UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ACATLÁN

RENDIMIENTO DE LA TECNOLOGÍA DE HIPER-HILADO EN LAS APLICACIONES NUMÉRICAS EN GEOFÍSICA

TESIS

QUE PARA OBTENER EL TÍTULO DE:

LIC. EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

PRESENTA:

ANA LILIA DE LA CRUZ HERNÁNDEZ

ASESOR : DR. CARLOS COUDER CASTAÑEDA

FECHA: NOVIEMBRE 2015 Naucalpan, Edo. de Mex

A mi familia,
por todo el apoyo que me brindaron a lo largo de mi vida y sobre todo por el empujoncito que me dieron para cerrar este ciclo y nunca dudar de mis capacidades.

A mis amigos,
por estar conmigo en todo momento.

A la UNAM,
por todas las oportunidades que me brindó a lo largo de mi vida estudiantil.

Al Instituto Mexicano del Petróleo,
por haberme dado la oportunidad, el apoyo y las facilidades para poder cumplir con este objetivo.

AGRADECIMIENTOS

A la UNAM gracias por todas los retos y oportunidades que me brindó a lo largo de mi vida estudiantil y todos los grandes momentos que viví dentro de sus aulas.

Al Instituto Mexicano del Petróleo gracias por haberme dado la oportunidad y el apoyo para poder realizar los experminetos requeridos en este trabajo.

Al Dr. Carlos Couder Castañeda, por la orientación, las enseñanzas y dedicación que me brindó.

Al Mtro. Alfonso Ganzález Ibarra, por todas las facilidades y el apoyo que me brindó para poder cumplir con este objetivo.

Y a todos los maestros de la UNAM por su dedicación, sus invaluables consejos y los retos que me pusieron para poder superarme día a día.

RESUMEN

La tecnología de hiper-hilado ha estado presente en los procesadores Intel por poco más de una década, no obstante, al momento de evaluar el rendimiento de las aplicaciones intensivas en cómputo como son los programas numéricos aplicados a la geofísica no es tomada en cuenta aunado a la afinidad la cual consiste en la forma de organizar los hilos dentro de un sistema multi-núcleo.

Resultados previos en blogs y en artículos reportan que el hiper-hilado en general tiene un efecto negativo en el rendimiento, contrastando con los reportes técnicos de Intel que indican lo contrario, no obstante, no indican como son organizados los hilos dentro del sistema, es decir, no mencionan la afinidad.

Sería inverosímil probar un gran rango de aplicaciones, por lo que en este trabajo probamos dos tipos de aplicaciones comúnmente utilizadas en la industria del petróleo, dentro de la geofísica. Aplicada, que es el cálculo directo de la gradiometría de gravedad y la simulación de un flujo supersónico en la tobera de un eyector.

La primera consiste en calcular la solución analítica de las componentes del tensor gravimétrico, utilizando la ecuación del potencial gravitacional para un ensamble prismático de prismas de densidad constante, lo cual requiere un alto costo computacional, debido a que el potencial gravitacional de cada prisma del conjunto deber ser calculado contra todos los puntos de una malla de observación previamente definida, resultando en un programa intensivo en cómputo de punto flotante. La segunda consiste en resolver de manera directa las ecuaciones de Navier-Stokes para simular un flujo supersónico en la tobera de un eyector a través de un método de diferencias finitas.

De los resultados obtenidos vemos que el hiper-hilado puede o no beneficiar el rendimiento de la aplicación, en el caso del cálculo directo de las componentes del tensor gravimétrico el hiper-hilado proporciona un mejor rendimiento debido a que cada hilo de ejecución tiene su propio espacio de direcciones de memoria, en el caso del eyector vemos un decremento del rendimiento debido a que los hilos comparten en mismo espacio de memoria, de lo cual discernimos que el hiper-hilado puede o no mejorar el rendimiento de una aplicación multi-hilada y que depende de como es organizada la memoria y por lo ende la naturaleza de la aplicación.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO Y ACRONIMOS

API Application Programming Interface

GPU Graphics Processing Unit

Hilo Thread en inglés. Es la secuencia más pequeña de instrucciones que puede administrarse de forma independiente por el programador.

Hiper-hilado Permite que un solo procesador físico actúe como dos procesadores lógicos.

HT Hyper Threading

ILP Instruction Level Parallelism

Multi-núcleo Un procesador que combina dos o más microprocesadores independientes en un solo circuito integrado.

SMP Symmetric Multiprocessing Machine

SMT Simultaneous Multithreading

TLP Thread Level Parallelism

ANTECEDENTES

La computadora digital se ha convertido en una poderosa herramienta para el manejo de información y para llevar a cálculos que sería humanamente imposible realizar.

En los inicios de la computación moderna los programas eran diseñados y ejecutados de forma serial lo cual implica leer una instrucción de memoria, decodificarla, obtener los operandos de memoria, ejecutar la operación indicada, llevar el resultado a memoria y así sucesivamente con las siguientes instrucciones.

La computación secuencial funcionó adecuadamente durante décadas hasta que se empezaron a tener limitantes físicas, en primer lugar el límite de la velocidad de la luz, lo cual implica que una señal de transferencia de información no puede superar la velocidad de 300,000 Km/seg, lo que nos da 30 cm/ns, y actualmente la tecnología se esta acercando a su límite. En segundo lugar, la integración de cada vez más componentes (transistores) en menos espacio (miniaturización) está llegando al máximo número de componentes que se pueden integrar por unidad de superficie. Cuanto más próximos estén, mayor dificultad existirá en la disipación del calor que se generan por los componentes.

Actualmente aunque las computadoras desarrollan unas potencias computacionales elevadas comparado con las máquinas de apenas unas décadas, esta se ha incrementado gracias al desarrollo de las tecnologías multi-núcleo. Lo procesadores actuales para la computación de escritorio y de alto rendimiento pueden alcanzar velocidades de ciclo de reloj de 3.6 GHz. Al crecer la frecuencia de reloj también lo hace el consumo de energía, el calor generado y la interferencia electromagnética.

El paradigma de ejecución secuencial se vio transformado cuando se introdujeron distintas unidades de procesamiento para resolver un problema, con lo que dio surgimiento a la programación paralela y por ende a la ejecución en paralelo. El unir dos computadoras a través de una red o el desarrollo de los sistemas multi-procesador/núcleo conllevó a la necesidad de crear nuevos modelos de programación.

El hiper-hilado fue introducido por primera vez por Intel en el año 2002 y se puso a disposición a través del procesador Pentium IV, esta tecnología permite a un CPU actuar como múltiples CPU's, ocupando distintos componentes de un núcleo de procesamiento, por lo cual, un hilo puede utilizar la unidad de punto flotante mientras el otro la unidad de enteros, además de que posibilita conmutar rápida-

mente entre dos hilos de ejecución. Actualmente la tecnología de hiper-hilado se sigue utilizando en toda la gama de Intel desde los procesadores para celulares, NetBooks, máquinas de escritorio hasta los procesadores de gama alta como los Xeon, y se ha visto que proporciona un buen rendimiento en aplicaciones de tipo informático donde el uso de la unidad de punto flotante no es intensivo. Luego lo que se pretende investigar en el presente trabajo, es conocer el rendimiento que nos puede dar esta tecnología en aplicaciones intensivas en computo numérico como son los programas para geofísica.

INTRODUCCIÓN

La hipótesis central de este trabajo se basa en que la tecnología de hiper-hilado es una técnica mediante la cual un solo núcleo de procesamiento efectivo puede manejar dos hilos, es decir, puede atender dos procesos ligeros de manera concurrente, y desde el punto de vista del programador este proceso de administración de los hilos es transparente y en el sistema operativo aparecen como si fuesen dos núcleos aunque físicamente o realmente es uno solo. Esta tecnología tiene como objetivo conmutar de una forma más rápida entre los hilos de ejecución de una aplicación, es decir, el tiempo que le toma al núcleo dejar un hilo para atender otro.

Aunque este aumento en la velocidad de conmutación puede resultar beneficioso para algunas aplicaciones que no son tan demandantes computacionalmente como las aplicaciones ofimáticas, no obstante para otras como las aplicaciones numéricas puede producir una baja de rendimiento computacional si no son creados el doble de hilos con respecto al número de cores reales disponibles en la plataforma, por consiguiente, solo creamos una sobrecarga de trabajo. Por consiguiente, las aplicaciones numéricas intensivas no se benefician del hiper-hilado.

El objetivo principal de este trabajo de tesis es evaluar el rendimiento de los procesadores con tecnología de hiper-hilado en aplicaciones numéricas en geofísica.

Los objetivos particulares son:

- Explorar el rendimiento de un código de diferencias finitas para la simulación de un flujo supersónico.
- Evaluar el rendimiento en problemas de modelación directa en gravimetría de gravedad basado en un ensamble de prismas.

La computación paralela surge como solución a la demanda de procesadores más rápidos, eficientes y capaces de realizar tareas simultáneamente. Para mediados de la década de los 90's la computación secuencial puede considerarse que había llegado a un tope debido a las limitaciones físicas como lo eran los altos costos de seguir produciendo procesadores más potentes con un mayor número de transistores conduciendo a un uso de energía y sobrecalentamiento de estos, pero sobretodo el haber llegado al límite de la velocidad de transmisión de información interna y la velocidad del CPU. Como solución a dichas limitantes los fabricantes deciden que es más eficiente construir procesadores con muchos núcleos en vez de construir procesadores más rápidos (Almeida Francisco, 2008).

Con el surgimiento de los procesadores multi-núcleo se incrementa la complejidad de la programación debido a que tiene que ser paralela y al ser paralela se tienen nuevos retos que en una programación secuencial no existen como el no determinismo de la ejecución y el diseño de la comunicación entre unidades de procesamiento.

El principio básico del paralelismo es efectuar diferentes tareas al mismo tiempo en los distintos núcleos que se diferencia de la computación secuencial en que se ejecuta una tarea a la vez y después de que esta ha terminado se ejecuta la siguiente.

En la computación paralela se hace uso de los diferentes procesadores que se tienen para resolver una tarea. Cada procesador trabaja con una parte de la tarea de manera simultánea a los otros procesadores, posteriormente se lleva a cabo la comunicación de los datos entre los procesadores.

En la programación paralela se debe realizar la descomposición del problema en las partes que se pueden llevar a cabo en paralelo y las que no, las partes que no podemos realizar en paralelo es la fracción serial del problema, la mayor dificultad que se presenta en la programación paralela es que el acceso a los datos se vuelve no determinista y por lo tanto es más complejo.

Con la programación paralela se logró resolver problemas de gran escala que no caben en la memoria de un computador tradicional o bien reducir el tiempo de ejecución como en modelos computacionales tales como el SEISMIC_CPML (propagación de ondas sísmicas), el POM (Princenton Ocean Model), MM5(modelación del clima). Algunos de los campos de investigación beneficiados con esta programación son: el estudio meteorológico, modelado de fenómenos sísmicos, genoma humano, simulación de moléculas, modelado de la biósfera, etc.

La clasificación de los sistemas paralelos es conocida como la taxonomía de Flynn, la cual clasifica en cuatro modelos de acuerdo al flujo de datos y de instrucciones, y se enlistan a continuación:

- El modelo SISD (Single Instruction Single Data), es el comúnmente utilizado en la computación secuencial ya que se tiene un solo flujo de instrucciones de manera consecutiva las cuales van trabajando sobre un único conjunto de datos.
- El modelo SIMD (Single Instruction Multiple Data), consiste en un solo flujo de instrucciones sobre diferentes conjuntos de datos. Este es un modelo paralelo ya que se ejecuta la misma instrucción sobre distintos datos de manera simultánea, un ejemplo, de esta tecnología es la vectorización presente en las nuevas arquitecturas de procesadores.
- En el modelo MISD (Multiple Instruction Single Data), se ejecutan diferentes flujos de instrucciones al mismo tiempo sobre los mismos datos.

- El modelo MIMD (Multiple Instruction Multiple Data), es el esquema que siguen actualmente los sistemas paralelos tipo *cluster*, se tiene diferentes unidades de proceso asociadas cada una a un solo conjunto de datos los cuales están ejecutando un flujo de instrucciones distintas. Se tienen varios núcleos que comparten la memoria y varios hilos asignados a cada núcleo, estos hilos trabajan de manera independiente a pesar de que están ejecutando el mismo código, lo que hace que en un momento dado un hilo vaya por instrucciones distintas del código accediendo a distintas partes de la información a pesar de que comparten la memoria. Este modelo es seguido tanto para el paralelismo de memoria compartida como de memoria distribuída.

El modelo o esquema de programación paralela que se utiliza para desarrollar un programa esta fuertemente influenciado por la arquitectura de la máquina donde se pretende ejecutar. Básicamente, los sistemas los podemos clasificar por la arquitectura de su memoria que puede ser: compartida, distribuida o híbrida (ambas). Dependiendo del modelo de programación que estamos utilizando es la forma en que se van a repartir los datos para que sean procesados en paralelo. El modelo de memoria compartida es el modelo que se estudiará en este trabajo.

El modelo de memoria compartida se basa en que varios procesadores o núcleos tienen acceso al mismo espacio de memoria; la comunicación entre procesos es implícita (es decir, no la controla explícitamente el programador), lo cual la hace muy rápida, y la manipulación de los datos podría requerir sincronización de la información.

Alguno de los sistemas paralelos que podemos ver en la actualidad son:

- Los sistemas multi-núcleo. Es el sistema de cómputo más común actualmente, pues desde los telefonos celulares hasta los servidores de gran escala están basados en varios procesadores multi-núcleo y la memoria es compartida. El esquema de procesamiento está generalmente basado en hilos.
- GPU's. La necesidad de un mayor poder de cómputo condujo a utilizar tarjetas inicialmente destinadas para video juegos para ejecutar programas de propósito general.
- Clusters. Son una combinación de los sistemas anteriores unidos por una red de alta velocidad. Los clusters de gama alta están conectados por redes de un alto ancho de banda como es el Infiband. Esta clase de sistemas son los más comunes y cada nodo, o componente del cluster, consta de procesadores multi-núcleo que puede tener GPU's integrados.

La programación de los clusters es bastante compleja porque requiere de unir distintos paradigmas de programación paralela, como es el paso de mensajes, el manejo de hilos y de programación a bajo nivel de las GPU's. Un ejemplo de

esto, puede ser la unión de MPI (paso de mensajes) con OpenMP (hilos) más una programación a bajo nivel como el meta lenguaje C para CUDA, resultando en una tarea ardua y propensa a errores si no se tiene claro el concepto de paralelismo y concurrencia.

En los procesadores actuales multi-núcleo la mayoría tiene una tecnología de hiper-hilado con la cual, en teoría, se puede aumentar el rendimiento de las aplicaciones al permitir que un solo núcleo de procesamiento se comporte como si fueran dos, al permitir la ejecución concurrente de dos hilos en un núcleo.

El hiper-hilado fue desarrollado para permitir que múltiples hilos puedan competir y compartir todos los recursos de los procesadores como el caché, unidades de ejecución, control lógico, los buses y el sistema de memoria. La tecnología de hiper-hilado esta presente en todos los procesadores de Intel de última generación como el I7, I5, I3 y la familia Xeon que permite que un solo núcleo aparezca como dos núcleos lógicos con estados de arquitectura duplicados, pero compartiendo los recursos físicos. Esto permite que dos hilos de una misma aplicación, o de dos aplicaciones diferentes, puedan ejecutarse en concurrencia, incrementando la utilización de procesador y reduciendo el impacto de la latencia de memoria al traslapar la latencia de un hilo con la ejecución de otro.

El beneficio en el rendimiento que puede proporcionar el hiper-hilado sólo se obtiene si la aplicación es multi-hilada. Los compiladores de Intel C++/Fortran soportan la creación de hilos de manera nativa, o la delegan al compilador a través de directivas, como es el caso del modelo de OpenMP. Desde la perspectiva de la arquitectura significa que las instrucciones desde ambos procesadores lógicos persistirán y se ejecutarán simultáneamente compartiendo recursos de ejecución (Tian et al., 2002).

Comúnmente un programador que no esta interesado en el rendimiento producirá un programa que, en general, sub-utiliza todas las características de un procesador como es el hiper-hilado y la vectorización.

OpenMP es un conjunto de directivas y funciones enfocada a la programación paralela de multi-hilo de memoria compartida, desarrollado principalmente por compañias como Intel, SGI, Oracle y sistemas abiertos como GNU.

Fue establecido como el estándar actual para la programación de memoria compartida para los fabricantes de software. Esta implementado en los sistemas multi-núcleo y computadoras de altas prestaciones con memoria compartida. Actualmente la tendencia es utilizar una programación híbrida, es decir, utilizar OpenMP en conjunto con MPI para aumentar la escalabilidad y mejorar el balance del trabajo.

En OpenMP existen diferentes directivas que permiten al compilador generar las instrucciones que paralelizan el código. Se puede definir el tipo de variables que

existirán en cada región paralela, si es compartida o privada, crear hilos de forma dinámica y definir la forma en que se dividirá el trabajo para cada hilo, así como establecer el punto de sincronización para los datos. El compilador es el que realmente lleva a cabo la paralelización del código, esto permite que códigos secuenciales puedan ser paralelizados, en teoría, sin mucho esfuerzo. Las directivas de `OpenMP` siguen los estándares para las directivas de compilación en `C/C++`, y se distingue entre mayúsculas y minúsculas.

`OpenMP` esta basado en el modelo Fork-Join, donde un hilo maestro entra a la parte paralelizable del código y genera hilos esclavos, realizando el fork, una vez que los hilos terminan la región paralelizable se destruyen e incorporan la información al hilo maestro el cual continúa con el código secuencial, haciendo el join.

`OpenMP` permite crear hilos con sus directivas creando un hilo maestro a quien pone en marcha al resto de los hilos esclavos. Existe una barrera implícita al final de cada región paralela donde el hilo maestro espera a que el resto de hilos esclavos terminen sus procesos y sea el hilo maestro quien continúe con la ejecución del código.

Para poder realizar la programación paralela los datos no deben depender del cálculo de otras iteraciones y evitar así las condiciones de carrera. Esto se debe a que, como no conocemos el orden en que se ejecutará cada instrucción, no existiría congruencia en los datos debido al no determinismo.

El paralelismo anidado está permitido en `OpenMP`, y permite crear una región paralela dentro de otra, lo cual provoca que los hilos esclavos se conviertan en hilos maestros de esa nueva región.

Finalmente su gran ventaja es su portabilidad, ya que todos los fabricantes siguen un estándar y no existen variaciones en la descripción de `OpenMP` entre fabricantes de compiladores. Actualmente se ha avanzando en el desarrollo de versiones para sistemas GPU dando como resultado `OpenACC`.

1

ARQUITECTURA DE LAS MÁQUINAS DE PROCESAMIENTO SIMÉTRICO Y EL HIPER-HILADO

En este capítulo se introduce la arquitectura más común que se utilizan para realizar programación basada en hilos, por lo tanto se aborda el diseño de las máquinas de procesamiento simétrico o SMP (Symmetric Multiprocessing Machine) por sus siglas en inglés.

1.1 PROCESADORES MULTI-NÚCLEO

Varios procesadores están implementados sobre un solo chip, lo que conocemos como tecnología multicore o multinúcleo. En una arquitectura tipo multicore cada procesador contiene dos o más núcleos que pueden ejecutar instrucciones de forma simultánea. El sistema operativo percibe cada núcleo como un procesador independiente con todos sus recursos asociados. Este tipo de arquitectura se comporta como un multiprocesador implementado en un solo chip.

Las ventajas de este tipo de diseño son varias. Desde el punto de vista físico se produce un menor consumo de energía y una mejor disipación de calor, y desde el punto de vista lógico es posible aprovechar mejor un paralelismo basado en hilos, que pueden compartir niveles internos de memoria caché, con tiempos de acceso a los datos menores que en el caso de procesadores que tuvieran que acceder a una memoria externa. La mayoría de los fabricantes está apostando por esta tecnología para el desarrollo de sus próximos productos; las previsiones incluyen decenas e incluso centenares de núcleos en un chip, con miles de hilos en ejecución como es la arquitectura Xeon Phi (Chrysos, 2014).

Parece indicar que en un futuro inmediato las computadoras paralelas van a estar orientados hacia la tecnología multi-núcleo, tanto con una estructura homogénea de procesadores como con procesadores heterogéneos.

1.2 MÁQUINAS DE PROCESAMIENTO SIMÉTRICO

En computación, un Sistema de Multiprocesamiento Simétrico (SMP), es una arquitectura que contiene mas de un procesador mono-núcleo o multi-núcleo inte-

grado en la misma placa base, es decir, dos o más procesadores (en múltiplos de dos) idénticos están conectados a una sola memoria central compartida y son controlados por una única instancia del sistema operativo.

