INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN Y DESARROLLO DE TECNOLOGÍA DIGITAL

MAESTRÍA EN CIENCIAS EN SISTEMAS DIGITALES

DESCOMPOSICIÓN DE DOMINIO PARA MODELOS DE INTERFASE DIFUSA EN PROCESADORES GRÁFICOS

TESIS

QUE PARA OBTENER EL GRADO DE MAESTRÍA EN CIENCIAS

PRESENTA

ING. EDILBERTO SÁNCHEZ MORENO

BAJO LA DIRECCIÓN DE

DR. JUAN JOSÉ TAPIA ARMENTA

ENERO 2016 TIJUANA, B.C., MÉXICO

SIP-14

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de Tijuana, B.C. siendo las 13:30 horas del día 11 del mes de enero del 2016 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación de CITEDI para examinar la tesis titulada:

DESCOMPOSICIÓN DE DOMINIO PARA MODELOS DE INTERFASE DIFUSA EN PROCESADORES GRÁFICOS.

Presentada por el alumno:

SÁNCHEZ	**MORENO**	**EDILBERTO**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

B	1	3	0	9	5	9

aspirante de:

MAESTRÍA EN CIENCIAS EN SISTEMAS DIGITALES

Después de intercambiar opiniones, los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Director de tesis

DR. JUAN JOSÉ TAPIA ARMENTA

DR. OSCAR HUMBERTO MONTIEL ROSS

DR. SERGIO IVVAN VALDEZ PEÑA

DR. JULIO CÉSAR ROLÓN GARRIDO

M. EN C. ISAURA GONZÁLEZ RUBIO ACOSTA

PRESIDENTE DEL COLEGIO DE PROFESORES

DRA. MIREYA SARAÍ GARCÍA VÁZQUEZ

S.E.P.
INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN Y DESARROLLO
DE TECNOLOGÍA DIGITAL
DIRECCIÓN

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México, D.F. el día **18** del mes de **Enero** del año **2016**, el (la) que suscribe **Ing. Edilberto Sánchez Moreno** alumno(a) del Programa de **Maestría en Ciencias en Sistemas Digitales**, con número de registro **B130959**, adscrito(a) al **Centro de Investigación y Desarrollo de Tecnología Digital**, manifiesto(a) que es el (la) autor(a) intelectual del presente trabajo de Tesis bajo la dirección del (de la, de los) **Dr. Juan José Tapia Armenta** y cede los derechos del trabajo titulado **Descomposición de domino para modelos de interfase difusa en procesadores gráficos**, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del (de la) autor(a) y/o director(es) del trabajo. Este puede ser obtenido escribiendo a las siguientes direcciones **esanchezm@citedi.mx**. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Ing. Edilberto Sánchez Moreno

AGRADECIMIENTOS

Principalmente a Dios por acompañarme siempre y regalarme la vida día a día.

Al Instituto Politécnico Nacional y al Centro de Investigación de Desarrollo y Tecnología Digital por brindarme la oportunidad de pertenecer a este selecto grupo de aspirantes a un posgrado, así como al propio instituto y al Consejo Nacional de Ciencia y Tecnología (CONACyT) por los apoyos económicos brindados.

A mi director de tesis Dr. Juan José Tapia Armenta por confiar en mi y permitirme participar dentro de su equipo de trabajo.

Al Dr. Sergio Ivvan Valdéz Peña que gracias al apoyo recibido durante la estancia de investigación en la ciudad de Guanajuato esta tesis cumplió sus objetivos.

También agradezco al comité de revisión por haber aceptado ser parte del desarrollo de esta tesis.

A mi familia, en especial a mis padres el Sr. Edilberto Sánchez Antonio y la Sra. Gregoria Moreno Barrios por confiar en mí como persona y permitirme viajar de tan lejos para realizar este proyecto profesional y personal. A mi hermana Edith Yesmin y su familia porque nunca me negaron un consejo y mucho menos su amistad.

De manera muy especial a los amigos que la vida me regaló durante este trabajo. Paulina, Dana, Maribel y Oscar, porque con ellos compartí momentos inolvidables y fueron partícipes de este trabajo cuando me brindaron consejos de superación y perseverancia.

Finalmente a mis compañeros y amigos de laboratorio Sonia, Fernando, Rigoberto y Víctor con quienes compartí horas de trabajo y muchas actividades memorables.

DESCOMPOSICIÓN DE DOMINIO PARA MODELOS DE INTERFASE DIFUSA EN PROCESADORES GRÁFICOS

Resumen

En este trabajo se resolvió un modelo de interfase difusa en tres dimensiones formulado por la ecuación de Cahn-Hilliard aplicando el método de diferencias finitas con descomposición de dominio en un *cluster* de procesadores gráficos. Se aplicó la ecuación de Cahn-Hilliard para simular la evolución de la separación de las dos fases en un fluido binario con una interfase difusa.

La ecuación de Cahn-Hilliard es no lineal y de cuarto orden, lo cual demanda métodos numéricos robustos para su solución, en este trabajo se resolvió en 2D con un esquema semi-implícito del método de diferencias finitas aplicando el gradiente conjugado en un GPU. También se resolvió en 3D de forma iterativa con el esquema explícito del método de diferencias finitas.

El uso de mallas finas requiere de grandes cantidades de memoria para su almacenamiento. En la presente tesis se utilizó el método alternante de Schwarz como una técnica de descomposición de dominio mediante la cual es posible obtener resultados similares a los obtenidos con un solo dominio. Para optimizar el uso del método alternante de Schwarz se utilizó un *cluster* de dos nodos con procesadores gráficos.

Palabras clave. Interfase difusa, descomposición de dominio, diferencias finitas, GPU.

DOMAIN DESCOMPOSITION FOR DIFFUSE INTERFACES MODELS IN A GRAPHICS PROCESSORS.

Abstract

This work present a diffuse interface model in three dimensions given by Cahn-Hilliard equation, we using the finite difference method with a domain decomposition on a cluster of GPU's. The Cahn-Hilliard equation was used to simulate the evolution and separation between two phases in a binary fluid with a diffuse interface.

The Cahn-Hilliard equation is nonlinear and to fourth order, which need to robust numerical methods for their solution, this equation was solved in 2D with a semi-implicit scheme of the finite difference method using the conjugate gradient in a GPU. The Cahn-Hilliard equation was solved iteratively in 3D with the explicit scheme of finite difference method.

The finite difference method with fine mesh require more memory for storage. The Schwarz alternating method was used in this work as a domain decomposition method by to obtain similar result to those obtained results with a simple domain. A two-node cluster with graphics processors was used to optimize the use of the alternative method of Schwarz.

Keywords. Diffuse interface, domain decomposition, finite differences, GPU.

Índice general

Lista de figuras

Lista de tablas

Glosario de términos y lista de acrónimos

API Interfaz de programación de aplicaciones
(*Application Programming Interface*)

COO Almacenamiento de matrices ralas mediante una lista de coordenadas (renglones y columnas) y valores
(*Coordinate*)

CPU Unidad central de procesamiento
(*Central Processing Unit*)

CSC Almacenamiento con columnas comprimidas para matrices ralas
(*Compressed Sparse Column*)

CSR Almacenamiento con renglones comprimidos para matrices ralas
(*Compressed Sparse Row*)

CUDA Arquitectura de dispositivos de cómputo unificado
(*Compute Unified Device Architecture*)

EDP Ecuación diferencial parcial

ELL Almacenamiento por *Ellpack-Itpack*

GPGPU Computación de propósito general en unidades de procesamiento gráfico
(*General Purpose Computing on Graphics Processing Units*)

GPU Unidad de procesamiento gráfico
(*Graphics Processing Unit*)

HYB Almacenamiento híbrido
(*Hybrid*)

MDF Método de diferencias finitas

MPI Interfaz de paso de mensaje
(*Message Passing Interface*)

NFS Protocolo para sistemas de archivos distribuidos en una red local de computadoras (*Network File System*)

PCI Interconexión de componentes periféricos (*Peripheral Component Interconnect*)

RAM Memoria de acceso aleatorio (*Random Access Memory*)

SM Multiprocesador de transferencia continua (*Streaming Multiprocessor*)

SP Procesador escalar (*Scalar Processor*)

SPMD Programa único, datos múltiples (*Single Program, Multiple Data*)

SSH Protocolo seguro utlizado para acceder a máquinas remotas a través de una red (Secure SHell)

Capítulo 1

Introducción

1.1. Antecedentes y motivación

El estudio de fluidos multicomponente es de gran importancia en aplicaciones científicas y tecnológicas, ya que permiten dar solución a problemas sencillos como la separación entre mezclas de fluidos, formaciones de gotas de agua o burbujas, los cuales se encuentran presentes en gran número de fluidos complejos (emulsiones, espumas y soluciones poliméricas), lo que hace interesante el estudio de la dinámica de fluidos.

Debido a la importancia que han tenido durante estos últimos años el uso de fluidos multicomponentes en el procesamiento de materiales [1], el modelado computacional desempeña un papel importante al realizar análisis en esta área de investigación, ya que resulta ser más económico trabajar con simulaciones a partir de modelos que obtienen resultados similares a los de un experimento físico. Sin embargo, la dinámica de fluidos es un desafío computacional porque requiere de precisión al capturar datos en las distintas escalas de tiempo que intervienen en el fenómeno bajo estudio.

En la literatura hay modelos que permiten estudiar el comportamiento de las interfases de mezclas binarias. Estos modelos son herramientas importantes en el modelado de las ciencias de los materiales. Por ejemplo, cuando la cinética de una reacción es mayor que la difusión provoca físicamente una separación en dos estados o fases únicas. La separación de fases entre las aleaciones de metales y el crecimiento de un grano de cristal dentro de una aleación de metal en su etapa de cocimiento, son ejemplos de lo que sucede en cada capa de transición [2]. Dos ecuaciones que modelan la cinética de reacción en los sistemas de reacción-difusión, así como los modelos de difusión-convección en la dinámica de fluidos son las ecuaciones de Allen-Cahn y Cahn-Hilliard. En [2] se analiza una mezcla entre líquido-líquido con límites de espesor finito, allí se observa que hay un cambio rápido y suave en la frontera de ambos fluidos, a esta variación se le conoce como interfase difusa.

Durante los últimos años los modelos de interfase difusa se han utilizado como alternativa en el estudio de fluidos multicomponente, por ejemplo, en [3] se propone un esquema conservativo con el método de diferencias finitas para dar solución numérica a la ecuación de Cahn-Hilliard con características inherentes, que van desde la conservación de la masa hasta el decremento de energía total a partir del modelo y en [4] se revisa el desarrollo reciente de modelos con interfase difusa junto con los diferentes métodos numéricos utilizados para modelar fluidos multicomponente. El modelo se basa en un sistema de Navier-Stokes acoplado con la ecuación de Cahn-Hilliard a través de una fuerza de tensión superficial que depende de la fase de campo con diferentes variables como la densidad y la viscosidad, así como un término de advección.

Los modelos con interfase difusa tienen un parámetro de orden, es decir una capa de transición completamente desarrollada de magnitud pequeña, la teoría de interfases difusas permite analizar problemas en la fase de separación a través de este parámetro de orden el cual evoluciona con el tiempo. Es posible calcular este parámetro de orden con modelos de campo de fase (*phase-field*) [1].

El límite clásico infinitamente delgado de separación entre dos fluidos inmiscibles se sustituye por una pequeña región de transición, a tráves del cual la composición de la mezcla varía constantemente [5]. Algunos modelos de campo de fase son capaces de calcular los cambios topológicos como la división y fusión de fluidos multicomponente [4].

Hay muchas técnicas numéricas para dar solución a modelos evolutivos como los antes mencionados, el Método de Diferencias Finitas (MDF) [6] es una técnica sencilla de utilizar e implementar para dar soluciones aproximadas a ecuaciones diferenciales al transformarlas en ecuaciones algebráicas y que permite una resolución y precisión aproximada a la solución real. El MDF puede ser implementado utilizando diferentes esquemas, por ejemplo, en [7] se presentan diferentes técnicas utilizadas con el MDF en el modelo de Cahn-Hilliard, que van desde un esquema explícito básico, hasta utilizar un sistema no lineal estabilizado.

Recientemente se han realizado experimentos que modelan evoluciones morfológicas con el modelo de Cahn-Hilliard en 3D. En [8], se ha resuelto numéricamente este modelo por diferentes técnicas, entre las más utilizadas están diferencias finitas, elementos finitos, volumen finito y métodos espectrales.

El modelo de Cahn-Hilliard fue propuesto en 1958, para emular transiciones de fases en aleaciones binarias, posteriormente se realizaron muchas aplicadas a nanoescala y a escala planetaria, tales como el análisis de la formación de polímeros, procesamiento de imágenes, electromagnéticas y la dinámica planetaria [9].

Las aplicaciones del modelo de Cahn-Hilliard generalmente consumen mucho poder de cómputo. El uso de arquitecturas paralelas permite la ejecución de dos o más procesos al mismo tiempo utilizando más de un procesador. El procesamiento paralelo hace que un programa

se ejecute rápidamente porque existen más procesadores que trabajn en forma simultánea y coordinada en diferentes partes de un problema.

La construcción de aplicaciones más complejas ha requerido de computadoras más rápidas. Debido a la reducción de costos del hardware y al incremento del uso en mayor capacidad de cómputo surgen los procesadores multinúcleo (*Multicore*), los cuales están integrados por más de un núcleo (*core*) de procesamiento. Estos procesadores se encuentran desde pequeñas computadoras portátiles hasta estaciones de trabajo.

Otra vertiente de los procesadores multicores son las Unidades de Procesamiento Gráficos (GPU, del inglés *Graphics Processing Unit*). Una GPU está altamente segmentada, lo que indica que posee gran cantidad de unidades funcionales, y son usadas para apoyar al sistema operativo y la demanda de gráficos que requieren aplicaciones como los videojuegos. Con la finalidad de aplicar las GPUs en cómputo de propósito general, se ha introducido la filosofía GPGPU (del inglés General Purpose Computing on Graphics Processing Units), esto significa que las GPUs no solo son usadas para el procesamiento de gráficos, sino para una gran diversidad de aplicaciones de propósito general [10].

En los últimos años, las aplicaciones generales que combinan el cómputo en procesadores de propósito general y de procesamiento gráfico se ha incrementado considerablemente, debido a la capacidad de cómputo de los procesadores gráficos y la disminución en sus precios, por lo que se puede crear un sistema de cómputo de alto rendimiento masivamente paralelo con poca inversión [11].

Actualmente los *clusters* (agrupamientos) de computadoras desempeñan un papel importante en diversas áreas de las ciencias de la computación, ya que al tener disponible la capacidad de cómputo en cada elemento del *cluster*, es posible resolver problemas que requieren una alta capacidad de cálculo. Este tipo de *cluster* es denominado *cluster* de alto rendimiento [12]. Muchas de las computadoras más rápidas del mundo consisten de un *cluster* con GPUs, estos tienen la particularidad de que todos sus nodos incluyen una o varias GPUs [13].

Los métodos de descomposición de dominio son técnicas que se utilizan para resolver problemas donde se necesitan mayores cantidades de memoria, estos métodos trabajan a través de comunicaciones entre procesos. En [8] se propone un algoritmo paralelo para calcular las soluciones en equilibrio de la ecuación de Cahn-Hilliard-Cook. La ecuación se discretiza mediante un esquema de diferencias finitas centradas y el tamaño del paso de tiempo se decide por una adaptación tentativa en el tiempo, al igual que utiliza un algoritmo de Newton-Krylov-Schwarz para resolver el problema no lineal en cada paso de tiempo. En comparación con métodos explícitos, el esquema implícito permite dar pasos de tiempo mucho más grandes y se utiliza la técnica de Schwarz que es un método de descomposición de dominio trabajado hasta en 2048 procesadores obteniendo resultados razonables para el modelo de Cahn-Hilliard-Cook.

En este trabajo se propone una solución numérica al modelo de Cahn-Hilliard con el MDF y utilizando el método de Schwarz como una técnica de descomposición de dominio para optimizar el tiempo de cómputo en un *cluster* de GPUs.

1.2. Planteamiento del problema

La implementación de modelos matemáticos en un sistema informático es ahora una solución económica en comparación con el desarrollo de experimentos físicos, esto se convierte en un reto computacional si las técnicas numéricas utilizadas en los modelos requieren de grandes cantidades de procesamiento, lo que impide obtener resultados en un tiempo razonable.

El método de diferencias finitas es una técnica numérica sencilla de comprender y de aplicar a modelos matemáticos como el de Cahn-Hilliard. Una forma sencilla de aplicar el método de diferencias finitas es el esquema explícito, pero dado que el modelo de Cahn-Hilliard está expresado por una ecuación de cuarto orden no lineal, aplicar el esquema explícito requiere de discretizar el tiempo en intervalos muy pequeños y a consecuencia de esto, el tiempo de procesamiento se incrementa.

La arquitectura paralela que presentan las unidades de procesamiento gráfico permiten disminuir significativamente el tiempo de procesamiento en el modelado matemático, pero para el modelo de Cahn-Hilliard los tiempos de procesamiento siguen siendo altos.

Los métodos de descomposición de dominio son utilizados en cómputo paralelo como una alternativa para encontrar soluciones aproximadas a modelos matemáticos. Estos métodos permiten dividir un problema en diferentes subproblemas de menores dimensiones.

En la presente tesis se plantea disminuir el tiempo de procesamiento del modelo de Cahn-Hilliard utilizando el método de diferencias finitas con esquema explícito y aplicar técnicas de descomposición de dominio que permitan resolver un problema general mediante diferentes subrutinas.

1.3. Hipótesis

La implementación de técnicas de descomposición de dominio en un *cluster* de procesadores gráficos reduce el tiempo de ejecución del modelo de Cahn-Hilliard en comparación con el uso de un GPU.

1.4. Objetivo General

Implementar diferencias finitas con descomposición de dominio en paralelo para modelar fluidos con interfase difusa en procesadores gráficos.

1.5. Objetivos específicos

- Formulación de la ecuación de Cahn-Hilliard utilizando el método de diferencias finitas.
- Solución numérica a la ecuación de Cahn-Hilliard con el método de diferencias finitas en un programa secuencial.
- Solución numérica a la ecuación de Cahn-Hilliard con el método de diferencias finitas en un programa en paralelo.
- Formulación de la ecuación de Cahn-Hilliard con el método de diferencias finitas y el método alternante de Schwarz.
- Solución numérica de la ecuación de Cahn-Hilliard con descomposición de dominio en un procesador gráfico.
- Desarrollo e implementación de un modelo de interfase difusa mediena la ecuación de Cahn-Hilliard en un *cluster* de GPU's.

