UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Estudio del cambio de fase de un material inmerso en un medio de porosidad heterogénea, mediante el método de lattice Boltzmann con múltiples tiempos de relajación, implementado en python-CUDA

T E S I S

QUE PARA OPTAR POR EL GRADO DE

DOCTOR EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

P R E S E N T A

M.C. Benjamín Salomón Noyola García

Tutora:

Dra. Suemi Rodríguez Romo

Facultad de Estudios Superiores Cuautitlán

Ciudad de México, Mayo de 2021

Dedico ésta tesis al único Dios, creador del cielo y de la tierra.

Agradecimientos

- A mi tutora la Dra. Suemi Rodríguez Romo por su consejo y el apoyo invaluable.
- Al CONACYT por el apoyo económico.
- Al posgrado en ciencia e ingeniería de la computación.

Resumen

En éste trabajo de tesis se resuelven dos problemas de cambio de fase, sometidos a condiciones de frontera. En ambos casos se utiliza el Método de lattice Boltzmann con tiempos múltiples de relajación (MLB-TRM) cuya solución es implementada en paralelo. En éste trabajo de tesis se proporciona un software libre que se puede descargar libremente de [1], y se encuentra sustentado en [2].

El primer problema consiste en modelar el cambio de fase de una barra simple, y el segundo en modelar el cambio de fase del Galio (Ga) sólido-líquido inmerso en un medio poroso. En ambos casos se forma una interfaz sólido-líquido que se mueve cuando se alcanza la temperatura de fusión.

Con la solución del problema 1 se valida la implementación del MLB-TRM, el modelo de transferencia de calor, el método basado en la entalpía y también la implementación en python-CUDA. La validación de ésta solución se lleva a cabo mediante la solución analítica.

En el problema 2 se utiliza MLB-TRM para modelar los campos de temperatura y velocidad que se encuentran acoplados. Cuando el tiempo es igual a cero, el Galio inmerso en el medio poroso se encuentra en estado sólido, este se encuentra sometido a condiciones de frontera, en la frontera oeste la temperatura es mayor a la temperatura de fusión, las fronteras norte y sur son adiabáticas, por lo que cuando el tiempo avanza el Galio cambia de fase de sólido a estado líquido y se forma una interfaz sólido-líquido que se mueve en función del tiempo y el espacio. La posición de dicha interfaz se estima mediante el método basado en el cálculo de la entalpía.

La porosidad se obtiene a partir de una imagen, esto se logró transformándola en una imagen binaria donde el 1 representa a los poros y el 0 a representa a la parte sólida a escala de píxeles. La imagen binaria de la matriz porosa se dividió en pequeñas regiones para poder calcular una porosidad semi-local y luego se puede recuperar la porosidad global calculando el promedio de las porosidades semi-locales. La permeabilidad Se estima con la ecuación de Kozeny y al final del tratamiento digital de la imagen, se obtiene un conjunto de porosidades y permeabilidades que

son alimentadas al código principal.

En este trabajo se puede observar el impacto en el comportamiento de la velocidad del fluido cuando las simulaciones son ejecutadas sobre porosidad y permeabilidad homogéneas contra simulaciones cuando la porosidad no es homogénea.

Se muestran experimentos del comportamiento del cambio de fase cuando hay puntos calientes directamente en el medio poroso, considerandos como múltiples fuentes o sumideros.

La solución se implementó en el lenguaje python 3.8 junto a Numba-CUDA, que permite generar código máquina eficiente. La paralelización se llevó a cabo en los GPU's de una tarjeta de video Nvidia. Mientras que la implementación en serie se implementó utilizando python - Numpy.

Abstract

In this thesis, two phase change problems are solved, subject to boundary conditions. In both cases the Multiple Relaxation time lattice Boltzmann (MLB-TRM) is used whose solution is implemented in parallel. In this thesis work, a free software is provided that can be downloaded from [1], and is supported by [2].

The first problem is to model the phase change of a simple rod, and the second is to model the phase change of solid-liquid Gallium (Ga) immersed in a porous media. In both cases a solid-liquid interface is formed that moves when the melting temperature is reached.

With the solution of problem 1, the implementation of the Multiple Relaxation time Lattice Boltzmann, the heat transfer model, the method based on enthalpy and the implementation in python-CUDA are validated. The validation of this solution is carried out with the analytical solution.

In Problem 2 MLB-TRM is used to simulate the temperature and velocity fields that are coupled. At time equal to zero, the Gallium immersed in the porous medium is in a solid state but is also subjected to hot boundary conditions (above melting temperature), so that when time advances, the Gallium changes from phase from solid to a liquid state and an interface is formed that moves. The position of this interface is estimated using the enthalpy-based method.

Porosity is obtained from an image. This is achieved by transforming it into a binary image by labeling the pores as 1 and the solid part at pixel scale as 0. The binary image of the porous matrix was divided into small regions to find the porosity a semi-local and the global porosity can be recovered by calculating the average of all semi-local porosities. Then the permeability is estimated with the Kozeny equation and the set of porosities forms a thick matrix that is fed to the main code to calculate the phase change and its temperature and velocity fields.

In this work, the impact on the behavior of the fluid velocity can be observed when the simulations run with homogeneous porosity and permeability compared to when there is no porosity and homogeneous permeability.

Experiments on the behavior of the phase change are shown when there are hot spots directly in the porous medium, considered as multiple sources or sinks.

The solution was implemented in the python programming language together with Numba-CUDA, which allows the generation of efficient machine code. The parallelization was carried out on the GPU's of an Nvidia video card. Whereas the serial implementation was implemented using python - Numpy.

Índice general

Índice de figuras

Nomenclatura

α	Difusividad térmica
β	Coeficiente de expansión térmica
λ	Relación de difusividad térmica
ρ	Densidad
ρ_0	Densidad promedio
ϕ	Porosidad
φ	Fracción del líquido del PCM
σ	Relación de capacidad térmica
$\mathbf{\Lambda}$	Matriz diagonal de relajación
ν	Viscosidad cinemática
τ_T, τ_v	Tiempos de relajación adimensionales

Tabla 1: Letras griegas

m	Cambio de fase (melting)
s	Fase sólida
l	Fase líquida
e	Efectivo
h	Caliente (hot)
c	Frío (cold)

Tabla 2: Índices

MLB-TRM	Método de lattice Boltzmann con tiempos múltiples de relajación
CUDA	Arquitectura unificada de dispositivos de cómputo
MCF	Material que cambia de fase
MLUPS	Millones de lattices actualizados por segundo
LBGK	Modelo de lattice Boltzmann Bhatnagar-Gross-Krook
GPU	Unidad de procesamiento gráfico
Ra	Número de Rayleigh
Da	Número de Darcy
N_x, N_y, N_z	Longitud característica en direcciones x, y, z
H	Entalpía
Cp_l	Calor específico del líquido
Cp_s	Calor específico del sólido
K	Permeabilidad
F_0	Número adimensional de Fourier
S_t	Número de Stefan
Ma	Número de Mach
c	Velocidad del sonido
f_l	Fracción de líquido
$\mathbf{j}$	Vector unitario en la dirección y
J	relación de viscosidades
C_f	Coeficiente inercial
$\mathbf{u}$	Componentes de la velocidad
$\mathbf{F}$	Fuerza total de cuerpo
$\mathbf{G}$	Fuerzas externas de cuerpo
T	Temperatura
$\mathbf{m}$	Función de distribución en el espacio de los momentos
$\mathbf{M}$	Matriz de transformación lineal
La	Calor latente
$\mathbf{I}$	Matriz identidad
$\mathbf{S}$	Componentes del término de la fuerza
erf	Función error
erfc	Función error complementaria

Tabla 3: Abreviaciones

Introducción

El método de lattice Boltzmann es visto como un método numérico de multiescala, que a partir de la escala mesoscópica puede reproducir propiedades macroscópicas y que tiene sus orígenes en el método del autómata celular lattice-gas (LGA) por sus siglas en inglés [3]. La diferencia respecto a la simulación con métodos numéricos convencionales es que estos están basados en la discretización de las ecuaciones macroscópicas, mientras que el método de lattice Boltzmann está basado en la cinética de las ecuaciones mesoscópicas para pseudo partículas o campos de funciones de distribución.