Actualmente los SMP, son comunes y económicos de adquirir, y son una forma de sobre pasar el número de núcleos físicos que otorga un solo procesador. El uso de procesadores multi-núcleo actualmente es muy común y los encontramos tanto en dispositivos móviles (celulares, tablets) como en servidores de alto desempeño, no obstante, un procesador multi-núcleo de alto desempeño tiene una limitante actual de 15 núcleos (Xeon E7-8890 v2), y si se requiere de más núcleos de procesamiento es posible integrar hasta 8 procesadores en una misma placa base dando un total de un sistema con 120 núcleos de procesamiento.

Los sistemas SMP permiten que cualquier procesador trabaje en cualquier tarea, sin importar en que partede la memoria se alojen sus datos, a condición de que cada tarea en el sistema no está en ejecución en dos o más procesadores al mismo tiempo. Con el apoyo adecuado del sistema operativo, sistemas SMP pueden mover fácilmente las tareas entre los procesadores para equilibrar la carga de trabajo de manera eficiente.

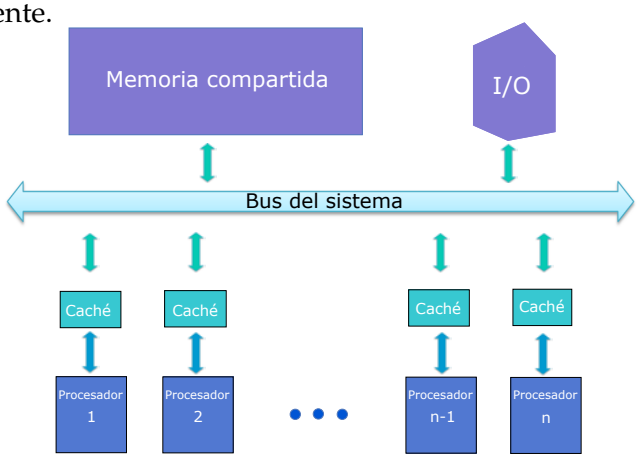

Figura 1.1: Arquitectura convencional de un sistema de Multiprocesamiento Simétrico. Hoy en día la mayoría de los sistemas de alto rendimiento como workstations y servidores usan una arquitectura SMP.

Entre las desventajas que se le pueden encontrar a un sistema SMP, es que el tiempo de acceso a una posición de memoria puede ser diferente en función del módulo en que se encuentre, una red de interconexión interna une a los distintos procesadores con los módulos de la memoria compartida y la comunicación entre los procesadores se consigue utilizando este espacio de direcciones común. Cada uno de los procesadores puede ser un procesador mono-núcleo o multi-núcleo. De esta forma, se puede encontrar dos niveles de paralelismo.

La comunicación entre procesos que se ejecutan en un SMP se realiza a través de la lectura y escritura en memoria por lo que puede existir un tráfico muy grande de datos entre los procesadores y la memoria, al tratar de acceder simultáneamente a las mismas posiciones de memoria generando cuellos de botella que limitan el rendimiento del sistema.

Cada vez disponemos de unidades capaces de procesar datos más rápidamente. Sin embargo para obtener el rendimiento adecuado, es necesario suministrarles datos a la velocidad suficiente, el cual es un problema crucial. En las máquinas actuales la memoria está organizada de forma jerárquica con el objetivo de aprovechar al máximo la localidad de las referencias.

Los programas referencian con mayor probabilidad aquellas posiciones de memoria que están próximas a posiciones de memoria ya referenciadas. Se denomina principio de las referencias. La jerarquía de memorias trata tener cerca del procesador la información que espera utilizarse de forma inmediata. Memorias intermedias: registros, memorias caché y memoria central. Los registros son memorias de acceso muy rápido, internas al procesador, pero con muy poca capacidad.

La memoria caché es una memoria pequeña y rápida que almacena el contenido que se está utilizando de una memoria grande y lenta. Este tipo de memoria es, aproximadamente, entre cinco y diez veces más rápidas que la memoria principal, por lo que permite una reducción substancial del tiempo de acceso a memoria. La memoria caché está organizada por bloques o tramas, y el intercambio de información con memoria central se hace también por tramas completas. El hardware es el encargado de ubicar los bloques de memoria (emplazamiento), cómo reemplazar una trama cuando se produce un fallo de lectura o de escritura (reemplazo), cuándo actualizar la memoria principal en caso de acierto o de fallo de escritura, etc.

De este modo el problema clave de los multiprocesadores de memoria compartida está proporcionando una visión constante de la memoria con varias jerarquías de caché. Este problema de coherencia en el caché es una corrección crítica y el rendimiento sensible del punto de diseño para apoyar el modelo de memoria compartida. Los mecanismo de coherencia de memoria caché no sólo rigen la comunicación en un multiprocesador de memoria compartida, también suelen determinar como el sistema de memoria transfiere los datos entre los procesadores, el caché y la memoria. Asumiendo que el modelo de programación de memoria compartida sigue siendo prominente, las cargas de trabajo futuras dependerán del rendimiento de la coherencia del sistema de memoria y la continua innovación en este ámbito es fundamental para alcanzar en el diseño del equipo.

La coherencia de caché ha recibido mucha atención en la comunidad de investigación, pero la prioridad de trabajo dirigido en las anteriores máquinas multiprocesador (MPs) estaban compuestas por varios procesadores de un solo núcleo.

Quizás la diferencia más importante en el diseño de CMP, en comparación con las prioridades anteriores MPs, es la oportunidad de tomar un enfoque global para diseñar. Las máquinas anteriores generalmente se construyeron de monoprocesadores de productos básicos donde el enfoque de diseño fue en el rendimiento de un solo núcleo. La coherencia del sistema de memoria caché es ahora un problema de diseño de primer orden a nivel de chip.

1.3 EL HIPER-HILADO

La tecnología hiper-hilado trae el concepto de multi-hilado simultáneo (SMT) para la arquitectura Intel. La tecnología de hiper-hilado hace que un solo procesador físico aparezca como dos procesadores lógicos; los recursos de ejecución físicos son compartidos y el estado de la arquitectura se duplica para los dos procesadores lógicos. Desde una perspectiva de software o de arquitectura, esto significa que los sistemas operativos y programas de usuario puede programar hilos para CPU lógicos como lo harían en múltiples CPUs físicos. Desde una perspectiva de la arquitectura, esto significa que las instrucciones de ambos procesadores lógicos persistirán y ejecutaran simultáneamente recursos de ejecución compartidos (Tian et al., 2003), (Zhang et al., 2004).

Cada procesador lógico contiene un conjunto completo del estado de arquitectura. El estado de arquitectura consta de registros, incluyendo el grupo de registros de propósito general, los registros de control, el controlador de interrupción programable (APIC), registros avanzados y algunos registros de estado de la máquina. Desde una perspectiva de software, una vez que se duplica la arquitectura de estado, el procesador aparece como dos procesadores. El número de transistores requeridos para almacenar el estado de la arquitectura es una fracción muy pequeña del total (Tian et al., 2003). Los procesadores lógicos comparten casi todos los otros recursos en el procesador físico, como cachés, unidades de ejecución, predictores de tramas, lógica de control, y los buses. Cada procesador lógico tiene su propio controlador de interrupciones o APIC. Interrupciones enviados a un procesador lógico específico son manejados solamente por ese procesador lógico.

Con la tecnología hiper-hilado, un solo núcleo puede emular dos núcleos lógicos y de esta manera procesar dos hilos simultáneamente. Esta tecnología puede ser una ventaja ya que mientras un hilo se encuentra en estado de espera el otro puede ser atendido llevando al máximo la utilización del núcleo, por lo tanto, la tecnología de hiper-hilado puede mejorar el rendimiento de los programas multi-hilado explotando de una manera más eficiente el uso de los recursos del procesador.

En la actualidad los sistemas de un solo procesador usan paralelismo a nivel instrucción (ILP Instruction Level Parallelism) al ejecutar múltiples instrucciones simultáneamente en diferentes partes del hardware en la ejecución de tuberías.

Hoy en día los sistemas de multiprocesadores en memoria compartida usan ILP, sin embargo, también pueden implementar paralelismo a nivel hilo (TLP Thread Level Parallelism). TLP permite la ejecución en paralelismo no solo de las instrucciones, pero de cada hilo, lo cual ayuda a las aplicaciones multihilo a correr substancialmente más rápido.

Por otra parte, aumentar sustancialmente el rendimiento puede traer mejoras a multitareas, multihilado, y clusters, usando la tecnología de Intel hiper-hilado y `OpenMP` en la interfaz de programación de la aplicación (API Application Programming Interface).

1.3.1 *Habilitando la tecnología de hiper-hilado con TLP*

La tecnología de hiper-hilado está diseñada para saltar la barrera entre sistemas de un solo procesador y múltiples procesadores con la habilitación de TLP en los sistemas de un solo procesador.

La tecnología de hiper-hilado permite que un solo procesador aparezca como dos procesadores (lógicos) ante el sistema operativo y la aplicación asociada.

Cada procesador con hiper-hilado habilitado contiene dos estados de arquitectura que comparte con un sistema de recursos del procesador en ejecución.

El HyperTransport es una tecnología de alta velocidad, baja latencia y conexiones punto a punto diseñada para aumentar la velocidad entre las comunicaciones entre circuitos integrados en los ordenadores, servidores, sistemas integrados, equipos de redes y equipos de telecomunicaciones hasta 48 veces más rápido que algunas tecnologías existentes.

La tecnología de HyperTransport ayuda a reducir el número de buses en un sistema, lo cual puede reducir los cuellos de botella de un sistema y permitir que los microprocesadores más rápidos de hoy en día utilicen la memoria de manera más eficientemente en sistemas multiprocesadores más sofisticados (Mehis, 2002).

La tecnología HyperTransport está diseñada para:

- Brindar un ancho de banda significativamente mayor que las tecnologías actuales
- Utilizar respuestas de baja latencia y bajo recuento de pines.

- Mantener la compatibilidad con buses heredados de PC, permitiendo al mismo tiempo una extensión para nuevos buses SNA (Systems Network Architecture)
- Aparecer de manera transparente para los sistemas operativos y ofrecer un bajo impacto sobre los drivers de los periféricos

La tecnología HyperTransport fue inventada en AMD con contribuciones de socios de la industria y es administrada y licenciada por la HyperTransport Technology Consortium, una corporación sin fines de lucro de Texas, USA.

1.4 COMPARATIVA DE LOS SISTEMAS SMP CONTRA LOS ACTUALES GPU'S

1.4.1 *Arquitectura GPU*

Un GPU solo sabe hacer una cosa, ofrecernos un color para cada pixel de nuestra pantalla utilizando unidades aritméticas programables conocidas como pixel shaders. Estos pixel shaders utilizan una posición (x, y) de la pantalla además de alguna información adicional que combina varias entradas utilizadas para computar el resultado del color final.

Esta información puede consistir de entradas de color, coordenadas de texturas o cualquier otro atributo que pueda ser pasado al shader cuando se ejecute. Resulta que esas entradas de datos no tienen por qué ser *color* y pueden ser cualquier tipo de datos. El único problema es que es necesario utilizar una librería gráfica como OpenGL, GLSL o DirectX HLSL para poder introducirlos.

Lo que hizo NVIDIA (aparte de facilitar la realización de operaciones de punto flotante) fue utilizar el lenguaje estándar C y añadirle un pequeño conjunto de rutinas con el propósito de aprovechar algunas características típicas de la arquitectura CUDA.

A los pocos meses del lanzamiento de la GeForce 8800 GTX, NVIDIA hizo pública la primera versión de un compilador para el lenguaje CUDA C. Así, CUDA C se convirtió en el primer lenguaje específicamente diseñado por una empresa de desarrollo de gráficas para facilitar la computación de carácter general en sus chips.

NVIDIA también porporciona de los controladores necesarios para explotar el potencial de la arquitectura de computación masiva CUDA. Los programadores ya no necesitan saber OpenGL o DirectX para poder hacer uso de ella.

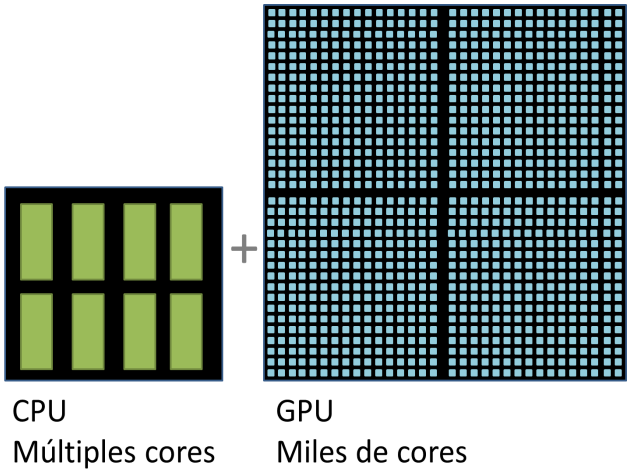

Figura 1.2: Número de núcleos contenidos en un CPU multi-núcleo vs el número de núcleos contenidos en un GPU.

1.4.2 *Diferencias entre el CPU y el GPU*

Una diferencia importante entre un CPU y un GPU es la forma en que manejan el procesamiento de tareas. Un CPU está compuesto de varios núcleos optimizados para el procesamiento en serie, mientras que un GPU está formado de miles de núcleos más pequeños y eficientes los cuales están diseñados para realizar múltiples tareas simultáneamente.

Los GPUs poseen miles de núcleos que procesan las cargas de trabajo de forma paralela y muy eficiente (ver Figura 1.2). No obstante, los núcleos de un GPU solo pueden llevar a cabo operaciones aritméticas y algunas operaciones de control, por lo que no pueden considerarse núcleos completos de procesamiento como los de un CPU convencional.

2

PARADIGMA DE LA COMPUTACIÓN PARALELA MULTIPROCESO Y MULTI-HILO

En este capítulo se aborda las metodologías de programación multi-hilo más comunes resaltando la facilidad de uso de `OpenMP` sobre el uso de otras.

2.1 MODELO DE EJECUCIÓN DE UN PROGRAMA PARALELO EN MEMORIA COMPARTIDA

Un proceso se define como una instancia de programa en ejecución y consiste en un único flujo de instrucciones; cada instrucción del código se ejecuta por turnos y exactamente una a la vez. Para poder obtener múltiples secuencias de instrucciones, bajo un esquema de procesos se pueden utilizar las llamadas al sistema `fork()` y `exec()` para crear varios procesos, cada uno con su propio flujo de ejecución.

En el sistema operativo cada proceso contiene información sobre los recursos del programa y de su estado, esto implica:

- Un espacio de direcciones virtuales.
- Recursos de direcciones virtuales.
- Instrucciones de programa.
- Herramientas de comunicación inter-proceso.

Estos recursos son privados a cada proceso. Para obtener cooperación interprocesos es necesario crear canales externos al proceso mediante los mecanismos que el sistema proporciona para conseguir comunicación entre procesos (IPC) lo que implica llevar a cabo una programación a bajo nivel.

Actualmente los procesos se pueden componer de un conjunto de hilos de ejecución, un hilo de ejecución es una forma ligera de implementar múltiples caminos de ejecución dentro de un proceso. En el modelo de programación multihilada, un proceso puede crear hilos de ejecución adicionales al hilo central, de tal manera que los nuevos hilos utilizan y conviven con los recursos del proceso que los ha creado, además es posible que el sistema operativo los planifique y ejecute como

entidades independientes (núcleos), en gran medida porque duplican únicamente la parte de los recursos que les permiten existir como código ejecutable.

Un hilo de ejecución es un control de flujo independiente por lo tanto mantiene las siguientes características de manera particular:

- Conjunto de registros,
- Pila para variables locales y direcciones de devolución del control,
- Mecanismos de planificación,
- Conjuntos de señales pendientes o bloqueadas.
- Datos específicos del hilo, como el identificador de hilo.

En cuanto al ciclo de ejecución, un hilo:

- Existe en un proceso y utiliza los recursos del proceso.
- Tiene su propio control fijo, mientras exista su proceso padre y el sistema operativo lo soporte.
- Duplica solo los recursos esenciales que necesita para poder planificarse de forma independiente.
- Puede compartir los recursos del proceso con otros hilos que actúan de igual forma independiente o dependiente.
- Es destruido si el proceso padre se destruye.
- Es ligero porque la mayoría de la sobrecarga se ha generado con la creación de su propio proceso.

Los hilos del mismo proceso comparten:

- Las instrucciones del proceso.
- La mayoría de los datos.
- Los descriptores de los ficheros abiertos.
- Las señales y gestores de las señales.
- El directorio de trabajo actual.
- ID de usuario y grupo
- Debido a que los hilo creados por el mismo proceso comparten recursos, se cumple:
- Los cambios hechos por un hilo sobre el sistema de recursos compartido pueden ser vistos por el resto de los hilos.

- Dos punteros que tienen el mismo valor apuntan al mismo dato.
- El posible leer y escribir en la misma posición de memoria y requiere sincronización explícita por parte del programador.

Un hilo normalmente comparte su memoria con otros hilos, mientras que para los procesos habitualmente cuentan con diferentes áreas de memoria para cada uno de ellos. La ventaja al utilizar hilos en vez de procesos es que el cambio de contexto entre hilos es mucho más rápido que el cambio de contexto entre procesos, además la comunicación entre dos hilos es más sencilla y rápida que la comunicación entre procesos.

El modelo de programación multihilo proporciona a los desarrolladores una abstracción bastante útil para la ejecución concurrente. Cuando el sistema es de tipo SMP, los hilos se ejecutan independientemente en un auténtico paralelismo. El programador es el encargado de evitar condiciones de carrera y otros comportamientos no intuitivos y no deseados. Para obtener una correcta manipulación de los datos, los hilos necesitan sincronizarse cada cierto tiempo y así procesar los datos en el orden correcto. Los hilos también pueden necesitar llevar a cabo operaciones atómicas (uso de semáforos) con el fin de prevenir que los datos sean modificados de forma simultánea, o leídos por un hilo al mismo tiempo que está siendo modificado por otro hilo.

La forma más habitual de comunicación entre los hilos creados por el mismo proceso es a través de la modificación del contenido de localizaciones de memoria que se encuentran en la zona compartida por todos los hilos. Este estilo de programación requiere la aplicación de algún tipo de mecanismo de acceso para establecer la coordinación entre hilos. Uno de los mayores problemas en la programación en memoria compartida es el prevenir la interferencia de los hilos paralelos.

En la ejecución simultánea de dos hilos no debemos suponer un orden de ejecución de las instrucciones, si dos hilos se ejecutan simultáneamente sin ningún mecanismo de control de acceso, o de sincronización, el resultado de nuestro programa podría ser erróneo. Nos podemos encontrar con dos problemas: el primero es la naturaleza no determinista del resultado, y el segundo es algún valor inconsistente.

Es importante que si vamos a realizar una operación se realicen de forma atómica. Una condición de competencia por los recursos es habitual que aparezca durante la ejecución de varios hilos de forma simultánea. Este tipo de condiciones pueden surgir cuando múltiples hilos leen y escriben en la misma zona de memoria y por lo tanto requieren una sincronización adecuada, si la sincronización no es adecuada se pueden producir valores incorrectos. La consecuencia de una condición de competencia que no es bien tratada produce un comportamiento no

determinista del programa y deben prevenirse serializando el acceso a memoria. Cuando no se trata de una operación que debe realizarse de forma atómica, sino que es un conjunto de instrucciones el que debe protegerse de la acción de otros hilos por medio de una sección crítica.

Durante la ejecución es frecuente encontrar problemas de seguridad en el uso de librerías, ya que al querer usar alguna rutina, que accede o modifica una estructura o una dirección en la memoria compartida por los hilos, si dos hilos invocan simultáneamente a esta rutina es posible que quieran modificar la estructura o dirección de memoria al mismo tiempo. Si la rutina no emplea algún mecanismo de sincronización para prevenir que los datos se corrompan, entonces la rutina no es segura y el programador debe implementar el mecanismo de control adecuado si quiere hacer uso de ella.

2.2 HILOS POSIX

En las arquitecturas multiprocesador de memoria compartida, los hilos se pueden utilizar para implementar el paralelismo.

Históricamente, los fabricantes de hardware han implementado sus propias versiones de hilos. Estas implementaciones difieren sustancialmente entre sí, por lo que es difícil para los programadores desarrollar aplicaciones con hilos portables.

Con el fin de aprovechar al máximo las capacidades proporcionadas por los hilos, fue necesario un estándar para la interfaz de programación. Para los sistemas UNIX, esta interfaz se ha especificado por la norma IEEE POSIX 1003.1c (1995).

Las implementaciones que se adhieren a esta norma se conocen como los hilos POSIX, o Pthreads. La mayoría de los proveedores de hardware ofrecen ahora Pthreads además de su API de propietario. El estándar POSIX ha seguido evolucionando y someterse a revisiones, incluyendo la especificación Pthreads.

Pthreads se definen como un conjunto de tipos de programación del lenguaje C y las llamadas a procedimiento, ejecutado con un archivo encabezazo incluido pthread.h y una biblioteca hilo, aunque esta biblioteca puede ser parte de otra biblioteca, como libc, en algunas implementaciones.

