Paralelización de la ecuación del transporte en arquitecturas de hardware masivamente paralelas

Tesis de Maestría presentada por
Marcelo Bondarenco

8 de enero de 2018

Universidad de la República
PEDECIBA INFORMÁTICA

PARALELIZACIÓN DE LA ECUACIÓN DEL TRANSPORTE EN ARQUITECTURAS DE HARDWARE MASIVAMENTE PARALELAS

Tesis de Maestría presentada por
MARCELO BONDARENCO

Salto

Director de tesis: Pablo Ezzatti
Co-Director de tesis: Pablo Gamazo

Índice general

Índice de figuras

Índice de tablas

Resumen

En la actualidad los modelos numéricos se han convertido en una herramienta indispensable para la evaluación de modelos conceptuales y para la integración de información de diversa índole en el área de mecánica de los fluidos. El aumento en la velocidad de procesamiento y capacidad de memoria de las computadoras ha sido acompañado por el desarrollo de códigos de modelación que han permitido trabajar con modelos hidrogeológicos cada vez más complejos (y que permiten obtener resultados más precisos). Sin embargo, en estos últimos años los procesadores de computadores (CPUs) han tenido en cuanto a prestaciones un enlentecimiento en su desarrollo en comparación a la tasa de años anteriores, principalmente debido a los límites físicos de los componentes microelectrónicos basados en semiconductores. En contrapartida, las tarjetas gráficas (GPUs, por sus siglas en inglés) surgieron como un recurso computacional de alto desempeño alternativo, y de mucho menor costo que el procesamiento paralelo tradicional (basado en cluster de computadoras). Es así que se ha popularizado en el ámbito científico el uso de placas gráficas, las cuales otorgan una gran aceleración incluso a computadoras de escritorio mediante la ejecución de cálculos en paralelo, permitiendo considerar así dominios de simulación más grandes y velocidades de simulación mayores. Además, este tipo de plataformas superan en general a las CPUs en la relación consumo energético versus desempeño computacional que ofrecen.

Los cambios en los paradigmas de ejecución, mencionados anteriormente, motivan el estudio de las diferentes técnicas numéricas para evaluar como se adaptan al uso de este tipo de plataformas de hardware modernas. Un estudio en amplitud de todas los términos numéricos obviamente es inabordable, aun cuando el estudio se centrara en los métodos presentes en la hidrología. Por lo tanto, en este trabajo se aborda como caso de estudio la ecuación del transporte (advectiva-difusiva). Los fenómenos de transporte son aquellos procesos en los que hay una transferencia neta o transporte de materia, energía o momento lineal en cantidades grandes o microscópicas. El fenómeno de transporte en fluidos tiene lugar cuando una sustancia está disuelta en otra, por ejemplo, como cuando una sal o un contaminante está disuelto en agua.

La modelación de la ecuación de transporte determina un sistemas de ecuaciones en derivadas parciales. Uno de los procedimientos ampliamente usado para resolver sistemas de ecuaciones en derivadas parciales es el método de las Diferencias Finitas. Aunque existen otros métodos numéricos, como pueden ser Elementos Finitos o Volúmenes Finitos, en este trabajo se utiliza el método de las Diferencias Finitas ya que es una técnica de carácter general, con amplia literatura y bastante sencilla de paralelizar. Dentro de las Diferencias Finitas se puede optar por trabajar con un esquema Implícito, más exacto y complejo de implementar, o un

esquema Explícito, más sencillo de implementar pero en general con menos precisión en los resultados que las variantes con el esquema anterior, o un esquema mixto como Crank-Nicolson. Para el caso de los esquemas Implícito y mixto, como parte del procedimiento es necesario resolver sistemas lineales de ecuaciones. Aprovechando que dichos sistemas de ecuaciones generan matrices penta o heptadiagonales (según el dominio del caso abordado, bidimensional o tridimensional respectivamente), en este trabajo se estudiaron dos solvers para resolver sistemas lineales dispersos. Por un lado, se utilizó el método SIP (*Strongly Implicit Procedure*) diseñado especialmente para resolver sistemas lineales con matrices penta y heptadiagonales presentes en problema de mecánica de los fluidos. Por otro lado, se evaluó el método iterativo general de resolución de sistemas lineales Gradiente Bi-Conjugado Estabilizado (BiCGStab).

El objetivo general del presente trabajo es evaluar diferentes esquemas numéricos para la resolución de la ecuación de transporte en 3D que incluyan estrategias de computación de alto desempeño sobre procesadores multi-core y plataformas de hardware masivamente paralelas, por ejemplo GPUs.

Uno de los principales aspectos distintivos del presente trabajo es que se estudia en forma acoplada la precisión numérica y el desempeño computacional del esquema Implícito, Explícito y mixto para la resolución de la ecuación de transporte mediante la comparación con soluciones analíticas y también la ponderación de la escalabilidad computaciónal de cada paradigma. Si bien se utiliza la ecuación del transporte como objeto de estudio se espera alcanzar conclusiones extrapolables a los métodos numéricos en general.

Capítulo 1

Introducción

La necesidad del hombre de entender su entorno natural y anticiparse a los acontecimientos tiene raíces en que, en gran medida, de ello depende su supervivencia. Es así que hoy en día se utiliza el método científico como el medio más efectivo para predecir el comportamiento de la naturaleza. En la actualidad, cuando se desea predecir el comportamiento de un sistema natural, los conocimientos científicos y tecnológicos se integran para definir primero un modelo físico. Luego, se convierte dicho modelo a un problema matemático, y de éstos solo algunos pocos pueden ser resueltos mediante técnicas analíticas. Es decir, en la gran mayoría de los casos se necesitan aproximaciones numéricas y, más específicamente, cuando se consideran problemas de grandes dimensiones, o con alto niveles de precisión, mediante el uso de programas que son ejecutados por computadoras.

Hace más de cuatro décadas que los modelos matemáticos vienen siendo utilizados en el campo de la hidrología, ver por ejemplo el trabajo de McDonald y Harbaugh [46]. En la actualidad los modelos numéricos se han convertido en una herramienta indispensable para la evaluación de modelos conceptuales y para la integración de información de diversa índole en esta área. El aumento en la velocidad de procesamiento y capacidad de memoria de las computadoras ha sido acompañado por el desarrollo de códigos de modelación que han permitido trabajar con modelos hidrogeológicos cada vez más complejos (y que permiten obtener resultados más precisos). Sin embargo, en estos últimos años los procesadores de computadores (CPUs) han tenido en cuanto a prestaciones un enlentecimiento en su desarrollo en comparación a la tasa de años anteriores, principalmente debido a los límites físicos de los componentes microelectrónicos basados en semiconductores [30]. En contrapartida, las tarjetas gráficas[1] (GPUs, por sus siglas en inglés) surgieron como un recurso computacional de alto desempeño alternativo, y de mucho menor costo que el procesamiento paralelo tradicional (basado en cluster de computadoras). Es así que se ha popularizado en el ámbito científico el uso de placas gráficas, las cuales otorgan una gran aceleración incluso a computadoras de escritorio mediante la ejecución de cálculos en paralelo, permitiendo considerar así dominios de simulación más grandes y velocidades de simulación mayores. Además, este tipo de plataformas superan en general a las CPUs en la relación consumo energético versus

[1] Existen otros aceleradores de Hardware, por ejemplo los procesadores Intel Xeon Phi, sin embargo las GPUs son una arquitectura más disruptiva.

desempeño computacional que ofrecen. El paralelismo brindado por las GPUs se basa en la ejecución de miles de hilos procesando las mismas instrucciones sobre diferentes datos. Este paradigma conocido como SIMD (del inglés Single Instruction, Multiple Data según la taxonomía de Flynn [23]) en el caso de las GPUs de NVIDIA se estila nombrar como SIMT (del inglés Single Instruction, Multiple Thread) una variante de la categoría anterior definida por la empresa NVIDIA y centrada en la ejecución múltiples hilos de cómputo.

Los cambios en los paradigmas de ejecución, mencionados anteriormente, motivan el estudio de las diferentes técnicas numéricas para evaluar como se adaptan al uso de este tipo de plataformas de hardware modernas. Un estudio en amplitud de todas los términos numéricos obviamente es inabordable, aun cuando el estudio se centrara en los métodos presentes en la hidrología. Por lo tanto, en este trabajo se aborda como caso de estudio la ecuación del transporte (advectiva-difusiva) [4]. Los fenómenos de transporte son aquellos procesos en los que hay una transferencia neta o transporte de materia, energía o momento lineal en cantidades grandes o microscópicas. El fenómeno de transporte en fluidos tiene lugar cuando una sustancia está disuelta en otra, por ejemplo, como cuando una sal o un contaminante está disuelto en agua. En este caso, se le llama soluto a la sustancia disuelta y solvente al medio en que se encuentra aquella. Al movimiento del soluto se le llama transporte. Cuando se estudia el transporte, es habitual llamar concentración a la masa del soluto por unidad de volumen del solvente. En esa clase de estudios, la concentración es función tanto de la posición como del tiempo y la ecuación del transporte describe como varía ésta en el espacio y el tiempo. Si bien la ecuación de transporte es una de las más importantes ecuaciones en el campo de la mecánica de los fluidos, se la puede encontrar en varias disciplinas de la ciencia. En particular, dicha ecuación gobierna el transporte de materia, energía, cargas eléctricas y cantidad de movimiento, y es usada en disciplinas como por ejemplo biología [57], estudios atmosféricos [76], medicina [79], química [71], o en física como es el caso de este trabajo. Desde el punto de vista matemático dicha ecuación es una ecuación en derivadas parciales o simplemente EDP.

Uno de los métodos ampliamente usado para resolver sistemas de ecuaciones en derivadas parciales es el método de las Diferencias Finitas [65]. Aunque existen otros métodos numéricos, como pueden ser Elementos Finitos [18] o Volúmenes Finitos [20], en este trabajo se utiliza el método de las Diferencias Finitas ya que es una técnica de carácter general, con amplia literatura y bastante sencilla de paralelizar. Dentro de las Diferencias Finitas se puede optar por trabajar con un esquema Implícito, más exacto y complejo de implementar, o un esquema Explícito, más sencillo de implementar pero en general con menos precisión en los resultados que las variantes con el esquema anterior, o un esquema mixto como Crank-Nicolson. Para el caso de los esquemas Implícito y mixto, como parte del procedimiento es necesario resolver sistemas lineales de ecuaciones.

Aprovechando que los sistemas lineales de ecuaciones antes mencionados son definidos por[2] matrices penta o heptadiagonales (según el dominio del caso abordado, bidimensional o tridimensional respectivamente), en este trabajo se estudiaron dos solvers para resolver sistemas lineales dispersos. Por un lado, se utilizó el método SIP (*Strongly Implicit Procedure*) [68] diseñado especialmente para resolver sistemas lineales con matrices penta y heptadiagonales presentes en problema de mecánica de los fluidos. Por otro lado, se evaluó el método iterativo

[2]En el caso de usar grillas uniformes.

general de resolución de sistemas lineales Gradiente Bi-Conjugado Estabilizado (BiCGStab).

La resolución de la ecuación del transporte utilizando GPUs ha sido abordada por otros autores, por ejemplo, Goncalves [8] estudió el problema evaluando diferentes métodos para la resolución de sistemas lineales en 1D, Cotronis et al. [13] aplicaron el método de las sobre-relajaciones sucesivas (SOR por su siglas en inglés), y Molnár [49] estudió la resolución de la ecuación difusiva-reactiva. Sin embargo, estos trabajos se centran en avanzar en implementaciones eficientes de ciertos métodos en GPU, no ofreciendo conclusiones basadas en una evaluación multicriterio que incluya precisión numérica, desempeño, esquema numérico y plataforma de hardware utilizado.

1.1. Objetivos generales y específicos

El objetivo general del presente trabajo es evaluar diferentes esquemas numéricos para la resolución de la ecuación de transporte en 3D que incluyan estrategias de computación de alto desempeño sobre procesadores multi-core y plataformas de hardware masivamente paralelas, por ejemplo GPUs. Uno de los principales aspectos distintivos del presente trabajo es que se estudia en forma acoplada la precisión numérica y el desempeño computacional de los esquemas Implícito, Explícito y mixto para la resolución de la ecuación de transporte mediante la comparación con soluciones analíticas y también la ponderación de la escalabilidad computaciónal de cada paradigma. Si bien se utiliza la ecuación del transporte como objeto principal de estudio se espera alcanzar conclusiones extensible a los métodos numéricos en general.

Por lo tanto, como objetivo específico de esta tesis se busca determinar cuando es más conveniente usar un esquema númerico u otro, en base a la cuantificación de los errores de precisión y los tiempos de ejecución insumidos por cada uno de los paradigmas y según el hardware utilizado.

En forma adicional, y considerando el objetivo general del trabajo, es necesario alcanzar implementaciones eficientes de los distintos esquemas cuando son ejecutados sobre las diferentes plataformas de hardware abordadas. Esto implica un manejo conceptual de los diferentes recursos de hardware abordados así como las técnicas empleadas para acelerar algoritmos sobre ellos.

1.2. Estructura de la Tesis

La estructura del resto del documento se detalla a continuación.

En el Capítulo 2 se explica el fenómeno del transporte en fluidos. se deduce la ecuación que lo representa y se muestran las restricciones sobre la modelación de dicho fenómeno. También en este capítulo, se describe la discretización de ecuaciones en derivadas parciales mediante el método de las diferencias finitas y los diversos esquemas posibles de aplicar sobre

éstas. Por último, se explica el Strong Implicit Procedure (SIP) y el BiCGStab, que son los solvers elegidos para la resolución de los sistemas lineales subyacente del uso del esquema Implícito para la descretización de las ecuaciones en derivadas parciales.

El Capítulo 3 detalla algunas características y presenta la evolución histórica de las arquitecturas de hardware que incluyen múltiples unidades de cómputo y, con particular atención las arquitecturas masivamente paralelas como son las GPUs.

En el Capítulo 4 se describe el diseño e implementación de las propuestas de paralelización para los diferentes métodos y arquitecturas abordadas. Se hace especial foco en la descripción del uso de las GPUs para acelerar los cómputos en los distintos métodos abordados.

Luego, en el Capítulo 5, se presenta la evaluación experimental realizada para estudiar cada método implementado. También se interpretan los resultado obtenidos tanto desde el punto de vista de calidad numérica como de desempeño computacional y se compararan entre ellos y con los ofrecidos por la literatura especializada.

Finalmente, en el Capítulo 6, se resumen las principales conclusiones arribadas durante la ejecución de este trabajo, se enumeran los resultados conseguidos en cuanto a reportes de divulgación científica y se describen las principales líneas abiertas plausibles de abordar en el futuro cercano.

Capítulo 2

La resolución de la ecuación del transporte

Los modelos matemáticos de los sistemas continuos están constituidos por balances de propiedades extensivas, por ejemplo los modelos de transporte de solutos (los contaminantes transportados por corrientes superficiales o subterráneas, son un caso particular de estos procesos de transporte). Dichos modelos se construyen haciendo el balance de la masa del soluto que hay en cualquier parte del espacio físico. Existen principalmente dos tipos de procesos físicos por los cuales los elementos químicos son transportados mediante fluidos en el medio ambiente; advección y difusión.

- **Advección:** Este proceso se debe al movimiento del fluido, como puede ser aire o agua. Se está frente a este proceso cuando el soluto es arrastrado pasivamente por el movimiento del fluido en el cual se encuentra disuelto. El movimiento advectivo es descrito matemáticamente por la dirección y la magnitud de su velocidad, siempre y cuando no se produzca absorción y/o retardo. La tasa a la cual el soluto es transportado por unidad de área (perpendicular a la dirección del movimiento) se expresa generalmente en términos de densidad de flujo ($\mathbf{j}$) de acuerdo a la siguiente expresión: $\mathbf{j} = c \cdot \mathbf{v}$, donde $\mathbf{j}$ es la densidad de flujo expresado en [M/(L^{2}T)], c es la concentración del soluto M/L^3 y $\mathbf{v}$ es la velocidad del fluido $[L/T]$, (siendo M masa, L longitud y T tiempo).

- **Difusión:** En este segundo tipo de proceso, el soluto se mueve desde un lugar donde su concentración es relativamente alta hacia otro donde es menor, por efecto de un movimiento aleatorio de las moléculas (difusión molecular). La ley de Fick [34] es usada para describir la densidad de flujo debido a la difusión turbulenta, y se expresa para una dimensión como $\boldsymbol{j} = -\boldsymbol{D}(dc/dx)$, donde $\boldsymbol{j}$ es la densidad de flujo [M/(L^{2}T)], D es el coeficiente de difusión [L^2/T], c es la concentración del elemento o compuesto químico [M/L^3] y x es la distancia sobre la cual se consideran cambios en la concentración [L]. La

ley de Fick puede ser también expresada en tres dimensiones usando notación vectorial en la forma $\mathbf{j} = -\mathbf{D} \cdot \nabla c$, donde ∇ es el operador gradiente y $\mathbf{D}$ indica el tensor de difusión.

- **Dispersión:** Las variaciones en la velocidad a escalas menores que la de trabajo también inducen un flujo de masa que se denomina dispersión. Dicho efecto se describe también con la Ley de Fick, añadiendo al tensor de difusión un componente dispersivo que depende del campo de velocidades. En el presente estudio se considerará un tensor de difusión constante.

Deducción de la Ecuación de Transporte:

Como ya fue mencionado, en la naturaleza el transporte de un soluto se describe a través de la combinación de los procesos de advección y de difusión. Considerando el balance de masa, y que se puede trabajar ambos procesos en forma independiente, se puede escribir la ecuación del transporte como el resultado de la suma de un término advectivo más otro término difusivo-dispersivo, ver la Ecuación 2.1, donde Vol_{Cont} indica el Volumen de Control.

$$\Delta_{mv} = \sum \text{Masa que entra al } Vol_{Cont} - \sum \text{Masa que sale del } Vol_{Cont}. \tag{2.1}$$

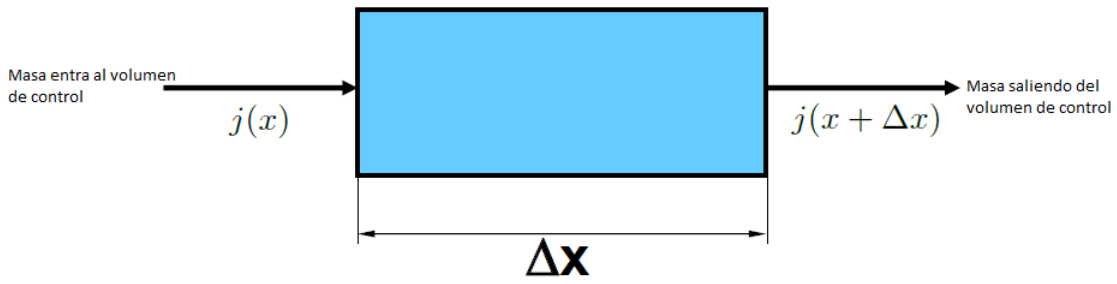

Figura 2.1: Volumen de Control (Vol_{Cont}).

Entonces, si se llama V al volumen [L^3], A al área [L^2], $\mathbf{j}$ al flujo másico [$M/(L^2T)$] y la variación de la concentración por unidad de tiempo[$\frac{c(x+\Delta t)-c(x)}{\Delta t}$] [$M/(L^3T)$], se puede comenzar a escribir la ecuación como la diferencia de masa que entra al volumen de control menos la masa que sale por unidad de tiempo:

$$V \cdot (\frac{c(t + \Delta t) - c(t)}{\Delta t}) = A \cdot j(x) - A \cdot j(x + \Delta x). \tag{2.2}$$

Observando que, V/A = Δx, dividiendo por A y reagrupando términos se obtiene,

$$\frac{c(t+\Delta t)-c(t)}{\Delta t} = \frac{j(x)-j(x+\Delta x)}{\Delta x}. \tag{2.3}$$

Si se se toma Δx → 0 y Δt → 0

$$\lim_{\Delta t \to 0} \frac{c(t+\Delta t)-c(t)}{\Delta t} = \frac{\delta c}{\delta t}. \tag{2.4}$$

Así se obtiene la formulación de la ecuación del transporte en una única dimensión para su forma más general,

$$\frac{\delta c}{\delta t} = -\frac{\delta j}{\delta x}. \tag{2.5}$$

En este contexto, el flujo puede ser advectivo y/o difusivo. A continuación se desarrollará en primer lugar la expresión para el flujo advectivo másico y luego el difusivo. Se considerará el transporte de partículas como análogo al transporte de solutos. En la Figura 2.2 se tienen dos volúmenes de control llamados I y II, y se evalúa el pasajes de partículas desde el vólumen I al II.

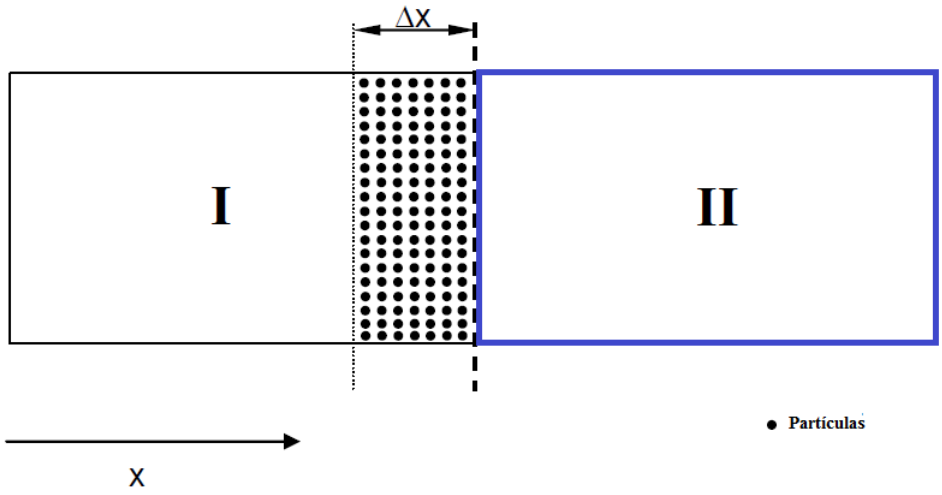

Figura 2.2: Flujo advectivo.

Asumiendo que las partículas solo van en el sentido positivo respecto de x, Δx es la distancia que cubre una partícula en un tiempo Δt. El número de partículas que se mueven desde el volumen I al II en un intervalo de tiempo Δt se puede calcular usando la siguiente fórmula:

$$N = c \cdot \Delta x \cdot A, \tag{2.6}$$

donde N [M] es el número de partículas (análogo a masa) que se mueven del volumen I al II en un intervalo de tiempo Δt, y c [M/L^3] es la concentración de las partículas disuelta en

el volumen I. Entonces, si se divide el resultado anterior (Ecuación 2.6) entre Δt, se obtiene cuantas partículas pasan de un volumen a otro en una unidad de tiempo Δt, y esta expresión se resume en la Ecuación 2.7.

$$Q = \frac{N}{\Delta t} \tag{2.7}$$

Ahora, dividiendo por el área se obtendrá el número de partículas que pasan de un volumen a otro por unidad de tiempo y unidad de área, es decir se obtendrá el flujo, como se expresa en las ecuaciones 2.8, 2.9, y 2.10.

$$\frac{Q}{A} = \frac{\Delta x}{\Delta t} \cdot c = j_{advectivo} \tag{2.8}$$

$$j_{advectivo} = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t} \cdot c = \frac{\delta x}{\delta t} \cdot c \tag{2.9}$$

$$j_{advectivo} = \frac{\delta x}{\delta t} \cdot c \tag{2.10}$$

Llamando **v** a la velocidad,

$$\mathbf{v} = \frac{\delta x}{\delta t}, \tag{2.11}$$

$$j_{advectivo} = \mathbf{v} \cdot c. \tag{2.12}$$

Para determinar el flujo difusivo, se aplica la ley de Fick que determina el flujo de masa entre los volúmenes I y II.

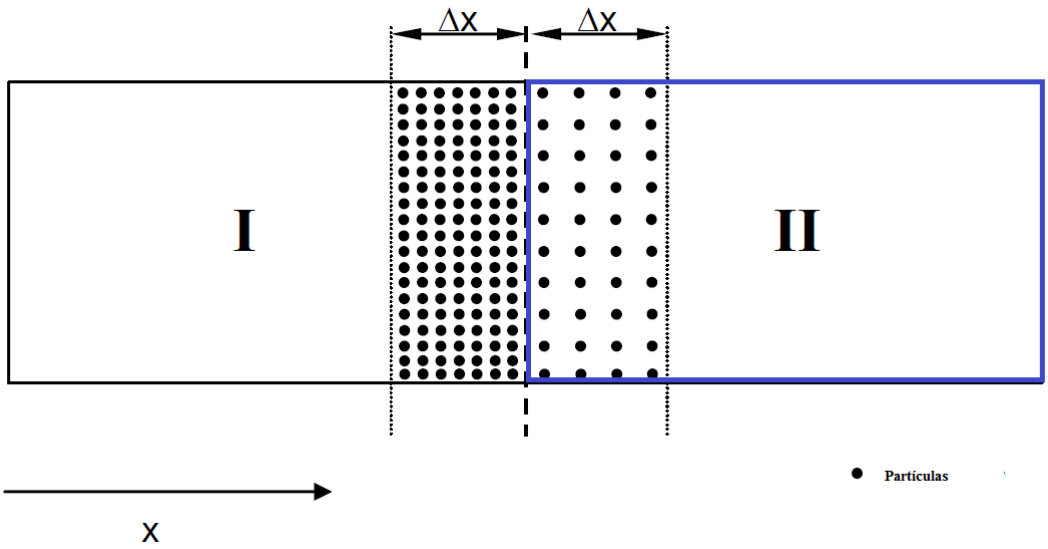

Figura 2.3: Flujo difusivo, las partículas se mueven desde zonas de alta concentración (volumen I), a zonas de bajas concentración (volumen II).

De esta manera el flujo difusivo estará definido de la siguiente manera:

$$j_{difusivo} = -\mathbf{D} \cdot \frac{\delta c}{\delta x}. \tag{2.13}$$

Como ya se mostró anteriormente, la ecuación del transporte tiene la siguiente forma:

$$\frac{\delta c}{\delta t} = -\frac{\delta j}{\delta x}, \tag{2.14}$$

donde j = $j_{advectivo}$ + $j_{difusivo}$ y, sustituyendo en la ecuación anterior j por la suma de $j_{advectivo}$ y $j_{difusivo}$ se obtiene la siguiente ecuación:

$$\frac{\delta c}{\delta t} = -\delta \frac{-\mathbf{D} \cdot \frac{\delta c}{\delta x}}{\delta x} - \mathbf{v} \cdot \frac{\delta c}{\delta x}. \tag{2.15}$$

Considerando **D** constante:

$$-\frac{-\mathbf{D} \cdot \frac{\delta c}{\delta x}}{\delta x} = \mathbf{D} \cdot \frac{\delta^2 c}{\delta x^2}, \tag{2.16}$$

se alcanza la ecuación de transporte en el espacio (ver Ecuación 2.17). Matemáticamente, como se dijo antes, es un sistema de ecuaciones en derivadas parciales con términos parabólicos e hiperbólicos.

$$\frac{\delta c}{\delta t} = \mathbf{D} \cdot \frac{\delta^2 c}{\delta x^2} - \mathbf{v} \cdot \frac{\delta c}{\delta x} \tag{2.17}$$

Números adimensionados, Número de Fourier y Péclet

Un número adimensional es un número que no tiene unidades físicas que lo definan y por lo tanto es un número puro. Los números adimensionales se definen como productos o cocientes de cantidades que sí tienen unidades, de tal forma que todas éstas se simplifican. Dependiendo de su valor estos números tienen un significado físico que caracteriza determinadas propiedades para algunos sistemas.