El método de lattice Boltzmann cuenta con un fundamento teórico bien establecido y es utilizado en diversos campos de la ciencia con gran éxito, sin perder la simplicidad que lo caracteriza [4].

Actualmente en la literatura hay simulaciones de materiales embebidos en un medio poroso, que cambian de fase, pero todos ellos basados en el operador de colisión BGK [5]. Sin embargo cuando se aplica el operador de colisión BGK, se tienen deficiencias, tales como la inestabilidad numérica a bajas viscosidades, y la inexactitud al tratar con condiciones de frontera complejas [6, 7]. En este trabajo de investigación se utiliza el método de tiempos de relajación múltiples (MLB-TRM) para modelar los campos acoplados de temperatura y velocidad. En la comunidad científica se ha aceptado que MLB-TRM es superior a BGK en términos de estabilidad numérica y exactitud [8].

El objetivo de éste trabajo de investigación es utilizar MLB-TRM como base para simular el cambio de fase sólido-líquido del Galio inmerso en un medio poroso, sometido a convección natural.

Hay múltiples aplicaciones para este trabajo, como el almacenamiento de energía, procesos de soldadura y fundición, congelación y derretimiento de suelos, congelamiento de suelos, entre otras aplicaciones, ver la referencia: [8].

La solución de los problemas de cambio de fase es compleja, debido a las no linealidades causadas por la formación y propagación de la interfaz sólido líquido, que se está moviendo con el tiempo, además de las complejidades físicas y computacionales.

En la solución del problema de cambio de fase, se resuelve la ecuación de energía

y de momento que se encuentran acopladas. Actualmente hay solución analítica para la ecuación de energía, que en este caso sirve para validar el modelo basado en la entalpía.

En el pasado se utilizaron los métodos numéricos convencionales (diferencias finitas, volúmenes finitos y elementos finitos) para modelar el cambio de fase sólido-líquido con convección natural debido a la gravedad y la transferencia de calor en el medio poroso basado en la discretización de algunos modelos semi-empíricos, por ejemplo el modelo de Darcy, el modelo de Darcy-Brinkman, y el modelo de Darcy-Forchheimer. [9, 10]

En el modelo se emplea la aproximación de Brinkman-Forchheimer, también conocido como modelo generalizado no Darciano el cual sirve para describir la transferencia de momento en el medio poroso. A partir del método basado en la entalpía se puede encontrar la fracción de líquido en el medio poroso y por lo tanto determinar la interfaz sólido-líquido que constantemente se está moviendo en el medio poroso.

La primera aportación de este trabajo de tesis es la implementación paralela en Python, que es un lenguaje de programación de alto nivel que permite la generación de código máquina eficiente, utilizando bibliotecas como Numba y también permite la implementación sobre los GPU's de una tarjeta de video NVIDIA logrando speedups superiores a 100. Hace algunos años, el poder computacional en el orden de los teraflops solo estaba disponible para las supercomputadoras, pero ahora hay tarjetas de video que corren en computadoras personales con capacidades superiores a los 5 teraflops y con menores requerimientos de energía que un conjunto de unidades centrales de procesamiento y además con rendimientos similares.

Para aprovechar las capacidades que proporcionan las tarjetas de vídeo, es necesario desarrollar software adecuado, con algoritmos eficientes. En este trabajo de tesis se desarrollaron algoritmos que se benefician de la arquitectura de CUDA. Generalmente éste tipo de algoritmos se desarrollan para solucionar un problema en específico, en este caso se resuelve el problema de cambio de fase con convección natural dentro de una matriz porosa basado en MLB-TRM sobre CUDA.

La segunda aportación es que el problema de cambio de fase se implementa en Python, que es un lenguaje de fuente abierta (open surce), junto a Numba, Numba-CUDA, Numpy, Scipy, Sympy y Matplotlib. Queda en evidencia el potencial de estas nuevas herramientas y además se muestra que construir código máquina con Numba es muy sencillo y se obtienen rendimientos similares que los obtenidos con fortran o C++. Aprovechando la facilidad y limpieza que implica programar en Python, sin sacrificar eficiencia.

La tercera aportación es la inclusión de una estructura mesoscópica de un medio poroso natural, en cuyo interior se encuentra el material que cambia de fase, donde esta es obtenida de la imagen de un medio poroso natural. A partir de un tratamiento digital de la imagen, se encuentra una matriz de porosidades y permeabilidades que luego son introducidas al código principal.

La cuarta aportación es el estudio del comportamiento de la interfaz sólido-líquido cuando hay fuentes y/o sumideros en el dominio. La magnitud de las fuentes o sumideros es obtenida aleatoriamente y la posición es obtenida a partir de una distribución normal.

Capítulo 1

Introducción al método de lattice Boltzmann

1.1. Método de lattice Boltzmann

El método de lattice Boltzmann se originó a partir del modelo de lattice gas cellular automata (LGCA), luego en 1988 aparecieron las primeras ecuaciones de lattice Boltzmann que fueron propuestas por McNamara y Zanetti [11] con el objetivo de solucionar los problemas de ruido estadístico y los inconvenientes de los autómatas celulares que tenía su antecesor LGCA, el objetivo era reemplazar los números booleanos de ocupación n_i por sus correspondientes promedios en ensamble $f_i = \langle n_i \rangle$. Luego, en el año de 1997 He y Luo [6] desarrollaron la ecuación de Lattice Boltzmann a partir de la ecuación continua de Boltzmann, discretizando el tiempo y las variables del espacio de fases (coordenadas espaciales y velocidades). Esto demostró que la ecuación de lattice Boltzmann es una aproximación en diferencias finitas de la ecuación de la ecuación continua de Boltzmann, y es independiente de LGCA [12].

El método de lattice Boltzmann se ha utilizado en la ciencia y en la ingeniería para simular diferentes problemas de flujo de fluidos, hay muchas aplicaciones en la literatura que demuestran su eficacia y potencial, desde flujos laminares hasta flujos altamente turbulentos, flujos sobre geometrías móviles y complejas, flujos multifásicos con diferentes fenómenos acoplados, entre muchas otras aplicaciones.

1.2. Ecuación de transporte de Boltzmann

Un sistema se puede explicar mediante una descripción estadística utilizando una función de distribución de probabilidad $f(r, c, t)$ la cual es el número de partículas que hay en un tiempo dado y se encuentran entre r y $r + dr$ con velocidades entre c y $c + dc$. Si existe una fuerza externa que actúa sobre las partículas, entonces cambiará la velocidad de c a $c + Fdt$ y su posición cambia de r hasta $r + cdt$, considerando al momento Fdt por unidad de masa. $f(r, c, t)$ es el número de partículas antes de aplicar una fuerza externa, mientras que $f(r + cdt,\, c + Fdt,\, t + dt)$ es el número de partículas después de aplicar una fuerza externa. Cuando se lleva a cabo la colisión entre las partículas [13]. Cuando hay colisión entre las partículas, existe una tasa de cambio en el número de partículas dentro del intervalo $drdc$, la cual se mide por el operador de colisión Ω. que se puede expresar con la ecuación 1.1

$$f(r + cdt,\, c + Fdt,\, t + dt)\, dr\, dc - f(r, c, t)\, dr\, dc = \Omega(f)\, dr\, dc\, dt \tag{1.1}$$

Dividiendo la ecuación 1.1 por $dt\, dr\, dc$ y tomando el límite $dt \to 0$, por cuestión de simplicidad a lo largo de ésta tesis se escribe la función de distribución como f, se obtiene la ecuación 1.2

$$\frac{df}{dt} = \Omega(f) \tag{1.2}$$

La velocidad de cambio de la función de distribución es igual a la tasa de colisiones.

La velocidad total de cambio de la función de distribución se expresa como:

$$df = \frac{\partial f}{\partial r}dr + \frac{\partial f}{\partial c}dc + \frac{\partial f}{\partial t}dt$$
$$\frac{df}{dt} = \frac{\partial f}{\partial r}\frac{dr}{dt} + \frac{\partial f}{\partial c}\frac{dc}{dt} + \frac{\partial f}{\partial t}$$

La ecuación de transporte de Boltzmann cuando hay una fuerza externa F tiene la siguiente forma:

$$\frac{df}{dt} + \frac{\partial f}{\partial r} \cdot c + \frac{F}{m} \cdot \frac{\partial f}{\partial c} = \Omega$$

Y cuando no hay fuerza externa la ecuación se escribe como se muestra en la ecuación 1.3:

$$\frac{df}{dt} + \nabla f = \Omega \tag{1.3}$$

El método de lattice Boltzmann se basa en la voxelización equidistante del espacio, de manera que las celdas solo interactúan con sus vecinos directos. Este método opera

a nivel mesoscópico, en las celdas residen distribuciones de partícula que viajan en un conjunto discreto de direcciones de movimiento. El número de direcciones que forman este conjunto y el número de dimensiones que determinan el modelo de lattice Boltzmann que se utiliza, siendo los estenciles más comunes D_2Q_9 y D_3Q_{19} en 2D y 3D respectivamente [14].

