UNIVERSIDAD DE COSTA RICA
SISTEMA DE ESTUDIOS DE POSGRADO

Paralelización del modelo PCell para la simulación de celdas de plasma convectivo

Tesis sometida a la consideración de la Comisión del Programa de Estudios de Posgrado en Computación e Informática para optar al grado y título de Maestría Académica en Computación e Informática

Daniel Alvarado González

Ciudad Universitaria Rodrigo Facio, Costa Rica

2018

Agradecimientos

Le agradezco a mi comité y a todos los que me ayudaron con este trabajo, en especial a Tracy, Guillermo, Gustavo, Jonatán y sobre todo a mis padres por su apoyo incondicional.

"Esta tesis fue aceptada por la Comisión del Programa de Estudios de Posgrado en Computación e Informática de la Universidad de Costa Rica, como requisito parcial para optar al grado y título de Maestría Académica en Computación e Informática"

Dr. Juan José Vargas Morales
Representante del Decano
Sistema de Estudios de Posgrado

Dr. Álvaro de la Ossa Osegueda
Director de Tesis

Dr. Esteban Meneses Rojas
Asesor

Dr. Francisco Frutos Alfaro
Asesor

Dra. Gabriela Marín Raventós
Directora Programa de Posgrado en Computación e Informática

Daniel Alvarado González
Sustentante

Índice general

Índice de figuras

Índice de cuadros

Índice de Abreviaturas

En este trabajo se utilizan las siguientes abreviaturas o acrónimos.

MHD:	Magnetohidrodinámica
CAR:	Computación de alto rendimiento
OpenMP:	Open Multi-Processing
MPI:	Message Passing Interface

Resumen

Las simulaciones de plasma son inherentemente complejas debido a la cantidad de procesos que ocurren naturalmente a la materia en este estado. Las simulaciones computacionales y las visualizaciones de plasma ayudan a investigadores y científicos a entender y estudiar sus propiedades y comportamiento. En esta investigación se desarrolló una implementación paralela de una aplicación utilizada para simular y visualizar el proceso de convección en celdas de plasma. Esta aplicación implementa un enfoque de magnetohidrodinámica (MHD) a la hora de simular los procesos del plasma. Los resultados de las evaluaciones experimentales con la aplicación paralelizada se presentan y analizan. Se alcanzó una aceleración del programa secuencial por un factor de alrededor de 42× después de optimizar algunas funciones y paralelizarlas con OpenMP utilizando 128 núcleos de un servidor Intel Xeon Phi KNL. También se logró obtener una escalabilidad lineal del tiempo de ejecución con respecto al tamaño de la matriz espacial utilizada y también con respecto a la cantidad de iteraciones temporales de la simulación.

Palabras clave

Magnetohidrodinámica, simulación computacional, plasma, OpenMP, procesamiento paralelo.

Capítulo 1

Introducción

En este capítulo se define el problema y se da una breve justificación de su relevancia. También se plantean los objetivos del trabajo.

1.1. Definición del problema

El plasma es un estado de la materia, en el que está la mayoría de la materia conocida en el Universo. En este estado, la materia se forma en una nube de protones, neutrones y electrones, en la que estos últimos se han desprendido de sus moléculas y átomos. Esto lo hace similar a un gas, con la diferencia de que el plasma actúa como una sola unidad, y no como diversos grupos de moléculas o átomos.

Su evolución involucra muchos procesos cinemáticos (de colisiones) y fuerzas electromagnéticas muy grandes, lo que hace que su comportamiento sea altamente complejo. La separación de cargas entre iones positivos y electrones en el plasma da lugar a campos eléctricos y estos flujos de partículas, a su vez, generan campos magnéticos, lo que produce un conjunto de procesos complejos.

El estudio del plasma tiene como propósito ayudar a comprender su comportamiento en la naturaleza a gran escala, así como las reacciones de fusión nuclear. Este fenómeno está siendo actualmente estudiado con miras a desarrollar muchas aplicaciones prácticas, por ejemplo, el plasma ha sido utilizado en la construcción de chips semiconductores y en el desarrollo de máquinas de rayos X más compactas [R.J Goldston, 1995].

Debido a la complejidad de los procesos que se llevan a cabo en los plasmas y a la

inmanejable escala en la que estos se presentan en objetos estelares, crear modelos matemáticos que simulen de una manera adecuada el comportamiento del plasma es vital para campos como la astrofísica y la física nuclear. Estas áreas de estudio han impulsado desarrollos fundamentales en física de plasma, incluyendo los estudios de propagación de ondas en plasmas, el confinamiento de plasma utilizando campos magnéticos y la magnetohidrodinámica (MHD), que es uno de los modelos más populares para plasmas magnéticos [Rémi Sentis, 2014] y en el cual se hará énfasis en este trabajo y se describirá con mayor detalle más adelante [1].

El programa PCell [Carboni and Frutos-Alfaro, 2005] fue implementado por investigadores de la escuela de Física de la Universidad de Costa Rica. Este es un programa que genera simulaciones y visualizaciones de plasma con un modelo de MHD. El programa se describe más a fondo en la sección 2.4.

En esta investigación se proponen mejoras al programa PCell, de acuerdo con la hipótesis siguiente:

Hipótesis: Es posible disminuir considerablemente el tiempo de ejecución del programa PCell y abarcar soluciones a problemas de mayor granularidad espacial aplicando estrategias de paralelización en arquitecturas masivamente paralelas.

1.2. Justificación

La simulación computacional ha sido un catalizador de la investigación y ha impulsado el desarrollo científico en las últimas décadas. La lista de disciplinas que la han incorporado crece constantemente y actualmente incluye la astrofísica, física de partículas, ciencia e ingeniería de materiales, mecánica de fluidos, ciencias del clima, biología evolutiva, ecología, economía, medicina, sociología, y muchas otras.

La simulación computacional es en esencia un método de experimentación que resuelve modelos matemáticos a través de métodos numéricos. Usualmente se utiliza como adición a, o sustituto de, los métodos tradicionales; especialmente cuando no es posible resolver los modelos de forma analítica o es muy costoso realizar directamente experimentación sobre el fenómeno, como en el caso del plasma.

[1] Ver sección 3.1

La física, particularmente, ha sacado mucho provecho de esta metodología, pues existe una gran cantidad de fenómenos *inaccesibles*, difíciles de observar o manipular, por ejemplo, el estudio de la dinámica de galaxias o la meteorología. La simulación computacional permite reproducir el comportamiento de dichos sistemas y manipularlos en un computador, lo que ayuda a entenderlos mejor. Los resultados de las simulaciones se comparan con datos tomados a partir de mediciones del fenómeno real, de esta manera se puede refinar el modelo, generando un lazo de retroalimentación que permite estudiar el fenómeno con mayor detalle [Hager and Wellein, 2011].

Las simulaciones computacionales se basan en modelos matemáticos que describen los sistemas bajo estudio. Algunos de estos modelos se construyen empleando ecuaciones diferenciales parciales. Los métodos numéricos para aproximar las soluciones a estas ecuaciones fueron inventados, desarrollados y aplicados a fenómenos reales mucho antes de la invención de las computadoras digitales, pero el desarrollo de estas ha impulsado su aplicación en distintas disciplinas [Glowinski and Neittaanmaki, 2008]. Estos nuevos modelos requieren de soluciones eficientes para obtener resultados en un tiempo aceptable.

El presente trabajo se enfoca en mejorar el tiempo de respuesta de la solución numérica de un esquema de ecuaciones diferenciales parciales que modelan algunos procesos del plasma convectivo. Se hace énfasis en la evolución del campo magnético del mismo.

Este tipo de sistemas permite el estudio de las condiciones solares, lo que es de suma importancia para los científicos espaciales, pues la mayoría de los eventos solares tiene repercusiones en la Tierra. Por ejemplo, las eyecciones de masa coronaria y eventos similares afectan el clima espacial [2], también pueden interferir con las comunicaciones satelitales y dañar las redes de transmisión de energía eléctrica. Estas simulaciones computacionales son relevantes, pues nos ayudan a estudiar y comprender los fenómenos físicos en la superficie del Sol y otros objetos estelares, para así poder desarrollar predicciones y estrategias para proteger a la población y a la infraestructura global de telecomunicaciones, entre otras cosas [Space and Societal, 2008].

El programa PCell es un software especializado para la simulación de la evolución del campo magnético en celdas de plasma conectivo. Este software es utilizado por

[2] El clima espacial consiste en el conjunto de condiciones creadas por el viento solar, las cuales son diferentes al espacio interestelar, dominado por rayos cósmicos

investigadores de la Escuela de Física de la Universidad de Costa Rica. El interés por este tipo de herramientas de simulación ha hecho que los investigadores realicen sus propias implementaciones computacionales de los modelos que estudian.

PCell fue desarrollado en 1994 como parte de una tesis de maestría. Los resultados obtenidos a partir de este programa son de gran utilidad para mejorar la comprensión del comportamiento de los campos magnéticos en cuerpos celestes [Carboni, 1994]. Permite estudiar también la manera en la que el número de Reynolds magnético afecta a los campos magnéticos, tanto su forma, como su comportamiento y propiedades. Este programa ha sido específicamente utilizado para estudiar el fenómeno de la reconexión magnética, que es un fenómeno que se da en el Sol cuando las líneas del campo magnético de una erupción coronaria solar se reconectan, dando lugar a una expulsión de material [Galloway and Weiss, 1981].

El Sol tiene una región convectiva con alta turbulencia, lo que produce números de Reynolds magnéticos del orden de 10^6 a 10^{10} [Carboni, 1994]. En su estado actual, el programa PCell permite estudiar fenómenos solares con ciertas limitaciones. La primera de ellas es que el programa se limita a procesar correctamente simulaciones complejas con números de Reynolds inferiores a 10^3, sin embargo, en la superficie del Sol se presenta una gran cantidad de regiones para las que ese número es muy superior a 10^3, y que por lo tanto no pueden ser estudiadas con PCell.

Una segunda limitación es el dominio espacial de la simulación. PCell ha sido típicamente utilizado con rejillas cuadradas de 100 x 100 puntos para simular el espacio físico del plasma. Para cada uno de esos puntos se debe de resolver un conjunto de ecuaciones de MHD, es decir, debe de resolver en total 10000 conjuntos de ecuaciones por unidad de tiempo simulado. Un aumento en el dominio espacial, por pequeño que sea, se convierte en una carga computacionalmente significativa para PCell.

La tercera limitación de PCell corresponde a su escala temporal. Para que las simulaciones de PCell sean significativas para su área de estudio, es necesario que las mismas tengan una longitud temporal específica para los parámetros del fenómeno que se está estudiando. Dependiendo del número de Reynolds y del valor del campo de velocidad del plasma, las simulaciones requieren de más tiempo para alcanzar estados relevantes para su estudio, como por ejemplo, alcanzar la densidad de energía magnética media máxima. En [Carboni, 1994] se muestra que existe una correlación entre el aumento del

número de Reynolds magnético y el aumento del tiempo necesario para que la simulación alcance el estado de densidad de energía magnética media máxima. Este estado es de interés para la investigación pues es cuando el sistema alcanza un estado estable después de haber estado en un estado oscilatorio o caótico. El tiempo de ejecución total del modelo secuencial depende de la arquitectura donde se ejecute. En Kabré, para una ejecución de PCell corta (con 100 pasos temporales), el programa tarda entre 10 y 15 minutos en hacer una simulación.

Debido a las limitaciones que presenta el software, en este trabajo de investigación se propone una implementación paralela de PCell, que llamaremos en adelante *ParaPCell*, con el fin de solventar algunas de ellas.

En este trabajo también se expondrá un caso de evaluación de un método de paralelización aplicado a un método numérico para resolver ecuaciones diferenciales aplicadas a plasma. Esto permitirá hacer un análisis de la aptitud de la estrategia de paralelización que se utilizó para este tipo de sistemas.

1.3. Objetivos

1.3.1. Objetivo Principal

- Mejorar el rendimiento del modelo PCell mediante la aplicación de técnicas de paralelización para disminuir su tiempo de ejecución y ampliar las escalas espaciales y temporales de las simulaciones

1.3.2. Objetivos Específicos

- Diseñar e implementar en paralelo el programa PCell para mejorar el tiempo de respuesta del programa a escalas más grandes
- Analizar y comparar empíricamente los tiempos de ejecución de la versión secuencial y la versión paralela de PCell en arquitecturas de alto rendimiento
- Evaluar la precisión de la implementación paralela de PCell

En el siguiente capítulo se discutirá parte del trabajo relacionado en el área de

simulaciones de MHD. Se expondrán algunas herramientas que tienen modelos similares, pero que resuelven problemas distintos al de PCell.

Capítulo 2

Trabajo relacionado

La mejora del rendimiento de los programas que resuelven problemas de física sigue siendo un tema relevante en trabajos recientes. Conforme se realizan modelos más precisos, complejos y que representan mejor los fenómenos físicos, aumenta la necesidad de mejorar el rendimiento de los mismos para que presenten resultados en tiempos razonables.

Estos esfuerzos no se aplican únicamente al ámbito de la MHD, sino también con muchos otros tipos de soluciones para modelado de plasma, como por ejemplo, el acercamiento de *particle in cell* [Tang et al., 2017], que es una técnica para resolver ecuaciones diferenciales parciales generalmente aplicada a simulaciones en las que en lugar de ver el fluido como un solo cuerpo con características hidrodinámicas, como en la MHD, se simula como un conjunto de partículas que interactúan entre sí en un espacio confinado [Dawson, 1983]. Específicamente en problemas de MHD se han explorado varias soluciones de modelado que incluyen un componente de computación paralela y de alto rendimiento, la gran mayoría relativamente recientes, algunas de las cuales se describen adelante.

Existen varias implementaciones de modelos de MHD para plasma, pero este es un campo amplio que puede ser abarcado desde muchos posibles ángulos. PCell utiliza el modelado de MHD para simular y visualizar la evolución del campo magnético del plasma convectivo en celdas confinadas. Las aplicaciones de física de plasmas suelen ser para un área de estudio bien definida y limitada, lo que hace que exista una gran cantidad de programas tanto especializados como de propósitos más generales que pue-

den ser utilizados para ejecutar un modelo de MHD. La diferencia clave yace en la especialización del área de estudio de los programas que implementan estos modelos, es decir, el énfasis en el fenómeno o conjunto específico que se desee estudiar.

2.1. XTOR

En [Lütjens and Luciani, 2008] se describe el programa XTOR, que resuelve un sistema de ecuaciones de MHD para simular el comportamiento del plasma dentro de una estructura especializada para su contención llamada *tokamak*. Esta es una estructura toroidal de metal utilizada por investigadores de física de plasma para contenerlo de manera segura mediante un campo magnético. A pesar de que este programa simula plasma mediante el modelo de MHD, se limita a simulaciones en tokamak. Las simulaciones que se desea hacer con PCell describen el comportamiento del plasma en el Sol, y no en un ambiente controlado de laboratorio. Por esta razón, XTOR no es una herramienta adecuada para el problema de esta investigación.

Las simulaciones realizadas con XTOR son en dominios tridimensionales, lo cual incrementa su complejidad. En el trabajo se menciona la importancia que tuvo el uso de estrategias de paralelismo e infraestructura de computación de alto rendimiento (CAR) para que la tarea se pudiera resolver en un tiempo aceptable, mencionando que las simulaciones típicas que ejecutaban están en el orden de 10 horas CPU cada una.

2.2. WHAM

El programa WHAM [Tchekhovskoy et al., 2007] modela plasma a través de MHD en objetos estelares a gran escala. Este programa no hace énfasis en los campos magnéticos, sino en los flujos de masa y energía que se dan en el plasma convectivo en objetos astronómicos como agujeros negros o estrellas, en la manera en la que se forman expulsiones de materia a grandes velocidades y la energía asociada a los mismos. Este es un dominio muy distinto al que se pretende estudiar con PCell, a pesar de que WHAM hace uso de un transfondo fuerte de MHD, su área de aplicación última es muy distinta.

Los autores mencionan que utilizan una estrategia de paralelización a nivel de datos espaciales. Separan el área de simulación en dominios distintos que posteriormente dis-

tribuyen a distintos procesadores utilizando el estándar de paralelización para memoria distribuida MPI. Mencionan también que esta separación de tareas les permite alcanzar un incremento de no menos que el 70 por ciento de eficiencia paralela con respecto a una versión secuencial del mismo programa en una arquitectura de 256 CPUs.