Las subrutinas que comprenden el API Pthreads pueden ser informalmente agrupadas en cuatro grupos principales:

- Administración de hilos: Las rutinas que trabajan directamente en los hilos - creando, separando, uniéndose, etc. También incluyen funciones para establecer/consultar atributos de hilos (acopables, programables, etc.)

- Mutexes: Rutinas que se ocupan de la sincronización, llamados "mutex", que es una abreviatura de exclusión mutua. Las funciones mutex brindan para crear, destruir, bloquear y desbloquear exclusiones mutuas. Estos se complementan con funciones de atributos mutex que establecen o modifican los atributos asociados con exclusiones mutuas.
- Variables de estado: Rutinas que abordan la comunicación entre hilos que comparten un mutex. Basado en las condiciones especificadas del programador. Este grupo incluye las funciones para crear, destruir, esperar y señales basadas en los valores de variables especificadas. Funciones para ajustar/-consultar atributos de variable de estado también se incluyen.
- Sincronización: Rutinas que gestionan la lectura/escritura de bloqueos y barreras.

La API de Pthreads contiene alrededor de 100 subrutinas.

2.3 OPENCL

OpenCL (Open Computing Language) es una API que permite crear aplicaciones con paralelismo las cuales permiten ejecutarse en plataformas heterogéneas como el CPU, GPU, entre otros procesadores. El lenguaje está basado en C99. Con el uso de la API de OpenCL, los desarrolladores pueden lanzar kernels escritos de computación usando un subconjunto limitado del lenguaje de programación C en una GPU.

Apple creó la especificación original la cual fue desarrollada en conjunto con AMD, IBM, INTEL y NVIDIA. OpenCL fue creado originalmente por Apple, quien le propuso al Grupo Khronos convertirlo en un estándar libre y abierto, para que no dependiese de un hardware de un determinado fabricante, a diferencia de CUDA que sólo está disponible para tarjetas gráficas NVIDIA y Stream para tarjetas ATI. El 16 de junio de 2008 Khronos creó el Compute Working Group para llevar a cabo el proceso de estandarización. En 2013 se publicó la versión estándar 2.0.

OpenCL es la alternativa libre a las tecnologías que intentan aprovechar la potencia de los procesadores gráficos para realizar operaciones intensas repartidas entre el procesador del equipo CPU y el GPU de cualquier tarjeta gráfica compatible.

Que sea abierto y libre permite llevar OpenCL a un entorno multiplaforma, permitiendo ser aprovechado sobre cualquier plataforma y sistema operativo. Esto es importante ya que evitan esfuerzos a los programadores para adaptar las aplicaciones y que aprovechen las ventajas de OpenCL, haciendo que no dependan del

hardware o sistema operativo que tenga la máquina del cliente. Todo esto permite que los desarrolladores cuenten con una opción multiplataforma que permita un aumento de rendimiento de las aplicaciones compatibles con OpenCL.

OpenCL define una jerarquía de memoria de cuatro niveles para el dispositivo de cómputo (ver Figura 2.3):

- Memoria global: compartido por todos los dispositivos de cómputo, pero tiene una latencia de acceso de alta;
- Memoria de sólo lectura: más pequeño, de baja latencia, permisos de escritura para la CPU del host, pero no los dispositivos de cómputo;
- Memoria local: compartido por múltiples elementos de procesamiento dentro de un dispositivo;
- La memoria privada por elemento, (registros).

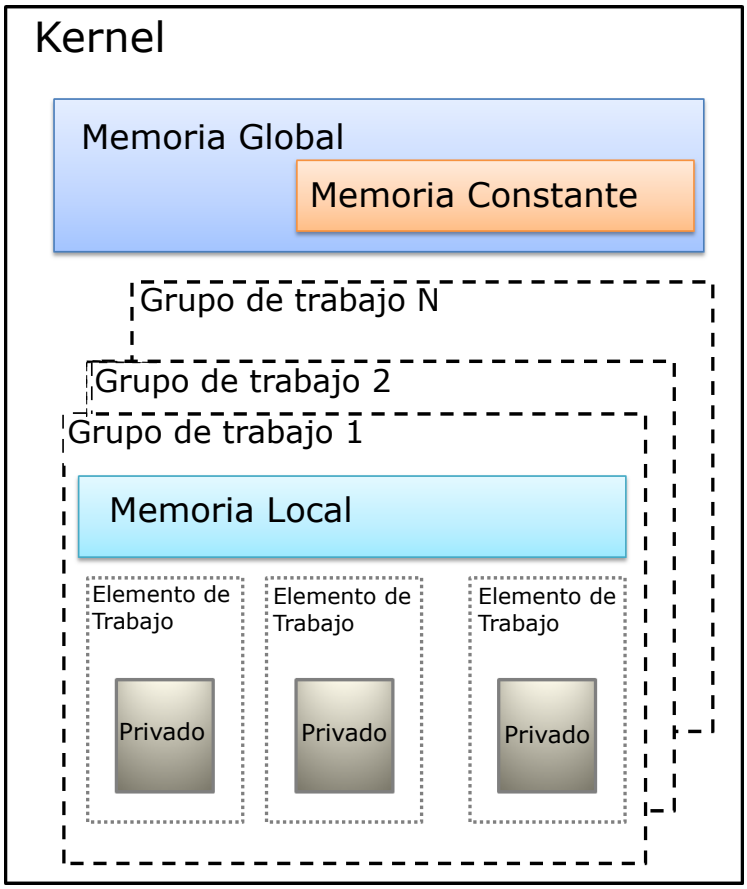

Figura 2.3: Modelo de memoria OpenCL

2.4 INTEL CILK PLUS

Es una extensión de los lenguajes C y C++ que soporta el parlelismo de datos y tareas. A diferencia de otros paquetes de hilado, Intel Cilk Plus no es solo una biblioteca. Es una extensión del lenguaje que se implementa con el compilador y el tiempo de ejecución de Intel Cilk Plus, que permite gastos generales más bajos que las soluciones de la biblioteca solamente. Ofrece una forma rápida, fácil y confiable de mejorar el rendimiento de ambos procesamientos multinúcleo y vectorial. El lenguaje de programación Cilk surgió a partir de tres proyectos separados en el Laboratorio de Ciencias de la Computación del Instituto Tecnológico de Massachusetts para:

- El trabajo teórico en aplicaciones multihilo.
- StarTech - un programa de ajedrez en paralelo construido para funcionar en el Thinking Machines Corporation's Connection Machine modelo CM-5
- PCM/Threaded-C - un paquete basado en C para la programación del estilo de paso continuo en la CM-5

En abril de 1994 los tres proyectos fueron combinados y bautizados `Cilk`. El nombre `Cilk` no es un acrónimo, sino una alusión a los *buenos hilos de seda* (silk) y el lenguaje de programación C.

En 2009, Intel adquirió `Cilk` y se fusionó con la notación de matriz para proporcionar una extensión de lenguaje comprensivo para la implementación de paralelismo tanto en tareas y vectores. `Intel Cilk Plus` fue lanzado por Intel en 2010 como parte del compilador `Intel C++ Composer XE`. Las características clave incluyen:

- Soporta tanto C y C++.
- Compatible con depuradores estándar.
- Utiliza convenciones de llamada estándar, marcos de función `Cilk` son asignadas en la pila de modo que las funciones C/C++ se pueden llamar funciones `Cilk` libremente

Intel ha hecho que las especificaciones de `Intel Cilk Plus` estén disponibles libremente en la web. En 2011, Intel anunció que estaba implementando `Intel Cilk Plus` en la rama de `GCC Cilk plus`. La implementación inicial fue completada en 2012 y fue presentada en la conferencia de Herramientas Cualdron del CCG 2012. Intel también ha propuesto `Intel Cilk Plus` como un estándar para el cuerpo estándar de C++ (ver Figura 2.4).

Intel Cilk Plus permite:

- Escribir programas paralelos usando un modelo simple, con solo 3 palabras clave para aprender, los desarrolladores de C y C++ pueden moverse rápidamente en el dominio de la programación paralela.
- Identificar el paralelismo de datos mediante el uso de notaciones de matriz simples que incluyen funciones elementales.
- Aprovechar las herramientas de serie ya existentes, la semántica de serie de `Intel Cilk Plus` permite el uso de un depurador estándar.
- Escalabilidad para el futuro ya que el sistema de ejecución opera sin problemas en sistemas con cientos de núcleos. Las herramientas están disponibles para analizar la aplicación y predecir lo bien que va a escalar.

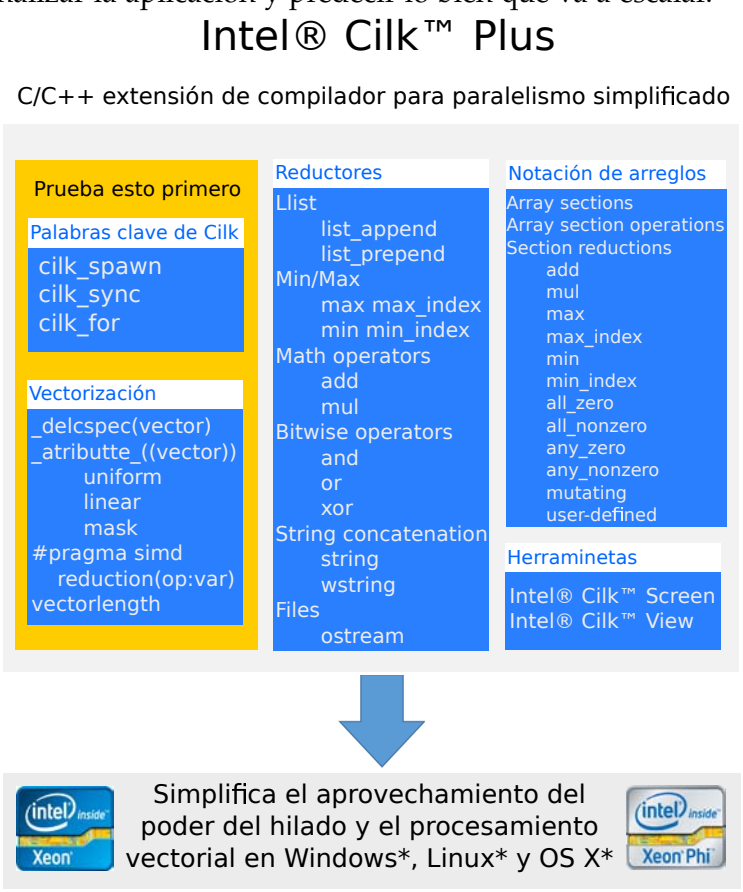

Figura 2.4: Intel Cilk Plus

Como los sistemas multinúcleo y SMP se vuelven mas frecuentes, los nuevos saltos de rendimiento llegarán a la industria conforme se vayan adoptando las técnicas de programación en paralelo. Sin embargo, muchos entornos paralelos consisten en confusas, complejas y propensas a errores reglas y constructores. El lenguaje `Intel Cilk Plus`, construido sobre la tecnología `Cilk` desarrollado en el MIT en las últimas dos décadas, se ha diseñado para proporcionar un modelo sen-

cillo, bien estructurado que hace que el desarrollo, la verificación y el análisis sea fácil. Debido a que `Intel Cilk Plus` es una extensión a C y C++, los programadores normalmente no necesitan reestructurar los programas de manera significativa con el fin de añadir paralelismo.

2.5 OPENMP

`OpenMP` es una API (Application Program Interface), que es utilizado para diseñar explícitamente programas paralelos multihilado en memoria compartida. Un programa `OpenMP` es un programa secuencial con directivas dispuestas en los puntos apropiados del código fuente, y este esquema de programación permite expresar paralelismo. Un programa que contiene directivas `OpenMP` no es un programa paralelo, en realidad el que genera el programa paralelo es el compilador, las directivas solo le indican como debe comportarse.

`OpenMP` está compuesto de tres elementos:

- Las directivas de compilación.
- Las rutinas de librería (Runtime Library).
- Las variables de entorno.

La API ha sido especificada para `C/C++` y `FORTRAN` y es soportado en muchas plataformas. Las especificaciones de `OpenMP` son diseñadas por el ARB (Architectural Review Board), cuya misión es la de estandarizar APIs para el procesamiento en memoria compartida.

La primera versión de `OpenMP` aparece en octubre de 1997. Se trata de la versión 1.0 para `FORTRAN`. Posteriormente, a finales de 1998, se presentan las versiones 1.0 para `C/C++`. En junio de 2000 se presenta la versión 2.0 para `FORTRAN` y en abril de 2002 la versión 2.0 para `C/C++`.

`OpenMP` no está orientado para sistemas de memoria distribuida (clusters) y entre sus características destacables se encuentran:

- Proporciona un estándar para una variedad de plataformas y arquitecturas de memoria compartida.
- Es relativamente fácil de usar porque establece un conjunto de directivas para programar máquinas de memoria compartida, proporcionando una aproximación incremental al desarrollo de programas paralelos y la capacidad de implementar paralelismo de grano fino y paralelismo de grano grueso simultáneamente.

- Es sumamente portable, ya que prácticamente cualquier compilador de FORTRAN (77, 90 y 95), C y C++, sobre cualquier plataforma lo soporta.

El modelo de programación en `OpenMP` se basa en que un proceso puede consistir de múltiples hilos de ejecución en la máquina de memoria compartida. Utiliza el modelo `fork-join` y el programador tiene control explícito de la paralelización. El programa puede contener regiones de código secuenciales y regiones paralelizables y las directivas intervienen en las regiones paralelizables.

Básicamente un programa `OpenMP` es un programa secuencial al que se añaden las directivas en las líneas adecuadas. Cuando al compilador no se le indica procesar las directivas las ignorará y generará el código ejecutable de forma habitual. Un compilador con capacidades `OpenMP` reconocerá este tipo de directivas y generará un código ejecutable paralelizado que se podrá ejecutar en máquinas de memoria compartida.

Las directivas `OpenMP` especifican la sección de programa en la que los hilos se crean y ejecutan. Tales secciones se denominan regiones paralelas. Dependiendo de la semántica y de la lógica del programa, la directiva permitirá expresar distintas construcciones paralelas. Las regiones paralelas se especifican, por tanto, mediante constructores paralelos. No hay límite sobre el número de constructores paralelos a especificar en un programa.

Los hilos paralelos ejecutan la tarea que el hilo maestro habría abordado de forma individual en un programa secuencial. Las sentencias del programa que se encuentran en el constructor paralelo, se ejecutan en paralelo por cada hilo. Una vez que se ha terminado el constructor paralelo, los hilos del equipo se sincronizan y solo el hilo maestro continúa su ejecución.

En una computadora de memoria compartida cada hilo paralelo puede ejecutarse en un solo núcleo compartiendo la misma memoria física, sin embargo, el modelo no garantiza que no haya más de un hilo ejecutándose en un procesador simultáneamente.

Las especificaciones de `OpenMP` soportan además paralelismo anidado, la inclusión de constructores paralelos en el interior de constructores paralelos, y soporta también la gestión dinámica de hilos de ejecución.

Es necesario señalar que en `OpenMP` es posible controlar la ejecución del código paralelo a través variables de entorno.

Finalmente, algunas de las dificultades que presenta el modelo de programación en memoria compartida son originadas por la necesidad de coordinación en el acceso a los recursos por parte de los hilos en ejecución, con el fin de evitar condiciones de carrera, interbloqueos, ejecuciones semánticamente incorrectas, no

obstante, `OpenMP` ofrece un conjunto de directivas orientadas a sincronización, la ejecución en exclusión mutua y para la manipulación consistente de datos.

3

ESQUEMAS NUMÉRICOS DE LAS APLICACIONES NUMÉRICAS EN ESTUDIO

En este capítulo se darán los esquemas numéricos de las aplicaciones que se pretenden implementar y examinar utilizando un esquema multi-hilado. Se abordarán dos aplicaciones, la simulación de flujos supersónicos en eyectores y el cálculo directo de la gravimetría a partir de un ensamble de primas.

3.1 CÁLCULO DIRECTO DE LA GRAVIMETRÍA A PARTIR DE UN ENSAMBLE DE PRISMAS

Los estudios de gravimetría y magnetometría, conocidos como métodos potenciales, son herramientas utilizadas en una gran variedad de objetivos de exploración geofísica.

La determinación de cuerpos bajo la superficie de la Tierra, a través de medidas del campo de gravedad o magnetismo, es un problema de inversión geofísica, que se caracteriza por ser mal planteado, no único y de solución no lineal.

Un problema de inversión se puede considerar como un proceso de optimización en el que se busca un modelo adecuado y que mejor se adapte a ciertos datos observados, minimizando una función de error que representa la diferencia entre los datos reales y los que describen un modelo propuesto. En los problemas no lineales, la solución puede ser obtenida iterativamente aproximando valores de un modelo linealizado; sin embargo, las técnicas de inversión por modelos linealizados pueden sufrir algunas inestabilidades numéricas debido al mal condicionamiento de las matrices y con datos para los cuales la inversión obtiene soluciones que no son únicas. En este sentido, las técnicas de optimización pueden considerarse como una alternativa para obtener soluciones a problemas de inversión geofísica.

La solución en inversión de datos aplicando métodos de optimización global consiste en la búsqueda sobre el espacio de parámetros, en el que se encuentran soluciones por ensayo y error comparándolas con subsecuentes modelos. Este proceso continúa hasta que la diferencia entre las observaciones del modelo propuesto y observaciones de datos reales se reducen hasta un error deseado. Así, las soluciones se obtienen directamente del espacio de parámetros, eliminando cualquier

restricción hacia el condicionamiento de la matriz de sensitividades, estos métodos también se denominan heurísticos, porque buscan soluciones óptimas o cuasi óptimas sin garantizar el óptimo absoluto.

Los datos sintéticos utilizados en este trabajo se generan a partir del modelo interpretado de secciones sísmicas, para el cual se obtiene la respuesta gravimétrica como modelo inicial (forward modelling) y luego se obtiene una solución de los parámetros del modelo de interés de forma iterativa hasta que se obtiene una coincidencia aceptable entre los datos observados y los datos sintéticos.

Así la interpretación obedece a una serie de pasos en el que se determina la distribución espacial de las fuentes. Para este propósito se debe contar con algoritmos que puedan aproximar datos conforme a la realidad y que muy a menudo en los modelos tridimensionales, suelen ser cálculos de alto costo computacional y gran requerimiento de memoria debido a la resolución de la información.

El modelado directo se basa en la información inicial obtenida de un modelo construido comúnmente de interpretaciones geológicas. De este modelo se calcula su anomalía y se compara con la anomalía observada, luego se observan las diferencias y se ajustan los parámetros (cuerpos con densidades o susceptibilidad magnética) de manera repetida hasta que los datos observados y los calculados se consideren suficientemente parecidos.

Una vez que se han completado los estudios de exploración gravimétrica o magnética, además de aplicarse los procesos necesarios para remover los campos regionales de manera apropiada, se comienza la etapa de interpretación. En el problema se trata de estimar los parámetros de las fuentes que corresponden a las observaciones de campo potencial, incorporando la información geológica – geofísica disponible, entre algún otro tipo de información.

Esta aplicación sirve para extraer información más detallada de las masas ocultas en el subsuelo mediante la medición de los gradientes gravitatorios. La fuerza de gravedad G tiene tres elementos que representan los componentes de la gravedad en las tres direcciones ortogonales x, y, z (G_x, G_y, G_z). El gradiente de G representa el campo gravitacional en forma de tensor y tiene nueve componentes: (G_{xx}, G_{yx}, G_{zx}, G_{xy}, G_{yy}, G_{zy}, G_{xz}, G_{yz}, G_{zz}) como se muestra en la Figura 3.5.

La cartografía de los gradientes gravitacionales permite mejorar el estudio convencional gravitacional al ofrecer una mejor resolución del subsuelo. La medición de la diferencia entre dos sensores para obtener el gradiente permite la eliminación del origen del error.

El modelado directo de la gravedad gradiometría consiste en la conformación directa de los datos gravimétricos donde el modelo inicial del cuerpo de la fuente se construye a partir de la intuición geológica y geofísica. Calculamos las anomalías del modelo y los comparamos con la anomalía observada después de lo cual los

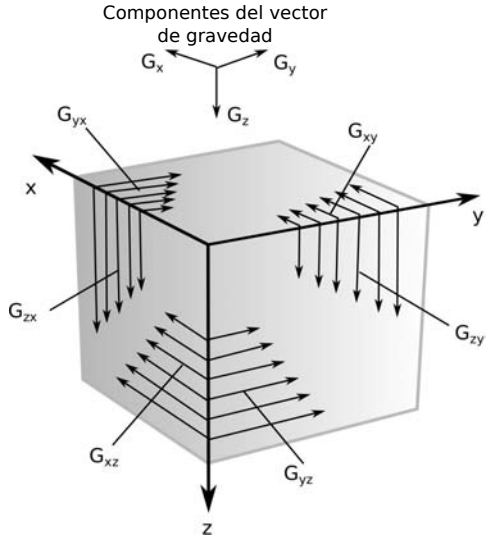

Figura 3.5: Componentes del gradiente del tensor gravitacional.

parámetros se adaptan con el fin de mejorar el ajuste entre ambas anomalías. Estos tres pasos para el ajuste de las propiedades del cuerpo, es decir, el cálculo de las anomalías, la comparación de las anomalías y los ajustes entre ambas anomalías, se repiten hasta que las anomalías observadas y calculadas son suficientemente similares con algún criterio de error.

