamaños pequeños en ELEANORRIGBY; y alrededor del 15 % para tamaños de problemas medianos y grandes, y del 6 % para tamaños pequeños en MICROMACHIN. La desviación media entre el tiempo de ejecución obtenido usando el modelo EXPERIMENTAL y el ÓPTIMO tiene un promedio del 20 % para tamaños de problemas medianos y grandes, y del 8 % para tamaños pequeños en ELEANORRIGBY; y al alrededor del 15 % para tamaños de problemas medianos y grandes, y del 7 % para tamaños pequeños en MICROMACHIN. A través de la desviación se puede ver que el modelo EXPERIMENTAL posee un valor más exacto que el modelo EMPÍRICO, lo que se obtiene a costa de un mayor tiempo de instalación.

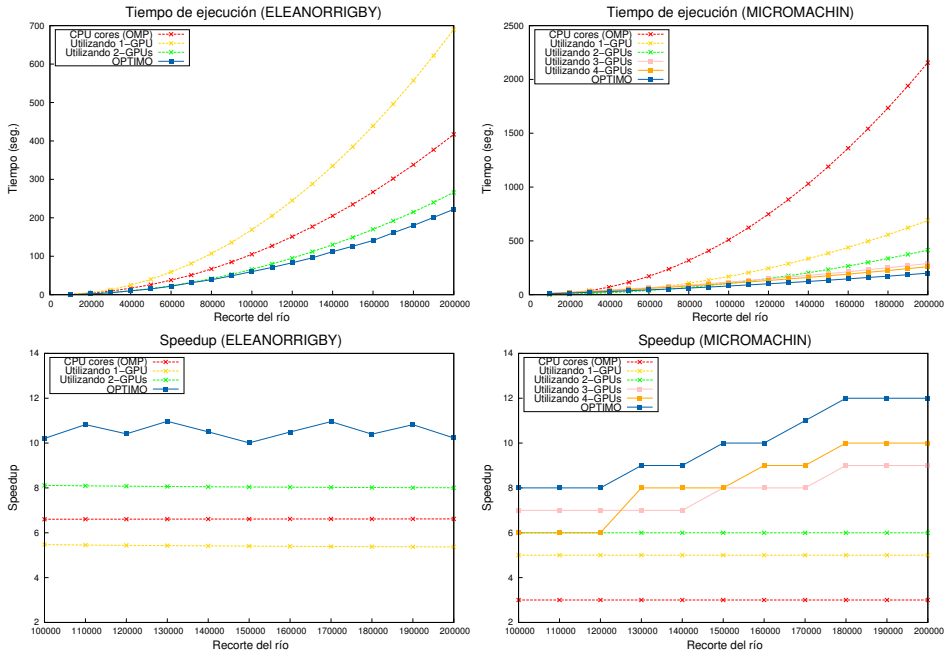


Figura 7.4: Tiempo y *speedup* obtenidos con la autooptimización aplicada al modelo hídrico del ríos.

La Figura 7.6 muestra la representación gráfica de los tiempos de ejecución teniendo como base la elección de los parámetros AP para tres tipos de usuarios. El usuario llamado **UsuarioAT** utiliza la autooptimización y aplica los parámetros óptimos obtenidos en el proceso de instalación con el modelo teórico presentado anteriormente. En este caso, los usuarios tienen una forma de predecir el tiempo de ejecución, incluso antes de ejecutar el algoritmo, y en la mayoría de los casos esos valores pueden estar cerca del óptimo. El segundo tipo de usuario, llamado **UsuarioEO**, sabe predecir el óptimo. Este tipo de usuario obtiene el tiempo de ejecución sustituyendo los parámetros algorítmicos para el entorno de ejecución elegido. El último grupo de usuarios, **UsuarioNE**, está compuesto por usuarios no expertos en paralelismo. Este tipo de usuario utiliza 2-GPUs eligiendo de forma aleatoria los parámetros AP. De esta forma, casi siempre obtiene valores que están lejos del óptimo experimental. Puede verse que los tiempos de ejecu-

