Métodos Paralelos para Ecuaciones Diferenciales Parciales en Plataformas Multinúcleo

por

Roberto Julián Mora Salinas

Tesis sometida como requisito parcial para

obtener el grado de

MAESTRO EN CIENCIAS EN EL ÁREA DE CIENCIAS COMPUTACIONALES

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica

Agosto 2011

Tonantzintla, Puebla.

Supervisada por:

Dr. Gustavo Rodríguez Gómez, INAOE

Dr. Saúl Eduardo Pomares Hernández, INAOE

Instituto Nacional de Astrofísica,
Óptica y Electrónica

Métodos Paralelos para Ecuaciones Diferenciales Parciales en Plataformas Multinúcleo

por
Roberto Julián Mora Salinas

Tesis sometida como requisito parcial para obtener el grado de Maestro en Ciencias en el área de Ciencias Computacionales en el Instituto Nacional de Astrofísica, Óptica y Electrónica

Supervisada por

Dr. Gustavo Rodríguez Gómez
INAOE
Dr. Saúl Eduardo Pomares Hernández
INAOE

Tonantzintla, Pue.

Agosto 2011

Resumen

Se presenta un trabajo basado en métodos paralelos de descomposición de dominio, con el objeto de proporcionar evidencia experimental del modelo de escalabilidad de plataformas multinúcleo, aplicado a la solución numérica de ecuaciones en derivadas parciales. El estudio defiende la escalabilidad de las plataformas multinúcleo mediante una implementación de la solución de un problema científico, sirviendo como evidencia de que las plataformas multinúcleo son capaces de resolver este tipo de problemas a la medida de sus capacidades y poder servir como herramientas poderosas en un futuro. El problema que sirve como base para la experimentación es la solución de una ecuación diferencial parcial del tipo advectiva-difusiva, y se resuelve mediante el Método de Descomposición de Dominio Schwarz Aditivo.

Los modelos de escalabilidad multinúcleo que se evalúan son: tamaño-fijo, tiempo-fijo, y memoria-acotada. Mediante la programación del método paralelo y técnicas de evaluación de desempeño paralelo se muestra que las plataformas de Hardware multinúcleo son capaces de resolver problemas científicos de tamaño moderado según los modelos es escalabilidad multinúcleo.

Finalmente se afirma la hipótesis de esta investigación que asegura que mediante técnicas de programación multinúcleo, las computadoras actuales, pueden brindar escalabilidad suficiente a los programas paralelos como para poder ejecutar problemas científicos de tamaño moderadamente grande, aun si en ellos existen cuellos de botella en la transferencia de datos, y se confirmó de manera exitosa.

Abstract

We present a work based on parallel domain decomposition, in order to provide experimental evidence of the model of multi-platform scalability, applied to the numerical solution of partial differential equations. The study supports the scalability of multicore platforms through implementation of a scientific problem solving, serving as evidence that multi-core platforms are capable of solving these problems to the extent of their capabilities and can serve as powerful tools in a future. The problem that serves as a basis for experimentation is the solution of a partial differential equation advective-diffusive type, and is solved by the Method of Additive Schwarz Domain Decomposition.

Multi-scale models tested are: size-fixed, fixed time-and memory-bounded. By programming the parallel method and techniques for parallel performance evaluation shows that the multi-core hardware platforms are able to solve scientific problems of moderate size depending on the model is scalable multicore.

Finally, the hypothesis says this research ensures that multicore programming techniques, today's computers, can provide sufficient scalability to programs to run parallel and scientific problems of moderately large, even if they exist bottlenecks in the data transfer, and confirmed successfully.

Agradecimientos

Al Dr. Gustavo Rodríguez Gómez, por todo el apoyo tanto personal como profesional que me ha otorgado, la paciencia y la orientación que recibí de él, así como toda la dedicación que brindó en la realización de este trabajo.

Al Dr. Saúl Eduardo Pomares Hernández, por los buenos consejos y por la orientación que me enseño una nueva visión sobre mi desarrollo profesional.

A mi esposa, por el apoyo que me brindó junto durante todo este trabajo, así como con todas las desveladas que sufrió conmigo.

A mis familiares, por el apoyo siempre presente durante este proyecto.

Al Consejo Nacional de Ciencias y Tecnología (CONACyT) por el apoyo otorgado a través de la beca No. 234550.

Dedicatorias

A mi muy amada esposa Marisol

A mis padres Roberto y Elisa, y a mi hermana Ely

Índice general

1. Introducción **1**

1.1. El problema 1

1.2. Situación Actual 2

1.3. Hipótesis 5

1.4. Objetivo 5

1.4.1. Objetivos particulares 5

1.5. Metodología 6

1.5.1. Selección de un conjunto de ecuaciones diferenciales advectiva-difusivas 6

1.5.2. Análisis de los métodos Descomposición de Dominio . . . 6

1.5.3. Diseño y Construcción del algoritmo paralelo 7

1.5.4. Evaluación 8

1.6. Organización de la tesis 9

2. Estado del Arte **11**

2.1. Métodos de Descomposición de Dominio 11

2.1.1. Métodos de Dominios con Traslape. 12

2.1.2. Métodos de Dominios sin Traslape. 13

2.2. Software para la Solución de Métodos de Descomposición de Dominio 13
2.3. Métodos de Descomposición de Dominio usando Paralelismo . . 14
2.3.1. Paquetería con Paralelismo 15
2.4. Software para Resolver EDPs con otras Técnicas 16
2.5. Resumen del Estado del Arte 16

3. Fundamentos Teóricos de Métodos de Descomposición de Dominio 19

3.1. Aplicación de Métodos de Descomposición de Dominio 20
3.2. Principios de Descomposición de Dominio 21
3.2.1. Método Schwarz Alternante 21
3.3. Métodos de Schwarz: Aditivo y Multiplicativo 22
3.3.1. Schwarz Aditivo 25
3.3.2. Schwarz Multiplicativo 27
3.4. Análisis Comparativo 29
3.4.1. Implementación 29
3.4.2. Rapidez de Convergencia 30
3.4.3. Uso de Paralelismo 30
3.5. Resumen del Capítulo 31

4. Fundamentos Teóricos de Paralelismo 33

4.1. Arquitectura de Computadoras Paralelas 34
4.1.1. Taxonomía de Flynn 34
4.1.2. Organización de Memoria 35
4.1.3. Niveles de Paralelismo en Procesadores 36
4.2. Paralelismo a Nivel de *Thread* 38

4.2.1. *Multithreading* Simultáneo 38
4.2.2. Procesadores Multinúcleo 38
4.2.3. Arquitectura de procesadores multinúcleo 39
4.3. Procesamiento Multinúcleo 39
4.3.1. Niveles de Paralelismo 40
4.4. Análisis de Desempeño para programas en Paralelo 42
4.4.1. Evaluación del desempeño de Sistemas Computacionales 42
4.4.2. Métricas de Evaluación 43
4.5. Programación de Threads 46
4.5.1. OpenMP 46
4.5.2. *Fork-Join* 48
4.6. Modelos de Escalabilidad Multinúcleo 48
4.6.1. Modelo de Tamaño-Fijo 49
4.6.2. Modelo de Tiempo-Fijo 50
4.6.3. Modelo de Memoria-Acotada 50
4.7. Resumen del Capítulo 52

5. Algoritmo para Experimento 53

5.1. Ecuación Advectiva-Difusiva 53
5.1.1. Ecuación en 1D 54
5.1.2. Ecuación en 2D 55
5.2. Consideraciones de Implementación 57
5.3. Diseño del Algoritmo 57
5.3.1. Método Schwarz Aditivo Paralelo 59
5.4. Análisis de Complejidad 60
5.5. Resumen del Capítulo 62

6. Evaluación **65**

6.1. Desempeño 66

6.1.1. Rapidez de ejecución 69

6.1.2. Eficiencia 71

6.1.3. Modelo Tamaño-Fijo 72

6.2. Escalabilidad 75

6.2.1. Modelo Tiempo-Fijo 76

6.2.2. Modelo de Memoria-Acotada 79

6.3. Discusión Sobre los Resultados 80

6.4. Resumen del Capítulo 81

7. Conclusiones **83**

7.1. Plataformas multinúcleo 83

7.2. Aportaciones 84

7.3. Trabajo futuro 85

A. Acrónimos **87**

B. Símbolos **89**

C. Condiciones de Fronteras **91**

Bibliografía **93**

Índice de tablas

2.1. Resumen de Paqueterias para Métodos de Descomposición de Dominio 17

3.1. Resumen de Características de Métodos de Descomposición . . 31

5.1. Complejidad de los Métodos de Descomposición de Dominio . . 63

6.1. Parámetros del Experimento 1 (Grupo de Control) 67

6.2. **Experimento 1:** Tiempos de Ejecución del MDD Aditivo Secuencial (segundos) 68

6.3. Parámetros del Experimento 2 70

6.4. **Experimento 2:** Tiempos de Ejecución Paralela (segundos) . . 71

6.5. **Experimento 2:** Eficiencia 73

6.6. Parámetros del Experimento 3 78

6.7. **Experimento 3:** Tiempo de Ejecución del MDD Tamaño Variable (segundos) 79

Índice de figuras

1.1. Evolución de los Procesadores de Cómputo Personal [**?**] 3

2.1. Jerarquia Simplificada de los Métodos de Descomposición de Dominio [SW90] 12

3.1. Esquema gráfico de las Fronteras Artificiales 21

4.1. Taxonomia de Flynn segun Flujos de Datos y de Instrucciones [Fly72] 35

4.2. Jerarquia Genérica de la Arquitectura de Procesadores Multinúcleo 39

4.3. Modelos para Sistemas Paralelos ordenados según Abstracción [HR92] 41

4.4. Tiempo de Respuesta Computacional 43

4.5. Representación gráfica de Fork-Join 48

4.6. Arquitectura Multinúcleo segun Niveles de Memoria 52

5.1. Comportamiento de una Ecuación Advecita-Difusiva en 1D . . . 55

5.2. Comportamiento de una Ecuación Advecita-Difusiva en 2D . . . 58

6.1. Superficie Resultado en MDD 2D. Experimento 1 69

6.2. Rapidez de ejecución en MDD Schwarz Aditivo con 8 subdominios. Experimento 2 72

6.3. Eficiencia de MDD Schwarz Aditivo con 8 subdominios 73

6.4. Rapidez de ejecución del modelo Tamaño-Fijo de la arquitectura multinúcleo 75

6.5. Rapidez de ejecución del modelo Tiempo-Fijo de la arquitectura multinúcleo 77

Capítulo 1

Introducción

Hoy en día las herramientas computacionales son ampliamente utilizadas en múltiples rubros del ambiente científico. Algunos de los usos de éstas son las simulaciones computacionales, las cuales se pueden encontrar desde en las ciencias sociales hasta biológicas, incluyéndose también en problemas de ingeniería. A esta área de aplicación se conoce como Cómputo Científico; éste se define como la colección de herramientas, técnicas y teorías requeridas para resolver en una computadora modelos matemáticos de problemas en ciencia e ingeniería [GO92]. La importancia de esta área, se hace notar en todas las posibles aplicaciones que pueden tener las simulaciones científicas, tales como: simulaciones de galaxias, comportamiento de supernovas, aerodinámica, simulación de colisiones, combustión, cambios climáticos, diseño de fármacos y muchas más.

1.1. El problema

El modelado de problemas científicos a menudo se representa mediante sistemas de ecuaciones diferenciales, usualmente no lineales, y definidas en dominios grandes. Esto quiere decir que, el conjunto de valores para los que el

sistema de ecuaciones se encuentra definido es difícil de manipular computacionalmente debido a su gran extensión. Es por esta dificultad la solución de estos sistemas de ecuaciones, demanda una gran cantidad de recursos computacionales. Estos problemas científicos pueden ser imposibles de resolver si no se cuenta con una supercomputadora a la mano, o algún equipo con gran poder de cómputo que solo en ciertos centros de investigación se pueden encontrar. Una respuesta a esta limitación tecnológica es subdividir el problema en otros más pequeños. Esto es, calcular la solución por partes, y eventualmente, la unión de las soluciones parciales será la solución global del problema [DJK07]. Esta técnica se le conoce como Método de Descomposición de Dominio.

El poder de cómputo siempre ha sido un factor decisivo al hacer investigación debido a que, como se mencionó anteriormente, acota el tamaño de los problemas que se pueden tratar. La rapidez de ejecución de los programas de computadora es dependiente, entre otras cosas, de la velocidad del procesador en el que se ejecuta. El desarrollo de los sistemas computacionales ha evolucionado de cierta manera, que las computadoras personales son capaces de tener cualidades que antes parecían imposibles, una de ellas, el paralelismo.

1.2. Situación Actual

Debido a la revolución que representan los procesadores de múltiples núcleos (multinúcleo), las computadoras personales pueden realizar procesamiento en paralelo, que en el pasado, era exclusivo de las supercomputadoras y clusters [Hug08]. El paralelismo o cómputo paralelo significa poder realizar más de una tarea a la vez, viéndose esto reflejado en menores tiempos de ejecución. La idea de los procesadores multinúcleo fue originalmente planteada desde los inicios de la década de 1990 [Hug08], pero aplicada recientemente como una solución natural al problema que representan las frecuencias de reloj muy

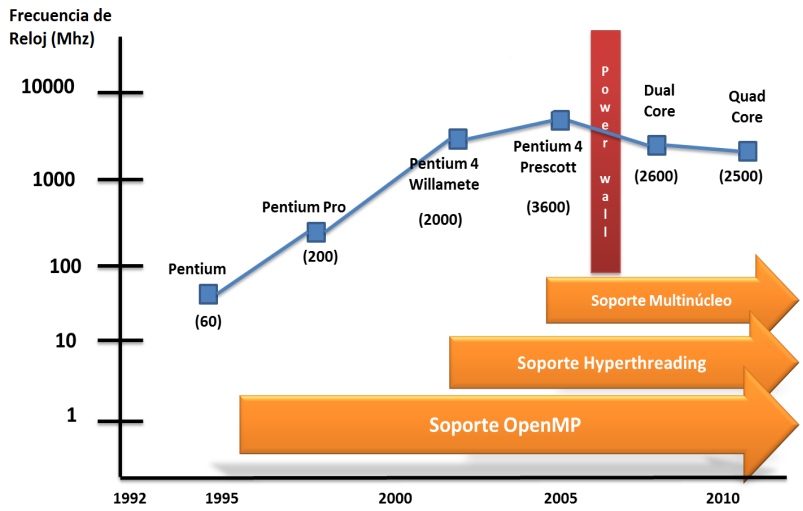


Figura 1.1: Evolución de los Procesadores de Cómputo Personal [**?**]

altas y su excesiva generación de calor. De esta manera se rompió la tendencia de fabricación, anterior a la de los procesadores multinúcleo, de mejorar el poder de cómputo a través de la construcción de procesadores, en los cuales se aumentaban simplemente la frecuencia de reloj. La figura 1.1 muestra el comportamiento anteriormente descrito y muestra el suceso conocido como ***Power Wall*** o **Barrera de Potencia** que fue el punto en el que se hizo el cambio a procesadores multinúcleo debido al excesivo calor que producían los procesadores de un solo núcleo.

Habiendo entrado los procesadores a la era del multinúcleo se presenta una incógnita, ¿qué tanto se puede escalar un sistema de cómputo? El concepto de computación escalable para este caso se refiere a la capacidad de manejar problemas de mayor tamaño mediante el aumento del número de núcleos en un procesador. Respecto a esta capacidad de escalabilidad existe un debate con ideas encontradas, en el cual existen investigadores como Hill y Marty que sostienen pesimistamente, que el futuro de procesadores multinúcleo escalables es cuestionable [HM08]. Sin embargo, también existen argumentos a favor, como Sun y Chen, con resultados más optimistas [SC10], quienes afirman

que los procesadores si poseen capacidades de escalabilidad en su artículo "*Reevaluating Amdahl's law in the multicore era*".

El artículo de Sun y Chen hace énfasis en el estudio realizado sobre la escalabilidad multinúcleo bajo las condiciones de tiempo-fijo (*fixed-time*) y memoria-acotada (*bounded-memory*), así como de la perspectiva llamada barrera de memoria (*memory wall*). Haciendo uso del modelo de costos de hardware usualmente utilizado encontraron un modelo de desempeño más optimista que los anteriormente vistos. De esta manera, animan a los lectores a las discusiones que puedan surgir en esa dirección para obtener un mejor entendimiento sobre la escalabilidad de las arquitecturas multinúcleo. Estos conceptos se explican con mayor detalle en el Capítulo 4.

Hablar de algoritmos paralelos no es un concepto simple, existen varios paradigmas y técnicas de programación que permiten ejecutar varias instrucciones a la vez. Un paradigma presente actualmente es el paralelismo a nivel de datos, que permite la ejecución de bucles en varios procesadores o núcleos por separado, haciendo su ejecución en menor tiempo que si se ejecutaran en un solo procesador. La ley de Amdahl [Amd67] proporciona las cotas teóricamente para el incremento de velocidad de un programa al ser paralelo. Ya que la mayor parte del software científico se conforma por pocos bucles con muchos cálculos, esta ley sustenta que si los bucles se paralelizan, la eficiencia del programa será alta [SL06]. También, se ha revaluado la ley se Amdahl considerando la arquitecturas multinúcleo y se ha demostrado que poseen una capacidad de escalabilidad [SC10].

Este trabajo se enfoca en trabajar los algoritmos de Descomposición de Dominio, para la solución de ecuaciones diferenciales. Se considera como la línea a seguir, la utilización de un enfoque de algoritmos paralelos para mejorar la eficiencia de la solución, mediante programación de múltiples núcleos. Haciendo esto, se podrá tener soluciones eficientes de este problema, y con esto, se ten-

drá la posibilidad de resolver problemas parecidos, de tamaño considerable, sin tener que recurrir a computadoras de gran tamaño, haciéndolo en computadoras de escritorio convencionales. Así también se comprueban las deducciones sobre la escalabilidad de las plataformas multinúcleo de manera experimental, de acuerdo al modelo de escalabilidad multinúcleo de tamaño-fijo, tiempo-fijo y memoria-acotada.

1.3. Hipótesis

La hipótesis de este trabajo es que mediante técnicas de programación multinúcleo, las computadoras actuales, pueden brindar escalabilidad suficiente a los programas paralelos como para poder ejecutar problemas científicos de tamaño moderadamente grande aún si en ellos existen cuellos de botella en la transferencia de datos. Esto para reafirmar el modelo de escalabilidad multinúcleo presentado por Sun y Chen [SC10].

1.4. Objetivo

Dar evidencia experimental, mediante métodos paralelos de descomposición de dominio, del modelo de escalabilidad de las plataformas multinúcleo propuesto por Sun y Chen, aplicado a la solución numérica de ecuaciones en derivadas parciales.

1.4.1. Objetivos particulares

- Identificar los diferentes dominios en los que aparezcan ecuaciones advectiva-difusivas para la selección de un conjunto de ellos.
- Analizar diferentes estrategias de descomposición de dominio en sus va-

riantes aditivas y multiplicativas con coincidencias de nodos.

- Diseñar e implementar el método paralelo de descomposición de dominio para aproximar la solución de la ecuación advectiva-difusiva.
- Diseñar los experimentos que permitan ratificar o rectificar las predicciones teóricas sobre la escalabilidad de las plataformas multinúcleo.

1.5. Metodología

Esta sección contiene una breve descripción de la metodología a utilizar para alcanzar el objetivo de este trabajo.

1.5.1. Selección de un conjunto de ecuaciones diferenciales advectiva-difusivas

Esta etapa consiste en la identificación de problemas existentes que se modelen mediante ecuaciones en diferencias parciales del tipo advectivo-difusivo, seleccionando un problema de modelado físico, que se represente en 1D o 2D, conociendo la solución analítica y verificando que se aplique a una región sencilla.