2.3. Uso de integradores exponenciales en otras aplicaciones

Uno de los principales problemas en MHD es la integración temporal. No sólo es difícil realizar este proceso de manera analítica, sino que también las soluciones numéricas que existen para este tipo de ecuaciones tienen una alta demanda de poder computacional debido al rango temporal que tienen las ecuaciones [Jardin, 2012]. Es por esto que el desarrollo de integradores temporales es uno de los mayores retos del desarrollo de MHD. En [Einkemmer et al., 2017] se describe el uso de integradores exponenciales como una alternativa eficiente para integrar sistemas de ecuaciones diferenciales grandes que modelan procesos de MHD. La aplicación de estos integradores en este campo es un área de estudio reciente. Los autores mencionan que aún no tienen resultados de su método en sistemas de gran escala, pero mencionan la mejora en rendimiento que aporta la implementación y adaptación de sus métodos para arquitecturas paralelas que han desarrollado en C++ y utilizando el estándar de paralelización MPI. Específicamente mencionan incrementos en el rendimiento con estrategias de paralelización con respecto al incremento en la escala de algunas de las magnitudes de parámetros importantes como el número de Reynolds. Sus métodos fueron probados en algunos programas que resuelven ecuaciones de MHD para modelar plasma, mostrando finalmente buenos resultados con respecto a métodos más tradicionales de integración.

2.4. PCell

Una de las implementaciones que existen del modelo de MHD para plasma es el programa PCell [Carboni and Frutos-Alfaro, 2004]. Este software es utilizado para simular y visualizar la evolución del campo magnético en celdas de plasmas de diversas conductividades cuando este es sometido a un movimiento convectivo, que es un movimiento

causado por la transferencia de energía en el material. Estas celdas son espacios finitos y limitados por la simulación y en el que se definen condiciones de frontera. El programa resuelve de manera numérica algunas ecuaciones de MHD y toda la información de la evolución del campo magnético que estas dan como resultado es desplegada de manera gráfica mediante un archivo de video que muestra las transformaciones de la intensidad magnética y la densidad de energía magnética en el plasma simulado [Carboni, 1994].

La versión original de PCell es secuencial, lo que restringe tanto las dimensiones del problema como el nivel de detalle de la simulación. Por ejemplo, si se utiliza una escala espacial muy fina, la cantidad de cálculos que debe de realizarse para resolver las ecuaciones para un instante de tiempo incrementa grandemente, pues el programa utiliza una matriz para simular el espacio en dos dimensiones, por lo que el aumento en dichas dimensiones, implica un crecimiento cuadrático en la cantidad de cálculos que se deben de hacer, esto hace que la solución existente del problema sea temporalmente compleja. Una implementación más eficiente de PCell, como por ejemplo una versión paralela, podría permitir a investigadores en astrofísica estudiar más detallada y ampliamente el comportamiento del plasma convectivo.

Capítulo 3

Marco Teórico

A continuación se proporcionará una explicación breve de algunos conceptos que corresponden a la base teórica en la que se fundamenta la física que se simula con el programa PCell, que será la base de la construcción de la herramienta que se desarrollará en este trabajo. Se hace también una breve explicación de algunas tecnologías de paralelismo que serán utilizadas para mejorar el rendimiento en la nueva herramienta, y algunas definiciones básicas de conceptos de paralelismo, perfilado y medición de rendimiento.

3.1. Magnetohidrodinámica

La magnetohidrodinámica (MHD) es el estudio de la interacción entre campos magnéticos y fluidos conductores en movimiento. El movimiento relativo de un fluido conductor con respecto a un campo crea un campo electromagnético que se desarrolla de acuerdo con la ley de inducción de Faraday [Davidson, 2001].

Las ecuaciones de Maxwell [Murphy, 2014] determinan el comportamiento del campo electromagnético de fluidos cósmicos. Es posible hacer una simplificación de estas ecuaciones al considerar al plasma como un solo fluido con muy alta conductividad. Dicha consideración implica que cualquier separación de cargas será cortada, debido a la conductividad casi perfecta del medio. El enfoque de la MHD puede ser aplicado cuando las escalas espaciales y temporales que interesa modelar son lo suficientemente grandes [Carboni and Frutos-Alfaro, 2004].

MHD es un tipo de modelo de plasma que se enfoca en las cualidades macroscópicas como la presión, la densidad y la conductividad. Cuando las escalas espaciales y temporales que interesa modelar son lo suficientemente grandes, este enfoque se convierte en un método adecuado para modelar este tipo de sistemas [Verboncoeur, 2005].

Comúnmente las simulaciones de plasma puede tomar uno de varios enfoques, por ejemplo: modelado de partículas, descripción cinética, MHD, mecánica cuántica, entre otros [Chen, 2016]. El programa PCell resuelve un sistema de ecuaciones diferenciales que modela el campo magnético del plasma bajo la perspectiva de MHD.

3.2. Número de Reynolds magnético

El número de Reynolds magnético es un parámetro de los modelos de MHD. Este parámetro escalar influye sobre el comportamiento dinámico del plasma. Este número es análogo al número de Reynolds que se utiliza regularmente en la mecánica de fluidos, que determina la turbulencia de un fluido. Este número da un estimado de los efectos relativos de la inducción de un campo magnético dado por el movimiento de un medio conductor con respecto a la difusión magnética, dado por:

$$R_m = \frac{UL}{\eta}, \tag{3.1}$$

donde U es la velocidad del flujo, L es la longitud característica y η es la viscosidad magnética [Carboni and Frutos-Alfaro, 2004].

Este parámetro determina la turbulencia del plasma que se simulará en el programa PCell. El número de Reynolds afecta la complejidad de la simulación y por lo tanto la cantidad de cálculos necesarios para que esta alcance un estado estacionario. En la figura (3.1) se muestran varios ejemplos con distintos valores de número de Reynolds y velocidad del fluido de plasma y cómo estos afectan a la simulación. Entre mayor es este parámetro, mayor es la turbulencia del plasma, lo que hace que sea más propenso que se generen torbellinos en el fluido.

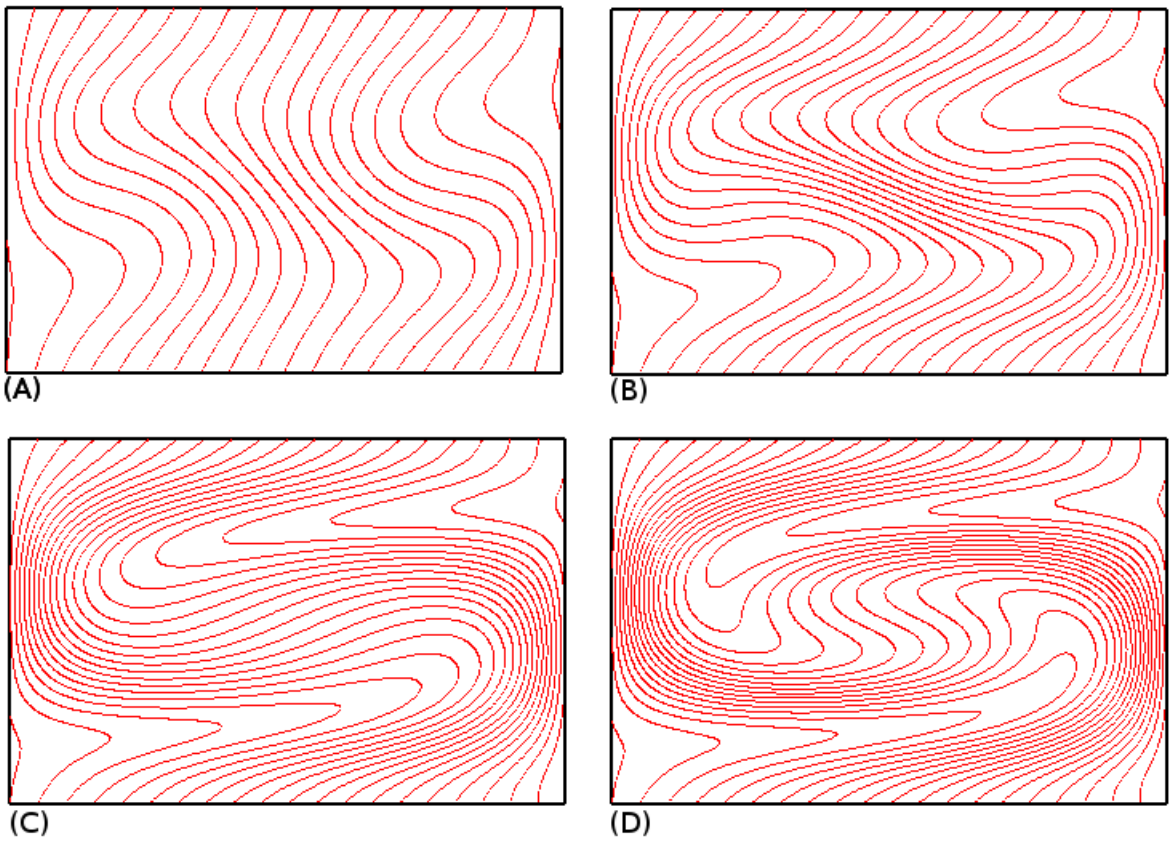

Figura 3.1: Estados de un fluido magnético para distintos valores del número de Reynolds (R_m) y velocidad del plasma (m): (a) $R_m = 200, m = 0{,}1$, (b) $R_m = 500, m = 0{,}3$, (c) $R_m = 800, m = 0{,}7$, (d) $R_m = 1000, m = 0{,}9$. Elaboración propia

3.3. Ecuación de inducción magnética

Los modelos de MHD dependen de sistemas de ecuaciones diferenciales, y en el caso de PCell específicamente de la ecuación de inducción, que es parte de las ecuaciones de Maxwell para describir el comportamiento de los campos electromagnéticos. Esta ecuación juega un rol importante en estos sistemas [Balsara and Käppeli, 2017]. La ecuación se escribe a continuación, y el desarrollo de las ecuaciones subsiguientes se toma del planteamiento original del modelo PCell descrito en [Carboni and Frutos-Alfaro, 2015]:

$$\frac{\partial B}{\partial t} = \nabla \times (v \times B) + \eta \nabla^2 B, \tag{3.2}$$

donde $\eta = 1/\mu\sigma$ es la viscosidad magnética, μ la permeabilidad magnética, σ la conductancia, B el campo magnético, y v la velocidad del campo que describe el movimiento del plasma. Para efectos de esta simulación, los efectos producidos por el gradiente de temperatura y las fluctuaciones de densidad de las partículas cargadas no son tomados en cuenta.

La ecuación de inducción (3.2) puede ser simplificada para el dominio bidimensional si se escribe como una función del potencial vectorial. Se toma el campo magnético y la velocidad de campo limitada al plano X−Y, luego el campo magnético se obtiene de la componente del potencial vectorial $A = Ak$ de la siguiente manera:

$$B = \nabla \times A = \left(\frac{\partial A}{\partial y}, -\frac{\partial A}{\partial x}, 0 \right) \tag{3.3}$$

Después de hacer la substitución en (3.2), el resultado es:

$$\frac{\partial A}{\partial t} = -u \cdot \nabla A + \eta \nabla^2 A \tag{3.4}$$

Las variables de posición y velocidad se definen como una función de los parámetros característicos (velocidad máxima U y longitud máxima L) de la siguiente forma:

$$u = u'U, x = x'L, \tag{3.5}$$

con una definición análoga para la coordenada en y. Las derivadas espaciales se dan por:

$$\frac{\partial}{\partial x} = \frac{1}{L}\frac{\partial}{\partial x^t} \tag{3.6}$$

Y

$$\frac{\partial^2}{\partial x^2} = \frac{1}{L^2}\frac{\partial}{\partial x^{t2}} \tag{3.7}$$

Después de hacer la substitución de estas relaciones y definir el tiempo característico del movimiento mecánico $\tau_0 = L/U$, si se toma por ejemplo la velocidad de gránulos, mesogránulos y supergránulos en el Sol, tenemos $V = 900, 60$ y $400 m/s$, y $L = 1{,}4 \times 10^3, 7 \times 10^3$ y 3×10^5km, respectivamente dan un $\tau_0 = 26$ min, 1,35 y 0,87 días.

Este tiempo τ_0 mide cuánto le toma al plasma movilizarse desde el fondo hasta la

cima de la celda. Con $t = t'\tau$ obtenemos

$$\frac{\partial A}{\partial t} = u \cdot \nabla A + \frac{1}{R_m}\nabla^2 A \tag{3.8}$$

donde las primas se han removido para mayor claridad.

Esta ecuación (3.8) es la que resuelve PCell bajo la condición de que no existe reacción del campo magnético en el plasma, lo que deja la velocidad de campo independiente del tiempo. Este acercamiento es válido si la energía magnética es pequeña comparada con la energía cinemática del plasma [Carboni and Frutos-Alfaro, 2015].

Para resolver la ecuación 3.8, se divide el espacio bidimensional en una rejilla de puntos con el propósito de discretizarlo. De la misma manera se divide el tiempo en puntos discretos. A partir de esta discretización, la variación temporal se aproxima por medio de diferencias finitas centradas de segundo orden sobre cada punto.

3.4. Diferencias finitas

Los métodos de diferencias finitas se utilizan para resolver ecuaciones diferenciales por medio de aproximaciones. Son métodos de discretización para aproximar las derivadas de las ecuaciones. Son uno de los principales grupos de métodos numéricos utilizados para resolver ecuaciones diferenciales parciales. Este método es el que utiliza PCell para solucionar las ecuaciones del modelo.

Antes de resolver una ecuación mediante algún método de diferencias finitas, es necesario delimitar y discretizar el dominio para el que se desea resolver. Esto generalmente se hace con una grilla o matriz, de forma que el dominio se convierte en un conjunto de puntos discretos y finitos. Los puntos de la grilla están separados de los adyacentes por una distancia h en todas direcciones.

Uno de los métodos de diferencias finitas es la diferencia central, que es una medida de las diferencias anteriores y posteriores. Está dada por:

$$\delta_h f(x) = f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)$$

Como la derivada de la función f en un punto x está definida por el límite [AMES, 1977]:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

en diferencia central, conforme el valor de la distancia h se aproxima a cero, el valor resultante será cada vez más cercano a la derivada de la ecuación. Este método es el utilizado en PCell para discretizar el espacio bidimensional de la simulación.

3.5. Simulaciones de plasma y MHD

Crear un software general para realizar simulaciones en alguna área de la física es una tarea difícil debido a la gran diversidad de problemas que existen y la gran variedad de modelos que continuamente se están creando y refinando para estudiar y experimentar con los distintos fenómenos. La cantidad de parámetros, las condiciones iniciales y de frontera, y otros aspectos pueden variar mucho entre problemas y entre modelos.

En particular, las simulaciones de plasma deben de incluir todos los fenómenos relacionados con gases altamente ionizados. Estos gases contienen iones positivos y negativos y electrones libres. Todas estas partículas acarrean cargas eléctricas que generan campos electromagnéticos cuando están en movimiento, los cuales a su vez afectan a otras partículas a su alrededor. Las partículas de plasma están en constante movimiento, esto da lugar a una separación de cargas que genera campos eléctricos y a su vez estos campos generan una reacción en cadena que ponen en movimiento a otras partículas de plasma [Verboncoeur, 2005].

La gran cantidad de interacciones entre estos procesos hace que el problema rápidamente se torne difícil de manejar mediante una solución computacional secuencial. Muchos de estos problemas se suelen resolver con soluciones paralelas que explotan adecuadamente los recursos computacionales con los que se cuenta actualmente [Einkemmer et al., 2017].

El enfoque de MHD ayuda a simplificar un poco estos modelos. Sin embargo, esto no quiere decir que los modelos de MHD no sean también muy demandantes computacionalmente. Conforme los modelos matemáticos de simulación son refinados, su precisión de representación del fenómeno que simulan incrementa, al aumentar esta complejidad

de los modelos, también aumenta su demanda de poder computacional para poder resolverlos, por lo que la integración de estos modelos con técnicas de computación de alto rendimiento se vuelve cada vez más una necesidad [Kageyama, 2001].

3.6. Paralelismo

El incremento del poder computacional en unidades de procesamiento por unidad de tiempo ha sido un propulsor vital del uso de herramientas computacionales para resolver problemas científicos en las últimas décadas, especialmente debido a la complejidad que tienen las soluciones a problemas relevantes. Conforme se hacen avances en la ciencia, las aplicaciones necesariamente van creciendo en complejidad y es cada vez más necesario utilizar herramientas de paralelismo.

La ventaja principal de diseñar e implementar programas para arquitecturas paralelas es el incremento en el rendimiento. Las soluciones a problemas científicos pueden fácilmente crecer en cuanto a la cantidad de tiempo necesario para resolverlos si se hacen de manera secuencial debido a lo complejo que suelen ser los procesos y a la cantidad masiva de datos con los que se suele trabajar [Sevcik, 1989].

Existe u n umbral en la capacidad que tienen los materiales con los que se construyen los transistores y otros componentes de los microprocesadores sobre todo en espacios tan pequeños. Este umbral fija un límite bajo el cual los componentes funcionan adecuadamente, que involucra ciertas condiciones de temperatura y carga eléctrica. Como la necesidad de procesamiento ha ido incrementando a un ritmo sostenido y estas limitaciones físicas imponen una barrera para la capacidad de procesamiento de los procesadores, los últimos avances de la computación de alto rendimiento han sido dirigidos hacia el paralelismo [Hager and Wellein, 2011].