Por ejemplo el número de Fourier [58], F_0, es un número adimensional que representa la relación entre la propagación por difusión y la capacidad de almacenamiento[1]:

$$F_0 = \frac{\mathbf{D} \cdot T}{L^2}, \tag{2.18}$$

donde **D** es el coeficiente de difusión, T el tiempo característico, y L es la longitud característica.

El comportamiento de un fluido queda fuertemente determinado por el valor de este número, entonces:

- Si $F_0 > 1$ la variación de almacenamiento por difusión en la longitud característica se desarrolla en un tiempo menor al característico. Por lo tanto es de esperar que si existe una diferencia de concentraciones en una longitud L, se establezca un flujo difusivo significativo en un tiempo T.

- Si $F_0 < 1$ la variación de almacenamiento por difusión en la longitud característica se desarrolla en un tiempo mayor al característico. No se aprecia un flujo difusivo significativo ya que en una longitud superior a L, la mayor parte de la masa se está almacenando en la longitud L.

Por otro lado, en mecánica de fluidos el número de Péclet [60], P_e, es un número adimensional que relaciona la velocidad de advección de un flujo **v** y la velocidad de difusión, según la siguiente ecuación:

$$P_e = \frac{\mathbf{v} \cdot L}{\mathbf{D}}. \tag{2.19}$$

Nuevamente, este número caracteriza el comportamiento del fluido:

- Si $P_e > 1$ domina el fenómeno de advección.

[1] Es la capacidad que tiene un material de almacenar cierta cantidad de una sustancia.

- Si $P_e < 1$ domina el fenómeno de difusión.

Es importante mencionar que, los valores que toman las ecuaciones 2.18 y 2.19 influyen en la estabilidad numérica de los diferentes métodos de resolución [21].

2.1. Diferencias Finitas

Cuando se intenta modelar un determinado fenómeno físico, por lo general se plantea un sistema de ecuaciones diferenciales (ordinarias o) parciales, válidas para determinada región (o dominio), al cual se le imponen condiciones de borde e iniciales apropiadas. En esta etapa, el modelo matemático está completo, y es aquí donde aparece la mayor dificultad, dado que solamente las formas más simple de las ecuaciones, con fronteras geométricamente triviales son capaces de ser resuelta en forma exacta (analítica) con los métodos matemáticos disponibles. Uno de los pocos ejemplos para los cuales se dispone de procedimientos matemáticos clásicos de resolución son las ecuaciones diferenciales ordinarias con coeficientes constantes. Por lo tanto, y con el fin de evitar tales dificultades y lograr resolver el problema con la ayuda de computadoras, es necesario presentar el problema de una manera puramente algebraica. Entre los diferentes métodos que permiten representar el problema en forma algebraica, uno de los más sencillos es el método de las Diferencias Finitas [65].

El método de las Diferencias Finitas es un método de carácter general que permite la resolución aproximada de ecuaciones diferenciales en derivadas parciales definidas en recintos finitos. Mediante un proceso de discretización, el conjunto infinito de números que representan la función o funciones incógnitas en el continuo, es reemplazado por un número finito de parámetros incógnita, y este proceso requiere alguna forma de aproximación.

Si se desea determinar la función $f(x)$ que satisface una ecuación diferencial en un dominio determinado, junto a condiciones iniciales del problema, se tiene que empezar por diferenciar la variable independiente x, para después construir una grilla o malla, con puntos discretos sobre el dominio establecido. Luego el método define formulas de aproximación de las derivadas como combinación lineal de los valores funcionales en puntos de la grilla o malla. Finalmente, se deben reemplazar las derivadas en la ecuación diferencial por las formulas de aproximación que define el método. En este método se puede trabajar en un esquema Explícito, por ejemplo los valores de la grilla en el tiempo $t + \delta t$ son calculados con valores funcionales en el tiempo t, o también se puede optar por un esquema Implícito, los valores en el tiempo $t + \delta t$, son calculados en base a algunos valores funcionales en tiempo t y otro valores funcionales en tiempo $t + \delta t$, más adelante en este documento se detallan los procedimientos de cálculo de estos esquemas.

Existen otros métodos que son también ampliamente usados para este tipo de aplicaciones, como son Elementos Finitos [18] y Volúmenes Finitos [20], en este trabajo se decidió usar el método de Diferencias Finitas, ya que presenta una formulación sencilla, es de aplicación general y existe gran cantidad de aplicaciones previas que utilizan procesadores masivamente paralelos. De todas formas varios de los resultados obtenidos en este trabajo son directamente extrapolables a los otro métodos.

Las soluciones de los problemas de la física matemática por medio de Diferencias Finitas comienzan con el trabajo de Courant, Friedrichs y Lewy en 1928 [14]. Una aproximación de Diferencias Finitas se definió por primera vez para la ecuación de onda y se demostró que la condición de estabilidad de Courant-Friedrichs-Lewy era necesaria para la convergencia. En 1930 Gerschgorin obtiene los errores de borde para las aproximaciones de la Diferencias Finitas de los problemas elípticos cuyo trabajo se basó en la ecuación de Laplace [25]. Durante la decada de los 60 este enfoque y varias aproximaciones de ecuaciones elípticas con condiciones de bordes fueron analizadas. La teoría de las Diferencias Finitas para problemas parabólicos con condiciones iniciales, tuvieron entonces un intenso período de desarrollo durante los años 1950 y 1960, Cuando se exploró el concepto de estabilidad en el teorema de la equivalencia de Lax [78] y los lemas de la matriz de Kreiss [39]. Como se puede observar el dessarrollo principal de este método aplicado en este campo, tiene su florecimiento junto con la irrupción de la computación, y el comienzo del uso de está última para la resolución de problemas de la física. Además, con la aparición y masificación del uso de co-procesadores (aceleradores) con arquitecturas de hardware masivamente paralelas en la pasada década vuelve a tener un impulso importante la técnica.

A continuación se derivan las formulaciones principales para el método de las Diferencias Finitas.

2.1.1. Aproximación de la derivada primera en 1 dimensión

- Formula hacia adelante: se alcanza desarrollando la función mediante la serie de Taylor hasta el segundo orden,

$$f(x+\Delta x) = f(x) + f'(x)\cdot\Delta x + \frac{\Delta x^2}{2}\cdot f''(\xi). \tag{2.20}$$

Despejando:

$$f'(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x} - \frac{\Delta x}{2}\cdot f''(\xi) = f'(x). \tag{2.21}$$

Se obtiene la fórmula hacía adelante de la derivada primera:

$$f'(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}. \tag{2.22}$$

Con un error: E= $|\frac{\Delta x}{2}\cdot f''(\xi)| \leq \frac{\Delta x}{2}\cdot$M, donde M= max|f''(x)|, a $\leq$ x $\leq$ b.

- Formula hacia atrás, desarrollando la función mediante la serie de Taylor hasta el segundo orden:

$$f(x-\Delta x) = f(x) - f'(x)\cdot\Delta x + \frac{\Delta x^2}{2}\cdot f''(\eta), \tag{2.23}$$

y, despejando:

$$f'(x) \approx \frac{f(x)-f(x-\Delta x)}{\Delta x} + \frac{\Delta x}{2}\cdot f''(\eta) = f'(x). \tag{2.24}$$

Se obtiene la fórmula hacía atrás de la derivada primera:

$$f'(x) \approx \frac{f(x)-f(x-\Delta x)}{\Delta x}, \tag{2.25}$$

y un error: E= $|\frac{\Delta x}{2}\cdot f''(\eta)| \leq \frac{\Delta x}{2}\cdot$M, donde M= max|f''(x)|, a ≤ x ≤ b.

- Fórmula diferencias centradas, el primer paso del procedimiento es desarrollar la función mediante la serie de Taylor hasta el tercer orden, para $x+\Delta x$ y $x-\Delta x$:

$$f(x+\Delta x) = f(x) + f'(x)\cdot\Delta x + \frac{h^2}{2}\cdot f''(x) + \frac{\Delta x^3}{6}\cdot f'''(\xi), \tag{2.26}$$

$$f(x-\Delta x) = f(x) - f'(x)\cdot\Delta x + \frac{\Delta x^2}{2}\cdot f''(x) - \frac{\Delta x^3}{6}\cdot f'''(\eta), \tag{2.27}$$

Si se resta (2.26) y (2.27)

$$f(x+\Delta x) - f(x-\Delta x) = 2\Delta x\cdot f'(x) + \frac{\Delta x^3}{6}\cdot(f'''(\xi) + f'''(\eta)), \tag{2.28}$$

y, despejando:

$$\frac{f(x+\Delta x)-f(x-\Delta x)}{2\Delta x} - \frac{\Delta x^2}{6}\cdot f'''(\vartheta) = f'(x). \tag{2.29}$$

De esta manera se obtiene la formula para diferencias centradas:

$$f'(x) \approx \frac{f(x+\Delta x)-f(x-\Delta x)}{2\Delta x}. \tag{2.30}$$

Con un error: E= $|\frac{\Delta x^2}{6}\cdot f'''(\vartheta)| \leq \frac{\Delta x^2}{6}\cdot$M, donde M= max|f'''(x)|, a ≤ x ≤ b.

2.1.2. Aproximación de la derivada segunda en 1 dimensión

Desarrollando la función mediante la serie de Taylor hasta el cuarto orden, para $x + \Delta x$ y $x - \Delta x$:

$$f(x + \Delta x) = f(x) + f'(x) \cdot \Delta x + \frac{\Delta x^2}{2} \cdot f''(x) + \frac{\Delta x^3}{6} \cdot f'''(x) + \frac{\Delta x^4}{12} \cdot f''''(\xi), \tag{2.31}$$

$$f(x - \Delta x) = f(x) - f'(x) \cdot \Delta x + \frac{\Delta x^2}{2} \cdot f''(x) - \frac{\Delta x^3}{6} \cdot f'''(x) + \frac{\Delta x^4}{12} \cdot f''''(\eta). \tag{2.32}$$

Si se suman (2.31) y (2.32) :

$$f(x + \Delta x) + f(x - \Delta x) = 2f(x) + \Delta x^2 \cdot f''(x) + \frac{\Delta x^4}{12} \cdot (f''''(\xi) + f''''(\eta)), \tag{2.33}$$

se obtiene:

$$\frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{\Delta x^2} - \frac{\Delta x^2}{12} \cdot (f''''(\vartheta)) = f''(x). \tag{2.34}$$

De esta manera, la fórmula de la derivada segunda queda expresada como:

$$f''(x) \approx \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{\Delta x^2}, \tag{2.35}$$

y con un error: E= $|\frac{\Delta x^2}{12} \cdot \mathrm{f}''''(\vartheta)| \leq \frac{\Delta x^2}{12} \cdot \mathrm{M}$, donde M= $\max|\mathrm{f}''''(\mathrm{x})|$, en el intervalo definido por $a \leq x \leq b$.

2.1.3. Esquema Explícito

En el esquema de diferencias finitas Explícito, la variable en cualquier celda en el tiempo $t + \Delta t$ se calcula a partir del valor de la variable en la misma celda y en las celdas vecinas para el tiempo anterior t. De aquí que el cálculo de la concentración de una celda en un determinado tiempo es independiente de la variable de las demás celdas para el mismo tiempo. Aunque el método ofrece facilidades de cálculo, sufre limitaciones en la selección del valor de Δt. En particular, para un incremento de espacio dado el intervalo de tiempo debe ser compatible con los requisitos de estabilidad. Con frecuencia, esta limitante dicta el uso de

valores extremadamente pequeños de Δt, y se necesita un número muy grande de intervalos de tiempo para obtener la solución con un nivel de precisión razonable. La discretización de la Ecuación 2.17 en una dimensión y considerando un esquema explícito, tiene la siguiente expresión:

$$\frac{c^{t+\Delta t}(x) - c^t(x)}{\Delta t} = \mathbf{D} \cdot \left(\frac{c^t(x+\Delta x) - 2c^t(x) + c^t(x-\Delta x)}{\Delta x^2}\right) - \mathbf{v} \cdot \left(\frac{c^t(x+\Delta x) - c^t(x-\Delta x)}{2\Delta x}\right). \tag{2.36}$$

De esta manera se despeja la expresión $c^{t+\Delta t}(x)$:

$$c^{t+\Delta t}(x) = \left(\mathbf{D} \cdot \left(\frac{c^t(x+\Delta x) - 2c^t(x) + c^t(x-\Delta x)}{\Delta x^2}\right) - \mathbf{v} \cdot \left(\frac{c^t(x+\Delta x) - c^t(x-\Delta x)}{2\Delta x}\right)\right) \cdot \Delta t + c^t(x). \tag{2.37}$$

Si se cuenta con los valores de $c(x)$ en el tiempo t es fácil calcular el valor de $c(x)$ en el tiempo $t + \Delta t$, como se explicó anteriormente. Notar que hay restricciones sobre el Δt, específicamente la condición de estabilidad para este esquema es:

$$\frac{\mathbf{D} \cdot \Delta t}{\Delta x^2} \leq \frac{1}{2}. \tag{2.38}$$

2.1.4. Esquema Implícito

Para evitar los problemas de inestabilidad de los esquemas Explícitos, se puede utilizar un esquema implícito.

2.1.4.1. Esquema Puramente Implícito

En este esquema, la ecuación en diferencias finitas se aproximan mediante una formula de diferencias retrasadas para la derivada temporal. Esta práctica genera un esquema puramente Implícito en diferencias finitas, donde la concentración de la celda en el instante actual $(t + \Delta t)$, depende de su propia concentración en el instante anterior (t), pero también de la concentración de las celdas vecinas en el instante actual $(t + \Delta t)$. Este esquema es incondicionalmente estable.

$$\frac{c^{t+\Delta t}(x) - c^t(x)}{\Delta t} = \mathbf{D} \cdot \left(\frac{c^{t+\Delta t}(x+\Delta x) - 2c^{t+\Delta t}(x) + c^{t+\Delta t}(x-\Delta x)}{\Delta x^2}\right) - \mathbf{v} \cdot \left(\frac{c^{t+\Delta t}(x+\Delta x) - c^{t+\Delta t}(x-\Delta x)}{2\Delta x}\right). \tag{2.39}$$

Si se reordena la Ecuación 2.39 pasando a la izquierda todos los términos en $t + \Delta t$ y a la derecha los termino en t:

$$(c^{t+\Delta t}(x)) \cdot \underbrace{(\frac{1}{\Delta t} + \frac{2\mathbf{D}}{\Delta x^2})}_{C_1} + (c^{t+\Delta t}(x + \Delta x)) \underbrace{(-\frac{\mathbf{D}}{\Delta x^2} + \frac{\mathbf{v}}{2\Delta x})}_{C_2} + (c^{t+\Delta t}(x - \Delta x)) \cdot \underbrace{(-\frac{\mathbf{D}}{\Delta x^2} - \frac{\mathbf{v}}{2\Delta x})}_{C_3} = \frac{c^t(x)}{\Delta t}. \tag{2.40}$$

Encontrar las soluciones del sistema, $c^{t+\Delta t}(x)$, implica resolver un sistema lineal de ecuaciones $A \cdot x = b$ donde el vector b está formado por los valores de $c^t(0)$, x es el vector de soluciones y A es una matriz cuadrada, en bandas, y cuyo coeficientes estarán formados por C_1, C_2 y C_3 según la Ecuación 2.40. A continuación se presenta una matriz para una discretización de 5 celdas:

$$\begin{bmatrix} C_1' & C_2' & 0 & 0 & 0 \\ C_3 & C_1 & C_2 & 0 & 0 \\ 0 & C_3 & C_1 & C_2 & 0 \\ 0 & 0 & C_3 & C_1 & C_2 \\ 0 & 0 & 0 & C_3' & C_1'' \end{bmatrix} \begin{bmatrix} c_1(x) \\ c_2(x) \\ c_3(x) \\ c_4(x) \\ c_5(x) \end{bmatrix} = \begin{bmatrix} c_1(0) \\ c_2(0) \\ c_3(0) \\ c_4(0) \\ c_5(0) \end{bmatrix}.$$

.

En la primer y última fila los coeficientes son afectados por el tipo de condición de contorno considerada $(C_1', C_2', C_3', C_1'')$.

Este esquema es más costoso computacionalmente si se lo compara con el esquema Explícito, pero en general tiene un error de aproximación menor.

2.1.4.2. Esquema Crank-Nicolson

El esquema de Crank-Nicolson mejora el resultado en cuanto a la precisión numérica con respecto a los métodos anteriores y es numéricamente estable ya que es un método Implícito[2]. Se basa en sumar los dos esquemas antes descritos y promediar dicha suma, según la siguiente fórmula:

$$\frac{c^{t+\Delta t}(x) - c^t(x)}{\Delta t} = \theta \cdot (\text{esquema Implícito}) + (1-\theta) \cdot (\text{esquema Explícito}). \tag{2.41}$$

Notar que con $\theta = 0$ se obtiene el esquema Explícito, con $\theta = 1$ el Implícito y con $\theta = 1/2$ el esquema de Crank-Nicolson [26]. Este último esquema provee mejoras en los resultados desde el punto de vista de la precisión numérica, pero tiene un costo computacional más alto, porque implica mayor número de operaciones, ya que suma al cálculo del esquema puramente Implícito el producto de una matriz por un vector. En el presente trabajo no se explorarán las diferencias entre el esquema puramente Implícito y el de Cranck-Nicolson. Se trabajará con el esquema de Crank-Nicolson únicamente.

2.1.5. Esquema Predictor-Corrector

Este esquema es general y plantea realizar un primer cálculo que estime el resultado y luego iterar mejorando la solución parcial. Para esta tesis se creó un desarrollo particular, es decir un método iterativo basado en el esquema explícito que aproxima a Crank-Nicolson, con mejor precisión numérica que el esquema Explícito.

El método utilizado calcula, con una fórmula explícita, el valor de la solución preliminar (paso predictivo) y, después de eso, a partir de este valor se calcula una solución intermedia (posterior en el tiempo a la solución inicial), para culminar esta etapa, se promedia estas dos soluciones para obtener una solución parcial (paso correctivo). Por último, si la diferencia entre dos soluciones parciales consecutivas es menor que un ϵ la iteración termina, en otro caso se sigue iterando sobre la solución intermedia para refinarla.

Este esquema se puede expresar en forma de ecuación como:

$$\begin{aligned}\frac{c^{t+\Delta t}(x) - c^t(x)}{\Delta t} = \theta \cdot (\mathbf{D} \cdot (&\frac{c^{t+\Delta t}(x+\Delta x) - 2c^{t+\Delta t}(x) + c^{t+\Delta t}(x-\Delta x)}{\Delta x^2}) \\ &- \mathbf{v} \cdot (\frac{c^{t+\Delta t}(x+\Delta x) - c^{t+\Delta t}(x)}{\Delta x})) \\ + ((1-\theta) \cdot (\mathbf{D} \cdot (&\frac{c^t(x+\Delta x) - 2c^t(x) + c^t(x-\Delta x)}{\Delta x^2}) \\ &- \mathbf{v} \cdot (\frac{c^t(x+\Delta x) - c^t(x-\Delta x)}{2\Delta x}))).\end{aligned} \tag{2.42}$$

[2] A menos que se utilice $\theta = 0$ en la Ecuación 2.41.

Reagrupando en la izquierda de la ecuación los términos en x, se alcanza la Ecuación (2.43).

$$\begin{aligned}\frac{c^{t+\Delta t}(x)-c^{t}(x)}{\Delta t}+\theta\cdot\frac{2\mathbf{D}\cdot c^{t+\Delta t}(x)}{\Delta x^2}=\theta\cdot(\mathbf{D}\cdot(\frac{c^{t+\Delta t}(x+\Delta x)+c^{t+\Delta t}(x-\Delta x)}{\Delta x^2})\\ -\mathbf{v}\cdot(\frac{c^{t+\Delta t}(x+\Delta x)-c^{t+\Delta t}(x)}{\Delta x}))\\ +((1-\theta)\cdot(\mathbf{D}\cdot(\frac{c^{t}(x+\Delta x)-2c^{t}(x)+c^{t}(x-\Delta x)}{\Delta x^2})\\ -\mathbf{v}\cdot(\frac{c^{t}(x+\Delta x)-c^{t}(x-\Delta x)}{2\Delta x}))).\end{aligned} \tag{2.43}$$

Despejando el término $c^{(t+\Delta t,k)}(x)$, para el tiempo $t+\Delta t$ e iteración k se obtiene la Ecuación (2.44).

$$\begin{aligned}c^{(t+\Delta t,k+1)}(x)=\frac{1}{(\frac{1}{(\Delta t)})+(\frac{2\theta\mathbf{D}}{\Delta x^2})}(\theta\cdot(\mathbf{D}\cdot(\frac{c^{(t+\Delta t,k)}(x+\Delta x)+c^{(t+\Delta t,k)}(x-\Delta x)}{\Delta x^2})\\ -\mathbf{v}\cdot(\frac{c^{(t+\Delta t,k)}(x+\Delta x)-c^{(t+\Delta t,k)}(x)}{\Delta x}))\\ +((1-\theta)\cdot(\mathbf{D}\cdot(\frac{c^{t}(x+\Delta x)-2c^{t}(x)+c^{t}(x-\Delta x)}{\Delta x^2})\\ -\mathbf{v}\cdot(\frac{c^{t}(x+\Delta x)-c^{t}(x-\Delta x)}{2\Delta x})))+c^{t}(x).\end{aligned} \tag{2.44}$$

Otra forma de derivar el sistema de ecuaciones es en forma matricial, donde $\mathbf{D}$, $\mathbf{M}$ y $\mathbf{N}$ son matrices y $\mathbf{D}$ además es diagonal, el sistema queda expresado como lo muestra la Ecuación (2.45). Además, con esta formulación se puede ver con claridad el procedimiento iterativo que se despeja de la expresión:

$$\mathbf{D}\cdot\mathbf{c}^{(t+\Delta t,k+1)}=\mathbf{M}\cdot\mathbf{c}^{(t+\Delta t,k)}+\mathbf{N}\cdot\mathbf{c}^{(t)}. \tag{2.45}$$

Ahora se puede plantear el método del cálculo Predictivo-Correctivo utilizando el planteo matricial la Ecuación (2.45), como se resume en el Algoritmo 1.

Corresponde mencionar que en el algoritmo implementado, en general, no se utilizan operaciones entre matrices, ni matriz-vector, ya que se usan las formulas explícitas resultantes de la discretización de las diferencias finitas.

Algoritmo 1 Método Predictivo-Correctivo

Entrada : $\mathbf{c}^{(0)}$ Solución Inicial
Salida : $\mathbf{c}$ *vector* Solución Final
$\mathbf{c}^{(t+\Delta t,k)} = \mathbf{c}^{(t)}$
while (k < maxIteraciones and error > maxError) **do**
  $\mathbf{c}^{(k+1)} = \mathbf{D}^{-1} \cdot \mathbf{M} \cdot \mathbf{c}^{(k)} + \mathbf{D}^{-1} \cdot \mathbf{N} \cdot \mathbf{c}$.
  $error = abs(\mathbf{c}^{(k+1)} - \mathbf{c}^{(k)})$
  $\mathbf{c}^{(k)} = \mathbf{c}^{(k+1)}$
  $k = k + 1$
end while

2.2. Solvers de sistemas de ecuaciones lineales

Como se dijo anteriormente, una discretización usando Diferencias Finitas muy a menudo implica la resolución de sistemas lineales de ecuaciones[3], estos sistemas puede escribirse en forma matricial como sigue:

$$\mathbf{Ax} = \mathbf{b}, \tag{2.46}$$

donde los coeficientes del vector $\mathbf{x}$ representan los valores a ser calculados (temperatura, velocidad, concentración, etc.), mientras los coeficientes de la matriz $\mathbf{A}$ y el vector $\mathbf{b}$ son generados por la discretización y las condiciones de borde.

2.2.1. Solvers Iterativos vs Directos

Un Solver iterativo trata de resolver una ecuación o un sistema de ecuaciones mediante aproximaciones sucesivas a la solución, comenzando con una estimación inicial. Si el método converge se obtiene una solución aproximada del problema, cuyo error satisface algún criterio prefijado. Esta aproximación contrasta con los métodos directos, que llegan a la solución en un número específico de pasos, en función del tamaño del sistema a resolver y consiguen, a menos de errores numéricos, la solución exacta del sistema lineal. Los métodos iterativos son útiles para resolver problemas que involucran un número grande de variables (a veces del orden de millones), donde los métodos directos tendrían un costo prohibitivo incluso con la potencia del mejor computador disponible. Cuando se trabaja con sistemas lineales dispersos como es el caso de esta tesis, un solver directo puede generar "llenado" (problema conocido como $fill - in$ en la literatura anglosajona), que puede ocurrir cuando se realiza la factorización de una matriz dispersa. Por ejemplo si se tiene una matriz A dispersa y se realiza una factorización LU, es muy posible que la cantidad de elementos distintos de 0 de la matriz $\mathbf{A}$ sea mucho menor que la suma de elementos distintos de 0 de $\mathbf{L}$ y $\mathbf{U}$, debido a términos nuevos que aparecen al operar con los coeficientes (ocasionados por el $fill-in$) en las matrices $\mathbf{L}$ y $\mathbf{U}$.

[3]Cuando se trabaja con esquemas Implícitos

En este trabajo se profundiza en particular, en dos métodos iterativos. En primer término el método Strong Implicit Procedure (SIP), que es un solver desarrollado particularmente para trabajar con matrices hepta-diagonales o penta-diagonales, que son las que se generan en este tipo de fenómenos físicos (cuando se utilizan grillas regulares). El otro solver abordado fue el Gradiente Biconjugado estabilizado (BiCGStab por sus siglas en inglés), que es un solver iterativo para sistemas lineales generales ampliamente utilizado por la comunidad especialmente en contextos de matrices dispersas.

2.2.2. Strong Implicit Procedure (SIP)

En esta sección se describe el método Strong Implicit Procedure (SIP), que fue propuesto por Herbert L. Stone en 1968 [17] [68]. Este método para resolver sistemas lineales es apropiado para ser aplicado sobre sistemas subyacentes al discretizar sistemas de ecuaciones en derivadas parciales (EDPs), y por esto, ampliamente usado para resolver problemas de mecánica de los fluidos.