1.5.2. Análisis de los métodos Descomposición de Dominio

Esta etapa consiste en una recopilación de información para conocer los métodos de Descomposición de Dominio existentes, sus características y su clasificación. Así también ayudó a determinar cuál de esos métodos se utilizó para elaborar el algoritmo paralelo en el que se basó esta tesis. Esta etapa se divide en las siguientes fases:

Identificación y estudio de los distintos métodos del tipo Descomposición de Dominio

Esta fase implicó el estudio conceptual de los métodos existentes en sus variantes aditivas y multiplicativas con coincidencias de nodos, la obtención de características y tipo de problemas de los cuales proporcionan soluciones, clasificándolas por tamaño y tipo problema que resuelven.

Análisis de ventajas y desventajas de cada método

En esta fase se contempló la realización de un análisis comparativo de los métodos estudiados anteriormente, para poder facilitar la selección del método adecuado. Aquí también se identifican las características que pueden ser potenciadas mediante un enfoque de programación multinúcleo.

Selección del método a paralelizar

En esta fase se identificó de cada algoritmo qué características son beneficiosas para su implementación en programación paralela y cuáles no son deseadas puesto que complicaría la implementación. Aquí se hace un análisis de los diferentes paradigmas que se siguen para resolver un problema mediante programación paralela, con el fin de elegir un método respondiendo los siguientes cuestionamientos, ¿cómo se paralelizan los métodos de descomposición de dominio? y ¿cuál conviene hacerlo en plataformas multinúcleo?

1.5.3. Diseño y Construcción del algoritmo paralelo

En esta etapa se implementó el método de descomposición de dominio en sus versiones paralela y secuencial. Consta de las siguientes fases:

Construcción del método seleccionado en un algoritmo secuencial

En esta fase se propone el algoritmo implementación del método de Descomposición de Dominio seleccionado en la primera etapa, codificado secuencialmente.

Propuesta de algoritmo paralelo

Se propone el algoritmo paralelo implementado mediante la identificación realizada en la etapa anterior, realizándose la implementación del método de Descomposición de Dominio usando un paradigma de programación multinúcleo.

Implementación del algoritmo propuesto

El algoritmo propuesto anteriormente se implementó usando OpenMP.

1.5.4. Evaluación

En esta etapa se diseñaron las pruebas y experimentos que evaluaron el impacto del trabajo realizado sobre algoritmo propuesto contra la versión secuencial del mismo. Las métricas para poder evaluar al algoritmo son las siguientes:

1. Rapidez de ejecución.

2. Eficiencia.

3. Escalabilidad.

1.6. Organización de la tesis

En el capítulo 2, *Estado del arte*, se presenta un resumen de los trabajos relacionados con la implementación paralela de métodos de descomposición de dominio, así como desarrollos en la solución de ecuaciones diferencias parciales. En el capítulo 3, *Fundamentos Teóricos de Métodos de Descomposición de Dominio*, se presentan de manera breve los conceptos teóricos necesarios para comprender los métodos. En el capítulo 4, *Fundamentos Teóricos de Paralelismo*, se explican los conceptos necesarios de computación paralela y de alto desempeño. El capítulo 5, *Algoritmo para Experimento*, explica de manera detalla el desarrollo del diseño e implementación del método de descomposición de dominio usando técnicas de programación multinúcleo. Las pruebas se analizan en el capítulo 6, *Evaluación*, en donde se verifica el desempeño del algoritmo propuesto en distintos casos. Se finaliza el trabajo con un capitulo de *Conclusiones*, en el que se expone los principales resultados de la investigación, las limitaciones de ésta y el trabajo futuro.

Capítulo 2

Estado del Arte

En este capítulo se presentan los principales trabajos de Métodos de Descomposición de Dominio (**MDD**) que emplean estrategias de paralelismo; explicándose sus ventajas y desventajas. También, se revisa el software para la solución de Ecuaciones Diferenciales Parciales (**EDP**) en el cual se utilizan otras técnicas.

2.1. Métodos de Descomposición de Dominio

Los métodos de Descomposición de Dominio resuelven problemas representados por medio de ecuaciones diferenciales con condiciones iniciales y de frontera, dividiéndolo en problemas más pequeños (subdominios) e iterando para converger a la solución entre subdominios adyacentes. Los problemas divididos en subdominios son independientes, lo cual hace que los métodos de Descomposición de Dominio sean compatibles con la computación en paralelo. Estos métodos se pueden dividir en dos clases, la primera conformada por dominios con traslape (*overlapping domains)* y la segunda por dominios sin traslape (*nonoverlapping domains*) [DJK07]. La figura 2.1 muestra la jerarquia

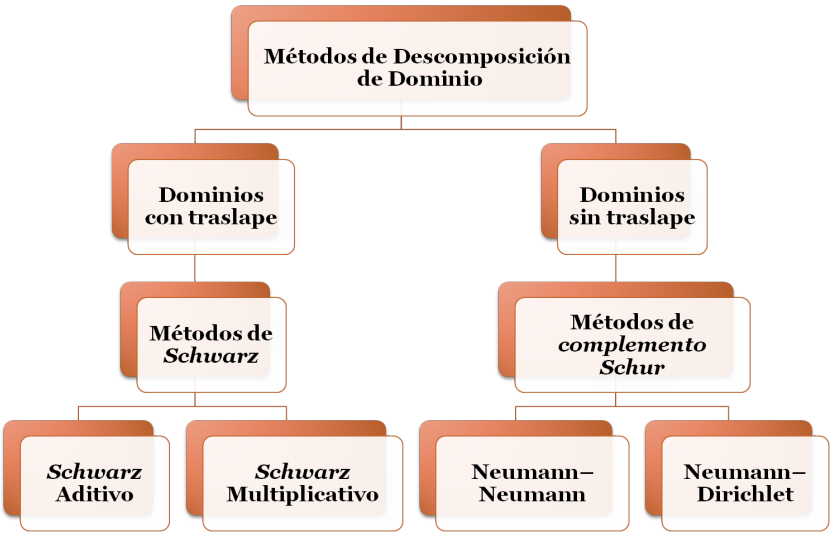


Figura 2.1: Jerarquia Simplificada de los Métodos de Descomposición de Dominio [SW90]

simplificada de los métodos existentes.

2.1.1. Métodos de Dominios con Traslape.

Los MDD con dominios traslapados son aquellos métodos en los cuales los subdominios manejan una coincidencia de los subdominios no sólo en las fronteras sino en parte interna del subdominio, esto significa, intersección en parte del subdominio. Estos métodos aparecieron inicialmente con las ideas expuestas en el artículo original de Schwarz [Sch69], dando inicio a los métodos Schwarz (también conocidas como maquinaria Schwarz o *Schwarz machinery*). Los métodos Schwarz más conocidos son: **Schwarz Alternante** desarrollado por Schwarz en 1890 [Sch90], **Schwarz Aditivo** introducido como una variante del anterior por Dryja y Widlund en 1987 [DW87] , y finalmente una variante multiplicativa **Schwarz Multiplicativo** del método aditivo y alternante [Lio88].

2.1.2. Métodos de Dominios sin Traslape.

Existen MDD en los cuales los subdominios sólo se intersectan en las fronteras; éstos son los métodos de dominios sin traslape. Este tipo de métodos reflejan la técnica clásica de subestructuración de elemento finito, en donde el sistema global de elemento finito es reducido a un sistema más pequeño pero más denso (problema de interfaz o complemento de Schur). Estos métodos fueron desarrollados inicialmente por Przemieniecki [Prz63]. Los métodos de dominios sin traslape más conocidos son: el algoritmo **Dirichlet-Neumann** [BW86], el cual resuelve un problema de valor frontera Neumann en uno de los dominios y un problema de valor frontera Dirichlet en otro (para mayor información sobre condiciones de frontera ver el Apendice C); y el algoritmo **Neumann-Neumann** [SW90] que resuelve problemas de valor frontera Neumann en todos los subdominios.

2.2. Software para la Solución de Métodos de Descomposición de Dominio

Existe trabajos realizados respecto a la solución de EDPs que usando MDD han logrado obtener excelentes resultados dentro del ámbito científico y tecnológico. Los primeros a mencionar son los paquetes de software que implementan los MDD para resolver EDPs de tipo elípticas, como **PLTMG** (***P****iecewise* ***L****inear* ***T****riangle* ***M****ulti-****G****rid*) [Ban90], que proporciona un recurso para resolver EDPs en regiones generales del plano para un solo procesador. También se encuentra ***FISHPAK*** [ASS80], que es un conjunto de subprogramas de Fortran para la solución de EDPs elípticas separables, que muestra gran eficiencia al vectorizarse y paralelizar algunas de sus rutinas [Swe91]. Por otro lado, en el software orientado a objetos se puede encontrar ***Ex-***

tensible PDE Solvers Package [Smi94], un aporte que mediante el uso de **PETSc** (***P****ortable,* ***E****xtensible* ***T****oolkit for* ***S****cientific* ***C****omputation*) [SG94], permita flexibilidad, reusabilidad y una buena eficiencia para problemas del "mundo real"usando MDD.

2.3. Métodos de Descomposición de Dominio usando Paralelismo

La solución numérica de ecuaciones diferenciales parciales, enfocadas a simulación científica, demanda una gran cantidad de recursos computacionales en términos de tiempo de procesamiento y memoria. Al modelarse problemas físicos usualmente se presentan EDPs, con las cuales se representan sistemas dinámicos (cambiantes en el tiempo). Al discretizar estas EDPs se llega a sistemas compuestos por un número muy grande de ecuaciones algebraicas, estás son las que representan la complicación de resolverse. Una solución a este problema lo ha brindado el uso de MDD en simulaciones científicas. Un ejemplo lo proporciona Aldama [AA94], quien utilizó MDD para realizar la simulación de flujo en ríos de difícil topología. Otro ejemplo lo proporcionó Bjørstad [Bjø94] en su simulación de reservas de petróleo. Algunos de los avances teóricos a favor de las simulaciones computacionales fueron realizados por Cai [CFS98], quien realizó un estudio sobre el traslape mínimo necesario para el método Schwarz en la aplicación de simulación de flujo en 3D. Recientemente, Barker realizó un método paralelo escalable para el modelado de la sangre [BC10].

Las aplicaciones de Paralelismo que se han tenido en el ámbito de MDD se han enfocado en mayor parte a cómputo distribuido. En los primeros enfoques paralelos se encuentra el de Chan en 1995 [CS95] quien analizó la complejidad paralela de los MDD mediante un estimado de los tiempos de cálculo aritmético

así como el tiempo de comunicación. Una manera de generar paralelismo en la resolución del método de Schwarz se muestra en [Qia08], donde Quiang considera la solución de sistemas de ecuaciones lineales que surgen de problemas de elemento finito parabólicos y solución de problemas de ingeniería. Muestran que tienen buena escalabilidad y calidad numérica, pero no muestran nada sobre la implementación ni las posibles limitaciones que pueden tener de acuerdo al paradigma empleado. En otros trabajos, Sarkar [SBG06], Galante [GRD07], Mukaddes [MU09]y Bahcecioglu [BOK09] muestran el uso de cómputo paralelo distribuido, en los cuales se usan estaciones de trabajo conectadas en una red para generar los recursos de cálculo paralelo. En todos estos casos se usan técnicas de descomposición de dominio adaptado al ambiente paralelo distribuido de las estaciones de trabajo conectadas en red y el de supercomputadoras. El problema de las implementaciones mencionadas anteriormente es que requieren equipos de cómputo distribuido grandes, siendo sus pruebas ejecutadas en *clusters* formados por varios nodos; esto se convierte en una limitante para cualquiera que no tenga acceso a ese tipo de equipos.

2.3.1. Paquetería con Paralelismo

Existe software que ayuda a la Solución de EDPs mediante MDD, un ejemplo existente para Fortran es **PMD**(***P****arallel* ***M****ulti-Domain* ***D****ecomposition*) [Lit98], que es un módulo para Fortran 90 que ayuda a resolver Ecuaciones elípticas lineales de segundo orden. También, se encuentra **PJDPack** (***P****arallel* ***J****acobi-****D****avidson* ***Pack****age*) [Wei09], que usando **PETSc** [SG94], encuentra algunos valores de problemas de eigenvalores polinomiales usando un método Schwarz aditivo paralelo.

2.4. Software para Resolver EDPs con otras Técnicas

La mayor parte del software para la solución de EDPs cuentan con múltiples métodos de integración incluidos, siendo que el software permite al usuario seleccionar métodos adecuados al tipo de ecuación diferencial a resolver. El software más conocido para lidiar con EDPs se conforma por dos paqueterías llamadas **LAPack** (***L****inear* ***A****lgebra* ***Pack****age*) [AfIM99] y **BLAS** (***B****asic* ***L****inear* ***A****lgebra* ***S****ubprograms*) [LHKK79]; ambas proveen rutinas para la solución de sistemas de ecuaciones lineales, eigenvalores y descomposición de valores singulares. LAPack no ofrece solución directa de EDPs, sino que ofrece solución de sistemas $Ax = b$ en todas sus variantes, cuando A es una matriz densa, que a su vez se utiliza en los métodos de solución para EDPs.

2.5. Resumen del Estado del Arte

Existen dos tipos de métodos de descomposición de dominio: con dominios con traslape y sin traslape; para ambos se han desarrollado nuevas aplicaciones prácticas para realizar simulaciones científicas avanzadas. En la tabla 2.1 se presentan las paqueterías más importantes en cuanto al desarrollo de software científico para resolver EDPs usando métodos de descomposición de dominio.

Tabla 2.1: Resumen de Paqueterias para Métodos de Descomposición de Dominio

Nombre	Lenguaje	Paralelismo	Comentarios
PLTMG [Ban90]	Fortran	No	Contiene un generador automático de mallado, contiene rutinas en lenguaje C para fácil visualización
FISHPAK [ASS80]	Fortran	No (nativamente)	Se ha demostrado su eficiencia mediante la implementación manual de paralelismo
Extensible PDE Solver Package [Smi94]	C	No	Paradigma orientado a objetos, permite encapsulamiento de datos, ser portable, y extenderse.
PMD [Lit98]	Fortran	Si	Se necesita MPI para poder implementarse, con un enfoque de cómputo distribuido
PJDPack [Wei09]	C	Si	Permite encontrar Eigenvalores
PETSc [SG94]	C y Fortran	Si	La documentación no se encuentra completa, la implementación de paralelismo es obligatorio mediante MPI
Método Paralelo	Fortran	Si	Implementación 1D y 2D de problemas Advectivos-Difusivos usando OpenMP

Capítulo 3

Fundamentos Teóricos de Métodos de Descomposición de Dominio

Los Métodos de Descomposición de Dominio (MDD) han sido desarrollados desde hace mucho tiempo, pero recientemente se han revivido mediante las Conferencias de Descomposición de Dominio, iniciándolas en París en 1987 hasta la más reciente, en su vigésima edición en San Diego 2011. Estas conferencias involucran a los mundos tanto teórico como práctico de las técnicas de Descomposición de Dominio y la creación de aplicaciones de software para computadoras masivamente paralelas.

El inicio de los MDD se remonta al año 1869, con el trabajo de H.A Schwarz sobre la existencia de funciones armónicas con fronteras complicadas [Sch69]. Continuó con los parámetros variacionales del método alternante de Schwarz por S.L. Sobolev [Sob34], convirtiéndose en la base de lo que hoy son las técnicas clásicas de Elemento Finito (EF).

En este capítulo se revisarán de forma breve los conceptos teóricos necesarios

para comprender los métodos de descomposición de dominio con dominios traslapados. Se comienza por los principios del origen de los métodos de descomposición, consecutivamente se muestra la solución para un ejemplo de la ecuación de Laplace. Posteriormente se muestra la solución del problema mediante dos aproximaciones diferentes: el método de Schwarz aditivo y Schwarz multiplicativo. Para finalizar se muestra el análisis comparativo entre métodos.

3.1. Aplicación de Métodos de Descomposición de Dominio

El modelado de problemas científicos requiere el uso de sistemas de ecuaciones diferenciales parciales, en la mayoría de los casos, no lineales y definidas en dominios que pueden tener un gran tamaño y formas complejas. El método numérico elegido para resolver el problema requerirá que se discretice el dominio, y el número de grados de libertad puede ser mayor al que el equipo de cómputo pueda manejar. Un ejemplo ocurre en el modelo del flujo de aire alrededor de una aeronave con elemento finito en 3D, éste requiere la discretización del dominio circundante con al menos unos millones de puntos; complementado a esto se asocian múltiples incógnitas a cada uno de esos puntos. El esquema numérico puede envolver un sistema lineal con un número grande de incógnitas. Este tipo de problemas (simulaciones científicas o de problemas de ingeniería) requieren el uso de equipos de cómputo muy avanzados y se puede optar por una solución más sencilla: subdividir el problema en problemas más pequeños, obtener las soluciones a esos problemas para que eventualmente la solución global sea la suma de las soluciones parciales[DJK07].

3.2. Principios de Descomposición de Dominio

Schwarz investigó la existencia de funciones armónicas en dominios Ω con fronteras complicadas $\partial\Omega$ utilizando la ecuación del Laplaciano como problema de valor de frontera [Sch69]:

Dada una función de frontera $g(x)$, encontrar una función u tal que:

$$-\Delta u(x) = 0 \quad \forall x \in \Omega \tag{3.1}$$

$$u(x) = g(x) \quad \forall x \in \partial\Omega \tag{3.2}$$

donde Δ representa al operador laplaciano. El dominio complicado $\Omega = \Omega_1 \bigcup \Omega_2$ se establece como la unión de dos subdominios más simples Ω_1 y Ω_2, con fronteras $\partial\Omega_1$ y $\partial\Omega_2$ las cuales cumplen la condición que $\partial\Omega = \partial\Omega_1 \bigcup \partial\Omega_2$, la frontera artificial Γ_1 creada al interior de Ω y que no pertenece a $\partial\Omega_1$, y a su vez la otra frontera artificial Γ_2 creada al interior de Ω y que no pertenece a $\partial\Omega_2$. La figura 3.1 muestra las fronteras artificiales de manera visual. Las fronteras que no son parte de la frontera artificial se les denota $\partial\Omega \backslash \Gamma$.

3.2.1. Método Schwarz Alternante

Schwarz propuso un proceso iterativo para resolver las ecuaciones 3.1 y 3.2, donde se resuelven problemas similares de manera alternada en Ω_1 y Ω_2. El

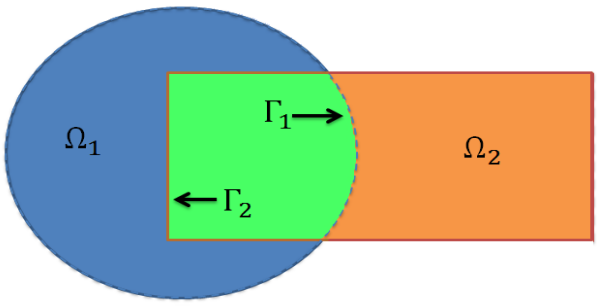


Figura 3.1: Esquema gráfico de las Fronteras Artificiales

método que Schwarz propuso se conoce como **Método Schwarz Alternante** [Sch90], que empieza por la suposición inicial, u_2^0, para los valores en Ω_2 (aunque en realidad solo se necesitan los valores de Γ_1), entonces iterativamente para $n = 1, 2, 3, ...$ se resuelve el problema de frontera:

$$\begin{aligned} Lu_1^n &= f \qquad \text{en } \Omega_1, \\ u_1^n &= g \qquad \text{en } \partial\Omega_1 \backslash \Gamma_1, \\ u_1^n &= u_2^{n-1}|_{\Gamma_1} \qquad \text{en } \Gamma_1, \end{aligned} \tag{3.3}$$

Para u_1^n. Esto seguido por la solución del problema de frontera:

$$\begin{aligned} Lu_2^n &= f \qquad \text{en } \Omega_2, \\ u_2^n &= g \qquad \text{en } \partial\Omega_2 \backslash \Gamma_2, \\ u_2^n &= u_1^n|_{\Gamma_2} \qquad \text{en } \Gamma_2. \end{aligned} \tag{3.4}$$

Si la región de traslape no esta vacía, el método converge a la solución u para el problema global presentado en las ecuaciones 3.1 y 3.2 mientras $n \to \infty$. El análisis de convergencia fue realizado por Schwarz usando el principio de máximos [Nev39].