1.2.1. Factor de colisión propuesto por Bhatnagar, Gross y Krook

Se aproxima el factor de colisión como lo propusieron Bhatnagar, Gross and Krook (BGK) en 1954:

$$\Omega = \omega(f^{eq} - f) = \frac{1}{\tau}(f^{eq} - f)$$

donde ω es la frecuencia de colisiones y τ es el tiempo de relajación, la función de distribución de equilibrio local es denotada por f^{eq}. Sustituyendo la aproximación BGK a la ecuación de Boltzmann sin fuerzas externas se obtiene:

$$\frac{\partial f}{\partial t} + c \cdot \nabla f = \frac{1}{\tau}(f^{eq} - f) \tag{1.4}$$

En el método de Lattice Boltzmann la ecuación 1.4 se discretiza y es válida a lo largo de direcciones especificas.

$$\frac{\partial f_i}{\partial t} + c_i \cdot \nabla f_i = \frac{1}{\tau}(f_i^{eq} - f_i) \tag{1.5}$$

La ecuación 1.5 es el caballo de batalla del método de Lattice Boltzmann y tiene las siguientes características [13]

- Es una ecuación diferencial parcial lineal
- Se parece a una ecuación de advección con un término fuente
- El lado izquierdo representa el paso de propagación
- El lado derecho representa el paso de colisión

La ecuación 1.5 discretizada es:

$$f_i(r + c_i\Delta t,\, t + \Delta t) - f_i(r, t) = \frac{1}{\tau}[f_i^{eq}(r, t) - f_i(r, t)] \tag{1.6}$$

El algoritmo básico de LB consiste en dos pasos: propagación y colisión más la actualización de los nodos

En el paso de propagación se copian a las celdas vecinas las fracciones de partículas que se mueven en la dirección correspondiente, matemáticamente escrito como:

$$f_i^*(r + c_i\Delta t,\, t + \Delta t) = f_i(x, t) \tag{1.7}$$

donde Δt denota el salto temporal y está normalizado a 1, así como también lo está el incremento espacial, lo cual hace posible que la propagación se describa como una simple operación de copia. El valor de f_i^* es temporal y no se guarda en realidad. Sin embargo, únicamente la propagación no simula el comportamiento real, de modo que se necesita el paso de colisión, Se tienen en cuenta las diferentes distribuciones de partículas sopesadas.

El paso de colisión consiste en relajar las distribuciones de partículas (fuera del equilibrio local) de las celdas hacia su estado de equilibrio. Esto se consigue sopesando cada f_i con su correspondiente f_i^{eq} usando:

$$f_i(r, t) = f_i^*(r, t) - \omega(f_i^*(r, t) - f_i^{eq})$$

y posteriormente se implementan las condiciones de frontera.

En el modelo de lattice Boltzmann se utilizan diversos arreglos para referirnos a la dimensión y al número de velocidades, se utiliza D_nQ_m, donde n representa la dimensión del problema y m se refiere al número de velocidades (número de enlaces con otros nodos contenidos en la celda). Por ejemplo D_2Q_5 el número total de partículas en cualquier instante de tiempo no puede ser mayor a 5. Los factores de peso son ω_i son 6/12, 2/12, 2/12, 1/2, 1/2 para f_0, f_1, f_2, f_3, f_4, f_5 respectivamente. Se puede observar el esténcil unitario D_2Q_5 en la figura 1.1

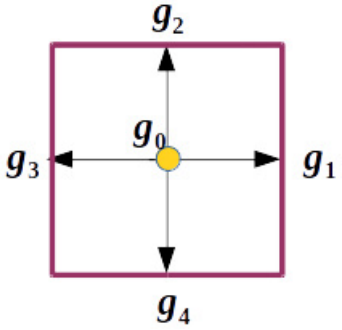


Figura 1.1: Sténcil unitario D_2Q_5

El sténcil unitario D_2Q_9 es muy común, especialmente para resolver problemas de flujo de fluidos, los factores de peso son ω_i son 4/9, 1/9, 1/9, 1/9, 1/9, 1/36, 1/36, 1/36, 1/36 para f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8 respectivamente. En la figura 1.2 se muestra el esténcil unitario D_2Q_9.

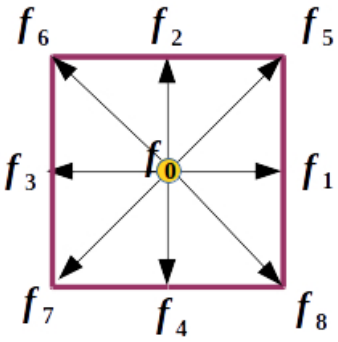


Figura 1.2: Sténcil unitario D_2Q_9

1.3. Condiciones de frontera

El comportamiento de la dinámica de fluidos depende en gran medida de su entorno, dicha influencia se escribe matemáticamente a través de las condiciones de frontera.

Sea el flujo de un fluido que se encuentra en un dominio Ω rodeado por fronteras $\partial\Omega$, la formulación de las condiciones de frontera consiste en encontrar una expresión apropiada de la relación de la función de distribución que entra como " $f_i^<$respecto a una $f_i^>$ que sale en dirección i. Las condiciones de frontera consisten en determinar las funciones de distribución en las fronteras, por ejemplo para un dominio con esténcil tipo D_2Q_5, sería como determinar la función de distribución representada por las flechas rojas, como se muestra la figura 1.3

Encontrar la función de distribución en la frontera, para un dominio con esténcil tipo D_2Q_9, como se muestra la figura 1.4

En esta tesis se implementan las siguientes condiciones de frontera:

- Periódicas
- Bounce-Back
- Dirichlet
- Adiabáticas
- Neumann

Los pesos utilizados para D_2Q_5 son:

$$\tilde{\omega}_0 = \frac{1-\overline{w}}{5}$$
$$\tilde{\omega} = \frac{4+\overline{w}}{20} \tag{1.8}$$

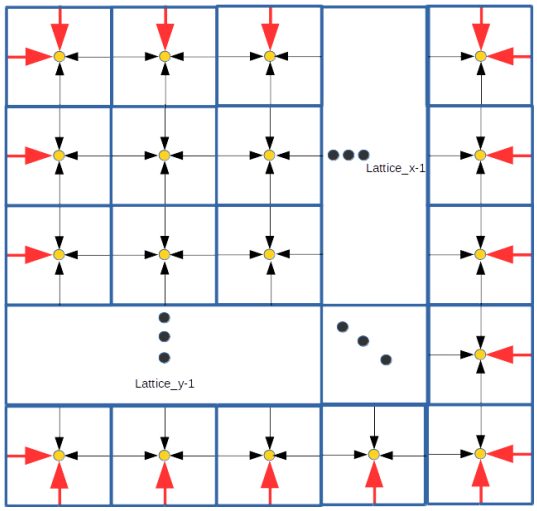


Figura 1.3: Condiciones de frontera, esténcil D_2Q_5

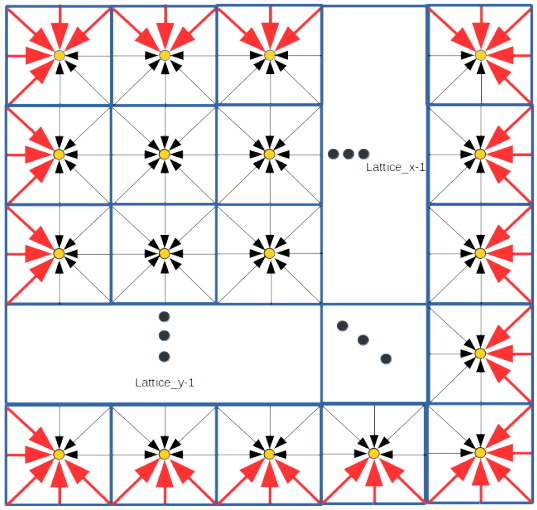


Figura 1.4: Condiciones de frontera, esténcil D_2Q_9

en este trabajo: $\overline{w} = -2$

Los pesos utilizados para D_2Q_9 son:

$$\omega_0 = \frac{4.}{9.}$$

$$\omega_{1-4} = \frac{1.}{9.} \tag{1.9}$$

$$\omega_{5-8} = \frac{1.}{36.}$$

1.3.1. Condiciones de frontera periódicas

Las condiciones de frontera periódicas son las más sencillas para que sean implementadas, se utilizan para aislar una condición de flujo repetida. Por ejemplo, en este trabajo se aplican condiciones de frontera periódicas en la simulación del cambio de fase de una barra delgada y horizontal, donde los efectos de la frontera lado norte y sur son los mismos. En los anexos se muestra la implementación en python.