La aplicación numérica, básicamente, consiste en la adaptación de un conjunto de prismas rectangulares que aproximan el volumen de la masa en el subsuelo. Si se elige suficientemente pequeño, cada prisma puede ser considerado para ser de densidad constante. Luego, por el principio de superposición, la anomalía gravitacional del cuerpo en cualquier punto se puede aproximar mediante la adición de los efectos de todos los prismas en ese punto. Aunque este método parece simple, mientras que la reducción del tamaño de los prismas para ajustar el cuerpo de la fuente, el tiempo de cálculo se incrementa considerablemente. Además, aunque hay otros tipos de aproximaciones tales como puntos de masa o de tesseroids, a menudo por razones de simplicidad, los investigadores prefieren generar los prismas para modelar un cuerpo.

De este modo, los cambios computacionales consisten, en esencia, en el cálculo de la respuesta gravimétrica producida por el cuerpo de prisma rectangular con una constante de densidad o constante de magnetización con respecto a un grupo de puntos de observación (ver Figura 3.6). El conjunto de prismas es conocido como un ensamble de prismas y no es necesariamente regular. Un ensamble de prismas puede ser configurado en cualquier orientación con el único principio de que no pueden superponerse. Puesto que el campo gravitacional (o campo magnético) se reúne con la propiedad de superposición con respecto a la observación,

si f es la respuesta calculada en un punto (x, y), entonces la respuesta observada en el punto $f(x, y)$ viene dada como:

$$f(x, y) = \sum_{k=1}^{M} G(\rho_k, x, y), \tag{1}$$

donde M es el número total de prismas y ρ es la densidad del prisma.

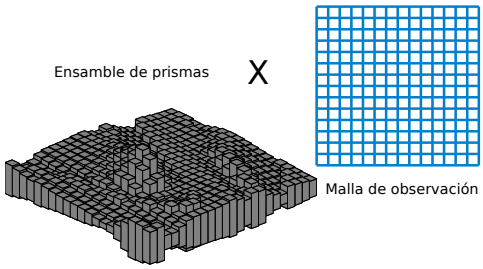

Figura 3.6: Configuración del conjunto de prismas en contra de su malla de observación. Para cada prisma en el conjunto de una anomalía se calcula contra cada punto de observación.

Una de las características de `OpenMP` es que la división del cómputo se lleva a cabo de manera implícita. Por lo tanto, la partición de M prismas que conforman el cuerpo en el subsuelo se realiza automáticamente mediante un algoritmo de equilibrio incluido en `OpenMP`. En este caso, la decisión se deja al compilador, en el 99 % de los casos es óptimo (Zhang et al., 2004).

Las exploraciones a través de la gradiometría de la gravedad o la gravedad del tensor completo (FTG) es una técnica de exploración que está basada en múltiples variaciones del campo gravitacional, las variaciones son medidas a través de los gradientes locales en diferentes direcciones. La principal ventaja que viene con la comparación de las componentes del vector de gravedad se encuentra en una resolución mejorada de las fuentes poco profundas. De esta manera es posible construir modelos más precisos que puede detallar objetos tales como gas, embalses y depósitos de petróleo o de mineral de hidrocarburos.

El FTG consta de 9 componentes, sin embargo, sólo 5 están registrados como medidas independientes desde diversas perspectivas. Estas medidas están relacionadas por el hecho de que pueden ser grabadas de la misma fuente geológica. Así, el proceso de co-juntar puede proporcionar una respuesta de datos con una mejor localización de la fuente anomalía.

Cada uno de los tensores describe un objeto de una manera particular. El tensor G_{zz} encuentra el origen, G_{xx} y G_{yy} identifican los límites de la fuente, G_{xz} identifica los límites, G_{yz} identifica los ejes, tanto de alta y baja, que a su vez definen las tendencias de falla, y G_{xy} muestra las anomalías que apuntan hacia el centro de la masa de origen.

La respuesta gradiométrica o modelado directo se obtiene por medio de un algoritmo que requiere el cálculo de un significativo número de prismas. Así, el

coste computacional es considerablemente alto si el propósito es la aproximación de geometrías complejas con pequeños prismas.

El campo gravitatorio puede ser expresado en términos del potencial de gravedad $U_g(\mathbf{r})$ como:

$$\mathbf{g}(\mathbf{r}) = \nabla U_g(\mathbf{r}), \tag{2}$$

la potencia U_g por el volumen V tiene la siguiente aproximación (Pedersen and Rasmussen, 1990):

$$U_g(\mathbf{r}) = \gamma \iiint_V \frac{1}{|r - r'|} \rho(\mathbf{r}') dv, \tag{3}$$

donde γ es la constante universal de gravedad, ρ es la densidad en el volumen, r es la posición del punto de observación y r' es el elemento del volumen dado por dv.

La primera derivada espacial del potencial U_g (Eq. 3) puede ser expresada como:

$$G(\mathbf{r})_\alpha = \gamma \iiint_V \frac{\partial}{\partial \alpha} \frac{1}{|r - r'|} \rho(\mathbf{r}') dv, \tag{4}$$

donde α es cualquiera de las direcciones x, y, z, por lo tanto $G(\mathbf{r})$ en las diferentes direcciones se puede escribir como:

$$G_x(\mathbf{r}) = -\gamma \iiint_V (x - x') \frac{1}{|r - r'|^3} \rho(\mathbf{r}') dv, \tag{5}$$

$$G_y(\mathbf{r}) = -\gamma \iiint_V (y - y') \frac{1}{|r - r'|^3} \rho(\mathbf{r}') dv, \tag{6}$$

$$G_z(\mathbf{r}) = -\gamma \iiint_V (z - z') \frac{1}{|r - r'|^3} \rho(\mathbf{r}') dv. \tag{7}$$

Para el cálculo de los tensores aplicamos la segunda derivada parcial a la Eq. (4) para obtener:

$$G_{\alpha\beta}(\mathbf{r}) = \frac{\partial^2}{\partial \alpha \partial \beta} U_g(\mathbf{r}) \ , \quad \alpha, \beta = x, y, z, \tag{8}$$

por lo tanto los tensores pueden ser aproximados como:

$$G_{xx}(\mathbf{r}) = \gamma \iiint_V \frac{3(x - x')^2 - (r - r')^2}{|r - r'|^5} \rho(\mathbf{r}') dv, \tag{9}$$

$$G_{xy}(\mathbf{r}) = \gamma \iiint_V \frac{3\,(x-x')\,(y-y')}{|r-r'|^5}\rho(\mathbf{r}')dv, \tag{10}$$

$$G_{xz}(\mathbf{r}) = \gamma \iiint_V \frac{3\,(x-x')\,(z-z')}{|r-r'|^5}\rho(\mathbf{r}')dv, \tag{11}$$

$$G_{yy}(\mathbf{r}) = \gamma \iiint_V \frac{3\,(y-y')^2-(r-r')^2}{|r-r'|^5}\rho(\mathbf{r}')dv, \tag{12}$$

$$G_{yz}(\mathbf{r}) = \gamma \iiint_V \frac{3\,(y-y')\,(z-z')}{|r-r'|^5}\rho(\mathbf{r}')dv, \tag{13}$$

$$G_{zz}(\mathbf{r}) = \gamma \iiint_V \frac{3\,(z-z')^2-(r-r')^2}{|r-r'|^5}\rho(\mathbf{r}')dv. \tag{14}$$

Por conveniencia podemos denotar el tensor gravitacional en forma matricial de la siguiente manera:

$$\mathbf{r} = \begin{bmatrix} G_{xx} & G_{xy} & G_{xz} \\ G_{yx} & G_{yy} & G_{yz} \\ G_{zx} & G_{zy} & G_{zz} \end{bmatrix}, \tag{15}$$

donde el seguimiento satisface la ecuación de Laplace ($Trace(\Gamma) = G_{xx} + G_{yy} + G_{zz} = 0$).

La respuesta del componente del tensor G_{zz} pueden aproximarse en una manera discreta debido al hecho de que un prisma rectangular de densida constante (ρ) puede ser representado (ver Figura 3.7):

$$G_{zz} = \gamma\rho \sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2} \mu_{ijk} \arctan\frac{x_i y_i}{z_k r_{ijk}}, \tag{16}$$

donde $x_i = x_p - \xi_i$, $y_i = y_p - \eta_i$, $z_i = z_p - \zeta_i$, $r_{ijk} = \sqrt{x_i^2 + y_i^2 + z_i^2}$ y $\mu_{ijk} = (-1)^i(-1)^j(-1)^k$. La aproximación discreta para el resto de componentes puede ser revisado en (Couder-Castaneda et al., 2015). En la Figura 3.8 mostramos el resultado del cálculo de los componentes de un prisma de medición $200m^3$ a una

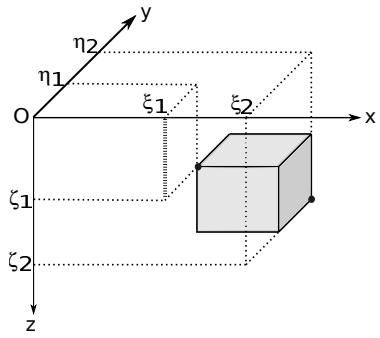

Figura 3.7: Delimitación del prisma rectangular en el plano cartesiano.

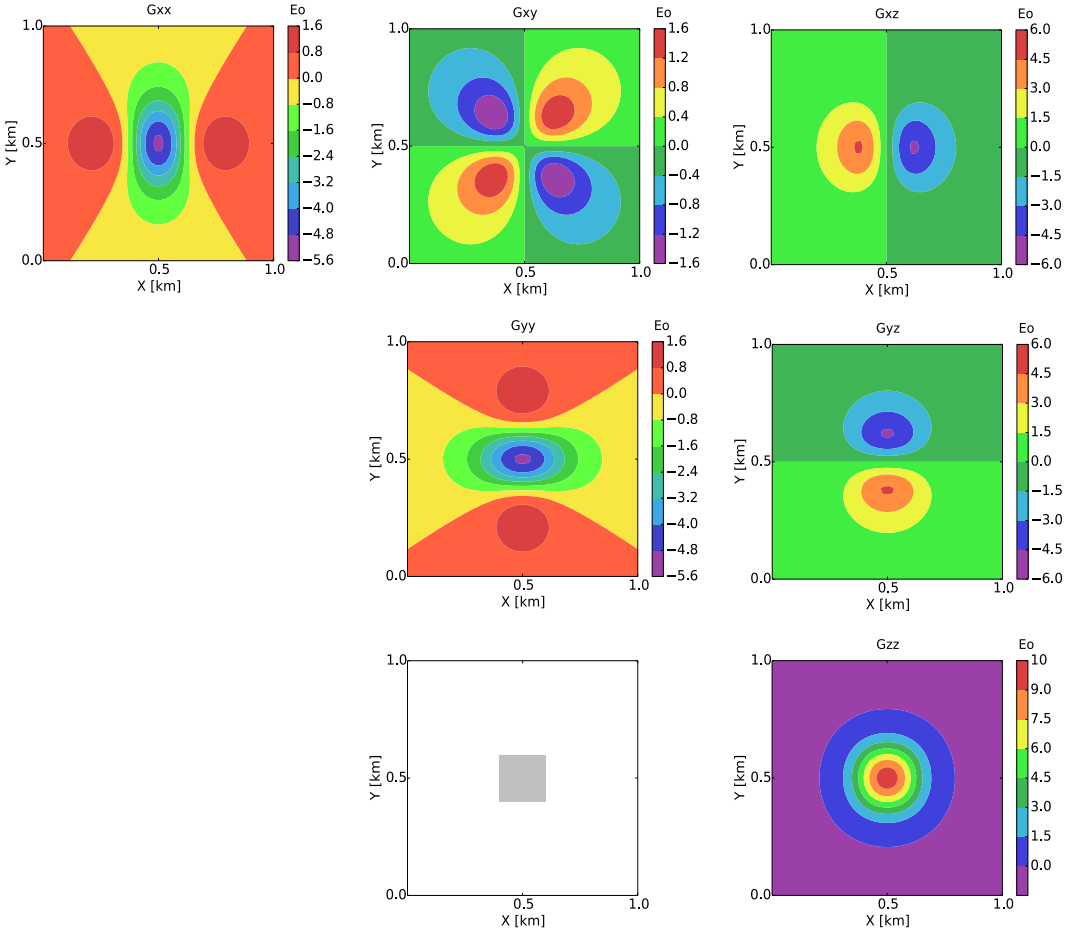

Figura 3.8: Respuesta gradiométrica para un prisma.

altura de 0.01Km y con la malla de observación de 1Km x 1Km y discretiza cada 20m.

La aplicación del cálculo directo de los componentes FTG se lleva a cabo en una parte de un área de exploración en el Golfo de México para el que contamos con la información geológica, sin embargo omitimos la ubicación exacta y los detalles de la zona debido a la confidencialidad empresarial. El objetivo es localizar las características relacionadas con la distribución y la geometría de los cuerpos salinos, mientras que la integración de la información de la zona (véase la Figura 3.9). Se muestran varios horizontes incluyendo la distribución de los cuerpos de solución salina, y de esta manera la información geológica determinará los límites precisos de la densidad, determinando de ese modo las amplitudes de las anomalías en la zona en cuestión.

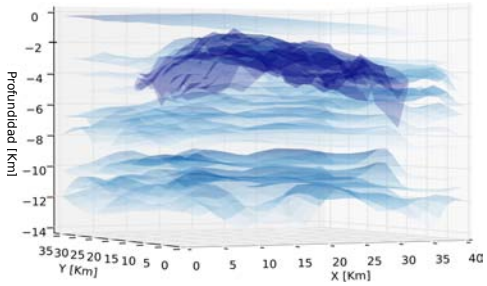

Figura 3.9: Horizontales geológicos con cuerpos salinos.

Basando el modelo en un sistema de referencia cartesiano y profundidad positiva, la geometría está compuesta por un conjunto de prismas de densidad variable pero con dimensiones constantes de 500, tanto en las direcciones x y y, y 50m en la dirección z. El dominio mide 40.5 x 36.5 x 14.1 kilometros al este-oeste, norte-sur, y las instrucciones de profundidad, respectivamente (véase la Figura 3.10). El número total de prismas que componen el modelo es 1,667,466 con una maflla de observación de $321 \times 289 = 92,769$ puntos. El número de prismas con una densidad distinta de cero es 1,455,425.

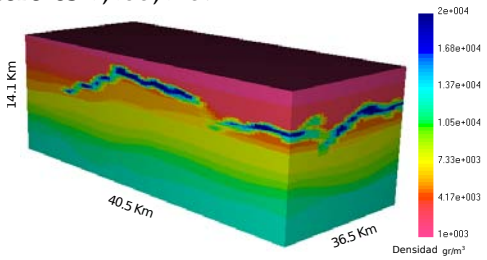

Figura 3.10: Modelo de los prismas correspondientes a los horizontes geológicos (sin escala).

3.2 SIMULACIÓN DE FLUJOS SUPERSÓNICOS EN EYECTORES

La aplicación que se pretende acelerar es un código para simular el flujo supersónico en eyectores, comúnmente utilizados en la industria petrolera. Los eyectores

confinan fluidos en movimiento bajo condiciones controladas, que descargan a una presión intermedia entre las presiones del fluido motor y de succión. El funcionamiento de un eyector está dado por el principio de conservación de cantidad de movimiento de las corrientes involucradas. Los eyectores son vistos como venturis, cuyo trabajo se basa en la trasmisión de energía por impacto de un fluido a gran velocidad, contra otro en movimiento o en reposo. Este impacto genera una mezcla de fluido a una velocidad moderadamente elevada, que luego disminuye hasta obtener una presión final mayor que la del fluido de menor velocidad.

Los eyectores se emplean comúnmente para extraer gases de los espacios donde se hace vacío, como por ejemplo en los condensadores, en los sistemas de evaporación, en torres de destilación al vacío y en los sistemas de refrigeración, donde los gases extraídos son generalmente incondensables, como el aire. También, se usan en el mezclado de corrientes en los procesos de sulfitación en ingenios azucareros. En la Figura 3.11 se muestran los componentes del eyector.

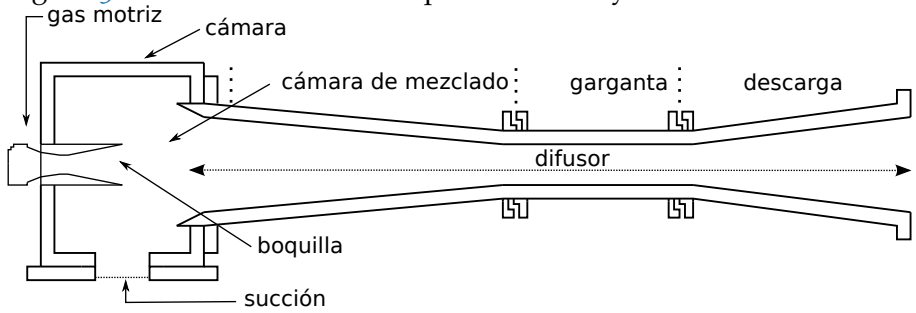

Figura 3.11: Diferentes partes del eyector. Las simulaciones numéricas se llevan a cabo a través del difusor.

Sin embargo, las simulaciones han sido altamente costosas en cuanto a tiempo de cómputo se refiere, traduciéndose a su vez, en tiempos de espera que retrasan los resultados. Por ejemplo, en un procesador Intel Xeon a 3.47 Ghz, 3 segundos de simulación real toman aproximadamente 65 horas de tiempo de cómputo en forma serial. Por esta razón, es necesario generar alternativas para la reducción del tiempo de cómputo, pero manteniendo el código fuente intacto lo más posible, debido a la complejidad del esquema numérico utilizado.

La conducta que rige el flujo viene dada por las ecuaciones compresibles no lineales de Navier-Stokes, que se pueden escribir en la forma de conservación como:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0. \tag{17}$$

Los vectores U, F y G están definidos como:

$$U = \begin{cases} \rho \\ \rho u \\ \rho v \\ \rho e \end{cases}$$

$$F = \begin{cases} \rho u \\ \rho u v + p - \tau_{xx} \\ \rho u v - \tau_{xy} \\ \rho u \left(e + \frac{u^2+v^2}{2} \right) + p u - u\tau_{xx} - v\tau_{xy} + q_x \end{cases}$$

$$G = \begin{cases} \rho v \\ \rho u v - \tau_{xy} \\ \rho v^2 + p - \tau_{yy} \\ \rho v \left(e + \frac{u^2+v^2}{2} \right) + p v - u\tau_{xy} - v\tau yy + q_y \end{cases}$$

Donde ρ representa la densidad, u y v las componentes horizontal y vertical, respectivamente, de la velocidad, p es la presión, e la energía, τ_{xx}, τ_{yy}, τ_{xy} el tensor de esfuerzos viscosas y q_x, q_y los flujos de calor difusivos.

Dado que estamos trabajando con un gas perfecto es posible reemplazar e a favor de u, v, p y ρ de la siguiente manera: $\frac{u^2+v^2}{2} + \frac{p}{(\gamma-1)\rho}\ \rho$, donde γ es la constante isentrópica.

Para mayor claridad en la implementación de código cada elemento de las variables de flujo U, F y G pueden ser denotadas por:

$$U_1 = \rho,$$
$$U_2 = \rho u,$$
$$U_3 = \rho v,$$
$$U_4 = \frac{\gamma}{\gamma-1} p + \frac{u^2+v^2}{2} \rho.$$

$$F_1 = \rho u,$$
$$F_2 = \rho u^2 + p - \tau_{xx},$$
$$F_3 = \rho u v - \tau_{xy},$$
$$F_4 = \frac{\gamma}{\gamma-1} p u + \rho u \frac{u^2+v^2}{2} - u\tau_{xx} - v\tau_{xy} + q_x.$$

$$G_1 = \rho v,$$
$$G_2 = \rho u v - \tau_{xy},$$
$$G_3 = \rho v^2 + p - \tau_{yy},$$
$$G_4 = \frac{\gamma}{\gamma-1} p v + \rho v \frac{u^2+v^2}{2} - u\tau_{xy} - v\tau_{yy} + q_y.$$

Los términos de estrés viscosos están escritos en términos de la derivada de la velocidad como:

$$\tau_{xy} = \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right), \tag{18}$$

$$\tau_{xx} = \lambda\left(\nabla \cdot \mathbf{V}\right) + 2\mu\left(\frac{\partial u}{\partial x}\right), \tag{19}$$

$$\tau_{yy} = \lambda\left(\nabla \cdot \mathbf{V}\right) + 2\mu\left(\frac{\partial v}{\partial y}\right). \tag{20}$$

donde μ es la viscosidad dinámica y λ está definido como $\lambda = 2/3\mu$. Del mismo modo, los componentes del vector de flujo de calor (de la ley de Fourier de calor) se definen como:

$$q_x = -k\frac{\partial T}{\partial x}, \tag{21}$$

$$q_y = -k\frac{\partial T}{\partial y}. \tag{22}$$

Las variaciones de la viscosidad dinámica y la conductividad térmica se consideran dependientes de la temperatura y que se aproximan mediante la ley Shutherland como:

$$\mu\left(T\right) = \mu_0\left(\frac{T}{T_0}\right)^{\frac{3}{2}}\frac{T_0 + S}{T + S} \tag{23}$$

y

$$k(T) = \frac{\mu(T)\gamma R}{(\gamma - 1)\,\mathrm{Pr}} \tag{24}$$

donde μ es la viscosidad, S la temperatura Sutherland y Pr el número de Prandtl. En esta aplicación son usados los valores típicos de $T_0 = 273\mathrm{K}$, $S = 110.5\mathrm{K}$, $\mu_0 = 1.68 \times 10^{-5}$ Pa, $\gamma = 1.4$ y $\mathrm{Pr} = 0.71$.