2.2.2.1. Introducción

El SIP fue diseñado especialmente para trabajar con matrices hepta y pentadiagonales, es decir matrices de bandas. Si se hace una descomposición **LU** aproximada de la matriz A ($\mathbf{LU} \approx \mathbf{A}$), se obtienen dos matrices $\widetilde{\mathbf{L}}$ de banda inferior y $\widetilde{\mathbf{U}}$ de banda superior. El producto de estas dos matrices genera una matriz **M**, que contiene más diagonales que la matriz **A**. Esto es debido a que la matriz **A** es de banda, entonces se puede afirmar que:

$$\mathbf{A} \approx \mathbf{LU} = \mathbf{M} = \mathbf{A} + \mathbf{N}, \tag{2.47}$$

donde **N** es una matriz que contiene las diagonales extras de la matriz **M** con respecto a la matriz **A**. La matriz **N** se puede tomar como una medida del error producida por la descomposición. Entonces, para que sea pequeña la desviación con respecto a la matriz A, se requiere que:

$$(\mathbf{A} + \mathbf{N}) \cdot \mathbf{x} \approx \mathbf{b} \longrightarrow \mathbf{N} \cdot \mathbf{x} \approx 0. \tag{2.48}$$

Los coeficientes de esas diagonales extras pueden ser calculados a través de operaciones con los coeficientes vecinos. Haciendo una aproximación lineal y usando un parámetro α se obtiene:

$$x_{nw} \approx \alpha(x_w + x_n - x_p). \tag{2.49}$$

Donde la nomenclatura se refiere a la posición dentro de la matriz, nw significa $NorthWest$,

w *West*, n *North* y p *Point* . Entonces, la factorización LU de la matriz incompleta, intenta aproximar a la descomposición LU pero sin generar diagonales extras.

$$
\begin{aligned}
L_B^l &= A_B^l[1+\alpha(U_N^{l-N_i\cdot N_j}+U_E^{l-N_i\cdot N_j})]^{-1}\\
L_W^l &= A_W^l[1+\alpha(U_N^{l-1}\cdot U_T^{l-1})]^{-1}\\
L_S^l &= A_S^l[1+\alpha(U_E^{l-N_i}\cdot U_T^{l-N_i})]^{-1}\\
L_P^l &= A_P^l....L_B^l....U_E^{l-N_i\cdot N_j}\\
U_N^l &= ...L_P^l...\\
U_T^l &= ...L_P^l...\\
U_E^l &= ...L_P^l...
\end{aligned}
$$

La nomenclatura *B, W, S, P*, etc. se refiere a *Bottom, West, South, Point*, respectivamente. El índice superior l hace referencia al índice del vector, o sea que L_B, L_W, A_P, etc. son vectores que representan las diagonales a la matriz. El mapeo de 3D a la posición l se hace utilizando la siguiente regla $l = k\cdot N_i\cdot N_j + i\cdot N_j + j$ donde N_i, N_j, N_k son las dimensiones de cada una de las coordenadas de la grilla.

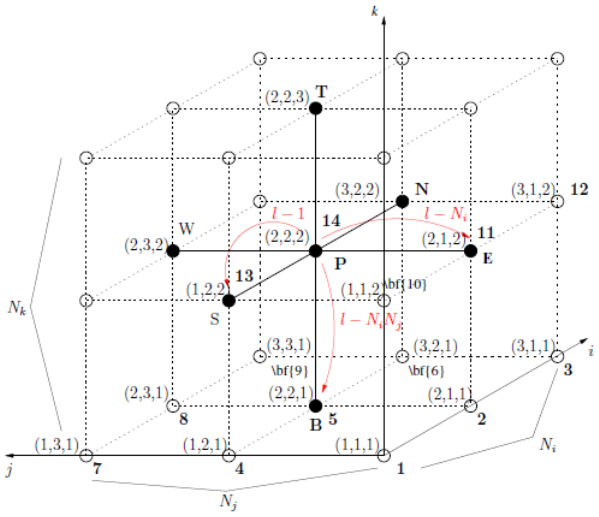

Figura 2.4: En este ejemplo $N_i = N_j = N_k = 3$, el punto (2,2,2) se mapea con la posición $l = 2\cdot 3\cdot 3 + 2\cdot 3 + 2 = 26$ del vector. Extraído de [17].

Si se itera en x para aproximar la solución, en el paso n se tendrá:

$$\mathbf{A}\cdot\mathbf{x}^n = \mathbf{b} - \mathbf{r}^n. \tag{2.50}$$

Donde $\mathbf{r}^n$ es el resto. Si $\mathbf{x}$ es la solución del sistema, se puede definir el error de convergencia como:

$$\epsilon^n = \mathbf{x} - \mathbf{x}^n. \tag{2.51}$$

Se puede deducir entonces que:

$$\mathbf{A} \cdot \epsilon^n = \mathbf{A} \cdot \mathbf{x} - \mathbf{A} \cdot \mathbf{x}^n = \mathbf{b} - \mathbf{b} + \mathbf{r}^n = \mathbf{r}^n \longrightarrow \mathbf{A} \cdot \epsilon^n = \mathbf{r}^n. \tag{2.52}$$

Como el vector $\mathbf{x}$, que es la solución exacta, no es conocida se define la siguiente formula iterativa:

$$\mathbf{A} \cdot (\mathbf{x}^{n+1} - \mathbf{x}^n) = \mathbf{r}^n. \tag{2.53}$$

$$\mathbf{A} \cdot \Delta x = \mathbf{L} \cdot \mathbf{U} \cdot \Delta x = \mathbf{r}^n. \tag{2.54}$$

$$\mathbf{U} \cdot \Delta x = \mathbf{L}^{-1} \cdot \mathbf{r}^n = \mathbf{R}^n. \tag{2.55}$$

Teniendo la descomposición incompleta LU de la matriz $\mathbf{A}$, se puede calcular $\mathbf{r}^n$, luego el vector $\mathbf{R}^n$ y por último Δx, de esta forma se puede definir el método según lo resumido en el Algoritmo 2.

Algoritmo 2 Algoritmo del SIP

1: **procedure** SIP
2: *Entrada* : $\mathbf{A}$ *matriz*, $\mathbf{b}$ *vector de terminos independientes*, $\mathbf{x_0}$ Solución Inicial
3: *Salida* : $\mathbf{x}$ *vector* Solución Final
4: Hacer la descomposición incompleta LU $\mathbf{A} \approx \mathbf{LU}$.
5: **while** (el resto no cumpla con el criterio de convergencia) **do**
6: Calcular el Resto $\mathbf{r}^n = \mathbf{b} - \mathbf{A} \cdot \mathbf{x}^n$.
7: Calcular el vector $\mathbf{R}^n$ (Sustitución hacia adelante) $\mathbf{R}^n = \mathbf{L}^{-1} \cdot \mathbf{r}^n$.
8: Calcular Δx (Sustitución hacia atrás) $\mathbf{U} \cdot \Delta x = \mathbf{R}^n$.
9: **end while**
10: **end procedure**

2.2.2.2. Paralelización

El método Strong Implicit Procedure (SIP), está basado en la descomposición incompleta LU, seguido de sucesivas sustituciones hacia adelante y hacía atrás intentando minimizar el resto. Se puede ver que el método presenta dependencia de datos, lo que limita la paralelización de éste. En particular, en el algoritmo de la descomposición y la sustitución hacía adelante, para calcular el punto (i, j, k), es necesario contar con los puntos

$(i-1,j,k),(i,j-1,k),(i,j,k-1)$, y para la sustitución hacia atrás la dependencia de datos es a la inversa.

Esta dependencia deja tres posibles opciones de paralización:

- Por hiperplanos $i+j+k = constante$.
- Por hiperlíneas $j+k = constante$.
- Paralelización por bloques.

Paralelización por Hiperplanos.

En esta opción el paralelismo se da dentro de los hiperplanos $i+j+k = constante$, se trabaja con un hiperplano a la vez, y cuando se termina se sigue con el siguiente hiperplano en el sentido de j, ver Figura 2.5.

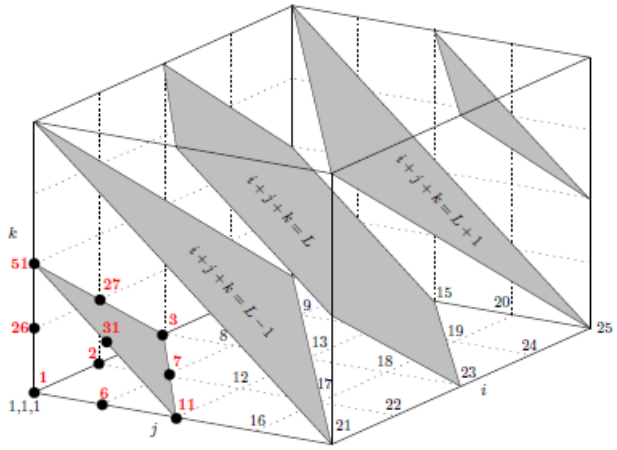

Figura 2.5: División del dominio de trabajo del SIP en Hiperplanos $i+j+k = constante$. Extraído de [17].

Paralelización por Hiperlíneas.

Aquí el paralelismo se da dentro de los hiperlíneas $j+k = constante$, todas las hiperlíneas $j+k = constante$ en la dirección de i, pueden ser procesadas en paralelo, ver Figura 2.6 para una explicación gráfica.

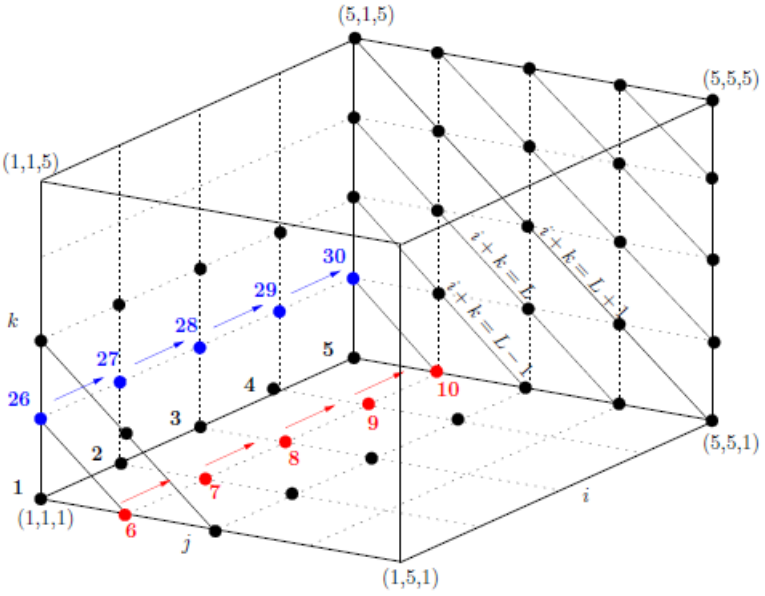

Figura 2.6: División del dominio de trabajo del SIP en Hiperlíneas $j+k = constante$. Extraído de [17].

Paralelización por bloques.

En esta tercera opción de paralelización, se divide el dominio en bloques tal como lo muestra la Figura 2.7, los bloques con la misma letra pueden ser procesados en paralelos, pero cada bloque puede ser procesado solamente luego de que el bloque de la izquierda y el bloque de abajo con respecto a dicho bloque hayan finalizado. Dentro de cada bloque se debe recorrer los puntos primero en la dirección de j y luego en la dirección de i.

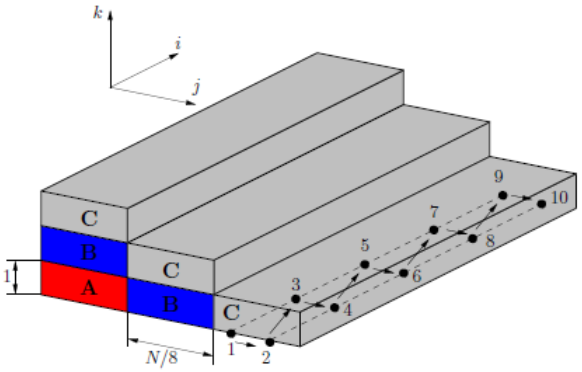

Figura 2.7: División del dominio de trabajo en Bloques. Extraido de [17].

2.2.3. Gradiente Biconjugado Estabilizado (BiCGStab)

El método del gradiente biconjugado estabilizado, generalmente abreviado como BiCGStab, es un método iterativo propuesto por H. A. van der Vorst [73] para la resolución numérica

de los sistemas de ecuaciones lineales no simétricos. Es un método que explota el uso de los subespacio de Krylov y en especial una variante del método del gradiente biconjugado (BiCG). Ofrece convergencia más rápida y suave cuando se lo compara con el original BiCG así como otras variantes basados en subespacios de Krylov como el método del gradiente conjugado cuadrado (CGS).

La idea básica del método del gradiente biconjugado al igual que método CGS consiste en construir una base de vectores ortogonales y utilizarla para realizar la búsqueda de la solución en forma más eficiente. Tal forma de proceder generalmente no sería aconsejable porque la construcción de una base ortogonal utilizando el procedimiento de Gramm-Schmidt requiere, al seleccionar cada nuevo elemento de la base, asegurar su ortogonalidad con respecto a cada uno de los vectores construidos previamente. La gran ventaja del método radica en que cuando se utiliza este procedimiento, basta con asegurar la ortogonalidad de un nuevo miembro con respecto al último que se ha construido, para que automáticamente esta condición se cumpla con respecto a todos los anteriores.

El método CGS es valido únicamente cuando se trabaja con matrices simétricas, en cambio el método BiCG también permite trabajar con matrices no simétricas. Para que dicho método pueda trabajar con matrices no simétricas, se agrega una variable ficticia $\widetilde{x}$ y el sistema queda planteado de la siguiente forma:

$$\begin{bmatrix} & \mathbf{A} \\ \mathbf{A}^T & \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{x}} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

.

Aplicando el método CGS a este sistema, genera dos sistemas (ver Ecuación 2.56), donde $\mathbf{p_k}$ y $\widetilde{\mathbf{p}}$ son los vectores de direcciones y $\mathbf{r}$, $\widetilde{\mathbf{r}}$ los restos. De estos dos sistemas solo interesa calcular $\mathbf{x}$, por lo que los vectores relacionados con la resolución de $\mathbf{A}^t\widetilde{\mathbf{x}} = \widetilde{\mathbf{b}}$, $\widetilde{\mathbf{r}}$ y $\widetilde{\mathbf{p}}$, sólo son necesarios para obtener los escalares en cada iteración. El método BiCG define los residuos y los vectores conjugados asociados a los sistemas a través de un par de polinomios asociados a las matrices que definen ambos el sistema de ecuaciones lineales.

$$\left.\begin{aligned} \mathbf{A}\mathbf{x} = b \Rightarrow \mathbf{r_k} = \phi_k(\mathbf{A})\mathbf{r_0}, \quad p_k = \pi_k(\mathbf{A})\mathbf{r_0} \\ \mathbf{A}^T\widetilde{\mathbf{x}} = 0 \Rightarrow \widetilde{\mathbf{r_k}} = \phi_k(\mathbf{A}^t)\widetilde{\mathbf{r_0}}, \quad \widetilde{p_k} = \pi_k(\mathbf{A}^t)\widetilde{\mathbf{r_0}} \end{aligned}\right\} \quad \text{con } \phi_k(0) = 1 \tag{2.56}$$

Donde ϕ_j y π_k son polinomios de grado k.

De este modo los escalares α_k y β_k quedan definidos como:

$$\alpha_k = \frac{\widetilde{\mathbf{r_k}}^T\mathbf{r_k}}{p_k\mathbf{A}p_k} = \frac{(\phi_k(\mathbf{A}^T)\widetilde{\mathbf{r_0}})^T\phi_k(\mathbf{A})\mathbf{r_0}}{(\pi_k(\mathbf{A}^T)\widetilde{\mathbf{r_0}})^T\mathbf{A}\pi_k(\mathbf{A})\mathbf{r_0}} = \frac{(\widetilde{\mathbf{r_0}})^T\phi_k^2(\mathbf{A})\mathbf{r_0}}{(\widetilde{\mathbf{r_0}})^T\mathbf{A}\pi_k^2(\mathbf{A})\mathbf{r_0}}, \tag{2.57}$$

$$\beta_k = \frac{\widetilde{\mathbf{r_{k+1}}}^T\mathbf{r_{k+1}}}{\mathbf{p_k}\mathbf{A}\mathbf{p_k}} = \frac{(\phi_{k+1}(\mathbf{A}^T)\widetilde{\mathbf{r_0}})^T\phi_{k+1}(\mathbf{A})\mathbf{r_0}}{(\pi_k(\mathbf{A}^T)\widetilde{\mathbf{r_0}})^T\mathbf{A}\pi_k(\mathbf{A})\mathbf{r_0}} = \frac{(\widetilde{\mathbf{r_0}})^T\phi_{k+1}^2(\mathbf{A})\mathbf{r_0}.}{(\widetilde{\mathbf{r_0}})^T\mathbf{A}\pi_k^2(\mathbf{A})\mathbf{r_0}}. \tag{2.58}$$

Seguidamente, se transforman las recurrencias asociadas a los residuos y los vectores conjugados en términos de polinomios, y se calculan las expresiones que definen el cuadrado

de los polinomios, que son la base para definir los residuos y los vectores conjugados asociados al método:

$$\mathbf{r_{k+1}} = \mathbf{r_k} - \alpha_k \mathbf{A}\mathbf{p_k} \Rightarrow \phi_{k+1} = \phi_k - \alpha_k \mathbf{A}\pi_k \Rightarrow \phi_{k+1}^2 = (\phi_k - \alpha_k(\mathbf{A})\pi_k)^2 \Rightarrow \widetilde{\mathbf{r_k}} = \phi_k^2(\mathbf{A})\mathbf{r_0}. \tag{2.59}$$

$$\mathbf{p_{k+1}} = \mathbf{r_{k+1}} + \beta_k \mathbf{p_k} \Rightarrow \pi_{k+1} = \phi_{k+1} + \beta_k \pi_k \Rightarrow \pi_{k+1}^2 = (\phi_{k+1} + \beta_k \pi_k)^2 \Rightarrow \widetilde{\mathbf{p_k}} = \pi_k^2(\mathbf{A})\mathbf{r_0}. \tag{2.60}$$

Por lo tanto:

$$\alpha_k = \frac{\widetilde{\mathbf{r_k}}^T \widetilde{\mathbf{r_k}}}{\widetilde{\mathbf{r_k}}^T \mathbf{A}\widetilde{\mathbf{p_k}}},\ \beta_k = \frac{\widetilde{\mathbf{r_k}}^T \widetilde{\mathbf{r_{k+1}}}}{\widetilde{\mathbf{r_k}}^T \widetilde{\mathbf{r_k}}}. \tag{2.61}$$

Este método funciona bien en general, pero el hecho de duplicar los polinomios magnifica los errores de redondeo lo que implica grandes alteraciones en los residuos, y por tanto el cálculo de su norma puede ser bastante inexacto. El método BiCGStab trata de mejorar la convergencia irregular del método CGS, utilizando ideas de los métodos BiCG y GMRES (método generalizado de los mínimos residuales). La idea detrás del método es usar un polinomio alternativo para la generación de residuos y vectores conjugados asociados al sistema $\mathbf{A}^T\widetilde{\mathbf{x}} = \widetilde{\mathbf{b}}$, $\widetilde{\mathbf{r}}$ y $\widetilde{\mathbf{p}}$, utilizando el hecho de que este sistema no debe ser resuelto. La definición de este polinomio, y su relación con los residuos y vectores conjugados del sistema $\mathbf{Ax} = \mathbf{b}$, es la siguiente:

$$\psi_{k+1}(\mathbf{A}) = (1-\omega_k\mathbf{A})\psi_k(\mathbf{A}) \Rightarrow = \begin{cases} \widetilde{\mathbf{r_k}} = \psi_k(\mathbf{A})\phi_k(\mathbf{A})\mathbf{r_0} \Rightarrow \widetilde{\mathbf{r_{k+1}}} = (1-\omega_k\mathbf{A})(\widetilde{\mathbf{r_k}} - \alpha_k\mathbf{A}\widetilde{\mathbf{p_k}}); \\ \widetilde{\mathbf{p_k}} = \psi_k(\mathbf{A})\pi_k(\mathbf{A})\mathbf{r_0} \Rightarrow \widetilde{\mathbf{p_{k+1}}} = \widetilde{\mathbf{r_{k+1}}} + \beta_k(I - \omega_k\mathbf{A}\widetilde{\mathbf{p_k}}). \end{cases}$$

Por su parte, el cálculo de los escalares implica la transformación de las expresiones definidas en el método BICG para que contengan los nuevos vectores, dando lugar a las siguientes expresiones:

$$\widetilde{\mathbf{p_k}}(\mathbf{A}) = \widetilde{\mathbf{r_0}}^T \widetilde{\mathbf{r_k}} \Rightarrow \alpha_k = \frac{\widetilde{\mathbf{r_k}}^T \widetilde{\mathbf{r_k}}}{\widetilde{\mathbf{r_k}}^T \mathbf{A}\widetilde{\mathbf{p_k}}} = \frac{\widetilde{\mathbf{p_k}}}{\widetilde{\mathbf{r_0}}^T \mathbf{A}\widetilde{\mathbf{p_k}}},\ \beta_k = \frac{\widetilde{\mathbf{p_{k+1}}}\beta_k}{\widetilde{\mathbf{p_k}}\omega_k}. \tag{2.62}$$

La elección de ω_k definen las propiedades numéricas del método, pudiéndose determinar un conjunto de métodos que varían únicamente en la manera en el que se calcula este escalar. En el caso del método BiCGStab clásico, dicho escalar se obtiene aplicando una etapa del método GMRES y se calcula como:

$$\mathbf{s_k} = \widetilde{\mathbf{r_k}} - \alpha_k \mathbf{A}\widetilde{\mathbf{p_k}} \Rightarrow \widetilde{\mathbf{r_{k+1}}} = (1 - \omega_k \mathbf{A}\mathbf{s_k}),\ \omega_k = \frac{\mathbf{s_k}^T \mathbf{A}\mathbf{s_k}}{(\mathbf{A}\mathbf{s_k})^T \mathbf{A}\mathbf{s_k}}. \tag{2.63}$$

Con respecto a la actualización de la solución, ésta se puede calcular directamente de la definición del residuo de la siguiente forma:

$$\widetilde{\mathbf{r_{k+1}}} = \widetilde{\mathbf{r_k}} - \alpha_k \mathbf{A}\widetilde{\mathbf{p_k}} - \omega_k \mathbf{A}\mathbf{s_k} \Rightarrow \mathbf{x_{k+1}} = \mathbf{x_k} + \alpha_k \widetilde{\mathbf{p_k}} + \omega_k \mathbf{s_k}. \tag{2.64}$$

Algoritmo 3 Gradiente Biconjugado Estabilizado (BiCGStab)

procedure BiCGStab
  Entrada : $\mathbf{A}$ *matriz*, $\mathbf{x_0}$ Solución Inicial
  Salida : $\mathbf{x}$ *vector* Solución Final
  $j = 0$
  while $\tau_j > \tau_{max}$ **do**
    $\mathbf{v_j} = \mathbf{A}\mathbf{p_j}$
    $\alpha_j = \sigma_j / \mathbf{r_j}^T \mathbf{v_j}$
    $\mathbf{s_j} = \mathbf{r_j} - \alpha_j \mathbf{v_j}$
    $\tau_j = \|\mathbf{s_j}\|_2$
    if $\tau_j < \tau_{max}$ **then**
      $\mathbf{x_{j+1}} = \mathbf{x_j} + \alpha_j \mathbf{p_j}$
      break
    end if
    $\mathbf{v_j} = \mathbf{A}\mathbf{s_j}$
    $\rho_j = (\mathbf{v_j}^T \mathbf{s_j}) / (\mathbf{v_j}^T \mathbf{v_j})$
    $\mathbf{x_{j+1}} = \mathbf{x_j} + \alpha_j \mathbf{p_j} + \rho_j \mathbf{s_j}$
    $\mathbf{r_{j+1}} = \mathbf{s_j} - \rho_j \mathbf{v_j}$
    $\zeta_j = \mathbf{r_0}^T \mathbf{r_{j+1}}$
    $\beta_j = (\zeta_j / \sigma_j) * (\alpha_j / \rho_j)$
    $\sigma_{j+1} = \zeta_j$
    $\tau_{j+1} = \|\mathbf{r_{j+1}}\|_2$
    $j = j + 1$
  end while
end procedure

Capítulo 3

Paralelización de los métodos de resolución de la ecuación de transporte

En este capítulo se describen las diferentes implementaciones realizadas. En particular se hicieron implementaciones de los tres métodos descritos en el Capítulo 2, en forma secuencial y paralela en lenguaje C, utilizando tanto la API OpenMP como el framework CUDA para explotar el paralelismo sobre arquitecturas multicore y masivamente paralelas (por ejemplo, GPUs) respectivamente. Las implementaciones secuenciales son directas y tienen el único objetivo de ser las implementaciones de referencia, por ello no se entra en detalle sobre dichos desarrollos.

3.1. Paralelización del esquema Implícito

En este esquema para cada paso de tiempo, se procesa la multiplicación de una matriz por un vector y luego se resuelve un sistema de ecuaciones lineales. La paralelización del producto de una matriz por un vector no presenta mayores desafíos e incluso puede obtenerse por la utilización de bibliotecas de álgebra, en especial implementaciones de la operación GEMV (general matrix-vector) de BLAS (Basic Linear Algebra Subprograms) o en el caso disperso SPMV (sparse matrix-vector). Por lo cual en este método el mayor reto es la paralelización de la resolución del sistema lineal en cada paso de tiempo. Para ello, como fue comentado, se evaluó el uso del algoritmo SIP (Storng Implict Procedure) y el algoritmo gradiente biconjugado estabilizado (BiCGStab) de forma de poder comparar el desempeño de estos dos métodos.

3.1.1. Paralelización del SIP

En este algoritmo hay cuatro procedimientos que tienen que ser computados en un cierto orden, es decir, los cuatro pasos definen un camino crítico en el grafo de ejecución del método[1]. Entonces si se quiere paralelizar el SIP hay que paralelizar cada uno de estos procedimientos, pero manteniendo la lógica secuencial de las tareas.

3.1.1.1. Paralelización de la descomposición Incompleta LU

Dentro de las tres estrategias de paralelización del SIP, se eligió la paralelización por hiperplanos ya que es la opción con mayor grado de paralelismo. Sin embargo, el mapeo de índices a hiperplanos no es directo. Por esta razón fue necesario implementar una función auxiliar que calcule los puntos de cada hiperplano. Además esta función devuelve dos vectores más, uno con los *offset* de los hiperplanos y el otro con la cantidad de puntos de cada hyperplano, todo esto basado en el trabajo de Igounet et al. [33].

$$\texttt{IJKV} = [\underbrace{1,}_{L=1}\ \underbrace{2,6,26,}_{L=2}\ \underbrace{3,7,11,27,31,51,}_{L=3}\ldots]$$

Figura 3.1: Vector con los puntos de cada hiperplano para paralelizar el SIP.

La Figura 3.1 muestra el vector con todos los puntos de los hiperplanos, el hiperplano L_1 tiene *offset* de 0 y una cantidad de elementos igual a 1, el hiperplano L_2 tiene un *offset* de 1 y una cantidad de elementos igual a 3, y así sucesivamente con los demás hiperplanos. El número de hilos que se utiliza para procesar cada hiperplano se calcula en base a los puntos del mismo.

El Algoritmo 4 presenta el pseudocódigo de la descomposición LU incompleta en su versión paralela considerando el uso de hiperplanos.