El algoritmo 1 describe el método Schwarz Alternante para resolver un problema de valor de frontera para un problema modelo presentado en las ecuaciones 3.1 y 3.2.

3.3. Métodos de Schwarz: Aditivo y Multiplicativo

El Método Schwarz Alternante propone en general la mecánica para resolver problemas de subdominios traslapados sin importar la malla que tengan presentes. En algunas aplicaciones es posible usar mallas que encajen en la región

Algoritmo 1: Schwarz Alternante (Iterativo)

begin

$u_2^0 = g$ en Γ_1

for $n = 1, 2, 3, ...$ **do**

$Lu_1^n = f$ en Ω_1

$u_1^n = g$ en $\partial\Omega_1 \backslash \Gamma_1$

$u_1^n = u_2^{n-1}|_{\Gamma_1}$ en Γ_1

$Lu_2^n = f$ en Ω_2

$u_2^n = g$ en $\partial\Omega_2 \backslash \Gamma_2$

$u_2^n = u_1^n|_{\Gamma_2}$ en Γ_2

if $\left\|e^k\right\| \leq \epsilon$ **then**

Condición de paro

de traslape. La maquinaria de Schwarz puede ser utilizada y optimizada para tomar ventaja de este hecho y reducir el uso de memoria y uso de la aritmética de punto flotante. A esas técnicas se les conoce como Schwarz Aditivo y Schwarz Multiplicativo.

Para ilustrar los principios del método, se expondrá un ejemplo del problema de la ecuación del Laplaciano como problema de valor de frontera en 1 dimensión. El problema se representa mediante la siguiente ecuación:

$$\begin{cases} -\Delta u(x) + c(x)u(x) &= f(x) \quad para \quad x \in \Omega \subset \mathbb{R}^n \\ u = g \quad sobre \quad \partial\Omega \end{cases} \tag{3.5}$$

La solución en diferencias finitas que se obtiene se puede observar en la

siguiente expresión:

$$\begin{cases} -u''(x) + c(x)u(x) &= f(x) \quad para \quad x \in (a,b) \\ u(a) = u_a & \\ u(b) = u_b & \end{cases} \tag{3.6}$$

en donde a y b son las fontreras que delimitan el dominio a evaluar. Esta expresión conduce a un sistema lineal de tipo $Au = B$, donde se tiene que A es una matriz tridiagonal de dimensiones $n \times n$:

$$A = \frac{1}{h^2} \begin{pmatrix} 2+h^2c_1 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & 2+h^2c_2 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & 2+h^2c_3 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 2+h^2c_{n-1} & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 2+h^2c_n \end{pmatrix}, \tag{3.7}$$

siendo n el número de discretización o puntos de la malla, h es la distancia uniforme que existe de separación entre los nodos, y $c_i = c(x_i)$ y B es el siguiente vector:

$$B = \frac{1}{h^2} \begin{pmatrix} f(a+h) + \frac{u_a}{h^2} \\ f(a+2h) \\ \vdots \\ \vdots \\ f(b-h) + \frac{u_b}{h^2} \end{pmatrix}. \tag{3.8}$$

Una vez teniendo el sistema de tipo $Au = B$ se puede utilizar algún método de tipo directo o indirecto para la solución del sistema lineal y obtener la respuesta deseada al problema original. Los métodos directos dan la solución de un sistema de ecuaciones lineales mediante una secuencia finita de operaciones, aquí se podrá utilizar el método de Gauss o el de Descomposición LU [Kai80]. Mientras que los métodos indirectos o iterativo encuentran la solución a un

sistema de ecuaciones lineales, utilizando una estimación inicial para generar aproximaciones sucesivas a una solución. Entre estos últimos encontramos el Método de Jacobi y el de Gauss-Seidel [Kel95].

3.3.1. Schwarz Aditivo

El **MDD Schwarz Aditivo** es una variante del método Schwarz Alternante, y al igual que el anterior, sirve para resolver un problema de valor de frontera para EDPs por la partición en problemas más pequeños. La formulación del nombre Aditivo proviene del hecho que la solución total del dominio completo es la suma de soluciones de los subdominios [DW87].

Asumiendo que se tiene la discretización de una EDP en la malla entera, se obtiene el sistema lineal que se representa por:

$$Au = B \tag{3.9}$$

donde u es el vector cuyos coeficientes representan valores discretos de la solución aproximada en los puntos de la malla. El método Schwarz aditivo de traslape puede escribirse en formato de matrices compactas de la siguiente manera:

$$u^{n+1} \leftarrow u^n + \left(\begin{pmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{pmatrix} \right) (B - Au^n). \tag{3.10}$$

La separabilidad del problema que se observa mediante una suma (adición) de matrices en la expresión anterior se utiliza para realizar el algoritmo 2.

Por simplicidad en el Método Schwarz Aditivo se descompone el dominio computacional $[a, b]$ en dos subdominios con traslape, donde se escoge un valor impar n y dos valores enteros i_l y i_r simétricos con respecto a $\frac{n+1}{2}$ tal que $i_l < \frac{n+1}{2} < i_r$. Entonces se establece que $x_l = i_l h$ y $x_r = i_r h$, definiendo los intervalos (a, x_r) y (x_l, b) con el traslape no vacío $[a, x_r] \bigcup [x_l, b] = [x_l, x_r] \neq \emptyset$.

La solución para cada uno de los subdominios es la siguiente:

$$(P_1)\begin{cases} -u_1''(x) + c(x)u_1(x) = f(x) \quad \text{para} \quad x \in (a, x_r); \\ u_1(a) = u_a \\ u_1(x_r) = \alpha \end{cases} \tag{3.11}$$

$$(P_2)\begin{cases} -u_2''(x) + c(x)u_2(x) = f(x) \quad \text{para} \quad x \in (x_l, b); \\ u_2(x_l) = \beta \\ u_2(b) = u_b. \end{cases} \tag{3.12}$$

Ya que se tienen los problemas separados se puede aplicar el algoritmo 2, para actualizar los valores de frontera artificial. Las expresiones resultan de la siguiente manera:

$$(P_1)\begin{cases} -u_1''(x) + c(x)u_1(x) = f(x) \quad para \quad x \in (a, x_r); \\ u_1(a) = u_a \\ u_1(x_r) = u_2^{k-1}(x_r) \end{cases} \tag{3.13}$$

$$(P_2)\begin{cases} -u_2''(x) + c(x)u_2(x) = f(x) \quad para \quad x \in (x_l, b); \\ u_2(x_l) = u_1^{k-1}(x_l) \\ u_2(b) = u_b \end{cases} \tag{3.14}$$

Si la región de traslape es no vacía, el algoritmo converge a la solución u para el problema global presentado en la ecuación 3.5 mientras $k \to \infty$. El algoritmo por otra parte posee una instrucción de paro al calcular un error lo suficientemente bajo en la zona de traslape.

Algoritmo 2: Schwarz Aditivo

begin

Interpolación de fronteras artificiales

$u_1^0 = \alpha$ en Γ_1

$u_2^0 = \beta$ en Γ_2

for $k = 1, 2, 3, ...$ **do**

Se resuelve el problema para u_1^n

$Lu_1^k = f$ en Ω_1

$u_1^k = g$ en $\partial\Omega_1 \backslash \Gamma_1$

$u_1^k = u_2^{n-1}$ en Γ_1

Se resuelve el problema para u_2^n

$Lu_2^k = f$ en Ω_2

$u_2^k = g$ en $\partial\Omega_2 \backslash \Gamma_2$

$u_2^k = u_1^{n-1}$ en Γ_2

if $\left\|e^k\right\| \leq \epsilon$ **then**

Condición de paro

3.3.2. Schwarz Multiplicativo

El **MDD Schwarz Multiplicativo** es una variante del método Schwarz Alternante, y al igual que el método Alternante y el método Aditivo, sirve para resolver un problema de valor de frontera para EDPs por la partición en subproblemas.

El método Schwarz Multiplicativo de traslape puede escribirse en formato de matrices compactas de la siguiente manera: el primer paso de la iteración es

$$u^{n+1/2} \leftarrow u^n + \begin{pmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{pmatrix} (B - Au^n) \tag{3.15}$$

Similarmente el segundo paso de la iteración es:

$$u^{n+1} \leftarrow u^{n+1/2} b + \begin{pmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{pmatrix} (B - A u^{n+1/2}) \tag{3.16}$$

Se puede notar que es un método conformado por dos pasos dependientes uno del otro, en el cual, la segunda parte de la expresión (ecuación 3.16) para obtener u^{n+1} es dependiente del previo cálculo de $u^{n+1/2}$(ecuación 3.15). Este hecho de utilizar datos constantemete actualizados se refleja en una convergencia más rápida con respecto al método de carácter aditivo.

Algoritmo 3: Schwarz Multiplicativo

begin

Interpolación de fronteras artificiales

$u_1^0 = \alpha$ en Γ_1

$u_2^0 = \beta$ en Γ_2

for $k = 1, 2, 3, ...$ **do**

Se resuelve el problema para u_1^n

$Lu_1^k = f$ en Ω_1

$u_1^k = g$ en $\partial\Omega_1 \backslash \Gamma_1$

$u_1^k = u_2^{n-1}$ en Γ_1

Se resuelve el problema para u_2^n

$Lu_2^k = f$ en Ω_2

$u_2^k = g$ en $\partial\Omega_2 \backslash \Gamma_2$

$u_2^k = u_1^n$ en Γ_2

if $\left\| e^k \right\| \leq \epsilon$ **then**

Condición de paro

El Método Schwarz Multiplicativo que se obtiene a partir de las expresiones anteriores se observa en el algoritmo 3. Al aplicarse al ejemplo del Laplaciano se obtienen los siguientes problemas de valor de frontera:

$$(P_1)\begin{cases} -u_1''(x) + c(x)u_1(x) = f(x) \quad para \quad x \in (a, x_r); \\ u_1(a) = u_a \\ u_1(x_r) = u_2^{k-1}(x_r) \end{cases} \tag{3.17}$$

$$(P_2)\begin{cases} -u_2''(x) + c(x)u_2(x) = f(x) \quad para \quad x \in (x_l, b); \\ u_2(x_l) = u_1^{k}(x_l) \\ u_2(b) = u_b \end{cases} \tag{3.18}$$

De igual manera que el método anterior, si la región de traslape es no vacía, el algoritmo converge a la solución u para el problema global original mientras $k \to \infty$.

3.4. Análisis Comparativo

Las secciones anteriores han explicado los algoritmos básicos de los métodos Schwarz, ahora en esta sección hablaremos de las ventajas y desventajas que involucran. Los Métodos Schwarz brindan una oportunidad para resolver problemas con EDPs utilizando menores recursos de memoria (por el uso de matrices más pequeñas), pero incluyen un costo por la naturaleza iterativa del método. Esto último depende del tamaño y número de subdominios seleccionados, por lo cual debe seleccionarse con cuidado para asegurar competitividad del método. Se realizará una evaluación comparativa tomando los siguientes aspectos: implementación, rapidez de la convergencia y uso del paralelismo.

3.4.1. Implementación

La implementación computacional de ambos algoritmos es esencialmente la misma para un mismo problema. La diferencia recae en la actualización

de los valores de frontera; en el método multiplicativo los valores de frontera artificial al ser calculados en la solución del subdominio Ω_i deben ser utilizados para la solución de Ω_{i+1} para la misma iteración k. En el método aditivo siempre se utilizan los valores de frontera artificial calculados en la iteración anterior $k-1$ para la solución de la iteración actual k. El ordenamiento de la solución de los subdominios en el método aditivo no tiene importancia, pueden realizarse sin un orden específico. Por otro lado en el método multiplicativo se debe programar un cierto orden para poder utilizar las fronteras artificial previamente calculadas en la misma iteración.

3.4.2. Rapidez de Convergencia

Las soluciones para ambos métodos Schwarz convergen más rápido conforme el traslape sea mayor, pero a mayor traslape las soluciones de los subdominios tienden a ser más parecidas al problema completo. El número de iteraciones del método aditivo es más sensible al tamaño de traslape que la versión multiplicativo, y como regla general, el número de iteraciones de la versión multiplicativa será aproximadamente la mitad de iteraciones necesarias por el método aditivo para converger [SW90].

3.4.3. Uso de Paralelismo

Anteriormente se mencionó que el orden de los subdominios generados en la resolución de un problema mediante el método Schwarz Aditivo no tiene mayor relevancia. Esto significa que pueden separarse los subdominios en tareas diferentes y resolverse en paralelo, ya que no se tiene que compartir información hasta la siguiente iteración. El caso del método multiplicativo tiene poco potencial para la aplicación de paralelismo, pero puede realizarse mediante un algoritmo de coloreo para los subdominios generados. Para cada subdominio

se le asocia un color para que ningún subdominio que posea puntos en común tenga el mismo color. Una vez coloreados todos los subdominios se resuelven en paralelo subdominios por color.

3.5. Resumen del Capítulo

En este capítulo se pudo observar el origen del método clásico de Schwarz, y los algoritmos que representan esa metodología computacionalmente: el método aditivo y multiplicativo. Se estudiaron las ventajas y desventajas que presentan cada uno de los métodos, que es necesario comprenderlas, para que en los siguientes capítulos se aterricen estas ventajas mediante la implementación de un problema científico. La tabla 3.1 muestra un resumen de los aspectos más importantes de los MDD.

Tabla 3.1: Resumen de Características de Métodos de Descomposición

Característica	Método Aditivo	Método Multiplicativo
Sensibilidad al tamaño de traslape	Alta	Baja
Velocidad de convergencia	Lenta	Rápida
Uso del paralelismo	Directo	Mediante Algoritmo de Coloreo

El método Schwarz Aditivo cumple ciertas características que se buscaban para poder realizar la implementación:

- **La implementación paralela se hace de manera natural:** su implementación en paralelo se puede implementar de manera natural mediante *Fork-Join* que se explicará en el siguiente capítulo.
- **La convergencia es lenta a comparación de la solución tradicional:** esta característica pondrá un nivel de dificultad más alto al

momento de realizar las evaluaciones, ya que se comparará con métodos más rápidos.

- **El tamaño de traslape de los subdominios tiene importancia significativa en el método:** esta característica permitirá diseñar pruebas con parámetros lo suficientemente flexibles como para aumentar la dificultad de solución de un mismo problema.

Es debido a este conjunto de características que se seleccionó el MDD Schwarz Aditivo para realizar las pruebas de computación paralela.

Capítulo 4

Fundamentos Teóricos de Paralelismo

La simulación de problemas científicos es un área importante de ciencias e ingeniería, cuyos ámbitos y alcances están creciendo en importancia. En las últimas décadas, la investigación de alto desempeño ha dado frutos en nuevos desarrollos en tecnologías de hardware y software en paralelo [RR10]. Ejemplos populares de computación en paralelo son los diseños de autopartes mecánicas que requieren métodos basados en elementos finitos, o las simulaciones de predicciones climáticas basadas en complejos modelos matemáticos que dependen de ecuaciones diferenciales parciales. En este capítulo se presenta los fundamentos básicos de computación paralela, dando un enfoque en especial a la tendencia de programación con múltiples hilos de ejecución (*threads*). Se expondrán las arquitecturas paralelas actuales y los paradigmas actuales de programación paralela. Se abordarán los niveles de paralelismo tomando cierto énfasis en el nivel de *threads*. Se explicarán los métodos para analizar el desempeño obtenido por una implementación paralela usando algunas las métricas de evaluación. Finalmente se concluye este capítulo con los conceptos básicos de la herramienta a utilizar para paralelizar llamada OpenMP.

4.1. Arquitectura de Computadoras Paralelas

La posibilidad de ejecución en paralelo de cálculos depende fuertemente de la arquitectura de la plataforma de ejecución [RR10]. En seguida se presentan su clasificación por la taxonomía de Flynn, organización de memoria y su nivel de paralelismo.

4.1.1. Taxonomía de Flynn

En general, una computadora paralela puede ser caracterizada como una colección de elementos de procesamiento que se comunican y cooperan para resolver problemas grandes de manera rápida. Esta definición es vaga, pero permite englobar una gran variedad de plataformas paralelas. La taxonomía de Flynn [Fly72] caracteriza las computadoras paralelas de acuerdo al flujo de control, control global y los datos resultantes:

- ***Single-Instruction, Single-Data*** **(SISD):** Hay un componente de procesamiento que tiene acceso a un solo programa y almacenamiento de datos. Es el modelo convencional secuencial de computadora de acuerdo al modelo von Neumann.

- ***Multiple-Instruction, Single-Data*** **(MISD):** Existen múltiples componentes de procesamiento cada uno con su memoria privada, pero existe solo un acceso común a la memoria global. Cada elemento de procesamiento lee el mismo dato, pero realizan su instrucción privada y la aplica a él.

- ***Single-Instruction, Multiple-Data*** **(SIMD):** Existen múltiples elementos de procesamiento; cada uno tiene un acceso privado a los datos de memoria (privados o distribuidos), pero sólo existe una memoria de

programa de la cual un procesador especial de control extrae la instrucción y se la entrega a los demás. Esto es, se aplica la misma instrucción a diferentes cargas de datos.

- ***Multiple-Instruction, Multiple-Data*** **(MIMD):** Existen múltiples elementos de procesamiento, cada uno con instrucciones separadas y acceso a datos de programa y memoria de datos. Los procesadores Multinúcleo y sistemas de cluster son ejemplos de este tipo.

Se puede observar la organización de la taxonomía según el flujo de datos y de instrucciones en la figura 4.1.

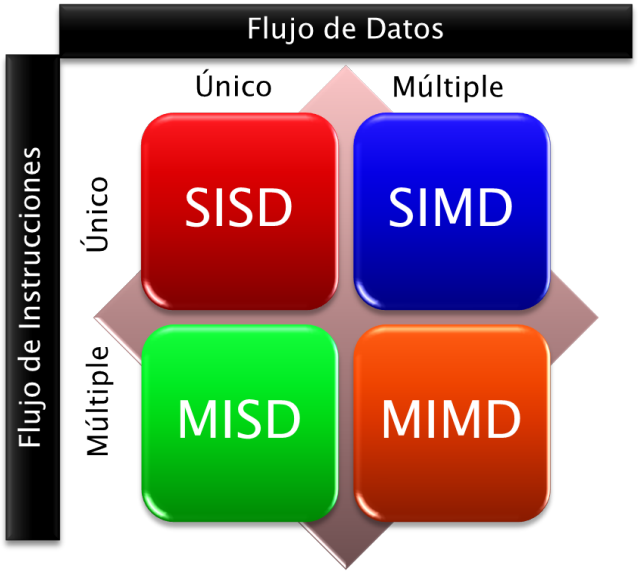


Figura 4.1: Taxonomia de Flynn segun Flujos de Datos y de Instrucciones [Fly72]

4.1.2. Organización de Memoria

Casi todas las computadoras paralelas están basadas en el modelo MIMD. Una clasificación más específica se puede hacer mediante dos rubros: organización de la memoria física y la vista que tiene el programador sobre la memoria.