3.2.1 *La transformación curvilínea*

Para generar la malla del difusor del eyector que se muestra en la Figura 3.12, se utiliza un sistema de coordenadas curvilíneas$(\xi, \eta) \in [0, L] \times [0, 1]$ representado en

la Figura 3.13 para la geometría del eyector. Las líneas que conforma el plano ξ y η forman una malla rectangular en el plano computacional. La transformación curvilínea adecuada para generar un sistema de coordenadas ajustado a las fronteras del difusor del eyector se define como sigue (Couder-Castaneda, 2009):

$$\xi = x, \tag{25}$$

$$\eta = \frac{y - y_s(x)}{y_z(x) - y_s(x)}, \tag{26}$$

donde $y_s(x)$ y $y_z(x)$ son las funciones que describen respectivamente las paredes inferior y superior del difusor y están definidas en metros como:

$$y_s(x) = \begin{cases} 0.0 \text{ para } x \leqslant 10.0 \\ \frac{8}{167{,}875}(x - 10.0) \text{ para } 10.0 < x \leqslant 177{,}875 \\ 8.0 \text{ para } 177{,}875 < x \leqslant 299{,}125 \\ -\frac{8}{89}(x - 299{,}125) + 8 \text{ para } x > 299{,}125, \end{cases} \tag{27}$$

$$y_z(x) = \begin{cases} 42.0 \text{ para } x \leqslant 10.0 \\ -\frac{8}{167{,}875}(x - 10.0) + 42 \text{ para } 10.0 < x \leqslant 177{,}875 \\ 34.0 \text{ para } 177{,}875 < x \leqslant 299{,}125 \\ \frac{8}{89}(x - 299{,}125) + 34 \text{ para } x > 299{,}125. \end{cases} \tag{28}$$

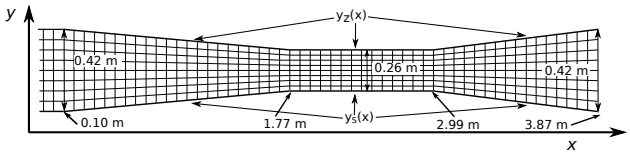

Figura 3.12: El tamaño de la malla del eyector es $1,101 \times 41$ puntos discretos y se generan usando el criterio de marcha CFL.

Con esta transformación, los valores de ξ van desde 0 hasta L (la longitud horizontal del difusor) y η varía de 0.0 (pared inferior) a 1.0 (pared superior) en el plano computacional. El límite de entrada se produce en $\xi = 0.0$, y el límite de salida física está en $\xi = L$. Con esta transformación, podemos manejar adecuadamente las ondas de compresión o de expansión que se producen en un flujo supersónico debido al cambio de la geometría del difusor (Couder-Castaneda, 2009). Por simplicidad denotamos las diferentes métricas de la transformación de la siguiente manera:

$\frac{\partial \xi}{\partial x} = \xi_x$, $\frac{\partial \xi}{\partial y} = \xi_y$, $\frac{\partial \eta}{\partial x} = \eta_x$, $\frac{\partial \eta}{\partial y} = \eta_y$.

Estas matrices estan definidas por las derivadas de las ecuaciones (27) y (28) como se muestra: $\xi_x = 1$, $\xi_y = 0$, $\eta_x = \frac{1}{y_z(x)-y_s(x)}$, $\eta_y = \frac{\eta[y_s'(x)-y_z'(x)]}{y_z(x)-y_s(x)}$. Con la transformación de las matrices, la ecuación (17) es entonces reformulada como:

$$\frac{\partial U_i}{\partial t} = -\left[\frac{\partial F_i}{\partial \xi} + \frac{\partial F_i}{\partial \eta}\eta_x\right] - \left[\frac{\partial G_i}{\partial \eta}\eta_y\right], \; i = 1...4 \tag{29}$$

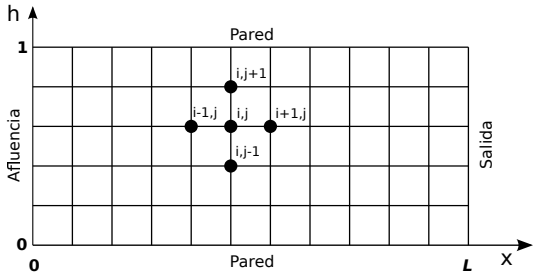

Figura 3.13: Plano computacional

3.2.2 *Parámetros iniciales y condiciones de frontera*

Las condiciones de frontera utilizados en la salida del difusor es la formulación CPML. Esta técnica se puede implementar fácilmente en un código de diferencias finitas existente sin capas perfectamente adaptadas (LMP), simplemente cambiando las derivadas espaciales ∂_x con $\frac{1}{\kappa_x}\partial_x + \Psi_x$ y avanzando Ψ_x en el tiempo utilizando el mismo esquema de evolución en el tiempo como para el resto de variables. Si aplicamos esta reformulación de las derivadas de espacio a la ecuación principal (17) y si llevamos a cabo una transformación curvilínea del dominio computacional, las derivadas de $\frac{1}{\kappa_x}\partial_x F_i + \Psi_x$ a lo largo de x los ejes se calculan de acuerdo a la regla de la cadena como $\frac{1}{\kappa_x}\left(\partial_\xi F_i \xi_x + \Psi_\xi^{F_i} + \partial_\eta F_i \eta_x + \Psi_\eta^{F_i}\eta_x\right)$. Con la formulación de la CPML hecha, podemos sustituir la ecuación que rige (29) de la siguiente manera:

$$\frac{\partial U}{\partial t} = -\left[\left(\frac{1}{k_x}\frac{\partial F}{\partial \xi} + \Psi_\xi^F\right) + \left(\frac{1}{k_x}\frac{\partial F_i}{\partial \eta} + \Psi_\eta^{F_i}\right)(\eta_x)\right] - \left[\left(\frac{1}{k_y}\frac{\partial G_i}{\partial \eta} + \Psi_\eta^{G_i}\right)(\eta_y)\right]. \tag{30}$$

Podemos aplicar el CPML actualizando la variable de matriz Ψ para cada flujo F_i y G_i a lo largo de la dirección respectiva ξ o η en cada paso de tiempo. En ξ dirección Ψ_x se obtiene de la siguiente manera:

$$\Psi_x^{n+1}(f) = b_x \Psi_x^n(f) + a_x(\partial_x f)^{\left(n+\frac{1}{2}\right)}, \tag{31}$$

donde f puede ser F_i, G_i, $b_x = e^{\frac{d_x}{k_x}+\alpha_x}$ y $\alpha_x = \frac{d_x}{k_x(d_x+k_x\alpha_x)}(b_x - 1)$. En la ecuación que gobierna (17) podemos reagrupar los términos de memoria como:

$$\Psi_x^{F_i} = \Psi_\xi^{F_i}\xi_x + \Psi_\eta^{F_i}\eta_x, \tag{32}$$

$$\Psi_y^{G_i} = \Psi_\eta^{G_i}\eta_x. \tag{33}$$

y aplicamos (31) lo que lleva a

$$\Psi_x^{n+1}(F_i) = b_x\Psi_x^n(F_i) + a_x\left(\frac{\partial F_i}{\partial\xi}\xi_x + \frac{\partial F_i}{\partial\eta}\eta_x\right)^{n+\frac{1}{2}} \tag{34}$$

$\Psi_y^{G_i} = 0$, entonces $\Psi = 0$ porque no hay límites de PML en las paredes inferior y superior ($a_y = b_y = 0$).

El campo de flujo se inicializa con una presión, densidad y temperatura igual a 101,000Pa, 1.23Kg/m^3 y 286K, respectivamente; el campo de velocidades es inicializado para ambos u y v con 0.0m/s y en la entrada a $u = 600$m/s y $v = 0.0$. En la frontera de aguas arriba se inyecta una velocidad de Mach 2.0. La simulación se realiza durante 20 millones de pasos de tiempo, con un $\Delta t = 1 \times 10^{-6}$ s.

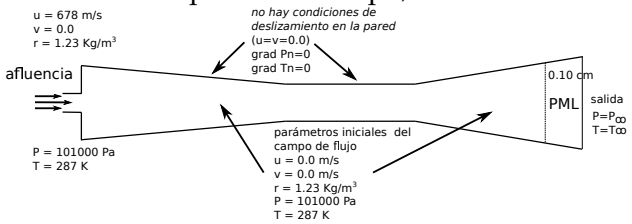

Figura 3.14: Las condiciones iniciales estan dadas, así como el tipo de flujo de entrada, las condiciones de salida y de la pared. A la izquierda, las condiciones de entrada constante se establecen para componentes de la velocidad, temperatura y presión en todo el tiempo de simulación. En las paredes superior e inferior, se establecen las condiciones de velocidad antideslizantes ($u = v = 0.0$m/s) y de presión nula y gradientes de temperatura ($\nabla P \cdot n = 0$ y $\nabla T \cdot n = 0$) se imponen en cualquier momento. En la parte derecha de la capa de PML 20 puntos, se imponen condiciones de Dirichlet.

3.2.3 *El esquema numérico*

Se utiliza un esquema de tiempo de paso a paso *predictor-corrector* de segundo orden para hacer frente a los cambios de las diferentes variables que intervienen y se aplica a la ecuación de gobierno final (30) que representa un sistema de ecuaciones y procede de la siguiente manera:

Paso pronosticador:

El $U^{n+\frac{1}{2}}$ pronosticado se calcula como:

$$U_{i,j}^{n+\frac{1}{2}} = U_{i,j}^{n} + \Delta t \left[\frac{1}{\kappa_{\xi_{i,j}}} \frac{F_{i-1,j}^{n} - F_{i,j}^{n}}{\Delta \xi_{i-1}} + \Psi_{\xi}^{n+\frac{1}{2}}[F^{n}] \right] + \frac{\Delta t}{\Delta \eta}(F_{i,j-1}^{n} - F_{i,j}^{n})\eta_{x_{i,j}} + \frac{\Delta t}{\Delta \eta}(G_{i,j-1}^{n} - G_{i,j}^{n})\eta_{y_{i,j}} + S_{i,j}^{n+\frac{1}{2}} \quad (35)$$

donde la viscosidad artificial $S_{i,j}^{n+\frac{1}{2}}$ es calculada como:

$$S_{i,j}^{n+\frac{1}{2}} = \frac{C_x |p_{i+1,j}^{n} - 2p_{i,j}^{n} + p_{i-1,j}^{n}|}{p_{i+1,j}^{n} + 2p_{i,j}^{n} + p_{i-1,j}^{n}} \times \left(U_{i+1,j}^{n} - 2U_{i,j}^{n} + U_{i-1,j}^{n}\right) + \frac{C_y |p_{i,j+1}^{n} - 2p_{i,j}^{n} + p_{i,j-1}^{n}|}{p_{i,j+1}^{n} + 2p_{i,j}^{n} + p_{i,j-1}^{n}} \times \left(U_{i,j+1}^{n} - 2U_{i,j}^{n} + U_{i,j-1}^{n}\right); \quad (36)$$

la viscosidad artificial se introduce para evitar posibles oscilaciones cerca de gradientes fuertes.

En la ecuación (36) C_x y C_y son dos parámetros que tienen valores típicos y oscilan de 0.01 a 0.3. Su escala típica se determina aproximadamente como: máx $\left[(|U| + c)\Delta t \sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}}\right]$ que está a la orden del valor de número de la estabilidad CFL. Para esta aplicación usamos $C_x = C_y = 0.1$.

De acuerdo con la formulación CPML descrita en (Komatitsch and Martin, 2007), la evolución temporal variable $\Psi[F^n]$ (variable de memoria) de la derivada de la variable F en la dirección ξ, en el paso pronosticador posible se calculará de la siguiente manera:

$$\Psi_{\xi}^{n+\frac{1}{2}}[F_{i,j}^{n}] = b_x \Psi_{\xi}^{n-\frac{1}{2}}[F_{i,j}^{n}] + a_x \left[\frac{1}{\Delta \xi} \left(F_{i-1,j}^{n} - F_{i,j}^{n}\right) \right] \quad (37)$$

donde $b_x = e^{(d_x/k_x + \alpha_x)\Delta t}$ y $a_x = \frac{d_x}{k_x(d_x + k_x\alpha_x)(b_x - 1).}$

La discrerización de b_x y a_x procede de la siguiente manera:

$$b_{x,i} = e^{-(d_x/k_x + \alpha_x)\nu_i \Delta t} \text{ y } a_{x,i} = \frac{d_x}{k_x(d_x + k_x\alpha_x)}(b_{x,i} - 1) \quad (38)$$

donde el subíndice i corresponde a los i-ésima iteración dada por un esquema temporal-paso a paso pronosticador-corrector. Aquí i toma valores 1 o 2 porque

estamos en el segundo orden en tiempo y ν_i es una fracción del paso de tiempo que toma valores $\nu_1 = 0.5$ y $\nu_2 = 1.0$. Esto puede ser reformulada en órdenes superiores en el tiempo después de (Martin et al., 2010). Para más detalles de la formulación de las funciones a_x y b_x puede consultarse en (Komatitsch and Martin, 2007; Martin, Komatitsch and Gedney, 2008; Martin and Komatitsch, 2009; Zhang et al., 2009; Drossaert and Giannopoulos, 2007).

Antes de proceder con el paso corrector, es necesario decodificar la presión (p), a partir de las variables de flujo U, como sigue:

$$p_{i,j}^{n+\frac{1}{2}} = (\lambda - 1)\left[U_{4_{i,j}}^{n+\frac{1}{2}} - \frac{1}{2U_{1_{i,j}}^{n+\frac{1}{2}}}\left[\left(U_{2_{i,j}}^{n+\frac{1}{2}}\right)^2 + \left(U_{3_{i,j}}^{n+\frac{1}{2}}\right)^2\right]\right. \tag{39}$$

con el valor de p podemos completar el paso pronosticador calculando los términos de flujo $F^{n+\frac{1}{2}}$ y $G^{n+\frac{1}{2}}$ como sigue:

$$F_{1_{i,j}}^{n+\frac{1}{2}} = U_{2_{i,j}}^{n+\frac{1}{2}},\ F_{1_{i,j}}^{n+\frac{1}{2}} = \frac{\left(U_{2_{i,j}}^{n+\frac{1}{2}}\right)^2}{\left(U_{1_{i,j}}^{n+\frac{1}{2}}\right)^2} + p_{i,j}^{n+\frac{1}{2}},\ F_{3_{i,j}}^{n+\frac{1}{2}} = \frac{U_{2_{i,j}}^{n+\frac{1}{2}} U_{3_{i,j}}^{n+\frac{1}{2}}}{U_{1_{i,j}}^{n+\frac{1}{2}}},$$

$$F_{4_{i,j}}^{n+\frac{1}{2}} = \frac{U_{2_{i,j}}^{n+\frac{1}{2}}}{U_{1_{i,j}}^{n+\frac{1}{2}}}\left[U_{4_{i,j}}^{n+\frac{1}{2}} + p_{i,j}^{n+\frac{1}{2}}\right],\ G_{1_{i,j}}^{n+\frac{1}{2}} = U_{3_{i,j}}^{n+\frac{1}{2}},\ G_{2_{i,j}}^{n+\frac{1}{2}} = \frac{U_{2_{i,j}}^{n+\frac{1}{2}} U_{3_{i,j}}^{n+\frac{1}{2}}}{U_{1_{i,j}}^{n+\frac{1}{2}}},$$

$$G_{3_{i,j}}^{n+\frac{1}{2}} = \frac{\left(U_{3_{i,j}}^{n+\frac{1}{2}}\right)^2}{\left(U_{1_{i,j}}^{n+\frac{1}{2}}\right)^2} + p_{i,j}^{n+\frac{1}{2}},\ G_{4_{i,j}}^{n+\frac{1}{2}} = \frac{U_{3_{i,j}}^{n+\frac{1}{2}}}{U_{1_{i,j}}^{n+\frac{1}{2}}}\left[U_{4_{i,j}}^{n+\frac{1}{2}} + p_{i,j}^{n+\frac{1}{2}}\right].$$

Paso Corrector:

La variable de evolución en el tiempo $\Psi[F^{n+\frac{1}{2}}]$ es calculada como:

$$\Psi_{\xi}^{n+1}\left[F_{i,j}^{n+\frac{1}{2}}\right] = b_x \Psi_{\xi}^{n-1}\left[F_{i,j}^{n+\frac{1}{2}}\right] + \quad a_x\left[\frac{1}{\Delta\xi}\left(F_{i,j}^{n+\frac{1}{2}} - F_{i+1,j}^{n+\frac{1}{2}}\right)\right] \tag{40}$$

y finalmente el flujo de la variable U en el próximo tiempo $n+1$ es obtenida como:

$$U_{i,j}^{n+1} = U_{i,j}^{n} + \frac{1}{2}\left[U_{i,j}^{n+\frac{1}{2}} - U_{i,j}^{n} + \frac{1}{\kappa_{\xi_{i,j}}}\frac{F_{i,j}^{n+\frac{1}{2}} - F_{i+1,j}^{n+\frac{1}{2}}}{\Delta\xi_i}\right.$$
$$\left. + \Psi_{\xi}^{n+1}\left[F_{i,j}^{n+\frac{1}{2}}\right] + \frac{F_{i,j}^{n+\frac{1}{2}} - F_{i,j+1}^{n+\frac{1}{2}}}{\Delta\eta}\eta_{x_{i,j}} +\right]\Delta t + S_{i,j}^{n+1}. \tag{41}$$

la viscosidad artificial $S_{i,j}^{n+1}$ se añade en la etapa de corrector, como:

$$S_{i,j}^{n+1} = \frac{C_x|p_{i+1,j}^{n+\frac{1}{2}} - 2p_{i,j}^{n+\frac{1}{2}} + p_{i-1,j}^{n+\frac{1}{2}}|}{p_{i+1,j}^{n+\frac{1}{2}} + 2p_{i,j}^{n+\frac{1}{2}} + p_{i-1,j}^{n+\frac{1}{2}}} \times \left(u_{i+1,j}^{n+\frac{1}{2}} - 2u_{i,j}^{n+\frac{1}{2}} + u_{i-1,j}^{n+\frac{1}{2}}\right) + \frac{C_y|p_{i,j+1}^{n+\frac{1}{2}} - 2p_{i,j}^{n+\frac{1}{2}} + p_{i,j-1}^{n+\frac{1}{2}}|}{p_{i,j+1}^{n+\frac{1}{2}} + 2p_{i,j}^{n+\frac{1}{2}} + p_{i,j-1}^{n+\frac{1}{2}}} \times \left(u_{i,j+1}^{n+\frac{1}{2}} - 2u_{i,j}^{n+\frac{1}{2}} + u_{i,j-1}^{n+\frac{1}{2}}\right) \quad (42)$$

Para la discretización de la tensión viscosa o términos difusivos térmicas necesitamos una derivada ponderada de los componentes de la velocidad u, v y T en las direcciones ξ y η. Por razones de claridad y evitar extensas formulaciones discretizadas de las derivadas de componentes de la velocidad y la temperatura que están involucrados en los componentes de estrés y los flujos térmicos, denotamos ϕ una variable que puede tomar valores de u, v y T. Las derivadas discretizadas de ϕ junto con ξ y η se escriben como:

$$\left.\frac{\partial \phi}{\partial \xi}\right|_{i,j} = \frac{\phi_{i+1,j} + (\alpha_\xi^2 - 1)\phi_{i,j} - \alpha_\xi^2 \phi_{i-1,j}}{\alpha_\xi(\alpha_\xi + 1)\Delta\xi_i} \quad (43)$$

$$\left.\frac{\partial \phi}{\partial \eta}\right|_{i,j} = \frac{\phi_{i+1,j} + (\alpha_\eta^2 - 1)\phi_{i,j} - \alpha_\eta^2 \phi_{i-1,j}}{\alpha_\eta(\alpha_\eta + 1)\Delta\xi_i} \quad (44)$$

donde $\alpha_\xi = \frac{\Delta\xi_{i+1}}{\Delta\xi_i}$ y $\alpha_\eta = \frac{\Delta\eta_{j+1}}{\Delta\eta_j}$. Por lo tanto, podemos discretizar espacialmente los esfuerzos viscosos y los flujos de calor difusivos dadas por las ecuaciones (43) y (44) utilizando estas derivadas ponderadas y la sustitución de ϕ por los componentes de la velocidad u y v o la temperatura T en las formulaciones descompuestos de los flujos (18), (19), (20), (21) y (22).