3.6.1. Herramientas de paralelismo

Dentro de los paradigmas de paralelismo existentes, está el paradigma de memoria compartida. Este paradigma aporta ventajas en ciertos escenarios, de la misma manera, dependiendo del problema que se desee resolver y de la arquitectura de la(s) computadora(s) donde se ejecute, podrá presentar también algunas desventajas.

Memoria compartida

En el esquema de memoria compartida los distintos procesos que estén ejecutando una tarea tienen acceso a una misma sección de la memoria de la computadora donde están corriendo de manera asíncrona, esta memoria es utilizada por las distintas unidades de procesamiento con el fin de proveer comunicación entre ellas. Este paradigma es una manera eficiente de compartir datos, pues al compartir el mismo espacio de memoria, todas las unidades de procesamiento tienen acceso a las mismas variables. Esta estrategia de paralelismo es usualmente implementada con OpenMP [OpenMP Architecture Review Board, 2015].

OpenMP significa Open Multi Processing, su API consiste en un conjunto de directivas que el programador utiliza para implementar paralelismo en un nodo multi-núcleo. Estas directivas (o pragmas) le dicen al compilador dónde existen partes del código que pueden ser paralelizadas. Este paradigma permite hacer una paralelización a nivel de *hilos* dentro de un proceso. En cualquier aplicación de OpenMP existe un hilo principal que ejecuta el código desde el inicio del programa, una vez que la ejecución alcanza un punto en el código donde hay directivas de paralelización, se genera un número especificado de hilos "esclavos' que tendrán un espacio de memoria común para todos, la cantidad de hilos generados puede ser designado mediante una variable de ambiente o mediante código. Un compilador que no sea compatible con OpenMP simplemente ignorará las directivas y producirá un código secuencial [Hager and Wellein, 2011].

OpenMP hace paralelismo a nivel de hilos de un mismo proceso. En cualquier aplicación con OpenMP existe un hilo que ejecuta desde el inicio. Posteriormente, el usuario define secciones donde este hilo inicial se subdivide en un conjunto de hilos de ejecución que ejecuta las instrucciones concurrentemente dentro de la región definida.

OpenMP es una implementación de *multithreading*, que es un método de paralelización donde un hilo principal o "maestro', genera un conjunto de hilos "esclavos' para dividir las tareas de ejecución entre ellos. Cada uno de estos hilos ejecuta de manera concurrente, y el sistema se encarga de asignarlos a sus respectivas unidades de procesamiento. Una vez que los hilos esclavos terminan sus tareas, estos cesan su ejecución y liberan cualquier sección de memoria privada que tuvieran, el hilo maestro continúa con cualquier instrucción que le falte fuera de la sección paralela.

Para poder explotar de una manera adecuada el poder de procesamiento de una

computadora, es necesario adaptar el programa para aprovechar la arquitectura en la que se va a ejecutar. También es necesario conocer bien el programa, de manera que se puedan identificar las partes del código que requieren de mayor tiempo de procesamiento. Este proceso de análisis se realiza comúnmente con ayuda de herramientas llamadas *profilers*.

3.7. Profiling

En análisis de software, el *profiling* (perfilado o análisis de rendimiento) es un proceso mediante el cual se toman algunas medidas de la ejecución de un programa para realizar una evaluación de la implementación realizada. Este proceso le ayuda a los desarrolladores a identificar problemas de rendimiento o posibles áreas de mejora en sus programas [Graham et al., 1982].

Los programas escritos para hacer este proceso de manera automática se conocen como *profilers*. Estos programas permiten encontrar las fuentes de bajo rendimiento en un programa, como por ejemplo cuellos de botella de entrada y salida, ciclos recorridos de forma ineficiente o simplemente la parte de un programa que toma la mayor cantidad de tiempo en ejecutar. Una vez que se identifican las secciones de código que consumen la mayor cantidad de tiempo, el programador puede enfocarse en mejorar la eficiencia de estas secciones críticas y así mejorar el rendimiento del programa. Este proceso se puede repetir iterativamente hasta lograr un rendimiento aceptable [Gaur et al., 1989].

3.7.1. Gprof

Gprof es una herramienta de análisis de rendimiento (profiler) para aplicaciones Unix [Graham et al., 1982]. Este profiler realiza un desglose de cada llamado a rutinas o funciones que el programa analizado ejecuta. La salida muestra una tabla con la duración total en segundos que tiene cada una de las rutinas llamadas, la cantidad de veces que fue llamada cada una de ellas y el porcentaje del tiempo total de ejecución que representa.

Gprof muestra las relaciones de llamados entre subrutinas o funciones del programa, gracias a esto se puede conocer el orden de ejecución y la jerarquía de llamados del programa. Este profiler utiliza interrupciones del sistema para realizar muestreos

periódicos del *program counter* del software. La instrumentación del código se inserta de forma automática en el momento de compilación utilizando la bandera *-pg* con el compilador *gcc*. Una vez que se ejecuta el programa, el profiler genera de forma automática un archivo 'gmon.out', el cual puede ser después analizado con la herramienta de visualización de datos del mismo paquete, la cual genera de forma automática un desglose de funciones ejecutadas y tiempos de ejecución y una tabla donde se despliegan los datos de las mediciones hechas [Graham et al., 1982].

La salida consiste en un perfilado temporal y un grafo de llamados. El primero consiste en una tabla donde se despliega el tiempo que se invirtió en cada función y el porcentaje del tiempo de ejecución total que representó esa función. También se reporta el conteo de llamados a cada función. Esta tabla es ordenada de mayor a menor por el porcentaje del total del tiempo de ejecución que abarcó cada rutina. La segunda parte de la salida es el grafo de llamados, que para cada función muestra la función que la llamó y a quiénes esta llamó.

La tabla de tiempos permite enfocarse de manera precisa en la parte del código que consume la mayor parte del procesamiento. Esta herramienta es útil para el programador, pues le brinda datos de mediciones reales de ejecución que pueden ser utilizadas para saber con certeza dónde se deben de enfocar los esfuerzos de paralelización y mejora del rendimiento por parte del programador.

3.7.2. Valgrind

Valgrind es una herramienta de programación para hacer depuración de memoria. Con ella se puede hacer profiling y análisis de los posibles errores en el código que causen fugas de memoria. Su principal enfoque es el análisis del uso de la memoria por parte del programa para detectar errores o malos manejos de la memoria y de hilos [val, 2017].

Originalmente fue diseñado para ser una herramienta gratuita que funcionara para plataformas Linux en x86, pero desde su origen ha evolucionado para convertirse en uno de los *frameworks* más populares para hacer este tipo de análisis de memoria. En este momento cuenta con implementaciones para Linux, MacOS, Solaris y Android.

Valgrind es en esencia una máquina virtual. Nada del código original del programador se ejecuta directamente en el procesador de la máquina donde se ejecuta. *Valgrind*

primero traduce el programa a una representación intermedia, que es un código independiente de la arquitectura en la que se ejecuta. Luego de generar este código intermedio, la herramienta lo manipula para ingresar las rutinas de monitoreo de memoria. Posteriormente, se traduce el código de vuelta a código de máquina para ser ejecutado por el procesador. Una cantidad considerable del rendimiento del programa analizado se pierde en estas transformaciones, por lo que *Valgrind* es bueno para hacer depuración, pero debe de ser retirado a la hora de ejecutar el programa en un ambiente de pruebas de rendimiento.

Para realizar las pruebas de memoria, *Valgrind* inserta código de monitoreo alrededor de casi todas las instrucciones para mantener un registro de la validez de la memoria que se utiliza y al mismo tiempo verificar si es posible para el programa direccionar las posiciones de memoria referenciadas y de esta manera asegurarse de que no se accede en ningún momento a alguna posición inválida. Esta herramienta detecta y notifica sobre problemas como la utilización de memoria no inicializada, lectura/escritura de memoria después de ser liberada, lectura/escritura fuera de memoria asignada y otros tipos de fugas de memoria [val, 2017].

La salida de la herramienta consiste en un conjunto de mensajes que describen cada uno de los errores de memoria. La primera línea de cada mensaje indica el tipo de error que se dio. Luego se despliega el historial de la pila para indicar dónde sucedió el problema, tanto la posición de memoria como la línea de código donde se dio el problema.

Realizar un proceso de perfilado, nos puede ayudar a conocer las áreas de mejora del programa. Para obtener el máximo rendimiento para el programa, es necesario también conocer bien la arquitectura en la que se va a ejecutar. De esta manera se conoce la estrategia de paralelización que se adapta mejor al equipo computacional que se tiene disponible.

3.8. Taxonomía de Flynn

La taxonomía de Flynn es una clasificación de arquitectura de computadoras propuesto por Michael J. Flynn [Flynn, 1972]. Existen cuatro clasificaciones definidas por Flynn basadas en el número de instrucciones concurrentes o flujos de datos con los que

trabaje. Esta taxonomía sirve para clasificar arquitecturas de computadoras.

3.8.1. Single instruction single data (SISD)

Una única instrucción, un único dato. Son las computadoras secuenciales, que no implementan arquitecturas paralelas. Tienen una sola unidad de control que procesa las instrucciones una a una con capacidad para trabajar en un solo dato a la vez.

3.8.2. Single instruction multiple data (SIMD)

Una única instrucción, múltiples datos. Describe computadoras con múltiples unidades de procesamiento que realizan la misma operación en múltiples puntos de datos de manera simultánea. Estas explotan el paralelismo a nivel de datos, pero no concurrencia de instrucciones.

3.8.3. Multiple instruction single data (MISD)

Múltiples instrucciones, un único dato. Estas arquitecturas poseen muchas unidades funcionales que le realizan diferentes operaciones a un mismo dato. Este tipo de arquitectura es la menos común.

3.8.4. Multiple instruction multiple data (MIMD)

Múltiples instrucciones, múltiples datos. Este tipo de máquinas tienen un número de unidades de procesamiento que funcionan de manera asíncrona e independiente. En cualquier instante, diferentes procesadores pueden estar ejecutando diferentes conjuntos de instrucciones en diferentes conjuntos de datos. Esta es uno de los tipos de arquitecturas más comunes en computadoras recientes.

3.9. Ley de Amdahl

En el área de análisis de algoritmos, la ley de Amdahl establece que la mejora en rendimiento de un programa a través de aplicación de técnicas de paralelismo está limitada por la fracción de tiempo que dura la porción de código paralelizada. Posteriormente

se formaliza en una fórmula que ayuda a calcular la aceleración teórica. Esta fórmula es usualmente utilizada para obtener predicciones teóricas de la aceleración de una aplicación al utilizar técnicas de paralelización y múltiples unidades de procesamiento. Es llamada así por el científico Gene Amdahl, quien la planteó originalmente en 1967 [Amdahl, 1967].

La fórmula de la ley de Amdahl es la siguiente:

$$A = \frac{1}{(1 - F_m) + \frac{F_m}{A_m}}$$

donde A es la aceleración total obtenida, F_m es la fracción de tiempo total que se paraleliza, A_m es el factor de mejora de la parte paralelizada.

Por ejemplo, si para un programa, un algoritmo que dura el 45 % del tiempo total de ejecución se logra paralelizar de forma que ejecute 3 veces más rápido que su versión secuencial, según la ley de Amdahl, se obtendrá:

- $A_m = 3$
- $F_m = 0{,}45$
- $A \approx 1{,}429$

es decir, se mejora la velocidad del programa en un factor de alrededor de 1.429.

Esta fórmula nos proporciona un límite teórico en la aceleración de un programa dada la proporción del mismo que puede ser acelerado y la cantidad de unidades de procesamiento con las que se trabaje.

3.10. Análisis asintótico

El análisis asintótico es un método utilizado para describir el comportamiento de límite, es decir, definir funciones que describen límites entre los que se dará el comportamiento real de un programa. Este tipo de análisis es usualmente utilizado para hacer un estimado del crecimiento del tiempo o espacio que tiene un programa en términos del tamaño de la entrada. También ayuda a determinar si un programa tiene una solución en un tiempo aceptable [Murray, 1984].

Esto se hace encontrando una función que relacione la cantidad de espacio utilizado por el algoritmo o la longitud de su tiempo de ejecución con el tamaño de la entrada. Se dice que un algoritmo es eficiente si estas funciones crecen lentamente comparadas al crecimiento del tamaño de la entrada.

Como no solo el tamaño de la entrada afecta el tiempo de ejecución o la cantidad de espacio utilizado por un algoritmo, estos análisis se suelen realizar para el *mejor* caso , que intenta encontrar una función que aproxime a la función real del programa por debajo, también llamada notación Ω, para el *peor*, que intenta encontrar una función que aproxime a la función real del programa por encima, también llamada notación O y para el caso *promedio*, que intenta hacer una aproximación precisa para casos promedios, también llamada notación Θ.

Cuando se dice que un algoritmo ejecuta en tiempo $T(n)$, se refiere generalmente a la notación O, en otras palabras, el tiempo del algoritmo para el peor caso. El algoritmo podría tomar menos tiempo en ejecutar, pero su tiempo de ejecución se puede estimar de forma proporcional mediante esta métrica.

Si una función es de la forma $T(n) = c * n^3 + k * n^2$, se dice que $T(n) = O(n^3)$, siendo n el tamaño de la entrada, c y k constantes. Para la notación O se toma la cota superior que más cercanamente aproxime a la función, de manera que se dice que $T(n)$ crece de forma asintótica no más rápido que n^3, en el caso de este ejemplo.

3.11. Relojes de Lamport

El reloj de lamport es un algoritmo que se utiliza para determinar el orden de los eventos en un sistema distribuido de computadoras. Esto sirve para sincronizar procesos que necesitan seguir un orden determinado en su ejecución. Es típicamente utilizado para proveer orden a las tareas añadiendo a la vez poco *overhead*. Se llaman así por su creador Leslie Lamport [Lamport, 1978].

3.12. Representación de números en punto flotante de IEEE

La IEEE estandarizó la representación computacional binaria de los números de punto flotante en 1985. Este es el estándar IEEE 754 [iee, 1985]. Define formatos de aritmética para estos números, reglas de redondeo, operaciones, y manejo de excepciones entre otras cosas.

Un formato de punto flotante se especifica mediante una base, que en el caso del estándar IEEE 754 puede ser binaria o decimal, una precisión, y un rango de exponentes. Los números son representados por tres números que son el signo (cero o uno), un coeficiente, que no puede tener más dígitos que lo especificado por la precisión, y debe de estar en la base del formato, y un exponente.

A continuación se mencionan dos de los formatos básicos para su representación:

- Precisión simple: usualmente utilizado para representar los "float" en el lenguaje C. Este formato binario ocupa 32 bits y tiene una precisión de representación máxima de alrededor de 7 dígitos decimales.
- Doble precisión: usualmente se utiliza para representar los "double" en el lenguaje C. Este formato ocupa de 64 bits y tiene una precisión de representación de alrededor de 16 dígitos decimales.

En la siguiente sección, se describe la metodología seguida para alcanzar los objetivos propuestos en este trabajo.

Capítulo 4

Metodología

Para mejorar el rendimiento del programa PCell y cumplir con los objetivos propuestos, se realizaron las tareas que se describen en este capítulo, divididas en cuatro etapas distintas.

En la primera etapa se perfiló el rendimiento de PCell con el fin de identificar aquellas áreas del código que se podían acelerar, ya sea mejorando el código secuencial o aplicando técnicas de paralelización.

Posteriormente se implementó la versión paralela de PCell, siguiendo distintas estrategias de paralelización del código de acuerdo con lo aprendido en la etapa de análisis.

En la tercera etapa se sometió a prueba la aplicación paralela. Primero se validó la aplicación, es decir, se comprobó que para cada entrada el programa produce la salida deseada, y que además no habían errores de ejecución. Luego de esto, se corroboró que los datos que se obtuvieron de cada simulación fueron correctos y equivalentes al programa original en los dominios en los que una comparación es posible, esto es, para números de Reynolds menores o iguales a 10^3 y para mallas espaciales de 100×100 puntos como máximo.

En la cuarta y última etapa se sometió al programa implementado a pruebas de rendimiento para calcular, además de las medidas estándar de aceleración, eficiencia, trabajo y *overhead*, sus tiempos reales de respuesta (tiempo reloj, no tiempo computacional), para compararlos con los tiempos exhibidos por PCell secuencial para los mismos problemas.

4.1. Perfilado de rendimiento y mejoras al programa

En esta primera tarea se realizó un análisis del programa PCell para identificar áreas cuyo rendimiento valía la pena de ser mejorado y proponer estrategias de mejora del rendimiento.

Estas incluyeron la distribución de tareas en distintos hilos de un mismo proceso, la separación de tareas que no tengan dependencias funcionales, o la separación de datos dentro de una misma tarea, de manera que se pudieran ejecutar simultáneamente las instrucciones sobre dos o más datos.