1.3.2. Condiciones de frontera Bounce-Back

Éste método es utilizado para modelar condiciones de frontera en estado estacionario o en movimiento, en éste trabajo de tesis las fronteras se encuentran en estado estacionario. El método es bastante simple y consiste en hacer rebotar a una partícula que choca con la frontera, y devolverla hacia el dominio. Por ejemplo, en el lado oeste en la figura 1.2 faltan las funciones de distribución en color rojo, en éste caso se toma que $f_1 = f_3$, $f_5 = f_7$ y $f_8 = f_6$.

1.3.3. Condiciones de frontera tipo Dirichlet

En el lado oeste de la figura 1.3 se necesita determinar la función de distribución f_1 marcada en rojo. La condición de frontera tipo Dirichlet consiste en determinar las funciones de distribución desconocidas, que se encuentran a partir de las funciones de distribución que si se conocen. En este trabajo de investigación se utiliza el esquema D_2Q_5 para modelar el campo de temperatura, mientras que D_2Q_9 para modelar el campo de velocidades, cabe mencionar que ambos fenómenos se encuentran acoplados. La condición de frontera tipo Dirichlet se utiliza para el campo de temperatura. De modo que para encontrar g_1 en el lado oeste se calcula como:

$$T_B = \sum_{i=5}^{5} g_i \; = \; g_0 + g_2 + g_3 + g_4 + g'(w_1)$$

Entonces:

$$g' = \frac{T_B - (g_0 + g_2 + g_3 + g_4)}{w_1}$$

Si se desea encontrar g_1 en el lado oeste, entonces:

$$g_1 = \overline{\omega}_1 \, g_1$$

donde $g_i | i = 0, 1, 2, 3, 4$ son las funciones de distribución utilizadas en D_2Q_5 para modelar el campo de temperatura y $f_i | i = 0, 1, 2, 3, 4, 5, 6, 7, 8$ son utilizadas en D_2Q_9 para modelar el campo de velocidades.

1.3.4. Condiciones de frontera adiabáticas

En este trabajo de investigación se aplican condiciones de frontera adiabáticas en las paredes norte y sur cuando se modela el cambio de fase de un material inmerso en un medio poroso. Esto es porque no fluye calor a través de estas fronteras, es decir que el flujo de calor a través de dichas fronteras es cero:

$$\frac{\partial T}{\partial y} = 0$$

Utilizando una aproximación con diferencias finitas se obtiene:

$$\frac{T(i, j+1) - T(i, j)}{\Delta y} = 0$$

Simplificando se obtiene:

$$T(i, j+1) = T(i, j)$$

En términos de la función de distribución se puede escribir.

$$g_1(i,j)+g_2(i,j)+g_3(i,j)+g_4(i,j) = g_1(i,j+1)+g_2(i,j+1)+g_3(i,j+1)+g_4(i,j+1)$$

Se puede asumir que:

$$\begin{aligned} g_1(i,j) &= g_1(i,j+1) \\ g_2(i,j) &= g_2(i,j+1) \\ g_3(i,j) &= g_3(i,j+1) \\ g_4(i,j) &= g_4(i,j+1) \end{aligned} \tag{1.10}$$

la ecuación 1.10 aplica para las fronteras norte y sur, con los índices adecuados.

1.3.5. Lattice Boltzmann con tiempos de relajación múltiple

El método de lattice Boltzmann que utiliza el modelo BGK basado en un tiempo de relajación, como se describió anteriormente, es la forma más simple y mas popular de dicho modelo, sin embargo esta simplicidad tiene algunas deficiencias tales como problemas de inestabilidad numérica y número fijo de Pradtl. Para afrontar estas deficiencias del modelo BGK se desarrolló el modelo de tiempo de relajación múltiple (MRT), que se ha utilizado con mucho éxito en la simulación de varios problemas de flujo de fluidos [15].

El método de lattice Boltzmann con tiempos múltiples de relajación (MLB-TRM) fue propuesto por Dhumieres [16]. Éste permite modelar altos números de Reynolds en retículos relativamente pequeños, sin perder la simplicidad del método de lattice Boltzmann.

En cuanto a MLB-TRM es más conveniente que el operador de colisión contenido en la ecuación de transporte de Boltzmann sea operado en el espacio de momentos que en el espacio de velocidades. Reescribiendo la ecuación de transporte de Boltzmann en el esquema tiempos de relajación múltiples es [13]:

$$f_i(x + c\,\Delta t,\, t + \Delta t) - f_i(x,t) = -\mathbf{M^{-1}\Lambda}[\mathbf{m}(\mathbf{x},t) - \mathbf{m}^{(eq)}(\mathbf{x},t)]$$

donde $m(x,t)$ y $m^{eq}(x,t)$ son vectores de momentos $m = (m_0,\, m_1,\, m_2,\, ..m_n)^T$. La matriz de relajación $\mathbf{\Lambda}$ es una matriz diagonal.

1.4. Transformaciones lineales entre los espacios de momentos y velocidades

La transformación lineal del espacio velocidad al espacio momento se lleva a cabo de la siguiente manera:

$$\mathbf{m} = \mathbf{Mf} \quad y \quad \mathbf{f} = \mathbf{M^{-1}m}$$

Donde m se encuentra en el espacio de los momentos y f en el espacio de las velocidades. $\mathbf{M}$ es una matriz ortogonal de transformación lineal que puede ser construida con el procedimiento de ortogonalización de Gram-Schmidt [**?**] y se encuentra bien definida en la literatura [8, 13, 15].

En este trabajo de investigación se utilizan las siguientes matrices de transformación lineal:

$$\mathbf{N_{D_2Q_5}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ -4 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & 1 & -1 \end{pmatrix} \tag{1.11}$$

La ecuación 1.11 es la matriz de transformación lineal para D_2Q_5.

$$\mathbf{M_{D_2Q_9}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -4 & -1 & -1 & -1 & -1 & 2 & 2 & 2 & 2 \\ 4 & -2 & -2 & -2 & -2 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & -2 & 0 & 2 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \\ 0 & 0 & -2 & 0 & 2 & 1 & 1 & -1 & -1 \\ 0 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \end{pmatrix} \tag{1.12}$$

La ecuación dada en 1.12 es la matriz de transformación lineal para D_2Q_9.

En éste método se lleva a cabo la colisión en el espacio de los momentos, mientras que la propagación en en el espacio de las velocidades. En el capítulo 4 se abordará el detalle de la implementación de este método.