1.6. Organización de la tesis

Este trabajo está organizado en seis capítulos. En el capítulo 1 se presenta la introducción del desarrollo de la tesis, así como el planteamiento del problema y objetivos que se desean obtener durante la evolución del trabajo. En el capítulo 2 se abordan conceptos generales sobre los modelos de interfase difusa y se presentan los modelos de Allen-Cahn y Cahn-Hilliard. El desarrollo de la tesis se basa en el modelo de Cahn-Hilliard. En el capítulo 3 se realiza la formulación con el método de diferencias finitas y el método alternante de Schwarz para el modelo de Cahn-Hilliard. En el capítulo 4 se muestran los conceptos generales para la implementación de algoritmos utilizando diferentes bibliotecas de funciones como CUDA, CUBLAS, CUSPARSE y MPI. Las diferentes aplicaciones, métodos y técnicas utilizadas en los capítulos 3 y 4 se presentan como resultados en el capítulo 5, donde se abordan diferentes argumentos de interpretación para el modelo de Cahn-Hilliard. En el capítulo 6 se dan a conocer las conclusiones obtenidas y las propuestas a trabajo futuro que se consideran en el desarrollo de la presente tesis.

Capítulo 2

Modelos de interfase difusa

Los modelos clásicos utilizados en la dinámica de fluidos definen a la tensión superficial como una capa infinitamente delgada dotada de ciertas propiedades físicas. En los modelos de interfase difusa, este concepto se remplaza por una capa de variación continua con un parámetro de orden (por ejemplo, la densidad). Para el estudio y el análisis de una interfase difusa se utilizan modelos evolutivos de campo de fase [9].

La ciencia de los materiales utiliza los modelos de campo de fase para analizar fenómenos ocurridos durante el desarrollo de los materiales. La idea principal de estos modelos es utilizar un parámetro de orden o fase (ϕ) que muestre el estado actual del fluido en cada instante de tiempo. Esto provoca que el parámetro de orden varíe constantemente durante la evolución del modelo hasta alcanzar un estado donde la variación de las fases sea uniforme [9].

La conservación del parámetro de orden durante la evolución de los modelos de campo de fase permite analizar efectos tensoactivos como las fuerzas a largo alcance y la viscoelasticidad, posibles de modelar mediante modificaciones adecuadas en la energía libre. Un ejemplo es el proceso de recocido por el cual un metal, vidrio o cualquier otro material se calienta y se deja enfriar. Las tres etapas de recocido consisten en la recuperación, recristalización y el equilibrio. La primera etapa es el reblandecimiento del metal a través de la eliminación de defectos cristalinos. La segunda exhibe granos que crecen para reemplazar a las deformaciones internas. La tercera etapa es el estado por el cual este crecimiento ha alcanzado el equilibrio [14]. Particularmente, en este proceso el metal se fortalece, mejora su estructura y ductilidad. Los ejemplos más comunes en la práctica son los metales tales como el Cobre, Plata, acero y bronce. La recristalización es el proceso que puede ser modelado con las ecuaciones de Allen-Cahn y Cahn-Hilliard, los granos deformados se reemplazan por granos no deformados que se generan y crecen hasta que los granos originales se han consumido por completo [14].

2.1. Modelo de Allen-Cahn

El modelo de Allen-Cahn fue propuesto por Allen y Cahn en 1979 [15] y ha sido ampliamente utilizado para estudiar diversos fenómenos en la naturaleza, particularmente se ha convertido en una ecuación que permite el desarrollo y el estudio de la interfase difusa para el estudio de la dinámica y las transiciones de la interfase dentro de la ciencia de los materiales [16]. Este modelo ha sido implementado para dar solución a problemas como el procesamiento de imágenes [17, 18], el movimiento de flujos [19] y la formación de cristales [20].

La ecuación de Allen-Cahn

$$\frac{\partial \phi}{\partial t} = \varepsilon \nabla^2 \phi - f^{'}(\phi), \tag{2.1}$$

fue propuesta originalmente para describir el movimiento de las fronteras en las fases provocada por cristales sólidos que son impulsados por la tensión superficial. En este contexto, ϕ representa la concentración de uno de los dos componentes metálicos de la aleación, el parámetro ε representa el grosor de la interfase que es considerablemente muy pequeño en comparación con el resto del compuesto, donde

$$f(\phi) = \frac{1}{4}(1-\phi^2)^2. \tag{2.2}$$

Se espera que la solución de la ecuación de Allen-Cahn se comporte como una función constante en dos secciones una con valor de $\phi = 1$ y la otra con valo de $\phi = -11$ cuando el valor de ε es muy pequeño la interfase difusa es muy delgada.

2.2. Modelo de Cahn-Hilliard

El modelo de Cahn-Hilliard fue propuesto en 1958 [21] y permite estudiar el comportamiento de una mezcla homogénea entre dos sustancias mediante el uso de un parámetro de orden (ϕ), a este comportamiento se le conoce como proceso de difusión.

Siendo ϕ una concentración relativa de los componentes y $\phi(x) = \phi_m$ para algún valor de ϕ_m constante, se tiene que los perfiles de equilibrio se pueden encontrar mediante la minimización de la enenergía libre de

$$H[\phi] = \int_\Omega \left\{ f(\phi(\mathbf{x})) + \frac{1}{2}\varepsilon^2 |\nabla \phi(\mathbf{x})|^2 \right\} \mathbf{dx}, \tag{2.3}$$

sujeto a la restricción de la conservación de la masa

$$\int_\Omega \phi(\mathbf{x}) \mathbf{dx} = \phi_m \left|\Omega\right|, \tag{2.4}$$

donde Ω es la región del espacio ocupado por el sistema. El gradiente marca la energía de la

superficie y el espesor de la interfase es dado por ε, donde $f(\phi(\mathbf{x}))$ es la densidad de la energía.

Para obtener un modelo simétrico se toma la función de doble potencial

$$f(\phi) = \frac{1}{4}(1-\phi^2)^2. \tag{2.5}$$

El potencial químico μ está dado por la primera derivada variacional de (2.3)

$$\mu(\phi) = \frac{\delta H[\phi]}{\delta\phi(\mathbf{x})} = f'(\phi(\mathbf{x})) - \varepsilon^2\nabla^2\phi(\mathbf{x}) \tag{2.6}$$

y el estado de equilibrio está dado por la solución de $\mu(\phi) =$ constante. Cahn-Hilliard generalizan el problema a una situación dependiente del tiempo mediante la aproximación de la interfase entre flujos binarios que se obtiene por el gradiente del potencial químico para así hacer cumplir la conservación del campo donde

$$\frac{\partial\phi(t,\mathbf{x})}{\partial t} = -\nabla\cdot\mathbf{J} \quad con \quad \mathbf{J} = -M(\phi)\nabla\mu \tag{2.7}$$

y $M(\phi) > 0$ es el coeficiente de movilidad de Osanger, mediante este proceso se obtiene la ecuación de Cahn-Hilliard,

$$\frac{\partial\phi(t,x)}{\partial t} = \nabla\cdot\left[M(\phi)\nabla\mu(\phi)\right], \qquad \mu(\phi) = -\varepsilon^2\nabla^2\phi + f'(\phi), \quad para \quad x\in\Omega \tag{2.8}$$

La ec. (2.8) modela la creación, evolución y disolución de una interfase en un campo de fase difuso controlado. Las condiciones de frontera periódicas y de no flujo ($\mathbf{n}\cdot\nabla\phi = 0$ y $\mathbf{n}\cdot M\nabla\mu = 0$), se utilizan generalmente para la solución numérica de esta ecuación.

La ecuación de Cahn-Hilliard inicialmente fue utilizada para modelar la transición de fases en aleaciones binarias [21, 22], posteriormente se encontraron aplicaciones de este modelo a nanoescalas [23] y macroescalas [24], por ejemplo el uso de polímeros [25], el procesamiento de imágenes electromagnéticas [26] y dinámica planetaria [24]. Desde los primeros trabajos de Langer et al [27] los métodos numéricos para la ecuación Cahn-Hilliard han sido ampliamente estudiados entre los cuales destacan el método de diferencias finitas [28, 29], elementos finitos [30, 31], volumen finito [32, 33] y los métodos de Monte-Carlo [34]. También es posible modelar la separación de la macrofase que produce un fluido binario isotérmico cuando esta mezcla se inactiva por debajo de una temperatura crítica en la que se vuelve inestable.

En la ecuación de Cahn-Hilliard se tienen derivadas de cuarto orden y un operador Laplaciano en el término no lineal $f'(\phi)$, por lo tanto los métodos de integración de tiempo explícito requieren un paso pequeño en el tiempo para la estabilidad. Se han propuesto varias discretizaciones totalmente implícitas y semi-implícitas de tiempo para mejorar las limitantes en la

estabilidad de alto orden.

El cálculo del problema de valor inicial para la ec. (2.8) exige alta resolución en el espacio y en el tiempo. Espacialmente, la solución se desarrolla para microestructuras finas en tiempos cortos $O(\varepsilon^2)$ y un proceso de engrosamiento que separa las dos fases en una escala de tiempo mucho más largo $O(\varepsilon^{-1})$. Para modelar con precisión los sistemas experimentales reales, esta capa tiene que ser muy pequeña en relación con el tamaño del dominio, lo que implica que la solución tendrá gradientes de grosor $O(\varepsilon^{-1})$, por lo que el cálculo exacto de la solución requiere un enfoque adaptativo que sólo puede ser puesto en práctica si los intervalos de desplazamientos en el tiempo son pequeños.

La ecuación de Cahn-Hilliard también aparece en el modelado de varios fenómenos que van desde la evolución en la mezcla de los componentes de dos materiales, la dinámica de dos poblaciones hasta el modelado biomatemático de una película bacteriana [35]. También es utilizado en el análisis de separación de fases en el modelado de una mezcla binaria de metales. La evolución del parámetro de orden se forma por la aproximación al mínimo local de la energía libre que corresponde a la conservación de campo de fase y como resultado las capas de la interfase no deterioran su dinámica.

Otra de las aplicaciones que utilizan a los modelos de campo de fase es la formación de estructuras y evolución en los sistemas de flujo, un área de impacto tecnológico en el procesamiento de materiales blandos.

La ecuación de Cahn-Hilliard es difícil de resolver numéricamente por dos razones:

- Se requiere de un mallado fino porque la ecuación es de cuarto orden, esto provoca intervalos de tiempo pequeños para mantener la estabilidad ya que $\Delta t \sim \Delta x^4$ para métodos explícitos.

- La no linealidad en la función del potencial químico puede contribuir a una inestabilidad del proceso.

2.2.1. Equilibrio del modelo de Cahn-Hilliard

Para el caso de un sistema unidimensional, la solución es una función de paso centrado, lo que implica que las dos fases del fluido están separadas completamente por una interfase difusa. Las condiciones de frontera se definen de tal forma que el parámetro de orden y el potencial químico sean cero en ambos límites, esto también implica la conservación de la masa para las dos fases. La solución de estado estacionario dado en [15] es

$$\phi_{steady}(x) = \tanh\left(\frac{x}{\varepsilon\sqrt{2}}\right), \tag{2.9}$$

donde la condición inicial y el estado de equilibrio están representadas en la Fig. 2.1. La frontera del sistema unidemensional $\phi(x)$ esta dada por $\phi(x_0) = -1$ y $\phi(x_N) = 1$ mientras que la fase de separación se encuentra establecida en $x = 0$. La interfase es reubicada en el espacio para un lapso de tiempo posterior, pero esto no perjudica sus componentes ya que el valor de $x = 0$ se mantiene.

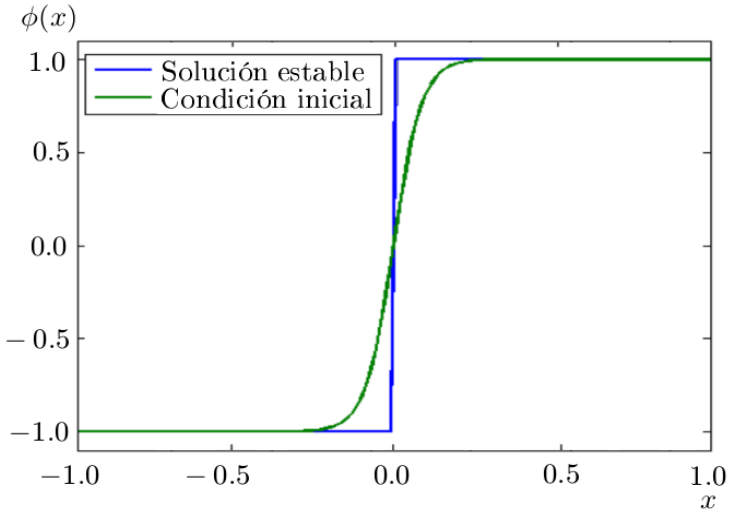


Fig. 2.1: Modelo unidimensional de Cahn-Hilliard con valor en la frontera de $\phi(-1) = -1$ y $\phi(1) = 1$ que evoluciona a un estado estacionario desde una condición inicial.

Capítulo 3

Solución numérica a modelos de interfase difusa

En este capítulo se describen diferentes técnicas de implementación numérica del modelo de Cahn-Hilliard utilizando un esquema discreto para el MDF.

El MDF consiste en obtener una solución numérica aproximada a la solución analítica de una EDP. Para el caso del modelo de Cahn-Hilliard, la función ϕ se discretiza utilizando una secuencia de puntos llamada malla. Dos parámentros importantes utilizados en el MDF son Δx, que representa la distancia de separación entre cada punto de la malla, y Δt, que es el incremento en el tiempo. En la Fig. 3.1 se muestra un ejemplo de discretización para una función en una dimensión con intervalos Δx y Δt. Para el desarrollo de esta tesis se consideran mallas unifomes con $\Delta x = \Delta y = \Delta z$.

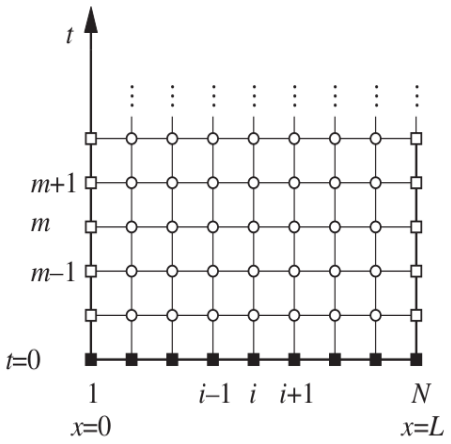


Fig. 3.1: Malla representativa en una dimensión, utilizada en la implementación del MDF para el modelo de Cahn-Hilliard.

Con este método se pueden resolver ecuaciones diferenciales utilizando distintos valores en la frontera e iniciales [6]. Para ello es necesario sustituir fórmulas obtenidas por el desarrollo de la serie de Taylor

$$\phi(x_i + \Delta x) = \phi(x_i) + \Delta x \frac{\partial \phi(x_i)}{\partial x} + \frac{\Delta x^2}{2} \frac{\partial \phi^2(x_i)}{\partial x^2} + \frac{\Delta x^3}{3!} \frac{\partial \phi^3(x_i)}{\partial^3 x} + \cdots \tag{3.1}$$

en cada punto de la malla, donde $\Delta x = x_{i+1} - x_i$.

El MDF ha sido implementado utilizando diferentes técnicas, por ejemplo, en [7] se resolvió el modelo de Cahn-Hilliard utilizando un esquema explícito y semi-implícito. El método explícito surge cuando se considera el valor de ϕ como condición inicial y se sustituye en las fórmulas del MDF. La ventaja de usar un esquema explícito es la simplicidad con que se trabaja en un programa de cómputo paralelo ya que cada punto de la malla puede se resuelto de manera independiente. La desventaja de esta técnica es que se requieren intervalos Δt muy pequeños. En un programa informático esto provoca muchas iteraciones para alcanzar un punto de equlibrio en el modelo.

El método implícito es una técnica numérica que permite encontrar una solución aproximada de una ecuación diferencial considerando la solución del modelo como un sistema de ecuaciones lineales $\mathbf{A}\phi = \mathbf{b}$, esto permite incrementar el tamaño de Δt y como consecuencia las iteraciones se reducen para llegar al punto de equilibrio. La desventaja de utilizar un esquema implícito es que aumenta la cantidad de procesamiento porque se almacenan valores para una matriz $\mathbf{A}$.

En cómputo paralelo, la solución de un sistema de ecuaciones es resuelto como un problema de optimización porque los métodos tradicionales tienen dependencias de datos y no es eficiente su paralelización. Hay otras técnicas resultantes de la combinación del método explícito e implícito [7]. El esquema semi-implícito se considera una técnica apropiada en la solución de modelos no lineales. Para el caso del modelo de Cahn-Hilliard, esto permite encontrar soluciones a través de un sistema de ecuaciones que resulta de la linealización del modelo en las derivadas espaciales.

3.1. Modelo de Cahn-Hilliard con el MDF

En esta sección se describe la implementación del modelo de Cahn-Hilliard utilizando el MDF. Para obtener una solución numérica se discretiza la ec. (2.8) en dominio de una, dos y tres dimensiones. En la formulación del modelo de Cahn-Hilliard se utiliza el esquema explícito y semi-implícito de Euler.

El modelo de Cahn-Hilliard es

$$\frac{\partial \phi}{\partial t} = \nabla \cdot \left[M(\phi) \nabla \mu(\phi) \right]. \tag{3.2}$$

Para el desarrollo de la presente tesis se considera a $M(\phi) = 1$ y se sustituye en la ec. (3.2) para tener

$$\frac{\partial \phi}{\partial t} = \nabla^2 \mu, \tag{3.3}$$

donde

$$\mu = f^{'}(\phi) - \varepsilon \nabla^2 \phi. \tag{3.4}$$

La solución numérica con el MDF para la ec. (3.2) puede resolverse de dos formas. La primera es con un operador *bi-armónico* que resulta de la formulación de la cuarta derivada y que se obtiene al sustituir la ec. (3.4) en la ec. (3.3)

$$\frac{\partial \phi}{\partial t} = -\varepsilon \nabla^4 \phi + \nabla^2 \left(\phi^3 - \phi \right). \tag{3.5}$$

La segunda solución está dada por la formulación de las ecs. (3.3) y (3.4) como un sistema de dos ecuaciones acopladas

$$\frac{\partial \phi}{\partial t} = \nabla^2 \mu, \tag{3.6}$$

$$\mu = f^{'}(\phi) - \varepsilon \nabla^2 \phi. \tag{3.7}$$

3.1.1. Modelo de Cahn-Hilliard como un sistema de ecuaciones con el método explícito de Euler

El primer método que se utiliza es un esquema explícito, es decir que cada instante de tiempo se calcula una nueva solución ϕ utilizando solo la información almacenada en el tiempo anterior. Para ello se discretiza la segunda derivada de orden espacial en la ec. (3.7) utilizando un esquema de diferencias centrales. La Fig. 3.2 presenta la discretización del método explícito.