Para esto se utilizó la herramienta gprof de GNU [Graham et al., 1982]. El perfilado reveló que la función *convect* es la que tardaba más tiempo del total de la ejecución del programa, y que las demás funciones representaban una pequeña parte de ese total. La salida del perfilador se puede ver en la Figura 4.1.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
91.94	0.34	0.34	100	3.40	3.60	convect
5.41	0.36	0.02	100	0.20	0.20	mean_field
2.70	0.37	0.01	1	10.00	10.00	potinc
0.00	0.37	0.00	100	0.00	0.00	itoa
0.00	0.37	0.00	100	0.00	0.00	reverse
0.00	0.37	0.00	1	0.00	360.17	convec_plasma
0.00	0.37	0.00	1	0.00	0.00	create_form_Convection
0.00	0.37	0.00	1	0.00	0.00	create_gnuplot

Figura 4.1: Resultados obtenidos de la ejecución de gprof sobre el programa original PCell antes de hacerle las mejoras y la paralelización

Una vez que se obtuvo esta información, se determinó que la mayor parte de los esfuerzos de paralelización y mejora de la eficiencia se tenían que enfocar en esta función. Seguido de esto, se hizo un proceso de revisión de las funciones del programa, haciendo énfasis en *convect*.

Al hacer la revisión, se determinó que algunas de las funciones en el programa original eran redundantes o innecesarias para el programa, por lo que se hizo un proceso de revisión y optimización de las mismas. Entre las funciones que se eliminaron, están

itoa y *reverse*, cuyas tareas eran innecesarias dadas las correcciones del programa, pues estas funciones se encargaban de dar formato a un conjunto de *strings* que ultimadamente se utilizaban para imprimir datos de salida en consola o archivos. Las tareas de estas funciones se optimizaron y se integraron con la función *convect* de manera que la impresión en consola y la escritura en archivos de los datos de salida fueran reducidos al mínimo necesario.

Algunas de las funciones de *mean_field* fueron también integradas con los cálculos de la función *convect*. La mayoría fueron eliminadas completamente debido a que eran innecesarias para la ejecución del programa. La función *potinc* se mantuvo, pero se optimizó para que no hiciera operaciones innecesarias, pues algunas de las tareas que hacía, era repetitivas.

Adicionalmente, se sustituyeron los arreglos estáticos que estaban en el código original. Esto con el fin de poder aumentar la escala espacial de la simulación, pues originalmente los arreglos tenían siempre un tamaño fijo, entonces no se podía variar el tamaño de la matriz utilizada para simular los puntos espaciales. Esto a pesar de que la interfaz del programa original indicaba que se podía cambiar el tamaño, pero en el fondo la matriz siempre tenía un tamaño de 100×100 puntos.

Al cambiarse estos arreglos a memoria dinámica, se puede ahora indicar al inicio del programa el tamaño que esta matriz debe de tener para la simulación. Como casi todo en el programa estaba *alambrado* para funcionar con la matriz de tamaño fijo, fue necesaria una reescritura completa de las funciones *potinc* y *convect*. Lo único que se mantuvo exactamente igual con el programa original fue el cálculo de la función de inducción en los ciclos más internos de la función *convect*.

4.2. Implementación paralela

Una vez que se generó el perfil de rendimiento, se procedió a implementar la versión paralela de PCell, aplicando las estrategias de paralelismo propuestas en la etapa anterior.

En el proceso de paralelización se utilizó OpenMP para dividir tareas localmente entre hilos, esto ayudó a explotar la arquitectura MIMD (según la taxonomía de Flynn) que tiene el cluster donde se probó el modelo. Para el proceso de implementación se

siguieron una serie de pasos que se describen a continuación y que están basados en algunos de los patrones descritos en [McCool et al., 2012].

El primer paso consistió en analizar cada una de las áreas del código identificadas en la etapa anterior. En primer lugar se analizó la posible separación de tareas entre los hilos, el producto de esto fue una lista de las tareas separables que podían ser paralelizadas. En segundo lugar, se hizo un análisis de las dependencias funcionales entre estas tareas. Finalmente, se determinan los requerimientos de comunicación entre las unidades de procesamiento del código paralelizado que ejecutan las subtareas.

El paso siguiente consistió en implementar el programa y escoger el estándar de paralelización adecuado para hacer la separación de las tareas. El análisis determinó que la mejor estrategia de paralelización era una a nivel de hilos, las escalas de cantidad de datos con las que trabaja el programa son grandes, pero no lo suficiente para que tenga sentido escalar a arquitecturas distribuidas.

La programación se hizo en el lenguaje C, debido a su capacidad para realizar cálculos precisos, y su fácil integración con los estándares y distintas bibliotecas que implementan OpenMP. Además de que el programa original está escrito en este lenguaje. Adicionalmente, la experiencia que se tiene desarrollando aplicaciones para arquitecturas paralelas usando el lenguaje C fue una ventaja en el desarrollo de este proyecto, pues no fue necesario invertir tiempo en conocer el lenguaje ni la manera en la que este se integra con OpenMP.

4.3. Análisis de ParaPCell

En esta sección se describe el análisis del desempeño de ParaPCell, que es el nombre que se le dio a la implementación paralela. Este análisis será necesario para comprobar la hipótesis de investigación.

4.3.1. Recolección de datos relevantes

Las evaluaciones experimentales sirvieron para recolectar datos relacionados con los tiempos de ejecución, específicamente el tiempo real transcurrido (*walltime*) y la precisión de los datos de salida producidos por ParaPCell. Estos últimos son:

- La forma y evolución del campo magnético. Esta se describe en función del potencial vectorial, cuyos valores son dados en *teslas*.
- La formación de torbellinos en el plasma. Esta es representada por la cantidad de torbellinos que se forman dentro de la grilla espacial.
- Densidad de energía magnética. Es el total de energía almacenada en el plasma por unidad de área, dado en *teslas*.

En síntesis, se ejecutaron las pruebas descritas en el cluster para obtener mediciones del tiempo total de ejecución del programa paralelo. Con esos datos se obtendrán medidas de la aceleración del programa paralelo con respecto al tiempo de ejecución con una sola unidad de procesamiento.

4.3.2. Análisis de varianza

En esta etapa de la investigación se realizó un análisis de varianza de los resultados de las pruebas para determinar si en la media los datos obtenidos muestran un comportamiento similar, o si por el contrario hay diferencias significativas que deban ser consideradas, por ejemplo, para verificar que no existen errores ni en la implementación paralela de PCell, ni en la preparación o ejecución de las pruebas que puedan ser la causa de esas diferencias.

Una vez que se comprobó que no hubo diferencias significativas, se hizo la evaluación de la precisión de los datos producidos por ParaPCell.

4.4. Evaluación de la precisión

Además de los tiempos de ejecución, se realizaron también pruebas de comparación de los resultados generados por PCell y por la versión paralela. El propósito de estas pruebas es ayudar a verificar que la calidad (precisión) de los resultados producidos por la versión paralela no haya disminuido en comparación con la versión secuencial.

La evaluación se hizo comparando las salidas de ambos programas, tomando como base el resultado del programa original (secuencial) calculando una métrica de error que será función de la diferencia entre los datos generados por la versión paralela y la

versión secuencial. Para conservar la validez del programa, el error debe de ser pequeño, de manera que no afecte el resultado final de la simulación. Este valor significativo de tolerancia al error se determina comprobando que los resultados de la simulación son lo suficientemente similares, es decir, la salida generada por ambos programas que consiste en un conjunto de valores obtenidos a partir del cálculo de la ecuación de inducción en cada punto de la matriz espacial para cada iteración de tiempo debe de tener cierta similitud. Para determinar esto se contó con la validación por parte de uno de los autores del programa original, quien determinó que la diferencia entre las salidas de los programas no podía ser mayor a un 5 % para poder decir que los dos modelos son equivalentes.

4.5. Evaluación del rendimiento

La evaluación del rendimiento se llevó a cabo midiendo los tiempos totales de ejecución del programa con diferentes tamaños de problema. Esta medición se hizo empíricamente con respecto al tiempo real transcurrido (*walltime*) que tiene cada implementación. El objetivo es disminuir el tiempo de ejecución del programa, por lo que no fue necesario medir el tiempo de ejecución de cada hilo individualmente, sino el tiempo de ejecución total del programa.

Una vez obtenidos los datos de estas mediciones, se calculó la aceleración, la eficiencia, el trabajo y el *overhead* de la implementación paralela. A continuación se describen las fórmulas que se utilizaron para obtener estas métricas de rendimiento.

Aceleración

La aceleración S es una relación entre el tiempo secuencial T_s y el tiempo paralelo T_p:

$$S = T_s/T_p \tag{4.1}$$

Eficiencia

La eficiencia E es una medida del uso de los recursos. Es una relación entre la aceleración S y la cantidad p de procesadores con los que se ejecutó la tarea:

$$E = S/p = T_s/pT_p \tag{4.2}$$

Trabajo

El trabajo W es el tiempo total de ejecución. Para la aplicación secuencial este es $W_s = T_s$, y para la paralela:

$$W_p = \sum_{i=1}^{p} W_i = pT_p \tag{4.3}$$

Overhead

El *overhead* T_o es el trabajo paralelo que no forma parte del secuencial:

$$T_o = pT_p - T_s \tag{4.4}$$

En la siguiente sección, se discutirá más a fondo el proceso que se siguió para paralelizar el código. También se hace un análisis del programa original y se elabora más sobre el proceso de mejora del código.

Capítulo 5

Paralelización de PCell

Después de hacer el análisis del código y de determinar las áreas donde la paralelización sería más provechosa, se incluyeron las directivas de OpenMP. A continuación se describen los pasos que se siguieron.

Inicialmente se ejecutó el profiler *gprof* en el código original de PCell, con este resultado se descubrió que el cálculo de la función de convección consumía un 91.94 % del tiempo total de ejecución del programa. Uno de los ciclos más grandes dentro de esta función es el que calcula la ecuación de inducción (3.2), debido a que para cada punto de la matriz espacial el programa debe de iterar cuatro veces.

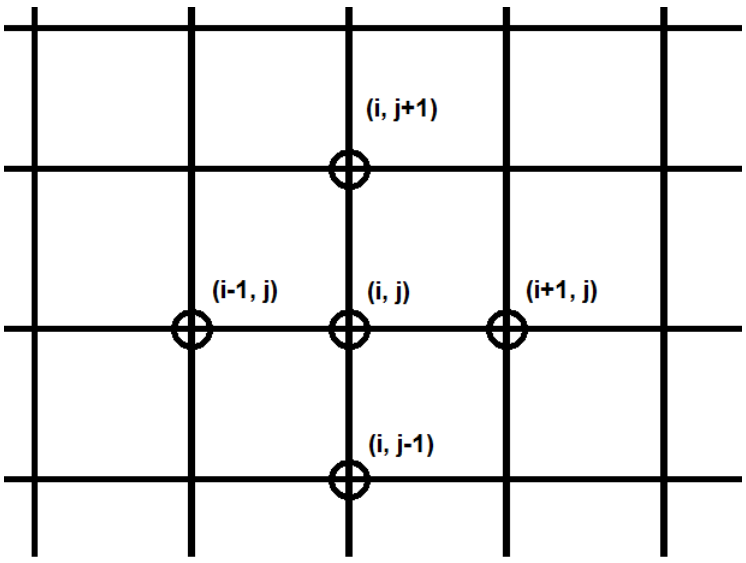

Figura 5.1: Grilla espacial para la simulación

En cada paso temporal de la simulación, se define una matriz de puntos espaciales. La ecuación de inducción se calcula para cada uno de estos puntos basado en el estado del plasma en la iteración temporal anterior, excepto para la primera iteración para la cual se calculan valores de un estado inicial de plasma con un campo magnético que muestra líneas paralelas verticales. Como cada estado depende del anterior, esto significa que existe dependencia temporal en estos cálculos. Después de hacer una revisión cuidadosa de los ciclos que recorren la matriz espacial, se determinó que no existe dependencia entre los cálculos a nivel espacial, es decir, los cálculos espaciales se pueden hacer en paralelo, pues no dependen de otros cálculos aparte de los de la iteración temporal anterior para hacerse.

La Figura 5.1 ilustra la forma en la que se hace el cálculo de la función de inducción para cada punto espacial. En cada uno de estos puntos es necesario tomar en cuenta los cuatro puntos a su alrededor. El análisis del código ayudó a descubrir que no existe dependencia funcional ni de datos entre cada cálculo individual por punto, pues los datos que se utilizan se toman de la iteración anterior. Este análisis ayudó a determinar que los ciclos internos de la función *convect* son embarazosamente paralelizables, lo que quiere decir que no existe dependencia de datos o dependencia funcional entre las tareas que se pueden separar, y que la comunicación necesaria entre las unidades de procesamiento que las resuelven es poca o nula [Foster, 1995].

Antes de paralelizar el programa, se mejoraron varios aspectos del código original. Se removieron algunos ciclos y variables redundantes o innecesarias. PCell fue escrito originalmente en Fortran y traducido con f2c [Feldman, 1990], que es un programa que traduce de forma automática código de Fortran a C. Esta traducción automática muchas veces introduce errores que posteriormente deben de ser corregidos en revisiones hechas por un programador. En el caso de PCell, algunas de las secciones de código que debieron ser arregladas, parecían haber sido afectadas por este proceso. Después de estas mejoras iniciales al programa secuencial, se integró con OpenMP.

A continuación se muestra el pseudocódigo de la función *convect*, que es la parte principal del programa que se paralelizó.

Algoritmo 1 función convect(Tlim,Xlim,Ylim) → a

precondición: reyn: Número de Reynolds magnético

precondición: m: parámetro de velocidad de campo

precondición: Tlim: parámetro: dimensión temporal

precondición: Xlim: parámetro: dimensión espacial en x

precondición: Ylim: parámetro: dimensión espacial en y

precondición: a: arreglo tridimensional que tiene los valores de los cálculos anteriores y actuales

precondición: vx: cálculo de la función de movimiento del flujo del plasma para la dimensión x, arreglo

precondición: vy: cálculo de la función de movimiento del flujo del plasma para la dimensión y, arreglo

```
    potinc()
 2: dx := 1/Xlim
    dy := 1/Ylim
 4: dt := 1/Tlim
    re := 1/reyn
 6: ex := dt/dx*12
    ye := dt/dy*12
 8: l := re*dt/(dx*dx)
    m := re*dt/(dy*dy)
10: para t := 1 to Tlim :
        para n := 0 to 4 :
12:         para x := 0 to Xlim :
                para y := 0 to Ylim :
14:                 fa := 0
                    para can := 1 to 2 :
16:                     si can == 1 entonces
                            s := -1
18:                         z := 1
                        si no
20:                         s := 1
                            z := 0
22:                     fin si
```

Algoritmo 1 función convect (continuada)

$ca := -s * ex * vx[x+s][y] * (a[n+1][x+s][y] * 8 - a[n+z][x-2][y]) +$
$(a[n+1][x+s][y+2] - a[n+1][x+s][y-2]) * (vx[x+s][y+2] - vx[x+s][y-2])/8$

24:

$cb := -s * ye * vy[x][y+s] * (a[n+1][x][y+s] * 8 - a[n+z][x][y-2]) +$
$(a[n+1][x+2][y+s] - a[n+1][x-2][y+s]) * (vy[x+2][y+s] - vy[x-2][y+s])/8$

$fa := fa + ca + cb$

26:

$cc := l * (a[n+1][x+1][y] + a[n+1][x-1][y]) +$
$m * (a[n+1][x][y+1] + a[n+1][x][y-1])$

$cd := 1 - (ex * vx[x+s][y] + ye * vy[x][y+s]) + s * (l + m)$

28: $fe := fa + cc$

$fi := fe + a[n][x][y] * cd$

30: $a[n+2][x][y] := fi/cd$

fin para

32: **fin para**

fin para

34: **fin para**

fin para

Como se puede observar en el Algoritmo 1, la parte principal del programa consiste de cuatro ciclos. Lo primero que hace la función es llamar a la función *potinc*. Esta función define las condiciones iniciales de la matriz, como cada iteración requiere de cálculos hechos en la iteración anterior, se debe de definir un estado inicial de la matriz que no depende de ninguno anterior. Los dos ciclos más internos de la función (que son los que se encargan de iterar sobre la matriz que representa las dimensiones espaciales) realizan cálculos que son completamente independientes de los demás en el mismo instante de tiempo. En la sección más interna del código se calculan los componentes parciales de la función llamados *ca, cb, cc y cd*. En la variable *ca* (línea 23 en Algoritmo 1) se calcula el componente en x, mientras que en *cb* (línea 24 en Algoritmo 1) se calcula el componente en y. Ambos componentes se calculan con respecto a los puntos

que tienen a su alrededor y con respecto al valor de la función de flujo de movimiento del plasma en esos puntos espaciales a partir de los valores en los arreglos *vx y vy*. *cc y cd* son variables intermedias del cálculo final en las que se toma en cuenta las posiciones espaciales alrededor del punto que se está calculando en esa iteración.