Memoria Distribuida

Las computadoras con una memoria distribuida física también se conocen como ***D**istributed **M**emory **M**achines* (DMM). Cada elemento de procesamiento se le conoce como nodo, y se encuentran interconectados por medio de una red que soporta la transferencia de datos entre nodos. La memoria local de un nodo es privada, y para tener acceso a los datos de otro nodo es necesario el paso de mensajes. Técnicamente, las DMM son fáciles de ensamblar (computadoras personales). La diferencia entre un sistema DDM y un sistema *cluster* recae en el hecho de que los nodos en un *cluster* usan el mismo sistema operativo, y pueden usualmente no ser dirigidos individualmente, sino con un programador de tareas especial.

Memoria Compartida

Las computadoras con memoria física compartida se les llaman ***S**hared **M**emory **M**achines* (SMM). La memoria compartida también puede llamarse memoria global. Las SMM consisten en un número de procesadores o núcleos, una memoria física compartida, y una interconexión que conecta los procesadores con la memoria. Un modelo natural de programación para las SMM es el uso de variables compartidas que pueden ser accedidas por todos los procesados. Una variantes de las SMM son los multiprocesadores simétricos (SMP) [CSG99, HPG03], a los cuales un bus central les provee una ancho de banda constante. Esto hace que se tenga un tiempo uniforme de acceso a todas las localidades de memoria.

4.1.3. Niveles de Paralelismo en Procesadores

Los elementos claves de la arquitectura de las computadoras son los chips del procesador. Existen 4 tendencias de diseño o niveles que se pueden observar

en la fabricación de procesadores [CSG99].

1. **Paralelismo a nivel de bit:** Este nivel se condujo por las demandas en mejorar la exactitud de los números de punto flotante, y se vió como el paso incrementar de procesadores de instrucciones formadas por 4 bits hasta los 64 bits.

2. **Paralelismo por *pinelining*:** El *pipelining* es un traslape en la ejecución de múltiples instrucciones. La ejecución de cada instrucción es particionada en varios pasos, los cuales serán hechos por partes de hardware dedicado (etapas de *pipeline*). En ausencia de dependencias, todas las etapas del *pipeline* trabajan en paralelo, de tal forma que el número de particiones de la instrucción determina el grado de paralelismo que se obtiene en este nivel.

3. **Paralelismo por múltiples unidades funcionales:** Existen procesadores que son de múltiples usos, que usan unidades funcionales múltiples e independientes como ALUs (unidades lógicas aritméticas), FPUS (unidades de punto flotante) o unidades de lectura/escritura.

4. **Paralelismo a nivel de *thread*:** Una alternativa diferente a las vistas anteriormente en la que si interviene el programador y no sólo el compilador. Existe un flujo de control separado y depende mucho de las técnicas que se usen.

Las tres primeras tendencias son estrictamente dependientes de la arquitectura y fabricación de la plataforma de ejecución, que en el caso más común son los procesadores de computadora personal. La cuarta tendencia es la que otorga libertad al programador de establecer el grado de libertad a las aplicaciones realizadas, por lo que es necesario explicar a fondo el paralelismo a nivel de *thread*.

4.2. Paralelismo a Nivel de *Thread*

La organización de la arquitectura dentro de un procesador puede requerir el uso de programas explícitamente paralelos para el uso eficiente de los recursos que posee. Un ***thread*** o hilo de ejecución es una característica que permite a una plataforma o procesador realizar varias tareas a la vez. A continuación se explican dos tendencias de organizaciones de arquitectura en procesadores.

4.2.1. *Multithreading* Simultáneo

La idea del *Multithreading* Simultáneo (SMT) es usar varios threads y organizar instrucciones ejecutables de diferentes threads en un mismo ciclo si es necesario, así como usar las unidades funcionales del procesador de manera efectiva. Con 2 procesadores lógicos simultáneos se puede alcanzar hasta 30 % de mejora en un programa [MBH+02].

4.2.2. Procesadores Multinúcleo

Los procesadores multinúcleo integran múltiples núcleos de ejecución en un solo chip. Cada núcleo manejará un *thread* o múltiples si lo permite. La tendencia de fabricación de procesadores solía ser procesadores mononúcleo en la que se incrementaba la frecuencia de reloj, pero cambió a procesadores multinúcleo que aumentan en número de núcleos por 2 razones: incrementar el número de transistores en un chip incrementa la potencia requerida y la producción de calor; además que el tiempo de acceso a memoria no puede ser reducido en la misma tasa que la reducción del periodo del reloj del procesador [Koc05].

4.2.3. Arquitectura de procesadores multinúcleo

Los procesadores multinúcleo generalmente se organizan en configuraciones que tienen forma de árbol, y su tamaño de los *caches* se incrementa desde las hojas hasta la raíz (ver figura 4.2). La raíz representa la conexión hacia la memoria externa, mientras que cada núcleo tiene una memoria privada separada *cache* L1, y comparte el *cache* L2 con otros núcleos. Todos los núcleos comparten acceso a la memoria externa, resultando en una jerarquía de tres niveles. Esto puede ser extendido a más niveles, siendo que el típico uso de esta jerarquía es el de los SMP.

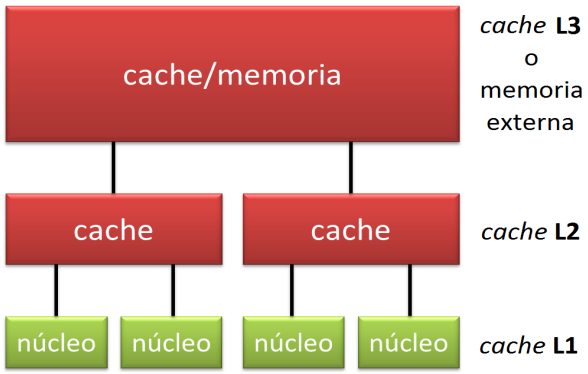


Figura 4.2: Jerarquia Genérica de la Arquitectura de Procesadores Multinúcleo

4.3. Procesamiento Multinúcleo

Para crear código eficiente es necesario tomar en cuenta el software y hardware. El gran contraste entre programación secuencial y paralela es que existen muchos detalles que considerar y diversidades en la aplicación de la programación secuencial. Esto puede resultar en varios programas diferentes para un mismo algoritmo. Existen 4 modelos de procesamiento paralelo que varían según su nivel de abstracción [HR92]:

1. **El modelo máquina:** el nivel más bajo de abstracción, consiste en la descripción del hardware y el sistema operativo.

2. **El modelo de Arquitectura:** es el siguiente nivel de abstracción. Las propiedades descritas en este nivel incluyen las interconexiones de las plataformas paralelas, organización de memoria, procesamiento síncrono o asíncrono, y modo de ejecución de instrucciones sencillas por SIMD o MIMD.

3. **El modelo computacional:** ofrece un método analítico para diseñar y evaluar algoritmos. La complejidad de un algoritmo debe reflejar el desempeño sobre una computadora real. En el caso de las computadoras secuenciales se usa la RAM (Random Access Machine) para el modelo de arquitectura von Neumann. Para el caso de computadoras paralelas se usa el modelo PRAM (***P****arallel* ***R****andom* ***A****ccess* ***M****achine*)

4. **El modelo de Programación:** Es el más alto nivel de abstracción, describe el sistema computacional paralelo en términos de semántica del lenguaje de programación o ambiente de programación. Esta visión es influenciada por la arquitectura, por el diseño de la arquitectura, el lenguaje, el compilador y las librerías de ejecución.

4.3.1. Niveles de Paralelismo

Los cálculos realizados por cierto programa proveen una cierta oportunidad para que se realice la ejecución en paralelo en diferentes niveles de paralelismo [DJK07]:

1. **Paralelismo en el nivel de instrucción:** Varias instrucciones de un programa pueden ser ejecutadas en paralelo al mismo tiempo si son independientes entre ellas.

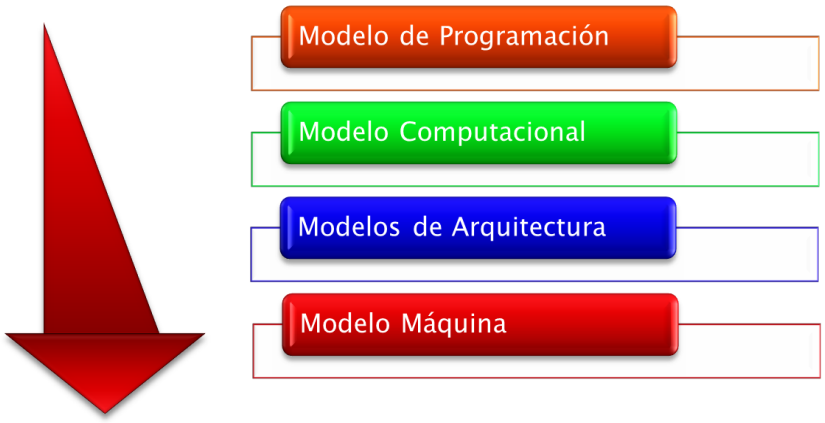


Figura 4.3: Modelos para Sistemas Paralelos ordenados según Abstracción [HR92]

2. **Paralelismo de datos:** En algunos programas, la misma operación puede ser aplicada a diferentes elementos de una estructura de datos grande. Los elementos de la estructura de datos son distribuidos uniformemente a través de los procesadores y cada procesador desempeña la operación en el elemento asignado. También, se le conoce como paralelismo a nivel de declaración.

3. **Paralelismo de bucles:** Algunos algoritmos realizan cálculos iterativamente a través de una estructura de datos grande. Usualmente los bucles son ejecutados secuencialmente, lo cual significa que los cálculos ubicados en la i-ésima iteración se empiezan a ejecutar una vez que se han completado los cálculos de la $(i-1)$-ésima. Si no existe dependencia entre las iteraciones, éstas pueden ser ejecutadas en orden arbitrario.

4. **Paralelismo funcional:** Cuando los programas contienen segmentos de código independientes y ellos pueden ser ejecutados en paralelo se conoce como paralelismo de tarea o paralelismo funcional.

4.4. Análisis de Desempeño para programas en Paralelo

La más importante motivación para usar un sistema paralelo es la reducción en el tiempo de ejecución en aplicaciones de computación intensiva. El tiempo de ejecución depende de muchos factores, incluyendo la arquitectura, el compilador y el sistema operativo de la plataforma de ejecución.

4.4.1. Evaluación del desempeño de Sistemas Computacionales

El usuario de un sistema computacional está interesado(a) en un **tiempo de respuesta** pequeño, es decir, el tiempo entre el inicio y la terminación del programa. Este tiempo se puede dividir en 3 partes para la ejecución de un programa A: el tiempo de usuario CPU (***user CPU time***), que captura el tiempo que el CPU invierte en la ejecución de A, el tiempo de sistema CPU (***system CPU time***) que captura el tiempo que el CPU gasta en la ejecución de rutinas del sistema operativo generados por A y finalmente el tiempo de espera (***waiting time***) que es el tiempo causado por la finalización de operaciones de E/S y por la ejecución de otros programas ejecutándose a la par (ver figura 4.4). Estos tiempos de CPU son dependientes de la traducción de las asignaciones a instrucciones por el compilador.

En el caso de la solución de problemas un menor tiempo de respuesta indica un mayor desempeño y viceversa. Por otro lado, un centro grande de cómputo está interesado en **altos rendimientos**, donde el rendimiento es el número promedio de unidades de trabajo que se puede ejecutar por unidad de tiempo.

Un término práctico para medir el desempeño en un sistema es mediante **MIPS**: (*Million Instruction Per Second*), que se representa por:

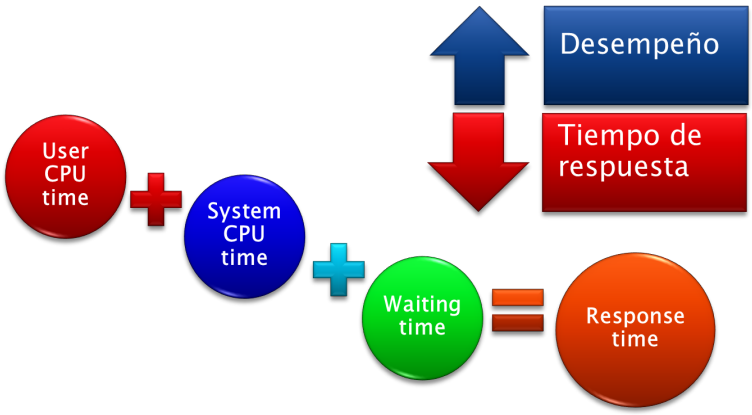


Figura 4.4: Tiempo de Respuesta Computacional. Se representa como la suma del tiempo de Usuario, Sistema y Tiempos de Espera mientras que el un mejor desempeño significa menores Tiempos de Respuesta

$$MIPS(A) = \frac{n_{instr}(A)}{T_{U\_CPU}(A) * 10^6} \tag{4.1}$$

donde A es un algoritmo, $n_{instr}(A)$ es el número de instrucciones del programa A y $T_{U\_CPU}(A)$ es el tiempo de CPU de Usuario; y **CPI** (*Clock cycles Per Instruction*) que es el promedio de ciclos que realiza en un programa determinado A para ejecutar una instrucción.

4.4.2. Métricas de Evaluación

Las métricas de desempeño para evaluar a un algoritmo y su implementación paralela de manera experimental y teorica que se seleccionaron son las siguientes:

Rapidez de Ejecución

Muestra la aceleración que se obtuvo al hacer paralelo un algoritmo. La rapidez de ejecución se mide a través del *Speedup* que se se define como:

$$S_p(n) = \frac{T^*(n)}{T_p(n)} \tag{4.2}$$

siendo n el tamaño de problema, $S_p(n)$ mide el factor de aceleración obtenido sobre un algoritmo usando p núcleos. Se denota la complejidad secuencial del problema mediante $T^*(n)$, aunque es común remplazarse por la cota del mejor algoritmo conocido que resuelva el problema. $T_p(n)$ representa el tiempo computacional paralelo que tarda en resolverse el problema con p núcleos.

Eficiencia

Provee un indicador de la utilización efectiva de los núcleos en relación al algoritmo dado, es decir, que tanto trabajo útil realzan los núcleos en el algoritmo paralelo. La eficiencia se define como:

$$E_p(n) = \frac{T_1(n)}{pT_p(n)} \tag{4.3}$$

donde un resultado para $E_p(n)$ que se aproxime a 1 para alguna p indica que el algoritmo corre aproximadamente p veces más rápido que usando un solo núcleo. $T_1(n)$ representa el tiempo computacional paralelo que tarda en resolverse el problema con 1 núcleo.

Ley de Amdahl

El tiempo de ejecución paralela de los programas no puede ser arbitrariamente reducido por la aplicación de recursos paralelos. Una importante restricción del tiempo de ejecución son las partes del programa que son inherentemente secuenciales (no se pueden paralelizar de ningún modo), esto como consecuencia trunca la rapidez obtenida como se puede ver a continuación:

$$S_p(n) = \frac{T^*(n)}{(1-f)\cdot T^*(n) + \frac{f}{p}T^*(n)} = \frac{1}{(1-f) + \frac{f}{p}} \tag{4.4}$$

donde f es la fracción de programa que debe ser ejecutada totalmente paralela, y $1-f$ la que debe ejecutarse de manera secuencial.

Escalabilidad

Este término se refiere al estudio del cambio en las medidas de rendimiento de un sistema cuando el número de núcleos es cambiado. Representa la medida de la capacidad del programa para disminuir el tiempo de ejecución con un número creciente de núcleos. La escalabilidad se describe como el cambio de la eficiencia conforme se aumenta el número de núcleos. Para esta situación se utiliza la medida de eficiencia a través de la siguiente expresión:

$$E(n) = \frac{S_p(n)}{p(n)} \tag{4.5}$$

donde S_p es la rapidez de ejecución paralela, y p es el número de núcleos trabajando en paralelo. La meta de la escalabilidad paralela que se pretende alcanzar es que la rapidez de ejecución obtenida sea directamente proporcional al número de procesadores empleados. La constante de proporcionalidad de $E(n)$ debe estar en un rango de 0.6 a 0.9 para poder considerar escalable.

Ley de Gustafson

El incremento en la rapidez para problemas cuyo tamaño n está creciendo no se puede apreciar en la ley de Amdahl. Para ello se requiere una variante de la ley de Amdahl en la cual se supone que la parte secuencial del programa no es una fracción constante, sino que decrece con el tamaño de entrada. Este comportamiento lo representa la ley de Gustafson [Gus88] para el caso especial en el que la parte secuencial del programa tiene un tiempo de ejecución constante, independiente del tamaño de problema.

Si τ_f es un tiempo de ejecución constante de la parte secuencial del programa y $\tau_v(n, p)$ es el tiempo de ejecución de la parte paralela del problema con tamaño n y un número de procesadores p, el rapidez escalada se representa por:

$$S_p(n) = \frac{\tau_f + \tau_v(n, 1)}{\tau_f + \tau_v(n, 1)}. \tag{4.6}$$

Si se supone que el programa paralelo es perfectamente paralelizable, entonces $\tau_v(n,1) = T^*(1) - \tau_f$y $\tau_v(n,p) = (T^*(n) - \tau_f)/p$ se ve lo siguiente:

$$S_p = \frac{\tau_f + T^*(n) - \tau_f}{\tau_f + (T^*(n) - \tau_f)/p} = \frac{\frac{\tau_f}{T^*(n)-\tau_f} + 1}{\frac{\tau_f}{T^*(n)-\tau_f} + \frac{1}{p}}, \tag{4.7}$$

al desarrollar las expresión anterior se obtiene la expresión:

$$\lim_{n\to\infty} S_p(n) = p. \tag{4.8}$$

Este modelo de escalabilidad intenta capturar como un tamaño de problema n debe ser incrementado en relación al número de procesadores p para obtener eficiencia constante.

4.5. Programación de Threads

Existen múltiples plataformas paralelas, entre las plataformas explicadas en las secciones anteriores existen las que manejan un espacio de memoria compartida, y en particular una de ellas son las plataformas multinúcleo. La manera natural de programar para esas arquitecturas es el modelo de hilos de ejecución o *threads*, en el cual todos los *threads* tienen acceso a las variables compartidas. Para coordinar el acceso a las variables distribuidas se utilizan mecanismos de sincronización para evitar inconsistencias en accesos concurrentes.

4.5.1. OpenMP

OpenMP es una Interfaz de Programación de Aplicaciones o **API** (***A**pplication **P**rogramming **I**nterface*) OpenMP para la programación de aplicaciones de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución ***fork-join*** (ver sección 4.5.2). El estándar OpenMP fue diseñado en

1997, pertenece y es mantenido por la **ARB** (OpenMP *Architecture Review Board*) [Cha01]. Desde entonces varios fabricantes han incluido OpenMP en sus compiladores. La mayoría soporta la versión 2.5 desde mayo del 2005.

La filosofía de OpenMP es emigrar fácilmente un programa secuencial al paralelismo. De esta manera el API brinda multiples herramientas para reutilizar el código programado previamente para procesadores mononúcleo y pasarlos con ligeros ajustes a las nuevas arquitecturas multinúcleo. OpenMP principalmente permite manejar 2 niveles de paralelismo: paralelismo de bucles y paralelismo funcional.

Paralelismo de Bucles

Los ciclos *for* en lenguaje C y los ciclos *do* en lenguaje Fortran se pueden realizar de manera paralela mediante la directivas de compilador: `¡$omp parallel do` para fortran y `#pragma omp parallel for` para C. Mediante estas directivas, el compilador distribuirá el trabajo entre los hilos de ejecución disponibles. Si el compilador no soporta OpenMP (o se decide no activarlo), los bucles se realizarán secuencialmente.