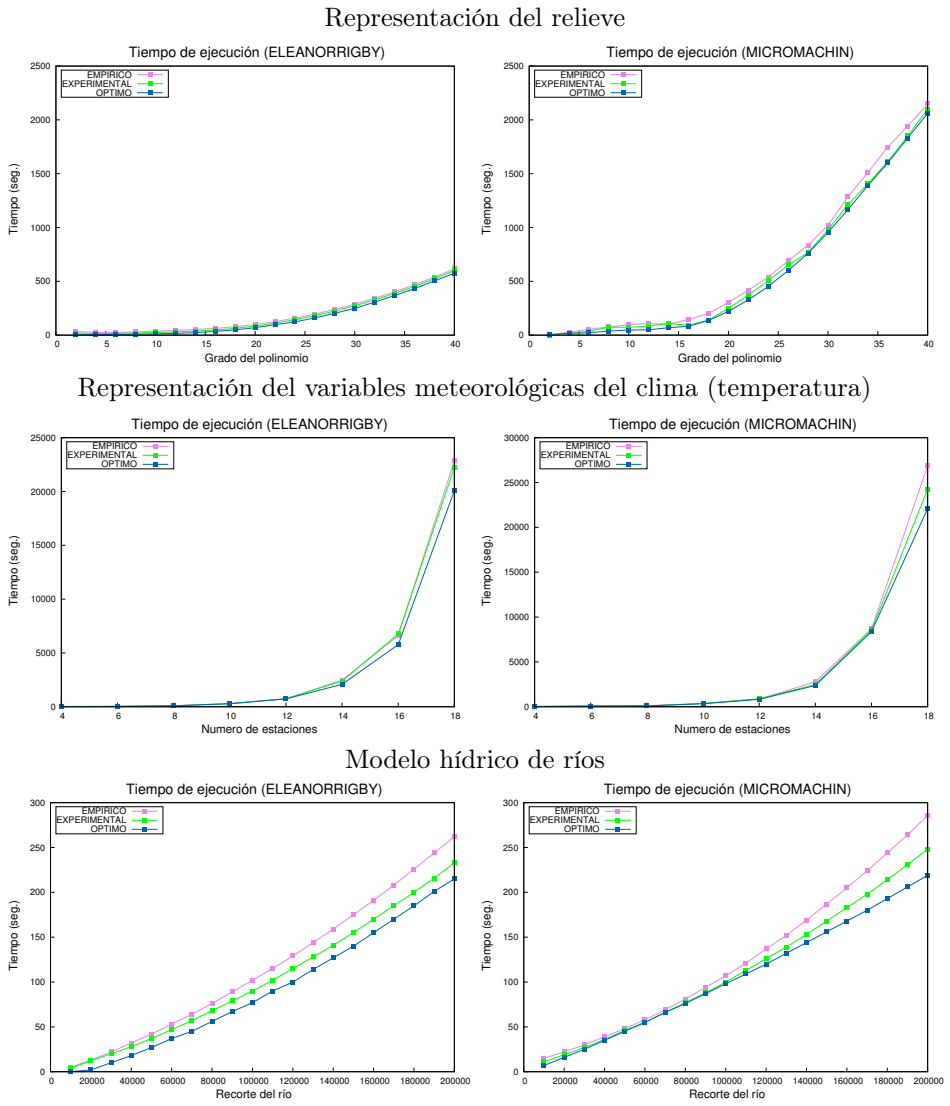


Figura 7.5: Diferencia entre modelos para la autooptimización aplicada a los tres casos de estudio.

ción obtenidos por los usuarios que utilizan la autooptimización no están demasiado lejos del valor óptimo y, al mismo tiempo, están lejos de los

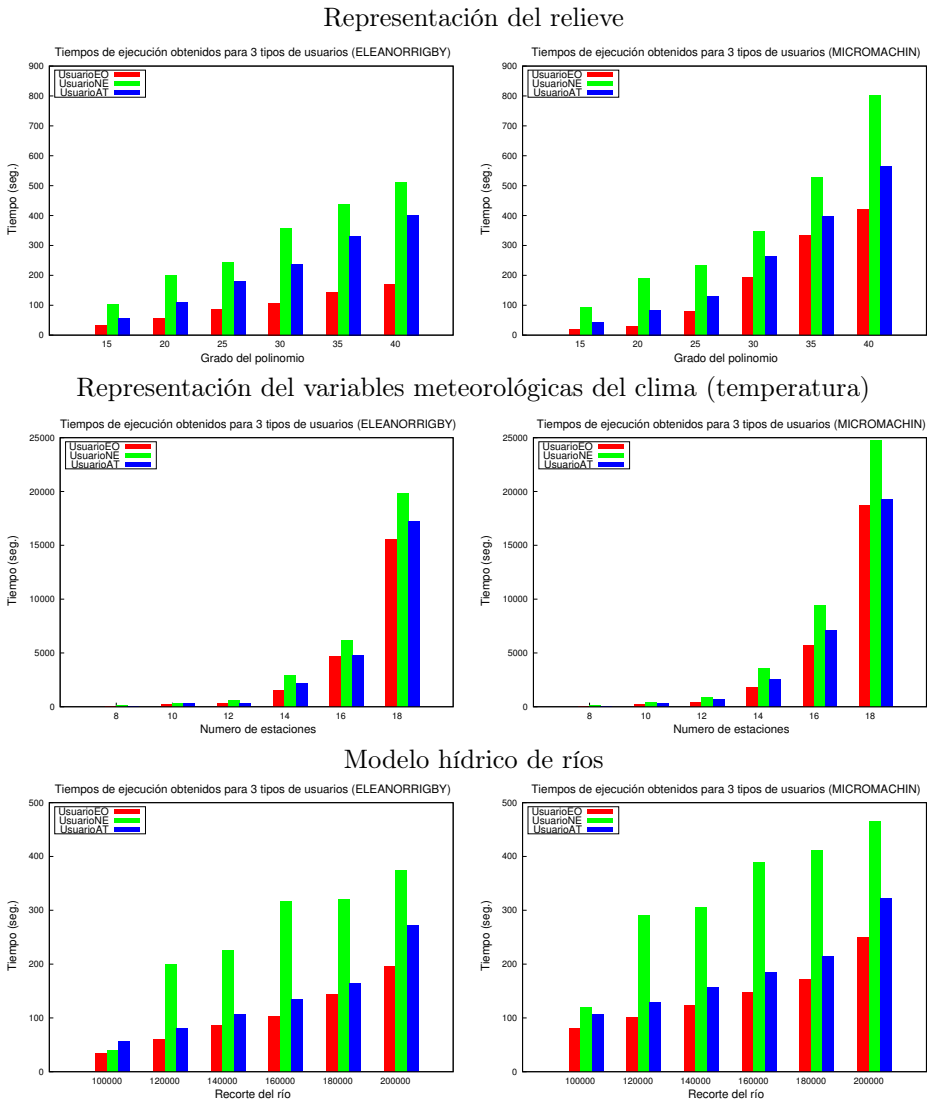


Figura 7.6: Tiempo obtenido mediante autooptimización según tipo de usuario para los tres casos de estudio.

valores obtenidos por los usuarios no expertos.

7.5 Conclusiones

En este capítulo se han aplicado dos metodologías de autooptimización que tienen por objeto balancear la distribución de trabajo entre los subsistemas CPU y GPU. Uno de los modelos se basa en la realización de experimentos de forma guiada para reducir el tiempo de experimentación en la búsqueda de la mejor distribución para un conjunto de tamaños de problema predeterminado. Los mejores valores obtenidos se almacenan para ser utilizados en tiempo de ejecución y poder así determinar la forma de repartir las tareas entre CPU y GPU. El segundo método está basado en un modelo teórico del tiempo de ejecución de la rutina híbrida. El modelo contiene parámetros que reflejan el comportamiento de la rutina en el sistema, y los valores de estos parámetros se estiman para cada sistema concreto por medio de experimentos. En la ejecución se utiliza un modelo adaptado al sistema para decidir el particionado de los datos.

Hemos adaptado estas técnicas y las hemos probado con los tres casos de estudio que hemos tratado en esta tesis. Podemos afirmar que, en general, con los dos modelos se obtienen distribuciones muy cercanas a la óptima en los dos sistemas heterogéneos en los que se ha experimentado. En nuestra evaluación hemos constatado que el modelo experimental es algo más preciso, pero a costa de un mayor tiempo de instalación.

Conclusiones 8

En este último capítulo resumimos las principales conclusiones que se pueden extraer del trabajo realizado con el fin de tener una visión general de los resultados obtenidos. También se enumeran las aportaciones generadas en el marco de esta tesis, así como los proyectos de investigación en los que se han desarrollado. Por último, se esbozan las diferentes vías de investigación que se proponen como trabajo futuro.