Se dividió la matriz espacial entre las unidades de procesamiento disponibles de forma que los cálculos se pudieran realizar de forma simultánea. Para estos ciclos utilizamos paralelización a nivel de hilos con OpenMP. Se utilizaron directivas para el compilador de forma que la carga de trabajo de iteración sobre esta matriz se dividiera entre los hilos disponibles.

Los ciclos se paralelizaron con la directiva *parallel for*. La mayoría de las variables utilizadas en esta parte del programa debieron ser declaradas como privadas en el bloque paralelizado para darle a cada hilo su propia copia, ya que la mayoría de estas son utilizadas para almacenar valores intermedios para el cálculo final de la ecuación de inducción en cada punto de la matriz.

En la Figura 5.2 se puede observar el diagrama de flujo del programa PCell original. En él se muestra la estructura principal de la ejecución del programa. Se desarrolla únicamente la sección del código que fue paralelizada, que es la que se determinó mediante el *profiler* que es la que consume más tiempo de ejecución.

En la Figura 5.3 se observa el diagrama de flujo de ParaPCell. En este segundo diagrama los puntos (...) indican que hay un número no específico de hilos que se generan idénticos a los que tienen a su lado. Se puede observar que los ciclos internos que recorren la matriz espacial fueron paralelizados, de forma que cuando se ejecuta esa sección del código, OpenMP genera una cantidad de hilos para ejecutarla concurrentemente separando el rango de los valores que toman los iteradores de la matriz, de esta forma se pueden separar los índices, por lo que cada uno de los hilos tendrá una sub-matriz para la que calculará la ecuación de inducción magnética en cada uno de sus puntos. Los hilos se van a encargar de hacer los cálculos de forma independiente para diferentes partes de la matriz. Cada hilo va a tener su variable local para recorrer la matriz, que iniciará en un valor distinto para cada uno, así mismo con el valor del límite hasta donde debe de llegar el ciclo local.

Como se puede observar, el cálculo de la función de inducción magnética para cada punto espacial fue el que se paralelizó. Este cálculo se puede ver en más detalle en el

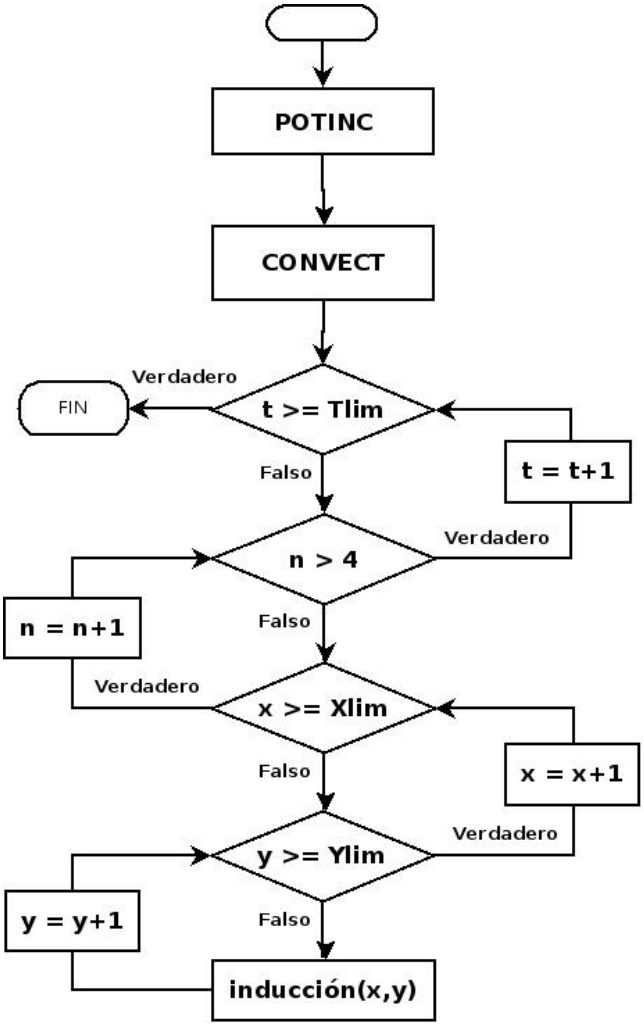

Figura 5.2: Diagrama de flujo de PCell

Algoritmo 1. Estas operaciones se pueden dividir debido a que no existen dependencias funcionales ni de datos entre ellas, es decir, un cálculo no depende de ningún otro dentro de la misma iteración temporal.

Una vez que se realizó el proceso de paralelización, fue necesario hacer evaluaciones para determinar no solo que el programa paralelo es más eficiente que el secuencial, sino también que las salidas del nuevo programa son equivalentes al programa original. En la próxima sección se describe esta evaluación.

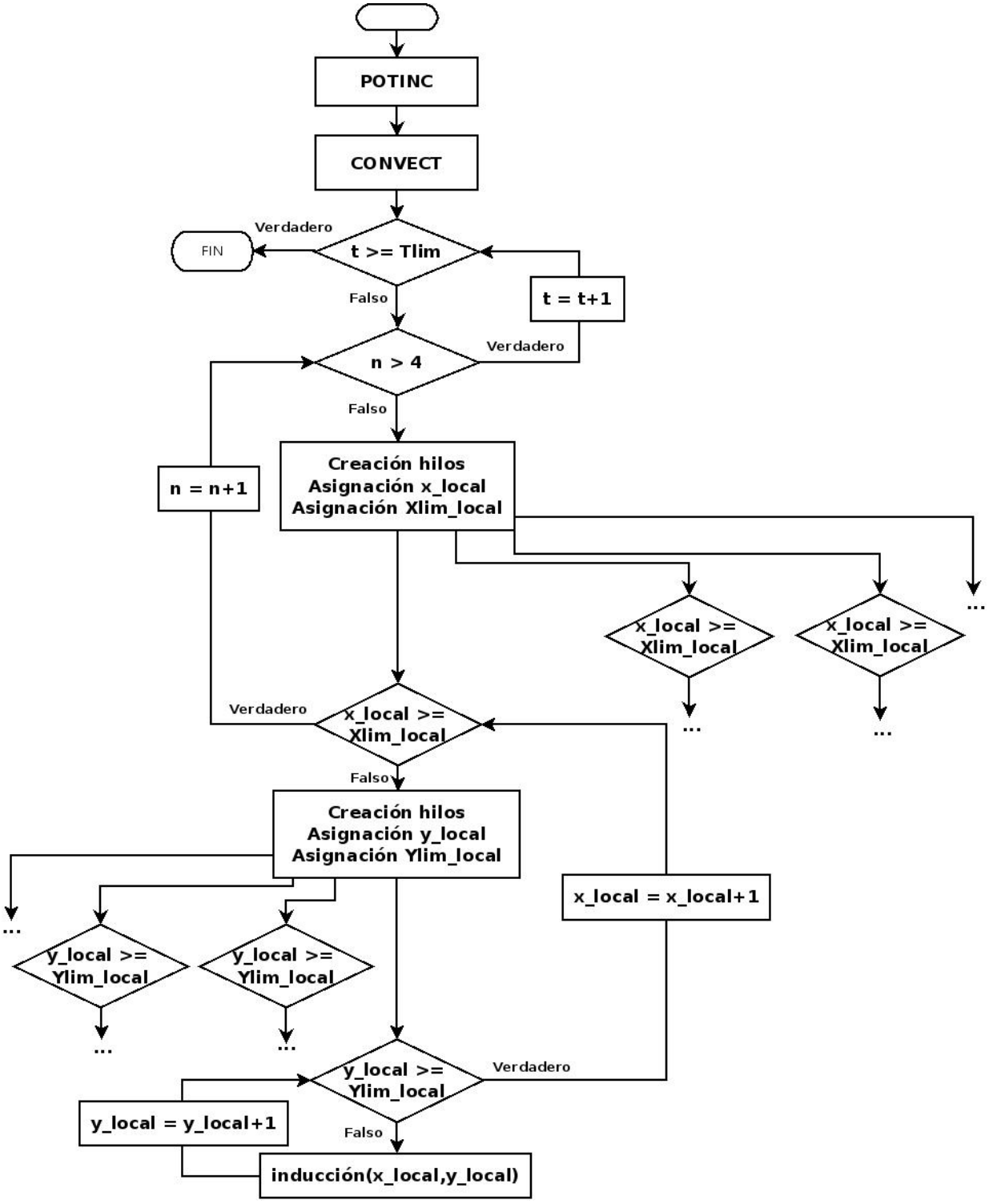

Figura 5.3: Diagrama de flujo de ParaPCell

Capítulo 6

Diseño y Ejecución de la Evaluación Experimental

En esta sección se detalla la evaluación experimental realizada con ParaPCell para medir su mejora en la eficiencia, y el aumento en las dimensiones espaciales y temporal.

6.1. Descripción de las variables utilizadas

Tres factores deben considerarse en las pruebas, dos de ellos asociados a variables del modelo de simulación, y uno a la capacidad de procesamiento.

Los dos factores del modelo son el tamaño de la grilla espacial, y el número de pasos temporales que serán simulados. El factor de capacidad de procesamiento es la cantidad de unidades de procesamiento (hilos) utilizados en la ejecución del programa.

6.1.1. El tamaño de la grilla espacial

El programa PCell tiene un rendimiento aceptable para grillas de tamaño 100×100 [Frutos-Alfaro,]. Para tamaños mayores de la grilla, dado algunos errores en la programación, la ejecución se encicla y no termina.

Una grilla de tamaño 100×100, limita la capacidad de visualización de fenómenos en el plasma. Por esta razón, el poder producir esos datos para grillas más grandes representaría una mejora considerable, que contribuiría a comprobar una parte de la

hipótesis de esta investigación: que es posible mejorar el rendimiento de PCell para abarcar soluciones a problemas del mayor granularidad *espacial*.

6.1.2. Cantidad de pasos temporales

Uno de los parámetros que recibe PCell antes de ejecutar las simulaciones es la cantidad de pasos temporales que debe de durar. Esto afecta la visualización final, pues entre más pasos temporales, más larga será la simulación. También afecta la cantidad de cálculos que se deben de hacer, pues en cada iteración temporal, se debe de calcular el valor de la ecuación de inducción para cada punto de la matriz.

6.1.3. Cantidad de procesadores

Se cuenta con acceso a un cluster para realizar la evaluación experimental con ParaPCell. El cluster está ubicado en el Colaboratorio Nacional de Computación Avanzada (CNCA), del Centro Nacional de Alta Tecnología (CeNAT).

El cluster del CNCA se llama Kabré, y tiene cinco blades, cada uno con cuatro nodos Intel Xeon Phi KNL de 64 núcleos, para un total de 20 procesadores y 1280 núcleos.

6.2. Diseño de la evaluación experimental

En esta sección se detalla el diseño de las pruebas realizadas. Una prueba corresponde a la ejecución del programa ParaPCell, con un valor (nivel) específico para cada una de las variables (factores).

Se realizaron tres distintas evaluaciones experimentales para obtener datos de mejoras relevantes que se le hicieron al programa. Las pruebas están diseñadas para medir tres de las propiedades que más interesan para la mejora del programa PCell y que sean de mayor provecho para los usuarios finales del mismo. Las tres propiedades que son medidas en las pruebas son aceleración, escalabilidad espacial y escalabilidad temporal.

6.2.1. Aceleración

La aceleración es importante para determinar la mejora en el rendimiento y la disminución en el tiempo de respuesta del programa PCell, sobre todo cuando se utilizan

tamaños de problema más grandes que los que manejaba el programa original. Para esta evaluación se decidió variar únicamente la cantidad de unidades de procesamiento que se utilizan para hacer la simulación. Al dejar esta como variable independiente, se puede obtener una métrica de qué tanto se reduce el tiempo total de ejecución del programa al añadir unidades de procesamiento de forma gradual, lo que nos permite medir para cada incremento la aceleración que alcanza el programa con respecto a su versión secuencial, esto nos permite ver qué tanto el nuevo programa paralelo aprovecha los recursos computacionales con los que se ejecuta.

El programa fue probado en un nodo del cluster con 1, 2, 4, 8, 16, 32, 64 y 128 hilos. Cada una de estas ejecuciones fue hecha para una matriz espacial de tamaño 500×500 y 1000 iteraciones temporales para cada ejecución.

Para medir la ganancia de rendimiento, se tomaron mediciones del tiempo total de ejecución (walltime) para cada ejecución. Cada una de estas mediciones se promediaron utilizando la media aritmética. Con este valor, se pueden obtener una medida de aceleración dividiendo el tiempo secuencial entre el tiempo paralelo, con esta medida, se pueden obtener las medidas de eficiencia y trabajo.

Factor 1: Cantidad de unidades de procesamiento utilizadas (p)

- Valor inicial: $p = 1$.
- Incremento y valores finales: $2, 4, 8, 16, 32, 64y128$.
- Niveles: 8

6.2.2. Escalabilidad Espacial

La escalabilidad espacial es importante para determinar la capacidad del programa de computar problemas con una mayor granularidad y detalle a nivel espacial, el programa PCell original no permitía incrementar la granularidad de la matriz espacial debido a algunas fallas en el código.

En esta prueba se variaron los tamaños de las dimensiones de la matriz espacial que es simulada, como la simulación se hace sobre una superficie de dos dimensiones, se trabajó variando el tamaño de la dimensión en X y la dimensión en Y. Las pruebas se realizaron a partir de valores de 100×100 hasta 1000×1000, con incrementos graduales

de 100 para cada dimensión. Debido a que para estas pruebas se aumentan gradualmente ambas dimensiones al mismo tiempo, el tamaño del problema se incrementa de forma cuadrática, por lo que si la aceleración del programa es lineal, se debe de observar un crecimiento cuadrático en el tiempo total de ejecución conforme se aumenta el tamaño de la entrada. Esta prueba se hizo con 64 hilos y 1000 iteraciones temporales para cada ejecución. Para medir la escalabilidad espacial se tomó el promedio (media aritmética) del tiempo total de ejecución para cada corrida.

Factor 2: Tamaño de la grilla espacial (Ge)

- Valor inicial: $Ge = 100 \times 100$
- Incremento: $\Delta Ge = 100 \times 100$
- Valor final: 1000×1000.
- Niveles: 10

6.2.3. Escalabilidad Temporal

La importancia de la escalabilidad temporal yace en la capacidad de hacer simulaciones que abarquen una mayor cantidad de tiempo, lo que permite estudiar mejor la evolución y comportamiento del campo magnético en escalas temporales que son provechosas para investigadores del área. El programa PCell original tenía serias limitaciones en este aspecto, pues incrementar un poco la cantidad de iteraciones temporales de la simulación generaba aumentos significativos en el tiempo de ejecución del programa.

Esta evaluación se hizo para cantidades de iteraciones entre 1000 y 5000 con incrementos de 500. Se utilizaron 64 hilos y un tamaño de matriz espacial de 1000×1000. Para medir la escalabilidad temporal se tomó el tiempo total de ejecución de cada una de las corridas.

Factor 3: Número de pasos temporales (R)

- Valor inicial: $t = 1000$
- Incremento: $\Delta t = 500$
- Valor final: 5000
- Niveles: 9

Cada una de las pruebas se ejecutó 20 veces para cada uno de los valores de las variables con el fin de obtener resultados estadísticamente significativos.

6.3. Configuración del equipo utilizado

Todas las pruebas se realizaron en la supercomputadora Kabré del Centro Nacional de Alta Tecnología. Las especificaciones de Kabré se detallan a continuación.

Kabré cuenta con nodos con procesadores Xeon Phi KNL, estos procesadores son altamente configurables. Tienen 64 núcleos @ 1.3 GHz y 96 GB de RAM cada uno. Cada núcleo tiene una cache L1 y cada par de núcleos se organiza en *celdas* con un nivel de cache L2 simétricamente compartido entre ellos. Todas las caches L2 que comparten los procesadores de una celda se conectan a las otras caches L2 espacialmente adyacentes, de manera que se forma una matriz donde cada enlace vertical u horizontal entre dos memorias es una conexión bidireccional.

Para mantener coherencia de caches, el procesador KNL tiene un directorio distribuido de etiquetas (DTD por sus siglas en inglés *distributed tag directory*), que se organiza como un conjunto de directorios de etiquetas por celda. Cada uno identifica el estado y localización de cualquier línea de cache. Se utiliza una función *hash* para identificar el directrio responsable por una dirección de memoria.

Cuando un programa hace una solicitud a memoria, la celda con el núcleo que generó la solicitud primero revisará la cache local para ver si los datos requeridos están ahí, si no lo está, la celda hará la consulta en el DTD por la línea de cache que contiene los datos. El "modo de agrupamiento de cache"se refiere a la manera en la que los mensajes de consulta de datos viajan desde una celda a las demás.

Los KNL en Kabré se configuraron en modo de agrupamiento de *cuadrante*. En este modo las celdas se dividen en cuatro partes iguales llamadas cuadrantes, que son espacialmente locales a cuatro grupos de controladores de memoria. Se garantiza que las direcciones de memoria atendidas por un controlador de memoria en un cuadrante se mapean únicamente a directorios contenidos dentro de ese cuadrante.