Paralelismo Funcional

Ambos lenguajes C y Fortran soportan tener segmentos de código separados que el compilador distribuirá en distintos *threads* por secciones mediante `!$OMP SECTIONS` y `#pragma omp sections`.

El modelo de memoria de OpenMP distingue entre memoria compartida y memoria privada. El modelo de programación en OpenMP se basa en creación y destrucción de *threads* en un patrón *Fork-Join*.

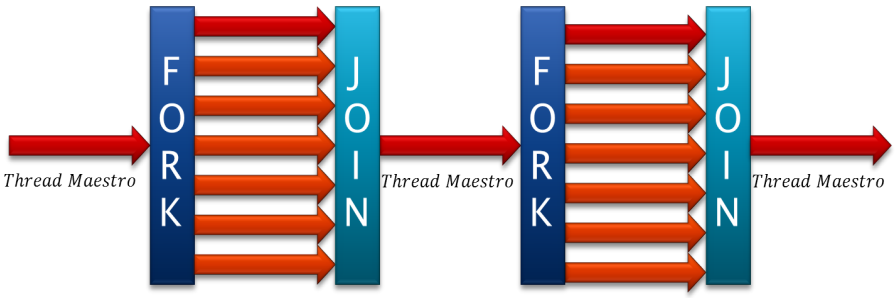


Figura 4.5: Representación gráfica de *Fork-Join*. Un *thread* maestro crea mediante operación `fork` múltiples esclavos, mediante la operación `join` los termina y vuelve a ser el único thread maestro [Con63].

4.5.2. *Fork-Join*

El patrón *Fork-Join* es un concepto simple para la creación de *threads* [Con63], donde un *thread* X crea un número de *threads* esclavos $X_1, ..., X_p$ con un declaración `fork`. Estos *threads* pueden ejecutar un programa en particular o función. Siendo que la ejecución del *thread* maestro X puede ejecutar la misma o diferente parte de la función, o esperar la terminación de $X_1, ..., X_p$ usando la declaración `join`.

4.6. Modelos de Escalabilidad Multinúcleo

El trabajo presentado por Sun y Chen [SC10] muestra un estudio realizado sobre la escalabilidad multinúcleo bajo las condiciones pesimistas de tamaño-fijo (*fixed-size*), y las optimistas de tiempo-fijo (*fixed-time*) y memoria-acotada (*bounded-memory*), así como de la perspectiva llamada barrera de memoria (*memory wall*). El análisis sobre escalabildad que realizan Sun y Chen en su articulo "*Reevaluating Amdahl's law in the multicore era*"[SC10] aplican los modelos que a continuación se explican.

4.6.1. Modelo de Tamaño-Fijo

El primer modelo que estudian Sun y Chen en su investigación es el modelo de escalabilidad en tamaño-fijo, esto debido a que en un trabajo anterior Hill y Marty proponen un modelo de hardware multinúcleo basado en este modelo en su investigación [HM08]. Siguiendo la ley de Amdahl, Hill y Mary concluyen que la velocidad de ejecución de las arquitecturas con sistemas multinúcleo es:

$$Sp = \frac{1}{\frac{1-f}{perf(r)} + \frac{f \cdot r}{perf(r) \cdot p}} \tag{4.9}$$

donde f es nuevamente la fracción de programa que debe ser ejecutada totalmente paralela, y $1 - f$ de manera secuencial, p es el número de núcleos trabajando en paralelo y $perf(r)$ es un valor dependiente de la implementación hardware actual, que para casos de análisis, se puede describir como una función arbitraria.

Con la ecuación 4.9, Hill y Marty intentaron demostrar que la escalabilidad de las plataformas multinúcleo es deficiente. Posteriormente, Sun y Chen muestran que la ecuación 4.9 no es más que una formulación diferente de la ley de Amdahl que invariablemente resulta de una suposición de carga de trabajo de tamaño fijo. Este modelo utiliza la ley de Amdahl (ver sección 4.4.2) para mostrar que tan escalable es un programa según su porción de programa inherentemente secuencial. Debido a que la evaluación de modelo utiliza un problema computacional de tamaño fijo (donde la carga de trabajo es fija y se divide en parte secuencial y paralela) es difícil encontrar una implementación que muestre escalabilidad.

4.6.2. Modelo de Tiempo-Fijo

El segundo modelo que estudian Sun y Chen en su investigación es el modelo de escalabilidad en tiempo-fijo. En este modelo utilizan la ley de Gustafson (ver sección 4.4.2) para mostrar que tan escalable es la arquitectura multinúcleo. Utiliza la filosofía de que los equipos de cómputo están diseñados para resolver problemas que incrementan su tamaño conforme se utilicen más procesadores. Muestran que la velocidad de ejecución escalada es la siguiente:

$$Sp = (1 - f) + mf \tag{4.10}$$

donde m muestra la el escalamiento del número de núcleos, y en concordancia a la ley de gustafson el incremento del tamaño de problema a resolver. La ecuación 4.10, muestra que las arquitecturas multinúcleo son escalables usando el modelo de computación escalable, y su velocidad de ejecución se incrementa linealmente con un factor de escalamiento. El nombre de tiempo-fijo parte del hecho que la porción inherentemente secuencial de un programa paralelo usualmente es la inicialización de datos y esta no cambiará drásticamente según el tamaño del problema, sino que la porción paralela es la que tiene un cambio radical según el tamaño de problema.

4.6.3. Modelo de Memoria-Acotada

El último modelo que estudian Sun y Chen en su investigación es el modelo de escalabilidad en memoria-acotada. Para este modelo utilizan un análisis similar al que utilizaron en el modelo de tiempo-fijo, y asumen que la carga de trabajo escalada tiene una restricción β. La velocidad de ejecución según el modelo de memoria acotada se describe como:

$$Sp = \frac{(1 - f)\beta + f\beta}{(1 - f)\beta + \frac{f\beta}{m}}. \tag{4.11}$$

Al igual que el modelo de tiempo fijo, este modelo revela que las arquitecturas multinucleo pueden escalar bien mientras el tamaño de la carga de trabajo se le permita crecer con el número de núcleos. El modelo refleja situaciones donde la capacidad de memoria es la restricción, en este caso los cache L1 de las arquitecturas multinúcleo. Este modelo explica la situación en la que el tiempo necesario para completar los cálculos de un programa está determinado por la cantidad de memoria disponible para almacenar los datos. Lo que acota o limita la capacidad de respuesta del programa es la velocidad de acceso a la memoria. Existe un salto grande entre la velocidad alcanzada por los procesadores y la velocidad que tiene el acceso a la memoria. A este gran salto se le conoce como barrera de memoria [HPG03].

Barrera de memoria

La barrera de memoria es la disparidad creciente entre la frecuencia del reloj de los procesadores y la memoria externa del procesador. Una importante razón de esta disparidad es el ancho de banda de comunicación acotado más allá de las limitantes de los *chips*. Dadas estas tendencias, se esperá que la latencia de la memoria se convierta en un cuello de botella enorme en el rendimiento del equipo. El los problemas computacionales de un tamaño razonable, los datos tienen que ser accedidos a través de la jerarquía de memoria, donde los retrasos de acceso de memoria son largos. En adición a esto se le añade el contenido del *cache* compartido **L2** y las trayectorias que deben seguirlos datos hacia el nivel más bajo de la memoria. La figura 4.6 ilustra la estructura general de un sistema multinúcleo, en donde se puede notar que cada núcleo posee su *cache* **L1**, y comparten *cache* **L2**.

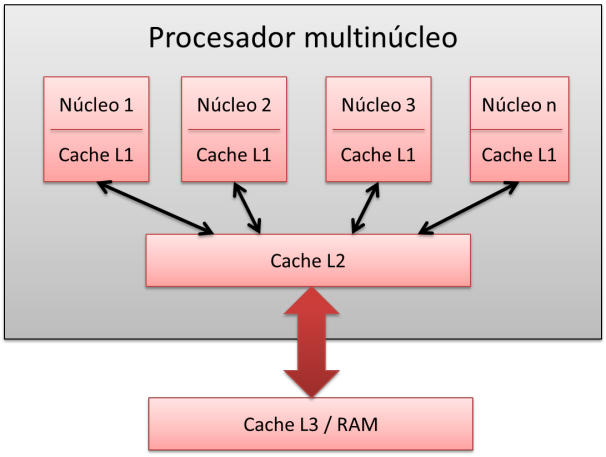


Figura 4.6: Arquitectura Multinúcleo segun Niveles de Memoria

4.7. Resumen del Capítulo

Este capítulo explica los fundamentos básicos de computación paralela, las arquitecturas existentes y la tendencia de programación que se utilizarán en los siguiente capitulos. Se explicaron las diferentes organizaciones de memoria dándole mayor importancia a la memoria compartida, la cual corresponde con el tipo de memoria usado por las plataformas multinúcleo. Se pudieron observaron las técnicas para evaluar implementaciones paralelas que serán utilizadas posteriormente en el capítulo 6 “Evaluación ”.

Capítulo 5

Algoritmo para Experimento

Las ecuaciones diferenciales parciales provienen de un gran número de problemas físicos, tales como el flujo de líquidos, transferencia de calor, mecánica de sólidos y procesos biológicos. Estas ecuaciones a menudo caen en uno de tres tipos: ecuaciones **hiperbólicas**, comúnmente asociadas con el proceso de advección, ecuaciones **parabólicas**, comúnmente asociada con la difusión y las ecuaciones **elípticas**, más comúnmente asociados con estados estacionarios de problemas, ya sean parabólicos o hiperbólicos. No todos los problemas caen fácilmente en uno de estos tres tipos. Los problemas de **Difusión-Advección** implican aspectos importantes de ambos problemas hiperbólicos y parabólicos. Casi todos los problemas de advección involucran una cantidad pequeña de difusión.

5.1. Ecuación Advectiva-Difusiva

La ecuación Advectiva-Difusiva posee tres componentes principales:

- Advección: es la variación de un escalar en un punto dado, por efecto de un campo vectorial. Representa el movimiento de transporte que tendrá el

flujo por una dirección y velocidad determinadas.

- Difusión: es un proceso en el que partículas materiales se introducen en un medio que inicialmente estaba ausente, aumentando la entropía del sistema conjunto formado por las partículas difundidas o soluto y el medio donde se difunden o disolvente. Esto representa el cambio de concentración que tendrá el flujo pasando de una concentración relativamente alta a una donde se tenga baja concentración.

- Acumulación: es la variación temporal de la función u respecto a la variable independiente t.

La ecuación Advectiva-Difusiva contemplando sus principales componentes resulta de la siguiente manera [GDN98]:

$$\frac{\partial u}{\partial t} + \overline{a}\frac{\partial u}{\partial x} = w\frac{\partial^2 u}{\partial x^2} \tag{5.1}$$

donde $\overline{a}$ representa el vector de velocidad del componente de advección, w representa el factor de difusividad (Segunda Ley de Fick) [**?**].

5.1.1. Ecuación en 1D

Cuando la función incógnita u depende únicamente de las variables independientes t, x; es decir, $u = u(t, x)$ y los parámetros a, w son números reales en la ecuación 5.1, se dice que tenemos un problema 1D. La ecuación en 1D se escribe de la siguiente manera:

$$\frac{\partial u}{\partial t} + a_x\frac{\partial u}{\partial x} = w\frac{\partial^2 u}{\partial x^2} \tag{5.2}$$

donde la ecuación diferencial está definida en un dominio en el espacio y tiempo, $0 < x < b$, $t \geq 0$. Además, la ecuación diferencial es suplementada con condiciones iniciales y condiciones de frontera.

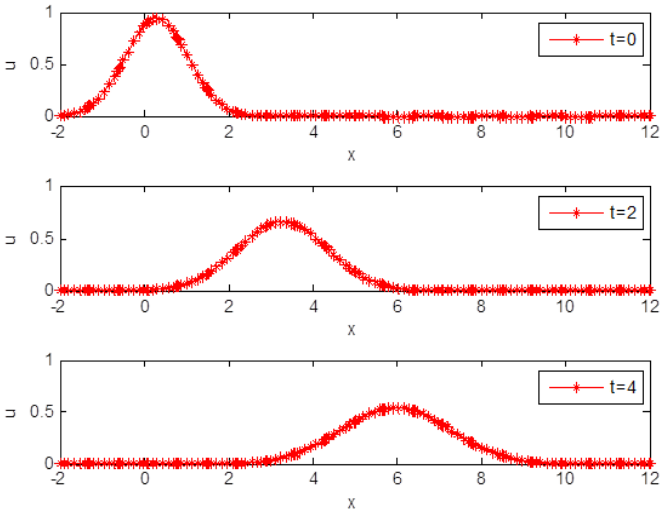


Figura 5.1: Comportamiento de una Ecuación Advecita-Difusiva en 1D de la función $\frac{\partial u}{\partial t} + a_x \frac{\partial u}{\partial x} = w \frac{\partial^2 u}{\partial x^2}$ de componentes $a_x = 2$, $w = 0.2$ y condición inicial $u(x, 0) = \exp(-x^2)$

Al tiempo de inicio $t = 0$, u debe coincidir con una función dada $u(x, 0) = f(x)$, $0 \leq x \leq b$. La llamadas condiciones de frontera de Dirichlet están dada por $u(0, t) = g_0(t)$, $u(b, t) = g_b(t)$, se supone que $w > 0$.

Un ejemplo del comportamiento de la ecuación advectiva-difusiva de una dimensión se presenta en la figura 5.1, donde se puede observar el desenvolvimiento para un parámetro de velocidad $a_x = 2$, $w = 0.2$ y con una condición inicial $u(x, 0) = \exp(-x^2)$. Se puede observar que la función gaussiana al inicio se dispersa (difusión) conforme avanza a través del eje x (advección).

5.1.2. Ecuación en 2D

En el caso 2D la función incógnita u ahora depende de las variables independientes t, x, y; es decir, función incógnita se escribre $u = u(t, x, y)$. Se presenta por la siguiente expresión:

$$\frac{\partial u}{\partial t} + a_x \frac{\partial u}{\partial x} + a_y \frac{\partial u}{\partial y} = w\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{5.3}$$

donde la ecuación está definida en una región en el espacio tridimensional, $0 < x < b$, $0 < y < v$, y $t \geq 0$. Las con condiciones iniciales se determinan en el tiempo de inicio $t = 0$, con u que debe coincidir con una función dada $u(x, y, 0) = f(x, y)$, para $0 \leq x \leq b$ y $0 \leq y \leq c$. La llamadas condiciones de frontera de Dirichlet están expresadas por:

$$\begin{cases} u(x, 0, t) = g_0(x, t) \\ u(0, y, t) = h_0(y, t) \\ u(x, c, t) = g_c(x, t) \\ u(b, y, t) = h_b(y, t) \end{cases} \tag{5.4}$$

que establecen los valores en las 4 fronteras del espacio rectangular definido por $[0, b] \times [0, c]$.

Un ejemplo del comportamiento de la ecuación advectiva-difusiva de una dimensión se puede observar en la figura 5.2, donde se puede observar el desenvolvimiento para unos parámetro de velocidad $a_x = 2$, $a_y = 2$, y un factor de dispeción $w = 0.2$ y con una condición inicial $u(x, y, 0) = \exp(-x^2 - y^2)$. Se puede observar que la función gaussiana de inicia se dispersa (difusión), conforme avanza a través de los ejes x y y (advección).

Las EDPs de tipo Advección-Difusión son utilizadas para realizar simulaciones computacionales de modelos de dispersión de poblaciones, transferencia de contaminantes, calidad del aire y efectos térmicos. El problema seleccionado para realizar los experimentos es la ecuación prototípica puesta en la ecuación 5.2, que sirve para simular la dispersión de partículas en una corriente de contaminantes en agua, donde u representará la concentración de partículas en un medio.

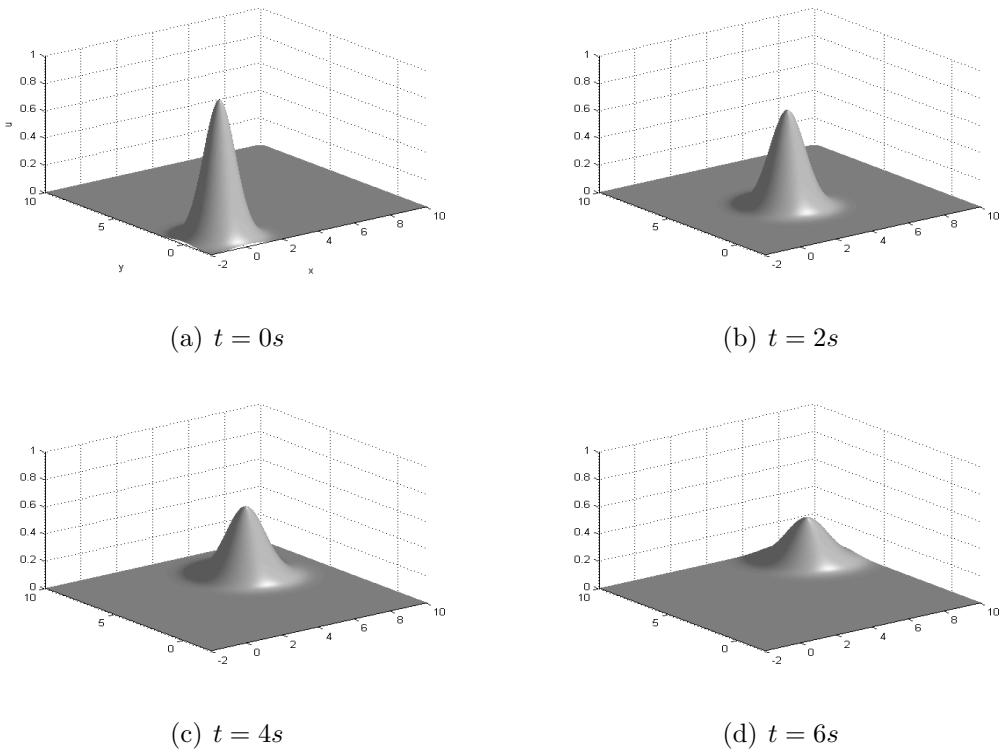
5.2. Consideraciones de Implementación

En el capítulo 3 se indicó que se optó por escoger el MDD conocido como Schwarz aditivo para realizar su implementación con el uso de paralelismo. Una de las ventajas que proporciona el método es la alta separabilidad de las soluciones que puede representarse por ciclos `for` (sentencias `do` para la implementación en Fortran), además de que estos ciclos se presentan de tal forma que son independientes unos con otros.

5.3. Diseño del Algoritmo

El método seleccionado fue el MDD Schwarz Aditivo, explicado en la sección 3.3.1 mediante el algoritmo 2, y se realizó la implementación de manera secuencial mediante programación en lenguaje fortran. En esta implementación se estableció inicialmente el número de subdominios a 2, teniendo la posibilidad de aumentar el número. También, se permite establecer el número de puntos de discretización, los valores de frontera y el porcentaje de traslape de los subdominios. La solución de los subdominios se restringe a un espaciado lineal uniforme en el caso 1D y mallas rectangulares para el caso 2D. La discretización del dominio se realiza mediante diferencia finita central.

El método Schwarz nos permite utilizar la filosofía del "divide y vencerás". Pero aún es necesario resolver los problemas generados para los subdominios. Para solucionar los sub-problemas que se generan como Sistemas de Ecuaciones Lineales se utiliza la factorización LU para resolver el sistema. Para facilitar la programación del proceso secuencial se utilizaron las librerías BLAS y LAPACK en su versión *thread-safe*, las cuales permiten al usuario utilizar las versiones de un solo *thread* en caso de que existan múltiples procesos que resuelvan problemas independientes a la vez [AfIM99].