En la ec. (3.6) se sustituye la derivada temporal

$$\frac{\partial \phi(x_i^m)}{\partial t} = \frac{\phi_i^{m+1} - \phi_i^m}{\Delta t}, \tag{3.8}$$

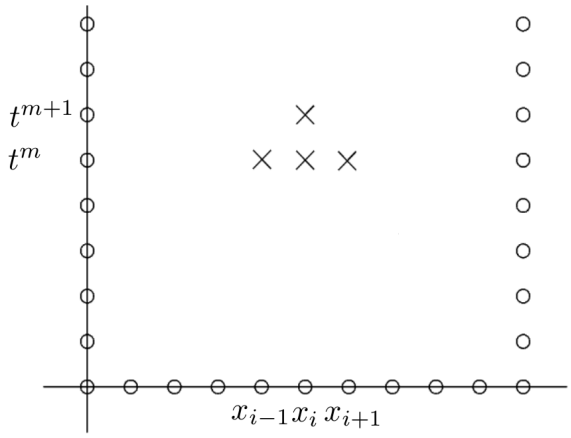


Fig. 3.2: Representación de un esquema discreto del método explícito, donde el nivel t^{m+1} y el nodo x_i es dependiente de los puntos x_{i-1}^m, x_i^m y x_{i+1}^m.

y se reemplaza a $\frac{\partial^2 \phi}{\partial x^2}$ en la derivada espacial de la (3.7) con

$$\frac{\partial^2 \phi(x_i^m)}{\partial x^2} = \frac{\phi_{i+1}^m - 2\phi_i^m + \phi_{i-1}^m}{\Delta x^2}. \tag{3.9}$$

Al sustituir las ecs. (3.8) y (3.9) en las ecs. (3.6) y (3.7) se tiene

$$\phi_i^{m+1} = \phi_i^m + \frac{\Delta t}{\Delta x^2}(\mu_{i-1}^m - 2\mu_i^m + \mu_{i+1}^m), \tag{3.10}$$

$$\mu_i^{m+1} = ((\phi_i^m)^3 - \phi_i^m) - \frac{\varepsilon^2}{\Delta x^2}(\phi_{i-1}^m - 2\phi_i^m + \phi_{i+1}^m). \tag{3.11}$$

En dos dimensiones se desarrolla la solución de forma similar que en una dimensión. El modelo discreto de Cahn-Hilliard en dos dimensiones es

$$\phi_{i,j}^{m+1} = \phi_{i,j}^m + \frac{\Delta t}{\Delta x^2}(\mu_{i-1,j}^m + \mu_{i,j+1}^m - 4\mu_{i,j}^m + \mu_{i,j+1}^m + \mu_{i+1,j}^m), \tag{3.12}$$

$$\mu_{i,j}^m = ((\phi_{i,j}^m)^3 - \phi_{i,j}^m) - \frac{\varepsilon}{\Delta x^2}(\phi_{i-1,j}^m + \phi_{i,j-1}^m - 4\phi_{i,j}^m + \phi_{i,j+1}^m + \phi_{i+1,j}^m). \tag{3.13}$$

Sucede lo mismo con el modelo de Cahn-Hilliard en tres dimensiones, se continúa con el

mismo procedimiento y la formulación en tres dimensiones es

$$\begin{aligned}\phi_{i,j,k}^{m+1} =&\phi_{i,j,k}^{m} + \frac{\Delta t}{\Delta x^2}(\mu_{i-1,j,k}^{m} + \mu_{i,j-1,k}^{m} + \mu_{i,j,k-1}^{m} - 6\mu_{i,j,k}^{m} + \mu_{i,j+1,k}^{m} + \\ &\mu_{i+1,j,k}^{m} + \mu_{i,j,k+1}^{m}), \qquad (3.14)\\ \mu_{i,j,k}^{m} =&((\phi_{i,j,k}^{m})^3 - \phi_{i,j,k}^{m}) - \frac{\varepsilon}{\Delta x^2}(\phi_{i-1,j,k}^{m} + \phi_{i,j-1,k}^{m} + \phi_{i,j,k-1}^{m} - 6\phi_{i,j,k}^{m} + \\ &\phi_{i,j+1,k}^{m} + \phi_{i+1,j,k}^{m} + \phi_{i,j,k+1}^{m}). \qquad (3.15)\end{aligned}$$

3.1.2. Modelo de Cahn-Hilliard como un sistema de ecuaciones con el método semi-implícito de Euler

Un esquema implícito del MDF consiste en encontrar la solución a una función $\phi(x)$ calculando el resultado en un mismo instante de tiempo, es decir que cada intervalo Δt tiene información numérica de $\Delta(t+1)$ y $\Delta(t-1)$, pero la solución de la derivadas espaciales se expresa solo en intervalos $\Delta(t+1)$. Esto hace que una solución para la función ϕ necesite resolver un sistema de ecuaciones $\mathbf{Ax} = \mathbf{b}$. En la Fig 3.3 se presenta una solución para un elemento x_i en un tiempo t^m, ahí se necesita conocer los valores de x_{i-1}, x_i y x_{i+1} en el tiempo t^{m+1}.

El esquema semi-implícito de Euler es una técnica de linealización entre el $\Delta\mu$ y el término de $\Delta\phi$ en el sistema de ecs. (3.7). Por lo tanto se evalúa $\frac{\partial\phi}{\partial t}$ en el tiempo m con un esquema de diferencias finitas hacia adelante y las derivadas espaciales se resuleven con diferencias finitas centradas en el tiempo $m+1$.

La formulación de la ec. (3.7) para una dimensión es

$$\phi_i^{m+1} - \frac{\Delta t}{\Delta x^2}(\mu_{i-1}^{m+1} - 2\mu_i^{m+1} + \mu_{i+1}^{m+1}) = \phi_i^m, \qquad (3.16)$$

$$\mu_i^{m+1} + \frac{\varepsilon^2}{\Delta x^2}(\phi_{i-1}^{m+1} - 2\phi_i^{m+1} + \phi_{i+1}^{m+1}) = ((\phi_i^m)^3 - \phi_i^m). \qquad (3.17)$$

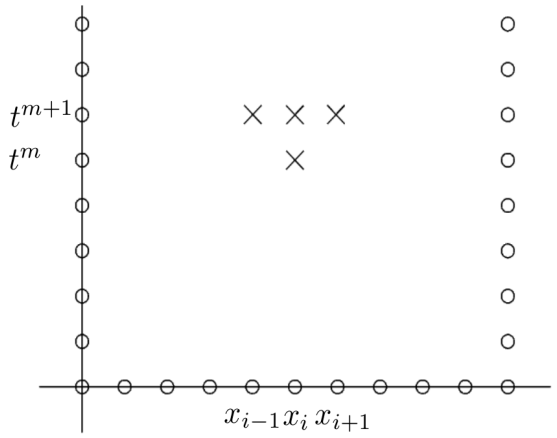


Fig. 3.3: La dependencia de un sistema discreto hacia atrás varía con la dependendencia del método explícito, ahora los valores del estado de tiempo t^m dependen de los valores de t^{n+1}, los cuales son desconocidos.

Para dos dimensiones se utiliza la misma técnica empleada en el modelo de una dimensión. La formulación del método semi-implícito en el modelo de Cahn-Hilliard en dos dimensiones es

$$\phi_{i,j}^{m+1} - \frac{\Delta t}{\Delta x^2}(\mu_{i-1,j}^{m+1} + \mu_{i,j-1}^{m+1} - 4\mu_{i,j}^{m+1} + \mu_{i,j+1}^{m+1} + \mu_{i+1,j}^{m+1}) = \phi_{i,j}^{m}, \tag{3.18}$$

$$\mu_{i,j}^{m+1} + \frac{\varepsilon^2}{\Delta x^2}(\phi_{i-1,j}^{m+1} + \phi_{i,j-1}^{m+1} - 4\phi_{i,j}^{m+1} + \phi_{i+1,j}^{m+1} + \phi_{i,j+1}^{m+1}) = ((\phi_{i,j}^{m})^3 - \phi_{i,j}^{m}). \tag{3.19}$$

La ec. (3.7) utilizando diferencias finitas centradas en las derivadas espaciales y diferencias finitas hacia adelante en la derivada temporal. La formulación del modelo de Cahn-Hilliard en tres dimensiones es

$$\begin{aligned}\phi_{i,j,k}^{m+1} - \frac{\Delta t}{\Delta x^2}(&\mu_{i-1,j,k}^{m+1} + \mu_{i,j-1,k}^{m+1} + \mu_{i,j,k-1}^{m+1} - 6\mu_{i,j,k}^{m+1} + \\ &\mu_{i,j+1,k}^{m+1} + \mu_{i+1,j,k}^{m+1} + \mu_{i,j,k+1}^{m+1}) = \phi_{i,j,k}^{m},\end{aligned} \tag{3.20}$$

$$\begin{aligned}\mu_{i,j,k}^{m+1} + \frac{\varepsilon^2}{\Delta x^2}(&\phi_{i-1,j,k}^{m+1} + \phi_{i,j-1,k}^{m+1} + \phi_{i,j,k-1}^{m+1} - 6\phi_{i,j,k}^{m+1} + \\ &\phi_{i+1,j,k}^{m+1} + \phi_{i,j+1,k}^{m+1} + \phi_{i,j,k+1}^{m+1}) = ((\phi_{i,j,k}^{m})^3 - \phi_{i,j,k}^{m}).\end{aligned} \tag{3.21}$$

3.1.3. Modelo de Cahn-Hilliard con operador biarmónico con el método explícito de Euler

Anteriormente se explicó que este método es una técnica sencilla de implementar porque que el cálculo solo depende de una solución en el tiempo t_0 para conocer un resultado en el tiempo t_1.

Para la formulación del modelo de Cahn-Hilliard en una dimensión con el MDF se desarrolla la ec. (3.5) como

$$\frac{\partial \phi}{\partial t} = -\varepsilon \frac{\partial^4 \phi}{\partial x^4} + \frac{\partial^2}{\partial x^2}(\phi^3 - \phi), \tag{3.22}$$

en la cual se aplican diferencias finitas hacia adelante en la derivada temporal y diferencias centradas en la derivada espacial. El desarrollo numérico de la ec. (3.22) esta dado por

$$\begin{aligned}\frac{\phi_i^{m+1} - \phi_i^m}{\Delta t} =& -\varepsilon^2 \left[\frac{\phi_{i-2}^m - 4\phi_{i-1}^m + 6\phi_i^m - 4\phi_{i+1}^m + \phi_{i+2}^m}{\Delta x^4}\right] \\ &+ \frac{(\phi_{i-1}^m)^3 - 2(\phi_i^m)^3 + (\phi_{i+1}^m)^3}{\Delta x^2} - \frac{\phi_{i-1}^m - 2\phi_i^m + \phi_{i+1}^m}{(\Delta x)^2},\end{aligned} \tag{3.23}$$

en donde se desea conocer el valor de $\phi_i^{(m+1)}$ y se obtiene que

$$\begin{aligned}\phi_i^{m+1} = \phi_i^m - \frac{\varepsilon \Delta t}{\Delta x^4}\phi_{i-2}^m - 4\phi_{i-1}^m + 6\phi_i^m - 4\phi_{i+1}^m + \phi_{i+2}^m + \\ \frac{\Delta t}{\Delta x^2}\left[\left(\phi_{i-1}^m\right)^3 - 2(\phi_i^m)^3 + (\phi_{i+1}^m)^3 - \phi_{i-1}^m - 2\phi_i^m + \phi_{i+1}^m\right].\end{aligned} \tag{3.24}$$

Para la solución numérica en dos dimensiones se tiene un término cruzado que resulta de resolver la cuarta derivada espacial en dos direcciones. A este término se le conoce como operador bi-armónico y para dos dimensiones esta dado por

$$\nabla^4 \phi(x,y) = \frac{\partial^4 \phi(x,y)}{\partial x^4} + \frac{\partial^4 \phi(x,y)}{\partial y^4} + 2\frac{\partial^4 \phi(x,y)}{\partial x^2 \partial y^2,}, \tag{3.25}$$

que al sustituir en la ec. (3.5) se tiene

$$\begin{aligned}\phi_{i,j}^{m+1} = \phi_{i,j}^m - \frac{\varepsilon \Delta t}{\Delta x^4}\left(\phi_{i-2,j}^m - 4\phi_{i-1,j}^m + 12\phi_{i,j}^m - 4\phi_{i+1,j}^m + \phi_{i+2,j}^m + \right. \\ \left.\phi_{i,j-2}^m - 4\phi_{i,j-1}^m - 4\phi_{i,j+1}^m + \phi_{i,j+2}^m\right) + \\ \frac{2\Delta t}{\Delta x^4}\left[\left(\phi_{i-1,j-1}^m - 2\phi_{i,j-1}^m + \phi_{i+1,j-1}^m\right) - 2\left(\phi_{i-1,j}^m - 2\phi_{i,j}^m + \phi_{i+1,j}^m\right) + \right. \\ \left.\left(\phi_{i-1,j+1}^m - 2\phi_{i,j+1}^m + \phi_{i+1,j+1}^m\right)\right] + \\ \frac{\Delta t}{\Delta x^2}\left[\left(\phi_{i-1,j}^m\right)^3 - 2(\phi_{i,j}^m)^3 + (\phi_{i+1,j}^m)^3 - \left(\phi_{i-1,j}^m - 2\phi_{i,j}^m + \phi_{i+1,j}^m\right) + \right. \\ \left.\left(\phi_{i,j-1}^m\right)^3 - 2(\phi_{i,j}^m)^3 + (\phi_{i,j+1}^m)^3 - \left(\phi_{i,j-1}^m - 2\phi_{i,j}^m + \phi_{i,j+1}^m\right)\right].\end{aligned} \tag{3.26}$$

En el caso de tres dimensiones se tiene que el operador bi-armónico contiene tres derivadas cruzadas

$$\nabla^4 \phi = \frac{\partial^4 \phi}{\partial x^4} + \frac{\partial^4 \phi}{\partial y^4} + \frac{\partial^4 \phi}{\partial z^4} + 2\frac{\partial^4 \phi}{\partial x^2 \partial y^2} + 2\frac{\partial^4 \phi}{\partial y^2 \partial z^2} + 2\frac{\partial^4 \phi}{\partial x^2 \partial z^2}, \tag{3.27}$$

esto incrementa el procesamiento del modelo en un programa informático. Se continúa con el desarrollo del modelo utilizando las fórmulas de diferencias centradas en las derivadas espaciales y en la derivada temporal se sustituyen los términos por diferencias finitas hacia adelante. La discretización del modelo se obtiene como

$$\begin{aligned}
\phi_{i,j,k}^{m+1} = \phi_{i,j,k}^{m} - \frac{\varepsilon\Delta t}{\Delta x^4}\big(&\phi_{i-2,j,k}^{m} - 4\phi_{i-1,j,k}^{m} + 18\phi_{i,j,k}^{m} - 4\phi_{i+1,j,k}^{m} + \phi_{i+2,j,k}^{m} + \\
&\phi_{i,j-2,k}^{m} - 4\phi_{i,j-1,k}^{m} - 4\phi_{i,j+1,k}^{m} + \phi_{i,j+2,k}^{m} + \phi_{i,j,k-2}^{m} - 4\phi_{i,j,k-1}^{m} - 4\phi_{i,j,k+1}^{m} + \phi_{i,j,k+2}^{m}\big) + \\
&\frac{2\Delta t}{\Delta x^4}\big[\big(\phi_{i-1,j-1,k}^{m} - 4\phi_{i,j-1,k}^{m} + \phi_{i+1,j-1,k}^{m}\big) - 2\big(2\phi_{i-1,j,k}^{m} - 6\phi_{i,j,k}^{m} + 2\phi_{i+1,j,k}^{m}\big) + \\
&\big(\phi_{i-1,j+1,k}^{m} - 4\phi_{i,j+1,k}^{m} + \phi_{i+1,j+1,k}^{m}\big) + \big(\phi_{i-1,j,k-1}^{m} - 4\phi_{i,j,k-1}^{m} + \phi_{i+1,j,k-1}^{m}\big) + \\
&\big(\phi_{i-1,j,k+1}^{m} - 4\phi_{i,j,k+1}^{m} + \phi_{i+1,j,k+1}^{m}\big)\big(\phi_{i,j-1,k-1}^{m} + \phi_{i,j+1,k-1}^{m}\big) + \big(\phi_{i,j-1,k+1}^{m} + \phi_{i,j+1,k+1}^{m}\big)\big] + \\
&\frac{\Delta t}{\Delta x^2}\Big[\big(\phi_{i-1,j,k}^{m}\big)^3 - 6(\phi_{i,j,k}^{m})^3 + (\phi_{i+1,j,k}^{m})^3 - \big(\phi_{i-1,j,k}^{m} - 6\phi_{i,j,k}^{m} + \phi_{i+1,j,k}^{m}\big) + \\
&\big(\phi_{i,j-1,k}^{m}\big)^3 + (\phi_{i,j+1,k}^{m})^3 - \big(\phi_{i,j-1,k}^{m} + \phi_{i,j+1,k}^{m}\big) + \\
&\big(\phi_{i,,j,k-1}^{m}\big)^3 + (\phi_{i,j,k+1}^{m})^3 - \big(\phi_{i,j,k-1}^{m} + \phi_{i,j,k+1}^{m}\big)\Big].
\end{aligned} \tag{3.28}$$

3.1.4. Condiciones de frontera

Para resolver una ecuación diferencial se debe tomar en cuenta las condiciones de frontera que pertenecen al problema. Generalmente estas condiciones son utilizadas en diferentes aspectos de la física. En el desarrollo de la presente tesis se resuelven problemas con condiciones de frontera de tipo Dirichlet, Neumann y periódicas.