Capítulo 2

Marco teórico

2.1. Ecuaciones gobernantes

En éste trabajo de investigación se asume que la matriz porosa y el material que cambia de fase (Galio en este caso) son isotrópicos, homogéneos y rígidos. El término de flotabilidad de la ecuación de momento se rige por la aproximación de Boussinesq, y las variaciones de la densidad durante el cambio de fase pueden ser despreciadas, el fluido es incompresible y laminar. De acuerdo a lo mencionado anteriormente, las ecuaciones gobernantes para modelar el cambio de fase con convección y transferencia de calor en un medio poroso, son como se muestra a continuación:

$$\nabla \cdot \mathbf{u} = 0. \tag{2.1}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\left(\frac{\mathbf{u}}{\varphi}\right) = -\frac{1}{\rho_0}\nabla(\varphi p) + \nu_\epsilon \nabla^2 \mathbf{u} + \mathbf{F}. \tag{2.2}$$

$$\sigma \frac{T}{t} + \mathbf{u} \cdot \nabla T = \nabla \cdot (\alpha_\epsilon \nabla T) - \phi \frac{L_\alpha}{C_{pl}} \frac{\partial f_l}{t}. \tag{2.3}$$

Donde $\mathbf{u}$, p, T son la velocidad promedio, presión y temperatura respectivamente, ρ_0 es la densidad media del galio, ν_ϵ es la viscosidad efectiva, α_ϵ es la difusividad térmica efectiva, σ es la relación de capacidad térmica, La es el calor latente de fusión del Galio, C_{pl} es el calor específico del líquido, ϕ es la porosidad, f_l es la fracción del Galio líquido dentro del poro. $\varphi = \phi\, f_l$ es la fracción de líquido dentro de un elemento de volumen [8]. Debido a que existen fuerzas externas dentro de medio poroso, la fuerza total de cuerpo es $\mathbf{F} = (F_x, F_y)$ y está dadas por la siguiente ecuación:

$$\mathbf{F} = -\frac{\varphi \nu_l}{K}\mathbf{u} - \frac{\varphi C_F}{\sqrt{K}}|\mathbf{u}|\mathbf{u} + \varphi \mathbf{G}. \tag{2.4}$$

donde ν_l es la viscosidad del fluido, $K = \frac{\varphi^3 d_m^2}{175(1-\varphi)^2}$ es la permeabilidad (aquí se utiliza la ley de Kozeny, d_m es el diámetro medio de la partícula del sólido), C_F es el coeficiente inercial calculado como $C_F = \frac{1{,}75}{\sqrt{175\varphi^3}}$, basado en la aproximación de Boussinesq y $\mathbf{G}$ son las fuerzas de cuerpo.

$$\mathbf{G} = g\beta(T - T_0)\mathbf{j}$$

donde g es la aceleración gravitacional, β es el coeficiente de expansión térmica. Se utilizan las siguientes variables adimensionales:

$$\mathbf{U} = \frac{\mathbf{u}H}{\alpha_l},\ F_0 = \frac{t\alpha_l}{H^2},\ \lambda = \frac{\alpha_e}{\alpha_l},\ St = \frac{C_{pl}\Delta T}{L_a},\ Pr = \frac{v_l}{\alpha_l},\ Ra = \frac{g\beta\Delta T H^3}{v_l\alpha_l},\ Da = \frac{K}{H^2} \tag{2.5}$$

A continuación se muestran las ecuaciones gobernantes adimensionales:

$$\nabla \cdot \mathbf{U} = 0 \tag{2.6}$$

$$\frac{1}{\varphi Pr}\left[\frac{\partial \mathbf{U}}{\partial F_0} + (\mathbf{U}\cdot\nabla)\left(\frac{\mathbf{U}}{\varphi}\right)\right] = -\nabla P + \frac{J}{\varphi}\nabla^{\mathbf{2}}\mathbf{U} \tag{2.7}$$

$$-\left(\frac{1}{Da} + \frac{C_F}{Pr\sqrt{Da}}|\mathbf{U}|\right)\mathbf{U} + Ra\theta\mathbf{j}$$

$$\sigma\frac{\partial\theta}{\partial F_0} + \mathbf{u}\cdot\nabla\theta = \nabla\cdot(\lambda\nabla\theta) - \frac{\phi}{St}\frac{\partial f_l}{\partial F_o}. \tag{2.8}$$

donde F_0 es el número de Fourier (tiempo adimensional), $\theta = \frac{T-T_c}{\Delta T}$ es la temperatura adimensional, $\Delta T = (T_h - T_c)$ es una diferencia de temperatura (en éste caso es la temperatura característica), $P = \frac{pH^2}{(\rho\nu)_l}$ es la presión adimensional, λ es la relación de la difusividad térmica efectiva respecto a la difusividad térmica del líquido, St es el número de Stefan, Pr es el número de Prandtl, Ra es el número de Rayleigh, Da es el número de Darcy, J es la relación de viscosidades, H es la longitud característica.

2.2. MLB-TRM para modelar el cambio de fase solido-líquido en un medio poroso

En ésta sección se aplica el MLB-TRM en tres casos:

- La transferencia de momentum (Se utiliza un estencil D_2Q_9)
- La transferencia de calor (Se utiliza un estencil D_2Q_5 ya que es suficiente y adecuado para modelar el campo de temperatura [8])

- La transición del cambio de fase del Galio sólido a líquido

En el MLB-TRM, el paso de colisión se ejecuta en el espacio de los momentos mientras que la propagación se ejecuta en el espacio de la velocidad.

2.2.1. Transferencia de momento

Para modelar el campo de flujo, en éste trabajo se utiliza el estencil D_2Q_9, y el MLB-TRM se puede escribir de la siguiente manera: [**?**, 17, 18]

$$\begin{aligned} \mathbf{f}\left(\mathbf{x}+\mathbf{e}\delta_t, t+\delta_t\right) &- \mathbf{f}\left(\mathbf{x},t\right) = \\ -\mathbf{M}^{-1}\mathbf{\Lambda}\left[\mathbf{m}(\mathbf{x},t)-\mathbf{m}^{eq}(\mathbf{x},t)\right] &+ \mathbf{M^{-1}}\delta_t\left(\mathbf{I}-\frac{\mathbf{\Lambda}}{2}\right)\mathbf{S} \end{aligned} \tag{2.9}$$

Donde M es una matriz de transformación ortogonal de tamaño 9×9, $\mathbf{\Lambda} = \mathbf{M}\,\tilde{\mathbf{\Lambda}}\,\mathbf{M^{-1}}$ es una matriz de relajación diagonal y positiva de tamaño 9×9, y $\tilde{\mathbf{\Lambda}}$ es la matriz de colisión. Dado que se está utilizando un estencil D_2Q_9 los vectores columna $\mathbf{m}, \mathbf{f}, \mathbf{S}$ y $\mathbf{m^{eq}}$ son de tamaño 9. $f_i(\mathbf{x},t)$ es un componente de $\mathbf{f}(\mathbf{x},t)$ la cual es una función de distribución de probabilidad con una velocidad discreta $\mathbf{e}_i = (e_{ix}, e_{iy})$ en el tiempo t y la posición $\mathbf{x} = (x,y)$. $\mathbf{m}(\mathbf{x},t)$ son los momentos y $\mathbf{m^{eq}}(\mathbf{x},t)$ son los momentos en el equilibrio en el tiempo t y la posición $\mathbf{x}$, S_i es el componente del término de la fuerza $\mathbf{S}$. Para obtener la función de distribución $\mathbf{f}$ se hace una transformación lineal del espacio de los momentos al espacio de las velocidades mediante la ortogonalización de Gram-Schmidt:

$$\mathbf{f} = \mathbf{M^{-1}}\,\mathbf{m}$$

La matriz $\mathbf{M}$ tiene la siguiente forma:

$$\mathbf{M} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -4 & -1 & -1 & -1 & -1 & 2 & 2 & 2 & 2 \\ 4 & -2 & -2 & -2 & -2 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & -2 & 0 & 2 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \\ 0 & 0 & -2 & 0 & 2 & 1 & 1 & -1 & -1 \\ 0 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \end{pmatrix} \tag{2.10}$$

De manera general la matriz de relajación tiene la forma

$$\mathbf{\Lambda} = diag\left(1, s_e, s_e, 1, s_q, 1, s_q, s_v, s_v\right)$$

donde $s_i \in (0,2)$ y para éste trabajo de tesis se utiliza $s_e = 1{,}1$ y $s_q = 1{,}2$ que trabajan adecuadamente [8].