(a) $t = 0s$ (b) $t = 2s$

(c) $t = 4s$ (d) $t = 6s$

Figura 5.2: Comportamiento de una Ecuación Advecita-Difusiva en 2D de la función $\frac{\partial u}{\partial t} + a_x \frac{\partial u}{\partial x} + a_y \frac{\partial u}{\partial y} = w(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$ de componentes $a_x = 2$, $a_y = 2$, $w = 0.2$ y con una condición inicial $u(x, y, 0) = \exp(-x^2 - y^2)$ en diferentes tiempos

Dado un conjunto de nodos distribuidos de manera uniforme, el esquema de partición de datos consiste en partir el dominio Ω mediante una cuadrícula de $N \times N$ para el caso en 2D, y en un eje con N particiones para el caso 1D. Se dividen Ω en C subdominios Ω_i donde $i = 1, 2, ..., C$ a través de un mismo eje, para poder tener un máximo de 2 fronteras artificiales por método sin importar si el problema es 1D o 2D. Debido a esto, un parámetro a considerar es el porcentaje de traslape entre subdominios, pudiéndose modificar mediante un parámetro de entrada. En el caso de $C > 2$, existen dos tamaños diferentes de subdominios, esto se debe a que las fronteras reales del dominio original Ω delimitan a los sub-dominios Ω_1 y Ω_C, lo cual hace que el tamaño de ellos sea ligeramente más pequeños que el resto de subdominios $(\Omega_{1+1}, ..., \Omega_{C-1})$ ya que

no existe traslape de subdominios en una de sus fronteras.

5.3.1. Método Schwarz Aditivo Paralelo

La implementación paralela del algoritmo se realizó mediante el esquema de trabajo `fork-join` (ver sección 4.5.2), dividiendo por cada *thread* la solución de un dominio. De esta manera se puede hacer realizar el intercambio de información una vez que cada *thread* haya terminado su parte, haciendo esperar a los *threads* mediante una barrera explícita, y utilizar la información de las fronteras artificiales en la siguiente iteración. El algoritmo 4 muestra de manera breve la implementación.

Mediante esta implementación se puede realizar una iteración a la vez, ya que se está ajustando el número de dominios al número de núcleos disponibles. Esto se hizo de tal forma que al momento de realizar las pruebas en la etapa de evaluación se pudiera aumentar o disminuir el número de núcleos sin comprometer la carga de trabajo, es decir, existe la posibilidad de resolver problemas más grandes de manera más precisa con mayor número de núcleos existentes.

La primera labor del programa es dar los valores de inicio a los datos necesarios para el control de la aplicación de manera secuencial, posteriormente es necesario crear la malla del dominio entero y guardarlo en la matriz que posteriormente será utilizada para que cada uno de los *threads* obtenga información del subdominio necesario. Hasta este punto es cuando se inicia la parte paralela del algoritmo, que se hace mediante un `parallel do`, se llama a la solución del dominio en cuestión mediante descomposiciones LU en versión *thread-safe* y una vez finalizada la resolución del sub-problema se espera mediante una barrera explícita. Posterior a este paso se actualizan los valores de las fronteras artificiales a la matriz principal para iniciar una nueva iteración.

Para finalizar la ejecución del problema se genera un vector que compara las

fronteras artificiales con los valores previamente obtenidos para ellas al interior de los subdominios (el error). A este vector e se le calcula su norma euclidiana y si es menor a un valor ϵ proporcionado al inicio se detiene la ejecución, si no lo es se prosigue a la siguiente iteración.

Algoritmo 4: Schwarz Aditivo Paralelo

begin

Interpolación de fronteras artificiales

$u_1^0 \longleftarrow \alpha$ en Γ_1

$u_2^0 \longleftarrow \beta$ en Γ_2

for $k = 1, 2, 3, ...$ **do**

i es el número de dominios; **for** $i = 1, 2, 3, ...$ **do in parallel**

Se resuelve el problema para u_i^k

$Lu_i^k \longleftarrow f$ en Ω_i

$u_i^k \longleftarrow g$ en $\partial\Omega_i \backslash \Gamma_i$

Barrera

Se Sincronizan las fronteras $u_i^k \longleftarrow u_i^{n-1}$ en Γ_i

if $\left\|e^k\right\| \leq \epsilon$ **then**

Condición de paro

5.4. Análisis de Complejidad

El análisis de complejidad del algoritmo 4 se realiza inicialmente mediante el caso de ejecución mononúcleo, es decir, sólo se tiene a la disposición un procesador para los cálculos. Se decidió utilizar la factorización LU para encontrar la solución del sistema de ecuaciones lineales debido a que es un método directo y permite tener un mejor control sobre la complejidad del

algoritmo. Aun cuando su costo computacional es alto en contraste con los métodos iterativos, no es un impedimento, ya que el ámbito de este trabajo se enfoca en como las plataformas multinúcleo son capaces de resolver problemas de tamaño moderado. Siendo esto, la factorización LU es suficientemente rápida como para considerarse adecuada a la solución. Si se deseara resolver problemas de mayor tamaño, sería conveniente remplazar este método por un método basado en iteraciones.

El costo computacional de la factorización LU para la primera solución es de $O(N^3)$, en donde N es el tamaño del vector de solución. Una vez realizada la obtención de la matriz L y U, cada sustitución hacia adelante y hacia atrás tiene un costo de $O(N^2)$ por lo cual, si el problema requiere resolver varios sistemas de ecuaciones que posean la misma matriz de coeficientes, como en este caso, cada solución extra requerirá un costo de $O(2N^2)$. El costo en espacio para resolver el sistema requiere tener 3 matrices disponibles de tamaño $N \times N$, por lo que resulta de $O(3N^2)$.

Si consideramos al dominio Ω que cuenta con un tamaño n, es decir, $|\Omega| = n$, el costo computacional para el dominio completo sin utilizar MDD será $O(n^3)$, mientras que el costo en memoria que requiere es de $O(3n^2)$. Ahora consideramos el caso en el que el dominio Ω es particionado en C subdominios donde C es potencia de 2, y el tamaño de cada subdominio es aproximadamente el mismo número de nodos Ω_i y lo denominaremos m, es decir $|\Omega_i| = n/C = m$ donde $i = 1, 2, .., C$. Para este caso se requiere la factorización inicial LU, y posteriormente resolver C problemas lineales por lo que queda la siguiente expresión:

$$O(Cm^3) + O(2Cm^2). \tag{5.5}$$

si se sustituye m queda en términos del problema original:

$$O(\frac{1}{C^2}n^3) + O(\frac{2}{C}n^2), \tag{5.6}$$

y ya que es un proceso iterativo debemos agregar un coeficiente k que nos indique ese costo adicional:

$$O(\frac{1}{C^2}n^3) + O(\frac{2k}{C}n^2). \tag{5.7}$$

Un mayor número de particiones C hará que disminuya el costo del algoritmo. Si bien sabemos que para usar MDD tienen que existir subdominios, y se logra igualar ese número al número de procesadores. La parte paralela solo se encuentra en la solución de sistemas lineales para subdominios y se hace de manera directa por secciones, porque se espera que en un caso ideal, el costo de esa parte se divida en el número de procesadores (núcleos) trabajando. El costo se representa por la siguiente expresión:

$$O(\frac{1}{C^2}n^3) + O(\frac{2k}{C}\frac{1}{C}n^2) = O(\frac{1}{C^2}n^3) + O(\frac{2k}{C^2}n^2) \tag{5.8}$$

La expresión anterior muestra una notoria mejora en la parte iterativa del algoritmo, ya que disminuye el tiempo de ejecución de esta parte con un factor de $\frac{1}{C}$, esto muestra una mejora en el tiempo de ejecución de un problema conforme aumenta el número de procesadores presentes.

5.5. Resumen del Capítulo

En este capítulo se pudo observar el tipo de ecuaciones a resolver, que son las EDPs del tipo Advección Difusión, su explicación y un ejemplo de su comportamiento a través del tiempo. Se decidió utilizar la ecuación que representa la dispersión de partículas en una corriente de contaminantes en agua en 1D y 2D. Posteriormente se explicó la metodología que se utilizó para realizar la implementación del algoritmo paralelo diseñado a partir del método Schwarz Aditivo.

La tabla 5.1 resume los costos computacionales y de espacio de los métodos. Mientras los métodos muestran un tiempo con exponente cúbico para las

matrices, la variación de los costos se encuentran en el tamaño de las matrices que se utilizan. La matriz original es de tamaño N, con el MDD se presenta un ahorro haciéndose más pequeña la matriz de tamaño $n < N$. Finalmente la implementación del MDD en paralelo permite un ahorro proporcional al número de procesadores usados para resolver el problema. Un mayor número de procesadores involucra un decremento del tamaño de matriz n aún mayor. (Las matrices se hacen más pequeñas conforme se utilizan más procesadores), esto implica una solución más rápida para un mismo problema, o el manejo de un problema de tamaño mayor ocupándose el mismo tiempo de solución. Este caso se abordará con mayor énfasis en el capítulo 6.

Tabla 5.1: Complejidad de los Métodos de Descomposición de Dominio

Complejidad	Método Directo	MDD Schwarz sobre 1 núcleo	MDD Schwarz sobre C núcleos
Costo Computacional	$O(N^3)$	$O(\frac{1}{C^2}n^3) + O(\frac{2k}{C}n^2)$	$O(\frac{1}{C^2}n^3) + O(\frac{2k}{C^2}n^2)$
Costo de espacio	$O(2N^2)$	$O(2n^2)$	$O(2n^2)$

Capítulo 6

Evaluación

La motivación más importante por la cual usar sistemas paralelos es la reducción de tiempos de ejecución en programas de computación intensiva. Los programas a los que se encuentra enfocada esta tesis son las simulaciones científicas, o sistemas en los cuales la respuesta rápida es vital. El tiempo de ejecución de programas paralelos es dependiente de varios factores, entre los que intervienen el compilador y el sistema operativo usado, la plataforma paralela en la que se ejecuta, el modelo de programación paralela en el que se basó el algoritmo y por último, la capacidad de realizar paralelismo proporcionada por el mismo algoritmo. En este capítulo, se abordará la verificación del algoritmo programado, las medidas de desempeño paralelo para poder evaluar tanto el desempeño del algoritmo, asi como la escalabilidad que posee dentro de las plataformas multinúcleo. Finalmente, realizamos un análisis sobre la interpretación de estos experimentos numéricos.

6.1. Desempeño

Para evaluar el desempeño de un programa paralelo es necesario obtener el tiempo de ejecución paralela $T_p(n)$ que se representa como el tiempo generado entre el inicio del programa y el fin de la ejecución de todos los procesadores (núcleos) participantes. Usualmente este tiempo se expresa para un número p de participantes con un tamaño de problema n. Para nuestro caso, el tamaño de problema n representa el número de nodos (discretización) del eje, en el problema. Es decir, n representa la calidad de malla, para un número grande el problema se resolverá con mayor precisión (en un mayor número de nodos). Dependiendo de la arquitectura de la plataforma de ejecución se puede divir T_p en lo siguiente:

- Tiempo de ejecución de cálculos locales. Es representado por los cálculos realizados por cada núcleo usando sus datos en la memoria local (memoria privada).
- Tiempo de ejecución del intercambio de datos entre procesadores. Es el tiempo que lleva las operaciones explícitas de comunicación.
- Tiempo de ejecución para la sincronización de los núcleos participantes. Esto se realiza cuando varios *threads* tienen que acceder a un espacio de memoria compartida visible para todos los núcleos.
- Tiempo de espera. Se genera al darse cargas de trabajo no balanceadas; los *threads* que hayan terminado su trabajo local deben esperar a que el trabajo de los demás *threads* haya terminado.

A los tiempos de intercambio de datos, espera y sincronización suelen considerarse como ***overhead*** (exceso de tiempo computacional) y por lo general son descartados de los resultados. En este caso se descartan los tiempos de

inicialización de datos, y el de sincronizaciones explícitas, mientras que los tiempos de espera si son utilizados ya que son un reflejo de la distribución de cargas de trabajo en *threads* [Cro94].

Los experimentos se realizaron con la siguiente configuración de sistema: PC con procesador Intel Xeon a 2.33GHz con 8 núcleos físicos de un solo thread, memoria RAM de 4 GB, sistema operativo Windows 7 64 bits. El código fue desarrollado en lenguaje Fortran bajo Microsoft Visual Studio 2010 Shell con el compilador PGI Visual Fortran 10.9.

Tabla 6.1: Parámetros del Experimento 1 (Grupo de Control)

	1D	2D
EDP	$\frac{\partial u}{\partial t} + a_x \frac{\partial u}{\partial x} = w \frac{\partial^2 u}{\partial x^2}$	$\frac{\partial u}{\partial t} + a_x \frac{\partial u}{\partial x} + a_y \frac{\partial u}{\partial y} = w(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$
a_x	1.5	2.5
a_y		2.5
w	0.3	0.4
Dominio	$-3 \leq x \leq 12$	$-3 \leq x \leq 12$ $-3 \leq y \leq 12$
Condición inicial	$u(x,0) = \exp(-x^2)$	$u(x,y,0) = \exp(-x^2 - y^2)$
Condiciones frontera	$u(-3,t) = 0$ $u(12,t) = 0$	$u(x,-3,t) = 0$ $u(-3,y,t) = 0$ $u(x,12,t) = 0$ $u(12,y,t) = 0$
Tiempo de simulación	$t_{max} = 8s$ $\Delta t = 0.05$	$t_{max} = 8$ $\Delta t = 0.05$
Tolerancia	$\epsilon = 1 \times 10^{-3}$	$\epsilon = 1 \times 10^{-3}$

En los experimentos deliberadamente se usó código que resuelve, mediante MDD, un problema de un caso de EDP advectiva-difusiva y no usa archivos de E/S. La medición de los tiempos se hace mediante la función proporcionada

por OpenMP `OMP_GET_WTIME()` y se mide sin inicialización de datos, solo la solución.

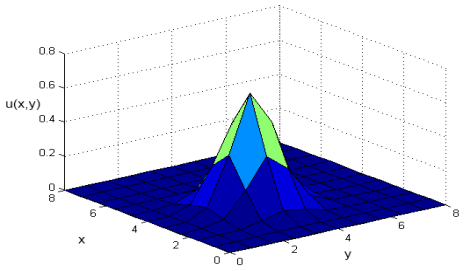
El objetivo principal de la primera prueba es validar que el programa paralelo implementado cumple los requisitos establecidos en la solución del problema: resolver el problema correctamente con la exactitud requerida y obtener incremento en la velocidad de respuesta. Posteriormente, se evaluará la escalabilidad multinúcleo del programa usando condiciones de los modelos **tiempo-fijo** y **memoria-acotada**, para finalizar con la perspectiva de **barrera de memoria**.

Para medir efectivamente la rapidez de ejecución (la disminución del tiempo de ejecución obtenida al paralelizar) es necesario tener primero un grupo de control, es decir, el tiempo de respuesta del algoritmo original resuelto con $p = 1$ (algorimo secuencial). La aplicación del método de Schwarz requiere la división del dominio; se manejan 2, 4 y 8 particiones del dominio principal, donde el número total de nodos por eje en la malla será $N = \{10, 25, 50\}$. El **Experimento 1** consiste en la obtención del grupo de control, y se observan los resultados en la tabla 6.2, que muestra los tiempos de ejecución, en segundos, para los tamaño considerados.

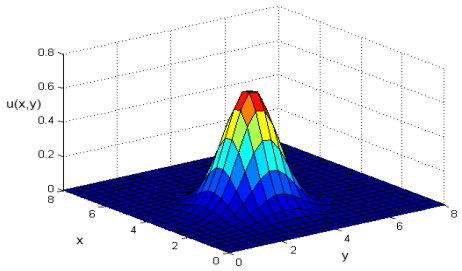
Tabla 6.2: **Experimento 1:** Tiempos de Ejecución del MDD Aditivo Secuencial (segundos)

	1D			2D		
Subdominios	$N = 10$	$N = 25$	$N = 50$	$N = 10$	$N = 25$	$N = 50$
2	0.080	0.450	6.310	2.420	98.430	434.505
4	0.020	0.110	1.543	0.588	24.101	106.420
8	0.007	0.042	0.588	0.224	9.188	40.569

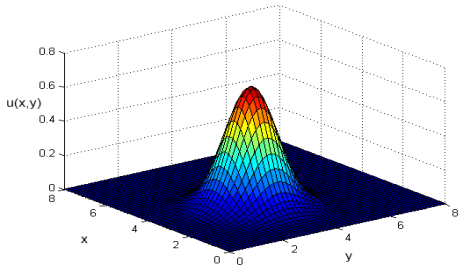
La información presentada en la tabla 6.2 se requiere para ratificar lo visto en la sección 5.4 en donde se explicó el porqué al aumentar el número de



(a) $N = 10$



(b) $N = 25$



(c) $N = 50$

Figura 6.1: Superficie Resultado en MDD 2D del Experimento 1. Comportamiento de la respuesta con diferente número de discretización $N = 10, 25, 50$.

subdominios disminuye el tiempo de ejecución. En la figura 6.1 se muestra como número de nodos afecta la resolución del problema, notandose una diferencia notable

6.1.1. Rapidez de ejecución

El **Experimento 2** se realizó con la implementación en 2D; se ejecutó múltiples veces usando un número determinado de núcleos p. Este experimento tiene dos objetivos, el primero es constatar la validez del algoritmo para realizar más pruebas en paralelo sobre plataformas multinúcleo, y la segunda es realizar las pruebas necesarias para evaluar al algoritmo mediante la rapidez de ejecución para el modelo de tamaño-fijo.

Tabla 6.3: Parámetros del Experimento 2

	2D
EDP	$\frac{\partial u}{\partial t} + a_x \frac{\partial u}{\partial x} + a_y \frac{\partial u}{\partial y} = w(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$
a_x	1.4
a_y	0.5
w	0.3
Condición inicial	$u(x, y, 0) = \exp(-x^2 - y^2)$
Dominio	$-3 \leq x \leq 12$ $-3 \leq y \leq 12$
Condición inicial	$u(x, y, 0) = \exp(-x^2 - y^2)$
Condiciones frontera	$u(x, -3, t) = 0$ $u(-3, y, t) = 0$ $u(x, 12, t) = 0$ $u(12, y, t) = 0$
Tiempo de simulación	$t_{max} = 8s$ $\Delta t = 0.05$
Tolerancia	$\epsilon = 1 \times 10^{-3}$

El número de particiones se restringe a ocho para ejecutar un mismo programa utilizando parámetros similares pero en un número diferente de procesadores, es decir, una ejecución en paralelo. Los parametros de ejecución se presentan en la tabla 6.3. Los tiempos obtenidos se pueden ver en la tabla 6.4, que muestra que tiempo se decrementa conforme aumenta el número de núcleos trabajando en el problema. Se observa que para el problema actual el tamaño de la matriz inicial marca un cambio considerable en el tiempo de ejecución así como en el impacto que tiene el número de procesadores sobre el paralelismo. Con los tiempos de ejecución paralela obtenidos en el experimento 2, se calcula la rapidez de ejecución (ver ecuación 4.2)y se obtienen los resultados que se

pueden observar en la figura 6.2, la cual muestra una tendencia muy parecida a la lineal para las ejecuciones con un mayor tamaño, esto cumple la meta establecida de escalabilidad para problemas de buen tamaño (arriba de 500 puntos de discretización) [ZT00]. En el caso de la implementación de tamaño $N = 10$ los problemas a resolver por los núcleos son relativamente pequeños, esto hace que la fracción paralela de la aplicación sea equiparable con la fracción secuencial del problema (ver Ley de Amdahl en la sección 4.4.2). Esto muestra que la ejecución en paralelo del algoritmo hace que mejore el desempeño para problemas grandes y no en problemas pequeños.