El uso de condiciones de frontera periódicas es común cuando se necesita modelar un fragmento dentro de un sistema grande y que éste se encuentre lejos de la frontera. La dinámica molecular es un ejemplo donde se utilizan las condiciones de frontera periódicas para simular acumulaciones de gas, líquidos, cristales, mezclas, etc [6]. De igual forma se puede dar solución al modelo Cahn-Hilliard utilizando condiciones de frontera de no flujo (Neumann).

Condición de frontera de Neumann para el modelo de Cahn-Hilliard como sistema de ecuaciones

A continuación se presenta el desarrollo de las fórmulas que se aplican en la frontera para el modelo de Cahn-Hilliard con condiciones de frontera de Neumann. Estas fórmulas se obtienen de la ecs. (3.7) donde se considera a $\phi = 0$ en la primera ecuación

$$\Delta\phi(x_0) = \frac{\partial\phi(x_0)}{\partial x} \qquad \therefore \qquad \frac{\partial\phi(x_0)}{\partial x} = 0, \tag{3.29}$$

entonces

$$\frac{\phi_1 - \phi_{-1}}{2\Delta x} = 0, \tag{3.30}$$

donde ϕ_{-1} es un elemento de la malla que no existe. Es este punto el que se toma para establecer la condición de frontera. Para encontrar el valor de ϕ_{-1} es necesario despejarla y por lo tanto la condición de frontera en un punto ϕ_0 es

$$\phi_{-1} = \phi_1, \tag{3.31}$$

también se considera a $\Delta\mu = 0$ como condición de frontera para las ecs. (3.7). La fórmula de μ_0 está dada por

$$\mu_{-1} = \mu_1. \tag{3.32}$$

Lo mismo ocurre en el otro extremo de la malla, es decir en el nodo con subíndice N de la malla. La condición de frontera para ϕ y μ estan dadas respectivamente por

$$\phi_{N+1} = \phi_{N-1} \qquad y \qquad \mu_{N+1} = \mu_{N-1}. \tag{3.33}$$

Las condiciones de frontera para todos los nodos con x_0 en el tiempo $m+1$ estan dado por

$$\phi_0^{m+1} = \phi_0^m + \frac{\Delta t}{\Delta x^2}(2\mu_1^m - 2\mu_0^m) \tag{3.34}$$

$$\mu_0^{m+1} = ((\phi_0^m)^3 - \phi_0^m) - \frac{\varepsilon^2}{\Delta x^2}(2\phi_1^m - 2\phi_0^m), \tag{3.35}$$

mientras que para los nodos de x_N y para el tiempo $m+1$ de la malla estan expresados como

$$\phi_N^{m+1} = \phi_N^m + \frac{\Delta t}{\Delta x^2}(2\mu_{N-1}^m - 2\mu_N^m) \tag{3.36}$$

$$\mu_N^{m+1} = ((\phi_N^m)^3 - \phi_N^m) - \frac{\varepsilon^2}{\Delta x^2}(2\phi_{N-1}^m - 2\phi_N^m). \tag{3.37}$$

Condición de frontera de Neumann para el modelo de Cahn-Hilliard con operador bi-armónico

Para la formulación de las condiciones de frontera del modelo de Cahn-Hilliard en dos dimensiones se considera la ec. (3.26). Al igual que en el caso anterior el valor de los términos $\phi_{-2,j}$ y $\phi_{-1,j}$ no se puede calcular de forma directa porque estos puntos no se encuentran explícitos en la malla. Para este caso particular se tomará como condición de frontera $\frac{\partial^3\phi}{\partial x^3} = 0$ y $\frac{\partial\phi}{\partial x} = 0$.

La solución para la condición $\frac{\partial\phi}{\partial x} = 0$ en los nodos donde $i = 0$ en la malla se presenta en la

ec. (3.31). Para

$$\frac{\partial^3 \phi}{\partial x^3} = 0, \tag{3.38}$$

y sustituyendo $\phi_{-1} = \phi_1$ en la fórmula de la tercera derivada se tiene

$$\frac{-\phi_{-2,j} + \phi_{2,j}}{2\Delta x^3} = \frac{\partial^3 \phi(x_0, y_j)}{\partial x^3} \quad \therefore \quad \frac{-\phi_{-2,j} + \phi_{2,j}}{2\Delta x^3} = 0, \tag{3.39}$$

que al despejar el valor de $\phi_{-2,j}$ se obtiene

$$\phi_{-2,j} = \phi_{2,j} \tag{3.40}$$

en los elemento donde $i = 0$ en dos dimensiones. De igual forma sucede para los nodos $i = N$, $j = 0$ y $j = N$. Las fórmulas para cada condición de frontera estan dadas respectivamente por

$$\phi_{i,-2} = \phi_{j,2}, \quad \phi_{N+1,j} = \phi_{N-1,j}, \quad y \quad \phi_{i,N+1} = \phi_{i,N-1}. \tag{3.41}$$

Para la formulación del término cruzado en un tiempo m de la ec (3.26) se considera que la discretización con el MDF está expresado como

$$2\frac{\partial^4 \phi(x,y)}{\partial x^2 \partial y^2} = \frac{2}{\Delta x^4}\left[(\phi^m_{i-1,j-1} - 2\phi^m_{i,j-1} + \phi^m_{i+1,j-1}) - \right. \tag{3.42}$$

$$\left. 2(\phi^m_{i-1,j} - 2\phi^m_{i,j} + \phi^m_{i+1,j}) + (\phi^m_{i-1,j+1} - 2\phi^m_{i,j+1} + \phi^m_{i+1,j+1})\right], \tag{3.43}$$

que al sustituir el valor dado por la ec. (3.31) en la ec. (3.43) se tiene que las condiciones de frontera para los términos cruzados de la malla en un tiempo m son

$$2\frac{\partial^4 \phi(x_0,y_j)}{\partial x^2 \partial y^2} = \frac{4}{\Delta x^4}\left[(\phi^m_{1,j-1} - \phi^m_{0,j-1}) - 2(\phi^m_{1,j} - \phi^m_{0,j}) + (\phi^m_{1,j+1} - \phi^m_{0,j+1})\right], \tag{3.44}$$

$$2\frac{\partial^4 \phi(x_N,y_j)}{\partial x^2 \partial y^2} = \frac{4}{\Delta x^4}\left[(\phi^m_{N-1,j-1} - \phi^m_{N,j-1}) - 2(\phi^m_{N-1,j} - \phi^m_{N,j}) + (\phi^m_{N-1,j+1} - \phi^m_{N,j+1})\right], \tag{3.45}$$

$$2\frac{\partial^4 \phi(x_i,y_0)}{\partial x^2 \partial y^2} = \frac{4}{\Delta x^4}\left[(\phi^m_{i-1,1} - \phi^m_{i-1,0}) - 2(\phi^m_{i,1} - \phi^m_{i,0}) + (\phi^m_{i+1,1} - \phi^m_{i+1,0})\right], \tag{3.46}$$

$$2\frac{\partial^4 \phi(x_i,y_N)}{\partial x^2 \partial y^2} = \frac{4}{\Delta x^4}\left[(\phi^m_{i-1,N-1} - \phi^m_{i-1,N}) - 2(\phi^m_{i,N-1} - \phi^m_{i,N}) + (\phi^m_{i+1,N-1} - \phi^m_{i+1,N})\right]. \tag{3.47}$$

3.2. Método del gradiente conjugado

El método del gradiente conjugado lineal fue propuesto por Hestenes y Stiefel en la década de 1950 como un método iterativo para resolver un sistema lineal representado con una ma-

triz simétrica y definida positiva. El rendimiento del método del gradiente conjugado lineal se determina por la distribución de los eigenvalores de la matriz de coeficientes [36].

El uso del método del gradiente conjugado cada vez es más interesante ya que esta técnica es utilizada para dar solución a grandes sistemas de ecuaciones lineales, este método es adaptable para obtener resultados a problemas de optimización no lineales. Al aplicar un precondicionamiento a la matriz se obtiene una distribución más adecuada y se mejora significativamente la convergencia del método.

El primer método del gradiente conjugado no lineal fue presentado por Fletcher y Reeves en la década de 1960. Es una de las técnicas más antiguas conocidas para resolver grandes problemas de optimización no lineales [36].

El método del gradiente conjugado es considerado como una técnica iterativa utilizada para resolver sistemas de ecuaciones $\mathbf{A}x = \mathbf{b}$, este sistema es comparado con un problema de minimización de un sistema de funciones cuadráticas convexas

$$\text{mín}\, f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}, \tag{3.48}$$

donde $\mathbf{x}$, $\mathbf{b} \in \mathbb{R}^n$ y $\mathbf{A} \in \mathbb{R}^{n\times n}$ es una matriz simétrica positiva definida. Para minimizar la función $f(\mathbf{x})$ se calcula el gradiente y que resulta en $f(\mathbf{x})$ como el residual de $\mathbf{x}$

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{r}(\mathbf{x}), \tag{3.49}$$

que para $x = x_k$

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k. \tag{3.50}$$

Una de las propiedades del método del gradiente conjugado es la capacidad de generar de manera económica un conjunto de vectores con una propiedad conocida como conjugación. Un conjunto de vectores no nulos $[\mathbf{p}_0, \mathbf{p}_1, \cdots, \mathbf{p}_N]$ se dice que son conjugados con respecto a la matriz $\mathbf{A}$ simétrica definida positiva

$$\mathbf{p}_i^T\mathbf{A}\mathbf{p}_j = 0, \qquad para \quad toda \quad i \neq j. \tag{3.51}$$

La importancia de la conjugación está en la posibilidad de reducir al mínimo una función $f(\cdot)$ en N pasos minimizando sucesivamente a lo largo de las direcciones individuales. Para comprobar esta afirmación se considera el método de direcciones conjugadas. Dado un punto de partida $\mathbf{x}_0 \in \mathbb{R}^n$ y un conjunto de direcciones conjugadas $[\mathbf{p}_0,\ \mathbf{p}_1, \cdots, \mathbf{p}_{N-1}]$ se genera una sucesión $\mathbf{x}_k$ a partir de

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k, \tag{3.52}$$

donde α_k es un minimizador unidimensional para funciones cuadráticas $f(\cdot)$ a lo largo de $\mathbf{x}_k + \alpha\mathbf{p}_k$, dado explícitamente como

$$\alpha_k = -\frac{\mathbf{r}_k^T\mathbf{p}}{\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k}. \tag{3.53}$$

Si se define $\mathbf{p}_{k+1}$ como la dirección más cercana al gradiente $\mathbf{r}_k$ bajo la restricción de ser conjugado. Esta dirección está dada por la proyección de $\mathbf{r}_k$ en el espacio ortogonal a $\mathbf{p}_k$ con respecto al producto interno inducido por $\mathbf{A}$, así

$$\mathbf{p}_{k+1} = -\mathbf{r}_k + \beta\mathbf{p}_k, \tag{3.54}$$

donde

$$\beta = \frac{\mathbf{p}_k^T\mathbf{A}\mathbf{r}_k}{\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k}. \tag{3.55}$$

En el algoritmo del gradiente conjugado se observa que la matriz $\mathbf{A}$ aparece repetidamente en el cálculo de las ecs. (3.53) y (3.55), esto provoca un intenso gasto en tiempo ya que el algoritmo requiere de repetidas operaciones matriz-vector para obtener un resultado. El valor de α de la ec. (3.52) puede plantearse como un método de busqueda directa

$$\alpha_0 = \underset{\alpha\geq 0}{arg\,min}\, f(\mathbf{x}_{(0)} + \alpha\mathbf{p}_{(0)}). \tag{3.56}$$

A través de los años, se han propuesto diferentes versiones del método del gradiente conjugado original. Las principales características para estos algoritmos es disminuir el uso de las operaciones vector-matriz.

- *Hestenes-Stiefel* afirman la posibilidad encontrar el valor de β en la ec. (3.55) sin verse en la necesidad de realizar repetidamente operaciones matriz-vector utilizando el teorema de inducción del algoritmo de direcciones conjugadas [6]

  $$\mathbf{r}_{k+1} = \mathbf{r}_k + \alpha\mathbf{A}\mathbf{p}_k, \tag{3.57}$$

  del cual se despeja $\mathbf{A}\mathbf{p}_k$ y se sustituye en la ec. (3.55)

  $$\beta_k = \frac{\mathbf{r}_{k+1}^T[\mathbf{r}_{k+1} - \mathbf{r}_k]}{\mathbf{p}_k^T[\mathbf{r}_{k+1} - \mathbf{r}_k]}. \tag{3.58}$$

- *Polak-Ribiere* plantean mejorar la ec. (3.58) tomando como referencia el producto resultante del denominador considerando el teorema de ortogonalidad del residuo del algoritmo de direcciones conjugadas [6]

  $$\mathbf{p}_{(k+1)}^T\mathbf{r}_{k+1} = 0, \tag{3.59}$$

y recordando el valor de $\mathbf{p}_{(k+1)}^T$ en la ec. (3.54) se tiene que

$$\mathbf{p}_k^T \mathbf{r}_k = -\mathbf{r}_k^T \mathbf{r}_k + \beta_{k-1} \mathbf{r}_k^T \mathbf{p}_{k-1} = -\mathbf{r}_k^T \mathbf{r}_{(k)}, \tag{3.60}$$

el cual se sustituye en el denominador de la ec. (3.58) y se obtiene que

$$\beta_k = \frac{\mathbf{r}_{(k+1)}^T [\mathbf{r}_{(k+1)} - \mathbf{r}_{(k)}]}{\mathbf{r}_k^T \mathbf{r}_k}. \tag{3.61}$$

- *Fletcher-Reeves* utilizan a la ec. (3.61) para dar inicio a una nueva propuesta para el gradiente conjugado, en esta ocasión se basan en el producto de numerador de la fórmula [6]

$$\beta_k = \frac{r_{k+1}^T r_k - r_{k+1} r_k}{\mathbf{r}_k^T \mathbf{r}_k}, \tag{3.62}$$

dando por hecho que $r_{k+1}^T r_k = 0$ se tiene que

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}. \tag{3.63}$$

El seudocódigo del método del gradiente conjugado se presenta en el Algoritmo 1.

Algoritmo 1 Algoritmo del gradiente conjugado.

$\mathbf{x}_0$, *Condición inicial*
$\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0$, *Residual inicial*
$\mathbf{p}_0 \leftarrow \mathbf{r}_0$, *Dirección de descenso inicial*
$k \leftarrow 0$
while $\mathbf{r}_k \neq 0$ **do**
  $\mathbf{w} \leftarrow \mathbf{A}\mathbf{p}_k$
  $\alpha_k \leftarrow \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{w}_k}$
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$
  $\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \alpha_k \mathbf{w}$
  $\beta_k \leftarrow \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$
  $\mathbf{p}_{k+1} \leftarrow -\mathbf{r}_{k+1} + \beta_{k+1} \mathbf{p}_k$
  $k \leftarrow k + 1$
end while

3.3. Método del gradiente biconjugado

Este método está basado en la técnica del gradiente conjugado y de igual forma sirve para resolver sistemas de ecuaciones lineales $\mathbf{Ax}$=$\mathbf{b}$, con la diferencia que $A \in \mathbb{R}^{N \times N}$ no tiene que

ser simétrica, este método requiere calcular un pseudo-residual $\widetilde{r}_k$ y una pseudo-dirección de descenso $\widetilde{p}_k$. El método se construye de tal forma que los pseudo-residuales $\widetilde{r}_k$ sean ortogonales a los residuales g_k y que las pseudo-direcciónes de descenso $\widetilde{p}_k$ sean ortogonales a la matriz $\mathbf{A}$ en la dirección de descenso p_k.

Si la matriz $\mathbf{A}$ es simétrica, entonces el método es equivalente al gradiente conjugado, esto no garantiza la convergencia del método en N pasos como lo hace el gradiente conjugado debido a que se requiere hacer dos multiplicaciones matriz-vector [36]. El seudocódigo del método del gradiente bi-conjugado se muestra en el Algoritmo 2.

Algoritmo 2 Algoritmo del gradiente bi-conjugado.

$\mathbf{x}_0$, *Condición inicial*
$\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0$, *Residual inicial*
$\widetilde{\mathbf{r}}_0 \leftarrow \mathbf{r}_0$, *Pseudo-residual inicial*
$\mathbf{p}_0 \leftarrow \mathbf{r}_0$, *Dirección de descenso inicial*
$\widetilde{\mathbf{p}}_0 \leftarrow \mathbf{p}_0$, *Pseudo-dirección de descenso inicial*
$k \leftarrow 0$
while $\mathbf{r}_k \neq 0$ **do**
  $\mathbf{w} \leftarrow \mathbf{A}\mathbf{p}_k$
  $\widetilde{\mathbf{w}}_k \leftarrow \mathbf{A}^T\widetilde{\mathbf{p}}_k$
  $\alpha_k \leftarrow \frac{\widetilde{\mathbf{r}}_k^T \mathbf{r}_k}{\widetilde{\mathbf{p}}_k^T \mathbf{w}_k}$
  $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$
  $\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \alpha_k \mathbf{w}$
  $\widetilde{\mathbf{r}}_{k+1} \leftarrow \widetilde{\mathbf{r}}_k - \alpha_k \widetilde{\mathbf{w}}_k$
  $\beta_k \leftarrow \frac{\widetilde{\mathbf{r}}_{k+1}^T \mathbf{r}_{k+1}}{\widetilde{\mathbf{r}}_k^T \mathbf{r}_k}$
  $\mathbf{p}_{k+1} \leftarrow -\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k$
  $\widetilde{\mathbf{p}}_{k+1} \leftarrow -\widetilde{\mathbf{r}}_{k+1} + \beta_{k+1}\widetilde{\mathbf{p}}_k$
  $k \leftarrow k + 1$
end while

3.4. Descomposición de dominio

El uso de métodos numéricos en la solución de EDP consume muchos recursos computacionales. Para el caso del MDF, entre más grande sea el tamaño de la malla, el procesamiento será mayor y por lo tanto el consumo de espacio en la memoria RAM se incrementa considerablemente. El uso de algoritmos paralelos permite mejorar la limitante del procesamiento de información en equipos tradicionales utilizando así las arquitecturas *multicore*[37].

Una de las mayores dificultades del procesamiento en paralelo es la coordinación de actividades entre los diferentes procesadores, así como el intercambio de información entre cada actividad [38]. Al usar métodos de descomposición de dominio conjuntamente con arquitecturas paralelas es posible resolver estos problemas de manera más eficiente en varios subdominios [39].