8.1 Conclusiones y resultados

El uso de modelos computacionales paralelos aplicados a problemas reales se ha estudiado utilizando como ejemplos: la representación del relieve, la representación de variables meteorológicas del clima, y un modelado computacional hídrico de ríos. A continuación resumimos algunas conclusiones específicas por temas:

Representación del relieve

Entre las herramientas de desarrollo más importantes para obtener el máximo rendimiento de los sistemas computacionales utilizados destacan am-

pliamente OpenMP y CUDA. En nuestra experiencia trabajando con el primero de los tres problemas, hemos conseguido acelerar la aplicación de manera muy significativa utilizando la GPU programada en CUDA. Con OpenMP hemos conseguido, también en esta aplicación, poner en marcha toda la maquinaria encerrada en un nodo heterogéneo para resolver este problema. Hemos puesto de manifiesto, utilizando este problema en particular como medio, el enorme potencial que tienen este tipo de arquitecturas heterogéneas. Los resultados experimentales obtenidos bajo la luz de este caso de estudio indican que se puede reducir significativamente el tiempo de ejecución mediante una cuidada clasificación de tareas y distribución de la carga de trabajo entre los recursos computacionales.

Evaluación de polinomios matriciales

El uso intensivo de las GPUs disponibles a través de librerías optimizadas para las mismas es capaz de ofrecernos unas prestaciones importantes, significativamente mayores que aquellas que ofrecen los procesadores de propósito general. Tanto es así que no hemos podido utilizar la heterogeneidad del nodo debido a la gran diferencia existente en la capacidad computacional entre las GPUs y los procesadores de propósito general. A través de la librería CUBLAS, concretamente utilizando la rutina de multiplicación de matrices, ofrecemos al usuario una aplicación realmente potente para la evaluación de polinomios matriciales. Queremos destacar en este trabajo el diseño recursivo de nuestro algoritmo, la metodología utilizada para generar código para varias GPUs mediante OpenMP, así como también el amplio abanico de posibilidades que ofrece nuestra aplicación para resolver funciones matriciales que exceden el ámbito de aplicación de este trabajo.

Representación de variables meteorológicas del clima

Este problema es crítico por dos motivos: su alto coste computacional y la necesidad de disponer de los datos meteorológicos en un breve espacio de tiempo. Con un número grande de estaciones logramos un nivel más satisfactorio de detalles pero requerimos más carga computacional. Utilizando una metodología de identificación de tareas y reparto de la carga, que es común a todos los problemas tratados en esta tesis, hemos conseguido una gran eficacia en la aplicación y hemos reducido significativamente el tiempo de respuesta. El impacto que tienen estos resultados en la agricultura de la zona es altamente significativo. Sin embargo, tal como es usual cuando

se mejoran las aplicaciones en paralelo, han quedado al descubierto otras partes del programa que no eran críticas antes de la paralelización pero sí lo son ahora, como es el acceso a la base de datos. También las técnicas de paralelismo contribuirán a la solución del este problema en el futuro.

Modelado de ríos

El modelado hídrico de ríos permite manejar criterios que permitan comparar cuencas de distintas dimensiones, así como relacionar parámetros morfométricos con el funcionamiento hidrológico de los mismos. Al igual que en los casos anteriores, el coste computacional crece con el aumento de la longitud de la sección del río seleccionada para ser modelado. El coste computacional aquí radica en la construcción de la matriz del sistema de ecuaciones. Hemos aprovechado el alto paralelismo intrínseco a la construcción de esta matriz para generar una aplicación altamente eficiente y así resolver el problema de manera óptima para los usuarios del mismo.

Autooptimización de software para arquitecturas multicore y multi-GPU

En este capítulo se comparan dos metodologías de autooptimización para balancear la distribución de trabajo entre CPU y GPU en problemas aplicados en un sistema multicore+multi-GPU. Uno de los modelos se basa en la realización de experimentos de forma guiada para reducir el tiempo de experimentación en la búsqueda de la mejor distribución para un conjunto de tamaños de problema predeterminado. Los mejores valores obtenidos se almacenan para usarlos en tiempo de ejecución para determinar la forma de partición de las tareas entre CPU y GPU. El segundo método se basa en un modelo teórico del tiempo de ejecución de la rutina híbrida. El modelo contiene parámetros que reflejan el comportamiento de la rutina en el sistema, y los valores de estos parámetros se estiman para cada sistema concreto por medio de experimentos. En la ejecución se utiliza un modelo adaptado al sistema para decidir el particionado de los datos. En general, con los dos modelos se obtienen distribuciones muy cercanas a la óptima. Su aplicación a los casos de estudio tratados en la tesis ha sido satisfactoria y ha permitido obtener un buen producto final que es, a la vez, eficiente y fácil de utilizar para usuarios no expertos en paralelismo u otras técnicas de computación de altas prestaciones.

8.2 Publicaciones generadas en el marco de la tesis

A continuación se enumeran todas las publicaciones asociadas a la tesis.

Publicaciones en revistas científicas

◇ M. Boratto, P. Alonso. D. Giménez and M. Barreto, ***Automatic Routine Tuning to Represent Landform Attributes on Multicore and Multi-GPU Systems***, The Journal of Supercomputing, April, Springer US, Volume 70, Issue 2, p. 733–745, 2014.

- **Resumen:** Diversas aproximaciones iniciales a la problemática de la optimización automática en plataformas heterogéneas tomando como ejemplo la representación del relieve.
- **Capítulo de la tesis:** 3 y 7.
- **Referencia:** [102].

◇ J. Peinado, P. Alonso, J. Ibáñez, V. Hernández and M. Boratto, ***Solving time-invariant differential matrix Riccati equations using GPGPU computing***, The Journal of Supercomputing, Springer US, Volume 70, Issue 2, p. 623–636, 2014.

- **Resumen:** Esta publicación esta enmarcada dentro del concepto general de la tesis aunque no trata ninguno de los problemas en particular tratados en la misma. Utiliza una partición de carga de trabajo en una arquitectura CPU–GPU para resolver un problema teórico de ecuaciones de Riccati.
- **Capítulo de la tesis:** 7.
- **Referencia:** [106].