3.1.1.2. Paralelización del cálculo del Resto

En el cálculo del resto sólo aparece una dependencia de datos en las coordenadas del plano K, por lo tanto se puede paralelizar en los planos IJ e iterar en K. El Algoritmo 5 resume el mecanismo antes descrito, RES es el vector que contiene el resto y FF es el vector solución. La cantidad de unidades de procesamiento que se utilizan es calculada según la cantidad de puntos de los planos IJ.

[1] Típicamente conocido como DAG (Directed Acyclic Graph).

Algoritmo 4 Factorización LU Incompleta

procedure FACTORIZACIÓN LU INCOMPLETA
  Entrada : $\mathbf{AB}, \mathbf{AT}, \mathbf{AW}, \mathbf{AE}, \mathbf{AS}, \mathbf{AN}$ 7 vectores conteniendo las diagonales de la matriz A
  Salida : $\mathbf{LB}, \mathbf{LW}, \mathbf{LS}, \mathbf{LP}, \mathbf{UN}, \mathbf{UT}, \mathbf{UE}$ 7 vectores conteniendo las diagonales de las matrices L y U (factorización incompleta LU)
  for i=1 **to** (# hiperplanos) **do**
    $OffSet = VectorOffSet(i)$ offset del hiperplano
    $Cantidad = VectorCantidadPuntos(i)$ cantidad de puntos del hiperplanos
    for ijk=OffSet **to** Cantidad **do**
      $LB(ijk) = AB(ijk)[1 + \alpha(UN((ijk) - N_i \cdot N_j) + U_E((ijk) - N_i \cdot N_j))]^{-1}$
      $LW(ijk) = AW(ijk)[1 + \alpha(UN((ijk) - 1) \cdot UT((ijk) - 1))]^{-1}$
      $LS(ijk) = AS(ijk)[1 + \alpha(UE((ijk) - N_i) \cdot UT((ijk) - N_i))]^{-1}$
      $LP(ijk) = AP(ijk)....LB(ijk)....UE((ijk) - N_i \cdot N_j)$
      $UN(ijk) = ...LP(ijk)...$
      $UT(ijk) = ...LP(ijk)...$
      $UE(ijk) = ...LP(ijk)...$
    end for
  end for
end procedure

Algoritmo 5 Cálculo del Resto

procedure CÁLCULO DEL RESTO
  Entrada : $\mathbf{AB}, \mathbf{AT}, \mathbf{AW}, \mathbf{AE}, \mathbf{AS}, \mathbf{AN}$ diagonales de la matriz A, **QE** vector de términos independientes, **FF** vector solución
  Salida : $resN$ valor del resto.
  for k=1 **to** N_k **do**
    for i=1 **to** N_i **do**
      for j=1 **to** N_j **do**
        $ijk = j + i \cdot N_j + k \cdot N_i \cdot N_j$
        $RES(ijk) = QE(ijk) - AP(ijk) \cdot FF(ijk) - AE(ijk) \cdot FF(ijk + 1)$
        $-AW(ijk - 1) \cdot FF(ijk + N_i))$
        $-AN(ijk) \cdot FF(ijk - N_i) - AS(ijk) \cdot FF(ijk + N_i \cdot N_j)$
        $-AT(ijk) \cdot AB(ijk) \cdot FF(ijk + N_i \cdot N_j)$
        $resN = resN + |RES(ijk)|$
      end for
    end for
  end for
end procedure

3.1.1.3. Paralelización de la Sustitución Hacia Adelante (Fordward)

La paralelización de la sustitución hacía adelante (ver el Algoritmo 6) es igual a la paralelización de la factorización LU incompleta, es decir se trabaja por hiperplanos y con igual configuración para el uso de los hilos.

Algoritmo 6 Sustitución hacia adelante

procedure FORDWARD
  $Entrada$: **LB**, **LW**, **LS**, **LP** P diagonales de la matriz L
  $Salida$: **RES** vector resultado de la sustitución hacia adelante.
  for i=1 **to** (# hiperplanos) **do**
    $OffSet = VectorOffSet(i)$ offset del hiperplano
    $Cantidad = VectorCantidadPuntos(i)$ cantidad de puntos del hierplanos
    for ijk=OffSet **to** Cantidad **do**
      $RES(ijk) = (RES(ijk) - LB(ijk) \cdot RES(ijk - N_i \cdot N_j) \cdot LW(ijk)$
        $\cdot RES(ijk - 1) - LS(ijk) \cdot RES(ijk - N_i)) \cdot LP(ijk)$
    end for
  end for
end procedure

3.1.1.4. Paralelización de la Sustitución Hacia Atrás (BackWard)

La paralelización de la sustitución hacia atras es similar al procedimiento anterior pero el sentido de la iteración sobre los hiperplanos es el opuesto, es decir en sentido decreciente. En el Algoritmo 7 se presenta el método.

Algoritmo 7 Sustitución hacía atrás

procedure BACKWARD
  $Entrada$: **UN**, **UT**, **UE** diagonales de la matriz U
  $Salida$: **RES** vector resultado de la sustitución hacia atrás.
  for i=# hiperplanos **downto** (1) **do**
    $OffSet = VectorOffSet(i)$ offset del hiperplano
    $Cantidad = VectorCantidadPuntos(i)$ cantidad de puntos del hierplanos
    for ijk=OffSet **to** Cantidad **do**
      $RES(ijk) = RES(ijk) - UN(ijk) \cdot RES(ijk + N_i) - UE(ijk) \cdot RES(ijk + 1)$
        $-UT(ijk) \cdot RES(ijk - N_i \cdot N_j)$
      $FF(ijk) = FF(ijk) + RES(ijk)$ Se actuliza la solución
    end for
  end for
end procedure

3.1.2. Paralelización del BiCGStab

Este método es apliamente utilizado para el tratamiento de diversos tipos de problemas que implican la resolución de un sistema lineal de ecuaciones. El propósito principal de su implementación fue la comparación de desempeño con el SIP cuando son utilizados sobre GPUs. En este contexto se escogió implementarlo solamente sobre CUDA, y la paralelización fue realizada explotando en cada una de las operaciones del Algoritmo 3 rutinas ofrecidas por las bibliotecas CUBLAS y CUSPARSE.

Este método cuenta con más operaciones que el SIP, por esa razón contiene más cuellos de botella a la hora de incluir técnicas de programacion paralelas. Por otro lado, estas operaciones son entre vectores y entre vectores con matrices, desde este punto de vista cuenta con una paralelización más sencilla que su contraparte del SIP. La estrategia de paralelización fue enviar las operaciones más costosas para que sean computadas por la GPU manteniendo operaciones que implican poco cómputo en CPU. Específicamente, como se dijo antes, se aprovecharon las funciones ofrecidas por CUBLAS para las operaciones sobre vectores (BLAS-1 y BLAS-2) y las de la biblioteca CUSPARSE para las operaciones sobre matrices dispersas, específicamente spmv.

3.2. Parlelización del esquema Explícito

Este esquema presenta menor dependencia de datos y su paralelización está acotada en un principio (solamente) por el número de unidades de procesamiento disponibles. Por ello se buscó maximizar el grado de paralelización y a la vez mantener acotado el *overhead* introducido por explotar el paralelismo de forma de alcanzar un método escalable en el tamaño del dominio. Como se trabaja con un volumen cúbico o rectangular, primeramente se crea un procedimiento por cada caso de borde, es decir:

- 8 procedimientos para procesar los vértices.
- 12 procedimientos para las aristas.
- 6 procedimientos para las caras.

Por último, se emplea un procedimiento para procesar el interior del volumen, es decir todas las celdas que no pertenecen a los bordes.

Todos estos procedimientos pueden correr en paralelo ya que solo dependen de datos calculados en pasos de tiempo anteriores. El proceso que maneja más datos es el que procesa las celdas interiores. En este caso, se diseñó un procedimiento que paraleliza por planos XY e itera en Z, de esta forma el procedimiento es más escalable porque en caso que se hubiera trabajado directamente en XYZ, no se tendría un buen balance entre cómputos y el overhead introducido por la división y sincronización de los datos. Además, de utilizar esta opción, no se podría crecer demasiado en las dimensiones del problema en GPU debido a las limitantes en los índices de los bloques. Este diseño se basó en la propuesta de Micikevicius [48].

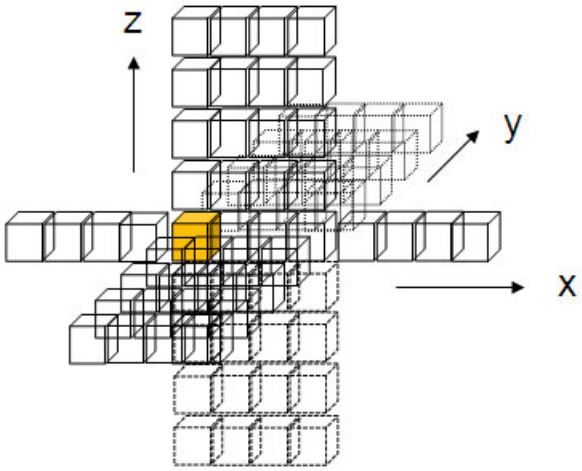

Figura 3.2: Procesamiento de cómputo de una celda en el esquema explícito.

Para procesar una celda, es necesario obtener el valor de 7 celdas en total, una de estas celdas pertenece al plano XY anterior, otra en el plano XY posterior y 5 celdas del plano XY actual. La estrategia que se siguió fue que cada hilo procese un punto del plano XY actual, que el plano XY actual se cargue en memoria compartida y que el plano actual pase a ser el plano anterior en la siguiente iteración, evitando así doble acceso a memoria para una misma celda. El único plano que se carga desde la memoria principal en cada iteración, es el plano posterior, y en la siguiente iteración los datos de dicho plano son copiados a memoria compartida (plano actual). En al Figura 3.3, el plano del medio es el plano actual y es el que se carga en memoria compartida. Notar que del plano anterior y posterior sólo se necesitan un punto de cada uno.

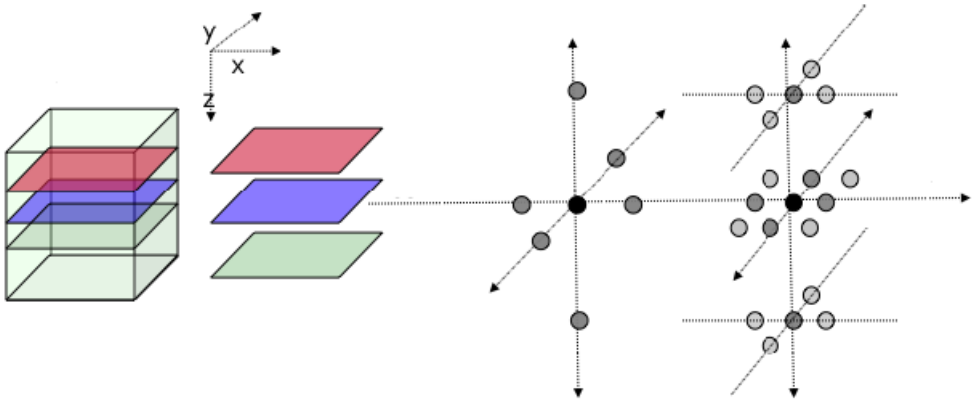

Figura 3.3: Procesamiento por planos y puntos que pertenecen a cada plano en el esquema explícito (plano anterior en rojo, plano actual en azul y plano posterior en verde).

La configuración de bloques e hilos que se utilizó se basó en el tamaño del plano XY, pero sin tener en cuenta las celdas que pertenecen a lados, vértices o aristas. Considerando

que, para calcular los valores de celdas interiores que están pegadas a una cara es necesario saber el valor de la celda que pertenece a la cara correspondiente, en memoria compartida se carga el plano XY a procesar más las caras. La Figura 3.4 muestra gráficamente los datos a cargar en memoria compartida.

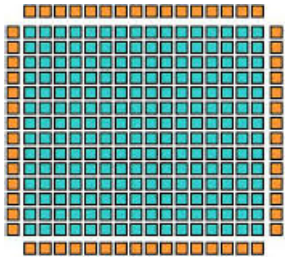

Figura 3.4: Celdas del plano XY más las celdas de las caras.

El proceso de cálculo de las celdas de las caras de la superficie es similar al de las celdas internas, en el sentido del uso de la memoria compartida, pero con la diferencia que no se itera en plano Z. Para el cálculo de las aristas, los bloques se definen como vectores, o sea en una única dimensión, y se cargan en memoria compartida dos valores externos del bloque o sea los vértices, y el sentido de la iteración en X o en Y o en Z, depende de cada caso. El cálculo de los vértices es el más sencillo ya que sólo se calcula un valor.

Como resumen de la mecánica antes enumerada se puede ver el Algoritmo 8 que describe el cálculo en el esquema explícito.

Algoritmo 8 Algoritmo Método Explícito

```
1: Entrada: VectorEntrada es el vector que contiene la Solución Inicial.
2: Salida: VectorSalida es el vector que contiene la Solución Final.
3: Inicializo(VectorEntrada)
4: for t=0 to Tiempo Final do
5:     KernelInterior(VectorSalida, VectorEntrada)
6:     KernelVertice1(VectorSalida, VectorEntrada)
7:     .....
8:     KernelArista1(VectorSalida, VectorEntrada)
9:     .....
10:    KernelCara1(VectorSalida, VectorEntrada)
11:    .....
12:    Sincronizacion
13:    VectorEntrada = VectorSalida
14: end for
```

3.3. Parlelización del esquema Predictivo-Correctivo

La paralelización de este esquema es similar al caso explícito, recordar que este método está basado en su contraparte desarrollada para el caso explícito, solo agrega el cálculo de

estimación del error del método. En cuanto al cálculo del error puede ser estimado mediante la diferencia entre dos soluciones consecutivas. La otra diferencia es el paso iterativo que se agrega, que implica un vector auxiliar para el cálculo de las iteraciones. El mencionado vector extra es el causante que aumenten los requerimientos de almacenamiento cuando se compara este método con la implementación de otros.

Obviamente, esta variante es más lenta que el método explícito por las iteraciones extras y el cálculo del error. Por último, se resume el método en el Algoritmo 9.

Algoritmo 9 Método Predictivo-Correctivo

```
1: Entrada: VectorEntrada es el vector que contiene la Solución Inicial.
2: Salida: VectorSalida es el vector que contiene la Solución Final.
3: VectorIteracion es el vector auxiliar que se utiliza en las iteraciones.
4: Inicializo(VectorEntrada)
5: VectorIteracion = VectorEntrada
6: for t=0 to Tiempo Final do
7:     k = 0
8:     while (k < maxIteraciones and error > maxError) do
9:         KernelInterior(VectorSalida, VectorIteracion, VectorEntrada)
10:        KernelVertice1(VectorSalida, VectorIteracion, VectorEntrada)
11:        .....
12:        KernelArista1(VectorSalida, VectorIteracion, VectorEntrada)
13:        .....
14:        KernelCara1(VectorSalida, VectorIteracion, VectorEntrada)
15:        .....
16:        Sincronizacion
17:        error = abs(VectorSalida − VectorIteracion)
18:        VectorIteracion = VectorSalida
19:        k = k + 1
20:    end while
21:    VectorEntrada = VectorIteracion
22: end for
```

Capítulo 4

Arquitecturas de Hardware

En este capítulo se hacer una breve reseña de la evolución histórica, características actuales y los detalles más importantes de los procesadores multi-core y los procesadores masivamente paralelos, y específicamente, se pone especial atención en las tarjeta gráficas (que soporten CUDA[1]) que son el foco del presente trabajo.

4.1. Multi-core

La industria de los microprocesadores sigue teniendo gran importancia en el curso de los avances tecnológicos desde su llegada en los años 70 [2]. El creciente mercado y la demanda de un mejor rendimiento impulsaron a la industria a fabricar chips más rápidos. Una de las técnicas clásicas y más probada para mejorar el rendimiento es llevar a mayor frecuencia el reloj del chip, esto permite al procesador ejecutar las instrucciones de los programas en un tiempo mucho más rápido [10], [55]. La industria ha seguido la tendencia antes descrita desde el año 1983 hasta el año 2002. En cuanto a valores de velocidades, esta situación se traduce en una evolución desde los 5 MHz hasta los aproximadamente 3 GHz al final del período [59]. También se han ideado otras técnicas adicionales para mejorar el rendimiento de los computadores, incluido el procesamiento en paralelo, el paralelismo a nivel de datos y el paralelismo a nivel de instrucciones, que han demostrado ser muy eficaces en difierentes contextos [27].

Una de estas técnicas, que mejora el rendimiento significativamente, es el aumento del uso de procesadores Multi-Core. Los procesadores Multi-Core han estado disponible desde la década pasada pero, sin embargo, han ganado más importancia en estos últimos años debido a las limitantes de la tecnología que presentan los procesadores mono-núcleo [77]. En especial esta clase de hardware permite enfrentar los desafíos tecnológicos que se buscan hoy en día, tales como obtener un alto rendimiento y, al mismo tiempo, alcanzar valores controlados de consumo energético.

[1]Es decir, de las líneas modernas de la empresa NVIDIA.

4.1.1. Evolución histórica

Impulsados por un mercado centrado en el rendimiento, los microprocesadores han sido diseñados por muchos años teniendo en cuenta principalmente el rendimiento y el costo económico. Gordon Moore, fundador de Intel Corporation, predijo que el número de transistores en un chip se duplicaría una vez cada 18 meses para satisfacer esta demanda cada vez mayor. Esta predicción se conoce popularmente como Ley de Moore en la industria de los semiconductores [63], [67].

La avanzada tecnología de fabricación de chips ha permitido integrar mil millones de transistores en un único chip para mejorar el rendimiento. Sin embargo, la regla de Pollack dice que el aumento de rendimiento debido a los avances en micro-arquitectura es aproximadamente proporcional a la raíz cuadrada de aumento de la complejidad [7]. Esto significaría que duplicar la lógica en un núcleo de procesador sólo mejoraría el rendimiento computacional en un valor en el entorno de 40 %.

Con las avanzadas técnicas de fabricación de chips viene otro gran cuello de botella, el problema de la disipación de energía. Los estudios han demostrado que la fuga de energía de los transistores aumenta a medida que el tamaño del chip se contrae más y más, lo que aumenta la disipación de energía estática a grandes valores como se muestra en la Figura 4.1. Como se dijo anteriormente, una alternativa para mejorar el rendimiento es aumentar la frecuencia del reloj del procesador que permite directamente una ejecución más rápida de los programas. Sin embargo, la frecuencia está limita otra vez a 4GHz. Notar que, actualmente cualquier aumento más allá de esta frecuencia aumenta la disipación de energía en forma importante. Es más, el consumo de energía ha aumentado a niveles tan altos que los microprocesador refrigeradas tradicionalmente por aire pueden requerir refrigeración por líquido. Es decir, los diseñadores finalmente estan llegando al límite de la cantidad de energía que un microprocesador podría disipar [56]. En otras palabras, la industria de los semiconductores, una vez impulsada por el rendimiento como principal objetivo de diseño, se está impulsando hoy por otras consideraciones importantes tales como los costos de fabricación de chips, la tolerancia a fallos, eficiencia de potencia y la capacidad de disipación de calor, etc. Este hecho queda expresado a la frase de "La duración de la batería y las restricciones de costes del sistema hacen que el equipo de diseño considere el consumo energético sobre el rendimiento en tal escenario" [9].

La situación descrita en los párrafos anteriores llevó al desarrollo de procesadores Multi-Core que han sido eficaces para enfrentar estos desafíos.

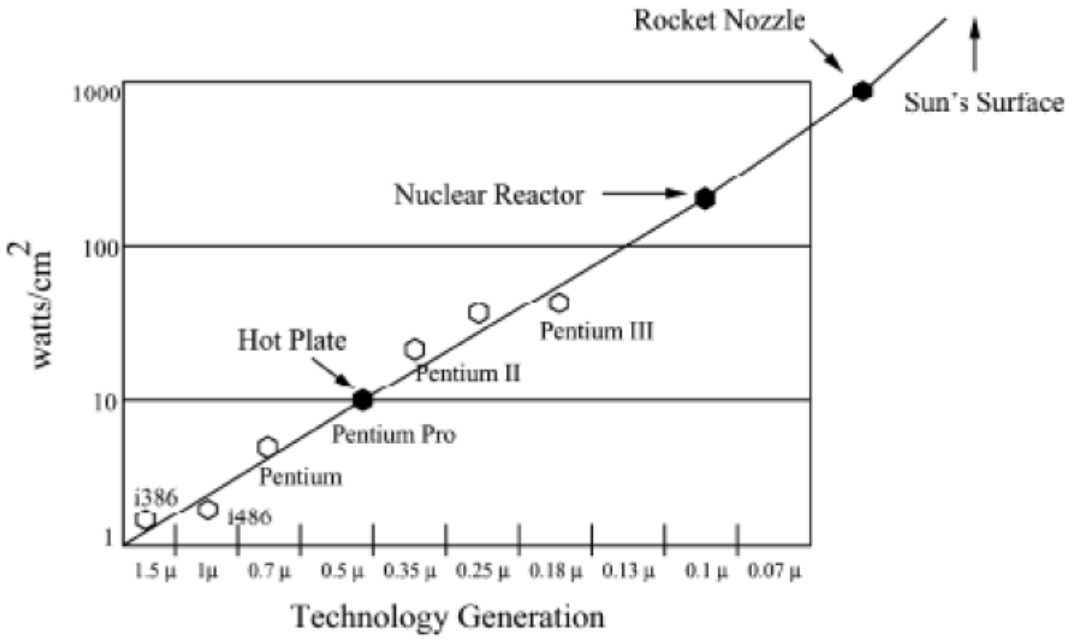

Figura 4.1: Disipación de energía de los procesadores de las distintas familias de Intel, según la dimensión del chip. Extraído de [75].

4.1.2. Procesadores multi-core

Un procesador multi-core es en palabras sencillas un único procesador que contiene varios núcleos en un chip [77]. Los núcleos son unidades funcionales compuestas de unidades de cómputo y memorias cachés. Estos múltiples núcleos en un sólo chip se combinan para replicar el rendimiento de un solo procesador más rápido. Los núcleos individuales en un procesador multi-core no necesariamente funcionan tan rápido como los procesadores de un sólo núcleo de mayor rendimiento, pero, mediante el manejo de más tareas en paralelo pueden mejorar el rendimiento general [24].

El aumento de rendimiento puede ser visto por la comprensión de la manera en que los procesadores de núcleo único y Multi-Core ejecutan programas. Los procesadores de núcleo simple que ejecutan varios programas asignarían un intervalo de tiempo para trabajar en un programa y luego asignarían diferentes intervalos de tiempo para los programas restantes. Si uno de los procesos tarda más en completarse entonces todo el resto de los procesos comienzan a rezagarse. Sin embargo, en el caso de los procesadores Multi-Core si se tienen varias tareas que se pueden ejecutar en paralelo (al mismo tiempo), cada una de ellas será ejecutada por un núcleo en paralelo, incrementando así el rendimiento como se muestra en la Figura 4.2.

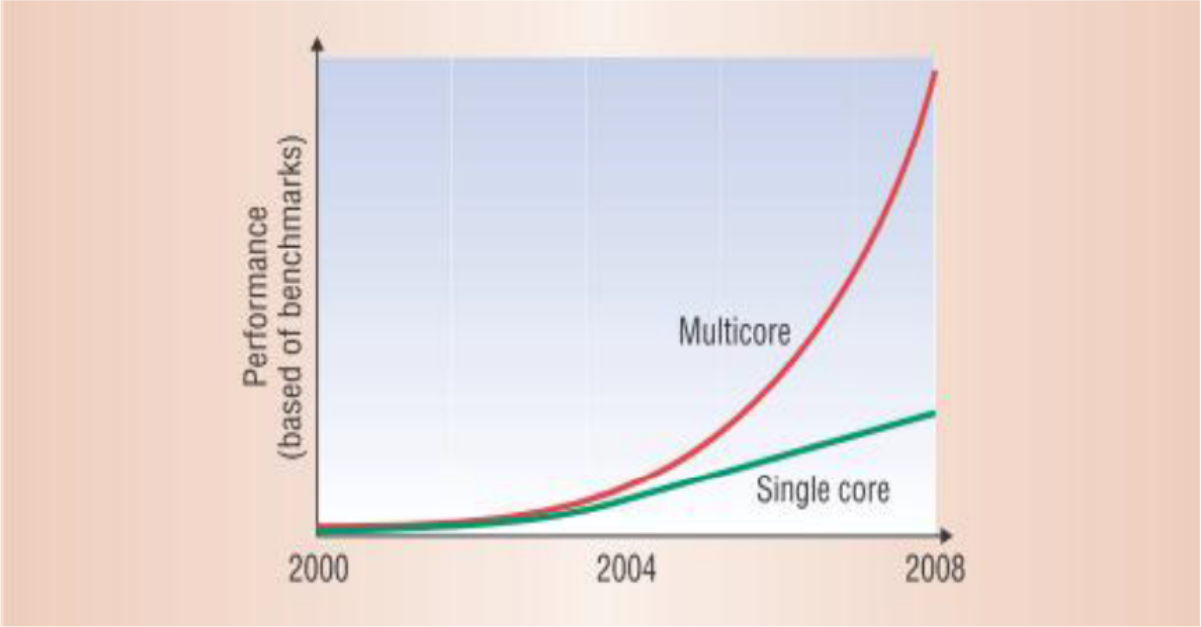

Figura 4.2: Desempeño computacional de procesadores multi y mono-núcleo en base a tests realizados por Intel usando los benchmarks SPECint2000 y SPECCfp2000 según la evolución histórica. Extraído de [5].

En general, los múltiples núcleos dentro del chip no se configuran a su frecuencia más alta, sino que su capacidad de ejecutar programas en paralelo es lo que finalmente contribuye al rendimiento general, permitiendoles alcanzar en la mayoría de los casos consumos de energía menor. Esta situación se muestra en la Figura 4.3.

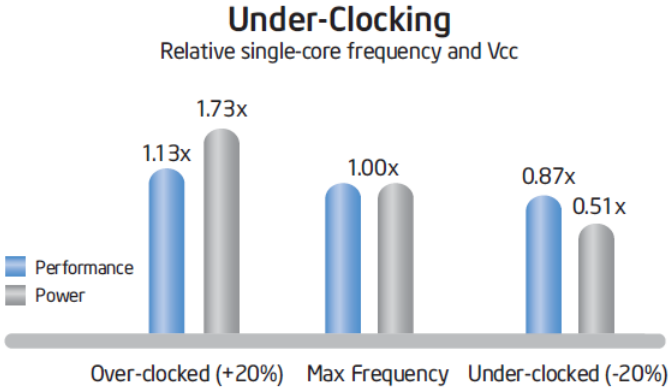

Figura 4.3: Consumo energético de CPUs. Un aumento de la frecuencia de reloj en un 20 % a un núcleo único ofrece una ganancia de rendimiento del 13 %, pero el consumo energético aumenta un 73 %. Por el contrario, la disminución frecuencia de reloj en un 20 % reduce el consumo energético en 49 %, pero tiene sólo un 13 % de pérdida de rendimiento. Extraído de [59].