Tabla 6.4: **Experimento 2:** Tiempos de Ejecución Paralela (segundos)

Procesadores	$N = 10$	$N = 25$	$N = 50$
1	0.224	9.188	40.569
2	0.123	4.640	20.325
4	0.073	2.366	10.203
8	0.048	1.229	5.142

6.1.2. Eficiencia

La eficiencia es un indicador de desempeño que funciona como alternativa a la rapidez de ejecución. La eficiencia captura la fracción de tiempo para la cual un procesador paralelo es útilmente empleado para resolver el problema. La eficiencia se define en la ecuación (4.3), la cual es la fórmula utilizada, junto con los tiempos de ejecución obtenidos del experimento 1 para obtener la tabla 6.5.

La figura 6.3 muestra de manera gráfica las tendencia que siguen las implementaciones respecto a la eficiencia. Un algoritmo entre más eficiente sea, debera tener la eficiencia cercana a 1, en el caso de $N = 25$ y $N = 50$ no se

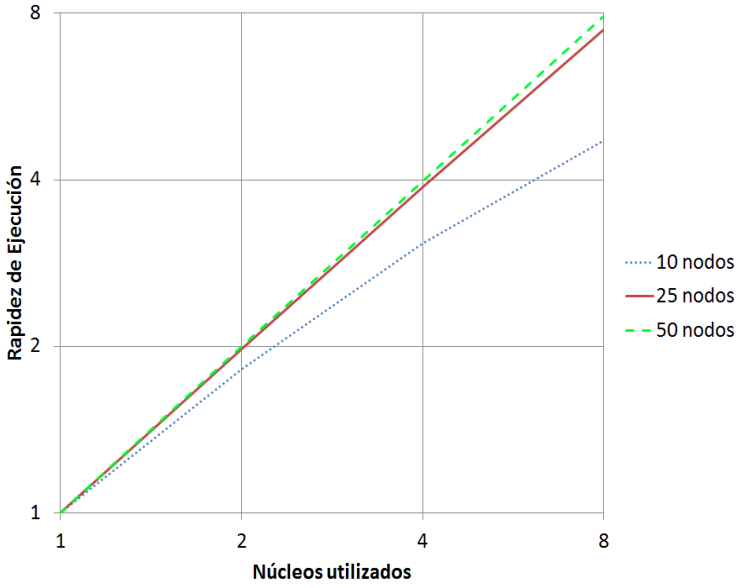


Figura 6.2: Rapidez de ejecución en MDD Schwarz Aditivo con 8 subdominios del Experimento 2.Se observa que en las ejecuciones de tamaño 25 y 50 se tiene una tendencia lineal del comportamiento de la rapidez conforme aumenta el número de núcleos.

perciben grandes separaciones del 1, pero en el caso de $N = 10$ se ve que la tendencia al crecer el número de procesadores es de volverse menos eficiente.

Mediante las dos métricas de evaluación se puede constatar que el algoritmo diseñado para la resolución del método Schwarz aditivo tiene un desempeño valido para una aplicación paralela. Se pudo observar que conforme el programa se hacía de mayor tamaño la aceleración que obtiene se acerca a la linealidad, característica muy deseada en cómputo paralelo.

6.1.3. Modelo Tamaño-Fijo

La idea original presentada por Amdahl fue una observación general acerca de las limitaciones sobre el mejoramiento del desempeño en cualquier intento de mejora en un programa paralelo, y luego fue condensada como la ley de Amdahl [Amd67]. En esta ley, la rapidez de ejecución se define como el cociente

Tabla 6.5: **Experimento 2:** Eficiencia

Procesadores	$N = 10$	$N = 25$	$N = 50$
1	1.000	1.000	1.000
2	0.909	0.990	0.998
4	0.769	0.971	0.994
8	0.588	0.935	0.986

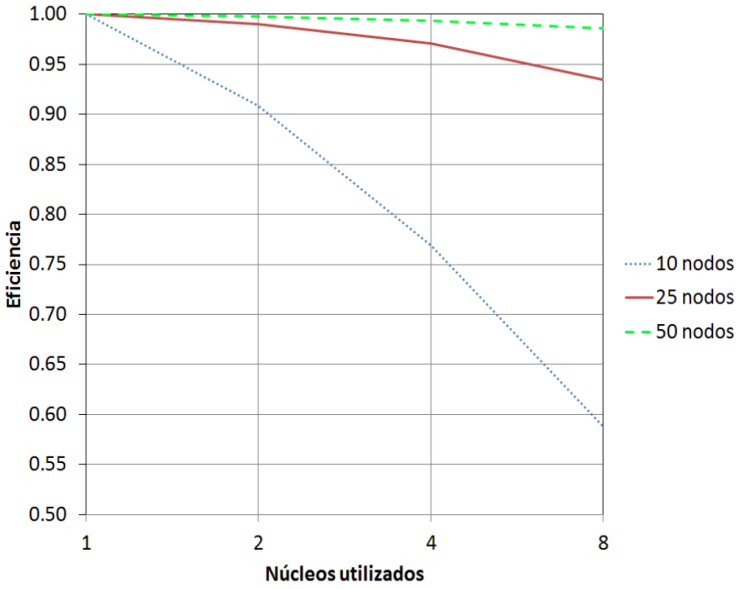


Figura 6.3: La gráfica denota una buena eficiencia en las ejecuciones con tamaño 25 y 50, y muestra la poca eficiencia de la ejecución con tamaño 10.

del tiempo de ejecución secuencial sobre el tiempo de ejecución paralelo.

En la ley de Amdahl se asume que la carga de trabajo (el tamaño del problema) se establece en la ejecución secuencial, y no cambiará para su ejecución paralela. Por lo tanto, el aumento de velocidad que se realiza mediante paralelismo (la reducción del tiempo de ejecución) se realiza sobre un mismo problema. Esta restricción acota las posibilidades de escalabilidad de un programa haciéndolas dependientes de la naturaleza misma del problema; y es por este motivo, que a la ley de Amdahl, también se le conoce como *modelo de*

rapidez de ejecución en tamaño-fijo.

El modelo de rapidez de ejecución multinúcleo para cargas de tamaño fijo obedece la siguiente expresión:

$$S_p = \frac{1}{(1-f)+(f/p)} \tag{6.1}$$

donde f es la fracción paralela de la implementación y p es el número de núcleos utilizados. Aplicando este modelo a un número de procesadores infinito, se tiene:

$$\lim_{p\to\infty} S = \frac{1}{1-f}. \tag{6.2}$$

Este modelo ilustra poca escalabilidad de las arquitecturas multinúcleo, como se puede observar en la figura 6.4, el comportamiento de la velocidad del programa se ve fuertemente influenciada por el porcentaje de paralelismo inherente del problema. Es necesario tener una alta porción de programa paralelo (arriba del 99 %) para que el incremento de núcleos se considere escalable, es decir, que se observe un comportamiento lineal. Se puede decir que la escalabilidad es aceptable cuando la porción de paralelismo es grande, que es el caso de nuestro algoritmo ya que crece con el tamaño de problema.

Este modelo debe ser tomado con discreción, ya que brinda información relevante sobre la escalabilidad de las plataformas multinúcleos para un problema de tamaño fijo. Esto en ningún momento debe aceptarse como un fundamento negativo a la escalabilidad que pueden proporcionar las plataformas multinúcleo. Lo que se puede acentuar respecto al experimento 2 mostrado anteriormente, es que un problema de tamaño considerable, cumple con la meta de escalabilidad del modelo de tamaño-fijo. En la siguiente sección se expone un modelo en el que si se le da importancia al tamaño de problema para evaluar la escalabilidad.

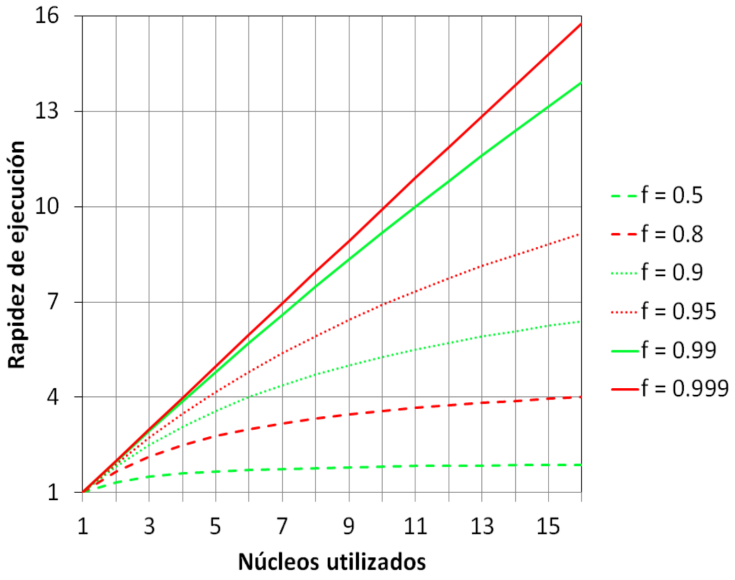


Figura 6.4: Se muestra la aceleración obtenida mediante las arquitecturas multinúcleo para diferentes fracciones, para que la implementación se comporte de manera escalable (con una tendendia lineal) es necesario que la fracción será mayor al 99 %.

6.2. Escalabilidad

Esta sección define como determinar si el algoritmo diseñado se puede considerar escalable o no; siendo que la escalabilidad es una medida que indica si una mejora de rendimiento puede llegar a ser proporcional al número de procesadores utilizados. La escalabilidad depende de varias propiedades de un algoritmo y de su ejecución en paralelo. A menudo, para un problema de tamaño fijo n se puede observar una saturación en la rapidez de ejecución cuando el número de procesadores p se incrementa. Sin embargo, el aumento del tamaño de n en un problemaejecutandose con un número de procesadores p incrementandose conduce a un aumento en la rapidez de ejecución alcanzada. Por lo tanto, la escalabilidad es una propiedad importante que poseen los programa paralelos, ya que mediante ella se puede expresar cuales problemas de tamaño grande podrian resolverse en el mismo tiempo que otros problemas

de tamaño pequeño, si el número suficiente de procesadores es empleado.

6.2.1. Modelo Tiempo-Fijo

El modelo de rapidez de ejecución de tiempo-fijo se basa en la ley de Gustafson [Gus88], que se puede resumir en la siguiente expresión:

$$S_p = (1 - f) + pf \tag{6.3}$$

donde f es la fracción de programa que se puede ejecutar en paralelo y p es el número de núcleos trabajando en paralelo, la ecuación (6.3) muestra que las arquitecturas multinúcleo son escalables siguiendo el modelo de computación escalable, y que la rapidez de ejecución medido mediante este modelo de Tiempo-Fijo, crecerá linealmente mediante un factor de escalamiento.

Este modelo ilustra gran escalabilidad de las arquitecturas multinúcleo, tal como lo muestra la figura 6.5, se observa que la importancia del tamaño de la fracción f, no es tan significativa, y que siempre se observará un ascenso proporcional al número de procesadores activos. Así también, hace notar que con un mayor porcentaje de fracción paralela f el incremento de velocidad que se obtiene es mayor. La rapidez de ejecución que se obtiene por este modelo proporciona una visión más optimista a las arquitecturas multinúcleo que la presentada en el modelo anterior.

El **Experimento 3** se realizó con ambas implementaciones en 1D y 2D, se ejecutaron multiples veces usando un número de núcleos p. El número de subdominios de la ejecución se iguala al número de núcleos trabajando, esto con la intención de que cada *thread* se encuentre ocupado con una tarea similar en carga de trabajo a los demas *threads*, sirviendo esto para ejecutar un mismo programa con los mismos parámetros, pero variando el tamaño del dominio con un número diferente de procesadores (discretización más fina para un mayor número de núcleos). Estas condiciones obedecen al modelo de tiempo-fijo, en el

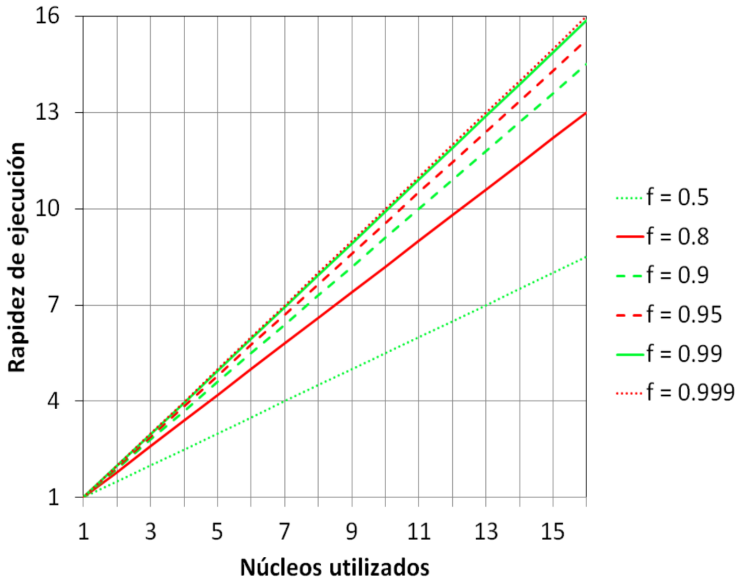


Figura 6.5: Se muestra la rapidez de ejecución obtenida mediante las arquitecturas multinúcleo para diferentes fracciones con posibilidad de paralelismo, en todos los casos se tienen tendencias lineales que crecen.

cual el tamaño del problema debe ser escalado conforme se aumenten el número de procesadores trabajando en él. Se propone una regla en el incremento del tamaño de problema para 3 diferentes tamaños iniciales $N = 10, 25, 50$, la regla dice que se incrementará el número de subdominios por prueba utilizando un coeficiente p. Esto es, el número de dominios sera igual al número de núcleos trabajando. El tamaño de dominio será $N_p = N_{ini}$, con la finalidad de que el incremento del dominio general, ajuste el tamaño de los dominios y siempre quede balanceada en el mismo tamaño de subdominio. La finalidad de este experimento es la obtención de los argumentos necesarios para evaluar el algoritmo mediante el modelo de la rapidez de ejecución en tiempo-fijo.

Los tiempos de ejecución obtenidos en el experimento 3 se pueden observar en la tabla 6.7. Esta tabla muestra resultados similares para la ejecución de problemas con tamaños que se incrementan en la misma tasa con la cual se incrementó la cantidad de paralelismo presente en ellas. Este comportamiento

Tabla 6.6: Parámetros del Experimento 3

	1D	2D
EDP	$\frac{\partial u}{\partial t} + a_x \frac{\partial u}{\partial x} = w \frac{\partial^2 u}{\partial x^2}$	$\frac{\partial u}{\partial t} + a_x \frac{\partial u}{\partial x} + a_y \frac{\partial u}{\partial y} = w(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$
a_x	1.2	1.3
a_y	N/A	0.7
w	0.5	0.2
Condición inicial	$u(x, y, 0) = \exp(-x^2)$	$u(x, y, 0) = \exp(-x^2 - y^2)$
Dominio	$-3 \leq x \leq 12$	$-3 \leq x \leq 12$ $-3 \leq y \leq 12$
Condición inicial	$u(x, 0) = \exp(-x^2)$	$u(x, y, 0) = \exp(-x^2 - y^2)$
Condiciones frontera	$u(-3, t) = 0$ $u(12, t) = 0$	$u(x, -3, t) = 0$ $u(-3, y, t) = 0$ $u(x, 12, t) = 0$ $u(12, y, t) = 0$
Tiempo de simulación	$t_{max} = 8s$ $\Delta t = 0.05$	$t_{max} = 8$ $\Delta t = 0.05$
Tolerancia	$\epsilon = 1 \times 10^{-3}$	$\epsilon = 1 \times 10^{-3}$

se debe a que el método divide en cargas de trabajo de tamaño similar, donde el tamaño de subdominios $|\Omega_i| = N_{ini}$ siempre sera el mismo.

Mediante los datos obtenidos en el Experimento 3 se puede ver que el modelo de escalabilidad multinúcleo de tiempo-fijo cumple con el comportamiento esperado para las plataformas multinúcleo. Según Gustafson, las plataformas paralelas de mayor tamaño deben ser aprovechadas para resolver problemas de mayor tamaño o de mayor extensión [Gus88]. En el caso de esta implementación, las plataformas paralelas multinúcleo sirven para resolver el mismo problema en una malla más densa (un problema de mayor precisión).

Tabla 6.7: **Experimento 3:** Tiempo de Ejecución del MDD Tamaño Variable (segundos)

	1D			2D		
Procesadores	$N = 10$	$N = 25$	$N = 50$	$N = 10$	$N = 25$	$N = 50$
1	0.140	0.860	8.524	13.204	146.359	841.469
2	0.112	0.852	8.439	13.172	144.895	833.054
4	0.122	0.843	8.607	13.140	143.446	824.724
8	0.128	0.835	8.521	12.709	142.012	816.477

6.2.2. Modelo de Memoria-Acotada

El avance en la frecuencia de reloj de los procesadores multinúcleo no ha sido el mismo que el avance presentado en el acceso a los datos en paralelo (lectura y escritura) de los procesadores multinúcleo. La diferencia radica en la capacidad de datos que pueda manejar el procesador, y no en la capacidad de los cálculos que pueda realizar. Para muchas aplicaciones, tales como meta-tareas, cómputo de alto desempeño, o aplicaciones perfectamente paralelizables, la porción secuencial de la carga de trabajo paralela no es el factor limitante del desempeño.

El modelo de la rapidez en memoria-acotada proporciona una cota superior del desempeño, donde todos los datos pueden ser almacenados en las memorias *cache* **L1**. Para cualquier aplicación con tamaño razonable, los datos pueden ser accedidos mediante la memoria jerarquica (ver figura 4.6), donde ocurre un retraso grande para el acceso de los datos, a esto se le conoce como el problema de **barrera de memoria** (*memory-wall problem*), esto aunado al retraso en el contenido compartido por el *cache* **L2** y los *datapaths* (conjunto de unidades funcionales que desempeñan el procesamiento de datos) hacia niveles más bajos en la jerarquía de memoria hacen mas lento este acceso. Este problema se debe a la disparidad en los avances tecnológicos entre la velocidad del CPU y la

latencia en el acceso a datos.

El experimento 3 utiliza el algoritmo diseñado con diferentes tamaños de problema, estos tamaños variaron desde la solución en 1D con $N_{ini} = 10$ en la cual la representación de las matrices de tamaño 10×10. Estas soluciones tienen que ser implementadas en números de precisión doble y requieren de 1.6 MB almacenamiento. En la solución 2D se inició con un tamaño de $N_{ini} = 10$ y las matrices representan la malla en 2D siendo de 100×100, la cual necesita un tamaño de 160MB. La plataforma de ejecución constituida por un procesador Xeon cuenta con un tamaño de *cache* **L1** de 32 KB por núcleo, *cache* **L2** de 256 KB por núcleo y *cache* **L3** de 24 MB para todos los núcleos, es evidente que las matrices de almacenamiento deben ser accedidas, en parte, desde memoria principal. Al incrementarse el tamaño de problema no se experimentaron retrasos en la ejecución debidos a la memoria-acotada, brindando una prueba más de la escalabilidad de las plataformas multinúcleo.

6.3. Discusión Sobre los Resultados

Sun y Chen estudiaron la escalabilidad de las arquitecturas multinúcleo, basándose en el concepto de computación escalable donde el problema puede incrementarse junto con el poder de cómputo (el número de núcleos), y derivaron en 3 modelos de desempeño: tamaño-fijo, tiempo-fijo y memoria-acotada. Estos tres modelos fueron abordados en nuestro estudio [SC10].