ParaPCell fue compilado en Kabré utilizando el compilador GCC versión 4.8.5 y la versión 3.1 de OpenMP. No se utilizaron optimizaciones automáticas del compilador.

En el capítulo a continuación se expondrán los resultados obtenidos de pruebas de equivalencia de los resultados entre los dos modelos. Además de mostrar la aceleración y escalabilidad del nuevo programa.

Capítulo 7

Resultados

En este capítulo se muestran los resultados de las ejecuciones del modelo paralelo y sus resultados con métricas de aceleración (*speedup*), eficiencia, trabajo y *overhead*. De la misma forma se muestran los resultados de escalabilidad tanto temporal como espacial.

7.1. Evaluación de la equivalencia de los resultados

Con el fin de determinar que las salidas de los programas no se distancian significativamente, se realizaron las siguientes pruebas. Se calculó el índice de correlación entre las salidas de los resultados encontrando una asociación estadísticamente significativa de 0.99; $r_{(10199)} = 0{,}99$, $p < {,}001$. Adicionalmente se realizó una prueba no paramétrica de diferencia de medias (Mann-Whitney U-Test) para grupos no pareados, se utilizó como parámetro alfa = 0,001. En esta prueba no se encontró que hubiera diferencias significativas entre los datos de salida del programa original PCell ($M = 9{,}50$, $SD = 2{,}92$) y la versión paralela ParaPCell ($M = 9{,}50$, $SD = 2{,}92$); $U_{(20400)} = 5{,}20$, $p = {,}92$. Esto indica que la prueba no es significativa, con lo que no podemos rechazar la hipótesis nula, y por ende debemos asumir que las medias son equivalentes estadísticamente.

En el Cuadro 7.1 se pueden observar los estadísticos descriptivos de los conjuntos de datos.

Cuadro 7.1: Análisis de los estadísticos descriptivos de los conjuntos de datos de salida para PCell y ParaPCell

Descriptivos	**PCell**	**ParaPCell**
N	10201	10201
Media	9.50e+7	9.50e+7
Error estd. media	28868	28869
Mediana	9.50e+7	9.50e+7
Suma	9.69e+11	9.69e+11
Desviación estándar	2.92e+6	2.92e+6
Varianza	8.50e+12	8.50e+12
Rango	1.01e+7	1.00e+7
Mínimo	9.00e+7	9.00e+7
Máximo	1.00e+8	1.00e+8
Kurtosis	-1.20	-1.20
Error estd. de kurtosis	0.0485	0.0485

7.2. Prueba de variabilidad de los resultados

Para poder determinar que los datos de las pruebas son relevantes se realizó un análisis de variabilidad con el fin de determinar que no existen valores extremos. Esto nos ayuda a determinar que la aceleración que se logró en el programa tiene un comportamiento confiable, de manera que cuando se utilice para hacer simulaciones en un ambiente real se tenga un estimado del comportamiento de la varianza del tiempo total de ejecución del programa.

La mayoría de los análisis de residuos estandarizados muestran que las mediciones no se separan significativamente entre sí, tal y como se observa en la Figura 7.1.

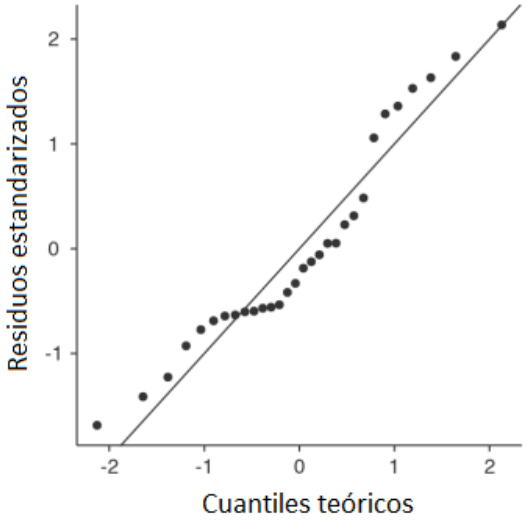

Figura 7.1: Análisis de residuos estandarizados para prueba de escalabilidad con tamaño de grilla 800×800

Solamente se presentaron valores extremos en dos de las pruebas: En la prueba de escalabilidad espacial para grilla de tamaño 600×600 y en la prueba de escalabilidad temporal con 5000 iteraciones temporales. En los dos casos cada prueba presentó únicamente un valor por encima de 3 desviaciones estándar.

Para el caso del valor extremo en la prueba de escalabilidad espacial, el *outlier* tiene un valor de 56,709 segundos y el valor promedio de esta prueba fue de 53,955 segundos, lo que quiere decir que la diferencia entre el promedio y el valor extremo es de menos de 3 segundos en el tiempo total de ejecución. La Figura 7.2 muestra los datos de la prueba de escalabilidad.

En el caso de la prueba temporal, el valor extremo tiene un valor de 10,58 segundos, y el valor promedio de esta prueba fue de 9,527 segundos. Aunque estadísticamente este valor presenta un comportamiento de *outlier* con respecto a los demás, la diferencia no es significativa dado el contexto de la prueba, pues la diferencia entre el *outlier* y el promedio es de alrededor de un segundo. La Figura 7.3 muestra los datos para esta prueba.

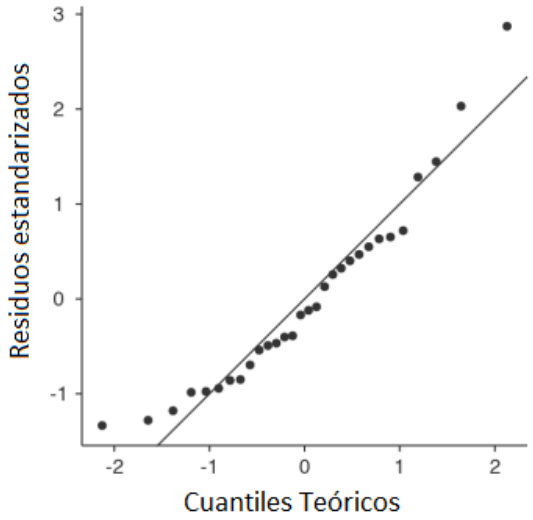

Figura 7.2: Análisis de residuos estandarizados para prueba de escalabilidad espacial con tamaño de grilla 600×600

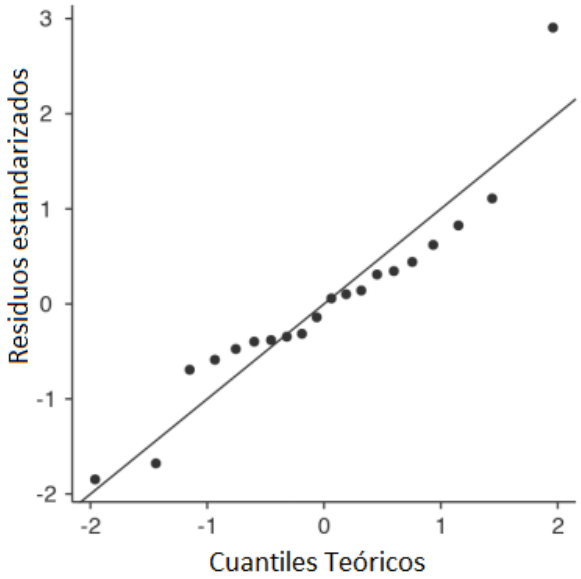

Figura 7.3: Análisis de residuos estandarizados para prueba de escalabilidad temporal con cantidad de iteraciones = 1000

7.3. Pruebas de Rendimiento

Los datos obtenidos de esta prueba muestran una escalabilidad casi lineal con respecto a la cantidad de unidades de procesamiento con las que se ejecuta el programa. Como se puede observar en la Figura 7.4, el programa escala de forma casi lineal hasta llegar a los 128 hilos. Esto se debe a que los procesadores KNL cuentan con 64 núcleos físicos. Cuando se utilizan 128 hilos se obtiene un poco más de aceleramiento, pero el crecimiento no presenta un comportamiento lineal.

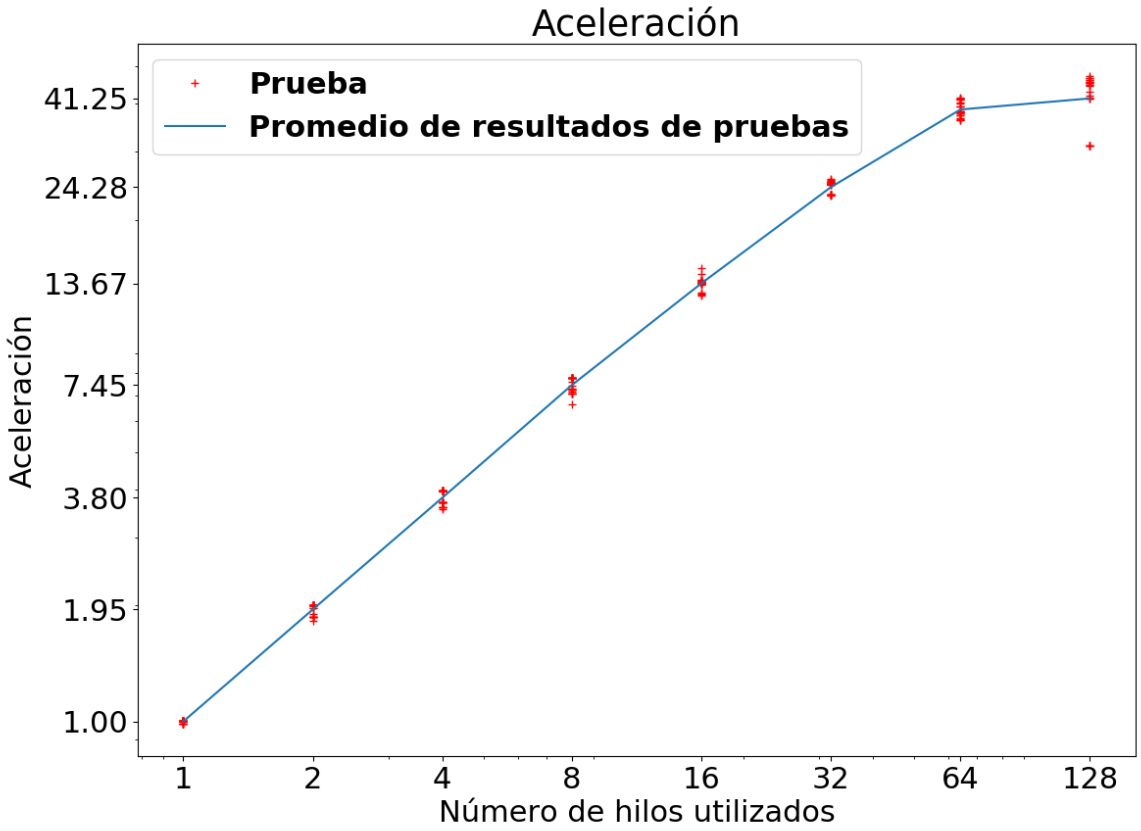

Figura 7.4: Aceleración del programa con distintas cantidades de hilos. Escala logarítmica

Se puede observar que la máxima aceleración obtenida está cercana a 42×. Esto nos muestra que nuestro método escala de forma eficiente con respecto a la cantidad

de unidades de procesamiento con las que se ejecute. Esta aceleración nos muestra una clara mejora en el rendimiento del modelo PCell, donde el mismo problema se resuelve en un tiempo mucho menor y mucho más aceptable, incluso con escalas mayores que con las que podía manejar el programa PCell original.

La Figura 7.6 muestra la eficiencia alcanzada en el programa paralelo. Se observa una clara reducción de la eficiencia para el caso de 128 hilos debido a que los Xeon Phi tienen solo 64 cores físicos. A pesar de tener multithreading, esto reduce significativamente la eficiencia.

En las Figuras 7.5 y 7.7 se puede observar el resultado de las mediciones de trabajo y overhead respectivamente. Estas medidas presentan un comportamiento similar, pues ambas dependen de la aceleración ganada con respecto a la cantidad de procesadores utilizados.

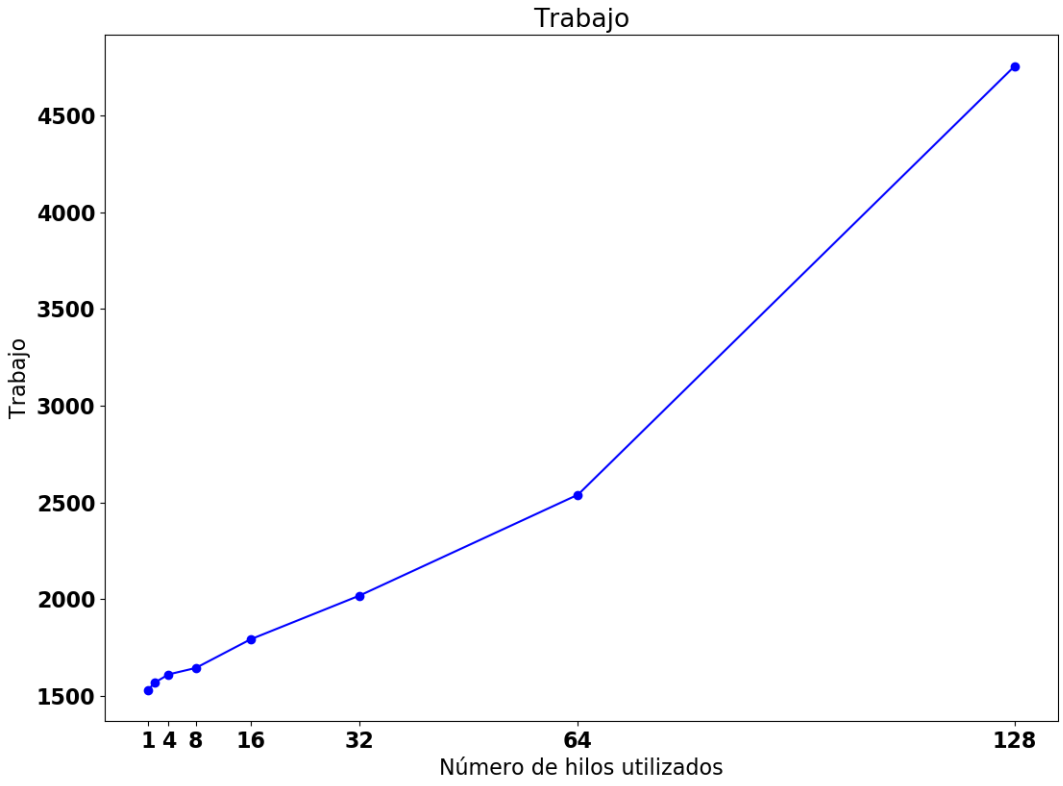

Figura 7.5: Trabajo obtenido para el programa paralelo con distintas cantidades de hilos. Escala lineal