La matriz de relajación para el campo de velocidades y en éste trabajo de investigación tiene la siguiente forma:

$$\mathbf{\Lambda} = diag(1, 1{,}1, 1, 1{,}1, 1, 1, 1{,}2, 1, 1{,}2, 1/\tau_v, 1/\tau_v) \tag{2.11}$$

Y el tiempo de relajación adimensional en términos de BGK $\tau_v = 1{,}5$ corresponde al inverso de la viscosidad cinemática ν_v. Dada la ecuación 2.9, a continuación se muestran los componentes del término de la fuerza $\mathbf{S}$.

$$S_0 = 0,\ S_1 = \frac{6\rho_0 \mathbf{u}\cdot\mathbf{F}}{\varphi},\ S_2 = -\frac{6\rho_0 \mathbf{u}\cdot\mathbf{F}}{\varphi},\ S_3 = \rho_0\, F_x,\ S_4 = -\rho_0\, F_x,$$

$$S_5 = \rho_0\, F_y,\ S_6 = -\rho_0\, F_y,\ S_7 = \frac{2\rho_0(u_x F_x - u_y F_y)}{\varphi},\ S_8 = \frac{\rho_0(u_x F_y - u_y F_x)}{\varphi}$$

En el MLB se pueden llevar a cabo las operaciones en dos partes. Primero el paso de colisión se puede operar en el espacio de los momentos y segundo la propagación en el espacio de las distribuciones de probabilidad. Éste concepto se utiliza a lo largo de toda la tesis.

Se asume que nuestro sistema no está localmente lejos del equilibrio; luego, el lado derecho de la ecuación 2.9 se linealiza y representa el paso de colisión que ocurre en cada una de las celdas. A pesar de esto, el sistema global puede estar alejado del equilibrio. El lado izquierdo de la ecuación 2.9 representa el paso de propagación hacia la celda vecina. Para operar el paso de las colisiones en el MLB-TRM se llevan a cabo en el espacio de los momentos como se muestra en la ecuación 2.12:

$$\mathbf{m}^{+}(\mathbf{x},\, t) = \mathbf{m}(\mathbf{x},\, t) - \Lambda\left[\mathbf{m}(\mathbf{x}, t) - \mathbf{m}^{(\mathbf{eq})}(\mathbf{x}, t)\right] + \delta_t\left(\mathbf{I} - \frac{\Lambda}{2}\right)\mathbf{S} \tag{2.12}$$

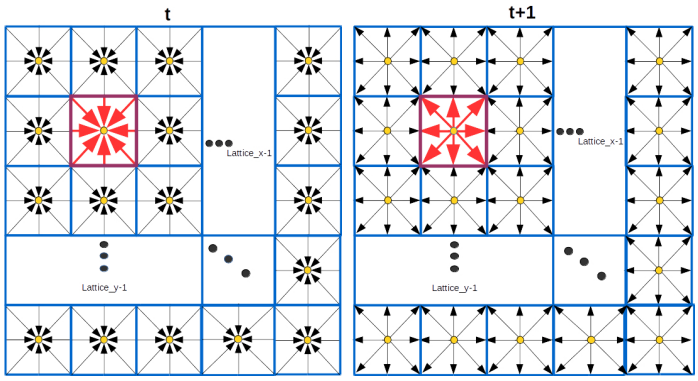
mientras que el paso de propagación (en el espacio de las distribuciones de probabilidad) se lleva a cabo con la ecuación 2.13:

$$f_i(\mathbf{x} + \mathbf{e}_i\delta_t,\, t + \delta_t) = f_i^{+}(\mathbf{x}, t), \tag{2.13}$$

donde $f_i^{+}(\mathbf{x}, t)$ se puede determinar mediante una transformación lineal como se observa en la ecuación 2.14

$$f^{+}(\mathbf{x}, t) = \mathbf{M}^{-1}\mathbf{m}^{+}(\mathbf{x}, t). \tag{2.14}$$

La ecuación 2.12 se refiere al paso de colisión, que mediante un esquema se puede representar con la figura 2.1.



(a) Colisión en t (b) Colisión en $t+1$

Figura 2.1: Paso de colisión del MLB-TRM

Mientras que la ecuación 2.13 que corresponde al paso de propagación se puede representar con la figura 2.2.

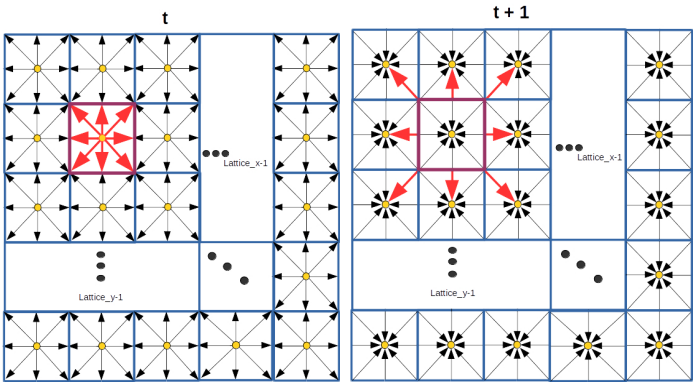
Los componentes de $f_i^{(eq)}$ para la función de distribución f_i, con el estencil D_2Q_9 se calculan de la siguiente manera:

$$f_i^{(eq)} = \omega_i \left\{ \rho + \rho_0 \left[\frac{\mathbf{e}_i \cdot \mathbf{u}}{c_s^2} + \frac{(\mathbf{e}_i \cdot \mathbf{u})^2}{2c_s^4 \varphi} + \frac{|\mathbf{u}|^2}{2c_s^2 \varphi} \right] \right\}, \tag{2.15}$$

los componentes $\{m_i^{(eq)} \, | i = 0, 1, ..., 8|\}$ para los momentos obtenidos de la transformación lineal de la función de distribución f_i son como se muestra a continuación.

$$\mathbf{m}^{(eq)} = \begin{pmatrix} \rho \\ -2\rho + 3\rho_0 \left|\mathbf{u}\right|^2 / \varphi \\ \rho - 3\rho_0 \left|\mathbf{u}\right|^2 / \varphi \\ \rho_0 u_x \\ -\rho_0 u_x \\ \rho_0 u_y \\ -\rho_0 u_y \\ \rho_0 (u_x^2 - u_y^2)/\varphi \\ \rho_0 u_x u_y / \varphi \end{pmatrix}.$$

Es importante recordar que para un estencil D_2Q_9 tienen las direcciones mostradas



(a) Propagación en t (b) Propagación en $t+1$

Figura 2.2: Paso de propagación del MLB-TRM

en la ecuación 2.16:

$$\mathbf{e}_i = \begin{cases} (0,0) & i = 0 \\ (cos[(i-1)\pi/2], sin[(i-1)\pi/2])\, c & i = 1,2,3,4 \\ (cos[(2i-9)\pi/4], sin[(i-1)\pi/4])\sqrt{2}c & i = 5,6,7,8 \end{cases} \tag{2.16}$$

sea $c = \frac{\delta_X}{\delta_t}$ y $\omega_0 = 4/9$, $\omega_{1,2,3,4} = 1/9$, $\omega_{5,6,7,8} = 1/36$.

La velocidad $\mathbf{u}$ y la densidad de fluido ρ se calculan de la siguiente manera:

$$\rho_0 \mathbf{u} = \sum_0^8 \mathbf{e} f_i + \frac{\delta_t}{2} \rho_0 \mathbf{F}. \tag{2.17}$$

$$\rho = \sum_{i=0}^{8} f_i \tag{2.18}$$

2.2.2. Transferencia de Calor

Para modelar el campo de temperatura, el MLB-TRM utiliza un término fuente no lineal y la ecuación se escribe de la siguiente manera [8]:

$$\begin{gathered} \mathbf{g}\left(\mathbf{x} + \mathbf{e}\delta_t, \mathbf{t} + \delta_t\right) - \mathbf{g}\left(\mathbf{x}, \mathbf{t}\right) = \\ -\mathbf{N^{-1}\Theta}\left[\mathbf{n}(\mathbf{x}, t) - \mathbf{n^{(eq)}}(\mathbf{x}, t)\right] + \mathbf{N^{-1}}\delta_t \tilde{\mathbf{S}} \end{gathered} \tag{2.19}$$

Para modelar el campo de temperatura, en esta tesis se utiliza un estencil D_2Q_5

$$e_i = \begin{cases} (0,0) & i = (0,0) \\ (cos[(i-1)\pi/2], sin[(i-1)\pi/2])\, c & i = 1,2,3,4 \end{cases} \tag{2.20}$$

En la ecuación (2.19), $\mathbf{N}$ es la matriz de transformación ortogonal de tamaño 5×5, $\mathbf{\Theta}$ es la matriz de colisión, los vectores columna $\mathbf{n}(\mathbf{x},t)$, $\mathbf{g}\,(\mathbf{x},t)$, $\tilde{\mathbf{S}}$, y $\mathbf{n}^{(eq)}(\mathbf{x},t)$ son de dimensión 5. $\mathbf{g}\,(\mathbf{x},t)$ Es una función de distribución que guarda una similitud con la ecuación (2.9).