3.4.1. Método de Schwarz

Para describir el método, se considera un dominio Ω que está formado de dos subdominios Ω_1 y Ω_2 traslapados, es decir $\Omega_1 \cap \Omega_2 \neq \emptyset$, entonces $\Omega = \Omega_1 \cup \Omega_2$ y se denota a $\sum_1 = \partial\Omega_1 \cap \Omega_2$, $\sum_2 = \partial\Omega_2 \cap \Omega_1$ y $\Omega_{1,2} = \Omega_1 \cap \Omega_2$, como se muestra en la Fig. 3.4 para dos dominios distintos

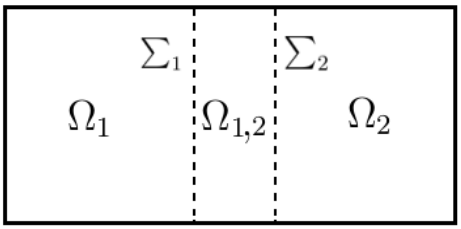


Fig. 3.4: Dominio Ω dividido en dos subdominios Ω_1 y Ω_2

La forma original del método iterativo de Schwarz consiste en resolver sucesivamente los siguientes problemas. Sea ϕ_0 una función inicializada y definida en Ω, que se nulifica en $\partial\Omega$, además se hace $\phi_1^0 = \phi(\Omega_1)^0$ y $\phi_2^0 = \phi(\Omega_2)^0$. Para $m \geq 0$ se definen dos sucesiones ϕ_1^{m+1} y ϕ_2^{m+1} para resolver respectivamente

$$\begin{cases} \phi_1^{m+1} = & f(\Omega_1) & en & \Omega_1 \\ \phi_1^{m+1} = & \phi_2^{m+1} & en & \sum_1 \\ \phi_1^{m+1} = & 0 & en & \partial\Omega_1 \cap \Omega \end{cases}, \tag{3.64}$$

y

$$\begin{cases} \phi_2^{m+1} = & f(\Omega_2) & en & \Omega_2 \\ \phi_2^{m+1} = & \phi_1^{m+1} & en & \sum_2 \\ \phi_2^{m+1} = & 0 & en & \partial\Omega_2 \cap \Omega \end{cases}. \tag{3.65}$$

resolviendo los problemas secuencialmente en cada subdominio (por ejemplo con el MDF).

Para el caso que ϕ sea subdividido en Ω_E para $E > 1$ subdominios, cada subdominio comparte ciertos niveles de traslape. La velocidad de convergencia depende del nivel de traslape que hay entre cada subdominio adyacente. Cuando se utiliza descomposición de dominio, la velocidad de convergencia se deteriora conforme se aumenta el número de subdominios.

Capítulo 4

Implementación de modelos de interfase difusa en procesadores gráficos

La computación paralela es una técnica de programación en la que diferentes instrucciones son ejecutadas simultáneamente. El cómputo paralelo puede ser trabajado a nivel de instrucción, de hilos o por procesos. Las computadoras paralelas se clasifican según el nivel de paralelismo que admite su hardware: equipos con procesadores multinúcleo, multi-procesadores que tienen múltiples elementos de procesamiento dentro de una sola máquina y *clusters* [40].

Los procesadores gráficos funcionan como coprocesadores para el procesador principal donde el CPU (del inglés Central Processing Unit) realiza operaciones secuenciales que corresponden al algoritmo y distribuye esta información a la memoria de la GPU (Graphics Processing Unit), para que ésta última realice las operaciones paralelizables. En general, este tipo de operaciones son las que demandan cálculos intensivos y que deben ser ejecutados en repetidas ocasiones ya sea de manera independiente o utilizando diferentes datos, permitiendo así realizar un aislamiento sencillo de resolver en una función independiente y ejecutarse en un procesador gráfico mediante una gran cantidad de hilos de ejecución, lo que disminuye la carga de información al CPU.

4.1. Unidad central de procesamiento (CPU)

Uno de los retos más importantes al que se ha enfrentado la computación ha sido aumentar la velocidad del reloj en el procesador principal para mejorar el rendimiento de los dispositivos computacionales. Debido a las diferentes limitantes que existen en la fabricación de circuitos integrados, en los últimos años los fabricantes se han visto obligados a buscar alternativas que mejoren el potencial de cálculo numérico, ya que no es factible depender de la velocidad del reloj como un medio para mejorar la velocidad de los ordenadores.

Durante varias décadas, las supercomputadoras también han hecho mejoras en el rendimiento de sus dispositivos de manera similar. El rendimiento de un procesador utilizado en un superordenador ha escalado astronómicamente las mejoras en el CPU de la computadora personal mediante el aumento en el número de procesadores. En 2005 los principales fabricantes de CPU comenzaron a ofrecer procesadores con dos núcleos de computación en lugar de uno. Durante los siguientes años se siguió este desarrollo con el lanzamiento de tres, cuatro, seis y ocho núcleos en el CPU [41].

4.2. Unidad de procesamiento gráfico (GPU)

La diferencia entre un CPU y una GPU está en la arquitectura de ambos componentes. Aunque están diseñados para funcionar de modo muy similar, las GPU están construidas para trabajar eficientemente el cálculo de información gráfica. Esto último las hace estar mucho mas optimizadas que un procesador convencional por el tipo de labor que realizan.

Los procesadores centrales han evolucionado en ambas ramas, ya sea en las velocidades de reloj o el incremento del número de núcleos. Mientras tanto, el procesamiento gráfico se sometió a una dramática revolución a finales de la década de 1980 y principios de 1990. El crecimiento intesivo provocado por los sistemas operativos de *Microsoft Windows* ayudó a crear en el mercado informático un nuevo tipo de procesador. A principios de 1990 los usuarios empezaron a comprar aceleradoras de gráficos en 2D para las computadoras personales.

En 1999 se introduce al mercado la primer GPU donde las operaciones correspondientes a la visualización de gráficos se realizan en la GPU y el CPU se libera de la carga computacional. Para programar estos procesadores gráficos era necesario utilizar una API (del inglés Application Programming Interface). En 2006, la compañía NVIDIA lanza al mercado la arquitectura CUDA, con la que es posible programar los procesadores gráficos de NVIDIA como computación de propósito general, en lenguaje estándar como C/C++ [41].

4.2.1. Arquitectura de las GPU

La arquitectura de los procesadores gráficos está conformada por arreglos de multiprocesadores (SM, del inglés Streaming Multiprocessor). Cada arquitectura contiene diferentes cantidades de multiprocesadores. La arquitectura Kepler (usada en esta tesis) se compone de 192 procesadores escalares SP (del inglés Scalar Processors), cada SP es un núcleo de la GPU [41]. El modelo de programación que se utiliza en la GPU es el SPMD (del inglés Single Program, Multiple Data), en el que todas las unidades de procesamiento ejecutan el mismo código y operan sobre diferentes datos. Esto reduce la complejidad de la unidad de control y permite que el

procesador gráfico dedique mayor espacio de trabajo.

4.2.2. Tipos de memoria

El manejo de distintos tipos de memoria en un procesador gráfico permite acelerar la ejecución de un programa informático mediante el uso correcto que proporciona la arquitectura. Dentro de la arquitectura de los procesadores gráficos se encuentran los siguientes tipos de memoria:

- **Memoria global:** Es la memoria principal de los procesadores gráficos la cual funciona como memoria RAM (del inglés Random Access Memory) para la GPU. El uso general de la memoria está dado por la lectura y el almacenamiento de la información ya que todos los hilos pueden acceder fácilmente a ella modificando únicamente los índices en el arreglo por el cual se ubica el elemento correspondiente. Sin embargo, como todos los hilos intentan acceder a la memoria al mismo tiempo, el acceso a la memoria global es mayor comparado con la velocidad de cómputo del procesador gráfico, lo que limita el rendimiento de la GPU.

- **Memoria compartida:** Se encuentra físicamente dentro de la memoria de cada multiprocesador, esto acelera el tiempo de acceso a los datos. Esta memoria puede ser compartida por todos los hilos en el mismo bloque que corresponden a los que ejecuta el multiprocesador. En este tipo de memoria está permitido la lectura y la escritura de datos a la memoria compartida, lo que probablemente pueda generar errores en la sincronización de hilos, es decir que un hilo lee un valor y posteriormente otro hilo reescribe otro dato. Por este motivo se utilizan barreras de sincronización lo que limita la paralelización del algoritmo [42].

- **Memoria local:** Cada hilo tiene reservada una memoria individual la cual no puede ser accedida por ningún otro hilo. Es utilizada para almacenar variables privadas y cálculos locales de cada hilo, aunque posteriormente la variable puede ser transferida a otra localidad de memoria en caso de ser necesario.

4.3. Plataforma de desarrollo de CUDA

La industria de la computación está al borde de una revolución con la computación paralela y NVIDIA CUDA C ha sido hasta ahora uno de los lenguajes de mayor éxito jamás diseñados para la computación paralela [41]. Una GPU es capaz de ejecutar grandes cantidades de procesamiento en paralelo, a través del uso de hilos (threads, en inglés) debido a que está compuesto

por una serie de núcleos de procesamiento dedicados al cómputo en paralelo. Un hilo es la unidad básica utilizada del procesador [43].

La función que se ejecuta en la GPU es llamada *kernel*. Para ejecutar una función *kernel* en la GPU se requiere de reservar un espacio de memoria tanto en el CPU como en la GPU, ambos espacios de memoria deben de ser del mismo tamaño ya que el CPU y la GPU cuentan con espacios de memoria RAM independientes y por ello los datos de entrada de la GPU deben copiarse tal y como se encuentran almacenados desde la CPU. Posteriormente se ejecuta la función *kernel* en la GPU. El resultado, el cual se encuentra almacenado en la memoria de la GPU debe ser transferido al CPU para su almacenamiento y visualización. En el Algoritmo 3 se presenta el proceso que se lleva a cabo para implementar programas en una GPU.

Algoritmo 3 Algoritmo para la implementación de una función *kernel*.

U_H, Asignación de memoria en el CPU.
U_D, Asignación de memoria en la GPU.
$cudaMemcpy$, Transferencia de memoria del CPU a la GPU.
$dim3$, Dimensionar el tamaño del bloque.
$kernel <<< grid, thread >>>$, Ejecución de la función *kernel*.
$cudaMemcpy$, Transferencia de memoria de la GPU al CPU.
$cudafree(U_D)$, Liberación de memoria en la GPU.
$free(U_H)$, Liberación de memoria en el CPU.

4.4. Biblioteca de funciones CUBLAS

La biblioteca *CUBLAS* es una implementación de *BLAS* (del inglés, Basic Linear Algebra Subprograms) para la plataforma de desarrollo CUDA, la cual permite al usuario acceder a los recursos computacionales de una GPU NVIDIA.

Desde sus inicios con CUDA 6.0 la biblioteca CUBLAS expone dos conjuntos de API (API CUBLAS y API CUBLASXT).

En API CUBLAS, la aplicación debe de asignar a las matrices y a los vectores requeridos un espacio de memoria en la GPU, a continuación se requiere llenar cada espacio de memoria con datos, después se llama una por una a las funciones de CUBLAS deseadas. Finalmente se devuelven los datos obtenidos del uso de cada una de las funciones de CUBLAS a la memoria del servidor o CPU. Esta API también proporciona funciones que ayudan a escribir y recuperar datos de la GPU sin necesidad de declarar una función extra para copiarlos [44].

La API CUBLASXT presenta una interfaz entre el CPU y multi-GPU, al utilizar esta API el programa sólo tiene que declarar las matrices necesarias en un espacio de memoria del CPU. No hay restricción en el tamaño de las matrices, siempre que pueden caber en la memoria del CPU. La API CUBLASXT se encarga de asignar los espacios de memoria a través de las GPUs

designadas y del envió de la carga de trabajo entre ellos. Finalmente se recuperan los resultados al CPU. La API CUBLASXT solo es compatible con las rutinas BLAS3 para el cálculo intensivo (por ejemplo, las operaciones entre matrices), donde las transferencias PCI (del inglés, *Peripheral Component Interconnect*) de ida y vuelta desde la GPU pueden ser amortizadas [44].

A diferencia en la API CUBLASXT se deben de mantener los datos en el CPU y la biblioteca se encarga de enviar una a una las operaciones a la GPU presente en el sistema.

4.5. Biblioteca de funciones CUSPARSE

Trabajar con métodos numéricos en la solución de sistemas de ecuaciones lineales como el caso de la ec. (3.17) presenta complicaciones de almacenamiento para una implementación computacional porque se requiere de resolver matrices de grandes dimensiones. En ocasiones la mayoría de los elementos de la matriz resultante suelen ser igual a cero. A este tipo de arreglos se le conoce como matrices ralas.

El resultado de la implementación del esquema semi-implícito del MDF en el modelo de Cahn-Hilliard es un sistema de ecuaciones **Ax=b** donde donde **A** es una matriz rala. En la literatura se describen muchas técnicas que permiten dar solución a operaciones entre matrices ralas únicamente tomando en cuenta los datos distintos a cero de la matriz.

La API CUBLAS fue diseñada para el uso en procesadores gráficos de NVIDIA y forma parte del conjunto de herramientas de CUDA. La biblioteca CUSPARSE es un conjunto de subrutinas básicas de álgebra lineal utilizadas en la manipulación de matrices ralas. Las rutinas de esta biblioteca estan diseñadas para ser llamadas desde C o C++ y se clasifican en cuatro categorías:

- Nivel 1: operaciones entre un vector en formato ralo y un vector en formato denso.
- Nivel 2: operaciones entre una matriz en formato ralo y un vector en formato denso.
- Nivel 3: operaciones entre una matriz en formato ralo y un conjunto de vectores en formato denso.
- Conversión: operaciones que permiten la conversión entre distintos formatos de matriz.

La biblioteca CUSPARSE permite a los desarrolladores acceder a los recursos de una GPU a pesar de que no sea autoparalelizable a través de múltiples GPUs ya que esta API considera que los datos de entrada y salida residen en la memoria de la GPU a menos que se indique lo contrario.

4.5.1. Matrices ralas por coordenadas (COO)

Este tipo de almacenamiento, COO, del inglés *coordinate*, Almacenamiento de matrices ralas mediante una lista de coordenadas (renglones y columnas) y valores, se distingue en su representación ya que cada entrada de la matriz **A** requiere de tres vectores. Los elementos necesarios para formar una matriz tipo COO estan dados en la Tabla 4.1.

Tabla 4.1: Elementos contenidos en una matriz rala de tipo COO.

nnz	Cantidad de datos diferentes de cero
$\mathbf{v}_k$	Vector con valores diferentes de cero
$\mathbf{i}_k$	Vector que contienen los índices del renglón al que pertence cada valor de v_k
$\mathbf{j}_k$	Vector que contienen los índices de la columna al que pertence cada valor de v_k

Para dar un ejemplo sencillo donde se muestre el almacenamiento de una matriz rala en formato COO se considera una matriz rala **A** dada por

$$\begin{pmatrix} 1 & 4 & 0 & 0 & 0 \\ 0 & 2 & 3 & 0 & 0 \\ 5 & 0 & 0 & 7 & 8 \\ 0 & 0 & 9 & 0 & 6 \end{pmatrix}, \tag{4.1}$$

donde el formato COO indexado a cero de la matriz **A** estaría expresado en

$$\begin{aligned} \mathbf{v}_k &= \begin{bmatrix} 1 & 4 & 2 & 3 & 5 & 7 & 8 & 9 & 6 \end{bmatrix}, \\ \mathbf{i}_k &= \begin{bmatrix} 0 & 0 & 1 & 1 & 2 & 2 & 2 & 3 & 3 \end{bmatrix}, \\ \mathbf{j}_k &= \begin{bmatrix} 0 & 1 & 1 & 2 & 0 & 3 & 4 & 2 & 4 \end{bmatrix}. \end{aligned} \tag{4.2}$$

4.5.2. Almacenamiento comprimido por renglones (CSR) y por columnas (CSC).

Una matriz rala **A** está representada en formato CSR (del inglés, *Compressed Sparse Row*) por los parámetros que se presentan en la Tabla 4.2.

Para el almacenamiento de una matriz rala en formato COO se enumera cada dato diferente de cero según la posición que se encuentre teniendo como priodidad el número de fila. Es decir, el arreglo de filas comprimidas contiene el valor asignado al primer elemento que aparece en el renglón. Ejemplo:

Tabla 4.2: Elementos contenidos en una matriz rala de tipo CSR.

nnz	Cantidad de datos diferentes de cero
v	Vector con valores diferentes de cero
i	Vector comprimido que contiene el subíndice del renglon al que pertence cada valor de v_k
j	Vector que contienen los índices de la columna al que pertence cada elemento de **v**

Se tiene una matriz rala de 4×5

$$\begin{pmatrix} 1 & 4 & 0 & 0 & 0 \\ 0 & 2 & 3 & 0 & 0 \\ 5 & 0 & 0 & 7 & 8 \\ 0 & 0 & 9 & 0 & 6 \end{pmatrix}, \tag{4.3}$$

donde el almacenamiento indexado a cero del formato CSR es

$$\begin{aligned} \mathbf{v} &= \begin{bmatrix} 1 & 4 & 2 & 3 & 5 & 7 & 8 & 9 & 6 \end{bmatrix}, \\ \mathbf{i} &= \begin{bmatrix} 0 & 2 & 4 & 7 & 9 \end{bmatrix}, \\ \mathbf{j} &= \begin{bmatrix} 0 & 1 & 1 & 2 & 0 & 3 & 4 & 2 & 4 \end{bmatrix}. \end{aligned} \tag{4.4}$$

Análogamente existe el método CSC (del inglés, *Compressed Sparse Column*) en el cual el almcenamiento se comprime por columnas en vez de renglones.

4.5.3. Formato *Ellpack-Itpack* (ELL)

Una matriz rala **A** con $m \times n$ elementos se almacena en formato ELL utilizando dos matrices densas de dimensiones $m \times k$, donde k es el número de elementos diferentes de cero de la fila que tiene la mayor cantidad de datos diferentes de cero. La primera matriz de datos contiene valores de los elementos no vacíos, mientras que la segunda matriz contiene los índices de la columna correspondiente.