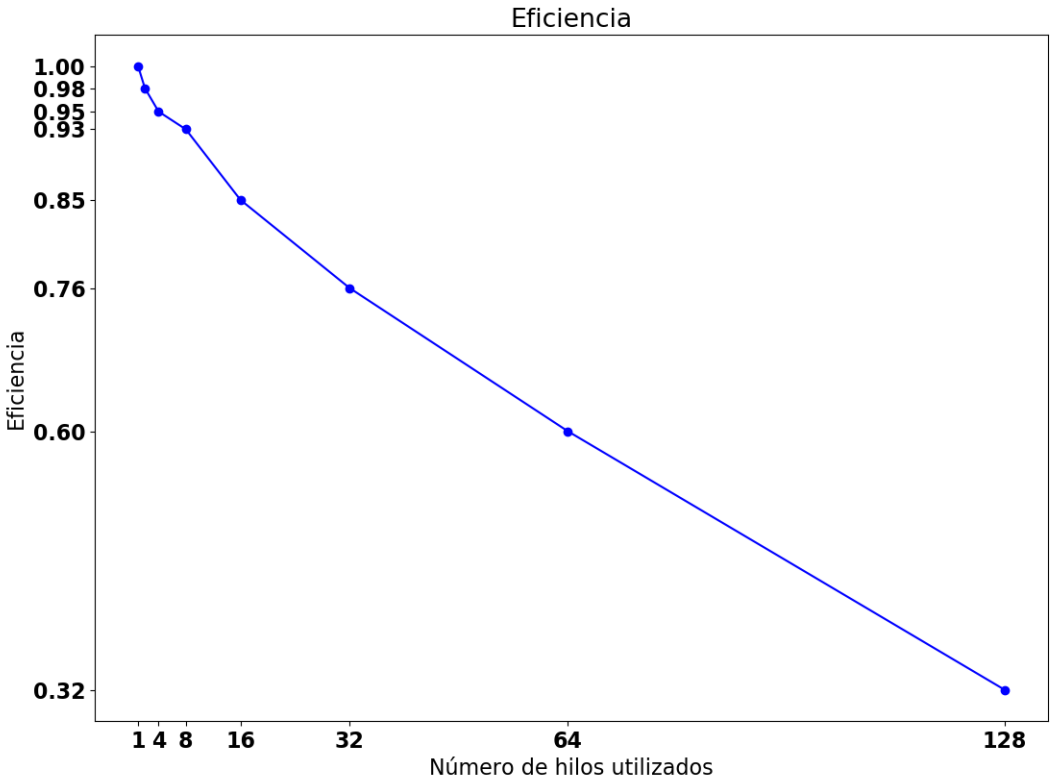

Figura 7.6: Eficiencia obtenido en el programa paralelo. Escala lineal

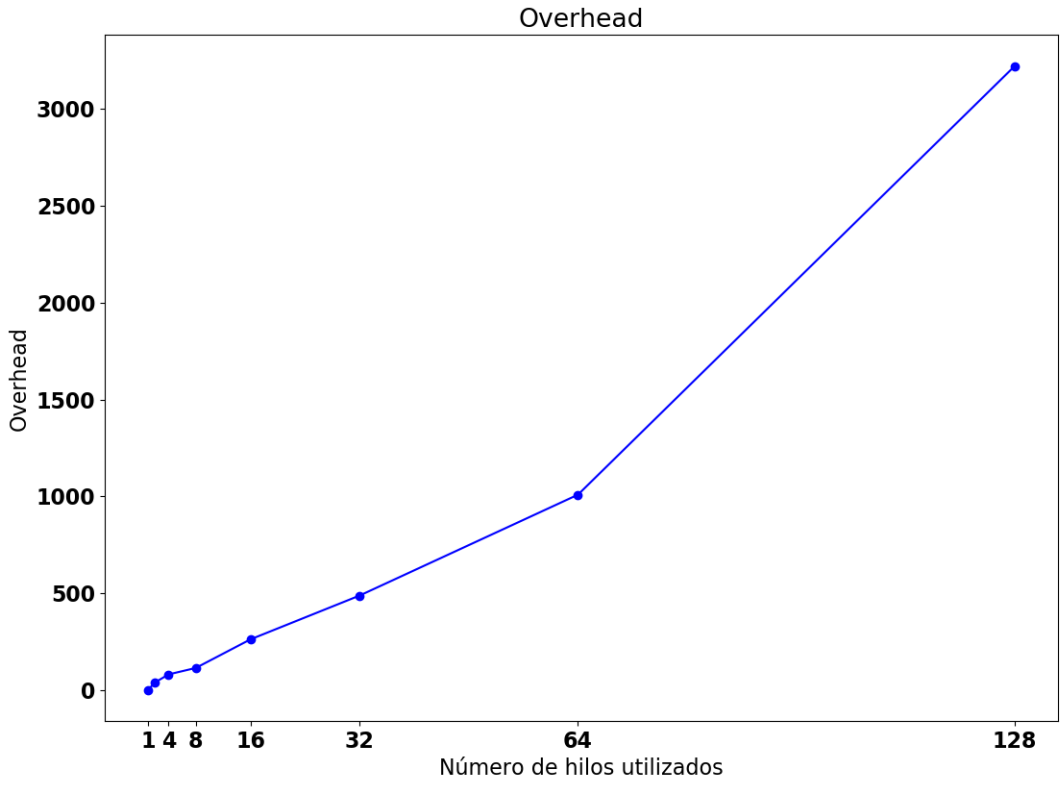

Figura 7.7: Overhead obtenido en las pruebas. Escala lineal

7.4. Pruebas de Escalabilidad Espacial

Los resultados de las evaluaciones experimentales de escalabilidad espacial nos muestran una tendencia cuadrática de crecimiento en el tiempo de ejecución, como se puede ver en la Figura 7.8. El tamaño de la entrada del problema se aumenta de forma cuadrática en cada una de las pruebas debido a que cuando se hace un aumento en el tamaño de las dimensiones de la matriz se hace en ambas al mismo tiempo, lo cual aumenta el tamaño de la matriz bidimensional.

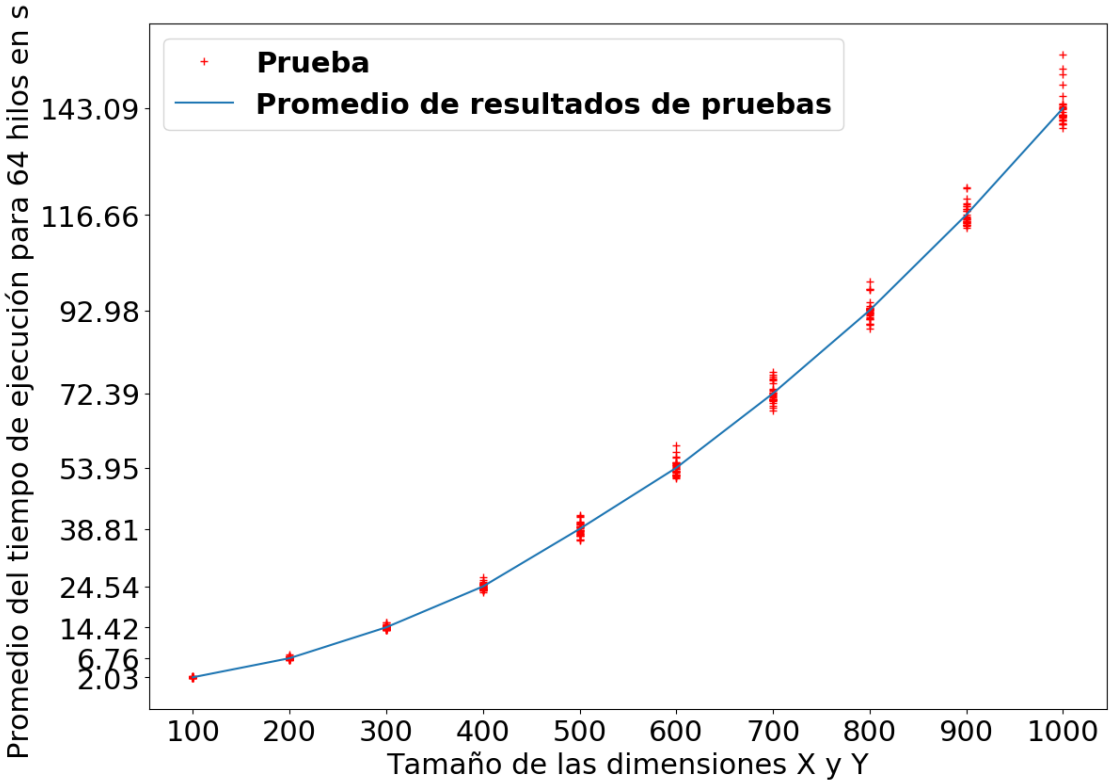

Figura 7.8: Escalabilidad espacial para incrementos de granularidad espacial en ambas dimensiones. Escala logarítmica

El comportamiento observado en los datos es el esperado en un programa que escale casi linealmente junto con el tamaño de entrada en el dominio espacial, en la Figura 7.9 se observa el crecimiento lineal del tiempo total de ejecución del programa con respecto al tamaño total de puntos en la matriz. En esta figura no se grafica el aumento en cada dimensión de la escala espacial, sino que se tiene el aumento en la cantidad total de puntos, se puede ver de una manera más clara la escalabilidad lineal que se alcanza en el programa paralelizado. Esto es evidencia de una buena escalabilidad espacial de parte de la nueva versión.

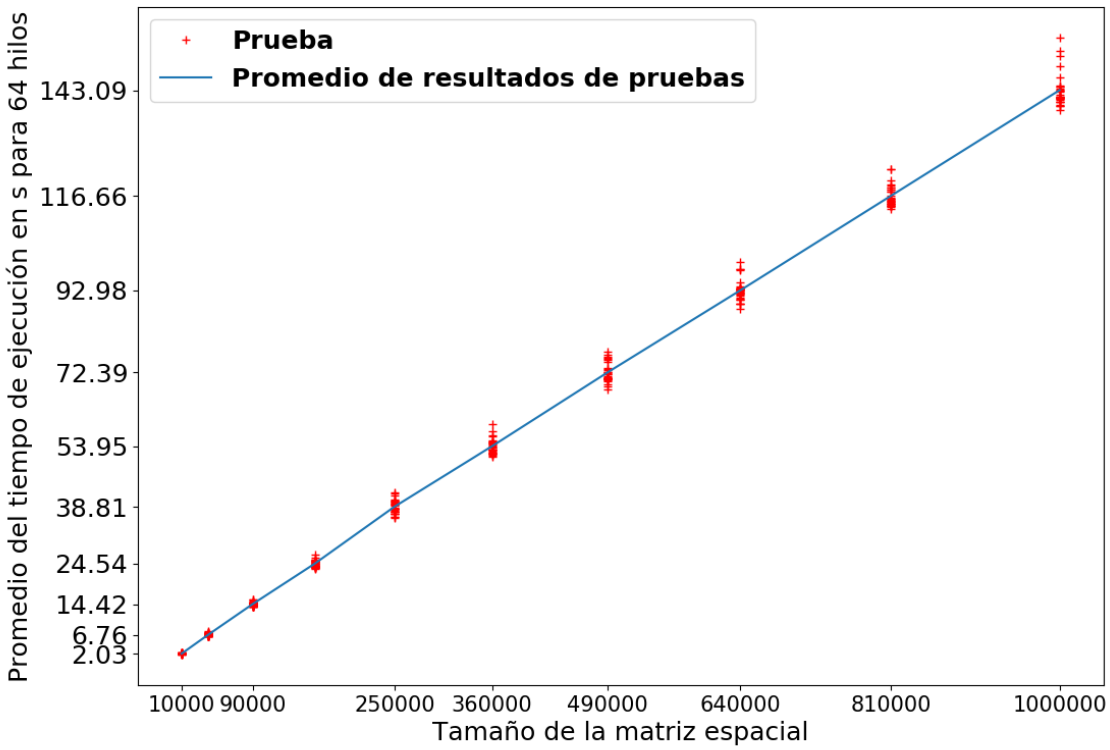

Figura 7.9: Escalabilidad espacial con cantidad total de puntos. Escala lineal

El programa PCell original tenía algunos errores de diseño que impedían incrementar el tamaño de la matriz espacial. Gracias a estas mejoras hechas, no solo se puede aumentar el tamaño, sino que también se puede llegar hasta una granularidad muy fina que ayuda a dar una mejor precisión a la simulación de plasma. Esto es beneficioso para los usuarios finales de la aplicación que deseen estudiar más a fondo algunos de los fenómenos que se dan en el campo magnético del plasma.

7.5. Pruebas de Escalabilidad Temporal

Los resultados de las evaluaciones experimentales de escalabilidad temporal muestran que el tiempo de ejecución escala de forma casi lineal con respecto a la cantidad de iteraciones temporales, como se puede observar en la Figura 7.10. Esto muestra que el

programa escala de forma aceptable junto con el aumento del tamaño en la dimensión temporal.

Con estas mejoras en el programa, se pueden hacer simulaciones más largas, lo que permite estudiar la evolución del plasma por períodos más largos. Esto permite a los investigadores analizar el comportamiento del plasma sin tener que estar restringidos a estados iniciales que avancen rápido hasta algún estado que sea de su interés, además de permitirles estudiar la evolución desde el inicio de algún proceso que tome muchos pasos temporales.

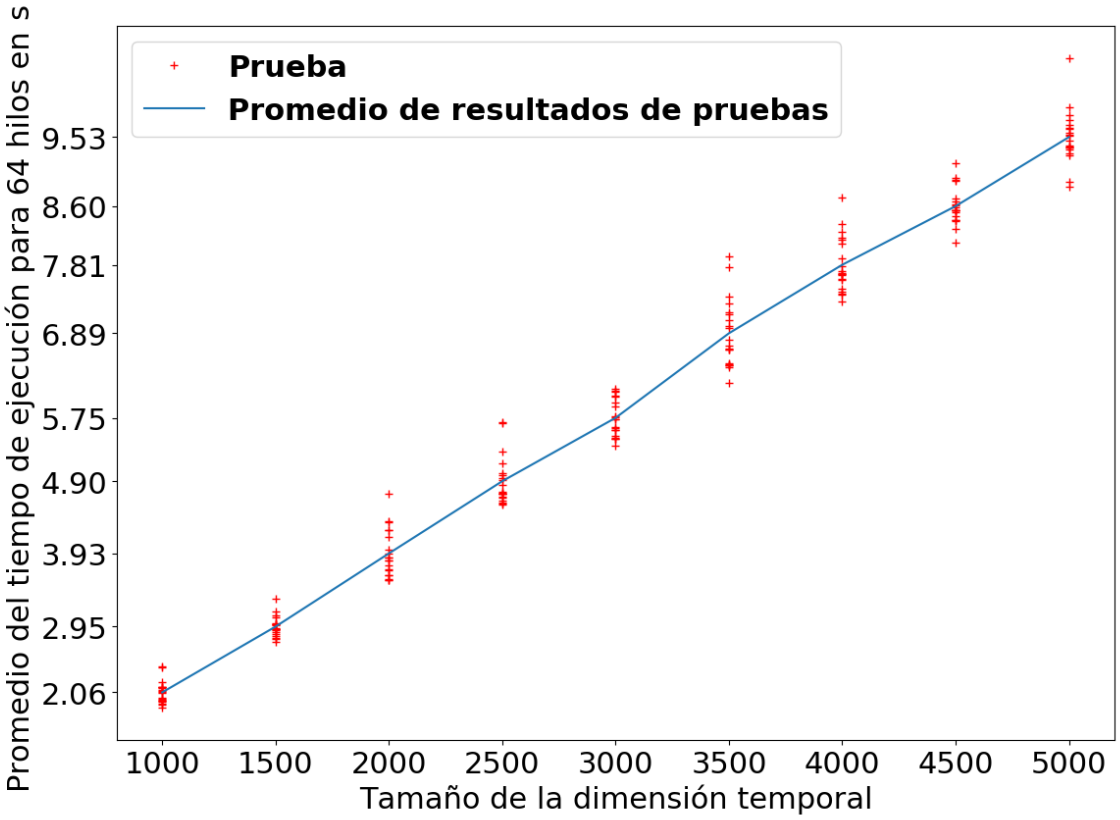

Figura 7.10: Escalabilidad temporal para distintas cantidades de iteraciones temporales en ParaPCell. Escala lineal

Dado el resultado de estas pruebas, se puede observar que la eficiencia del modelo mejoró. Además de que ahora se pueden ampliar las escalas de las simulaciones, lo cual

permitirá estudiar más a fondo los fenómenos, se logró obtener también una escalabilidad casi lineal con respecto a estos aumentos. Esto se discute más a fondo en el capítulo a continuación.

Capítulo 8

Conclusiones y Trabajo Futuro

En este trabajo se paralelizó el modelo PCell utilizando el paradigma de memoria compartida OpenMP para separar tareas entre hilos. No solamente se le hizo mejoras al modelo para ampliar el rango de los dominios de simulación, sino que también se disminuyó significativamente el tiempo total de ejecución del programa.

Se hizo una reescritura completa del programa original para corregir errores y limitaciones que tenía. En particular, se corrigieron problemas de ineficiencia en muchas de las funciones del programa, y se eliminaron tareas innecesarias.

Lo más importante, es que se modificó el diseño original para permitir al usuario aumentar las escalas espacial y temporal de la simulación. Ahora el límite de la escala espacial es mucho mayor al del programa original y está solo limitado por la capacidad computacional del equipo.

Por otro lado, los resultados de la experimentación con el nuevo programa muestran que este escala linealmente con el aumento del tamaño del problema, es decir, con el aumento de las ámbitos temporal y espacial de la simulación. Ahora el programa realiza simulaciones con escalas mucho mayores a las que permitía el programa original. Esto permite ahora al usuario hacer simulaciones más realistas.

Estas mejoras se lograron siguiendo una metodología de paralelización bien definida [McCool et al., 2012], que fue aplicada a un software científico. Se expuso el éxito de esta metodología para la paralelización de un modelo de simulación que utiliza ecuaciones diferenciales parciales. Dada la aceleración y la escalabilidad obtenida mediante el uso de esta metodología, se puede concluir que esta estrategia fue apta para mejorar el modelo.

Esta metodología presentó muy buenos resultados para esta aplicación y en futuros trabajos será utilizada para realizar otros proyectos de paralelización de aplicaciones científicas.

Con la paralelización usando OpenMP, se logró explotar de manera eficiente la arquitectura especializada en procesamiento multi-núcleo de los procesadores KNL del cluster. Con esto, se pudieron cumplir con suficiencia los objetivos propuestos, aprobados por el tribunal examinador de la propuesta de esta investigación. Por otro lado, a diferencia de OpenMP, una aplicación paralelizada con ayuda de MPI tendrá necesariamente un *overhead* mayor por causa de la carga de comunicación que se debe de dar entre procesos.