La historia se repite nuevamente. Hace dos décadas se debatió el futuro de la escalabilidad de las plataformas paralelas gracias al debate que se inició con la ley de Amdahl, y ahora se repite pero para las plataformas multinúcleo. La escalabilidad de las plataformas multinúcleo es una área vasta pero altamente inexplorada y aquí se presentó una discusión en esa dirección, brindando evidencia teórica y experimental apoyando la postura de Sun y Chen

El modelo pesimista presume que la paralelización de las cargas de trabajo no obtendrá un incremento en su rapidez de ejecución deseable, y que lo limitará la ley de Amdahl. Esto es cierto, pero es una verdad incompleta. La ley de Amdahl brinda las cotas superiores que podrán obtenerse en la rapidez de ejecución de una aplicación de tamaño fijo, pero no define a las arquitecturas multinúcleo completamente. Este modelo representa el desempeño de una plataforma respecto un problema determinado con diferente número de procesadores. El modelo optimista, que concuerda con la ley de Gustafson, representa a todo un conjunto de problemas en donde el tamaño de éste se incrementa con alguna relación al número de procesadores que se añaden. Este modelo muestra que no existe alguna restricción innata para la escalabilidad.

El experimento 2 muestra datos que sustentan que la implementación multinúcleo para un problema de tamaño fijo cumple la ley de Amdhal, obteniendo una rapidez de ejecución que se incrementa con el número de núcleos pero que a la larga tenderá a estancarse. El experimento 3 proporciona datos que sustentan que la implementación del algoritmo puede ser escalada junto con el número de procesadores y ser resueltos en un tiempo que no tiende a crecer. Esto indica que las arquitecturas multinúcleo son capaces de resolver problemas de mayor tamaño y seguir siendo escalable.

6.4. Resumen del Capítulo

En este capítulo se mostraron mediante las medidas de desempeño paralelo la evaluación general de la implementación de método de descomposición de dominio paralelo. También se evaluó la escalabilidad que posee el algoritmo dentro de las plataformas multinúcleo. Se mostraron los modelos existentes de la escalabilidad multinúcleo, y finalmente se hicieron interpretaciones que se pueden inferir sobre estas evaluaciones.

Capítulo 7

Conclusiones

En esta sección se presentan las conclusiones de este trabajo de tesis así como el posible trabajo futuro a desarrollar.

7.1. Plataformas multinúcleo

A lo largo del documento se presentaron los fundamentos teóricos y prácticos para sostener y defender un estudio sobre la escalabilidad multinúcleo mediante un Metodo de Descomposición de Dominio llamado Schwarz Aditivo. El estudio defiende la escalabilidad de las plataformas multinúcleo mediante una implementación de la solución de un problema científico, sirviendo como evidencia de que las plataformas multinúcleo son capaces de resolver este tipo de problemas a la medida de sus capacidades y poder servir como herramientas poderosas en un futuro.

La arquitectura de las plataformas multinúcleo ha evolucionado de tal forma que las computadoras actuales son capaces de realizar tareas que antes solo se hubieran vislumbrado en centros de cómputo intensivo. Los programadores necesitan evolucionar ahora para poder realizar todas sus aplicaciones idóneas

a las plataformas actuales y emergentes, es decir, poder sacarle jugo a toda la poderosa herramienta que se tiene ahora y se tendrá en un futuro.

El mundo del paralelismo multinúcleo es un área bastamente inexplorada, que necesita más estudio, y tal como fue en esta tesis, brindar un granito de arena para apoyar los modelos que regirán los modelos computacionales del futuro.

7.2. Aportaciones

Los experimentos que se desarrollaron durante esta investigación dan evidencia de que los modelos presentados por Sun y Chen son adecuados a las tecnologías multinúcleo actuales. Por otra parte, que las aparentes limitaciones intrínsecas de los procesadores multinúcleo, que aparentemente han sido las que han alentado el auge de esta tecnología, no son válidas para un área específica de la computación, ésta es el cómputo científico. De la misma manera, se puede afirmar que la hipótesis de esta investigación que asegura que mediante técnicas de programación multinúcleo, las computadoras actuales pueden brindar escalabilidad suficiente a los programas paralelos, como para poder ejecutar problemas científicos de tamaño moderadamente grande, aun si en ellos existen cuellos de botella en la transferencia de datos, y se confirmó de manera exitosa.

Durante el desarrollo de esta investigación surgieron algunas limitantes con respecto al acceso a las plataformas de cómputo necesarias para poder ejecutar los experimentos numéricos. Actualmente la mayor parte de las computadoras personales son de tecnología multinúcleo, teniendo como máximo entre 4 y 6 núcleos físicos disponibles. Para poder tener acceso a un procesador con un mayor número de núcleos se requiere un equipo de cómputo de alto desempeño. Esta limitante es tanto un aspecto negativo como positivo, ya que esta dificultad de conseguir los equipos de cómputo con múltiples núcleos es parte del

estigma que múltiples trabajos, junto con éste, esperan terminar. De la misma manera, grandes fabricantes de procesadores personales han anunciado que en los próximos años saldrán al mercado procesadores de 8 y 12 núcleos.

Los expertos en cómputo de alto rendimiento han aprendido a lidiar con estas arquitecturas, pero representan sólo una fracción de los programadores. Cuando, en un futuro no muy lejano, los procesadores multinúcleo desplacen a los normales, todos los programadores tendrán que adaptarse a ellos. Los ordenadores multinúcleo requieren que los programas actúen en paralelo porque cada núcleo, debe obtener su propio juego de instrucciones. Actualmente, la mayor parte del software disponible no está escrito para aprovechar la computación multinúcleo.

7.3. Trabajo futuro

Pensando en un futuro, la labor por realizar siguiendo esta línea de trabajo es la implementación de algoritmos en sistemas de procesamiento multinucleo especializados como **GPU**s (Unidades de Procesamiento Gráfico), o sistemas en sistemas multiprocesador. Esto brindaría mayor evidencia al modelo que establece el problema de memoria y permitirá manejar problemas de mayor tamaño que los usados actualmente.

Específicamente se buscará una implementación mediante la plataforma **CUDA** de **NVIDIA**. CUDA es la arquitectura de cómputo paralelo de los GPUs que se ha extendido más allá de aplicaciones en gráficas, para cómputo paralelo de propósito general. La capacidad de las tarjetas GPU de última generación, permite lograr rendimientos de inclusive de 100 veces comparado con procesadores convencionales, y son una de las opciones más efectivas para montar plataformas cómputo de alto desempeño en la actualidad.

Un posible punto de comparación se buscara mediante una implementación

del algoritmo con una plataforma específica a la ejecución de la aplicación. Esta plataforma se implementará mediante tarjetas FPGAs, en las cuales se puede generar un diseño especifico a la aplicación aprovechando al máximo el paralelismo del algoritmo.

Apéndice A

Acrónimos

ALU	*Arithmetic Logic Unit*
API	*Application Programming Interface*
BLAS	*Basic Linear Algebra Subprograms*
DMM	*Distributed Memory Machine*
EDP	*Ecuación Diferencial Parcial*
EF	*Elemento Finito*
FPU	*Floating Point Unit*
LAPack	*Linear Algebra Package*
MDD	*Métodos de Descomposición de Dominio*
MIMD	*Multiple-Instruction, Multiple-Data*
MISD	*Multiple-Instruction, Single-Data*
PETSc	*Portable, Extensible Toolkit for Scientific Computation*
PJDPack	*Parallel Jacobi-Davidson Package*
PLTMG	*PiecewiseLinear Triangle Multi-Grid*
PMD	*Parallel Multi-Domain Decomposition*
PRAM	*Parallel Random Access Machine*

RAM	*Random Access Machine*
SIMD	*Single-Instruction, Multiple-Data*
SISD	*Single-Instruction, Single-Data*
SMM	Shared Memory Machines
SMP	Symmetric Multi-Processors
SMT	Simultaneous Multi-Threading

Apéndice B

Símbolos

Ω	Dominio completo
Ω_i	Subdominio i
$\partial\Omega_i$	Frontera del subdominio i
Δ	Operador Laplaciano
Γ_i	Frontera artificial del subdominio i
$\partial\Omega\backslash\Gamma$	Frontera que no forma parte de la frontera artificial del subdominio i
$\|\Omega\|$	Cardinalidad del dominio completo
$\|\Omega_i\|$	Cardinalidad del subdominio i
u	Vector de solución númerica
A	Matriz de coeficientes
B	Vector de evaluación
k	Iteración de la solución del MDD
S_p	Rapidez de ejecución obtenida con p procesadores
E_p	Eficiencia obtenida con p procesadores
T_p	Tiempo de ejecución obtenido con p procesadores

$T*$	Mejor tiempo de ejecución secuencial
f	Fracción del programa que se puede ejecutar de manera paralela en p procesadores
X	*Thread*
a	Vector de velocidades advectivas
w	Factor de difusividad
e^k	Vector de error obtenido en la k-ésima iteración
ϵ	Error mínimo esperado
C	Número de subdominios
N	Tamaño del dominio general Ω (número de nodos)
n	Tamaño de los subdominios Ω_i (número de nodos)
β	Restricción de tiempo en acceso a memoria

Apéndice C

Condiciones de Fronteras

En caso de una ecuación en derivadas parciales tal como:

$$\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} = 0 \tag{C.1}$$

definida en un intervalo de $(0,1) \times (0,1)$, se establece la condición de **frontera de Dirichlet** (o de primer tipo) como un tipo de condición de frontera o de contorno cuandose le especifican los valores de la solución que necesita la frontera del dominio.

Las condiciones de frontera de Dirichlet toman la forma:

$$\begin{cases} u(x,0) = f_0(x) \\ u(0,y) = g_0(y) \\ u(x,1) = f_1(x) \\ u(1,y) = g_1(y) \end{cases} \tag{C.2}$$

donde f_0 y f_1 estan definidas para $0 \leq x \leq 1$, mientras que g_0 y g_1 estan definidas para $0 \leq y \leq 1$.

La condición de **frontera de Neumann** (o de segundo tipo) es un tipo de condición de frontera, cuando en una ecuación diferencial en derivadas parciales, se le especifican los valores de la derivada de una solución tomada sobre

la frontera o contorno del dominio, las condiciones de frontera de Neumann toman la forma:

$$\begin{cases} \frac{\partial u}{dx}(x,0) = f_0(x) \\ \frac{\partial u}{dy}(0,y) = g_0(x) \\ \frac{\partial u}{dx}(x,1) = f_1(x) \\ \frac{\partial u}{dy}(1,y) = g_1(x) \end{cases} \tag{C.3}$$

donde f_0 y f_1 estan definidas para $0 \leq x \leq 1$, mientras que g_0 y g_1 estan definidas para $0 \leq y \leq 1$.

Bibliografía

[AA94] Alvaro A. Aldama and Javier Aparicio, *Numerical simulation of flow in river networks with complex topology*, Computational Methods in Water Resources X (P.O. Box 17, 3300 AA Dordrecht, The Netherlands) (Alexander Peters et al, ed.), Kluwer Academic Publishers, 1994, pp. 1131–1138.

[AfIM99] E. Anderson, Society for Industrial, and Applied Mathematics, *Lapack user'guide*, Society for Industrial and Applied Mathematics, 1999.

[Amd67] GM Amdahl, *Validity of the single processor approach to achieving large scalecomputing capabilities*, Proceedings of the April 18-20, 1967, spring joint computer conference, ACM, 1967, pp. 483–485.

[ASS80] John Adams, PAUL Swarztrauber, and Roland Sweet, *Fishpak: A package of fortran subprograms for the solution of separable elliptic partial differential equations*, The National Center for Atmospheric Research, Boulder, CO, 1980.

[Ban90] Randolph E. Bank, *Pltmg: A software package for solving elliptic partial differential equations. users' guide 7.0*, SIAM, Philadelphia, PA, 1990.

[BC10] A.T. Barker and X.C. Cai, *Scalable parallel methods for monolithic coupling in fluid-structure interaction with application to blood flow modeling*, Journal of Computational Physics **229** (2010), no. 3, 642–659.

[Bjø94] Petter Bjøstad, *Domain decomposition applied to oil reservoir simulation*, Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering (D. Keyes, Y. Saad, and D. Truhlar, eds.), SIAM, 1994, To appear.

[BOK09] Tunc Bahcecioglu, Semih Ozmen, and Ozgur Kurc, *A comparative study on two different direct parallel solution strategies for large-scale problems*, The First International Conference on Parallel, Distributed and Grid Computing for Engineering (Topping, ed.), April 2009.

[BW86] P.E. Bjørstad and O.B. Widlund, *Iterative methods for the solution of elliptic problems on regions partitioned into substructures*, SIAM Journal on Numerical Analysis **23** (1986), no. 6, 1097–1120.

[CFS98] X.-C. Cai, C. Farhat, and M. Sarkis, *A minimum overlap restricted additive Schwarz preconditioner and applications to 3D flow simulations*, Contemporary Mathematics **218** (1998), 479–485.

[Cha01] R. Chandra, *Parallel programming in OpenMP*, Morgan Kaufmann, 2001.

[Con63] M.E. Conway, *A multiprocessor system design*, Proceedings of the November 12-14, 1963, fall joint computer conference, ACM, 1963, pp. 139–146.

[Cro94] L.A. Crowl, *How to measure, present, and compare parallel performance*, IEEE Parallel & Distributed Technology: Systems & Technology **2** (1994), no. 1, 9–25.

[CS95] T.F. Chan and J.P. Shao, *Parallel complexity of domain decomposition methods and optimal coarse grid size*, Parallel Computing **21** (1995), no. 7, 1033–1049.

[CSG99] D.E. Culler, J.P. Singh, and A. Gupta, *Parallel computer architecture: a hardware/software approach*, Morgan Kaufmann Pub, 1999.

[DJK07] I. Danaila, P. Joly, and SM Kaber, *An introduction to scientific computing: twelve computational projects solved with matlab*, Springer Verlag, New York, 2007.

[DW87] Maksymilian Dryja and Olof B. Widlund, *An additive variant of the Schwarz alternating method for the case of many subregions*, Tech. Report 339, also Ultracomputer Note 131, Department of Computer Science, Courant Institute, 1987.

[Fly72] M.J. Flynn, *Some computer organizations and their effectiveness*, Computers, IEEE Transactions on **100** (1972), no. 9, 948–960.

[GDN98] M. Griebel, T. Dornseifer, and T. Neunhoeffer, *Numerical simulation in fluid dynamics: a practical introduction*, Society for Industrial Mathematics, 1998.

[GO92] GH Golub and JM Ortega, *Scientific computing and differential equations: an introduction to numerical methods*, Academic Press, 1992.

[GRD07] G. Galante, R.L. Rizzi, and T.A. Diverio, *A Multigrid-Schwarz Method for the Solution of Hydrodynamics and Heat Transfer*

Problems in Unstructured Meshes, Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on, IEEE, 2007, pp. 87–94.

[Gus88] J.L. Gustafson, *Reevaluating Amdahl's law*, Communications of the ACM **31** (1988), no. 5, 532–533.

[HM08] M. D. Hill and M. R. Marty, *Amdahl's law in the multicore era*, Computer **41** (2008), no. 7, 33–38.

[HPG03] J.L. Hennessy, D.A. Patterson, and D. Goldberg, *Computer architecture: a quantitative approach*, Morgan Kaufmann, 2003.

[HR92] T. Heywood and S. Ranka, *A practical hierarchical model of parallel computation*, Journal of Parallel and Distributed Computing **16** (1992), no. 3, 212–232.

[Hug08] C. Hughes, *Professional multicore programming design and implementation for c++ developers*, Wiley-India, 2008.

[Kai80] T. Kailath, *Linear systems*, vol. 1, Prentice-Hall, NJ, 1980.

[Kel95] C.T. Kelley, *Iterative methods for linear and nonlinear equations*, vol. 16, Society for Industrial Mathematics, 1995.

[Koc05] G Koch, *Discovering multi-core: Extending the benefits of Moore's law*, IntelWhite Paper (2005).

[LHKK79] C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh, *Basic linear algebra subprograms for fortran usage*, ACM Transactions on Mathematical Software (TOMS) **5** (1979), no. 3, 308–323.

[Lio88] PL Lions, *On the Schwarz alternating method I*, Domain decomposition methods for partial differential equations (1988), 1–41.

[Lit98] Luc Litzler (ed.), *Pmd: A parallel fortran 90 module to solve elliptic linear second order equations*, HERMES Science Publications, Paris., December 1998, Calculateurs Paralleles Reseaux et Systemes Repartis.

[MBH+02] D.T. Marr, F. Binns, D.L. Hill, G. Hinton, D.A. Koufaty, J.A. Miller, and M. Upton, *Hyper-threading technology architecture and microarchitecture*, Intel Technology Journal **6** (2002), no. 1, 4–15.

[MU09] AMM Mukaddes and A. Uragami, *Parallel performance of domain decomposition method on distributed computing environment*, Computer and Information Technology, 2008. ICCIT 2008. 11th International Conference on, IEEE, 2009, pp. 617–622.

[Nev39] R. Nevanlinna, *Über das alternierende verfahren von schwarz.*, Journal für die reine und angewandte Mathematik (Crelles Journal) **1939** (1939), no. 180, 121–128.

[Prz63] JS Przemieniecki, *Matrix structural analysis of substructures*, AIAA J **1** (1963), no. 1, 138–147.

[Qia08] J. Qiang, *A Parallel Algorithm Based on Additive Schwarz Domain Decomposition Method for Parabolic Problems*, High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on, IEEE, 2008, pp. 903–906.

[RR10] T. Rauber and G. R
ünger, *Parallel Programming: for Multicore and Cluster Systems*, Springer-Verlag New York Inc, 2010.

[SBG06] A. Sarkar, N. Benabbou, and R. Ghanem, *Domain decomposition of stochastic PDES and its parallel implementation*, 20th International

Symposium on High-Performance Computing in an Advanced Collaborative Environment, 2006, pp. 14–17.

[SC10] Xian-He Sun and Yong Chen, *Reevaluating amdahl's law in the multicore era*, Journal of Parallel and Distributed Computing **70** (2010), no. 2, 183–188.

[Sch69] H.A. Schwarz, *Ueber einige Abbildungsaufgaben.*, Journal f ür die reine und angewandte Mathematik **1869** (1869), no. 70, 105–120.

[Sch90] HA Schwarz, *Gesammelte Mathematische Abhandlungen, Vol. 2, 133–143*, 1890.

[SG94] W.D.G.B.F. Smith and W. Gropp, *Scalable, extensible, and portable numerical libraries*, Proceedings of the Scalable ParallelLibrariesConference, IEEE, 1994, pp. 87–93.

[SL06] RW Shonkwiler and L Lefton, *An introduction to parallel and vector scientific computing*, Cambridge Univ Pr, 2006.

[Smi94] Barry F. Smith, *Extensible PDE solvers package users manual*, Tech. Report ANL-94/40, Argonne National Laboratory, September 1994.

[Sob34] S.L. Sobolev, *On analytic solutions of a system of partial differential equations with two independent variables*, Trudy Matematicheskogo Instituta im. VA Steklova **5** (1934), 265–282.

[SW90] B.F. Smith and O.B. Widlund, *A domain decomposition algorithm using a hierarchical basis*, SIAM Journal on Scientific and Statistical Computing **11** (1990), 1212.

[Swe91] R.A. Sweet, *Vectorization and parallelization of FISHPAK*, Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing, Society for Industrial and Applied Mathematics, 1991, pp. 637–642.

[Wei09] Z.H. Wei, *Parallel Jacobi-Davidson Algorithms and Software Developments for Polynomial Eigenvalue Problems in Quantum Dot Simulation*, Ph.D. thesis, National Central University, 2009.

[ZT00] O.C. Zienkiewicz and R.L. Taylor, *The finite element method: Solid mechanics*, vol. 2, Butterworth-Heinemann, 2000.