Ejemplo: Se tiene una matriz rala de 4×5 elementos

$$\begin{pmatrix} 1 & 4 & 0 & 0 & 0 \\ 0 & 2 & 3 & 0 & 0 \\ 5 & 0 & 0 & 7 & 8 \\ 0 & 0 & 9 & 0 & 6 \end{pmatrix}, \tag{4.5}$$

y su almacenamiento en formato ELL indizado a cero esta dado por las matrices de

$$\textbf{valores} = \begin{pmatrix} 1 & 4 & 0 \\ 2 & 3 & 0 \\ 5 & 7 & 8 \\ 9 & 6 & 0 \end{pmatrix}, \qquad \textbf{índices} = \begin{pmatrix} 0 & 1 & -1 \\ 1 & 2 & -1 \\ 0 & 3 & 4 \\ 2 & 4 & -1 \end{pmatrix}. \tag{4.6}$$

Los elementos de las filas con menos elementos diferentes de cero se rellenan con valores de -1. El formato ELL no está soportado directamente por la biblioteca de CUBLAS, sin embargo, se utiliza para el almacenamiento de la matriz en el formato HYB.

4.5.4. Formato híbrido (HYB)

El formato de almacenamiento HYB se compone de una parte regular (normalmente se almacena en formato ELL) y una parte irregular (normalmente se almacena en formato COO) [44]. HYB se implementa como un formato de datos ralos que requiere el uso de una operación de conversión para almacenar una matriz en el mismo. La operación de conversión divide a la matriz general en partes regulares e irregulares de forma automática o de acuerdo con criterios de desarrolladores especificado.

4.6. Matriz de la ecuación de Cahn-Hilliard

De la formulación semi-implícita del modelo de Cahn-Hilliard utilizando el operador biarmónico se obtiene un sistema de ecuaciones **Ax=b**, que para el caso de una dimensión ésta expresión se representa como:

$$A\phi = \alpha \begin{bmatrix} \frac{a_1}{\alpha} & \frac{a_2}{\alpha} & \frac{a_3}{\alpha} & 0 & 0 & 0 & 0 & 0 \\ -4 & 6+\frac{1}{\alpha} & -4 & 1 & 0 & 0 & 0 & 0 \\ 1 & -4 & 6+\frac{1}{\alpha} & -4 & 1 & 0 & 0 & 0 \\ 0 & 1 & -4 & 6+\frac{1}{\alpha} & -4 & 1 & 0 & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & 1 & -4 & 6+\frac{1}{\alpha} & -4 & 1 \\ 0 & 0 & 0 & 0 & 1 & -4 & 6+\frac{1}{\alpha} & -4 \\ 0 & 0 & 0 & 0 & 0 & \frac{c_1}{\alpha} & \frac{c_2}{\alpha} & \frac{c_3}{\alpha} \end{bmatrix} \begin{bmatrix} \phi_1^m \\ \phi_2^m \\ \phi_3^m \\ \phi_4^m \\ \vdots \\ \phi_{N-2}^m \\ \phi_{N-1}^m \\ \phi_N^m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_{N-2} \\ b_{N-1} \\ b_N \end{bmatrix} = \mathbf{b} \tag{4.7}$$

donde a_1, a_2, a_3, c_1, c_2, c_3, b_1 y b_N se obtienen de las condiciones de frontera,

$$\alpha = -\frac{\varepsilon \Delta t}{\Delta x^4}, \tag{4.8}$$

y

$$b_i = \phi_i^{m-1} + \frac{6\Delta t}{4\Delta x^2}\left(\phi_{i-1}^{m-1} - \phi_{i+1}^{m-1}\right)^2 + \frac{\left(3(\phi_i^{m-1})^2 - 1\right)\Delta t}{\Delta x^2}\left(\phi_{i-1}^{m-1} - 2\phi_i^{m-1} + \phi_{i+1}^{m-1}\right). \tag{4.9}$$

Si se desea trabajar el modelo semi-implícito de Cahn-Hilliard utilizando la matriz completa de la ec. (4.7), hay que tomar en cuenta que el tamaño de la matriz **A** es de $N \times N$ elementos y se requieren de $8(N \times N)$ bytes para almacenar dichos datos en un espacio de memoria, ya que los elementos de la matriz son de tipo *double*.

Para desarrollar la matriz COO de la ec. (4.7) se toma en cuenta que cada renglón de la matriz **A** contiene cinco elementos diferentes de cero, entonces, el número de elementos distintos de cero de la matriz COO es de $(N \times 5)$ y además se utilizan dos vectores con la misma cantidad de elementos, uno para el índice i y el otro para el índice j de la matriz completa. La cantidad total de elementos en una matriz COO son de $3 \times (N \times 5)$, mientras que el espacio de memoria utilizada es de $16 \times (N \times 5)$ bytes, porque se genera un primer vector el cual incluye los datos diferentes de cero de la matriz almacenados como tipo *double* más dos vectores que indican la posición y estos son almacenados en espacios de tipo *int*.

En la Tabla 4.3 se presenta una comparación entre la cantidad de recursos utilizdos por una matriz completa y el espacio que utiliza una matriz rala para el caso del modelo semi-implícito de Cahn-Hilliard formulado con el operador biarmónico en una dimensión.

Del desarrollo semi-implícito del modelo de Cahn-Hilliard utilizando el operador biarmónico en dos dimensiones se obtiene un sistema de ecuaciones **Ax=b** similar al de la ec. (4.7).

A diferencia de una dimensión, para el modelo semi-implícito de Cahn-Hilliard en dos dimensiones se tiene que el tamaño de la matriz es de $N^2 \times N^2$ elementos, cada elemento de la matriz se almacena en espacios de memoria de tipo *double*, por lo que la cantidad de memoria que se utiliza para su almacenamiento es de $8 \times (N^2 \times N^2)$ bytes.

Tabla 4.3: Comparación en el uso de memoria requerida entre la matriz completa y matriz COO para el modelo semi-implícito de Cahn-Hilliard en 1D.

N	Matriz A	Matriz COO	Memoria A	Memoria COO	Ahorro de memoria (%)
16	256	240	2 Kb	1.25 Kb	37.50
32	1024	480	8 Kb	2.5 Kb	68.75
64	4096	960	32 Kb	5 Kb	84.38
128	16384	1920	128 Kb	10 Kb	92.19
256	65536	3840	512 Kb	20 Kb	96.09
512	262144	7680	2 Mb	40 Kb	98.04

La matriz resultante del esquema semi-implícito del modelo de Cahn-Hilliard en dos dimensiones tiene trece elementos diferentes de cero en cada renglon, para utilizar la matriz COO se generan tres vectores, el primero esta compuesto por $(N^2 \times 13)$ elementos, los otros dos vectores se construyen con los valores de los indices i y j de la matriz $\mathbf{A}$. El primer vector es almacenado en espacios de memoria de tipo *double* y los vectores de índices en espacios tipo *int*. El tamaño total para una matriz COO es de $13 \times (N^2 \times 13)$ elementos y la cantidad de memoria utilizada es de $16 \times (N^2 \times 13)$ bytes.

En la Tabla 4.4 se muestra una comparación en el uso de la memoria que se requiere para almacenar la matriz completa y la ventaja de utilizar una matriz COO.

Para tres dimensiones se tiene un sistema de ecuaciones donde la matriz $\mathbf{A}$ es de $N^3 \times N^3$ elementos porque contiene N^3 renglones, cada elemento de la matriz se almacena en espacios de memoria de tipo *double*. Para el almacenamiento de esta matriz se requieren de $8 \times (N^3 \times N^3)$ bytes. En comparación a una matriz COO se observa que cada renglón de la matriz completa tiene 25 elementos diferentes de cero, esto genera los tres vectores de tamaño $(N^3 \times 25)$ elementos cada uno. Donde el primero es reservado por espacios de memoria de tipo *double* y los otros dos de tipo *int* para cada uno de los índices, esto da un tamaño total de la matriz COO de $3 \times (N^3 \times 25)$ elementos y el espacio total de la memoria requerida es de $16 \times (N^3 \times 25)$ bytes.

En la Tabla 4.5 se muestra la comparación con respecto al uso de memoria que se requiere si se trabaja con la matriz completa o con la matriz COO para el modelo de Cahn-Hilliard en tres dimensiones con el operador biarmónico.

Tabla 4.4: Comparación en el uso de memoria requerida entre la matriz completa y matriz COO para el modelo semi-implícito de Cahn-Hilliard en 2D.

N	Matriz A	Matriz COO	Memoria A	Memoria COO	Ahorro de memoria (%)
16×16	6.55×10^4	9984	512 Kb	52 Kb	89.84
32×32	1.05×10^6	39936	8 Mb	208 Kb	97.46
64×64	1.67×10^7	159744	128 Mb	832 Kb	99.36
128×128	2.68×10^8	638976	2 Gb	3.25 Mb	99.84
256×256	4.29×10^9	2555904	32 Gb	13 Mb	99.96
512×512	6.87×10^{10}	10223616	512 Gb	52 Mb	99.99

Tabla 4.5: Comparación en el uso de memoria requerida entre la matriz completa y matriz COO para el modelo semi-implícito de Cahn-Hilliard en 3D.

N	Matriz A	Matriz COO	Memoria A	Memoria COO	Ahorro de memoria (%)
$16 \times 16 \times 16$	1.67×10^7	3.07×10^5	1.56 Mb	1.56 Mb	98.47
$32 \times 32 \times 32$	1.07×10^9	2.45×10^6	12.49 Mb	12.49 Mb	99.84
$64 \times 64 \times 64$	6.87×10^{10}	1.96×10^7	100 Mb	100 Mb	99.98
$128 \times 128 \times 128$	4.39×10^{12}	1.57×10^8	800 Mb	800 Mb	99.99
$256 \times 256 \times 256$	2.81×10^{14}	1.25×10^9	6.24 Gb	6.24 Gb	99.99
$510 \times 510 \times 510$	1.80×10^{18}	1×10^{10}	50 Gb	50 Gb	99.99

4.7. Implementación de un *cluster* con GPUs

Un *cluster* es un sistema que se utiliza para el procesamiento paralelo o distribuido compuesto por un conjunto de computadoras que trabajan como un solo equipo de cómputo. Los *clusters* de CPUs constan de dos o más nodos conectados entre sí por un canal de comunicación, así mismo cada nodo únicamente necesita un elemento de proceso, memoria y una interfaz con la red del *cluster*. También necesitan software especializado, ya sea a nivel de aplicación o a nivel del núcleo y por lo tanto todos los elementos del *cluster* trabajan para cumplir una funcionalidad conjunta.

Los *clusters* son empleados para mejorar el rendimiento y la disponibilidad de ciertos servicios que normalmente no son proporcionados por una sola computadora, como alto rendimiento, alta disponibilidad, balanceo de carga, escalabilidad.

La construcción de un *cluster* es accesible y se pueden clasificar en tres tipos,

- Homogéneo (mismo *hardware* y sistema operativo)
- Heterogéneo (diferente *hardware* y sistema operativo)
- Híbrido (diferente rendimiento pero con arquitecturas y sistemas operativos diferentes)

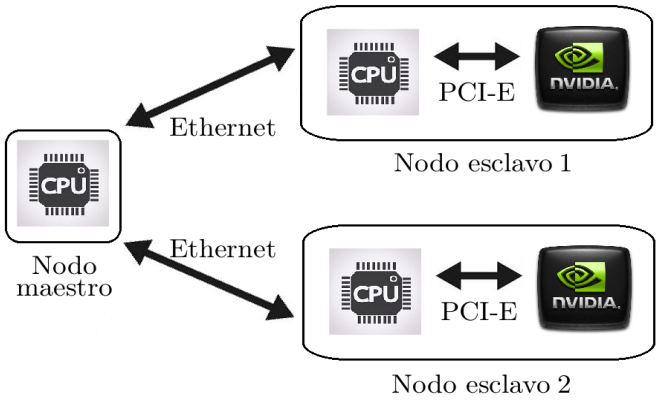


Fig. 4.1: Diagrama del diseño de un *cluster* de GPUs.

Con el incremento del uso de procesadores gráficos en el cómputo de alto rendimiento, se ha logrado implementar el uso de las GPU's en las supercomputadoras y *cluster* más potentes en el mundo.

En este apartado se presenta el diseño, implementación y gestión de un prototipo de *cluster* con GPUs para cómputo de alto rendimiento. Se describen todos los componentes necesarios para un *cluster* con GPUs utilizando software libre y con un costo mínimo de hardware.

Un *cluster* con GPUs, es un conjunto de computadoras interconectadas por una red donde cada elemento del *cluster* se le conoce como nodo. Cada nodo esta equipado por una unidad de procesamiento gráfico, de esta forma la potencia de cálculo de cada nodo se aprovecha en tareas que requieren de alta actividad para el cómputo de datos.

La interconexión de nodos en un *cluster* con GPUs requiere de un hardware capaz de transmitir grandes cantidades de datos en poco tiempo, esto podría aumentar el tiempo de operación sobre los datos. El tipo de interconexión depende también del número de nodos presentes en el *cluster*. Para obtener reultados de la presente tesis se implementó un *cluster* de tres CPUs donde cada nodo incluye una tarjeta gráfica de *NVIDIA* modelo *Geforce GTX 670* como se observa en la Fig. 4.1.

4.8. Interfaz de paso de mensaje (MPI)

El paso de mensajes es una técnica empleada en programación concurrente para aportar sincronización entre procesos y permitir el trabajo mutuo. La principal característica es que no requiere de memoria compartida, lo que es una importante aportación para la programación de sistemas distribuidos. Los elementos que intervienen en el paso de mensajes son el proceso que envía, el que recibe y el mensaje. La Interfaz de Paso de Mensajes (MPI, del inglés *Message Passing Interface*) es una estandarización encargada de realizar operaciones que utilizan paso de mensaje. MPI aborda problemas utilizando programación paralela en el que los datos se mueven mediante operaciones de acceso remoto a la memoria desde un espacio reservado para un proceso a otro de diferente proceso. Esto hace que todas las operaciones de MPI se expresen como funciones, subrutinas o métodos de acuerdo con los lenguajes que se encuentren trabajando (C, Fortran son parte del estándar MPI).

Algoritmo 4 Implementación del método alternante de Schwarz para el modelo de Cahn-Hilliard utilizando MPI.

ϵ, *Constante de interfase.*
L, *Longitud del dominio.*
N, *Cantidad de nodos en el dominio.*
Δx, *Incremento sobre el eje espacial x.*
Δy, *Incremento sobre el eje espacial y.*
Δz, *Incremento sobre el eje espacial z.*
Δt, *Incremento sobre el eje temporal.*
T, *Tiempo máximo de modelado.*
r, *Número de iteraciones para alternar las fronteras de subdominios.*
ϕ, *Condición incial del parámetro de orden.*
$Traslape$, *Número de nodos que ocupa el traslape.*
MPI_Init, *Se inicializa comunicación MPI.*
if (*Master*) **then**
    ϕ, *Se genera condición inicial para el parámetro de orden.*
    MPI_Send, *Se subdivide el parámetro* ϕ *y se envían datos a los nodos Slaves.*
    MPI_Recv, *Se reciben las particiones del parámetro* ϕ *de los nodos Slaves y se ordenan.*
    Save ϕ, *Se almacenan los resultados para su visualización.*
else
    MPI_Recv, *Se reciben la partición del parámetro* ϕ *del nodo Master.*
    for $(i = 0$ *to* $i < T/r)$ **do**
        for $j = 0$ *to* $j < r$ **do**
            $\frac{\partial \phi}{\partial t} = \nabla \cdot [M(\phi)\nabla\mu(\phi)]$.
        end for
        Coom_Slave, *Se alternan fronteras entre nodos Slave.*
    end for
end if

Las principales ventajas de establecer una comunicación estándar para programas de pa-

so de mensaje es la portabilidad y facilidad de uso. Esto proporciona a los proveedores un conjunto claramente definido de rutinas aplicables de manera eficiente, en algunos casos pueden proporcionar soporte de hardware, mejorando así la escalabilidad. El objetivo del MPI es desarrollar una comunicación estándar ampliamente utilizada para escribir programas paso de mensaje. Como tal, la interfaz debe establecer una norma práctica, portable, eficiente y flexible la comunicación de mensajes.

OpenMPI es un API de código abierto desarrollado para facilitar la programación paralela y distribuida que implementa el estándar MPI. Esta API permite la distribución de procesos de forma dinámica, provee de alto rendimiento y tolerancia a fallos, soporte para redes heterogéneas y es portable ya que funciona en diferentes sistemas operativos (Linux, OS-X y Solaris). Presenta también opciones de configuración durante la instalación, la compilación de programas y su ejecución. La implementación del método de Schwarz para modelo de Cahn-Hilliard en tres dimensiones por el esquema explícito y semi-implícito se representa en el Algoritmo 4.

Capítulo 5

Resultados

En esta sección se presentan los resultados obtenidos con las implementaciones realizadas al modelo de Cahn-Hilliard. Se muestra la evolución del modelo para una, dos y tres dimensiones utilizando diferentes condiciones de frontera con el esquema explícito y a su vez se realiza una comparación entre las ventajas y desventajas que se obtienen al trabajar con técnicas de descomposición de dominio en un GPU y en un *cluster* de GPU's.

5.1. Condiciones de los experimentos numéricos

El modelo de Cahn-Hilliard está representado por

$$\frac{\partial \phi}{\partial t} = -\varepsilon \nabla^4 \phi + \nabla^2 \left(\phi^3 - \phi \right) \tag{5.1}$$

para una constante $\varepsilon > 0$.

Los resultados que se presentan fueron realizados para diferentes tamaños de mallas. Para las mallas en dos y tres dimensiones se consideran valores de $\Delta x = \Delta y = \Delta z$. Debido a que el GPU trabaja con warps (grupos de 32 hilos), se utilizan tamaños de malla con $N = 32$,64, 96 y 128 para favorecer el acceso a la memoria del GPU, también se utilizan tarjetas gráficas modelo *Geforce GTX 670*. Las características de estas tarjetas se presentan en la Tabla 5.1.

Tabla 5.1: Características de la GPU *Geforce GTX 670*.

Especificaciones	Valor
Núcleos totales	1344
Memoria interna	2 Gb
Ancho de banda	192 Gb/s
Velocidad de la memoria	6.0 Gbps

5.2. Solución numérica del modelo de Cahn-Hilliard en 1D

En esta sección se resuelve el modelo de Cahn-Hilliard en una dimensión con el método semi-implícito. El método semi-implícito para el modelo de Cahn-Hilliard se obtiene al formular el término no lineal con el método explícito y la parte lineal utilizando el método implícito. Este método tiene implícito un sistema de ecuciones $\mathbf{Ax} = \mathbf{b}$, que para el caso del modelo de Cahn-Hilliard la matriz $\mathbf{A}$ está expresada en la ec. (4.7).