Publicaciones en congresos internacionales

◇ M. Boratto, P. Alonso, C. Ramiro, M. Barreto and L. Coelho, ***Parallel Algorithm for Landform Attributes Representation on Multicore and Multi-GPU Systems***, ICCSA (International Conference on Computational Science and its Applications), Springer, Lecture Notes in Computer Science, Part I, v. 7333, p. 29–43, 2012.

- **Resumen:** Esta publicación esta enmarcada dentro de uno de los problemas centrales de la tesis, modelado y optimización en la representación del relieve. El trabajo propone la primera aproximación de la solución del problema de alto coste computacional sobre una arquitectura heterogénea multicore y multi-GPU.
- **Capítulo de la tesis:** 3.
- **Referencia:** [107].

◇ M. Boratto, P. Alonso and C. Ramiro and M. Barreto, ***Heterogeneous Computational Model for Landform Attributes Representation on Multicore and Multi-GPU Systems***, ICCS (International Conference on Computational Science), Procedia Computer Science, v. 9, p. 47-56, 2012.

- **Resumen:** Esta publicación es la primera propuesta completa, y que complementa a la anterior publicación, abordando la representación del relieve sobre una arquitectura heterogénea multicore y multi-GPU.
- **Capítulo de la tesis:** 3.
- **Referencia:** [108].

◇ M. Boratto, P. Alonso and D. Giménez, ***Computational modelling of meteorological variables***, 13th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE 2013), Cabo de Gata, Almería, España, Junio, 2013.

- **Resumen:** Primeros estudios sobre modelado computacional de la representación de variables meteorológicas del clima.
- **Capítulo de la tesis:** 5.
- **Referencia:** [109].

◇ M. Boratto, P. Alonso and A. M. Vidal, ***A Threaded Divide and Conquer Symmetric Tridiagonal Eigensolver on Multicore Systems***, 2008 International Symposium on Parallel and Distributed Computing (ISPDC '08), IEEE Computer Society. Lecture Notes in Computer Science, p. 464-468, 2008.

- **Resumen:** Esta publicación constituye un trabajo preliminar sobre modelos óptimos de particionado de datos para ser distribuidos sobre arquitecturas multicore. El modelo es aplicado a un problema teórico de resolución de sistemas tridiagonales utilizando un modelo *Divide y Vencerás*. El trabajo está enmarcado dentro del concepto general de la tesis.
- **Capítulos de la tesis:** 7.
- **Referencia:** [110].

◇ M. Boratto, P. Alonso, J. Peinado, and J. J. Ibáñez, ***Evaluation of matrix polynomials on GPUs***, Enviado a The 6th International Workshop on Programming Models and Applications for Multicores and Manycores.

- **Resumen:** Evaluación de polinomios de matrices en varias GPUs.
- **Capítulos de la tesis:** 4.

Informes técnicos

◇ P. Alonso and M. Boratto and J. Peinado and J. Ibáñez, ***On the evaluation of matrix polynomials using several GPGPUs***, Departament of Information Systems and Computation, Universidad Politècnica de València, València, España, Julio, 2014.

- **Resumen:** Evaluación de polinomios de matrices en varias GPUs.
- **Capítulo de la tesis:** 4.
- **Referencia:** [111].

Proyectos de investigación

El trabajo de esta tesis se ha realizado en el marco de varios proyectos de investigación regionales y nacionales:

◇ Proyecto de la Fundación Séneca, Consejería de Cultura y Educación de la Región de Murcia: "Técnicas de optimización de rutinas paralelas y aplicaciones" (02973/PI/05), de 1 de enero de 2006 a 31 de diciembre de 2008.

- ◇ Proyecto CICYT, Ministerio de Educación y Ciencia: "Construcción y optimización automáticas de bibliotecas paralelas de computación científica" (TIN2008-06570-C04-02), de 1 de enero de 2009 a 31 de diciembre de 2011.

- ◇ Proyecto PROMETEO, Generalitat Valenciana, "High Performance Computing Tools for solving Signal Processing Problems on Parallel Architectures", (PROMETEO/2009/013), de 1 de enero de 2009 a 31 de diciembre de 2013.

- ◇ Proyecto REN-CAPAP-H2, Ministerio de Economía y Competitividad: Red de computación de altas prestaciones sobre arquitecturas paralelas heterogéneas (CAPAP-H), (TIN2007-29664-E), de 1 de junio de 2012 a 1 de junio de 2013.

8.3 Trabajos futuros

A la vista de los resultados obtenidos, se proponen algunas líneas de investigación relacionadas y complementarias con el trabajo realizado en esta tesis, que se consideran interesantes a corto y medio plazo y que se podrían agrupar en varias direcciones:

Otros problemas A corto plazo se va a trabajar con otros problemas aplicados: modelado matemático de control biológico de plagas de caña de azúcar, desarrollo de un modelo lagrangiano de transporte de aceite, etc. Se pretende validar la metodología de autooptimización para un rango amplio de esquemas paralelos que pueden ser aplicados de manera inmediata a nuevos problemas.

Explotar en totalidad la heterogeneidad del sistema Nuestra implementación está diseñada para la división y asignación de tareas a diferentes recursos computacionales en multicores con varias GPUs. Pretendemos extender las aplicaciones y las técnicas de autooptimización a clusters con GPUs para acelerar los problemas estudiados.

Nuevos modelos de autooptimización Se ha comprobado en el Capítulo 7 que los modelos obtenidos puede ser mejorables en el sentido de reducir el espacio de búsqueda en el proceso de experimentación. Las

ideas aquí consisten en utilizar heurísticas dependientes del problema y/o la configuración hardware subyacente.

Extensión a nuevas configuraciones hardware También estamos estudiando la aplicación y extensión de estas técnicas de autooptimización a nuevas configuraciones hardware que incluyan otros aceleradores hardware como el Intel Many-Integrated-Core Xeon Phi, u otros tipos de GPUs que no sean fabricados por la NVIDIA, lo que conduciría también a realizar implementaciones en OpenCL.

Bibliografía

[1] L. Namikawa and C. Renschler, "Uncertainty in digital elevation data used for geophysical flow simulation," in *GeoInfo*, 2004, pp. 91–108.

[2] J. Wallace and P. Hobbs, *Atmospheric Science: An Introductory Survey*, ser. Atmospheric Science, International Geophysics Series. Elsevier Science, 2006.

[3] C. Tucci, *Hidrical Models.* UFRGS (Universidade Federal do Rio Grande do Sul), 2005.