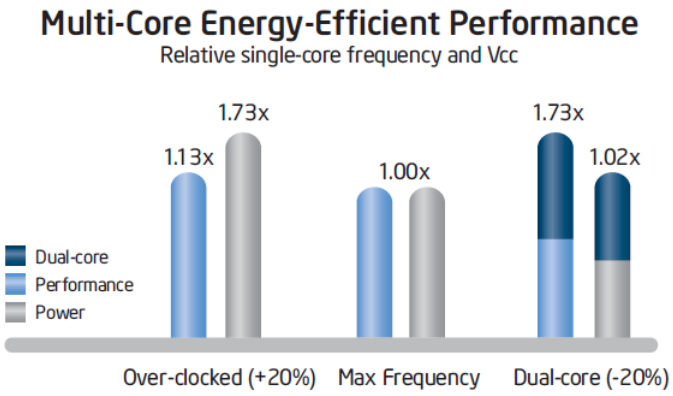

Figura 4.4: Consumo energético de CPUs. Notar que si se añade un segundo núcleo al equipo del ejemplo de la Figura 4.3 da como resultado un procesador de doble núcleo que, con la reducción de un 20 % en la frecuencia de reloj obtiene un 73 % más de rendimiento mientras que consume lo mismo que un procesador con un núcleo único a la frecuencia máxima. Extraído de [59].

Los procesadores Multi-Core son generalmente diseñado para que los núcleos no utilizados puedan ser apagados (o, al menos, que se ejecuten a niveles inferiores de desempeño) o encendidos cuando sea necesario, lo que contribuye al ahorro global del consumo de energía. Los procesadores de varios núcleos podrían implementarse de muchas maneras según los requisitos de las diferentes aplicaciones. Podrían ser implementados ya sea como un grupo de núcleos heterogéneos o como un grupo de núcleos homogéneos, esto sería un procesador heterogéneo.

En la arquitectura homogénea, todos los núcleos de la CPU son idénticos y se aplica el enfoque divide y conquista para mejorar el rendimiento general del procesador mediante la división de aplicaciones altamente demandantes desde el punto de vista computacional en aplicaciones más livianas que se ejecutan en paralelo. Otro de los principales beneficios del uso de un procesador Multi-Core homogéneo son la reducción de la complejidad del diseño, la reutilización, reducción del esfuerzo de verificación y, por lo tanto, es más fácil cumplir el criterio del tiempo de comercialización.

Por otra parte los núcleos heterogéneos consisten en núcleos diferentes, que cada uno puede ejecutar una aplicación particular con fuerte requerimientos que se ajustan a las características del núcleo que la procesa. Un ejemplo podría ser el procesador DSP que ejecuta aplicaciones multimedia que requieren cálculos matemáticos pesados, está formado por un núcleo complejo para aplicaciones intensivas computacionalmente y un núcleo más secillo que aborda aplicaciones menos intensiva [55]. Los procesadores Multi-Core también podrían ser implementados como una combinación de sistemas homogéneos y heterogéneos para mejorar el rendimiento teniendo las ventajas de ambas implementaciones. CELL, un procesador Multi-Core que fue un intento no muy exitoso de la empresa IBM, siguió este enfoque y contenía un único microprocesador de propósito general y ocho microprocesadores orientados a aplicaciones específicas, esta plataforma ha demostrado ser eficiente en cuanto a rendi-

miento [55]. Si bien los procesadores CELL no fueron un éxito comercialmente, sirvieron de ejemplo para varios esfuerzos actuales, como por ejemplo los procesadores Jetson TK1 de NVIDIA.

4.1.3. Desafíos de los procesadores multi-core

A pesar de las muchas ventajas que ofrecen los procesadores Multi-Core, hay algunos desafíos importantes que esta tecnología enfrenta. Uno de los principales problemas observados, es con respecto a ciertos programas no paralelizados que se ejecutan más lentamente en los procesadores Multi-Core en comparación con los procesadores de un solo núcleo. Notar que, las aplicaciones en sistemas Multi-Core no se aceleran automáticamente a medida que se incrementan los núcleos [10]. Por el contrario, los programadores deben implementar aplicaciones que exploten el creciente número de procesadores en un entorno Multi-Core sin alargar el tiempo necesario para el desarrollo del software. La mayoría de las aplicaciones utilizadas hoy en día se implementaron para ejecutarse en un solo procesador. Aunque las empresas de software pueden desarrollar programas de software capaces de utilizar al máximo los procesador Multi-Core, el gran desafío que enfrenta la industria es cómo transformar programas de software heredados en programas de software que aprovechen los procesadores Multi-Core. Rediseñar los programas aunque fuera posible, en realidad no es una decisión tecnológica en el entorno actual. Es más bien una decisión de negocio en la que las empresas tienen que decidir si van a seguir un rediseño de los programas de software teniendo en cuenta parámetros claves como es el tiempo de comercialización, la satisfacción del cliente y la reducción de costos. Estos aspectos están por fuera del alcance del presente estudio.

La industria está abordando este problema mediante el diseño de compiladores que pueden transformar los programas de software heredados a programas que serán capaces de utilizar la potencia de los procesadores multi-core. Los compiladores podrían generar instrucciones que se puedan ejecutar en paralelo. Intel lanzó importantes actualizaciones para las herramientas de C++ y Fortran apuntando a los programadores que explotan el paralelismo en los procesadores Multi-Core. También se ha impulsado el uso de la API OpenMP (Open Multi-Processing), que proporciona directivas para implementar códigos multiproceso de manera sencilla y eficiente.

En segundo lugar, las interconexiones en el chip se están convirtiendo en un cuello de botella crítico para cumplir con el rendimiento de los chips de varios núcleos [36]. Con el aumento del número de núcleos se hacen más notorios los retrasos de interconexión cuando los datos tienen que ser movidos desde el chip Multi-Core hacia la memoria en particular [31]. Notar que en muchos casos el rendimiento del procesador realmente depende de la rapidez con la que una CPU puede obtener datos en lugar de la rapidez con la que puede operar para evitar escenarios de inanición de datos [50]. Buffering e integración más inteligente de la memoria y los procesadores son algunas técnicas clásicas que han intentado abordar este problema. Aumento de la complejidad del diseño debido a las posibles condiciones de carrera[2] a medida que aumenta el número de núcleos en un entorno de varios núcleos. En

[2]Múltiples subprocesos que acceden simultáneamente a datos compartidos pueden dar lugar al error conocido como condición de carrera de datos [45], esta situación se da cuando al menos uno de los procesos efectúa actividades de escritura en memoria.

estos entornos de hardware, las estructuras de datos están abiertas al acceso desde todos los otros núcleos cuando uno específico lo está actualizando. En el caso que un núcleo secundario acceda a los datos incluso antes de que el primer núcleo termine de actualizar la memoria, el núcleo secundario procesará datos erróneos.

4.2. Procesadores Gráficos (GPUs)

Dadas las características de las tarjetas gráficas actuales como su bajo costo, elevada potencia de cálculo, y grandes capacidades de computación paralela, hoy en día se consideran muy útiles para usos diferentes de los que originalmente fueron concebidas. Desde el punto de vista del hardware las GPUs se han convertido en procesadores extremadamente flexibles y potentes. El uso y mercado de las GPUs, ya no es sólo como tarjetas gráficas para disfrutar de los últimos videojuegos con la mejor calidad, si no que también en el sector de la computación de alto desempeño (HPC por sus siglas en inglés) permitiendo acelerar ordenadores de escritorio para que puedan ejecutar miles de millones de operaciones de punto flotante en tiempos acotados.

4.2.1. Evolución histórica

Los procesadores gráficos fueron creados como coprocesadores, diseñados específicamente para realizar operaciones de punto flotante necesarias en el renderizado de gráficos en 3D durante la década del 80. Nacieron con un único procesador y luego fueron adquiriendo otras unidades procesamiento para tareas específicas.

Anteriormente se mostró cómo los procesadores evolucionaron en velocidad de reloj, y en la cantidad de núcleos. Mientras tanto, y en paralelo los procesadores gráficos tuvieron un cambio dramático. A finales de los años ochenta y principios de los noventa, los sistemas operativos con gráficos, como Microsoft Windows, ayudaron a crear un mercado para un nuevo tipo de procesador. A principios de los años noventa, los usuarios comenzaron a comprar tarjetas aceleradoras de gráficos para sus ordenadores personales.

Al mismo tiempo, en el mundo de la informática profesional, la empresa con nombre Silicon Graphics popularizó en la década de 1980 el uso de gráficos tridimensionales en una variedad de mercados. Entre otros, esta expansión incluyó aplicaciones para gobiernos, el ejercito, la investigaciones científica, así como el mundo del cine, ya que proporcionó herramientas para crear impresionantes efectos cinematográficos. En 1992, la empresa Silicon Graphics abrió la interfaz de programación de su hardware mediante la liberación de la biblioteca OpenGL. Silicon Graphics impulsó a OpenGL para ser utilizado como un estándar, independiente de la plataforma de hardware utilizado, para escribir aplicaciones de gráficos en 3D.

A mediados de los años noventa, la demanda de aplicaciones que empleaban gráficos 3D habían escalado rápidamente, preparando el escenario para dos desarrollos bastante sig-

nificativos. Primero, el lanzamiento de juegos en primera persona como Doom, Duke Nukem 3D y Quake ayudaron a crear ambientes 3D progresivamente más realistas para juegos de computadores. En los juegos de PC, la popularidad de la naciente primera persona aceleró significativamente la adopción de gráficos en 3D. Al mismo tiempo, compañías como NVIDIA, ATI Technologies,y 3dfx Interactive comenzaron a liberar aceleradores gráficos que eran asequibles económicamente. Estos desarrollos posicionaron a los gráficos en 3D como una tecnología que ocuparía un lugar prominente en los próximos años.

En 1999, con el lanzamiento de NVIDIA GeForce 256 se impulsó aún más las capacidades alcanzable por los consumidores de hardware de gráficos. Por primera vez, los cálculos de transformación e iluminación se realizaban directamente en el procesador gráfico, mejorando así el potencial para aplicaciones aún más interesantes visualmente. La tarjeta gráfica GeForce 256 fue el comienzo de una progresión natural donde cada vez más las etapas del pipeline gráfico se implementaría directamente en el procesador gráfico en detrimento de utiliar el procesador central.

Desde el punto de vista de la computación paralela, el lanzamiento de la serie GeForce 3 de NVIDIA, en 2001 representa posiblemente el avance más importante en la tecnología gráfica. La serie GeForce 3 fue el primer chip de la industria informática en implementar el entonces nuevo estándar de DirectX 8.0 de Microsoft. Es decir, por primera vez los desarrolladores tuvieron cierto control sobre los cálculos que se realizaban el hardware de sus tarjetas gráficas.

No sería hasta cinco años después del lanzamiento de la serie GeForce 3 que las GPUs estarían lista por primera vez para ofrecer la configuración/programación del pipeline gráfico. En noviembre de 2006, NVIDIA dio a conocer la primera GPU DirectX 10 de la industria, la GeForce 8800 GTX. Estas tarjetas gráficas fueron también las primeras GPUs que se construirá con la arquitectura CUDA de NVIDIA. Esta arquitectura incluyó varios componentes nuevos diseñados estrictamente para la GPU y que permitieron aliviar muchas de las limitaciones que impedían a los procesadores gráficos anteriores ser legítimamente útil para la computación de propósito general. Principalmente, se diseñó una homogenización y unificación de los núcleos de las tarjetas. Hasta ese momento tanto los pixels como los vertex shaders eran unidades de proceso separadas (procesadores dedicados). En las arquitecturas anteriores los pixels shaders (procesadores de pixeles) se encargaban del aspecto y color de la textura, mientras que los vertex (procesadores de vértices) se encargaban de la alteración de éstas según su posición en el mapa 3D (x, y, z). La falencia principal de estas arquitecturas anteriores era que las imágenes a procesar en general no contaban con igual número de vértices que de pixeles, sino todo lo contrario. Esta situación generaba que existieran sobre-cargas para los procesadores de un tipo (pixeles/vértices), mientras que los procesadores del otro tipo eran subutilizados. Con la unificación de procesadores, cada unidad es capaz de procesar tanto pixeles como vértices según sea requerido, de esta manera se maximiza la eficiencia en la utilización de los recursos de una tarjeta, ya que se puede hacer un balance de carga según lo requiera la aplicación, si se necesita mucha potencia en vértices o en pixels no estarán limitados por las unidades específicas de cada cual, sino que podrán hacer uso de todas las unidades si es necesario.

Desde el punto de vista de software este cambio fue revolucionario. Por un lado contar con un único tipo de procesador simplificó su uso (y programación). Por otro lado, y más

importante, el nuevo esquema exigía que el procesamiento gráfico se adaptara en su cálculo a los nuevos procesadores generales, implicando que los cálculos sean re-entrantes en dichos procesadores.

En la Figura 4.5 se pueden observar la evolución histórica de las arquitecturas de GPUs de NVIDIA, en cuanto a desempeño, consumo energético y las principales características que introdujo cada generación.

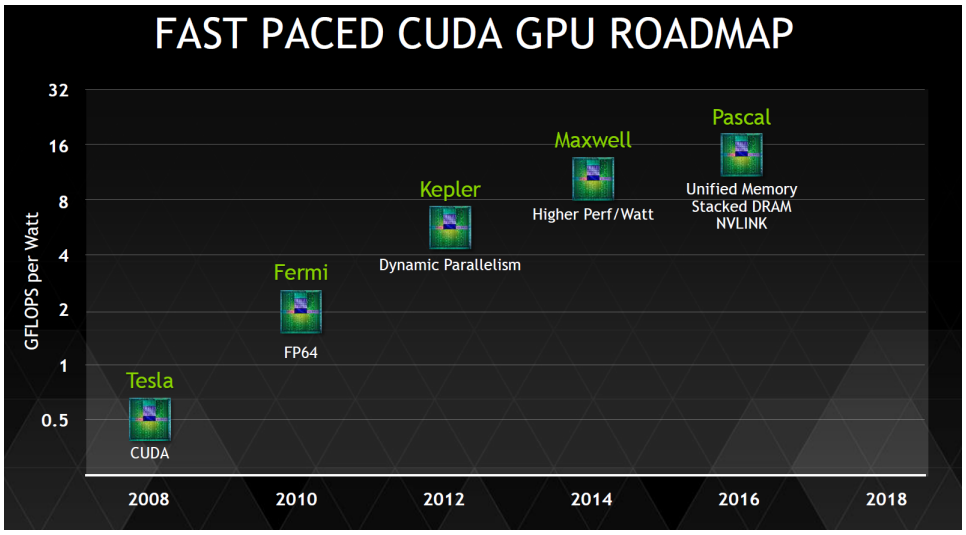

Figura 4.5: Rendimiento de las diferentes arquitecturas de NVIDIA, medidas en Gflops por watt consumido. Extraído de http://www.nvidia.com.

La arquitectura Fermi fue lanzado en 2010, los ingenieros de NVIDIA se propusieron diseñar una nueva arquitectura de GPU que abatiera gran parte de las limitantes de su predecesora. La arquitectura define los bloques de construcción de una GPU, cómo están conectados y cómo funcionan. Nombrado como el físico nuclear italiano Enrico Fermi, esta arquitectura incorpora el procesamiento de geometría completa a la GPU, permitiendo una técnica clave de DirectX 11 llamada tessellation con asignación de desplazamiento. Esta arquitectura soporta hasta 6 GB de memoria RAM GDDR5 con una interfaz de memoria de 384 bits. Cada núcleo CUDA contiene una Unidad Aritmética-Lógica (ALU) y una unidad de punto fotante (FPU). A diferencia de sus predecesoras, las GPUs Fermi soportan el estándar IEEE 754-2008 de aritmética en punto flotante de doble precisión, proporcionando la operación $fusedmultiply - add$ (FMA), la cual realiza la multiplicación y adición con un solo paso final de redondeo, evitando así la perdida de precisión en la adición. En el caso de la GPU C2070, los núcleos CUDA funcionan a una frecuencia de 1.15GHz, con un consumo energético de 238 W.

La arquitectura Kepler salió al mercado en 2012, con esta familia de GPUs NVIDIA dió a conocer sus planes para unirse al espacio de juego en la nube. Se desarrollan específicamente para la GRID y son eficientes en la configuración de HPC. Tiene clústeres de procesamiento de gráficos (o GPCs por su sigla en inglés), cada uno de los cuales contiene cuatro unidades multiprocesador de streaming (o SM). Además incluye GPU Boost para el overclocking automático. La arquitectura Kepler incluye un completo rediseño del multiprocesador, el cual

pasa a llamarse SMX. Cada uno de los (como máximo) 15 SMX, incluye 192 núcleos CUDA, que al igual que en Fermi cuentan con una ALU y una FPU que soporta el estándar IEEE 754-2008, 64 unidades de doble precisión, 32 unidades de funciones especiales y 32 unidades de lectura/escritura. A diferencia de los SM de arquitecturas previas, los SMX de Kepler utilizan el reloj principal de la GPU en lugar de utilizar el reloj del shader, que trabaja al doble de frecuencia. El reloj del shader fue introducido en la arquitectura Tesla G80 y utilizado desde entonces en las arquitecturas Tesla y Fermi. Si bien utilizar un reloj más rápido para las unidades de ejecución logra realizar más trabajo con menos unidades de procesamiento, lo cual en principio es una ventaja, pero la circuitería para trabajar con el reloj rápido demanda un mayor consumo energético. Por este motivo, la arquitectura Kepler agrega más unidades de procesamiento que operan a una frecuencia menor, obteniendo así un mayor desempeño para muchas aplicaciones, sin que esto signifique un mayor costo energético. Es importante resaltar que esta arquitectura resulta ventajosa en aplicaciones que tienen un alto grado de paralelismo, capaz de explotar al máximo el gran número de unidades de procesamiento. Notar que, para aplicaciones que tienen un nivel de paralelismo más modesto, la arquitectura Fermi, con su menor número de núcleos CUDA pero de mayor frecuencia, puede llegar a ser más conveniente.

Para hacer una idea general de las ventajas de Kepler, respecto a Fermi y Tesla, sólo basta observar la Fígura 4.6, en ella se muestra que el poder geométrico y de cómputo se ha incrementado notablemente con la nueva arquitectura de NVIDIA, como así también sus especificaciones bases de frecuencias. Como se puede ver los cambios más notable en términos cuantitativos de Kepler respecto a Fermi son: en primer lugar el número de CUDA Cores (1536 vs 512 respectivamente), la velocidad de núcleo 1066 Mhz vs 772 MHz, el poder de cómputo de 3090 GFLOPs de Kepler, respecto a los 1581 GFLOPs de Fermi en precisión simple, y con un consumo energético sensiblemente menor.

Lanzadas en 2014, las GPUs Maxwell tiene una nueva resolución dinámica (DSR) diseñada para poner contenido de calidad 4K en pantallas de baja resolución. Con una memoria RAM máxima de 12GB (Titan X o Quadro M6000) mientras que en Pascal (la arquitectura lanzada por NVIDIA en 2016) es de 32GB. Quad SLI (Scalable Link Interfaz es un método para conectar GPUs) en Maxwell soporta hasta 4 tarjetas gráficas mientras que Pascal de hasta 8 tarjetas gráficas. La primera generación de GPUs Maxwell (GM107, GM108) fue lanzada como Geforce GTX 745, 750/750 Ti, 850M/860M (GM107) y la GTX 830M/840M (GM108). Estas GPUs ofrecían pocas características nuevas para atraer al consumidor, ya que principalmente se enfocaba en la eficiencia de energía de la GPU. El Caché L2 fue incrementado de 256 KB en Kepler, a 2 MB en Maxwell, reduciendo la necesidad del ancho de banda de la memoria, el bus de la memoria fue reducido de 192 bits en Kepler a 128 bits, para ayudar a reducir el gasto de energía. El Streaming Multiprocesador diseñado para Kepler fue reestructurado y particionado, renombrándolo como SMM para esta nueva familia. La estructura del warp scheduler fue heredada de Kepler, junto con la unidad de texturas y núcleos FP64 CUDA que siguen siendo compartidos, pero el diseño de la mayoría de las unidades de ejecuciones fue particionado para que cada warp scheduler en un SMM controle un paquete de 32 núcleos FP32 CUDA, dividiéndose así en dos paquetes, uno de 8 cargar/guardar unidades y otro de 8 unidades especiales. Esto permitió un mejor manejo de recursos que en la arquitectura Kepler, y en especial ahorrando más energía cuando la carga de trabajo no es óptima para compartir recursos. NVIDIA dice que 128 CUDA cores SMM tienen el 90 % de rendimiento de 192 CUDA cores de SMX, incrementando la eficiencia energética por un

factor de 2×.

La Fígura 4.7 muestra la comparativa entre una tarjeta con tecnología Kepler y otra Maxwell de NVIDIA, como se aprecia, se sigue aumentando el rendimiento y bajando el consumo energético con cada nueva arquitectura.

GPU	GT200 (Tesla)	GF110 (Fermi)	GK104 (Kepler)
Transistors	1.4 billion	3.0 billion	3.54 billion
CUDA Cores	240	512	1536
Graphics Core Clock	648MHz	772MHz	1006MHz
Shader Core Clock	1476MHz	1544MHz	n/a
GFLOPs	1063	1581	3090
Texture Units	80	64	128
Texel fill-rate	51.8 Gigatexels/sec	49.4 Gigatexels/sec	128.8 Gigatexels/sec
Memory Clock	2484 MHz	4008 MHz	6008MHz
Memory Bandwidth	159 GB/sec	192.4 GB/sec	192.26 GB/sec
Max # of Active Displays	2	2	4
TDP	183W	244W	195W

Figura 4.6: Comparativa de las arquitecturas, Tesla, Fermi y Kepler. Extraído de [59].

Como se dijo anteriormente, lanzado en el año 2016 la arquitectura Pascal está diseñada para la mejor escalabilidad de las tarjetas gráficos. Incluye HBM2 para potenciar el ancho de banda de acceso a memoria y NVLink para mejorar el esquema de direccionamiento virtual unificado en las GPUs, y entre las CPUs y las GPUs. Por lo tanto, tiene la capacidad de permitir que los datos se muevan de 5 % a 12 % más rápido entre la memoria de las CPUs y las GPUs en comparación con PCI-Express tradicional.

GPU	GeForce GTX 680 (Kepler)	GeForce GTX 980 (Maxwell)
SMs	8	16
CUDA Cores	1536	2048
Base Clock	1006 MHz	1126 MHz
GPU Boost Clock	1058 MHz	1216 MHz
GFLOPs	3090	4612[1]
Texture Units	128	128
Texel fill-rate	128.8 Gigatexels/sec	144.1 Gigatexels/sec
Memory Clock	6000 MHz	7000 MHz
Memory Bandwidth	192 GB/sec	224 GB/sec
ROPs	32	64
L2 Cache Size	512KB	2048KB
TDP	195 Watts	165 Watts
Transistors	3.54 billion	5.2 billion
Die Size	294 mm^2	398 mm^2
Manufacturing Process	28-nm	28-nm

Figura 4.7: Comparativa de las arquitecturas, Kepler y Maxwell. Extraído de [59].

4.2.2. Arquitectura CUDA

Desde el punto de vista de hardware, mientra los multiprocsadores cuenta con algunos núcleos de cómputo complejos, las GPUs están conformadas por miles de núcleos livianos. Las CPUs tiene que ser mucho más versátiles y capaces de manejar todo tipo de tareas comunes de un computador, mientras que una GPU sólo tiene que manejar el procesamiento de imágenes, y como tal puede ser optimizado para dicho propósito. En otras palabras, las GPUs pueden manejar gráficos mejor porque los gráficos incluyen miles de cálculos livianos que necesitan ser procesados y esta tarea puede realizarse en forma concurrente. En lugar de enviar esos cálculos livianos a la CPU, que sólo puede manejar unos pocos a la vez, se envían a la GPU, que puede manejar muchos de ellos a la vez. Esto se debe a que una GPU está construida sobre una arquitectura Single Instruction Multiple Data, o SIMD[3], permitiendo a la GPU realizar operaciones en matrices de datos. Esto significa que cuando un conjunto de datos tiene la misma secuencia de operaciones que necesitan realizar, se programan en flujos de datos y se procesan todas juntas. Básicamente, las GPUs pueden ser usadas como un procesador de propósito específico que está optimizada para trabajar con grandes cantidades de datos y realizar las mismas operaciones, una y otra vez. Por otro lado, como muestra la Fígura 4.8, la arquitectura de CPU está orientada a la ejecución de diferentes tareas, y el énfasis está en el control de esas tareas y su posiblemente intrincado flujo de control. Mientras que en la GPU está orientado a ejecutar una única tarea con diferentes datos. La Figura 4.8 resume enste concepto en forma gráfica.

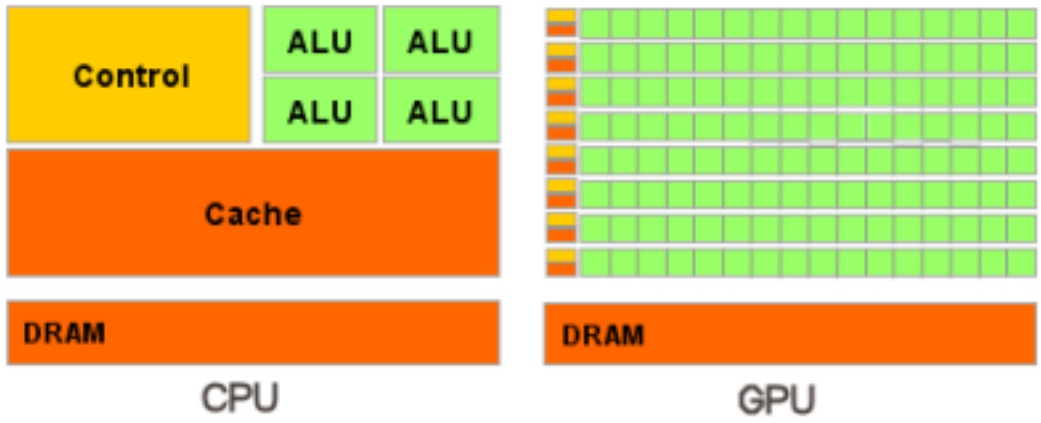

Figura 4.8: Arquitecturas de CPU y de GPU. Extraído de [62].

CUDA (Computer Unified Device Architecture) [53] es una arquitectura de cálculo paralelo desarrollado por NVIDIA que aprovecha la gran potencia de la GPU para ofrecer un incremento extraordinario del rendimiento del sistema. Proporciona un conjunto de bibliotecas y herramientas, que permiten el uso de las GPUs de NVIDIA como coprocesadores para acelerar ciertas partes de un programa, generalmente aquellas que presentan un alto costo computacional y un alto nivel de paralelismo de datos. Con cada nueva versión, CUDA extiende su API con nuevas funcionalidades, y añade soporte para las nuevas características introducidas en las sucesivas generaciones de arquitecturas de GPU, que se identifican por su

[3]En realidad NVIDIA llamó a esta arquitectura SIMT.

revisión de capacidad de cómputo (Compute Capability). Seguidamente se describen los elementos de paralelismo propios de los que dispone CUDA, que posibilitan a la GPU desplegar toda su potencia de ejecución masivamente paralela.