Se describió también el proceso de evaluación de los resultados mediante análisis estadístico. Estos análisis ayudaron a determinar que el modelo tiene una salida equivalente al programa original, de manera que nos aseguramos que el programa paralelo modela adecuadamente el fenómeno que se desea estudiar. Los análisis nos mostraron que no existen diferencias significativas entre ambos modelos, con lo que podemos asegurar que las salidas del modelo paralelo son equivalentes a las del modelo original, por lo que el modelo paralelizado puede ser utilizado de manera confiable para hacer el mismo tipo de simulaciones que se hacía con PCell original.

A partir del proceso de este trabajo y al haber logrado la aceleración y escalabilidad descritas antes, se aprendió sobre la capacidad de procesamiento multi-núcleo que tienen los Xeon KNL. Este trabajo presenta también un caso de estudio donde una aplicación de simulación en física de plasma se paraleliza y acelera explotando esta arquitectura de una manera adecuada, pues únicamente con aprovechar el procesamiento multi-núcleo se logró obtener una aceleración de alrededor de 42× como se ha expuesto. Esto es consistente con otros trabajos en los que se demuestra que la arquitectura saca más provecho de la paralelización multi-núcleo que de la paralelización con memoria distribuida y MPI [Jabbie et al., 2017]. En este trabajo se corroboró que la paralelización con OpenMP es considerable cuando se utiliza en las arquitecturas KNL.

Trabajo Futuro

Dentro de las posibles mejoras que se le desean hacer al programa, está su integración con *ParaView*. Esta es una aplicación *open-source*, multi-plataforma para análisis y

visualización de datos.

Esta aplicación fue desarrollada para crear visualizaciones interactivas en tres dimensiones. ParaView es particularmente útil para hacer visualizaciones de datos científicos, pues cuenta con soporte para computación distribuida, de manera que se pueda procesar grandes cantidades de datos. Tiene la capacidad de correr tanto en sistemas de memoria distribuida como en sistemas de memoria compartida con una sola unidad de procesamiento. Cuenta con soporte para plataformas Windows, macOS y Linux.

Otra mejora útil al programa ParaPCell consiste en ampliar el dominio de simulación a tres dimensiones. Esto sería muy útil para explorar fenómenos del plasma que suceden por debajo de la superficie o para generar simulaciones y visualizaciones de procesos como la reconexión magnética y las eyecciones de material que genera este fenómeno. Para ampliar el dominio a tres dimensiones no solo basta con mejorar la eficiencia del programa actual, sino que sería necesario reemplazar completamente el modelo matemático subyacente del programa, ya que las ecuaciones están diseñadas para un dominio espacial de dos dimensiones únicamente. Poder hacer una simulación en tres dimensiones mejoraría sustancialmente la representación del fenómeno real, pues de esta forma daría una mejor visión de los procesos que se dan en el plasma. La complejidad de este modelo en tres dimensiones haría necesario también el uso de estrategias paralelas para acelerar su ejecución, pues la cantidad de cálculos que se deben de hacer por cada punto de la matriz espacial aumentaría considerablemente al aumentar una dimensión.

Por otro lado, es de interés futuro investigar la paralelización del modelo de PCell utilizando MPI. Para esto habría que hacer grandes cambios al programa. En primer lugar, habría que empezar por cambiar los arreglos multidimensionales a arreglos de una sola dimensión para poder enviarlos fácilmente a través de mensajes. Esto también afectaría la forma en la que se recorren los mismos, y por lo tanto, sería necesario también reescribir la parte paralelizada con OpenMP.

La separación de las tareas también será algo que se debe de considerar al utilizar MPI, pues como cada cálculo de la función de convección utiliza los datos a su alrededor, la separación de la matriz espacial debe de tomar en cuenta la necesidad de que todos los datos necesarios para este cálculo estén en la memoria del mismo proceso.

Bibliografía

[iee, 1985] (1985). Ieee standard for binary floating-point arithmetic. *ANSI/IEEE Std 754-1985*.

[val, 2017] (2017). Valgrind. http://valgrind.org/. [Recurso en línea; accedido 30-May-2018].

[Amdahl, 1967] Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring)*, page 483, New York, New York, USA. ACM Press.

[AMES, 1977] AMES, W. F. (1977). 1 - fundamentals. In AMES, W. F., editor, *Numerical Methods for Partial Differential Equations (Second Edition)*, pages 1 – 40. Academic Press, second edition edition.

[Balsara and Käppeli, 2017] Balsara, D. S. and Käppeli, R. (2017). Von Neumann stability analysis of globally divergence-free RKDG schemes for the induction equation using multidimensional Riemann solvers. *Journal of Computational Physics*, 336:104–127.

[Carboni, 1994] Carboni, R. (1994). *Plasmas convectivos*. Tesis de maestría, Universidad de Costa Rica.

[Carboni and Frutos-Alfaro, 2004] Carboni, R. and Frutos-Alfaro, F. (2004). PCell: a 2D program for visualizing convective plasma cells. *Computing in Science and Engineering*, 6(4):101–104.

[Carboni and Frutos-Alfaro, 2005] Carboni, R. and Frutos-Alfaro, F. (2005). Computer simulation of convective plasma cells. *Journal of Atmospheric and Solar-Terrestrial Physics*, 67(17-18 SPEC. ISS.):1809–1814.

[Carboni and Frutos-Alfaro, 2015] Carboni, R. and Frutos-Alfaro, F. (2015). Computer Simulation of Convective Plasma Cells. *Journal of Atmospheric and Solar-Terrestrial Physics*, pages 1–18.

[Chen, 2016] Chen, F. F. (2016). *Introduction to Plasma Physics and Controlled Fusion*. Springer International Publishing, Cham.

[Davidson, 2001] Davidson, P. A. (2001). An Introduction to Magnetohydrodynamics. page 431.

[Dawson, 1983] Dawson, J. M. (1983). Particle simulation of plasmas. *Reviews of Modern Physics*, 55(2):403–447.

[Einkemmer et al., 2017] Einkemmer, L., Tokman, M., and Loffeld, J. (2017). On the performance of exponential integrators for problems in magnetohydrodynamics. *Journal of Computational Physics*, 330:550–565.

[Feldman, 1990] Feldman, S. I. (1990). A fortran to c converter. *SIGPLAN Fortran Forum*, 9(2):21–22.

[Flynn, 1972] Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960.

[Foster, 1995] Foster, I. (1995). *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Frutos-Alfaro,] Frutos-Alfaro, F. comunicación personal.

[Galloway and Weiss, 1981] Galloway, D.-J. and Weiss, N. (1981). Convection and magnetic fields in stars. *Astrophysical Journal*, 243:945–953.

[Gaur et al., 1989] Gaur, Y., Guarna, V. A., and Jablonowski, D. (1989). An environment for performance experimentation on multiprocessors. *Supercomputing '89:Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, (CSRD 865):8.

[Glowinski and Neittaanmaki, 2008] Glowinski, R. and Neittaanmaki, P. (2008). *Partial Differential Equations*, volume 16 of *Computational Methods in Applied Sciences*. Springer Netherlands, Dordrecht.

[Graham et al., 1982] Graham, S. L., Kessler, P. B., and Mckusick, M. K. (1982). gprof: a Call Graph Execution Profiler. *ACM SIGPLAN Notices*, 17(6):120–126.

[Hager and Wellein, 2011] Hager, G. and Wellein, G. (2011). *Introduction to High Performance Computing for Scientists and Engineers*.

[Jabbie et al., 2017] Jabbie, I. A., Owen, G., and Whiteley, B. (2017). Performance comparison of Intel Xeon Phi Knights Landing. (iii):268–281.

[Jardin, 2012] Jardin, S. (2012). Review of implicit methods for the magnetohydrodynamic description of magnetically confined plasmas. *Journal of Computational Physics*, 231(3):822–838.

[Kageyama, 2001] Kageyama, A. (2001). Magnetohydrodynamics Simulation in a Ball by Yin – Yang – Zhong Grid.

[Lamport, 1978] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565.

[Lütjens and Luciani, 2008] Lütjens, H. and Luciani, J. F. (2008). The XTOR code for nonlinear 3D simulations of MHD instabilities in tokamak plasmas. *Journal of Computational Physics*, 227(14):6944–6966.

[McCool et al., 2012] McCool, M., Reinders, J., and Robison, A. (2012). *Structured Parallel Programming*.

[Murphy, 2014] Murphy, N. (2014). Ideal Magnetohydrodynamics Overview of MHD.

[Murray, 1984] Murray, J. D. (1984). *Asymptotic Analysis*, volume 48 of *Applied Mathematical Sciences*. Springer New York, New York, NY.

[OpenMP Architecture Review Board, 2015] OpenMP Architecture Review Board (2015). The OpenMP API specification for parallel programming.

[Rémi Sentis, 2014] Rémi Sentis (2014). *Mathematical Models and Methods for Plasma Physics, Volume 1: Fluid Models*, volume 1. Birkhäuser Basel.

[R.J Goldston, 1995] R.J Goldston, P. R. (1995). *Introduction to Plasma Physics*, volume 19.

[Sevcik, 1989] Sevcik, K. C. (1989). Characterizations of parallelism in applications and their use in scheduling. *ACM SIGMETRICS Performance Evaluation Review*, 17(1):171–180.

[Space and Societal, 2008] Space, S. and Societal, W. E.-u. (2008). *Severe Space Weather Events–Understanding Societal and Economic Impacts*. National Academies Press, Washington, D.C.

[Tang et al., 2017] Tang, W., Wang, B., Ethier, S., Kwasniewski, G., Hoefler, T., Ibrahim, K. Z., Madduri, K., Williams, S., Oliker, L., Rosales-Fernandez, C., and Williams, T. (2017). Extreme Scale Plasma Turbulence Simulations on Top Supercomputers Worldwide. *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, (November):502–513.

[Tchekhovskoy et al., 2007] Tchekhovskoy, A., McKinney, J. C., and Narayan, R. (2007). WHAM: a WENO-based general relativistic numerical scheme - I. Hydrodynamics. *Monthly Notices of the Royal Astronomical Society*, 379(2):469–497.

[Verboncoeur, 2005] Verboncoeur, J. P. (2005). Particle simulation of plasmas: review and advances. *Plasma Physics and Controlled Fusion*, 47(5A):A231.

A P É N D I C E S

Apéndice A

Datos crudos

A.1. Resultados para pruebas variando cantidad de hilos

Prueba	1 Hilo	2 Hilos	4 Hilos	8 Hilos	16 Hilos	32 Hilos	64 Hilos	128 Hilos
1	1548.544	820.536	414.439	210.803	118.268	66.084	41.064	34.284
2	1526.595	766.151	414.404	210.778	120.307	65.965	40.228	33.626
3	1526.333	780.232	415.118	210.748	119.82	65.928	40.133	49.238
4	1525.885	765.905	414.434	230.271	119.164	66.345	40.974	33.723
5	1525.838	765.836	386.413	216.059	111.843	61.656	42.407	33.268
6	1526.88	765.381	386.802	196.583	111.599	60.414	40.282	49.278
7	1553.258	766.857	426.879	196.824	110.486	61.804	38.91	33.752
8	1553.732	765.134	386.549	216.139	109.458	61.306	38.879	34.247
9	1551.055	842.175	426.247	196.637	111.763	61.74	37.235	49.259
10	1548.788	765.38	386.629	196.561	110.975	61.886	40.301	32.959
11	1525.923	765.716	387.184	214.254	110.404	62.004	37.075	32.882
12	1525.907	765.829	386.405	210.02	110.003	62.052	38.291	33.26
13	1526.467	822.012	414.138	206.004	110.468	60.957	37.487	33.936
14	1527.38	820.71	431.386	196.695	111.91	61.541	38.063	49.28
15	1527.583	820.446	414.08	215.658	110.481	61.301	40.519	32.568
16	1529.066	820.565	414.505	196.571	102.128	66.109	37.37	33.792
17	1529.378	765.858	386.605	196.679	110.44	65.846	42.38	37.032
18	1525.646	766.807	386.635	196.548	112.485	66.006	40.591	37.087
19	1526.401	779.123	386.812	201.034	112.911	65.858	40.097	35.618
20	1525.932	766.7	386.408	196.503	111.653	62.306	42.002	36.582

Cuadro A.1: Datos de pruebas de aceleración. Cada valor muestra el tiempo de ejecución en segundos de cada una de las pruebas para la cantidad de hilos mostrada en el encabezado de cada columna

A.2. Resultados para pruebas variando tamaño de la grilla espacial

Prueba	100^2	200^2	300^2	400^2	500^2	600^2	700^2	800^2	900^2	1000^2
1	2.153	7.665	15.546	26.095	42.329	56.709	77.683	100.016	120.554	152.814
2	1.984	7.142	14.391	23.919	42.091	52.318	75.762	98.221	123.385	151.568
3	2.215	6.742	14.867	24.703	40.548	54.569	75.578	93.306	123.346	143.445
4	2.014	6.663	14.242	24.914	37.144	53.066	73.589	93.981	114.756	139.17
5	2.008	6.561	13.726	25.215	40.012	52.159	73.168	93.013	116.208	144.05
6	1.981	6.482	14.881	24.065	39.468	54.2	72.515	93.165	113.877	144.205
7	1.876	6.373	14.005	25.528	39.927	55.324	70.796	91.217	118.766	146.208
8	1.902	6.334	13.776	23.527	39.226	53.632	70.821	93.169	114.503	143.061
9	1.98	6.562	14.416	24.446	38.565	52.079	71.928	92.252	118.767	141.003
10	1.866	6.759	14.348	24.46	39.268	51.71	70.895	90.699	115.492	140.932
11	2.182	6.385	14.093	23.54	38.281	54.718	70.684	93.635	118.096	141.273
12	2.068	6.507	14.66	23.741	38.421	53.018	70.981	92.38	114.626	140.041
13	2.212	7.282	15.659	25.39	37.831	55	71.569	92.638	117.676	142.031
14	1.984	7.134	14.72	23.189	40.448	56.399	72.245	90.741	115.941	140.705
15	1.964	6.978	14.068	26.83	37.567	53.725	71.062	93.449	119.582	143.052
16	1.978	6.519	13.695	23.729	40.183	52.629	72.52	93.372	115.858	141.282
17	2.004	6.527	13.846	25.574	37.469	53.214	76.435	92.406	114.982	142.77
18	2.087	6.427	14.157	24.508	36.217	52.336	72.081	95.055	116.613	141.338
19	2.294	6.829	14.017	23.903	38.146	57.822	70.474	90.999	114.232	139.44
20	2.184	7.338	15.169	23.824	36.243	52.096	69.35	92.889	117.615	141.45

Cuadro A.2: Datos de pruebas de escalabilidad espacial. Cada valor muestra el tiempo de ejecución en segundos de cada una de las pruebas para la cantidad de puntos espaciales mostrada en el encabezado de cada columna

A.3. Resultados para pruebas variando la cantidad de pasos temporales

Prueba	1000	1500	2000	2500	3000	3500	4000	4500	5000
1	2.141	2.986	4.36	4.939	6.044	7.15	8.146	8.605	9.389
2	2.132	3.097	4.249	5.141	6.103	7.78	8.716	8.107	9.276
3	1.951	2.981	3.576	4.686	5.59	6.682	7.427	8.62	8.858
4	1.965	2.904	3.579	4.612	5.625	6.458	7.681	8.952	9.929
5	1.909	3.149	4.73	4.691	5.623	7.385	7.691	8.935	9.752
6	2.067	2.917	4.153	5.689	5.731	7.061	7.672	8.29	9.639
7	1.98	2.915	4.246	4.92	6.054	6.677	7.723	8.548	9.476
8	2.051	2.911	3.88	4.74	5.483	6.226	7.323	8.54	9.548
9	2.09	2.82	3.98	4.738	5.377	6.722	7.488	8.409	10.58
10	2.101	2.975	3.583	4.592	5.779	6.801	7.7	8.666	9.826
11	2.128	2.85	3.724	4.605	5.761	6.672	8.179	8.696	9.355
12	2.132	2.795	3.872	4.76	5.594	6.966	7.795	8.413	9.687
13	2.197	3.069	3.836	4.979	5.472	6.429	7.408	8.401	8.919
14	1.95	3.327	4.354	5.004	5.957	6.461	8.089	9.173	9.578
15	1.985	2.987	3.638	4.853	5.912	7.294	7.443	8.585	9.652
16	1.904	2.875	3.928	5.304	5.733	6.985	7.618	8.548	9.383
17	1.935	2.742	3.775	4.727	6.144	7.174	8.25	8.514	9.564
18	2.416	2.965	3.638	5.702	5.507	7.926	8.365	8.976	9.314
19	2.404	2.922	3.7	4.636	5.477	6.471	7.899	8.458	9.413
20	1.86	2.778	3.836	4.731	6.112	6.482	7.604	8.537	9.402

Cuadro A.3: Datos de pruebas de escalabilidad temporal. Cada valor muestra el tiempo de ejecución en segundos de cada una de las pruebas para la cantidad de iteraciones temporales mostrada en el encabezado de la columna