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

[5] L. Bastos, R. Abrantes, and B. Leal, "A mathematical formulation of a model for landform attributes representation for application in distributed systems," in *Web Information Systems and Technologies*, Funchal, Madeira, Portugal, 2008, pp. 259–263.

[6] C. Christensen, T. Aina, and D. Stainforth, "The challenge of volunteer computing with lengthy climate model simulations," in *Proceedings of the First International Conference on e-Science and Grid Computing.* Washington, DC, USA: IEEE Computer Society, 2005, pp. 8–15.

[7] D. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K. Schauser, R. Subramonian, and T. V. Eicken, "LogP: A Practical Model of Parallel Computation," *ACM*, vol. 39, no. 11, pp. 78–85, 1996.

[8] J. González, C. Rodríguez, G. Rodríguez, F. de Sande, and M. Printista, "A Tool for Performance Modeling of Parallel Programs," *Scientific Programming*, vol. 11, no. 3, pp. 191–198, 2003.

[9] S. Hammond, G. Mudalige, J. Smith, S. Jarvis, J. Herdman, and A. Vadgama, "WARPP: A toolkit for simulating high-performance parallel scientific codes," *Journal in Applied Mathematics*, pp. 1–19, 2009.

[10] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, and D. Wilcox, "Pace–a toolset for the performance prediction of parallel and distributed systems," *IJHPCA*, vol. 14, no. 3, pp. 228–251, August 2000.

[11] J. Roda, C. Rodríguez, D. González, and F. Almeida, "Predicting the Execution Time of Message Passing Models," *Concurrency - Practice and Experience*, vol. 11, no. 9, pp. 461–477, 1999.

[12] L. Valiant, "A bridging model for parallel computation," *ACM*, vol. 33, no. 8, pp. 103–111, August 1990.

[13] S. Akl, *Parallel Computation: models and methods.* Prentice Hall, 1997.

[14] R. M. Hord, *Understanding Parallel Supercomputing.* IEEE, 1999.

[15] L. Pastor and J. Bosque, "An efficiency and scalability model for heterogeneous clusters," in *Proceedings in Cluster Computing 2001, IEEE International Conference.* IEEE Computer Society, 2001, pp. 427–434.

[16] Y. Yan and Y. Song, "An Effective and Practical Performance Prediction Model for Parallel Computing on Nondedicated Heterogeneous NOW," *Journal of Parallel and Distributed Computing*, vol. 38, no. 1, pp. 63–80, 1996.

[17] O. Beaumont, A. Legrand, and Y. Robert, "Scheduling divisible workloads on heterogeneous platforms," *Parallel Computing*, vol. 29, no. 9, pp. 1121–1152, 2003.

[18] A. Legrand, H. Renard, Y. Robert, and F. Vivien, "Load-balancing iterative computations on heterogeneous clusters with shared communication links," in *PPAM*, 2003, pp. 930–937.

[19] M. Cierniak, M. Zaki, and L. Wei, "Compile-time scheduling algorithms for a heterogeneous network of workstations," *The Computer Journal*, vol. 40, no. 6, pp. 356–372, 1997.

[20] O. Beaumont, A. Legrand, and Y. Robert, "The master-slave paradigm with heterogeneous processors," in *Parallel and Distributed Systems, IEEE Transactions on (Volume:14 , Issue:9)*. IEEE Computer Society, 2001, pp. 419–426.

[21] M. Banikazemi, J. Sampathkumar, S. Prabhu, D. Panda, and P. Sadayappan, "Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations," in *Heterogeneous Computing Workshop*, 1999, pp. 125–130.

[22] O. Beaumont, A. Legrand, and Y. Robert, "Scheduling strategies for mixed data and task parallelism on heterogeneous clusters," *Parallel Processing Letters*, vol. 13, no. 2, pp. 225–244, 2003.

[23] C. Banino, O. Beaumont, A. Legrand, and Y. Robert, "Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms," *IEEE Transactions Parallel and Distributed Systems*, vol. 15, no. 4, pp. 319–330, 2004.

[24] J. Cuenca, D. Giménez, and J. Martínez, "Heuristics for work distribution of a homogeneous parallel dynamic programming scheme on heterogeneous systems," *Parallel Comput.*, vol. 31, no. 7, pp. 711–735, 2005.

[25] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *ISPASS*. IEEE, 2009, pp. 163–174.

[26] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *SIGARCH Computing Architecture News*, vol. 37, no. 3, pp. 152–163, Jun. 2009.

[27] A. Kerr, E. Anger, G. Hendry, and S. Yalamanchili, "Eiger: A framework for the automated synthesis of statistical performance models," *19th International Conference on High Performance Computing*, vol. 0, pp. 1–6, 2012.

[28] S. Baghsorkhi, M. Delahaye, S. Patel, W. Gropp, and W. Hwu, "An adaptive performance modeling tool for GPU architectures," *SIGPLAN Notices*, vol. 45, no. 5, pp. 105–114, January 2010.

[29] L. Carrington, M. Tikir, C. Olschanowsky, M. Laurenzano, J. Peraza, A. Snavely, and S. Poole, "An idiom-finding tool for increasing productivity of accelerators," in *Proceedings of the International Conference on Supercomputing*, ser. ICS'11. New York, NY, USA: ACM, 2011, pp. 202–212.

[30] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," in *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 152–163.

[31] W. Jia, K. Shaw, and M. Martonosi, "Stargazer: Automated regression-based GPU design space exploration," in *ISPASS*. IEEE, 2012, pp. 2–13.

[32] K. Kothapalli, R. Mukherjee, M. Rehman, S. Patidar, P. Narayanan, and K. Srinathan, "A performance prediction model for the CUDA GPGPU platform," in *HiPC*, 2009, pp. 463–472.

[33] E. Brewer, "Portable high performance supercomputing: High level platform dependent optimization," Ph.D. dissertation, Massachusetts Institute of Technology, 1994.

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

[35] BLAS, "Basic Linear Algebra Subprograms," Disponible en: http://www.netlib.org/blas/forum/.

[36] MKL, "Intel Math Kernel Library for Linux. User's Guide," Disponible en: http://developer.intel.com, October 2009.

[37] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, 2nd ed. Philadelphia: SIAM, 1999.

[38] L. 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*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997.