La transformación hacia el espacio de los momentos se hace de la siguiente manera:

$$\mathbf{n} = \mathbf{N}\mathbf{g}$$

Donde $\mathbf{N}$ tiene la siguiente forma:

$$\mathbf{N} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ -4 & 1 & 1 & 1 & 1 \\ 0 & 1 & -1 & 1 & -1 \end{pmatrix}. \tag{2.21}$$

La matriz de relajación para el campo de temperatura $\mathbf{g_i}(\mathbf{x},t)$ como se muestra en el artículo [8] es de la siguiente manera

$$\Theta = diag(1, 1/\tau_T, 1/\tau_T, 1{,}5, 1{,}5). \tag{2.22}$$

Dónde $\tau_T = 0{,}5 + \frac{\lambda\, c_s^2 (\tau_v - 0{,}5)}{J\,\sigma\, c_{st}^2\, Pr}$ y c_s es la velocidad del sonido al utilizar D_2Q_9 mientras que c_{st} es la velocidad del sonido en D_2Q_5.

De manera análoga a la ecuación 2.19 los componentes de $\tilde{\mathbf{S}}$ en el espacio de los momentos de $\mathbf{g}_i(\mathbf{x},t)$ están dado por los siguientes términos:

$$\tilde{S}_0 = -\frac{\phi La}{\sigma C_{pl}} \frac{\Delta f_l}{\delta_t}, \quad \tilde{S}_1 = 0, \quad \tilde{S}_2 = 0, \quad \tilde{S}_3 = -\overline{\omega} \frac{\phi La}{\sigma C_{pl}} \frac{\Delta f_l}{\delta_t}, \quad \tilde{S}_4 = 0.$$

El paso de colisión es llevado a cabo en el espacio de los momentos como se muestra en la siguiente ecuación 2.23:

$$\mathbf{n}^{+}(\mathbf{x},t) = \mathbf{n}(\mathbf{x},t) - \mathbf{\Theta}\left[\mathbf{n}(\mathbf{x},t) - \mathbf{n}^{(eq)}(\mathbf{x},t)\right] + \delta_t \tilde{\mathbf{S}}, \tag{2.23}$$

y el paso de propagación se lleva a cabo en el espacio de las velocidades, como se muestra en la ecuación 2.24:

$$\mathbf{g_i}\left(\mathbf{x} + \mathbf{e_i}\delta_\mathbf{t}, \mathbf{t} + \delta_\mathbf{t}\right) = \mathbf{g_i^{+}}\left(\mathbf{x}, \mathbf{t}\right), \tag{2.24}$$

recordando que la transformación lineal se lleva a cabo de la siguiente manera:

$$\mathbf{g}^{+} = \mathbf{N}^{-\mathbf{1}}\mathbf{n}^{+}$$

los componentes de $g_i^{(eq)}(\mathbf{X}, t)$ para la función de distribución son calculadas de la siguiente manera:

$$g_i^{(eq)} = \tilde{\omega}_i T \left(1 + \frac{\mathbf{e_i} \cdot \mathbf{u}}{\sigma c_{sT}^2}\right), \tag{2.25}$$

donde los coeficientes del peso $\{\tilde{\omega}_i | i = 0, 1, 2, 3, 4\}$ están dados por:

$$\tilde{\omega}_0 = (1 - \overline{\omega})/5$$

y

$$\tilde{\omega}_i = (4 + \overline{\omega})/20 \ (i = 1, 2, 3, 4)$$

dado que $\overline{\omega} = -2$

y los componentes de los momentos $\{n_i^{(eq)} | i = 0 - 4|\}$ al equilibrio, que son obtenidos de la transformación lineal de g_i, tienen la siguiente forma:

$$\mathbf{n}^{(eq)} = \begin{pmatrix} T \\ u_x T/\sigma \\ u_y T/\sigma \\ \tilde{\omega} T \\ 0 \end{pmatrix} \tag{2.26}$$

y la temperatura se calcula en cada celda de la siguiente manera:

$$T = \sum_{i=0}^{4} g_i. \tag{2.27}$$

donde i corresponde a cada dirección.

2.2.3. Transición del Ga de fase sólida a líquida

Para determinar el avance de la interfaz sólido-líquido del Ga inmerso en un medio poroso, se utiliza el método basado en la entalpía descrito por Zhaoli y Zhao [19]. Durante el cambio de fase del MCF se experimentan los fenómenos de transferencia de calor y transferencia de momento.

Se calcula la entalpía H_k en cada celda del dominio, determinando si el Ga en la celda correspondiente se encuentra en estado sólido, líquido o en transición.

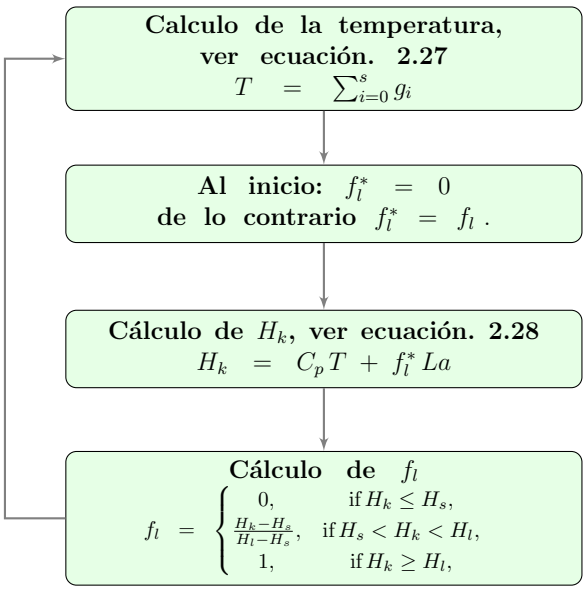


Figura 2.3: Algoritmo para calcular la fracción de líquido f_l.

Entiéndase la transición como la coexistencia de las fases sólido y líquido del MCF dentro de un mismo poro. H_k se calcula de la manera cómo está propuesto en [19].

$$H_k = C_p\, T + f_l\, La, \tag{2.28}$$

Luego de conocer la entalpía de la celda en el dominio, entonces se calcula la fracción de líquido de la siguiente manera:

$$f_l = \begin{cases} 0, & \text{if } H_k \leq H_s, \\ \frac{H_k - H_s}{H_l - H_s}, & \text{if } H_s < H_k < H_l, \\ 1, & \text{if } H_k \geq H_l, \end{cases} \tag{2.29}$$

dónde f_l es la fracción de líquido, H_s es la entalpía de la fase sólida, H_l es la entalpía de la fase líquida. Para explicar de mejor manera cómo se calcula la fracción de líquido, ver el diagrama 2.3

En la figura 2.3 se muestra el algoritmo para calcular la fracción de líquido. En cada iteración se calcula la temperatura, la entalpía y la fracción de líquido f_l de la siguiente manera: primero se calcula la temperatura de la celda, si el proceso apenas comienza, la fracción de líquido es igual a cero, de lo contrario la fracción de líquido se toma de la iteración anterior como f_l^* y luego se calcula la entalpía con la ecuación (2.28), posteriormente se recalcula la fracción de líquido con la ecuación (2.29).

Dado que el Ga está cambiando de fase sólida a líquida y también que hay gradientes de temperatura y presión, se forman vórtices. Para medir los componentes de la velocidad en cada celda, se hace de la siguiente manera:

$$\mathbf{u}_x = \frac{\mathbf{v}_x}{l_0 + \sqrt{l_0^2 + l_1 |\mathbf{v}|}} \tag{2.30}$$

$$\mathbf{u}_y = \frac{\mathbf{v}_y}{l_0 + \sqrt{l_0^2 + l_1 |\mathbf{v}|}} \tag{2.31}$$

donde

$$l_0 = \frac{1}{2}\left(1 + \varphi\frac{\delta_t}{2}\frac{\nu_l}{K}\right) \tag{2.32}$$

$$l_1 = \varphi\frac{\delta_t}{2}\frac{C_F}{\sqrt{K}}. \tag{2.33}$$

y $\nu_l = Pr * \alpha_e$ es la viscosidad de líquido.