El sistema de ecuaciones de la ec. (4.7) no puede resolver en paralelo con métodos tradicionales por lo que se plantea como un método de optimización y se resuelve con el método del gradiente conjugado.

Para el modelado de la ecuación de Cahn-Hilliard se consideró una malla con $N = 96$, $\Delta t = 1 \times 10^{-3}$. En la Fig. 5.1 se muestra la solución al modelo de Cahn-Hilliard para una condición inicial de $\phi(x, 0) = \sin\left(\frac{5\pi x}{2}\right)$ y condiciones de frontera de Dirichlet

$$\phi(-1, t) = -1, \quad \phi'(-1, t) = 0, \quad \phi(1, t) = 1 \quad y \quad \phi'(1, t) = 0, \tag{5.2}$$

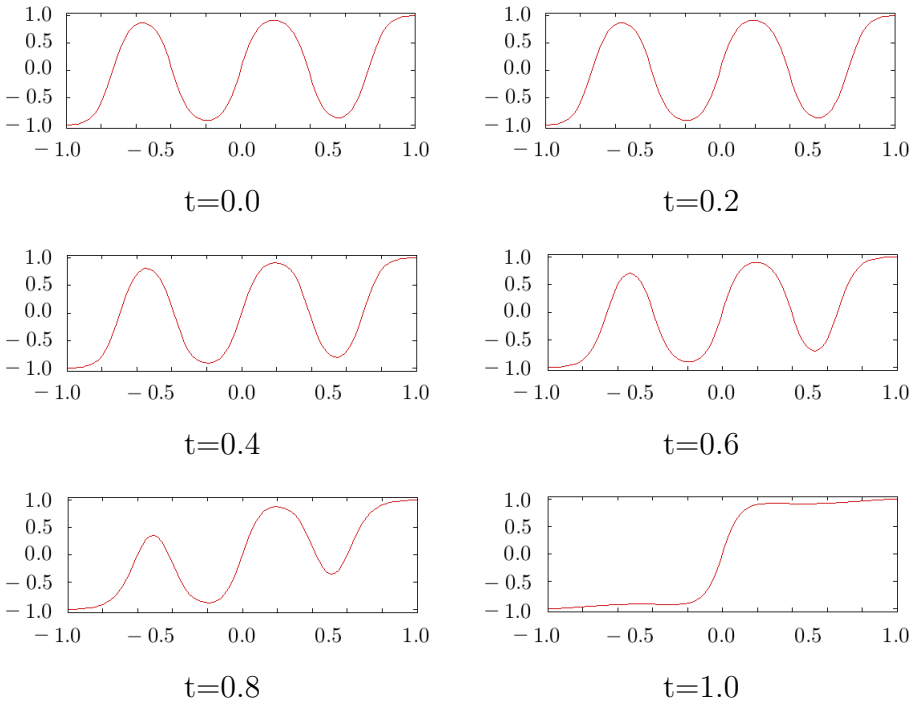


Fig. 5.1: Evolución del modelo semi-implícito de Cahn-Hilliard en 1D.

En la Fig. 5.1 se observa que el modelo de Cahn-Hilliard alcanza una estabilidad en $t > 1$. El uso de técnicas de paralelización no son necesarias para resolver este modelo en una dimensión ya que el procesamiento que requiere el modelo es muy poco en comparación a cuando se resuelve en dos y tres dimensiones.

5.3. Solución numérica del modelo de Cahn-Hilliard en 2D en un GPU

El método semi-implícito es una técnica que permite mejorar el tamaño de Δt, esto ayuda a disminuir el número de iteraciones para llegar a una solución estable y como consecuencia ayuda a disminuir el tiempo de procesamiento.

Una de las desventajas que se presenta al trabajar con esquema semi-implícito es que se requiere de mucha cantidad de memoria ya que se necesita almacenar la matriz que resulta de la linealización del modelo de Cahn-Hilliard.

La matriz que se obtiene al implementar el MDF en el modelo de Cahn-Hilliard es de tipo rala, y para en 2D con condiciones de frontera periódicas el número de elementos diferentes de cero en cada renglón es igual a 13. Para dar solución al sistema de ecuaciones que se obtiene mediante la linealización que se presentó en el capítulo 3 se utilizan matrices CSR y se plantea la solución como un método de optimización.

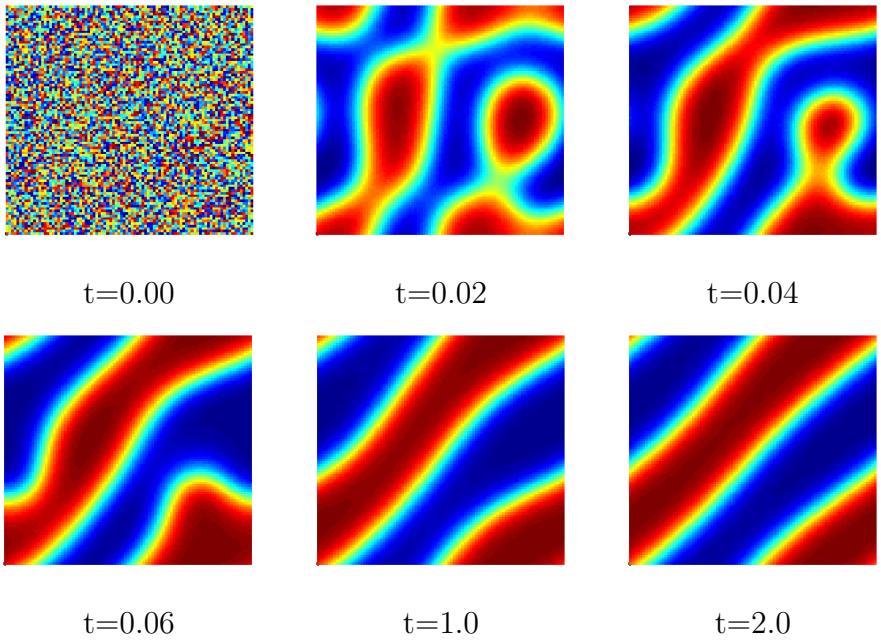


Fig. 5.2: Evolución del modelo semi-implícito de Cahn-Hilliard en 2D.

En la Fig. 5.2 se muestra la evolución del modelo de Cahn-Hilliard en dos dimensiones con condiciones de frontera periódicas, con una malla de 92×92, $\varepsilon = 0.008$ y modelado hasta $t = 2.0$. En este caso la matriz que resulta es completamente simétrica y definida positiva lo que permite el uso del método del gradiente conjugado.

Existen diferentes casos, por ejemplo, las condiciones de frontera de no flujo en el que la matriz no es simétrica ni definida positiva. Para este caso especial se puede dar solución al sistema de ecuaciones con el método del gradiente biconjugado.

5.4. Solución del modelo de Cahn-Hilliard en 3D en un GPU

Se resolvió la ecuación de Cahn-Hilliard utilizando el sistema de dos ecuaciones diferenciales de segundo orden dado en la ec. (3.7) con condiciones de frontera de Neumann. Debido a la naturaleza del MDF, el esquema explícito que resulta en el modelo de Cahn-Hilliard permite una paralelización sencilla del modelo de Cahn-Hilliard. Este método requiere solo operaciones de aritmética sencilla, esto a la vez presenta una desventaja cuando se discretiza ya que los pasos de Δt deben de ser muy pequeños. Este es un método altamente inestable por lo que se requiere de mallas muy finas, por lo tanto se requiere almancenar mucha información. A consecuencia de esto se desperdicia mucho tiempo de modelado. En la tabla 5.2 se muestran los tiempos de modelado que se obtuvieron de la implementación en un GPU con características ya mencionadas.

En este experimento se observa que el tiempo de ejecución del algoritmo es alto ya que conforme se incrementa el número de nodos de la malla, el tiempo también incrementa ya que el tamaño de Δt disminuye conforme el tamaño de la malla aumenta.

En la Fig. 5.3 se muestra la evolución del modelo de Cahn-Hilliard en tres dimensiones para una malla de $32 \times 32 \times 32$ con una $\varepsilon = 0.008$ hasta $t = 10.00$ utilizando condiciones de frontera de Dirichlet y condiciones iniciales aletorias.

Tabla 5.2: Resultados de medición del tiempo en el modelado de Cahn-Hilliard en tres dimensiones en un GPU.

N	Δt	ε	Tiempo
32	0.0025	0.008	8.21 seg
64	0.00001	0.0008	2 min, 64 seg
128	0.000001	0.0008	2 hrs, 56 min

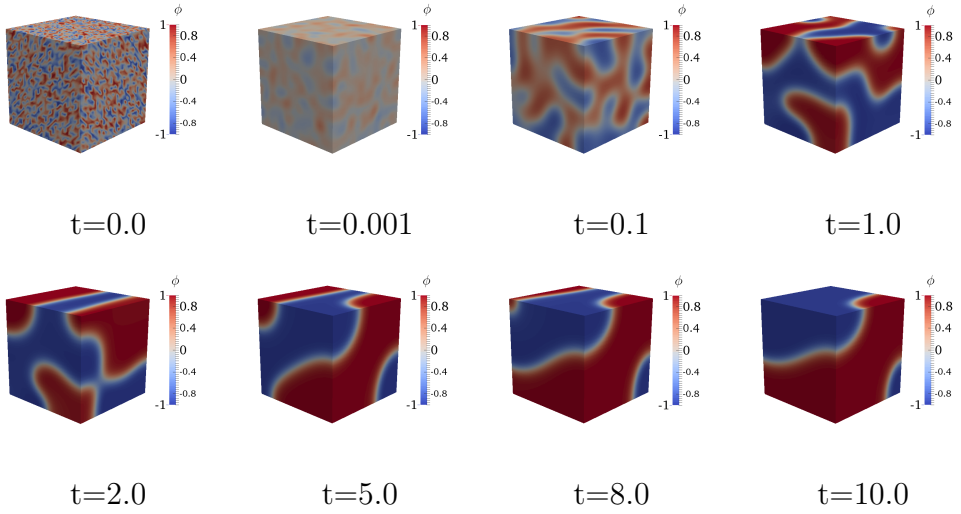


Fig. 5.3: Evolución del modelo explícito de Cahn-Hilliard en 3D con condiciones iniciales aleatorias.

En la Fig. 5.4 se muestra otra evolución del modelo de Cahn-Hilliard en tres dimensiones para una malla de $32 \times 32 \times 32$ con una $\varepsilon = 0.008$ hasta $t = 10.00$, esta vez se utilizaron condiciones de frontera de Dirichlet y condiciones iniciales alternando 1 y -1 en cada elemento de la malla.

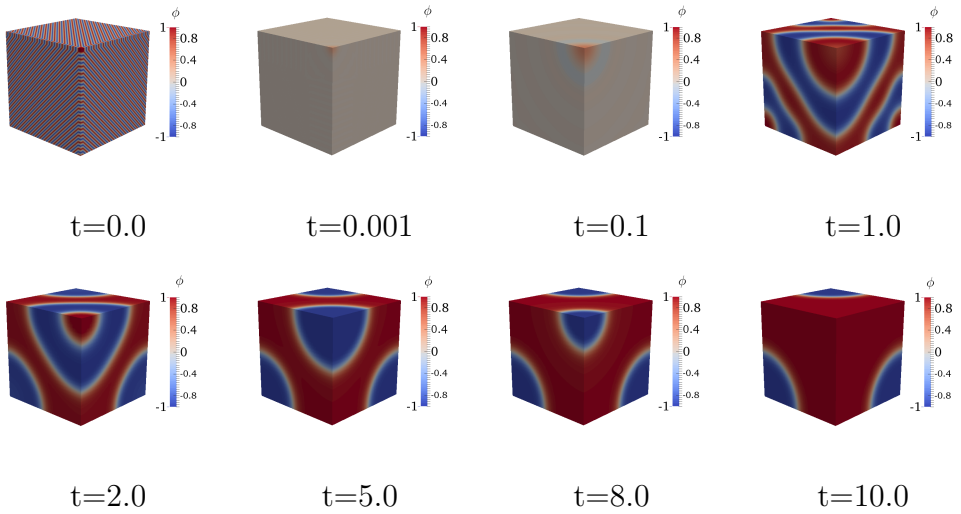


Fig. 5.4: Evolución del modelo explícito de Cahn-Hilliard en 3D con condiciones iniciales alternando 1 y -1 en cada elemento de la malla.

5.5. Solución del modelo de Cahn-Hilliard en 3D con descomposición de dominio en un GPU

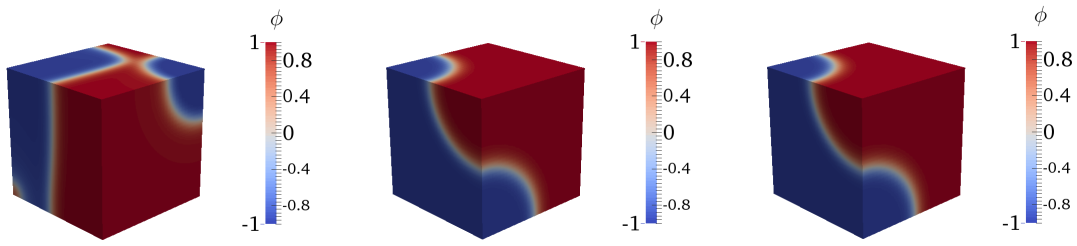
Es posible reducir el tiempo que se presentarón en la Tabla 5.2 si se le aplican técnicas de descomposición de dominio ya que el número de operaciones se compartirán en diferentes subprocesos, por lo que el proceso a resolverse disminuye de igual forma. En la Fig. 5.3 se observa que la evolución del modelo de Cahn-Hilliard mantiene un cambio rápido en los primeros instantes de tiempo, esta observación es importante para implementar el método alternante de Schwarz. Para ello se examina el comportamiento del modelo durante los primeros instantes de evolución con el objetivo de encontrar una forma óptima de mantener comunicación entre cada uno de los nodos del *cluster*.

Para la implementación del modelo de Cahn-Hilliard con el método alternante de Schwarz se utiliza la biblioteca de funciones de MPI para comunicar las subrutinas que realiza cada proceso. La implementación del algoritmo de Schwarz se complica para el modelo de Cahn-Hilliard debido a que es un modelo no lineal de cuarto orden. Esto provoca que se deban tomar precacuciones con la subdivisión del dominio y la asignación en los niveles de traslape. La comunicación entre los nodos que pertenecen a las fronteras interiores incrementa entre mayor sea el número de subdominios, si el intervalo de comunicación es mayor afecta al resultado del modelo. Por lo tanto es necesario considerar los datos presentados en la Tabla 5.3 para llevar a cabo la actualización de los valores en cada área de traslape de cada subdominio. A la frecuencia con la que se actuzalizan los datos en estos elementos lo conoceremos como resolución de transferencia.

En la tabla 5.3 se observa que si el nivel de traslape aumenta, el tamaño de la malla incrementa también, pero los subdominios requieren de menos intervalos de comunicación. Aumentar el número de traslapes tampoco es conveniente ya que el tamaño de la malla se incrementa y dicha descomposición de dominio no tendría impacto alguno ya que el objetivo es disminuir el uso de recursos computacionales.

Tabla 5.3: Resultados al modelar una malla de $32 \times 32 \times 32$ con el método alternante de Schwarz para dos subdomios en un GPU hasta $t = 10$, con $\varepsilon = 0.008$ y $\Delta t = 0.000025$.

Niveles de Traslape	Resolución de transferencia	Tiempo (seg)	Resolución de transferencia	Tiempo (seg)
4		154.01		150.05
6	100 Δt	167.93	200 Δt	166.57
8		174.53		171.93



4 niveles de traslape 6 niveles de traslape 8 niveles de traslape

Fig. 5.5: Modelo explícito de la ecuación de Cahn-Hilliard en 3D, utilizando el método de descomposición de dominio de Schwarz para una resolución de transferencia de datos de 200 Δt.

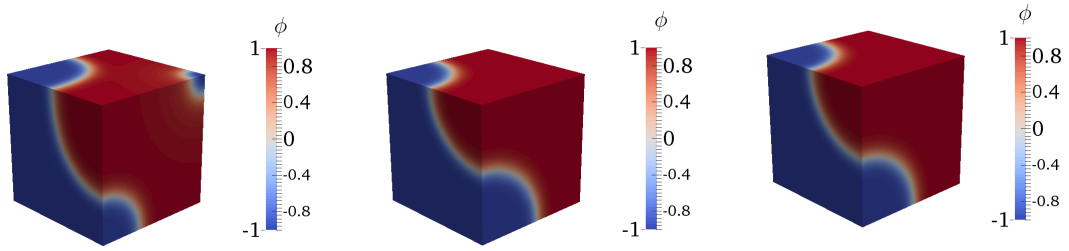
Para el caso de la Fig. 5.5 donde se utilizó el modelo explícito de la ecuación de Cahn-Hilliard en 3D, para una malla de $32 \times 32 \times 32$, una $\varepsilon = 0.008$ en $t = 10.00$ con una condición inicial de intervalos de 1, -1 y utilizando el método de descomposición de dominio de Schwarz para una resolución de transferencia de datos de 200 Δt, se observa que con cuatro niveles de traslape, el modelo no converge en una solución similar a la que se obtiene en la Fig. 5.4. Caso contrario de cuando se utilizan seis niveles de traslape donde la solución obtenida se considera aproximada de acuerdo a la solución obtenida en la Fig. 5.4.

Se trabajó en la implementación del modelo de Cahn-Hilliard en tres dimensiones utilizando descomposición de dominio con el método alternante de Schwarz en dos subrutinas y una tarjeta gráfica. Se consideran las mismas condiciones de la Tabla 5.3 para el modelado en una malla de $64 \times 64 \times 64$ e incrementos en los niveles de traslape considerados en la Tabla 5.3 para que estos sean proporcionales en la Tabla 5.4.

Al igual que en la Tabla 5.5, ahora se considera una resolución de 100 Δt. Por lo tanto se requieren 4000 ciclos de 100 iteraciones cada uno para encontrar una solución en el tiempo $t = 10.00$.

Tabla 5.4: Resultados al modelar una malla de $64 \times 64 \times 64$ con el método alternante de Schwarz para dos subdomios en un GPU hasta $t = 10$, con $\varepsilon = 0.008$ y $\Delta t = 0.000001$.