[39] J. Demmel, J. Dongarra, B. Parlett, W. Kahan, M. Gu, D. Bindel, Y. Hida, X. Li, O. 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," Department of Computer Science, University of Tennessee, LAPACK Working Note 181, February 2007.

[40] F. G. Van Zee and R. A. van de Geijn, "BLIS: A framework for rapidly instantiating BLAS functionality," *ACM Transactions on Mathematical Software*, 2013, accepted.

[41] B. Andersen, F. Gustavson, A. Karaivanov, M. Marinova, J. Wasniewski, and P. Yalamov, "LAWRA: Linear algebra with recursive algorithms," in *PARA*, 2000, pp. 38–51.

[42] ATLAS, "Automatically Tuned Linear Algebra Software," Disponible en: http://math-atlas.sourceforge.net/.

[43] R. C. Whaley, A. Petitet, and J. Dongarra, "Automated empirical optimizations of software and the ATLAS project," *Parallel Computing*, vol. 27, no. 1–2, pp. 3–35, 2001.

[44] mpC, "The mpC Parallel Programming Language," Disponible en: http://hcl.ucd.ie/Projects/mpC/.

[45] HeteroMPI, "Heterogeneous MPI," Disponible en: http://hcl.ucd.ie/Projects/HeteroMPI/.

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

[47] R. Reddy, "HMPI: A message-passing library for heterogeneous networks of computers," Ph.D. dissertation, Computer Science Department, University College Dublin, June 2005.

[48] R. Reddy and A. Lastovetsky, "HeteroMPI + ScaLAPACK: Towards a ScaLAPACK (Dense Linear Solvers) on Heterogeneous Networks of Computers," in *Lecture Notes in Computer Science*, vol. 4297. Bangalore, India: Springer, December 2006, pp. 242–253.

[49] Flamingo Auto-Tuning Framework, "Disponible en: http://http://mistymountain.co.uk/flamingo/."

[50] J. Cuenca, "Optimización automática de software paralelo de álgebra lineal," Ph.D. dissertation, Universidad de Murcia, 2004.

[51] D. Valencia, A. Lastovetsky, and A. Plaza, "Design and Implementation of a Parallel Heterogeneous Algorithm for Hyperspectral Image Analysis Using HeteroMPI," in *Parallel and Distributed Computing, 2006. ISPDC '06.* Timisoara, Romania: IEEE Computer Society Press, July 2006, pp. 301–308.

[52] INCO2, "Interdisciplinary Computation and Communication Group (Universidad Politécnica de Valencia)," Disponible en: http://www.inco2.upv.es/.

[53] OpenMP Architecture Review Board, "OpenMP application program interface version 3.0," May 2008, disponible en http://www.openmp.org/mp-documents/spec30.pdf.

[54] Programming Guide NVIDIA CUDA, "Disponible en: http://developer.nvidia.com/guide.html."

[55] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

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

[57] B. Zhang, X. Yang, F. Yang, X. Yang, C. Qin, D. Han, X. Ma, K. Liu, and J. Tian, "The CUBLAS and CULA based GPU acceleration of adaptive finite element framework for bioluminescence tomography," *Optics Express*, vol. 18, pp. 201–214, September 2010.

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

[59] M. Boratto, "Parametrización de esquemas paralelos divide y vencerás," Master's thesis, Universidad Politécnica de Valencia, 2007.

[60] M. Boyer, J. Meng, and K. Kumaran, "Improving GPU performance prediction with data transfer modeling," in *IPDPS Workshops.* IEEE, 2013, pp. 1097–1106.

[61] C. Bajaj, I. Ihm, and J. Warren, "Higher-order interpolation and least-squares approximation using implicit algebraic surfaces," *ACM Transactions on Graphics*, vol. 12, pp. 327–347, October 1993.

[62] J. Rawlings, S. Pantula, and D. Dickey, *Applied Regression Analysis: A Research Tool.* Springer, April 1998.

[63] D. Marr, F. Binns, D. Hill, G. Hinton, D. Koufaty, J. Miller, and M. Upton, "Hyper-threading technology architecture and microarchitecture," *Intel Technology Journal*, vol. 6, no. 1, pp. 1–12, 2002.

[64] M. Rutzinger, B. Hofle, M. Vetter, and N. Pfeifer, *Digital terrain models from airborne laser scanning for the automatic extraction of natural and anthropogenic linear structures.* Elsevier, 2011, pp. 475–488.

[65] M. S. Paterson and L. J. Stockmeyer, "On the number of nonscalar multiplications necessary to evaluate polynomials," *SIAM J. Comput.*, vol. 2, no. 1, pp. 60–66, 1973.

[66] J. I. Munro and M. Paterson, "Optimal algorithms for parallel polynomial evaluation," *J. Comput. Syst. Sci.*, vol. 7, no. 2, pp. 189–198, 1973.

[67] N. J. Higham, *Functions of Matrices: Theory and Computation.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008.

[68] M. Hochbruck, C. Lubich, and H. Selhofer, "Exponential integrators for large systems of differential equations," *The SIAM Journal on Scientific Computing*, vol. 19, no. 5, pp. 1552–1574, Sep. 1998.

[69] S. M. Cox and P. C. Matthews, "Exponential time differencing for stiff systems," *Journal of Computational Physics, Elsevier*, vol. 176, pp. 430–455, 2002.

[70] A.-K. Kassam and L. N. Trefethen, "Fourth-order time-stepping for stiff PDEs," *The SIAM Journal on Scientific Computing*, vol. 26, no. 4, pp. 1214–1233, 2005.

[71] J. Sastre, J. J. Ibáñez, E. Defez, and P. A. Ruiz, "Efficient orthogonal matrix polynomial based method for computing matrix exponential," *Applied Mathematics and Computation*, vol. 217, pp. 6451–6463, 2011.

[72] ——, "Accurate matrix exponential computation to solve coupled differential," *Mathematical and Computer Modelling*, vol. 54, pp. 1835–1840, 2011.

[73] ——, "Efficient scaling-squaring Taylor method for computing matrix exponential," *SIAM J. on Scientific Comput.*, (Accepted with modifications).

[74] E. Defez, J. Sastre, J. Ibáñez, and P. Ruiz, "Computing matrix functions arising in engineering models with orthogonal matrix polynomials," *Mathematical and Computer Modelling*, vol. 57, pp. 1738<96>–1743, 2011.

[75] E. Weisstein, *CRC Concise Encyclopedia of Mathematics, Second Edition.* Taylor & Francis, 2002.