Una GPU, está compuesta de multiprocesadores (SM), formados por núcleos CUDA sobre los que se implementa el modelo SIMT, a los que se les llama (SP), ver Figura 4.9. Los SP son procesadores escalares sencillos que comparten una unidad de control, y una pequeña memoria tan rápida como una caché (memoria compartida). Todos los SMs comparten un mismo espacio de memoria global (compuesto por la DRAM de vídeo), con un acceso de elevada latencia pero con gran ancho de banda. En CUDA, los cálculos son distribuidos a través de una malla (grid) de bloques de hilos o hebras (threads), en donde todos los bloques tienen el mismo tamaño (número de threads) y dimensión. En términos generales, un grid puede ser visto como una representación lógica de la propia GPU, un bloque como un SM, y un thread como cada uno de los distintos SP. A continuación se detalla el significado de los términos manejados en la programación de GPUs con CUDA.

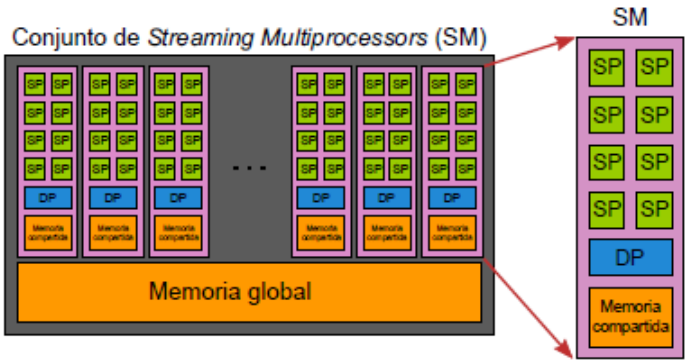

Figura 4.9: Arquitectura básica de una GPU, con 8 procesadores escalares (SP) por SM, una unidad de doble precisión (DP), 16 KB de memoria compartida por SM, y 32 KB (8K entradas de 32 bits) de registros por SM, esta arquitectura abarca hasta la generación Tesla. Extraído de [62].

- Threads: Forman las unidades elementales de ejecución y se deben mapear[4] (asignar) sobre núcleos de procesamiento hardware de forma lógica, a través de las herramientas de programación que proporciona CUDA se accede a diferentes datos según el identificador. Un thread se ejecuta sobre un único SP.

- Warp: Constituye la unidad mínima de ejecución, y está formado por 32 threads consecutivos.

- Bloque: Son agrupaciones de threads que han sido asignados a un SM. Los bloques se ejecutan cuando lo permiten los recursos disponibles, en particular, la cantidad requerida de registros o de memoria compartida, y se planifican mediante warps. Los bloques se pueden definir con 1, 2 ó 3 dimensiones. Los threads que componen un bloque pueden cooperar y sincronizarse a través de la memoria compartida del bloque así como del uso

[4]Esta tarea la realiza en forma automática el driver de CUDA.

de operaciones de sincronización.

- Grid: Es la unidad de trabajo en la que se descompone la ejecución de un kernel, y está formado por una agrupación de bloques con un mismo tamaño y dimensión. Los grid se pueden definir con 1 ó 2 dimensiones, y a partir de la capacidad de cómputo 2.0 hasta con 3 dimensiones. Los bloques de un grid, no pueden sincronizarse entre ellos de forma directa pero todos los threads de un grid se pueden comunicar a través de la memoria global.

- Stream: Se corresponde con una cola de trabajo en la que todos los kernels que lo compone se ejecutan secuencialmente, aunque kernels lanzados en "streams" diferentes pueden ser ejecutados en paralelo. Salvo que se especifique lo contrario, los kernels se lanzan en el stream 0.

- Kernel: Es una función que contiene una porción de código CUDA, que es ejecutada por todos los threads del grid, cada uno de los cuales actúa sobre un espacio diferente de datos que se define a partir de los identificadores de thread y de bloque. Junto a la llamada de un kernel, se debe especificar la dimensión de su grid (dimG = número de bloques y dimB = número de threads por bloque), así como la cantidad de memoria compartida necesaria (parámetro opcional en bytes) y su pertenencia a un determinado stream (por defecto al stream 0). Las variables dimG y dimB son del tipo dim3, cuyo contenido es accesible de la forma: (var.x, var.y, var.z). Estas variables pueden representar 1, 2 ó 3 dimensiones, para definir los tamaños de grid y de bloque mediante números enteros.

4.2.2.1. Jerarquías de memorias en CUDA

CUDA establece, de acuerdo a la arquitectura de la GPU, distintos niveles de memoria, cada uno con un propósito diferente:

- La memoria de registros, que está situada en el propio chip, se corresponde con los elementos de memoria más pequeños dentro de la arquitectura. Su número está limitado por la capacidad de cálculo del modelo de GPU, y dado que los registros se asignan a cada thread, tienen un ciclo de vida igual al del mismo.

- La memoria local, al igual que la de registros, se asigna a cada thread y tienen su mismo ciclo de vida, sin embargo este tipo de memoria se encuentra físicamente en la memoria global del dispositivo, por lo que tiene un costo de acceso elevado. Es manejada por el driver.

- La memoria compartida, que está dentro del chip, es asignada a cada bloque, presentando un ciclo de vida igual al ciclo de un bloque. Siempre y cuando no hayan conflictos entre bancos (consultar la capacidad de cómputo de la GPU [84]), el costo de acceso a memoria compartida para todos los theads de un warp es equivalente al acceso a registro, pero en caso contrario el acceso se serializa. Esta memoria también se utiliza para transferir información entre los threads de un bloque, y así colaborar para realizar

tareas intrabloque en un kernel.

- La memoria global (de dispositivo o GPU), que tiene una latencia mayor por situarse fuera del chip, es accesible por todos los threads de un grid, pero su vida útil está condicionada a las operaciones de reserva y liberación de memoria efectuadas desde el programa ejecutado en el host. El espacio de memoria global se subdivide en tres espacios bien diferenciados:

  - La memoria global propiamente dicha permite accesos tanto en modo escritura como en modo lectura. Su acceso es de alta latencia, pero esta condición ha mejorado con la introducción del manejo de ciertos niveles de caché a partir de la capacidad de cómputo 2.x. El patrón de su acceso determina la latencia asociada. Así, cuando se realiza una petición a memoria global, ésta es servida como mínimo por un warp, aunque el número de transacciones necesarias para atender dicha petición depende de las restricciones de la capacidad de cómputo del dispositivo. Aquellas condiciones que evitan la serialización de estas operaciones se pueden consultar en [84], y aunque eran muy estrictas al principio, en las GPUs más modernas (capacidad de cómputo 2.x o superior) sólo se obliga a que todos los threads de un warp accedan a palabras de tamaño igual o menor a 4 bytes de un mismo segmento de memoria, aún cuando su acceso no sea consecutivo.

  - La memoria de texturas permite realizar escrituras por parte del host y sólo lecturas por parte del dispositivo, con la ventaja de disponer de caché on-chip por SM muy rápida. Este espacio de memoria está optimizado para aprovechar la localidad de datos definidos en 2D, mientras que para otros usos puede ser incluso más lenta que la memoria global del dispositivo.

  - La memoria constante permite realizar escrituras por parte del host y sólo lecturas por parte del dispositivo. Su tamaño está limitado a 64 KB por SM y dispone de una caché muy eficiente. Es ideal para su uso en aquellas situaciones en las que todos los threads de un warp acceden a la misma dirección de memoria; en otro caso, los accesos se serializan. Su uso está indicado, por ejemplo, para almacenar coeficientes utilizados en repetidas ocasiones por una fórmula matemática.

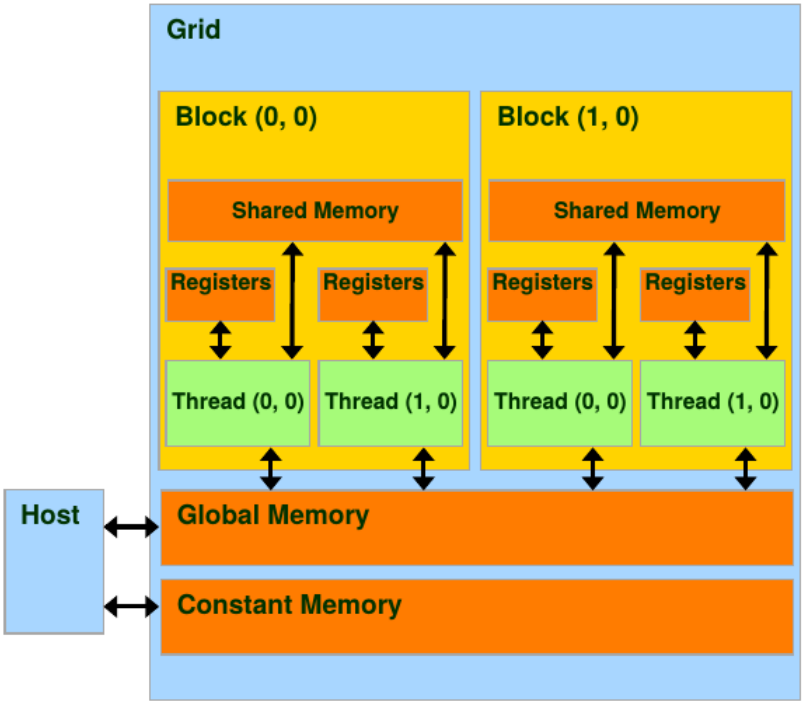

Figura 4.10: Jerarquías de memorias de las GPUs de NVIDIA. Extraído de [62].

Capítulo 5

Evaluación Experimental

En este capítulo se resumen los resultados obtenidos en la evaluación experimental de los métodos presentados en el capítulo anterior.

Primero, se describen las plataformas de hardware empleadas en la evaluación experimental, a continuación se presentan los casos de utilizados y, por último, se resumen los resultados experimentales obtenidas propiante dichos.

5.1. Plataforma de Hardware

Las diferentes pruebas se realizaron sobre un equipo de procesadores heterogéneos. En especial un computador, Salto I, que posee 2 procesadores Intel Xeon(R) CPU E5-2620v2 de 6 cores de 2.1 GHz (arquitectura Sandy-Bridge) y cuenta con 128 GB DDR3 de memoria RAM. El equipo cuenta con una tarjeta NVIDIA K40 GPU (2880 CUDA cores de 745 MHz, arquitectura Kepler, con 12 GB DDR5 de memoria) conectada via el bus PCI-e. El sistema operativo del equipo es Linux Centos v6.5 Linux.

Los códigos fueron compilados utilizando gcc v.4.4.7 y el compilador de CUDA (nvcc) en su versión 6.5. En ambos casos se utilizaron las mismas banderas (flags) de optimización, específicamente $-O3$ activa. Todas las implementaciones fueron evaluadas utilizando aritmética de doble precisión.

En la Tabla 1 se resumen las principales características del equipo.

Cores	2 x Intel Xeon(R) CPU E5-2620v2 de 6 cores de 2.1 GHz
Ram	128 GB DDR3
Sistema Operativo	Linux Centos v6.5 Linux
GPU	NVIDIA K40 GPU
Cores	2880 CUDA cores de 745 MHz
Ram	12 GB DDR5
CUDA	7.5

Tabla 1: Características principales del equipo Salto I.

5.2. Casos de Prueba

Como caso de prueba se plantea la resolución de la Ecuación del Transporte sobre un prisma rectangular, específicamente se consideró un domino con las siguientes dimensiones largo $2L$, ancho L y alto L.

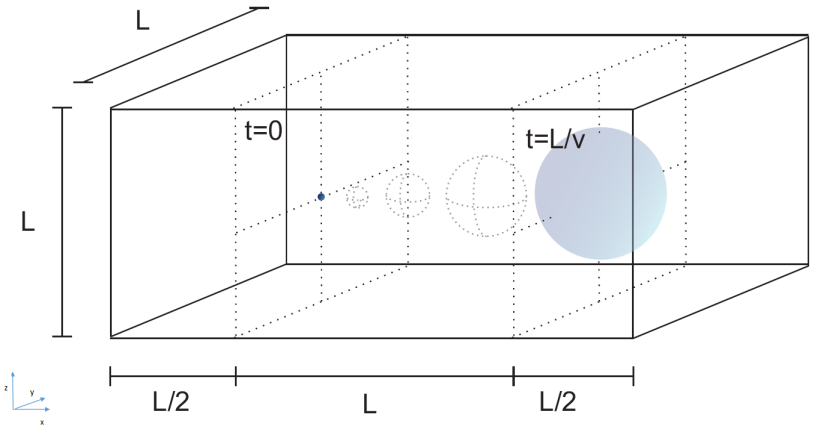

Figura 5.1: Problema advectivo-difusivo empleado como caso de de prueba.

Para la simulación se consideraron fronteras impermeables para el soluto, un tensor de difusión $\mathbf{D}$ isótropo con un valor de difusión D y una velocidad de solvente v, la cual solo tiene un componente no nulo en la dirección x, v_x.

Se plantearon diferentes casos de estudio que se diferencian en la discretización considerada, en particular se emplearon cuatro tamaños de grilla, $66 \times 34 \times 34$, $130 \times 66 \times 66$, $258 \times 130 \times 130$ y $514 \times 258 \times 258$, la información se resume en la Tabla 2.

Caso	Dimensión	Número de Celdas
1	66x34x34	76,296
2	130x64x64	532,480
3	258x130x130	4,360,200
4	514x256x256	33,685,504

Tabla 2: Dimensión y número de celdas de cada caso de prueba.

La simulación comienza con una concentración de valor 0 en todo el dominio excepto en el punto $x = L/2; y = L/2 = 2yz = L/2$ donde la misma tiene un valor de 1, y se desarrolla la solución hasta el tiempo t_f, en que el centro de la pluma viaja una distancia L en el dominio. Lo anteriormente mencionado introduce la siguiente restricción entre las variables del problema:

$$t_f = L/v_x. \tag{5.1}$$

A su vez, para simular el proceso difusivo sin un exceso de "reflexión" por la frontera impermeable para el soluto, se consideró un número de Fourier de 0.05 considerando como magnitudes característica el tiempo final de simulación y la menor distancia entre el punto con concentración inicial no nula y la frontera. Lo anterior introduce otra restricción entre las variables del problema:

$$D * t_f/(L/2)^2 = 0{,}05. \tag{5.2}$$

Los valores de coeficiente de difusión D de velocidad v_x considerados fueron $1E10^{-9}$ m^2/s y de $1E10^{-9}$ m/s respectivamente. Esta combinación de valores, y las restricciones mencionadas, resultan en un valor L de 40 m y un tiempo de simulación t_f de$1E10^{10}$s. Estos parámetros permiten simular los diferentes casos planteados en la Tabla 2, ya que para el Caso 1 se obtiene un número de Peclet de 1.2.

El paso de tiempo de simulación Δt para asegurar la convergencia del método explícito para cada caso, se calculó mediante la siguiente expresión:

$$\Delta t = (\Delta x^2)/D/7; \tag{5.3}$$

Dicha expresión es similar a considerar el número de Fourier en tres dimensiones y surge de calcular el tiempo en que una celda alcanza el equilibrio con las 6 celdas adyacentes, suponiendo que el valor de estas últimas es inicialmente cero y considerando únicamente procesos difusivos.

Se utilizó este caso de estudio porque para dicho caso existe una aproximación analítica, la que permite verificar la precisión numérica de las soluciones calculadas.

Dicha solución se basa en la solución analítica:

$$c(x,t) = \frac{M}{(4\pi t)^{3/2}\sqrt{D_x D_y D_z}} exp^{(-\frac{(X-v_x t)^2}{4D_x t} - \frac{(Y-v_y t)^2}{4D_y t} - \frac{(Z-v_z t)^2}{4D_z t})}. \tag{5.4}$$

La misma supone que en un medio infinito en el instante 0 se libera una masa M en el punto de coordenadas $(0,0,0)$, y considera procesos de transporte advectivos y difusivos no homogéneos. Para aproximar una solución analítica en el dominio finito aplicamos el principio de superposición (teoría de las imágenes), en el cual usamos 7 plumas como se muestra en la Figura 5.2.

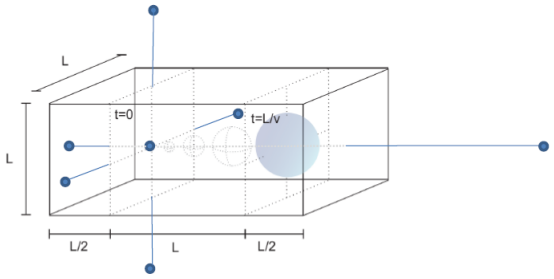

Figura 5.2: Se aplican 7 plumas sobre el volumen de control.

Se puede representar la solución analitica para cada pluma mediante la Ecuación5.5.

$$c_i(x,t) = \frac{M}{(4\pi t)^{3/2}\sqrt{D_x D_y D_z}} exp^{(-\frac{(X-X_i-v_x t)^2}{4D_x t} - \frac{(Y-Y_i-v_y t)^2}{4D_y t} - \frac{(Z-Z_i-v_z t)^2}{4D_z t})}. \tag{5.5}$$

Finalmente la solución analítica utilizada se puede expresar como muestra la Ecuación 5.6.

$$\sum_{i=1}^{7}(c_i(x,t)). \tag{5.6}$$

5.3. Resultados Experimentales

En todos los experimentos que implican el uso de la GPU, los tiempos de transferencia siempre se incluyen. Además, los resultados presentados en todos los casos son el promedio de 10 ejecuciones independientes.

5.3.1. Comparación de solvers lineales

Como se comentó anteriormente, la utilización de esquemas implícitos conlleva la resolución de un sistema de ecuaciones lineales en cada paso de tiempo. Para la resolución de dicho sistema se evaluaron dos solver iterativos para resolver sistemas lineales (dispersos). Por un lado, el SIP un solver específico para trabajar con el sistema de ecuaciones que se generan cuando se aborda la Ecuación de Transporte, el otro solver evaluado es BiCGStab que es un solver ampliamente usado y que trabaja con sistema de ecuaciones lineales en general. Entonces, el primer experimento se centró en la comparación de los desempeños de ambos solvers como parte del esquema Implícito. Específicamente, se compararon únicamente las implementaciones del esquema Implícito que sacan partido del poder de cómputo de la tarjeta gráfica.

Las pruebas se realizaron utilizando los mismos parámetros para cada problema. La Tabla 3 muestra los resultados obtenidos en cuanto a desempeño así como la desviación estándar (STD) de los mismos.

	Caso	CPU Tiempo (STD)	OpenMP Tiempo (STD)	GPU Tiempo (STD)	Nro. de Celdas
BiCGStab	1	1.548e0(1.5e-1)	4.15e-1(1.7e-1)	1.49e0(1.9e-1)	76,296
	2	3.31e1(2.1e0)	7.44e0(1.3e0)	1.85e1(1.2e0)	532,480
	3	9.73e2(1.1e2)	1.68e2(1.5e1)	7.78e2(4.2e1)	4,360,200
	4	6.60e4(1.5e3)	6.38e3(1.7e3)	1.53e4(1.0e3)	33,685,504
SIP	1	8.08e-1(6.4e-2)	2.33e-1(5.1e-3)	1.65e0 (6.9e-2)	76,296
	2	1.51e1(1.0e-1)	3.90e0(1.6e-1)	1.22e1 (1.2e-1)	532,480
	3	4.73e2(5.8e-1)	8.30e1(4.5e-1)	2.50e2 (4.2e-1)	4,360,200
	4	2.58e4(6.4e0)	2.70e3(2.8e0)	6.90e3 (7.8e1)	33,685,504

Tabla 3: Tiempos de ejecución (en segundos) de las implementaciones de los algortimos BiCGStab y SIP.

Los resultados muestran que para este problema el desempeño del SIP mejora en un orden de magnitud los tiempos de ejecución del BiCGStab. Si bien las operaciones que contiene el BiCGStab no presentan limitantes para la paralelización como sí la tienen las operaciones del SIP, el primero es penalizado respecto a este último debido a que implica un número mayor de operaciones. En cuanto la precsión numérica de los resultados los dos solvers convergieron al mismo error. Este estudio definió el uso del SIP como solver a utilizar en este trabajo.

5.3.2. Evaluación del uso de CPU

El siguiente estudio se centra en la optimización de los métodos en CPU, o sea la versión paralela de todos los esquemas paralizados con OpenMP. En otras palabras, se evalúa el rendimiento y la escalabilidad de los tres métodos desarrollados (explícito, implícito y predictor-corrector) en CPU. La Tabla 4 muestra los tiempos de ejecución de las siguientes implementaciones EXPL$_{cpu}$(implementación secuencial del esquema Explícito sobre CPU), IMPL$_{cpu}$ (implementación secuencial del esquema Implícito sobre CPU) y PR-CO$_{cpu}$(implementación secuencial del esquema Predictor Corrector sobre CPU) y usando 4, 6, 8 y 12 hilos (máximo número de cores de la plataforma de hardware utilizada) para todos los casos. Además, se incluye la comparación con las versiones de los métodos ejecutando en CPU pero en sus variantes secuenciales.

	Caso	**CPU** Tiempo (STD)	**4 th.** Tiempo (STD)	**6 th.** Tiempo (STD)	**8 th.** Tiempo (STD)	**12 th.** Tiempo (STD)	Nro. de Celdas
Explícito	1	1.77e-1(2.7e-3)	6.92e–2(2.7e-4)	5.42e-2(4.5e-4)	4.14e-2(4.2e-4)	3.58e-2(6.7e-4)	76,296
	2	2.71e0(3.1e-2)	1.06e0(4.2e-2)	7.95e-1(2.5e-2)	6.64e-1(1.3e-2)	5.68e-1(2.9e-2)	532,480
	3	8.01e1(1.7e-1)	2.67e1(1.5e-1)	1.91e1(2.9e-2)	1.41e1(1.1e-1)	9.79e0(3.3e-2)	4,360,200
	4	2.49e3(2.8e0)	8.29e2(2.7e0)	5.69e2(1.0e0)	4.30e2(3.0e0)	2.93e2(2.4e-1)	33,685,504
Implícito	1	8.08e-1(6.4e-2)	3.14e-1(7.4e-3)	2.72e-1(5.5e-4)	2.53e-1(2.4e-3)	2.31e-1(5.1e-3)	76,296
	2	1.51e1(1.0e-1)	6.57e0(1.4e-1)	5.12e0(1.9e-2)	4.43e0(1.3e-1)	3.92e0(1.6e-1)	532,480
	3	4.73e2(5.8e-1)	1.87e2(4.5e-1)	1.34e2(8.9e-1)	1.08e2(5.5e-1)	8.20e1(4.5e-1)	4,360,200
	4	2.58e4(6.4e0)	7.32e3(3.5e0)	4.96e3(2.8e0)	3.81e3(8.5e0)	2.70e3(2.8e0)	33,685,504
Pre-Cor	1	2.40e-1(5.5e-3)	1.35e-1(6.2e-3)	1.00e-1(5.5e-4)	7.44e-2 (5.5e-4)	6.01e-2(8.4e-5)	76,296
	2	6.12e0(8.9e-3)	2.25e0(7.9e-2)	1.73e0(2.4e-2)	1.37e0(5.6e-2)	1.13e0(4.7e-2)	532,480
	3	1.95e2(7.1e-1)	5.91e1(3.1e-1)	4.20e1(1.3e-1)	3.11e01(4.8e-2)	2.22e1(1.4e-1)	4,360,200
	4	9.86e3(1.7e0)	2.43e3(4.3e0)	1.72e3(5.5e0)	1.32e3(6.4e0)	9.54e2(4.5e-1)	33,685,504

Tabla 4: Tiempos de ejecución (en segundos) de las diferentes implementaciones para los diferentes esquemas sobre CPU.

Teniendo en cuenta los tiempos de ejecución que se resumen en la Tabla 4, se puede destacar en primer lugar que el esquema Implícito es la opción que más tiempo lleva, con una diferencia de aproximadamente un orden de magnitud con el esquema explícito, que es el método más económico (desde una perspectiva de costo computacional). Además, las diferencias en tiempo de ejecucción entre ambos enfoques parecen crecer con la dimensión de los problemas.

Considerando la aplicación de las técnicas de paralelismo en la CPU, en todos los casos el uso de paralelismo ofrece beneficios similares. Además, las mejoras crecen con la dimensión de los casos de prueba tratados, lo que muestra que las implementaciones alcanzan escalabilidad débil[1]. Cabe señalar que para el Caso 1 la aceleración alcanzada con 12 hilos es de casi 5 veces, mientras que para el Caso 4 los valores de aceleración crecen hasta el entorno de 10 veces. En cuanto a la escalabilidad fuerte[2] los resultados son menos sólidos. Notar que la aceleración

[1]En este tipo de escalabilidad se aumenta el número de unidades de procesamiento, manteniendo el tamaño del problema constante para cada unidad, consiguiendo un tiempo de cómputos constante para cada procesador.

[2]En este otro modelo de escalabilidad, se mantiene el tamaño del problema constante, el objetivo es dis-

conseguida para un mismo caso no aumenta en forma lineal al aumentar la cantidad de cores, en especial en los caso chicos.

5.3.3. Evaluación del uso de GPU

En tercera instancia, se evaluaron los beneficios que ofrece el uso de una plataforma masivamente paralela. La Tabla 5 presenta los tiempos de ejecución necesarios para resolver la Ecuación de Transporte para todos los casos de prueba y para los tres esquemas evaluados. En concreto, para cada esquema se detallan los tiempos de ejecución de la CPU secuencial, el mejor tiempo obtenido para CPU en sus variantes paralelas y el de las variantes que explotan el poder de cómputos de las GPUs.

		Caso 1 Tiempo(STD)	Caso 2 Tiempo(STD)	Caso 3 Tiempo(STD)	Caso 4 Tiempo(STD)
Explícito	CPU	1.75e-1(5,4e-3)	2.70e0(8.9e-3)	7.90e1(7.1e-1)	2.48e3(1.7e0)
	OpenMP	3.50e-2(8.4e-5)	5.02e-1(4.7e-2)	9.75e0(1.4e-1)	2.92e2(4.5e-1)
	CUDA	1.00e-2(3.3e-3)	8.00e-2(4.3e-3)	1.02e0(2.2e-1)	2.11e1(1.4e0)
Implícito	CPU	7.10e-1(6.4e-2)	1.49e1(1.0e-1)	4.72e3(5.8e-1)	2.58e4(6.4e0)
	OpenMP	2.33e-1(5.1e-3)	3.90e0(1.6e-1)	8.30e1(4.5e-1)	2.70e3(2.8e0)
	CUDA	1.65e0(6.9e-2)	1.22e1(1.2e-1)	2.50e2(4.2e-1)	6.90e3(7.8e1)
Pre-Cor	CPU	2.30e-1(2.7e-3)	6.11e0(3.1e-2)	1.94e2(1.6e-1)	9.86e3(2.8e0)
	OpenMP	6.00e-2(6.7e-4)	1.09e0(2.8e-2)	2.20e1(3.3e-2)	9.54e2(2.4e-1)
	CUDA	2.70e-1(3.5e-3)	1.21e0(2.7e-3)	1.65e1(2.7e-3)	4.04e2(1.6e-1)

Tabla 5: Tiempos de ejecucción (en segundos) de las mejores implementaciones de cada variante.