Niveles de Traslape	Resolución de transferencia	Tiempo (seg)	Resolución de transferencia	Tiempo (seg)
8		16900.56		17553.76
12	2500 Δt	18446.62	5000 Δt	19138.90
16		19911.54		20750.60



4 niveles de traslape 6 niveles de traslape 8 niveles de traslape

Fig. 5.6: Modelo explícito de la ecuación de Cahn-Hilliard en 3D, utilizando el método de descomposición de dominio de Schwarz para una resolución de transferencia de datos de 100 Δt.

En la Fig. 5.6 se observa que con el método explícito de la ecuación de Cahn-Hilliard en 3D, para una malla de $32 \times 32 \times 32$, una $\varepsilon = 0.008$ en $t = 10.00$ y una condición inicial de intervalos de 1 y -1 utilizando el método de descomposición de dominio de Schwarz para una resolución de transferencia de datos de 100 Δt el modelo alcanza un patrón esperado con cuatro niveles de traslape , pero el resultado se encuentra incompleto ya que la convergencia aún no es la esperada con respecto a la Fig. 5.4, por lo que si se quiere utilizar este nivel de transferencia es necesario utilizar seis niveles de traslape.

5.6. Solución del modelo de Cahn-Hilliard en 3D con descomposición de dominio en un *cluster* con GPUs

Se implementó el método alternante de Schwarz como una técnica de descomposición de dominio para el modelo de Cahn-Hilliard en 3D. Después de realizar un análisis donde se consideraron las condiciones para implementar el modelo de Cahn-Hilliard utilizando el método de Schwarz se utiliza una resolución de transferencia de 200 Δt, es decir que los subdominios se mantendrán comunicados cada 200 soluciones individuales.

Tabla 5.5: Resultados de medición del tiempo en el modelado de Cahn-Hilliard en tres dimensiones aplicando descomposición de dominio en un *cluster* de GPU con dos subdominios.

N	Δt	ε	Resolución de transferencia	Niveles de traslape	Tiempo
64	0.00001	0.0008	2500 Δt	12	1min, 32 seg
128	0.000001	0.0008	2500 Δt	24	1 hr, 56 min
256	0.0000001	0.00008	2500 Δt	48	2 hrs, 58 min

En la Tabla. 5.5 se presentan los resultados obtenidos para el modelo de Cahn-Hilliard utilizando condiciones de frontera de Neumann, condiciones iniciales aleatorias, para ello se usaron mallas con tamaños de $N \times N \times N$ donde $N = 32, 64$ y 128 nodos.

El método alternante de Schwarz se implementó en un *cluster* con dos nodos y cada nodo contiene un tarjeta gráfica. Los resultados obtenidos se presentan en la Tabla 5.5.

Con los resultados de la Tabla 5.5 se puede concluir que es recomendable aplicar descomposición de dominio al MDF y el modelo de Cahn-Hilliard ya que en comparación con la Tabla 5.2 para el caso de una malla de $128 \times 128 \times 128$ el tiempo de ejecución disminuye en razón de 33 % para una solución en $t = 1$. También se debe hacer mención que para una malla de $256 \times 256 \times 256$ no es posible encontrar una solución en la targeta GPU que se esta utilizando ya que la memoria no es suficiente, pero al aplicar descomposición de dominio se pudó obtener un resultado para este tamaño de malla gracias al uso de memoria distribuida.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

Después de implementar el método alternante de Schwarz al método de diferencias finitas utilizando el modelo de Cahn-Hilliard como un modelo con interfase difusa se llegó a las siguientes conclusiones:

- Utilizar el método explícito del MDF permite obtener resultados del modelo de Cahn-Hilliard con mayores cantidades de nodos en una malla en comparación con el método semi-implícito que requiere de almacenar una matriz, pero a consecuencia de esto el tiempo de procesamiento aumenta debido a que el método explícito presenta un problema de inestabilidad y eso requiere de disminuir el valores de la discretización en el tiempo.

- El método alternante de Schwarz permite disminuir el tiempo de procesamiento utilizando un *cluster* con procesadores gráficos permitiendo así incrementar la cantidad de nodos para un dominio completo ya que el método alternande de Schwarz subdivide al dominio completo en subdominios que trabajan en subrutinas independientes que son procesadas en diferentes GPU's.

- Es recomendable aplicar el método alternante de Schwarz como un método de descomposición de dominio en el MDF y el modelo de Cahn-Hilliard ya que el tiempo de ejecución disminuye en razón de 33 % para una malla de $128 \times 128 \times 128$ nodos.

- Para una malla de $256 \times 256 \times 256$ no es posible encontrar una solución numérica en la tarjeta *Geforce GTX 670* ya que la memoria interna no es suficiente, pero al aplicar descomposición de dominio si es posible obtener un resultado para este tamaño de malla ya que los datos son distribuidos en dos subrutinas.

6.2. Trabajo a futuro

Es importante identificar las líneas de trabajo para dar continuidad al esfuerzo invertido durante el desarrollo de la presente tesis. Por esto, en esta sección presentamos el trabajo futuro para continuar con las actividades realizadas. Estas líneas pueden resumirse en los puntos siguientes:

- Las ecuaciones de Navier-Stokes son utulizadas para analizar y estudiar el comportamiento de los flujos, por tal motivo sugerimos continuar con la metodología utilizada en el desarrollo de la tesis para analizar y resolver las ecuaciones de Navier-Stokes.
- El modelo H es una implementación entre el modelo de Cahn-Hilliard y las ecuaciones de Navier-Stokes el cual estudia el comportamiento de las fases mientras son sometidas a una fuerza. El uso de este modelo permitirá utilizar estructucturas más complejas para el desarrollo y evolución del modelo. Por lo tanto el tamaño del *cluster* y sus componentes incrementarán. Se propone implementar técnicas de descomposición de dominio para esta variante del modelo de Cahn-Hilliard.
- Los resultados obtenidos se visualizan mediante una etapa de posprocesamiento, para obtener una visualización simultánea al procesamiento se sugiere utilizar técnicas de interoperabilidad entre OpenGL y CUDA.

Referencias y bibliografía

[1] H. Ceniceros *et al*, "A robust, fully adaptive hybrid level-set/front-tracking method for two-phase flows with an accurate surface tension computation," *Commun. Comput. Phys.*, vol. 8, no. 1, pp. 51–94, 2010.

[2] H. D. Ceniceros, R. L. Nós, and A. M. Roma, "Three-dimensional, fully adaptive simulations of phase-field fluid models," *Commun. Comput. Phys.*, vol. 229, no. 17, pp. 6135–6155, 2010.

[3] D. Furihata, "A stable and conservative finite difference scheme for the Cahn–Hilliard equation.," *Numerische Mathematik*, vol. 87, no. 4, pp. 657–699, 2001.

[4] J. Kim, "Phase-field models for multi-component fluid flows," *Commun. Comput. Phys.*, vol. 12, no. 3, pp. 613–661, 2012.

[5] H. D. Ceniceros and C. J. Garcia, "A new approach for the numerical solution of diffusion equations with variable and degenerate mobility," *J. Comput. Phys.*, vol. 240, no. 1, pp. 1–10, 2013.

[6] S. C. Chapra and R. P. Canale, "Métodos numéricos para ingenieros," *MC Graw Hill*, 2007.

[7] S. Lee, C. Lee, H. Lee, and J. Kim, "Comparison of different numerical schemes for the Cahn-Hilliard equation," *J. Ksiam*, vol. 17, no. 3, pp. 197–207, 2013.

[8] X. Zheng *et al*, "A parallel domain descomposition-based implicit method for Cahn–Hilliard-Cook phase-field equation in 3D," *J. Comput. Phys.*, vol. 285, no. 1, pp. 55–70, 2015.

[9] D. M. Anderson, G. B. McFadden, and A. Wheeler, "Diffuse-interface methods in fluid mechanics," *Annu. Rev. Fluid Mech.*, vol. 30, no. 1, pp. 139–165, 1998.

[10] A. R. Brodtkorb, T. R. Hagen, and M. L. Saetra, "Graphics processing unit (GPU) programming strategies and trends in GPU computing," *J. Parallel Distrib. Comput*, vol. 73, no. 1, pp. 4–13, 2013.

[11] D. B. Kirk and W. W. Hwu, "Programming massively parallel processors: A hands-on approach," *Morgan Kaufmann*, 2010.

[12] F. Almeida *et al*, "Introducción a la computación paralela," *Paraninfo*, 2008.

[13] R. Farber, "CUDA application design and development," *Morgan Kaufmann*, 2011.

[14] J. Verhoeven, "Fundamentals of physical metallurgy," *Wiley*, 1975.

[15] S. Allen and J. Cahn, "A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening.," *Acta Metall*, vol. 27, no. 6, pp. 1085–1095, 1979.

[16] L.-Q. Chen, "Phase-field models for microstructure evolution," *Ann. Rev. Mater. Res.*, vol. 32, no. 1, pp. 113–140, 2002.

[17] M. Bene, V. Chalupecky, and K. Mikula, "Geometrical image segmentation by the Allen-Cahn equation," *Appl. Numer. Math.*, vol. 51, no. 3, pp. 187–205, 2004.

[18] J. Dobrosotskaya and A. Bertozzi, "A wavelet-Laplace variational technique for image deconvolution and inpainting.," *IEEE Trans. Image Process.*, vol. 17, no. 5, pp. 657–663, 2008.

[19] X. Feng and A. Prohl., "Numerical analysis of the Allen-Cahn equation and approximation for mean curvature flows.," *Numer. Math*, vol. 94, no. 1, pp. 33–65, 2003.

[20] A. Wheeler, W. Boettinger, and G. McFadden., "Phase-field model for isothermal phase transitions in binary alloys," *Phys. Rev.*, vol. A 45, no. 10, pp. 7424–7439, 1992.

[21] J. Cahn and J. Hilliard, "Free energy of a nonuniform system I, Interfacial free energy," *J. Chem. Phys.*, vol. 2, no. 28, pp. 258–267, 1998.

[22] J. Cahn and J. Hilliard, "Free energy of a nonuniform system II. Thermodynamic basis," *J. Chem. Phys.*, vol. 30, no. 5, pp. 1121–1135, 1959.

[23] B. Zhou and A. Powell, "Phase field simulation of early stage structure formation during immersion precipitation of polymeric membranes in 2D and 3D," *J. Membr. Sci.*, vol. 268, no. 2, pp. 150–164, 2006.

[24] S. Tremaine, "On the origin of irregular structure in Saturn's rings," *Astron. J.*, vol. 125, no. 2, pp. 894–901, 2003.

[25] R. Saxena and G. Caneba, "Studies of spinodal decomposition in a ternary polymer-solvent–nonsolvent systems," *Polym. Eng. Sci.*, vol. 42, no. 5, pp. 1019–1031, 2002.

[26] I. Dolcetta, S. Vita, and R. March, "Area preserving curve shortening flows: from phase transitions to image processing.," *Interfaces Free Bound*, vol. 4, no. 1, pp. 325–343, 2002.

[27] J. Langer, M. Baron, and H. Miller, "New computational method in the theory of spinodal decomposition," *Phys. Rev.*, vol. A-11, no. 4, pp. 1417–1429, 1975.

[28] D. Saylor *et al*, "Diffuse-interface theory for structure formation and release behavior in controlled drug release systems," *Acta Biomater*, vol. 3, no. 6, pp. 851–864, 2007.

[29] S. Choo, S. Chung, and K. Kim, "Conservative nonlinear difference scheme for the Cahn–Hilliard equation II," *Comput. Math. Appl.*, vol. 39, no. 2, pp. 229–243, 2000.

[30] C. Elliot, D. French, and F. Milner, "A second order splitting method for the Cahn–Hilliard equation," *Numer. Math*, vol. 54, no. 5, pp. 575–590, 1989.

[31] S. Zhang and M. Wang, "A nonconforming finite element method for the Cahn–Hilliard equation," *J. Comput. Phys.*, vol. 229, no. 19, pp. 7361–7372, 2010.

[32] L. Cueto-Felgueroso and J. Peraire, "A time-adaptive finite volume method for the Cahn–Hilliard and Kuramoto–Sivashinsky equations," *J. Comput. Phys.*, vol. 227, no. 24, p. 9985–10017, 2008.

[33] J. Kim and K. Kang, "A numerical method for the ternary Cahn–Hilliard system with a degenerate mobility," *Appl. Numer. Math.*, vol. 59, no. 5, pp. 1029–1042, 2009.

[34] A. Milchev, D. Heermann, and K. Binder, "Monte-Carlo simulation of the Cahn–Hilliard model of spinodal decomposition," *Acta Metall*, vol. 36, no. 2, pp. 377–383, 1988.

[35] J. Kim, K. Kang, and J. Lowengrub, "Conservative multigrid methods for Cahn–Hilliard fluids," *J. Comput. Phys.*, vol. 193, no. 1, pp. 511–543, 2004.

[36] E. Chong and S. Zak, "An introduction to optimization," *Wiley*, 2001.

[37] B. I. Wohlmuth, "Discretization methods and iterative solvers based on domain decomposition," *Springer*, 2001.

[38] W. Gropp, E. Lusk, and A. Skjellem, "Using MPI, portable parallel programming with the message passing interface," *The MIT Press*, 1999.

[39] A. Quarteroni and A. Valli, "Numerical approximation of partial diferential equations," *Springer*, 1994.

[40] S. V. Adve. and J. C. Hart, "Parallel computing research at Illinois: The UPCRC agenda," *Univ. of Illinois at Urbana-Champaign*, 2008.

[41] J. Sanders and E. Kandrot, "CUDA by example. An introduction to general-purpose GPU programming.," *Addison-Wesley*, 2011.

[42] Y. Kim and A. Shrivastava, "CuMAPmapz: A tool to analyze memory access patterns in CUDA," *In Proc. of the 48th Design Automation Conference New York, NY, USA*, pp. 128–133, 2011.

[43] H. El-Rewini and M. Abd-El-Barr, "Advanced computer architecture and parallel processing," *Wiley-Interscience*, 2005.

[44] S. Williams *et al*, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," *Parallel Computing*, vol. 35, no. 3, pp. 178–194, 2009.

Apéndice A

Configuración de un *cluster* en Ubuntu

En esta sección se describe cómo configurar un clúster utilizando el sistema operativo de Ubuntu.

Para la compresión de la guía, se asume un conocimiento básico del uso de la línea de comandos y *clusters*.

En el siguiente ejemplo se consideran dos nodos que se ejecutan en el servidor de Ubuntu con nombres de *host*: ub0 y ub1.

- Crear un usuario en cada nodo que del *cluster*, es importante que todos los usuarios sean creados con las mismas características en cada nodo (contraseña, nombre, grupo y número de usuario).
- Configurar las direcciones IP de cada computadora, para ello hay que editar el archivo *hosts*, *# /etc/hosts* como se observa en la Fig. A.1.

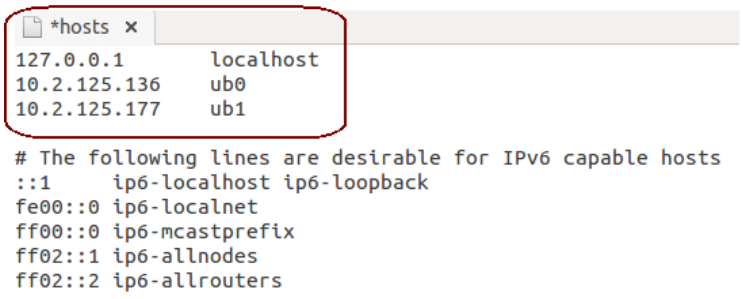


Fig. A.1: Configuración de las IP en el archivo *hosts* en cada nodo del *cluster*.

- Instalar NFS (del inglés, Network File System), el NFS permite al usuario crear una carpeta en el nodo maestro y sincronizarlo en todos los nodos del *cluster*. Esta carpeta se puede utilizar para almacenar programas. Para instalar el servidor NFS en el nodo maestro ejecute *# apt-get install nfs-server*, y en el nodo esclavo ejecute: *# apt-get install nfs-client*.

- Compartir la carpeta del nodo maestro, para esto se edita en el nodo maestro el archivo (*/etc/exports*) tal y como se observa en la Fig. A.2 Acontinuación se reinicia el servidor

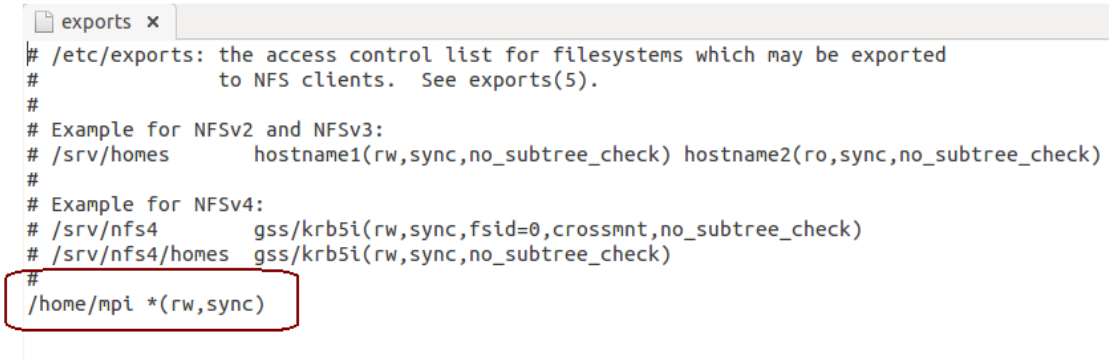


Fig. A.2: Configuración del archivo *exports* en el nodo maestro del *cluster*.

NFS para el nodo maestro y analizar que la configuración fue realizada, para esto se teclea en la terminal: *# service nfs-kernel-server restart*.

- Instalar del servidor SSH (del inglés, Secure SHell). Para esta acción se ejecuta el comando *# apt-get install openssh-server* en todos los nodos del *cluster* para instalar el servidor de OpenSSH.

- Configuración del servidor SSH para la comunicación entre nodos sin contraseña, primero se genera una clave de acceso al servidor RSA, para esto se teclea en la terminal: *"ssh-keygen -t rsa"*. Después de realizar esto se pueden realizar pruebas de acceso al nodo ingresando al directorio del servidor SSH (*cd. ssh*) e ingresando a la carpeta compartida (*ssh ub0* o *ssh ub1*).