[76] A. Fournier and J. Buchanan, "Chebyshev polynomials for boxing and intersections of parametric curves and surfaces," *Comput. Graph. Forum*, pp. 127–142, 1994.

[77] S. Jerez, "Herramientas de autooptimización para la realización de simulaciones climáticas con el modelo mm5," Master's thesis, Universidad de Murcia, 2011.

[78] Instituto Nacional de Meteorologia Brasileiro, Disponible en: http://www.inmet.gov.br.

[79] A. Augusto, E. Jakob, and A. Young, "O uso de métodos de interpolação espacial de dados nas análises sociodemográficas," *Anais do XV Encontro Nacional da Associação Brasileira de Estudos Populacionais*, 2006.

[80] R. Cecílio and F. Pruski, "Interpolação dos parâmetros da equação de chuvas intensas com uso do inverso de potências da distância," *Revista Brasileira de Engenharia Agricola e Ambiental*, vol. 7, no. 3, pp. 501–504, 2003.

[81] E. C. Wei and J. L. McGuinness, "Reciprocal distance squared method: a computer technique for estimating areal precipitation," Washington, D.C.:Agricultural Research Service, Tech. Rep., 1973.

[82] R. C. F. Amorim, A. Ribeiro, and B. G. Leal, "Avaliação do comportamento de variáveis meteorológicas espacializadas em áreas de relevo ondulado na bacia do Rio Doce," *Engenharia na Agricultura*, vol. 16, no. 1, pp. 19–26, 2008.

[83] R. C. F. Amorim, A. Ribeiro, C. Leite, B. G. Leal, and J. Batista, "Avaliação do desempenho de dois métodos de espacialização da precipitação pluvial para o Estado de Alagoas," *Science Technology Maringá*, vol. 30, no. 1, pp. 87–91, 2008.

[84] J. C. Myers, *Geostatistical error management: quantifying uncertainty for environmental sampling and mapping*, ser. Industrial Engineering Series. New York, Van Nostrand Reinhold, USA: Wiley, 1997.

[85] P. Piovesan, L. Araujo, and C. Dias, "Validação cruzada com correção de autovalores e regressão isotônica nos modelos de efeitos principais aditivos e interação multiplicativa," *Epub 24*, pp. 1018–1023, April 2008.

[86] C. J. Willmott and K. Matsuura, "On the use of dimensioned measures of error to evaluate the performance of spatial interpolators," *International Journal of Geographical Information Science*, vol. 20, no. 1, pp. 89–102, 2006.

[87] J. Fenlason and R. Stallman, "GNU gprof," Free Software Foundation, Tech. Rep., November 1998.

[88] B. Booth, "Using ArcGis 3D Analyst," *ESRI Press*, p. 220, 2001.

[89] M. Raupach and J. Finnigan, "The influence of topography on meteorogical variables and surface-atmosphere interactions," *Journal of Hydrology*, vol. 190, pp. 182–213, 1997.

[90] F. López, "Paralelización del modelo hidrodinámico secuencial COHERENS para sistemas multicore mediante OpenMP," Meeting on Parallel Routines Optimization and Applications, May 2008.

[91] Universidade do Vale do Río São Francisco, Disponible en: http://www.univasf.edu.br.

[92] S. Jerez, J. Montávez, and D. Giménez, "Optimizing the execution of a parallel meteorology simulation code," in *IPDPS*. Los Alamitos, CA, USA: IEEE Computer Society, 2009.

[93] 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*, vol. 10, no. 2, pp. 115–123, 1999.

[94] J. Cuenca, D. Giménez, and J. González, "Modelling the behaviour of linear algebra algorithms with message passing," *Parallel and Distributed Processing, 2001*, pp. 282–289, February 2001.

[95] 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*, vol. 1, no. 1/2/3, pp. 75–84, 2004.

[96] D. Giménez and J. P. Martínez, "Automatic optimization in parallel dynamic programming schemes," in *Proceedings of VECPAR2004*, Valencia, Spain, 2004, pp. 639–649.

[97] 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).

[98] J. 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.

[99] J. Cuenca, D. Giménez, and J. González, "Architecture of an automatically tuned linear algebra library," *Parallel Computing*, vol. 30, no. 2, pp. 187–210, 2004.

[100] 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.* New York, NY, USA: ACM Press, 2005, pp. 393–402.

[101] K. K. T. Katagiri and H. Honda, "RAO-SS: A prototype of run-time auto-tuning facility for sparse direct solvers," in *International Symposium on Parallel Architectures*, June 2005, pp. 1–10.

[102] M. Boratto, P. Alonso, D. Giménez, and M. Barreto, "Automatic routine tuning to represent landform attributes on multicore and multi-GPU systems," *The Journal of Supercomputing*, vol. 70, pp. 733–745, January 2014.

[103] J. Cámara, J. Cuenca, D. Giménez, L. García, and A. Vidal, "Empirical installation of linear algebra shared-memory subroutines for auto-tuning," *Journal of Parallel Programming*, vol. 18, pp. 110–119, 2013.

[104] L. Moreno, "Computación paralela y entornos heterogéneos," Ph.D. dissertation, Universidad de La Laguna, 2004.

[105] G. Bernabé, J. Cuenca, L. García, and D. Giménez, "Tuning basic linear algebra routines for hybrid cpu+gpu platforms," *The International Conference on Computational Science*, June 2014.

[106] J. Peinado, P. Alonso, J. Ibáñez, V. Hernández, and M. Boratto, "Solving time-invariant differential matrix riccati equations using GPGPU computing," *The Journal of Supercomputing*, vol. 70, pp. 1–14, October 2014.

[107] M. Boratto, P. Alonso, C. Ramiro, M. Barreto, and L. Coelho, "Parallel algorithm for landform attributes representation on multicore and multi-GPU systems," in *ICCSA (1)*, 2012, pp. 29–43.

[108] M. Boratto, P. Alonso, C. Ramiro, and M. Barreto, "Heterogeneous computational model for landform attributes representation on multicore and multi-GPU systems," *Procedia CS*, vol. 9, pp. 47–56, 2012.

[109] M. Boratto, P. Alonso, and D. Giménez, "Computational modelling of meteorological variables," in *CMMSE 2013*, Cabo de Gata, Almeria, June 2013.