Teniendo en cuenta los resultados resumidos en la Tabla 5, se puede concluir que la GPU no ofrece mejoras significativas (en comparación con la versión secuencial) cuando la dimensión de los casos tratados no son importantes. Este resultado está alineado con la literatura del área, ya que las GPUs necesitan importantes volúmenes de datos para cubrir los overheads incurridos por ejemplo en traspaso de datos. Para los casos grandes, la versión basada en GPU alcanza un rendimiento similar al de la contraparte paralela sobre CPU, con la mejor configuración de números de hilos, en el esquema Implícito y en el Predictivo-Correctivo. Sin embargo, para el esquema Explícito la versión basada en GPU ofrece un importante aumento de velocidad incluso cuando se compara con la mejor versión sobre CPU paralela. Notar que en el Caso 4 cuando se trabaja con el esquema Explícito, la versión sobre GPU (EXPL_{gpu}) tiene un aceleración de más de 10x comparado con la versión paralela sobre CPU utilizando 12 hilos EXPL_{cpu}. Estos resultados son de esperar, por el potencial que cuentan este tipo de plataformas masivamentes paralelas, las cuales están diseñadas para explotar el paralelismo de datos, por lo cual como se dijo anteriormente es obligatorio contar con un importante volumen de datos para obtener los mejores rendimientos. Además, las

minuir el tiempo de cómputos total directamente proporcional al aumento del número de unidades de procesamiento utilizado.

GPUs por lo general se benefician de dependencias de datos simples, y este es el caso del esquema Explícito.

5.3.4. Evaluación de la precisión numérica de los métodos

Por otra parte, los resultados de errores residuales, calculados en norma 2 de las diferencias entre las soluciones numéricas y la solución analítica, varían de un esquema a otro. En concreto, la Tabla 6 resume el error residual alcanzado en todos los casos de prueba por cada uno de los tres esquemas.

Cabe destacar que, a pesar de las inevitables diferencias en los resultados explicadas por el uso de la aritmética de punto flotante, todas las variantes de cada esquema presentan resultados comparables.

Caso (**Celdas**)	**1** (76,296)	**2** (532,480)	**3** (4,360,200)	**4** (33,685,504)
Explícito	3.57e-8	2.61e-9	2.72e-10	2.01e-11
Implícito	1.06e-8	9.50e-10	8.60e-11	7.72e-12
Pre-Cor	1.13e-8	1.30e-9	2.31e-10	6.36e-11

Tabla 6: Error para cada método y para cada caso de prueba comparado con la solución analítica.

Los resultados obtenidos confirman que si bien el esquema Explícito tiene mejor desempeño computacional que el esquema Implícito y el Predictor - Corrector (ver 5.3.3), pero los resultados del esquema Implícito ofrecen una mejor precisión numérica.

En otra línea, y en base a la aplicación de paralelismo, el rendimiento paralelo de la CPU para los tres métodos ofrece beneficios similares. Sin embargo, el uso de una GPU para los esquemas explícitos presenta resultados mucho mejores que su uso para los otros esquemas, especialmente para la variante implícita (notese que en el primer caso, el aumento de velocidad está cerca de un 10x, mientras que en el caso implícito los tiempos de ejecución son similares a los de la CPU en paralelo). Este resultado es importante porque, con estos valores de aceleración, las limitantes de precisión de los esquemas explícitos se pueden compensar mediante el uso de una grilla más fina para los cálculos. Cabe señalar que, la precisión para el Caso 2, cuando se resuelve mediante el esquema Implícito es comparable a la precisión en la Caso 3 utilizando el esquema explícito. Por otro lado, la versión basada en la GPU del esquema explícito para el Caso 3 supera la mejor implementación del esquema Implícito sobre el Caso 2.

5.3.5. Estudio del efecto de Δt

Los experimentos anteriores fueron realizados todos con el mismo Δt, en cambio este apartado está focalizado en el estudio de los efectos que causa la utilización de diferentes Δt sobre la precisión de cada esquema. En forma intuitiva, un Δt pequeño ofrece mejor precisión pero, implica mayor consumo de recursos computacionales y, por el contrario, un Δt grande invierte la relación.

En el caso del esquema explícito el Δt es acotado por el Número de Fourier (ver Ecuación 2.38), en este caso solo se puede trabajar con Δt pequeños. Para el esquema Implícito no existen estas limitantes, y por lo tanto se puede trabajar con valores de Δt mayores. En particular, se trabajó para el caso del esquema Explícito con Δt, $(\Delta t)/2$, $(\Delta t)/4$ y en el caso Implícito se utilizó Δt, $(\Delta t) \times 2$, $(\Delta t) \times 4$. En la Tabla 7 y Tabla 8 se resume esta información para el caso Explícito e Implícito respectivamente y, además, se incluyen los tiempos de ejecución de las variantes entre paréntesis.

Casos	$\Delta t/4$ Precisión(Tiempo)	$\Delta t/2$ Precisión(Tiempo)	Δt Precisión(Tiempo)
1	1.08E-08(5.0E-2)	1.62E-08(3.0E-2)	3.57E-08(1.0E-2)
2	9.64E-10(3.2E-1)	1.37E-10(1.6E-1)	2.61E-09(8.0E-2)
3	8.69E-11(4.0E+0)	1.22E-10(2.0E+0)	2.27E-10(1.0E+0)
4	7.83E-12(8.4E+1)	1,09E-11(4.2E+1)	2.10E-11(2.1E+1)

Tabla 7: Precisión númerica y tiempo de ejecución (en segundos) del esquema Explícito con diferentes Δt.

Todos los resultados, referentes a tiempo de ejecucción, para el estudio del efecto del cambio en Δt son sobre las variantes que utilizan la GPU.

En la Tabla 7 se puede observar que en el esquema Explícito para los casos de pruebas con menor discretización al disminuir el tamaño del Δt decrece el error, pero aumenta el tiempo de ejecución en forma lineal (como es lógico). Es más, para los casos de pruebas con mayor discretización la disminución del Δt producen una precisión comparable al esquema Implícito.

Para el esquema Implícito, resumido en la Tabla 8, se puede observar que el comportamiento es similar a la variante explícita, y en los casos de mayor dicretización este esquema

Casos	$\Delta t \times 4$ Precisión(Tiempo)	$\Delta t \times 2$ Precisión(Tiempo)	Δt Precisión(Tiempo)
1	2.09E-08(9.8E-1)	1.25E-08(1.1E+0)	1.06E-08(1.6E+0)
2	1.18E-09(4.9E+0)	9.96E-10(6.9E+0)	9.50E-10(1.3E+1)
3	9.08E-11(7.2E+1)	8.72E-11(1.3E+2)	8.60E-11(2.5E+2)
4	7.83E-12(1.8E+3)	7.75E-12(3.5E+3)	7.72E-12(7.0E+3)

Tabla 8: Precisión numérica y tiempo de ejecución (en segundos) del esquema Implícito con diferentes Δt.

Casos	$\Delta t \times 4$ Precisión(Tiempo)	$\Delta t \times 2$ Precisión(Tiempo)	Δt Precisión(Tiempo)
1	1.06E-08 (6.60E-1)	1.06E-08(3.50E+0)	1.06E-8(1.43E+0)
2	9.50E-10(3.4E+0)	9.50E-10(5.5E+0)	9.50E-10(1.1E+1)
3	4.02E-10(8.4E+1)	1.62E-10(1.4E+2)	8.62E-11(2.5E+2)
4	3.21E-10(1.8E+3)	1.17E-10(2.7E+3)	6.36E-11(5.1E+3)

Tabla 9: Precisión númerica y tiempo de ejecución (en segundos) del esquema Predictor Corrector con diferentes Δt.

se beneficia del uso de Δt mayores, ya que prácticamente mantiene el error y logra disminuir considerablemente los tiempos de ejecución.

El tercer experimento de este apartado compara la mejor configuración de Δt en cuanto al desempeño computacional de cada esquema, para el esquema Explícito se utilizó $\Delta t/4$ y en el caso del esquema Implícito $\Delta t \times 4$.

Caso		Nro. de Celdas	**Precisión Numérica**	**Tiempo(s)**
Explícito	1	76,296	1,08E-08	5,00E-02
	2	532,480	9.64E-10	3.2E-01
	3	4,360,200	8.69E-11	4.0E+00
	4	33,685,504	7,83E-12	2,4E+01
Implícito	1	76,296	2,09E-08	9,84E-01
	2	532,480	1,18E-09	4,9E+00
	3	4,360,200	9.08E-11	7,2E+01
	4	33,685,504	7,83E-12	1,8E+03

Tabla 10: Comparación del esquema Explícito utilizando como paso temporal $\Delta t/4$ y el esquema Implícito empleando $\Delta t \times 4$.

Considerando los resultados de la Tabla 10 se puede observar que el uso de los métodos Explícitos a medida que se disminuye el Δt se acercan en precisión a los resultados de los Implícitos y con tiempos de cómputos menor. Se puede decir desde este experimento que las variaciones del Δt tuvieron mayores beneficios para los métodos Explícitos que para los Implícitos, y además se puede agregar que trabajar con una discretización más fina tiene mayor impacto que la solo modificación del Δt.

Esta situación permite reafirmar que las restricciones de precisión de los métodos explícitos se pueden mitigar trabajando con discretizaciones más finas y que los sobre-costos implicados pueden ser abatidos mediante el uso de manera eficiente de arquitecturas masivamente paralelas.

El último experimento relativo al cambio del Δt es realizado sobre el esquema Predictor-Corrector, los resultados se pueden observar en la Tabla 9. En este esquema los cambios del Δt, mejoran el desempeño para un $\Delta \times 2$ proporcionalmente al decremento de la precisión,

y con un $\Delta \times 4$ ya no se mantiene la proporción entre la mejora de desempeño y la perdida de precisión, sino que el tiempo de cómputos no disminuye en la magnitud esperada. Esto es debido a que el paso predictivo del Predictivo-Corrector con un Δt grande produce una primera solución no muy buena y que el paso correctivo no logra refinarla. Los resultados de este esquema si se los compara con los demás esquemas no son buenos del punto de vista de precisión ni de desempeño.

5.3.6. Estudio multi-criterio

Como se vio a los apartados anteriores, para realizar una comparación justa entre los métodos/parámetros es necesario hacer una análisis multi-criterio. En este sentido, como último experimento se evaluaron cada uno de los esquemas con una baterías de pruebas, cada una de éstas varían en el Δt para el caso de los esquemas Explícitos y en los casos de los esquemas Predictor-Corrector e Implícito combinan diferentes Δt con diversos coeficientes de convergencia, lo cual permite hacer una amplia exploración del espacio de resultados. Aquí se pueden definir dos funciones objetivos a optimizar, por un lado minimizar el tiempo de cómputos y por el otro maximizar la precisión numérica (minimizar el error). Esta batería de pruebas generan un gran número de soluciones que son comparada según el criterio de Pareto [74]. Entonces se puede decir que la solución $S_1(x_1, x_1)$ domina a la solución $S_2(x_2, y_2)$, si se cumple algunas de las siguientes condiciones: $x_1 < x_2$ y $y_1 \leq y_2$, o $x_1 \leq x_2$ y $y_1 < y_2$. Ahora se puede definir el Frente de Pareto como el conjunto que forman las soluciones no dominadas por ninguna otra solución. Finalmente, en este estudio se confrontan los desempeños computacionales alcanzados con respecto a las precisiones numéricas obtenidas como resultado de esta batería de pruebas y se construyen los diferentes Frentes de Pareto [74] conformados.

Primero se evalúan las implementaciones del esquema Explícito sobre las diferentes plataformas (EXPL$_{cpu}$, SCEXPL$_{cpu}$, EXPL$_{gpu}$). En estas evaluaciones se trabaja sobre el Δt, como es un esquema numéricamente inestable se comienza con el máximo Δt y luego se lo va a dividiendo por 2, 3 y así sucesivamente hasta el valor 10. Naturalmente, con Δt más pequeños se obtienen desempeños más pobre pero se alcanzan mejores valores de precisión numérica.

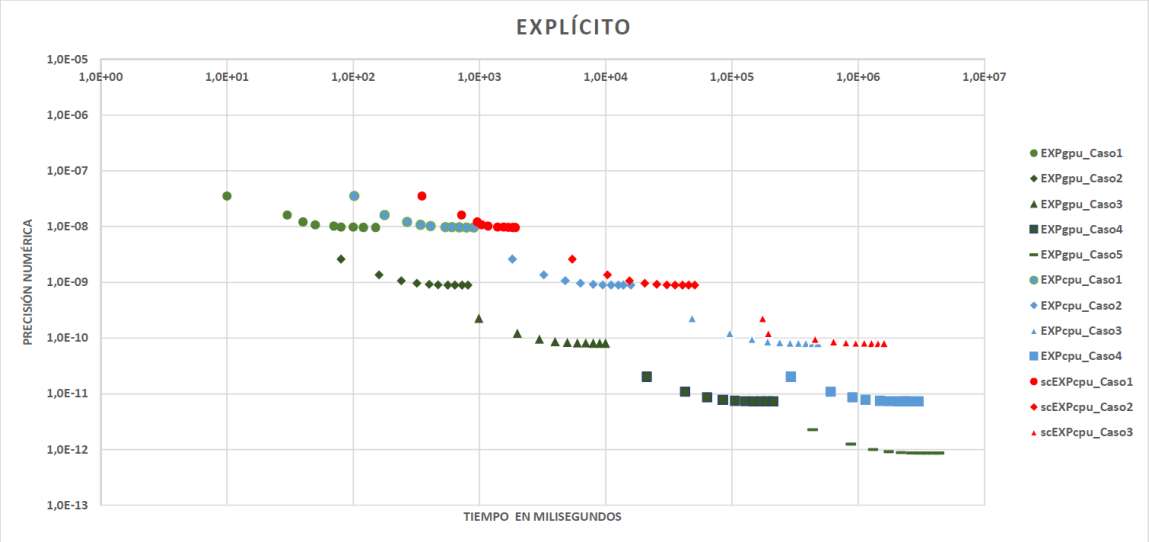

Figura 5.3: Frentes de Pareto para el esquema Explícito.

Para este estudio se utilizó una extensión de los casos de prueba. En especial se definió un nuevo caso, el Caso 5, que no fue mencionado anteriormente porque es solo alcanzado por el esquema Explícito. Los demás esquemas no llegan a estas dimensiones por limitantes de memoria. Este caso cuenta con 223.495.688 celdas, y llega a una precisión numérica por debajo de 1.00E-12.

La Figura 5.3, donde se compara tiempo insumido y la precisión alcanzada muestra como escala este esquema para las tres plataformas, mejorando en un orden aproximadamente el error al ir creciendo en las dimensiones del caso, pero disminuyendo el desempeño en también un orden. En todos los casos la implementación EXPL_{gpu}, mejora un orden el desempeño de la implementación EXPL_{cpu}, y esta última mejora en un orden la implementación SCEXPL_{cpu}.

En cuanto a la precisión numérica la implementación EXPL_{gpu}, recordar que la variante explícita permite trabajar con una discretización más fina, alcanza razonablemente los mejores valores.

Luego para el esquema Implícito se hacen diferentes pruebas combinando diversos Δt y coeficientes de convergencia del SIP. en este caso el esquema es numericamente estable y por lo tanto se pueden utilizar Δt de mayor tamaño. Los resultados se pueden apreciar en la Figura 5.4.

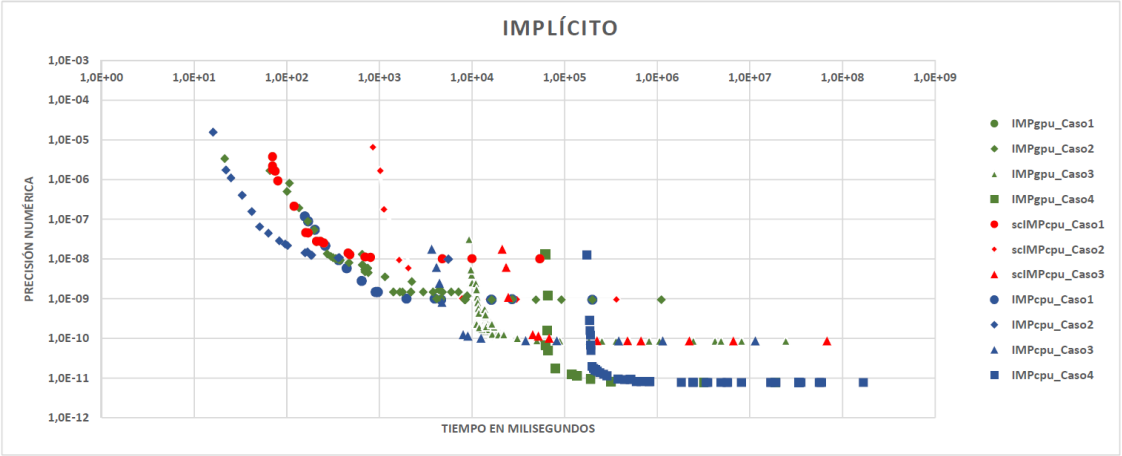

Figura 5.4: Frentes de Pareto para el esquema Implícito.

Los resultados obtenidos con este esquema (5.4) son similar al caso explícito, pero con un comportamiento más uniforme en los casos que presentan dimensiones más grandes. Como se puede apreciar la implementación EXPL$_{cpu}$ tiene mejor desempeño que la implementación EXPL$_{gpu}$, parte de este desempeño se explica porque el método SIP en el proceso de paralelización comienza y termina trabajando (hiperplanos iniciales y finales) con pocos puntos al mismo tiempo.

Finalmente, se estudió el esquema Predictivo-Corrector, nuevamente se evaluó una batería de pruebas con diferentes coeficientes de convergencia, como en el esquema Implícito se trabaja con diferentes Δt. En la Figura 5.5 están plasmados los resultados para el esquema Predictor-Correcto. Este esquema muestra un escalado uniforme para los valores de error (precisión numérica) y en cuanto al desempeño la diferencia con respecto al esquema Explícito crece proporcionalmente con la dimension del problema.

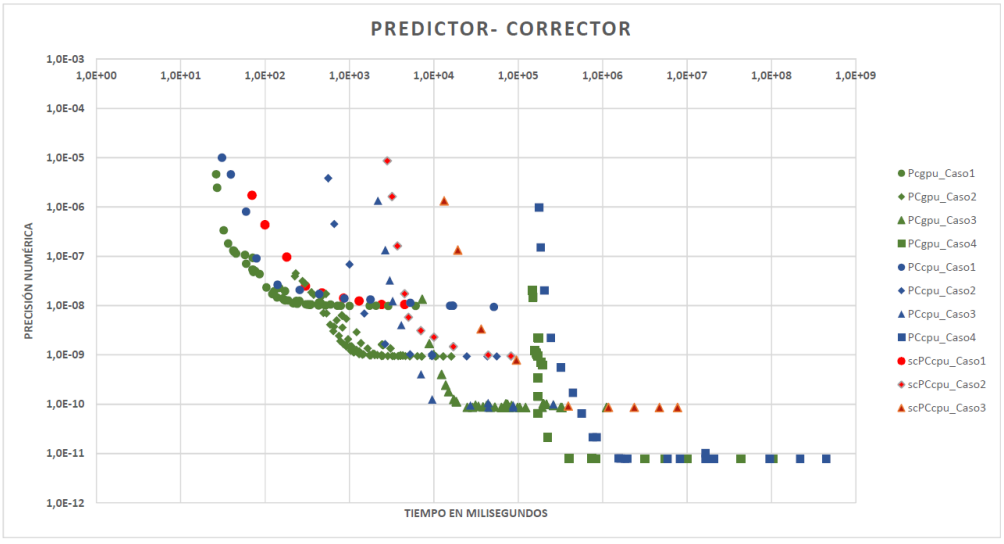

Figura 5.5: Frentes de Pareto para el esquema Predictor-Corrector.

La Figura 5.6 muestra los frentes de Pareto para las implementaciones sobre GPU. Estos frentes fueron construido con la unión de los datos de todos los casos de prueba de dichas implementaciones. Las gráficas fácilmente permiten apreciar que tanto para el esquema Implícito como para el Predictor-Corrector presentan un comportamiento similar. En el caso del esquema Explícito presenta un desempeño muy superior a los anteriores, y se puede observar que en algunos casos obtiene una diferencia de un orden de magnitud en el desempeño con respecto a los demás esquemas y manteniendo la precision numérica, y también se identifican caso donde a igual desempeño la diferencia en cuanto a la precisión numérica es de un orden de magnitud a favor del esquema explícito.

Parte de estos resultados habían sido observados en pruebas anteriores, pero este estudio hace un análisis más amplio del espacio de soluciones y, por lo tanto, permite sacar conclusiones más contundentes.

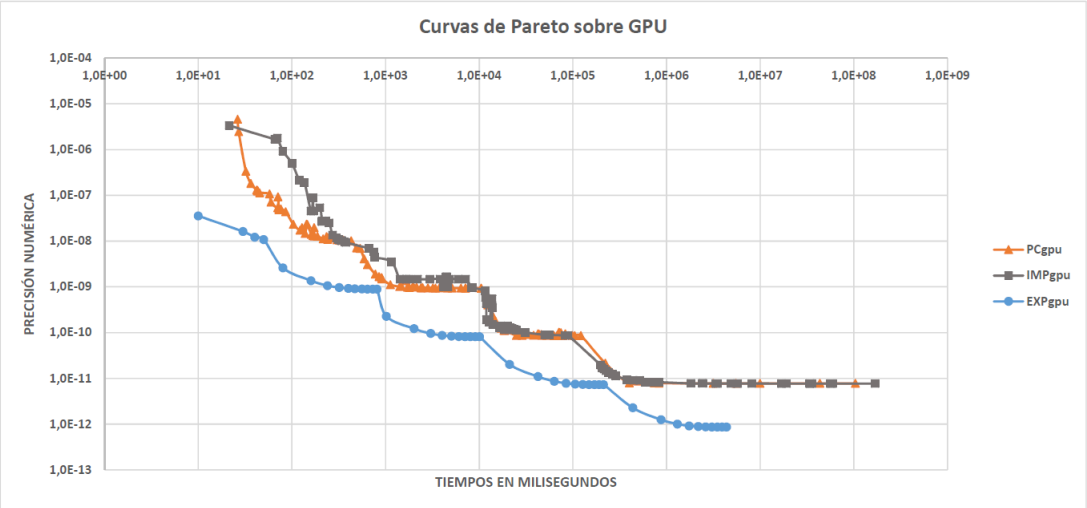

Figura 5.6: Frentes de Pareto de las implementaciones sobre GPU.

En las siguientes Figuras (5.7 y 5.8) expresan los resultados para las implementaciones sobre OpenMP y secuencial sobre CPU respectivamente. Los resultados permite ver un comportamiento similar para los dos casos, en los cuales se observa un mejor desempeño del esquema Implícito.

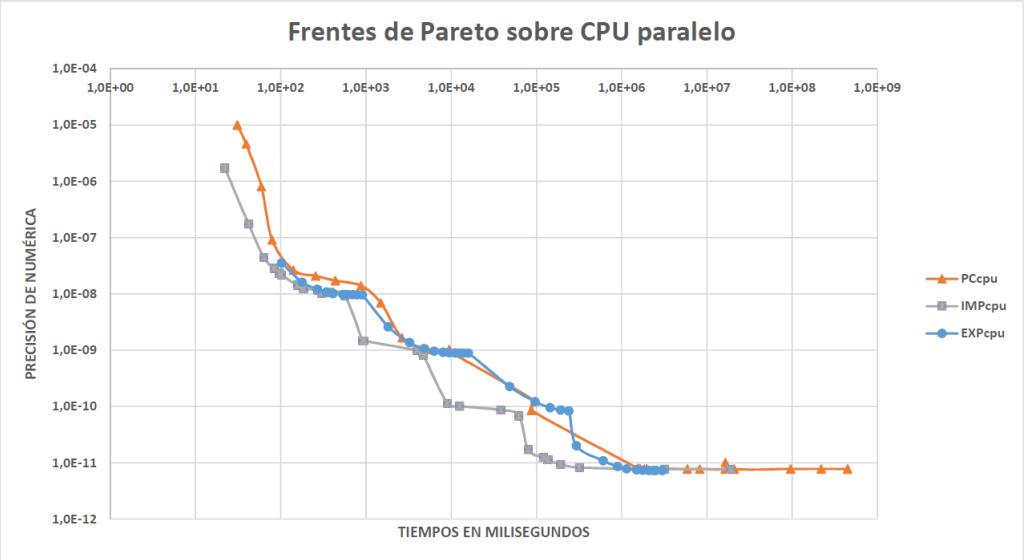

Figura 5.7: Frentes de Pareto de las implementaciones sobre OpenMP.

La Figura 5.7 muestra que la implementación $Impl_{cpu}$ tiene un desempeño superior que Expl_{cpu}, solamente cuando la discretización es gruesa (parte derecha de la gráfica) se igualan los desempeños pero, luego las diferencias de desempeño crecen a favor de la implementación $Impl_{cpu}$ a medida que la discretización utilizada se hace más fina (parte izquierda de la gráfica). La implementación Pre-Cor_{cpu} intermedio entre los dos tal cual paso sobre las implementaciones sobre GPU.

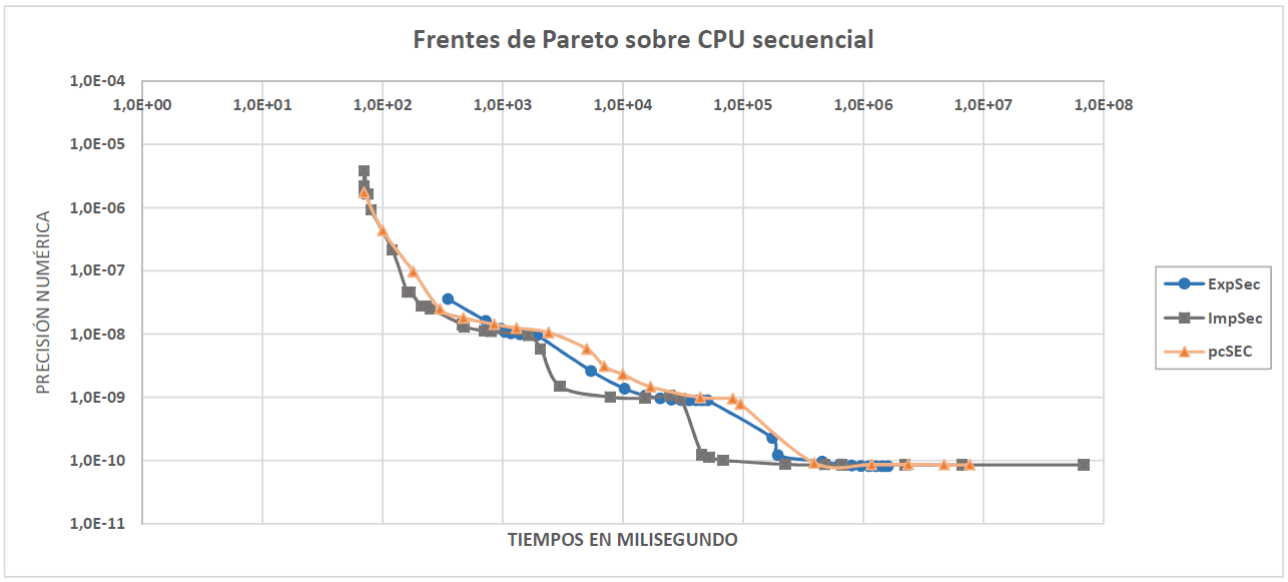

Figura 5.8: Frentes de Pareto de las implementaciones sobre CPU.