4

DISEÑO MULTI-HILADO DE LAS APLICACIONES

En este capítulo exponemos cómo diseñamos las aplicaciones expuestas en el Capítulo 3 de forma multi-hilada utilizando `OpenMP`.

4.1 DISEÑO MULTI-HILADO USANDO OPENMP PARA EL CÁLCULO DIRECTO DE LA GRAVIMETRÍA A PARTIR DE UN ENSAMBLE DE PRISMAS

La aplicación consiste en el cálculo de la anomalía gravimétrica producida por un cuerpo de prismas rectangulares con densidad constante con respecto al grupo de puntos de observación (ver Figura 4.15). Este conjunto de puntos de observación es conocido como un ensamble de prismas, el cual no es necesariamente regular. Un conjunto de prismas irregulares puede ser configurado mientras los prismas no estén superpuestos. Debido a que el campo gravitacional cumple con el principio de superposición con respecto a los puntos de observación, si f es la respuesta calculada en el punto (x, y), entonces la respuesta observada en el punto $f(x, y)$ está dada por:

$$f(x, y) = \sum_{k=1}^{M} G(\rho_k, x, y), \tag{45}$$

donde M es el número total de prismas y ρ es la densidad del prisma.

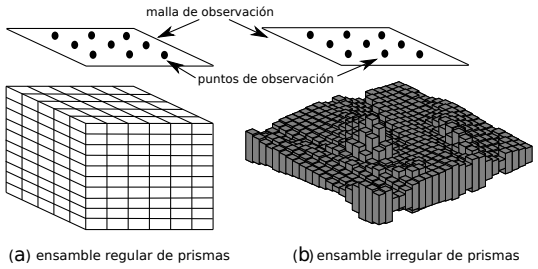

Figura 4.15: Descomposición del cálculo de M prismas con respecto a la malla de observación: (a) ensamble regular de prismas, (b) ensamble irregular de prismas.

Como sabemos que la función que calcula la anomalía para un prisma dado de un punto de observación es escrita como se muestra (Nagy et al., 2000):

$$g = f(x_l, y_l, z_l, x_r, y_r, z_r, x_p, y_p, z_p) \tag{46}$$

donde (x_l, y_l, z_l) es el vértice superior izquierdo del prisma (x_r, y_r, z_r) es el prisma inferior derecho y (x_p, y_p, z_p) es el punto de observación y ρ es la densidad del prisma, como se muestra en la Figura 4.16.

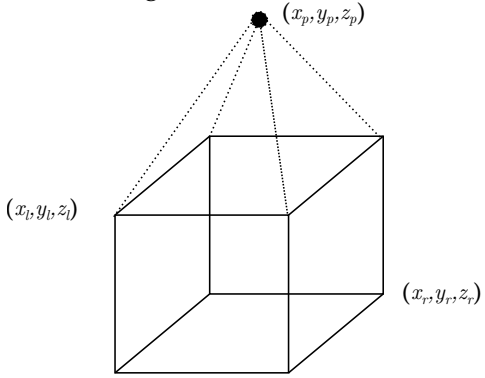

Figura 4.16: Cálculo de un prisma con respecto a un punto de observación.

Lo citado es un problema de gran escala, por ejemplo, un problema sintético conformado por un conjunto de prismas de 300 x 300 x 150 = 13,500,000 elementos, contra una malla de observación de 100 x 100 = 10,000 puntos, resulta en el cálculo de 135,000,000,000 integrales o diferenciales para resolver todo el problema.

La reducción del tiempo de cómputo en una simulación numérica es de gran importancia para disminuir los costos de investigación. Una simulación que tarda una semana es probable que sea costosa, no sólo debido al tiempo de máquina es costoso, sino que también priva de rápidos resultados para efectuar modificaciones y predicciones.

En muchos proyectos a ser paralelizados, el algoritmo serial no muestra una descomposición natural que permita fácilmente portarlo a un entorno paralelo, o la descomposición trivial no da buenos resultados de rendimiento. Por tal razón es conveniente el uso de la metodología de la programación híbrida. Esta metodología proporciona un diseño adecuado de programación para obtener un rendimiento superior.

Para desarrollar un programa paralelo es fundamental buscar la granularidad más fina, como la metodología propuesta por (Foster, 1995). En este caso es posible paralelizar por prisma o por puntos de observación. Uno de los requerimientos del diseño es que debe ser escalable, por lo tanto, el uso de sistemas híbridos es muy

apropiado; estos sistemas son los más comúnmente usados hoy en día. Siguiendo la metodología de Foster, es necesario iniciar con la más fina granularidad, en este caso corresponde a `OpenMP` porque está en el nivel más bajo.

4.1.1 *Implementación en OpenMP*

Comenzamos nuestro diseño con `OpenMP` porque maneja la memoria compartida y también es la granularidad más fina. En primer lugar particionamos el dominio en prismas, y para cada prisma paralelizamos el cálculo por puntos de observación, como se muestra en la Figura 4.17.

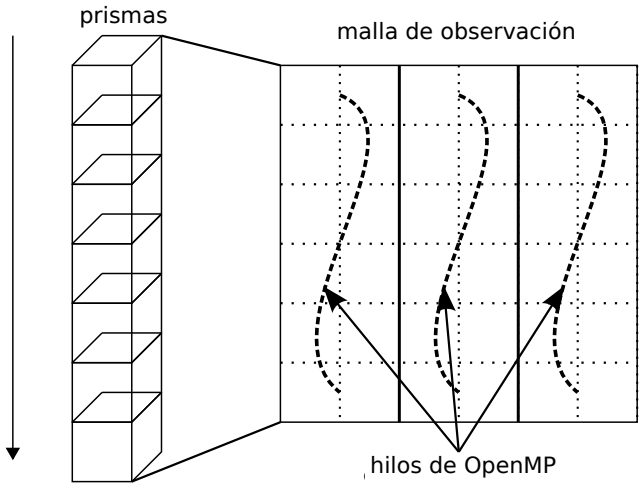

Figura 4.17: Partición por puntos de observación.

Esta paralelización por puntos de observación es trivial y no ofrece un gran desafío de diseño, ya que simplemente dividimos el cálculo con respecto a la malla de observación para cada prisma (véase el pseudo-código en el Listado 1). Sin embargo, este esquema tiene varios inconvenientes. Uno de ellos es que el rendimiento no es óptimo ya que el número de prismas es mucho mayor que el número de puntos de observación. En otras palabras, esta división es más eficiente siempre y cuando no haya demasiados hilos que trabajan en la malla de observación, evitando así un problema de cuello de botella como una consecuencia de los hilos trabajando en la misma asignación de memoria. Tal vez el peor inconveniente radica en el hecho que se crea y se cierra el entorno paralelo, es decir, para cada prisma, una función que calcula paralelamente las anomalías se ejecuta, pero tal ambiente es cerrado una vez que la ejecución ha terminado, y se vuelve a abrir para el siguiente prisma, lo que resulta en una sobrecarga innecesaria y, por lo tanto, disminuye el rendimiento.

Listado 1: Paralelización por puntos de observación.

```
For each prism from 1 t o M
!$OMP PARALLEL DO COLLAPSE( 2 )
        For each j from 1 t o Ny
                For each i from 1 t o Nx
                G( i , j ) = Gz ( parameters ) + G( i , j )
                End For
        End For
!$OMP END PARALLEL DO
End For
```

La otra opción de paralelización es utilizar los prismas, es decir, haciendo que los hilos dividan el trabajo por número de prismas (ver pseudo-código en el Listado 2). Para evitar los problemas de coherencia de la memoria caché es necesario crear un espacio de memoria diferente para cada hilo de ejecución, porque no es factible crear un solo espacio de memoria para una única malla de observación, compartido por todos los hilos. En la Figura 4.18 podemos observar las diferencias de comportamiento entre el pseudo-código del Listado 1 contra el pseudo-código del Listado 2.

Listado 2: Paralelización por puntos de observación

```
!$OMP PARALLEL DO
For each prism from 1 t o M
        For each j from 1 t o Ny
                For each i from 1 t o Nx
                G( Thread , i , j ) =Gz ( parameters ) +G( Thread , i , j )
                End For
        End For
End For
!$OMP PARALLEL DO
```

Como se observa en la Figura 4.19, es necesario crear una malla de observación por cada hilo de ejecución para evitar problemas de consistencia de memoria. Los problemas de cuello de botella de acceso a memoria se evitan, ya que cada hilo escribe en una dirección diferente de espacio de memoria. Si solo una malla fuera a ser utilizada, habría problemas de acceso a la malla compartida, lo que crearía inconsistencias numéricas.

Una de las características de `OpenMP` es que el cómputo está distribuido de manera implícita, por lo tanto el particionamiento de los M prismas que componen el problema se realiza automáticamente mediante un algoritmo de equilibrio incluido en `OpenMP`. En este caso, la decisión se deja al compilador, que es óptima en el 99 % de los casos (Zhang et al., 2004).

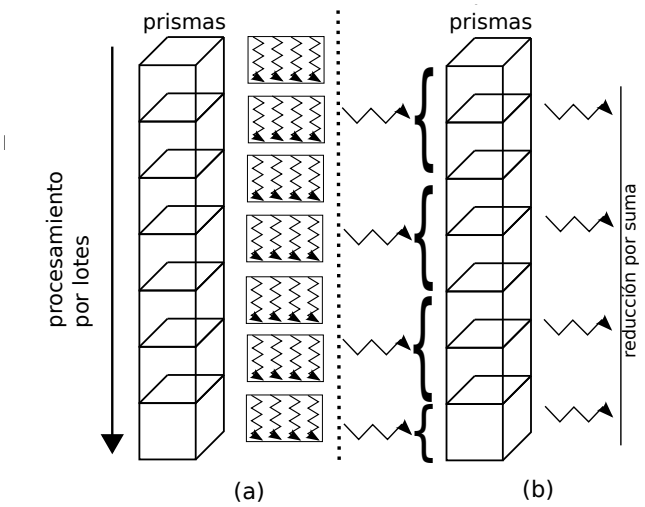

Figura 4.18: Comportamiento de la región paralela: (a) pseudo-código (1), (b) pseudo-código (2).

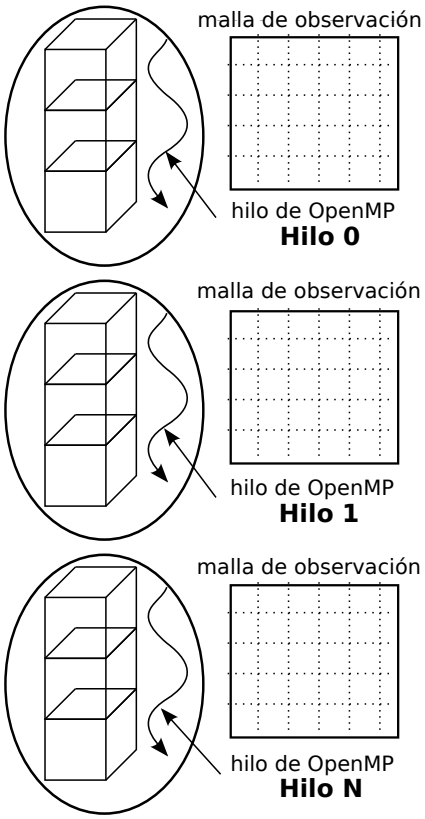

Figura 4.19: Particionamiento por prismas.

4.2 DISEÑO MULTI-HILADO USANDO OPENMP PARA LA SIMULACIÓN DE FLUJOS SUPERSÓNICOS EN EYECTORES

Como se usa un método numérico de diferencias finitas explícitas y el conjunto de ecuaciones se resuelve mediante un esquema *pronosticador-corrector*, el código es altamente paralelizable. Sin embargo, es necesario tener cuidado con la ejecución en paralelo con el fin de conseguir una reducción significativa del tiempo de cálculo.

Para utilizar `OpenMP`, el código debe incluir las directivas de compilación para generar los bucles paralelos, distribuyendo de ese modo el cálculo de forma automática (Zhang et al., 2004). Los bucles podrían ser paralelizados usando el Listado 3 o Listado 4. El cómputo se distribuye de forma implícita; Por lo tanto, la partición del bucle se efectúa automáticamente mediante un algoritmo de equilibrio. Aunque puede ser especificado por el desarrollador, en nuestra aplicación se deja la decisión al planificador.

Listado 3: El DO paralelo se implementa de manera más eficiente que una región paralela general que contiene un bucle.

```
!$OMP PARALLEL DO SHARED(...) &
!$OMP & FIRSTPRIVATE(...)
...
!$OMP END PARALLEL
```

Listado 4: Región paralela que contiene un bucle.

```
!$OMP PARALLEL SHARED(...) &
!$OMP & FIRSTPRIVATE(...)
    !$OMP DO
    ...
    !$OMP END DO
!$OMP END PARALLEL
```

Tenemos dos opciones para la paralelización de los lazos que deben tenerse en cuenta; el primero es el manejo de los bucles dentro de una región paralela (Listado 5), y el segundo es para crear una paralela DO para cada bucle (Listado 6). Ambas opciones son viables porque, incluso cuando una región paralela se cierra, los hilos permanecen activos y, cuando una nueva región en paralelo se vuelve a abrir, la sobrecarga no es significativo.

Listado 5: Apertura y cierre de la región paralela.

```
!$OMP PARALLEL DO
!$OMP END PARALLEL DO
```

```
!$OMP PARALLEL DO
!$OMP END PARALLEL DO
```

Listado 6: Región paralela persistente entre bucles.

```
!$OMP PARALLEL
    !$OMP DO
    !$OMP END DO

    !$OMP DO
    !$OMP END DO
!$OMP END PARALLEL
```

Por otro lado, la paralelización de los bucles DO/FOR en el nivel más alto posible puede conducir a un mejor desempeño, lo que requiere la paralelización del bucle más exterior y englobar múltiples bucles en la región paralela. En general, la creación de bucles dentro de las regiones paralelas reduce la sobrecarga para la paralelización evitando la creación de un DO paralelo.

Por ejemplo, el código mostrado en el Listado 3 es más eficiente que el código mostrado en el Listado 4.

Sin embargo, el constructor paralelo en `OpenMP` del código FORTRAN mostrado en el Listado 5 es menos eficiente que el código que se muestra en el Listado 6, debido a la creación de la DO paralelo. Sin embargo, aunque la sobrecarga es mínima, pero, en los códigos de diferencias finitas para flujos de fluidos, en los que son necesarios millones de iteraciones en el tiempo, una creación sobrecarga mínima no puede ser insignificante.

Implementaciones que utilizan el Listado 5(Figura 4.20 (a)) pueden encontrarse en algoritmos de alto rendimiento basados en métodos de diferencias finitas, como el algoritmo SEISMIC_CPML (Martin, Komatitsch and Ezziani, 2008; Komatitsch et al., 2010). Sin embargo, se puede obtener un mejor desempeño utilizando el esquema mostrado en el Listado 6 (Figura 4.20 (b)). En este caso, la región paralela persistente durante todo el tiempo de iteración. Además, preferimos el código por el camino del Listado 6 porque la legibilidad del código podría mejorarse. Esto es debido al comportamiento de las variables que sólo se declaran una vez al comienzo de la región paralela (SHARED, PRIVADO ...). Con el uso del esquema del Listado 6, el código contiene 16 bucles de ejecución en un tiempo de iteración y sólo se cierra y se vuelve a abrir en el comienzo del tiempo de iteración. De esta manera, se evitan la apertura y cierre de 16 regiones paralelas en cada iteración. Es importante aclarar que el compilador no puede conjuntar regiones paralelas de forma automática, ya que podría modificar la lógica del trabajo de flujo. Si la región paralela se abre y cierra continuamente, es obvio que la creación de una región paralela (creación de múltiples hilos) implicará el consumo de recursos de

cómputo, incluso si los hilos no se destruyen. El consumo de tiempo para crear una región paralela depende de las optimizaciones del compilador; por lo tanto se trata de una responsabilidad del desarrollador.

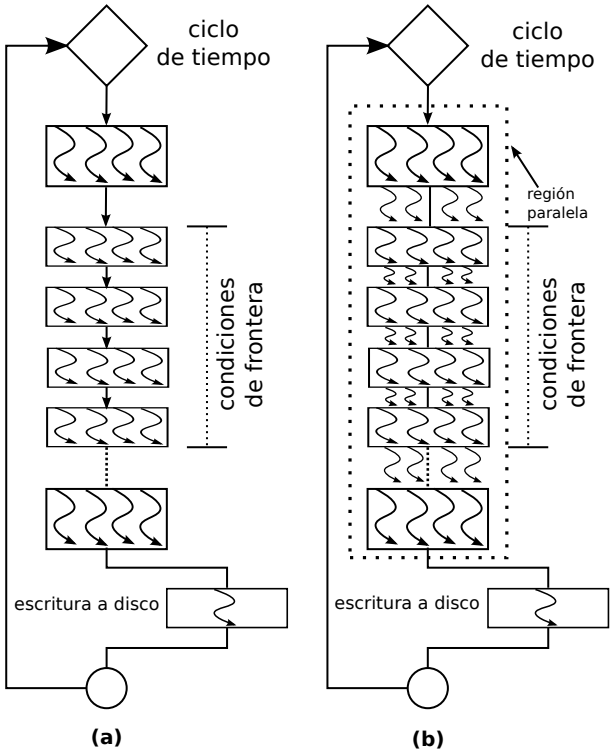

Figura 4.20: Diseño en `OpenMP`. (a) apertura y cierre de las regiones paralelas entre bucles, (b) regiones paralelas persistentes entre los bucles.

El esquema que se muestra en la Figura 4.20 (b) utiliza una sola región paralela. Este esquema fue seleccionado para desarrollar el código con las directivas `OpenMP`. Para los bucles de las condiciones de frontera, desde C_2 a C_5 y hasta C_9 a C_{12} donde las intensidades son $I < 1.0$, tenemos dos opciones. La primera es para realizar paralelización distribuyendo el bucle entre las unidades de procesamiento disponibles (núcleos), al igual que con los otros bucles con intensidades superiores a 1. Esta primera alternativa no podría ser conveniente porque puede producirse una falsa compartición; sin embargo, como los bucles son parte de un programa grande, parece ser una alternativa viable. La segunda opción, que es el preferido en el presente trabajo, es crear una tarea para un bucle (cuatro bucles para gestionar las condiciones de frontera). En nuestro caso, desde un hilo hasta cuatro pueden ser asignados para realizar las tareas. Si la tarea se considera computacionalmente no intensiva, un hilo puede realizar todas las tareas (todos los bucles), o si la tarea tarda mucho tiempo para ser completado por un hilo, otros hilos podrían ser asignados a otras tareas (ver Figura 4.21).

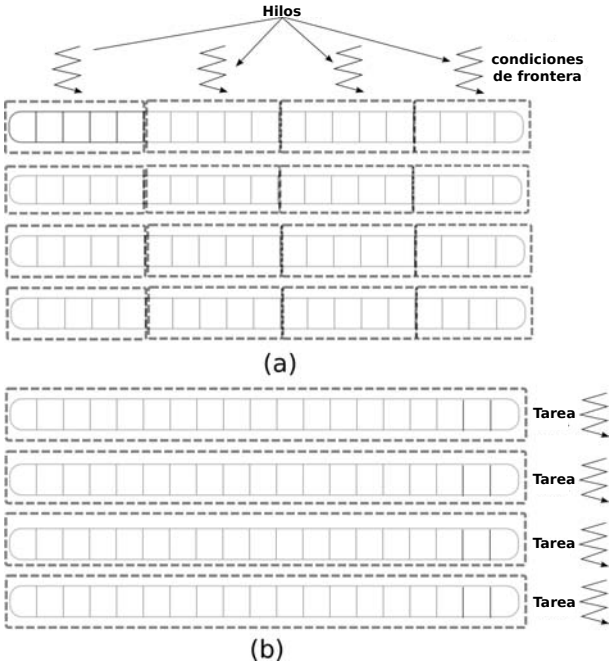

Figura 4.21: Implementación de las condiciones de frontera. (a) distribución de los bucles entre los hilos, (b) asignación de un bucle a una tarea para evitar posibles falsos intercambios debido a la intensidad de cómputo < 1.