[110] M. Boratto, P. Alonso, and A. Vidal, "A threaded divide and conquer symmetric tridiagonal eigensolver on multicore systems," *Proceedings of the 2008 International Symposium on Parallel and Distributed Computing*, pp. 464–468, 2008.

[111] P. Alonso, M. Boratto, J. Peinado, and J. Ibáñez, "On the evaluation of matrix polynomials using several GPGPUs," Departament of Information Systems and Computation, Universitat Politècnica de València, Tech. Rep., 2014.

Librería AUTO

A

Para facilitar las experimentaciones de las aplicaciones, se ha decidido crear una librería que permite aplicar la autooptimización a los problemas aplicados que hemos tratado en esta tesis. Esta librería se ha creado empleando como base los modelos desarrollados. Su filosofía es la de ser exportable a cualquier otro problema, y que para un usuario sin ninguna experiencia todo sea lo más transparente posible. En este estudio, se presenta una librería que permite una ejecución cercana a la óptima de forma automática.

Se ha creado una librería llamada AUTO, concebida para ser empleada en el momento de la instalación de los modelos empírico y experimental en un entorno de ejecución paralelo. Con esta librería se extrae información sobre la escalabilidad del código paralelo del modelo empírico junto con las características del entorno heterogéneo de ejecución. Se ha ejecutado la librería en paralelo utilizando varios recursos computacionales, según preferencias del usuario, y con esto se han realizado pruebas cortas dentro de un dominio de distintos tamaños y formas, y para diferentes particiones. Los experimentos han sido realizados a partir de esta librería, de la que se han comprobado todas sus funcionalidades, facilidad de manejo y, sobre todo, utilidad práctica para los problemas aplicados de la tesis. A continuación se muestra un resumen de como instalar y utilizar la librería junto con las explicaciones de las rutinas desarrolladas.

A.1 Rutinas de la Librería AUTO

A continuación comentaremos algunos aspectos generales de las rutinas de autooptimización desarrolladas, teniendo como referencia los casos de estudio: representación del relieve (Capítulo 3), representación de variables meteorológicas del clima (Capítulo 5), y modelado hídrico de ríos (Capítulo 6). De forma general se puede destacar que:

- ◇ Lo que hacen las rutinas desarrolladas es resolver el problema alterando los valores de autooptimización antes y después de la computación.

- ◇ Tienen la opción de realizar la autooptimización tanto de forma empírica cuanto experimental.

AUTO_POTENCIA

PROPÓSITO:
Retorna las potencias absolutas de la CPU y GPU de un sistema heterogéneo.

SINOPSIS:
RUTINA AUTO_POTENCY (IDGPU, *VECPOT(*, 2))

PARÁMETROS:
IDGPU - INTEGER
En la entrada, IDGPU especifica el identificador de la GPU de la que se quiere hallar la potencia absoluta.
VECPOT - DOUBLE PRECISION vector de DIMENSION (*, 2)
En la salida VECTPOT(0) retorna la potencia de la CPU, y VECTPOT(1) la potencia de la GPU.

AUTO_SPEEDUP

PROPÓSITO:
Retorna el speedup de 1 GPU con respecto a 1 núcleo de la CPU.

SINOPSIS:
RUTINA AUTO_SPEEDUP (IDGPU, SPEEDUP)

PARÁMETROS:
IDGPU - INTEGER
En la entrada, IDGPU especifica el identificador de la GPU de la que se quiere hallar el speedup.
SPEEDUP - DOUBLE
Retorna el speedup de la CPU en respecto a la GPU en un valor double.

AUTO_RELIEVE

PROPÓSITO
Calcula la representación del relieve, retornando los parámetros de ejecución de forma auto-optimizada.

SINOPSIS
RUTINA AUTO_RELIEVE (DEGREE, CORES, PROPGPU, NUMGPU, AUTO, *VECDEGREE(*, DEGREE))
PARÁMETROS
DEGREE - INTEGER
En la entrada, especifica el grado del polinomio de la superficie.
CORES - INTEGER
En la entrada representa el número de cores utilizado.
PROPGPU - INTEGER
En la entrada representa la proporción de computación asignada a cada recurso computacional, teniendo como base la GPU.
NUMGPU - INTEGER
Número de GPUs utilizadas en la computación.
AUTO - CHAR*1
Flag que representa la activación de la auto-optimización (1 si experimental), (2 si empírica) y (0 no).
VECDEGREE - DOUBLE PRECISION vector de DIMENSION (*, DEGREE)
En la salida devuelve los coeficientes del polinomio.

AUTO_CLIMA

PROPÓSITO
Calcula la representación de variables meteorológicas del clima, retornando los parámetros de ejecución de forma auto-optimizada.

SINOPSIS
RUTINA AUTO_CLIMA (STATIONS, CORES, PROPGPU, NUMGPU, AUTO, *SOLUTION(*, *))
PARÁMETROS
STATIONS - INTEGER
En la entrada, especifica el número de estaciones.
CORES - INTEGER
En la entrada representa el número de cores utilizado.
PROPGPU - INTEGER
En la entrada representa la proporción de computación asignada a cada recurso computacional, teniendo como base la GPU.
NUMGPU - INTEGER
Número de GPUs utilizadas en la computación.
AUTO - CHAR*1
Flag que representa la activación de la auto-optimización (1 si experimental), (2 si empírica) y (0 no).
SOLUTION - DOUBLE PRECISION vector de DIMENSION (*, *)
En la salida devuelve variables meteorológicas del clima.

AUTO_RIO

PROPÓSITO
Calcula el modelo hídrico de ríos, retornando los parámetros de ejecución de forma auto-optimizada.

SINOPSIS
RUTINA AUTO_RIO (PLAN, CORES, PROPGPU, NUMGPU, AUTO, *MAP(PLAN, PLAN))
PARÁMETROS
PLAN - INTEGER
En la entrada, especifica el recorte del río.
CORES - INTEGER
En la entrada representa el número de cores utilizado.
PROPGPU - INTEGER
En la entrada representa la proporción de computación asignada a cada recurso computacional, teniendo como base la GPU.
NUMGPU - INTEGER
Número de GPUs utilizadas en la computación.
AUTO - CHAR*1
Flag que representa la activación de la auto-optimización (1 si experimental), (2 si empírica) y (0 no).
MAP - DOUBLE PRECISION vector de DIMENSION (PLAN, PLAN)
En la salida devuelve el mapa hídrico del río.