La Figura 5.8, exhibe las tres implementaciones secuenciales, donde el comportamiento es similar a las implementaciones paralelas sobre CPU, dentro de estas implementaciones se

destaca la del método implícito.

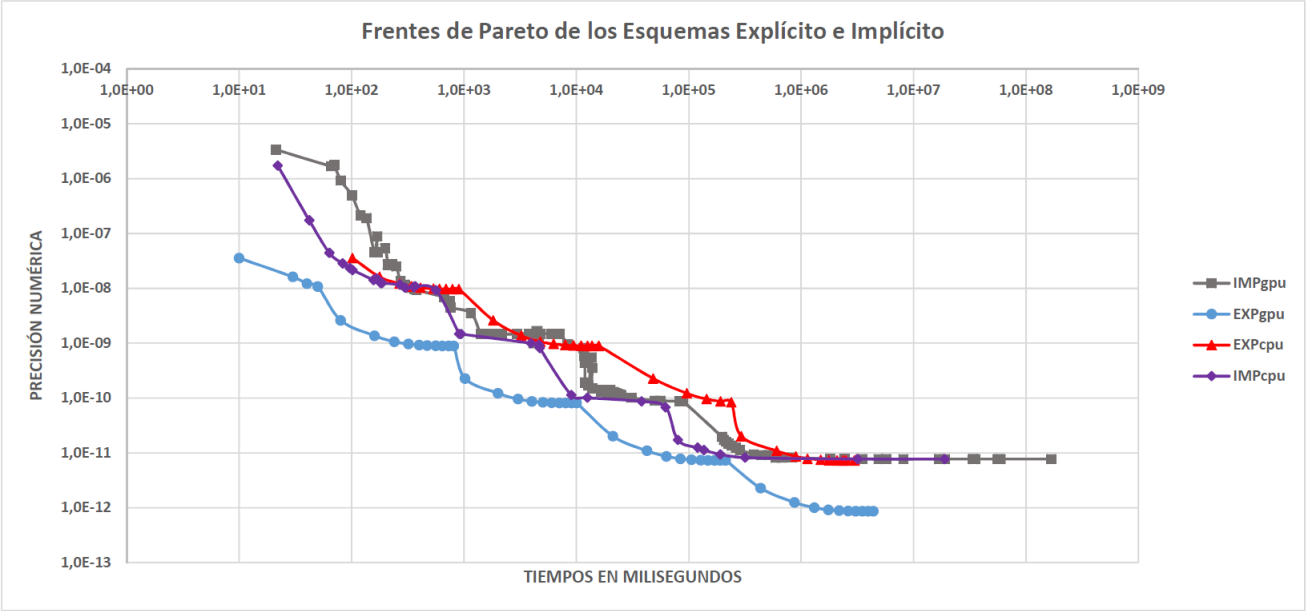

Figura 5.9: Frentes de Pareto para el esquema Explícito e Implícito de las implementaciones paralelas.

Finalmente en la Figura 5.9 se comparan los Frentes de Paretos de las implementaciones paralelas de los esquemas Explícito e Implícito, donde claramente queda expuesto que la implementación del esquema Explícito sobre GPU (EXPL$_{gpu}$) alcanza una precisión que mejora en un orden de magnitud la mejor implementación del esquema Implícito y con un tiempo de cómputos del mismo orden. En cambio cuando se trabaja sobre CPU la implementación IMPL$_{cpu}$ es superior a su propia implementación sobre GPU (IMPL$_{gpu}$) e incluso sobre la implementación EXPL$_{cpu}$. Como conclusión de esta última comparación se puede decir que si se trabaja sobre una plataforma que incluye GPUs es mejor abordar el problema desde un esquema Explícito, por otro lado, si se trabaja utilizando el CPU lo adecuado es emplear un esquema Implícito.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

Este trabajo presenta un estudio sobre la aceleración de la resolución de las ecuaciones diferenciales en derivadas parciales sobre arquitecturas masivamente paralelas. En concreto, se utilizó la ecuación del transporte (advectiva - difusiva) en 3D como caso de estudio para comparar el uso de diferentes esquemas numéricos de resolución, es decir Explícito, Implícito y Predictor - Corrector. La comparación se realizó tanto desde el punto de vista del desempeño computacional como de la precisión numérica alcanzada. Cada esquema se implementó en un código secuencial (en C) y en dos códigos paralelos (C junto con OpenMP y C junto con CUDA).

Sobre la base de la discretización de la ecuación del transporte usando el paradigma de diferencias finitas para el esquema Explícito se desarrollaron diferentes variantes. Una implementación secuencial sobre CPU (scExpl$_{cpu}$), esta implementación tiene como objetivo permitir medir la aceleración de las implementaciones paralelas. Una implementación paralela sobre CPU (Expl$_{cpu}$) utilizando la interfaz de programación de aplicaciones paralelas OpenMP. Esta implementación tiene como objetivo evaluar las bondades de las plataformas Multi-Core y comparar los beneficios con las implementaciones Many-Core. La última implementación en esta variante es para procesadores Many-core (Expl$_{gpu}$). Se intentó maximizar la paralelización y obtener buenos niveles de escalabilidad en las dimensiones del problema, además la estrategia utilizada minimiza las lecturas en memoria y hace un uso adecuado de la memoria compartida.

Por otro lado, se realizaron diferentes variantes para el esquema Implícito, utilizando el método de Crank-Nicolson. El principal problema a resolver en esta implementación fue la resolución de un sistema lineal en cada paso de tiempo. Para abordar esta resolución se utilizó por un lado el método iterativo SIP, este método permite trabajar con matrices penta y heptadiagonales, en el caso abordado en esta tesis las matrices que se generan desde la discretización de diferencias finitas, son heptadiagonales. Además, este método permite cierto grado de paralelismo al agrandar los dominios de resolución como se describió en el Capitulo 3 de la tesis. Las implementaciones Impl$_{cpu}$, Impl$_{gpu}$ están paralelizadas principalmente por

hiperplanos, por lo cual es necesario la utilización de un procedimiento auxiliar para el cálculo de los índices de dichos hiperplanos.

También se implementó el método BiCGStab, el cual se comparó con el método SIP, con el objetivo de verificar que este último es un solver con mejor desempeño para el tipo de problemas cubiertos en esta tesis.

Por último, se desarrolló un tercer esquema, Predictor - Corrector, con el propósito de aprovechar el grado de paralelización del esquema Explícito y la precisión numérica que ofrece el esquema Implícito. Como resultado se obtuvo un nuevo esquema, que mejora en precisión al esquema Explícito y en desempeño comparable al esquema Implícito.

Para los tres esquemas, como ya se comentó, se realizaron tres implementaciones contemplando los diferentes dispositivos de cálculo a utilizar en el trabajo. Entonces, se desarrollan las variantes secuenciales scExpl$_{cpu}$, scImpl$_{cpu}$, scPre-Cor$_{cpu}$, las paralelas usando OpenMP Expl$_{cpu}$, Impl$_{cpu}$, Pre-Cor$_{cpu}$, y las implementaciones sobre GPU, Expl$_{gpu}$, Impl$_{gpu}$, Pre-Cor$_{gpu}$.

En todos lo casos, las tres implementaciones de un mismo esquema alcanzan resultados con la misma precisión numérica y las pequeñas diferencias en los resultados numéricos son explicables por el uso de números de punto flotante.

Como primera conclusión de la evaluación experimental se puede decir que el esquema Explícito es el que permite mayor escalabilidad, ya que los otros esquemas requieren el uso de más datos en memoria, por ende dichos esquemas tienen limitantes que no les permiten alcanzar dominios de las dimensiones que alcanza el esquema Explícito.

Como segunda conclusión basada en los resultados obtenidos, se puede afirmar que el esquema Explícito tiene mejor desempeño computacional que el esquema Implícito y que el Predictor - Corrector. Además, las diferencias en el desempeño aumentan con la dimensión del problema abordado. Sin embargo, los resultados del esquema Implícito ofrecen una mejor precisión numérica.

En otra línea, y en base a la aplicación de paralelismo, el rendimiento paralelo de la CPU para los tres métodos ofrece beneficios similares. El uso de las implementaciones paralelas no otorga grandes beneficios para casos de pruebas con tamaños pequeños, pero en dominios con tamaños significativos estas implementaciones ofrecen grandes beneficios. Sin embargo, el uso de una GPU para los esquemas Explícitos presenta resultados mucho mejores que su uso para los otros esquemas, especialmente para la variante Implícita (notese que en el primer caso, el aumento de velocidad está cerca de un 10x al utilizar la GPU, mientras que en el caso implícito los tiempos de ejecución son similares a los de la CPU en paralelo). Este resultado es importante porque, con estos valores de aceleración, los problemas de precisión de los esquemas explícitos se pueden compensar mediante el uso de una grilla más fina para los cálculos. Cabe señalar que la precisión para el Caso 2, cuando se resuelve mediante el esquema Implícito es comparable a la precisión en el Caso 3 utilizando el esquema Explícito. Por otro lado, la versión basada en el uso de la GPU del esquema Explícito para el Caso 3 supera la mejor implementación del esquema Implícito sobre el Caso 2.

Finalmente, se realizó un estudio exhaustivo de cada método, combinando diferentes valores de Δt y coeficientes de convergencia para la realización de cada estudio y confrontando los resultados en frentes de Pareto.

La evaluación experimental llevada a cabo muestra que el método Implícito ofrece mejores resultados de precisión, pero el uso de las GPUs permite mayores valores de aceleración en el caso Explícito. Esta situación revela que los métodos Explícitos pueden superar sus contrapartes Implícitas, cuando son ejecutados sobre arquitecturas híbridas. Incluso si se considera establecer restricciones de precisión a alcanzar.

6.2. Difusión del trabajo en foros científicos

Algunos resultados obtenidos durante el desarrollo de la tesis aparecen publicados en diferentes foros de divulgación científica. A continuación se listan estas publicaciones agrupadas según si son revistas científicas, actas de congresos internacionales y actas de congresos regionales.

6.2.1. Revistas Científicas

- M.BONDARENCO; P.GAMAZO; P.EZZATTI. A comparison of various schemes for solving the transport equation in many-core platforms, *The Journal of Supercomputing*, January 2017, Volume 73, Issue 1, pp 469–481.

6.2.2. Conferencias Internacionales

- M.BONDARENCO; P.GAMAZO; P.EZZATTI. A trade-off between explicit and implicit schemes to solve differential equations on GPUs, *International Conference Computational and Mathematical Methods in Science and Engineering (CMMSE 2016)*, Rota-Cadiz, España, 2016.

- M.BONDARENCO; P.GAMAZO; P.EZZATTI. Assessing the explicit finite difference method on a massive parallel platform, *XLII Conferencia Latinoamericana de Informática (CLEI 2016)*, Valparaiso, Chile, 2016.

- M.BONDARENCO; P.GAMAZO; P.EZZATTI. Evaluación multi-criterio del uso de arquitecturas many-cores para acelerar la resolución de la ecuación de transporte, *International Conference Groundwater (ICGW 2017)*, Bogotá, Colombia, 2017.

6.2.3. Conferencias Regionales

- M.BONDARENCO; P.GAMAZO; P.EZZATTI. Evaluación de alternativas para la resolución de la ecuación de transporte en arquitecturas multi-many- cores, *Jornadas de Mecánica Computacional*, Arica, Chile, 2016.

- M.BONDARENCO; P.GAMAZO; P.EZZATTI. Resolución de la Ecuación de Transporte en arquitecturas Many-Cores evaluando esquemas Explícitos e Implícitos, *ENIEF*, La Plata, Argentina, 2017.

6.3. Trabajos Futuros

Durante el trabajo de tesis diferentes aspectos no fueron abordados o fueron cubiertos únicamente de forma preliminar. Muchos de estos aspectos merecen un abordaje en el futuro. A continuación se describe las líneas más promisorias:

- Se tiene la intención de estudiar el comportamiento de los esquemas abordados para otros casos de prueba (tanto de mayor dimensión como otras ecuaciones diferenciales). También evaluar otras plataformas, tarjetas gráficas de arquitecturas más modernas (Maxwell y Pascal) así como implementaciones multi-GPU, y sobre un cluster de servidores para estudiar casos de dimensiones mayores a las utilizadas en esta tesis.

- Desde el punto de vista físico se planea modificar la implementación para que use un campo de velocidades heterogéneo, es decir que cada celda de la dicretización contenga un campo de velocidades.

- En adición al punto anterior se plantea abordar problemas no lineales, para poder simular casos de estudios más complejos.

- Además, tenemos la intención de estudiar plataformas de hardware, que involucran otras características entre los que se destacan Intel Xeon Phi, procesadores ARM y plataformas distribuidas que incluyan a las anteriores.

Bibliografía

[1] S. Adams, J. Payne, and R. Boppana, "Finite difference time domain (FDTD) simulations using graphics processors," in *DoD High Performance Computing Modernization Program Users Group Conference, 2007*, June 2007, pp. 334–338.

[2] W. Aspray, "The intel 4004 microprocessor: What constituted invention?" *IEEE Annals of the History of Computing*, vol. 19, no. 3, pp. 4–15, 1997.

[3] L. B., "3d Finite Differences on Multi-core Processors," https://software.intel.com/en-us/articles/3d-finite-differences-on-multi-core-processors, 2011.

[4] R. B. Bird, W. E. Stewart, and E. N. Lightfoot, *Transport phenomena*. John Wiley & Sons, 2007.

[5] G. Blake, R. G. Dreslinski, and T. Mudge, *A survey of multicore processors*. IEEE, 2009, vol. 26, no. 6.

[6] J. Boris, M. Fritts, R. Madala, B. McDonald, N. Winsor, S. Zalesak, and D. Book, *Finite-difference techniques for vectorized fluid dynamics calculations*. Springer Science & Business Media, 2012.

[7] S. Borkar, "Thousand core chips: a technology perspective," in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, pp. 746–749.

[8] G. G. Brandao, "Solution of the Transport Equation using Graphical Processing Units," Ph.D. dissertation, MSc thesis IST, 2009.

[9] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. W. Cook, "Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors," *IEEE Micro*, vol. 20, no. 6, pp. 26–44, 2000.

[10] S. Cass, "Multicore processors create software headaches," *Technology Review*, vol. 113, no. 3, pp. 74–75, 2010.

[11] B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: portable shared memory parallel programming*. MIT press, 2008, vol. 10.

[12] J. Chen and J. Wang, "A 3D hybrid implicit-explicit FDTD scheme with weakly conditional stability," *Microwave and Optical Technology Letters*, vol. 48, no. 11, pp. 2291–2294, 2006.

[13] Y. Cotronis, E. Konstantinidis, and N. M. Missirlis, "A GPU implementation for solving the convection diffusion equation using the local modified sor method," in *Numerical Computations with GPUs*. Springer, 2014, pp. 207–221.

[14] R. Courant, K. Friedrichs, and H. Lewy, "Über die partiellen differenzengleichungen der mathematischen physik," *Mathematische annalen*, vol. 100, no. 1, pp. 32–74, 1928.

[15] V. Demir and A. Elsherbeni, "Compute unified device architecture (CUDA) based finite-difference time-domain (FDTD) implementation," *Journal of the Applied Computational Electromagnetics Society (ACES)*, vol. 25, no. 4, 2010.

[16] F. Deserno, G. Hager, F. Brechtefeld, and G. Wellein, "Basic optimization strategies for cfd-codes," *Technical report, Regionales Rechenzentrum Erlangen*, 2002.

[17] F. Deserno, "Basic Optimisation Strategies for CFD-Codes," Julio 2003.

[18] G. Dhatt, E. Lefrançois, and G. Touzot, *Finite element method*. John Wiley & Sons, 2012.

[19] L. Du, K. Li, and F. Kong, "Parallel 3d finite difference time domain simulations on graphics processors with Cuda," in *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*, Dec 2009, pp. 1–4.

[20] R. Eymard, T. Gallouët, and R. Herbin, "Finite volume methods," *Handbook of numerical analysis*, vol. 7, pp. 713–1018, 2000.

[21] Z.-G. Feng and E. E. Michaelides, "A numerical study on the transient heat transfer from a sphere at high reynolds and peclet numbers," *International Journal of Heat and Mass Transfer*, vol. 43, no. 2, pp. 219–229, 2000.

[22] J. H. Ferziger and M. Peric, *Computational methods for fluid dynamics*. Springer Science & Business Media, 2012.

[23] E. L. V. Galarza, "Taxonomías de flynn."

[24] D. Geer, "Chip makers turn to multicore processors," *Computer*, vol. 38, no. 5, pp. 11–13, 2005.

[25] S. Gerschgorin, "Fehlerabschätzung für das differenzenverfahren zur lösung partieller differentialgleichungen," *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 10, no. 4, pp. 373–382, 1930.

[26] M. B. Giles, "Crank–nicolson scheme," *Encyclopedia of Quantitative Finance*, 2010.

[27] J. Goodacre and A. N. Sloss, "Parallelism and the arm instruction set architecture," *Computer*, vol. 38, no. 7, pp. 42–50, 2005.

[28] W. G. Gray and G. F. Pinder, "An analysis of the numerical solution of the transport equation," *Water Resources Research*, vol. 12, no. 3, pp. 547–555, 1976.

[29] J. P. Guerrero, L. C. G. Pimentel, T. Skaggs, and M. T. van Genuchten, "Analytical solution of the advection–diffusion transport equation using a change-of-variable and integral transform technique," *International Journal of Heat and Mass Transfer*, vol. 52, no. 13, pp. 3297–3304, 2009.

[30] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[31] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHZ mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[32] P. Igounet, E. Dufrechou, M. Pedemonte, and P. Ezzatti, "A study on mixed precision techniques for a GPU-based SIP solver," 2012, pp. 7–12.

[33] P. Igounet, P. Alfaro, M. Pedemonte, and P. Ezzatti, "A Gpu implementation of the SIP method," in *Computer Science Society (SCCC), 2011 30th International Conference of the Chilean*. IEEE, 2011, pp. 195–201.

[34] F. P. Incropera, A. S. Lavine, T. L. Bergman, and D. P. DeWitt, *Fundamentals of heat and mass transfer*. Wiley, 2007.

[35] Joaquín Perez Badenes, "Consumo Energéticos de metodos iterativos para sistemas dispersos en procesadores gráficos," 2017, [Internet; descargado 17-marzo-2017]. [Online]. Available: {https://dialnet.unirioja.es/servlet/tesis?codigo=65869}

[36] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das, "A low latency router supporting adaptivity for on-chip interconnects," in *Proceedings of the 42nd annual Design Automation Conference*. ACM, 2005, pp. 559–564.

[37] V. Kindratenko, *Numerical Computations with GPUs*. Springer, 2014.

[38] D. B. Kirk and W. H. Wen-mei, *Programming massively parallel processors: a hands-on approach*. Newnes, 2012.

[39] H.-O. Kreiss, "Über die stabilitätsdefinition für differenzengleichungen die partielle differentialgleichungen approximieren," *BIT Numerical Mathematics*, vol. 2, no. 3, pp. 153–181, 1962.

[40] A. Kumar, D. K. Jaiswal, and N. Kumar, "Analytical solutions of one-dimensional advection-diffusion equation with variable coefficients in a finite domain," *Journal of Earth System Science*, vol. 118, no. 5, pp. 539–549, 2009.

[41] H.-J. Leister and M. Peric, "Vectorized strongly implicit solving procedure for a seven-diagonal coefficient matrix," *International Journal of Numerical Methods for Heat & Fluid Flow*, vol. 4, no. 2, pp. 159–172, 1994.

[42] Z. Li, J. Zhou, and D. Liu, "Performance of the Electromagnetic FDTD Parallel Program Based on OpenMP," *Modern Electronics Technique*, vol. 14, p. 048, 2008.

[43] M. Livesey, J. Stack, F. Costen, T. Nanri, N. Nakashima, and S. Fujino, "Development of a CUDA Implementation of the 3D FDTD method," *Antennas and Propagation Magazine, IEEE*, vol. 54, no. 5, p. 186–195, 2012.

[44] D. B. I. Lloyd N. Trefethen, *Numerical linear algebra*. Society for Industrial and Applied Mathematics, 1997.

[45] A. Marowka, "A Study of the Usability of Multicore Threading Tools," *International Journal of Software Engineering and Its Applications*, vol. 4, no. 3, pp. 1–8, 2010.

[46] M. G. McDonald and A. W. Harbaugh, "The history of MODFLOW," *Ground water*, vol. 41, no. 2, pp. 280–283, 2003.

[47] M. M. Meerschaert and C. Tadjeran, "Finite difference approximations for fractional advection–dispersion flow equations," *Journal of Computational and Applied Mathematics*, vol. 172, no. 1, pp. 65–77, 2004.

[48] P. Micikevicius, "3D finite difference computation on GPUs using CUDA," in *Proceedings of 2nd workshop on general purpose processing on graphics processing units*. ACM, 2009, pp. 79–84.

[49] F. Molnar, F. Izsak, R. Meszaros, and I. Lagzi, "Simulation of reaction–diffusion processes in three dimensions using CUDA," *Chemometrics and Intelligent Laboratory Systems*, vol. 108, no. 1, pp. 76–85, 2011.

[50] S. Moore, "Multicore is bad news for supercomputers," *IEEE spectrum*, vol. 45, no. 11, pp. 15–15, 2008.

[51] B. Nayfeh and K. Olukotun, "A single-chip multiprocessor," *Computer*, vol. 30, no. 9, pp. 79–85, 1997.

[52] B. Noye and H. Tan, "Finite difference methods for solving the two-dimensional advection–diffusion equation," *International Journal for Numerical Methods in Fluids*, vol. 9, no. 1, pp. 75–98, 1989.

[53] NVIDIA, "NVIDIA Corporation. NVIDIA CUDA C Programming Guide," 2017, [Internet; descargado 17-marzo-2017]. [Online]. Available: {http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz4bau3KTFA}

[54] C. Nvidia, "Nvidia CUDA C Programming Guide, version 7.0," *Nvidia Corporation*, 2015.

[55] J. Parkhurst, J. Darringer, and B. Grundmann, "From single core to multi-core: preparing for a new exponential," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 2006, pp. 67–72.

[56] D. Patterson, "The trouble with multi-core," *IEEE Spectrum*, vol. 47, no. 7, 2010.

[57] B. Perthame, *Transport equations in biology*. Springer Science & Business Media, 2006.

[58] T. Prosen and M. Robnik, "Energy transport and detailed verification of Fourier heat law in a chain of colliding harmonic oscillators," *Journal of Physics A: Mathematical and General*, vol. 25, no. 12, p. 3449, 1992.

[59] R. Ramanathan, "Intel® multi-core processors," *Making the Move to Quad-Core and Beyond*, 2006.

[60] A. Rosas and I. Herrera, "El número de peclet y su significación en la modelación de transporte difusivo de contaminantes," Ph.D. dissertation, Tesis de Licenciatura en Matemáticas. Facultad de Ciencias. UNAM. Director Dr. Ismael Herrera Revilla, 2005.

[61] A. Roy, J. Xu, and M. H. Chowdhury, "Multi-core processors: A new way forward and challenges," in *Microelectronics, 2008. ICM 2008. International Conference on*. IEEE, 2008, pp. 454–457.

[62] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents*. Addison-Wesley Professional, 2010.

[63] R. R. Schaller, "Moore's law: past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[64] J. Silva, J. Hagopian, M. Burdiat, E. Dufrechou, M. Pedemonte, A. Gutiérrez, G. Cazes, and P. Ezzatti, "Another step to the full GPU implementation of the weather research and forecasting model," *Journal of Supercomputing*, vol. 70, no. 2, pp. 746–755, 2014.

[65] G. D. Smith, *Numerical solution of partial differential equations: finite difference methods*. Oxford university press, 1985.

[66] S. A. Socolofsky and G. H. Jirka, "Special topics in mixing and transport processes in the environment," *Engineering—lectures, fifth ed., Coastal and Ocean Engineering Division, Texas A&M University*, 2005.

[67] A. C. Sodan, J. Machina, A. Deshmeh, K. Macnaughton, and B. Esbaugh, "Parallelism via multithreaded and multicore cpus," *Computer*, vol. 43, no. 3, 2010.

[68] H. L. Stone, "Iterative solution of implicit approximations of multidimensional partial differential equations," *SIAM Journal on Numerical Analysis*, vol. 5, no. 3, pp. 530–558, 1968.

[69] W. A. Strauss, *Partial differential equations*. Wiley New York, 1992, vol. 92.

[70] M. Thongmoon and R. McKibbin, "A comparison of some numerical methods for the advection-diffusion equation," 2006.

[71] A. M. Turing, "The chemical basis of morphogenesis," *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, vol. 237, no. 641, pp. 37–72, 1952.

[72] D. Unat, X. Cai, and S. B. Baden, "Mint: realizing CUDA performance in 3D stencil methods with annotated C," in *Proceedings of the international conference on Supercomputing*. ACM, 2011, pp. 214–224.

[73] H. A. Van der Vorst, "Bi-CGStab: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems," *SIAM Journal on scientific and Statistical Computing*, vol. 13, no. 2, pp. 631–644, 1992.

[74] D. A. Van Veldhuizen and G. B. Lamont, "Evolutionary computation and convergence to a pareto front," in *Late breaking papers at the genetic programming 1998 conference*, 1998, pp. 221–228.

[75] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys (CSUR)*, vol. 37, no. 3, pp. 195–237, 2005.

[76] J. Verwer, W. Hundsdorfer, J. Blom, *et al.*, "Numerical time integration for air pollution problems," *Surveys on Mathematics for Industry*, vol. 10, pp. 107–174, 2002.

[77] L. Wang, J. Tao, G. von Laszewski, and H. Marten, "Multicores in cloud computing: Research challenges for applications." *JCP*, vol. 5, no. 6, pp. 958–964, 2010.

[78] S. Wang, "Lax equivalence theorem," *Student's Book Numerical Functional Analysis*, p. 15, 2014.

[79] N. A. Wedge, M. S. Branicky, and M. C. Cavusoglu, "Computationally efficient cardiac bioelectricity models toward whole-heart simulation," in *Engineering in Medicine and Biology Society, 2004. IEMBS'04. 26th Annual International Conference of the IEEE*, vol. 2. IEEE, 2004, pp. 3027–3030.

[80] Wikipedia, "Taxonomía de Flynn — wikipedia, la enciclopedia libre," 2015, [Internet; descargado 16-marzo-2017]. [Online]. Available: {https://es.wikipedia.org/w/index.php?title=Taxonom%C3%ADa_de_Flynn&oldid=83092553}

[81] C. Zoppou and J. Knight, "Analytical solution of a spatially variable coefficient advection–diffusion equation in up to three dimensions," *Applied Mathematical Modelling*, vol. 23, no. 9, pp. 667–685, 1999.