Finalmente, es posible mantener la región paralela abierta, incluso durante el tiempo de iteración. Como se muestra en la Figura 4.22 (b), todos los hilos tienen control del tiempo de iteración, manteniendo de este modo la región paralela abierta durante todo el tiempo de simulación. Esto significa que sólo una región paralela se crea durante toda la ejecución. De esta manera, la posibilidad de sobrecarga se reduce a un mínimo. Con este último diseño, sólo una región paralela se abre durante toda la ejecución del programa y los lazos con intensidades por debajo de 1 son administrados con las tareas; sin embargo, es necesario mencionar que hay más de una posibilidades de diseño.

Otra cuestión importante a tener en cuenta es el colapso de bucles. La directiva colapso se utiliza para aumentar el número total de iteraciones que se particionan todo el número disponible de las discusiones. Mediante la reducción de la granularidad, el número de iteraciones paralelas a hacerse por lo tanto se incrementa por cada hilo. Si la cantidad de trabajo a realizar por cada hilo no es vectorizable (después se aplica el colapso), la escalabilidad paralela de la aplicación puede ser mejorada. Esta técnica se comparó contra la vectorización del bucle interno.

Cuando se contraen los bucles, el número de iteraciones distribuidos entre los hilos es $(nj-1)$ x $(ni-1)$ (ver Listado 7). Sin colapsar, el número de iteraciones

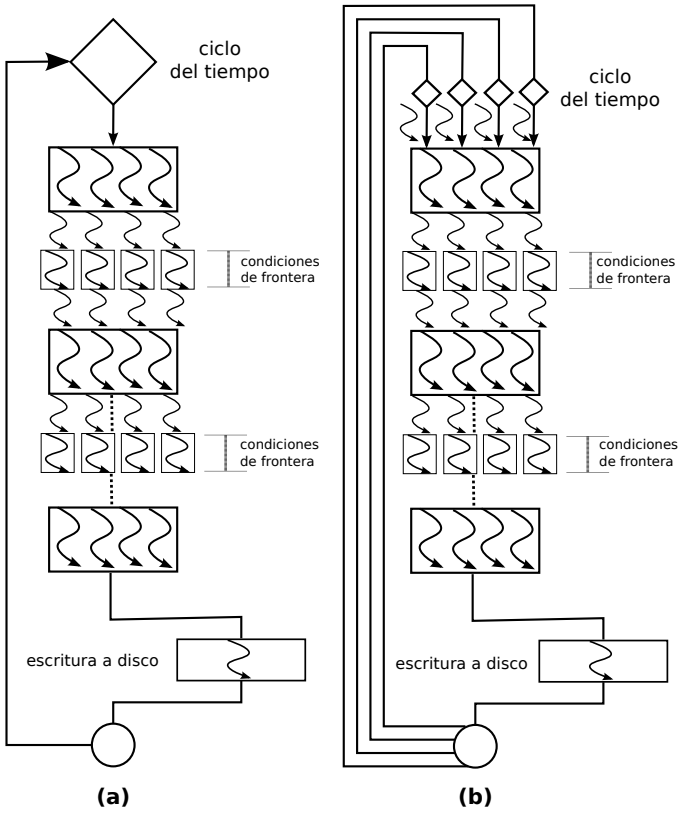

Figura 4.22: Diseño en OpenMP con el manejo de condición de frontera a través de tareas. (a) una condición de frontera es manejado por un solo hilo, (b) todos los hilos manejan el tiempo de creación de iteración de una sola región paralela para toda la ejecución del programa.

distribuidos entre los hilos es $(nj - 1)$. En el Listado 8 los lazos se derrumbaron y las operaciones están vectorizados; sin embargo, el número de operaciones de punto flotante debe ser suficiente para lograr la vectorización. Si las operaciones no son suficientes, el compilador no es capaz de generar un código vectorizado y una advertencia se muestra. En nuestro código este es el caso. Por esta razón la directiva OMP DO SIMD COLLAPSE(2) no se puede utilizar. La versión Intel Fortran Compiler 15.0 produce la advertencia # 13379: el bucle no está vectorizado con "SIMDz el compilador PGI produce el bucle de advertencia no vectorizado: puede no ser beneficioso. En el Listado 10, la directiva OMP DO SIMD COLLAPSE(2) podría ser aplicada con más éxito porque un bucle interno adicional se incluye.

Listado 7: Iteraciones de los bucles se unen para ser ejecutadas en paralelo por el equipo de hilos.

```
!$OMP DO COLLAPSE(2)
DO j=2,(nj-1)
```

```
    DO i=2,(ni-1)
    ...
    END DO
END DO
```

Listado 8: Iteraciones de los bucles se unen para ser ejecutados en paralelo y las operaciones internas están destinados a ser vectorizado.

```
!$OMP DO SIMD COLLAPSE(2)
DO j=2,(nj-1)
    DO i=2,(ni-1)
    ...
    END DO
END DO
```

Para distribuir las iteraciones del bucle exterior entre los hilos y vectorizar las operaciones (en general, el bucle interno), la directiva OMP DO SIMD (Listado 9) podría ser utilizada. Para algunos compiladores, la SIMD cláusula no es necesaria porque se aplica una vectorización automática.

Listado 9: Iteraciones del bucle exterior se distribuyen entre los hilos y las operaciones están destinadas a ser vectorizadas; como existe el bucle interior, está destinado a ser vectorizado.

```
!$OMP DO SIMD
DO j=2,(nj-1)
    DO i=2,(ni-1)
    ...
    END DO
END DO
```

Listado 10: Iteraciones de los bucles exteriores se unen para ser ejecutado en paralelo y las operaciones internas están destinadas a ser vectorizadas. Como existe el bucle interior, está destinado a ser vectorizados.

```
!$OMP DO SIMD COLLAPSE(2)
DO j=2,(nj-1)
    DO i=2,(ni-1)
        DO k=2,(nz-1)
        ...
        END DO
    END DO
END DO
```

Las posibilidades para paralelizar los bucles aplicables a nuestro código actual se muestran en los Listados 7 y 9.

5

EXPERIMENTOS NUMÉRICOS EN ARQUITECTURA INTEL

En este capítulo se dan los resultados de la experimentación utilizando las aplicaciones con y sin hiper-hilado en un Dual Intel Xeon.

Es importante la afinidad porque indica la manera en que son distribuidos los hilos de ejecución. En una arquitectura de dos sockets como es el Dual Xeon y por lo tanto es un sistema NUMA pueden existir diferencias en el rendimiento en la forma en como son distribuidos los hilos. En la Figura 5.23 se muestra la distribución de los hilos en un sistema Dual con seis núcleos por socket con el HT desactivado, en el inciso (a) de forma *compact* en el inciso (b) de forma *scatter*

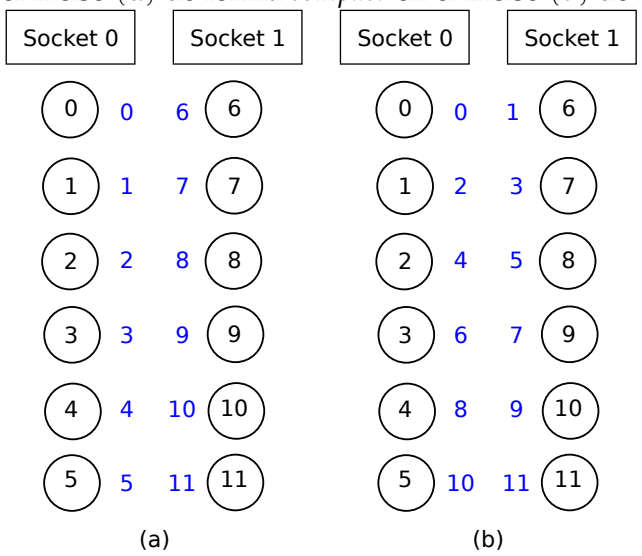

Figura 5.23: Distribución de los hilos en un sistema con el HT desactivado. (a) *compact*, (b) *scatter*

En la Figura 5.24 se muestra la distribución de los hilos en un sistema Dual con seis núcleos por socket con el HT activado, en el inciso (a) de forma *compact* en el inciso (b) de forma *scatter*. Las llaves indican que los hilos son atendidos por el mismo núcleo.

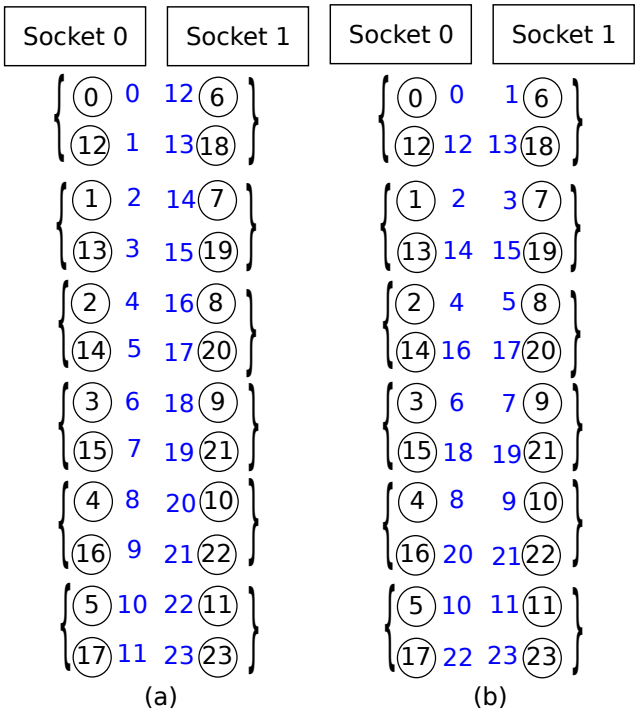

Figura 5.24: Distribución de los hilos en un sistema con el HT activado. (a) *compact*, (b) *scatter*

5.1 RESULTADOS DE LOS EXPERIMENTOS EN INTEL SIN HIPER-HILADO DEL CÁLCULO DIRECTO DE LA GRAVIMETRÍA A PARTIR DE UN ENSAMBLE DE PRISMAS

La máquina donde se llevaron a cabo las pruebas de rendimiento es una Dell Precision T7500 con las siguientes características:

- 2 Discos duro SATA de 1,5 TB, 3 Gb/s, 7.200 rpm, con DataBurst Cache de 16 MB
- CPU 1: Intel (R) Xeon(R) CPU @ 3.333 GHz Velocidad QPI: 6.400 GT/s Caché L2 del Procesador: 1536 KB Caché L3 del Procesador: 12 MB
- CPU 2: Intel (R) Xeon(R) CPU @ 3.333 Ghz Velocidad QPI: Xeon(R) CPU @ 3.333 GHz Velocidad QPI: GT/s Caché L2 del Procesador: 1536 KB Caché L3 del Procesador: 12 MB

Para llevar a cabo los experimentos se crearon de 1 a 12 hilos de ejecución, ya que la máquina solo dispone de 12 núcleos de procesamiento de forma nativa cuando el hiper-hilado se encuentra desactivado. Los resultados obtenidos se muestran en la Tabla 1 y su correspondiente gráfica en la Figura 5.25.

Tabla 1: Tiempos de cómputo obtenidos usando OpenMP sobre Xeon-CPU.
HT-Hiper hilado, D-Desactivado, CA-Afinidad Compact, SA-Afinidad Scatter

Hilos	HT-D CA	HT-D SA
1	9h 11m 18 s	9h 11m 11s
2	4h 35m 48s	4h 36m 01s
3	3h 10m 50s	3h 03m 58s
4	2h 23m 09s	2h 18m 18s
5	1h 54m 31s	1h 54m 32s
6	1h 35m 28s	1h 35m 34s
7	1h 21m 50s	1h 22m 00s
8	1h 11m 38s	1h 11m 40s
9	1h 03m 40s	1h 03m 43s
10	0h 57m 18s	0h 57m 20s
11	0h 52m 06s	0h 52m 08s
12	0h 47m 46s	0h 47m 48s

En la Figura 5.25 se grafican los tiempos obtenidos en la Tabla 1.

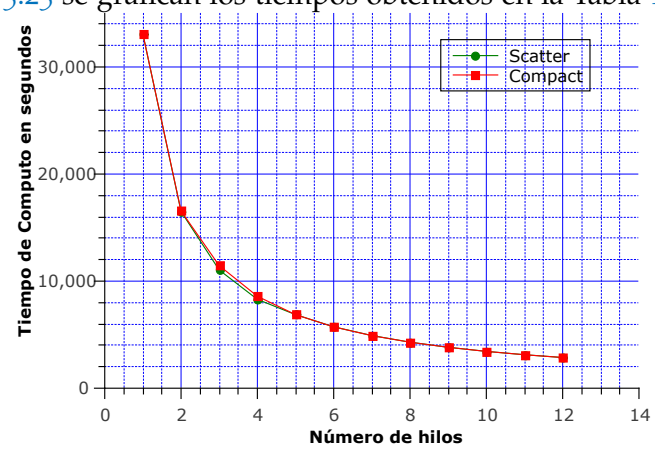

Figura 5.25: Tiempo de cómputo del modelo directo sin Hiper-hilado, se puede observar un decrecimiento exponencial del tiempo y prácticamente similar para ambos tipos de afinidades y por lo tanto no existe diferencia significativa en el tiempo.

Para poder dar una mejor perspectiva del comportamiento del tiempo de cómputo es necesario calcular el factor de velocidad comúnmente conocido como *speed-up*, el speed-up estimado se basa en la ley de Amdahl y es estimado como:

$$S(n) = \frac{n}{1 + (n-1)f} \tag{47}$$

donde n es el número de procesadores en este caso de hilos y f es la fracción serial la cual es la parte del programa que no puede ser paralelizada, para un speed-up perfecto $f = 0.0$.

En la Figura 5.26 se muestra el speed-up perfecto contra los speed-ups obtenidos para una afinidad de tipo *scatter* y *compact*.

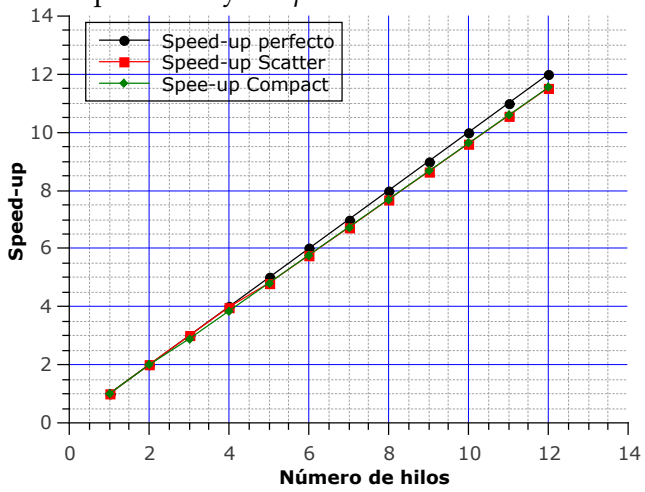

Figura 5.26: Speed-ups obtenidos del modelo directo sin hiper hilado, para las afinidades *compact* y *scatter*, se observa un speed-up cercano al perfecto para ambas afinidades lo que implica que prácticamente no existe carga en la paralelización.

De los resultados obtenidos puede verse que se obtiene un speed-up muy cercano al perfecto y que no existe diferencia significativa entre las afinidades. Comparando contra la ley de Amdahl con una fracción serial del 2 % y 6 % obtenemos la gráfica mostrada en la Figura 5.27, de cual podemos notar que la fracción serial se encuentra aproximadamente en el 2 % por lo cual se esta obteniendo un speed-up cercano al perfecto.

5.2 RESULTADOS DE LOS EXPERIMENTOS EN INTEL CON HIPER-HILADO DEL CÁLCULO DIRECTO DE LA GRAVIMETRÍA A PARTIR DE UN ENSAMBLE DE PRISMAS

Antes de proceder con los experimentos en esta sección es muy importante volver a remarcar que cuando el HT esta activado dos hilos de ejecución pueden ser atendidos por el mismo procesador, por ejemplo la pareja de hilos $(1, 12)$ son atendidos por el mismo núcleo. Los resultados obtenidos para 1 hasta 24 hilos de ejecución para ambas afinidades se muestran en la Tabla 2 y la gráfica correspondiente en 5.28.

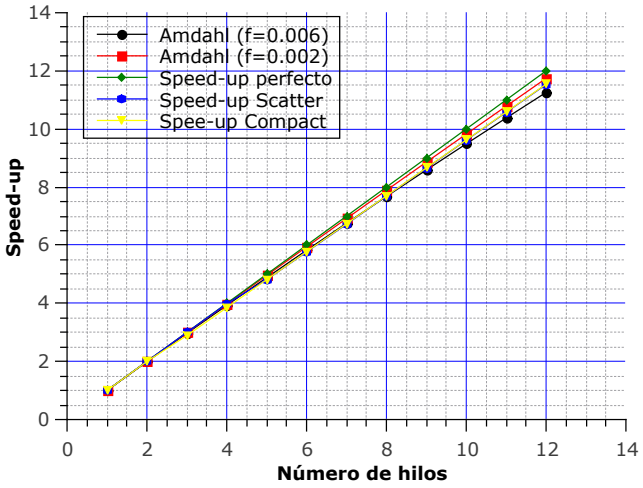

Figura 5.27: Comparación contra la ley de Amdahl del modelo directo sin Hiper-hilado, lo que se observa es un speed-up muy cercano al lineal, por lo cual existe muy poca sobrecarga en la comunicación, como lo muestra la comparación contra una fracción serial de entre 2 % y 6 %.

Tabla 2: Tiempos de cómputo obtenidos usando OpenMP sobre Xeon-CPU.
HT-Hiper hilado, A-Activado, CA-Afinidad Compact, SA-Afinidad Scatter

Hilos	HT-A CA	HT-A SA	Hilos	HT-A CA	HT-A SA
1	9h 11m 22s	9h 11m 16s	13	1h 00m 45s	1h 00m 06s
2	6h 22m 05s	4h 35m 47s	14	0h 56m 37s	0h 55m 48s
3	4h 14m 35s	3h 03m 54s	15	0h 52m 35s	0h 52m 03s
4	3h 10m 36s	2h 18m 01s	16	0h 49m 34s	0h 48m 55s
5	2h 36m 28s	1h 54m 31s	17	0h 46m 33s	0h 46m 21s
6	2h 11m 46s	1h 35m 26s	18	0h 43m 53s	0h 43m 46s
7	1h 52m 45s	1h 21m 48s	19	0h 41m 38s	0h 41m 30s
8	1h 38m 49s	1h 11m 37s	20	0h 39m 39s	0h 39m 25s
9	1h 27m 36s	1h 03m 38s	21	0h 37m 36s	0h 37m 35s
10	1h 18m 55s	0h 57m 16s	22	0h 35m 54s	0h 35m 50s
11	1h 11m 40s	0h 52m 04s	23	0h 34m 15s	0h 34m 15s
12	1h 05m 40s	0h 47m 55s	24	0h 32m 53s	0h 32m 54s

En la Figura 5.28 se muestra la gráfica correspondiente a la Tabla 2

En este caso la afinidad *scatter* hace que cada hilo utilice en un principio un núcleo físico, a diferencia de la *compact* que cubre primero un núcleo de procesamiento con los dos primeros hilos. Por tal motivo la afinidad *scatter* en un principio es mas eficiente, no obstante, cuando se cubren todos los procesadores lógicos son

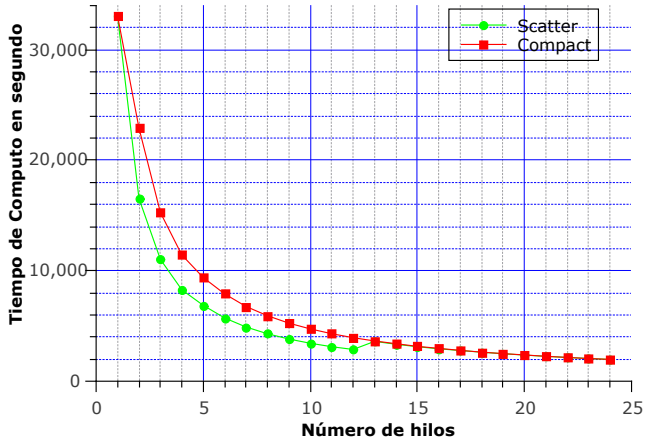

Figura 5.28: Tiempo de cómputo del modelo directo con Hiper-hilado para las afinidades *scatter* y *compacta*.

muy similares en rendimiento. De hecho el empalme sucede en el hilo 13 cuando se terminan el número de núcleos físicos, para tener una mejor perspectiva del rendimiento se muestran los speed-ups correspondientes a los tiempos de ejecución en la Figura 5.29.

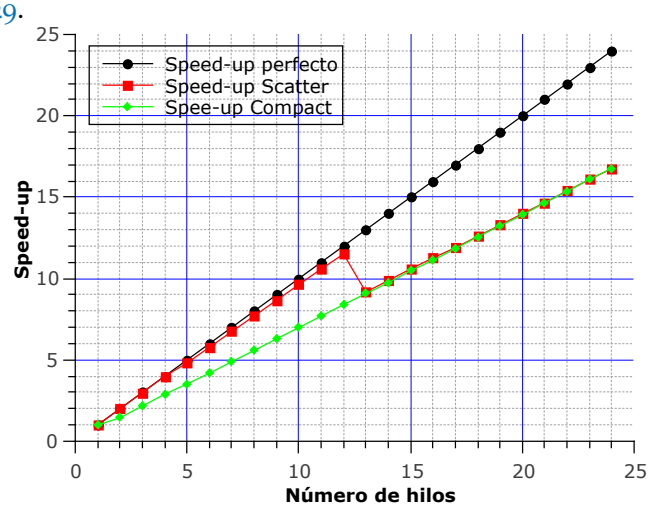

Figura 5.29: Speed-ups del modelo directo con hiper-hilado para ambas afinidades.

En la Figura 5.30 se muestra la comparación de los speed-ups obtenidos contra la ley de Amdahl con un fracción serial $f = 0.01$ y $f = 0.05$, como es de esperarse al compartir dos hilos el mismo núcleo no se obtiene un rendimiento del doble pero se obtiene una ganancia de un 45 % ya que los resultados se encuentran entre dichos valores.

Finalmente el mejor tiempo de cómputo que se obtiene con el HT activado es de 32m 53s contra 47m 48s sin el HT, lo que resulta en una mejora del 45 % con el HT activado.

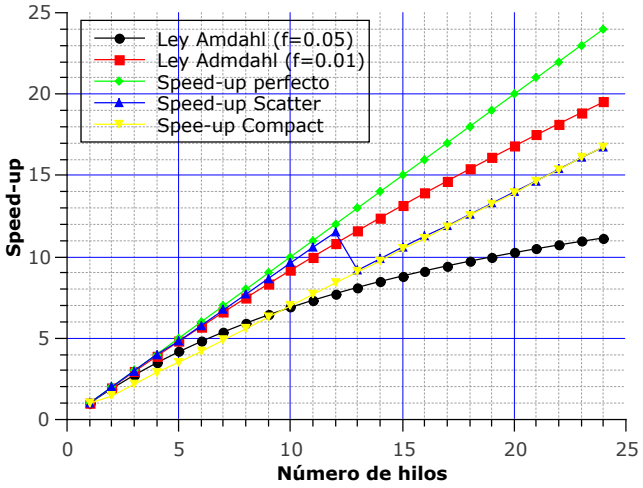

Figura 5.30: Comparación de la ley de Amdahl del modelo directo con Hiper-hilado con una fracción serial de entre el 1 % y el 5 %, la afinidad *scatter* en un principio muestra un comportamiento muy cercado al speed-up perfecto no obstante cuando se sobre pasa el número de cores físicos tiene un decremento que la empalman con la afinidad *compact* la cual crece de una forma regular debido a que utiliza primero el núcleo físico al máximo porque asigna dos hilos al mismo núcleo y ambos tipos de afinidad llegan al mismo tiempo de cómputo con 24 hilos de ejecución.

5.3 RESULTADOS DE LOS EXPERIMENTOS EN INTEL SIN HIPER-HILADO DE LA SIMULACIÓN DE FLUJOS SUPERSÓNICOS EN EYECTORES

En esta sección se evalúa el rendimiento de un código para la simulación de flujo en eyectores, a diferencia de la aplicación para el calculo directo de la gravimetría de gravedad en éste caso todos los hilos comparten el mismo espacio de direcciones en memoria, es decir, los arreglos. Los resultados para 1 a 12 hilos se muestran en la Tabla 3.

Tabla 3: Tiempos de cómputo obtenidos usando OpenMP sobre Xeon-CPU.
HT-Hiper hilado, D-Desctivado, CA-Afinidad Compact, SA-Afinidad Scatter

Hilos	HT-D CA	HT-D SA
1	44h 22m 59s	44h 33m 10s
2	22h 40m 49s	21h 30m 02s
3	15h 14m 45s	14h 35m 35s
4	11h 46m 44s	11h 10m 55s
5	09h 45m 57s	09h 13m 28s
6	08h 36m 24s	07h 56m 15s
7	07h 20m 18s	07h 00m 50s
8	06h 24m 40s	06h 09m 41s
9	05h 59m 59s	05h 55m 30s
10	05h 18m 49s	05h 13m 40s
11	05h 01m 51s	05h 09m 43s
12	04h 49m 16s	05h 00m 07s

En la Figura 5.31 se muestra la gráfica correspondiente a la Tabla 3.

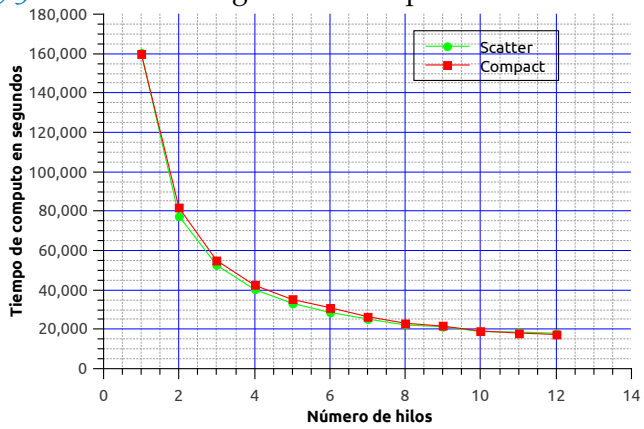

Figura 5.31: Tiempo de cómputo de la simulación de flujos supersónicos sin hiper hilado, se puede observar un decrecimiento exponencial muy similar para ambas afinidades.

En la Figura 5.32 se muestran los speed-ups obtenidos y en la Figura 5.33 su comparación contra la ley de Amdahl, lo que da una sobre carga de entre el 1 % y el 4 %.

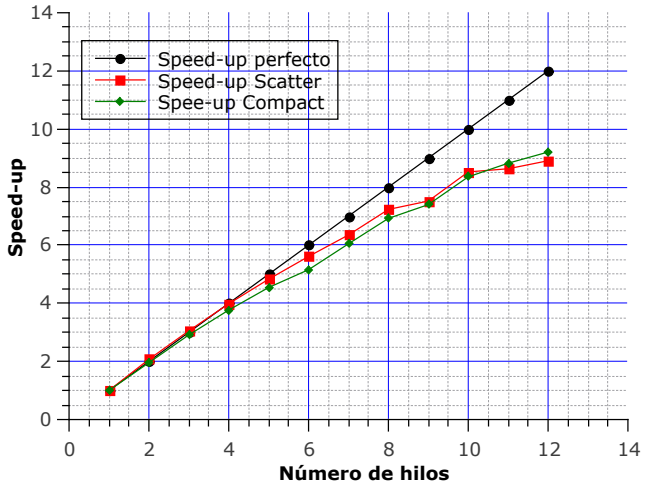

Figura 5.32: Speed-ups de la simulación de flujos supersónicos sin hiper hilado, para ambas afinidades *scatter* y *compact*. Ambas afinidades tienen un comportamiento muy similar y a partir del hilo 6 empieza a decrecer debido a que la sobrecarga influye más que en la aplicación de la sección anterior debido a que los arreglos son distribuidos entre los hilos.

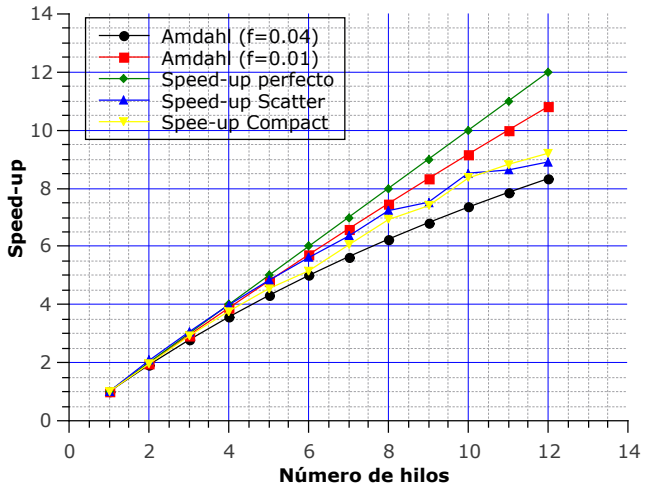

Figura 5.33: Ley de amdahl de la simulación del flujo supersónico sin hiper hilado, para ambas afinidades *scatter* y *compact*, de lo que puede concluirse que la fracción serial se encuentra entre el 1 % y 4 %.

5.4 RESULTADOS DE LOS EXPERIMENTOS EN INTEL CON HIPER-HILADO DE LA SIMULACIÓN DE FLUJOS SUPERSÓNICOS EN EYECTORES

En esta sección abordamos el problema del flujo supersónico en la tobera de un eyector, con el HT activado, por lo que es necesario crear de 1 a 24 hilos de ejecu-

ción para cubrir todos los procesadores lógicos. Los tiempos para ambos tipos de afinidades se muestran en la Tabla 4.

Tabla 4: Tiempos de cómputo obtenidos usando OpenMP sobre Xeon-CPU.
HT-Hiper hilado, A-Activado, CA-Afinidad Compact, SA-Afinidad Scatter

Hilos	HT-A CA	HT-A SA
1	44h 29m 23s	45h 02m 31s
2	31h 43m 09s	21h 26m 40s
3	21h 15m 27s	14h 36m 24s
4	16h 36m 17s	11h 09m 18s
5	13h 46m 12s	09h 13m 18s
6	12h 47m 19s	07h 57m 29s
7	11h 23m 51s	07h 02m 00s
8	10h 57m 20s	06h 11m 11s
9	10h 19m 48s	05h 58m 43s
10	10h 20m 56s	05h 17m 17s
11	09h 55m 47s	05h 10m 21s
12	09h 59m 41s	05h 00m 31s
13	09h 03m 00s	06h 45m 55s
14	08h 30m 16s	06h 35m 57s
15	08h 04m 14s	06h 05m 14s
16	07h 45m 44s	06h 00m 16s
17	07h 27m 21s	06h 34m 04s
18	07h 09m 07s	06h 31m 11s
19	06h 51m 15s	06h 27m 31s
20	06h 12m 18s	06h 01m 34s
21	05h 59m 50s	05h 49m 57s
22	05h 45m 38s	05h 53m 00s
23	05h 38m 31s	06h 20m 35s
24	05h 32m 31s	05h 51m 20s

En la Figura 5.34 se muestra los tiempos de cómputo de la Tabla 4 y los speed-ups correspondientes se muestran en la Figura 5.35 y su respectiva comparación contra la ley de Amdahl en la Figura 5.36.

La Figura 5.34 puede verse que a diferencia de la aplicación para el modelado directo de campos gravitacionales el HT introduce una sobre carga de aproximadamente el 6 % debido a la creación adicional de hilos ya que con 12 hilos y con el HT desactivado se obtiene el mejor rendimiento.

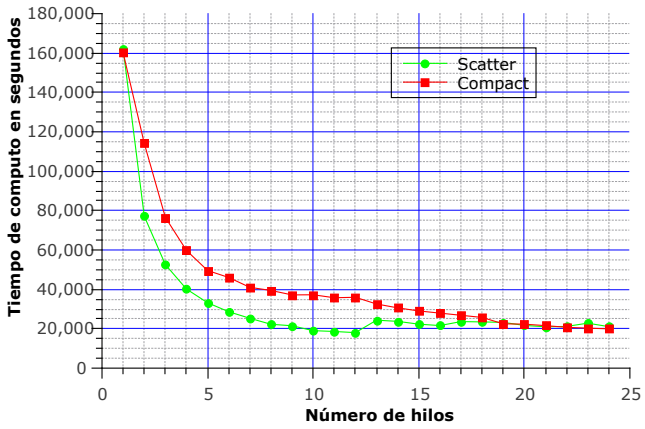

Figura 5.34: Tiempo de cómputo de la simulación de flujos supersónicos con hiper hilado con afinidad *scatter* y *compacta*. Al igual que en la sección anterior la afinidad *scatter* tiene un rendimiento mejor hasta llegar a los 12 núcleos físicos, y es de esperarse porque un hilo se asigna a un hilo a un núcleo físico, pero cuando se sobre pasa el número de núcleos físicos empieza a decrecer. La afinidad *compacta* tiene un rendimiento menor en un principio porque a cada núcleo físico le son asignados dos hilos de ejecución sacando de este modo el máximo provecho del núcleo.

Finalmente para esta aplicación el HT introduce una sobre carga de aproximadamente 6 %, por lo tanto, no beneficia al rendimiento como en la aplicación para la modelación directa de campos gravitacionales.

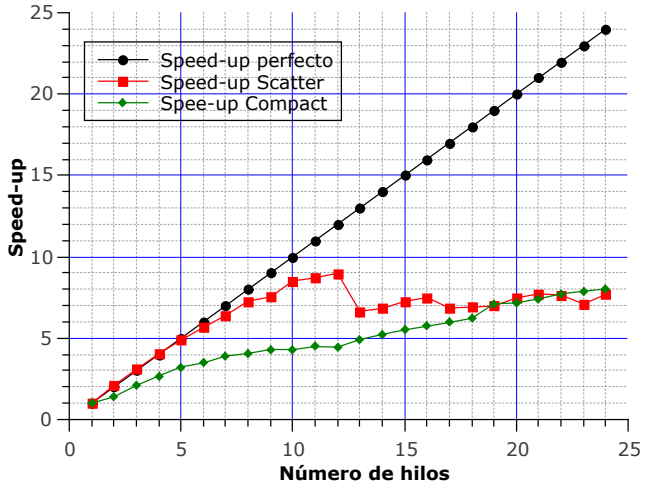

Figura 5.35: Speed-ups de la simulación de flujos supersónicos con hiper hilado, el speed-up obtenido comparado contra el speed-up perfecto que el HT en este caso esta muy lejos de proveer un mejor rendimiento, a partir del hilo 12 se puede observar un caída en el factor de velocidad.

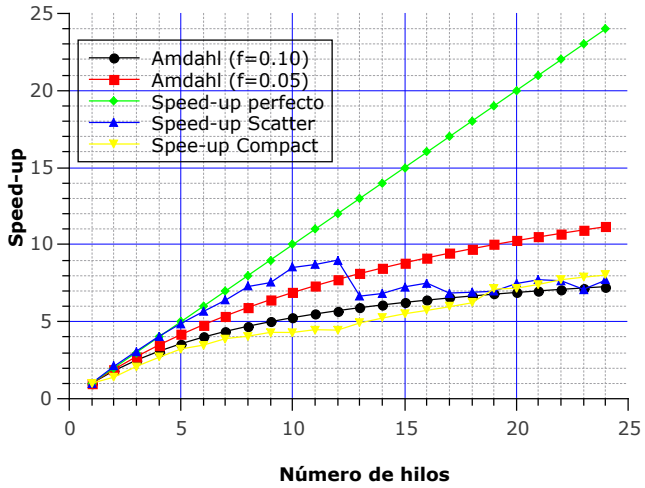

Figura 5.36: Ley de Amdahl de la simulación del flujo supersónico con hiper hilado. Al comparar contra la ley de Amdahl puede observarse que la fracción serial se encuentra entre el 5 % y el 10 %.

CONCLUSIONES Y COMENTARIOS FINALES

La tecnología de hiper-hilado se comercializa como un mejor uso de un núcleo de procesamiento, porque dos hilos pueden ser manejados de forma concurrente en el mismo núcleo prometiendo un mejor rendimiento de una aplicación multi-hilada. En varias aplicaciones informáticas esto es cierto porque no son intensivas en el uso del núcleo, como por ejemplo, un navegador web, en el cual se abren distintas páginas web, cada una en su propio hilo de ejecución, no obstante el usuario no trabaja en todas al mismo tiempo, al conmutar entre las distintas pestañas del navegador la tecnología de hiper-hilado permite retomar rápidamente el hilo de ejecución que atiende la pestaña, dando la impresión de una ejecución paralela.

En específico para las aplicaciones intensivas en cómputo analizadas en este trabajo, en base a los resultados encontrados vemos que para una aplicación en la cual todos los hilos comparten un espacio de memoria como es el caso para el eyector donde se utiliza un esquema de diferencias finitas, no existe una mejora, e incluso puede empeorar el rendimiento debido a la sobrecarga de hilos ya que para un sistema de n núcleos reales con hiper-hilado es necesario crear $2n$ hilos, y sucede que no se obtiene una mejora en el rendimiento, no obstante, cuando los hilos de ejecución trabajan sobre distintas regiones de memoria como es el caso del problema directo, podemos obtener un mayor provecho de un núcleo de procesamiento de hasta 1.45 X, o en su defecto como sucede en el caso del eyector un decremento de 1.15 X.

El Hiper-Hilado ha estado presente en los procesadores de Intel por alrededor de una década y como explotar la tecnología no es completamente claro para los programadores, por tal motivo, la gente que trabaja con cómputo de alto rendimiento usualmente reporta una nula ganancia o incluso un decremento en el rendimiento, por tal motivo en sistemas de cómputo de alto rendimiento el HT es deshabilitado, esto radica principalmente en que si tenemos una aplicación mono-hilada que se ejecuta en un sistema sin HT y posteriormente la ejecutamos puede existir una pérdida del rendimiento entre el 10 % y el 40 %.

Para poder restaurar la pérdida de rendimiento que se produce en una aplicación mono-hilada al ejecutarla con el HT habilitado, es necesaria convertirla en paralela, es decir multi-hilada, lo cual implica un trabajo adicional y ejecutarla con dos hilos y los hilos asignarlos al mismo núcleo, entonces veremos realmente si el HT produce o no un decremento en el rendimiento. Por lo cual, es recomendable entender como funciona realmente el hiper-hilado para decernir a priori si nuestro código va a ser o no beneficiado por la tecnología.

BIBLIOGRAFÍA

Almeida Francisco, Domingo Giménez, J. M. M. y. A. M. V. (2008). *Introducción a la programación paralela*, Paraninfo Cengage Learning.

Chrysos, G. (2014). Intel® xeon phi™ coprocessor-the architecture, *Intel Whitepaper* .

Couder-Castaneda, C. (2009). Simulation of supersonic flow in an ejector diffuser using the jpvm, *Journal of Applied Mathematics* .

Couder-Castaneda, C., Ortiz-Aleman, C., Orozco-Del-Castillo, M. and Nava-Flores, M. (2015). Forward modeling of gravitational fields on hybrid multithreaded cluster, *Geofisica Internacional* **54**(1): 17–26.

Drossaert, F. H. and Giannopoulos, A. (2007). A nonsplit complex frequency-shifted pml based on recursive integration for fdtd modeling of elastic waves, *Geophysics* **72**(2): T9–T17.

Komatitsch, D., Erlebacher, G., Göddeke, D. and Michéa, D. (2010). High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster, *jcp* **229**(20): 7692–7714.

Komatitsch, D. and Martin, R. (2007). An unsplit convolutional Perfectly Matched Layer improved at grazing incidence for the seismic wave equation, *geophysics* **72**(5): SM155–SM167.

Martin, R. and Komatitsch, D. (2009). An unsplit convolutional perfectly matched layer technique improved at grazing incidence for the viscoelastic wave equation, *Geophysical Journal International* **179**(1): 333–344.

Martin, R., Komatitsch, D. and Ezziani, A. (2008). An unsplit convolutional perfectly matched layer improved at grazing incidence for seismic wave equation in poroelastic media, *Geophysics* **73**(4): T51–T61.

Martin, R., Komatitsch, D. and Gedney, S. D. (2008). A variational formulation of a stabilized unsplit convolutional perfectly matched layer for the isotropic or anisotropic seismic wave equation, *cmes* **37**(3): 274–304.

Martin, R., Komatitsch, D., Gedney, S. D. and Bruthiaux, E. (2010). A high-order time and space formulation of the unsplit perfectly matched layer for the seismic wave equation using Auxiliary Differential Equations (ADE-PML), *cmes* **56**(1): 17–42.

Mehis, A. N. (2002). Hyper-threading and openmp, *Dell Power Solutions* **November**: 28–32.

Pedersen, L. and Rasmussen, T. (1990). The gradient tensor of potential field anomalies: some implications on data collection and data processing of maps, *Geophysics* **55**(12): 1558–1566.

Tian, X., Bik, A., Girkar, M., Grey, P., Saito, H. and Su, E. (2002). Intel® openmp c++/fortran compiler for hyper-threading technology: Implementation and performance., *Intel Technology Journal* **6**(1).

Tian, X., Chen, Y.-K., Girkar, M., Ge, S., Lienhart, R. and Shah, S. (2003). Exploring the use of hyper-threading technology for multimedia applications with intel® openmp compiler, *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, IEEE, pp. 8–pp.

Zhang, X., Han, L., Huang, L., Li, X. and Liu, Q. (2009). A staggered-grid high-order difference method of complex frequency-shifted pml based on recursive integration for elastic wave equation, *Chinese Journal of Geophysics (Acta Geophysica Sinica)* **52**(7): 1800–1807.

Zhang, Y., Burcea, M., Cheng, V., Ho, R. and Voss, M. (2004). An adaptive openmp loop scheduler for hyperthreaded smps, *In Proc. of PDCS-2004: International Conference on Parallel and Distributed Computing Systems*.