Durante la simulación se toman en cuenta fenómenos de transferencia de momento y energía que suceden en el interior del medio poroso, luego para estimar la velocidad se utiliza la ley de Forchheimer [20], que ha sido ampliamente utilizada por diversos autores como Schmidt, Barree y Conway [21, 22]. Se utiliza el coeficiente de Ergun en la ley de Forchheimer, de la siguiente manera:

$$\beta = \frac{C_E}{\sqrt{(K)}} \tag{2.34}$$

Aquí la magnitud de C_E se encuentra fuertemente influenciada por el régimen en el que se encuentra el fluido, para bajas velocidades se considera nulo, y entonces se recupera la ley de Darcy. C_E y la permeabilidad k son obtenidos experimentalmente, pero en éste caso se asume que en todo el medio poroso está formado por esferas empacadas por lo que K se puede estimar a partir de la ecuación de Kozeny-Carman:

$$K = \frac{\phi^3 d_m^2}{175\,(1-\phi)^2}, \tag{2.35}$$

donde el promedio del diámetro de las partículas es de $d_m = 25{,}2111$, por tanto la permeabilidad sólo queda en función de la porosidad, que aquí se considera variable en el medio poroso.

Para comenzar la simulación, se calcula la porosidad a partir de una imagen del medio poroso, llevando a cabo un tratamiento digital. Posteriormente se calcula la

permeabilidad con la ley de Kozeny-Carman, esto se explica a detalle en la sección 2.3.

2.3. Tratamiento digital de las imágenes del medio poroso

En este trabajo se calcula la porosidad y la permeabilidad a partir de la imagen de un medio poroso, la cual consiste en una vista transversal del medio, como puede observarse en la figura 2.5. La imagen del medio poroso se encuentra en una escala de grises (0-255) en un arreglo matricial. Posteriormente se busca determinar los píxeles que correspondan a un sólido y los que corresponden a un poro. Para éste tratamiento se siguió el algoritmo mostrado en la figura 2.4

A partir de una imagen digital, se calcula la porosidad y permeabilidad del medio poroso siguiendo el siguiente procedimiento:

1. En primer lugar se importa una imagen digital de un medio poroso en una escala de grises, esta imagen se encuentra en forma de matriz ($Im[x, y]$), cuyos valores se encuentran en un rango de 0 a 255.

2. Se define un umbral υ para convertir los valores de la matriz tal que si $Im[x, y] \leq \iota$ se trata de un elemento 'poro' y se reemplaza dicho elemento de la matriz por 0, de lo contrario se reemplaza por 1 y corresponde a un 'sólido'. Entonces la matriz ahora es binaria (0,1).

3. Luego a partir de la matriz binaria ($Im[x, y]$) se calcula la matriz de porosidades $\phi[X, Y]$ y permeabilidades $K[X, Y]$ cuya dimensión es por ejemplo 4×4. Con matriz gruesa se hace referencia a la matriz que se obtiene del cálculo la porosidad sobre un conjunto de celdas de ($Im[x, y]$) cuyo resultado se guarda en una celda de $\phi[X, Y]$.

4. En cada elemento de la matriz de porosidades $\phi[X, Y]$ se calcula la un elemento de la matriz de permeabilidades $K[X, Y]$ de forma que $\phi[X, Y]$ y $K[X, Y]$ tienen la misma dimensión.

Por ejemplo la matriz $\phi(X, Y)$ y $K(X, Y)$ que resultan del algoritmo mostrado en la figura 2.4 tendrá cualquiera de las formas mostradas en la figura 2.6 dependiendo del número de particiones que ésta tenga.

Para el tratamiento de la imagen del medio poroso se introduce una matriz de porosidades y otra de permeabilidades, en este punto es bueno recordar el algoritmo mostrado en las figuras 2.4 donde se construyen las matrices de porosidades y

permeabilidades, luego dichas matrices son requeridas en el algoritmo mostrado en la figura 3.2.

La función de porosidad $\phi(X, Y)$ se calcula a partir de la imagen y la permeabilidad $K(X, Y)$ de la ley de Kozeny aplicada dentro de cada partición de imagen como se describe en el diagrama 2.4. Se realizaron experimentos numéricos para los problemas donde la imagen completa (Figura2.5) del medio poroso se divide en 4×4, 8×8, 16×16 y 32×32 particiones. La porosidad y la permeabilidad se calculan en cada subdivisión, Ver la Figura 2.6. Ciertamente, cuantas más particiones se se hagan de la imagen, mejor representación de la porosidad $\phi(X, Y)$, y permeabilidad $K(X, Y)$ se obtendrá. Luego se guarda la salida como una matriz aproximada que se introduce en el algoritmo principal como se muestra en la Figura 3.2.

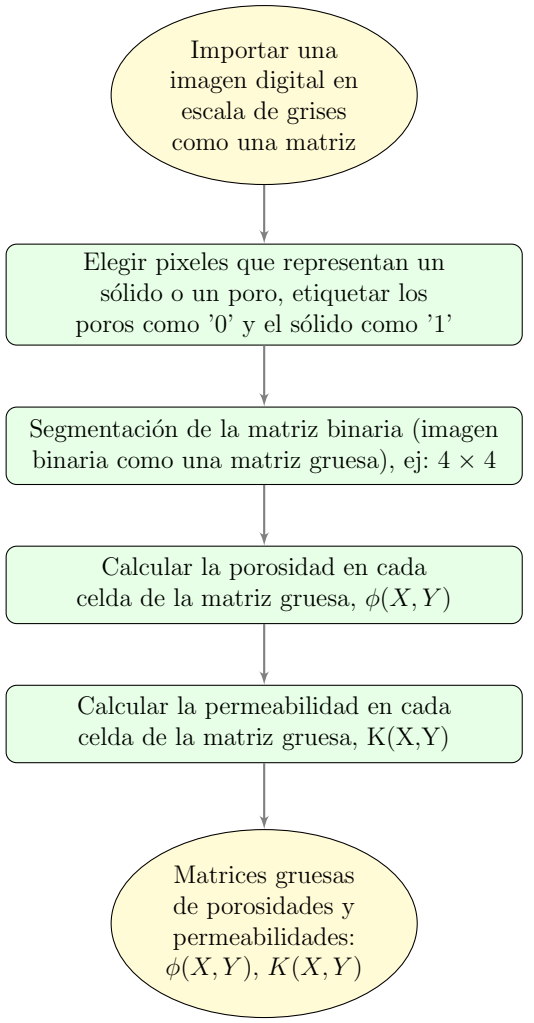


Figura 2.4: Algoritmo para el cálculo de las matrices de porosidades y permeabilidades.

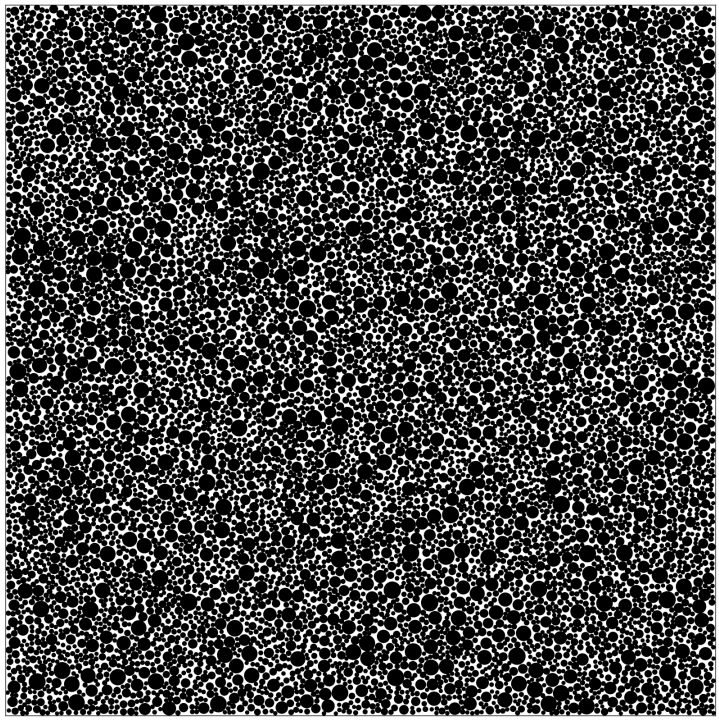


Figura 2.5: Imagen artificial de un medio poroso

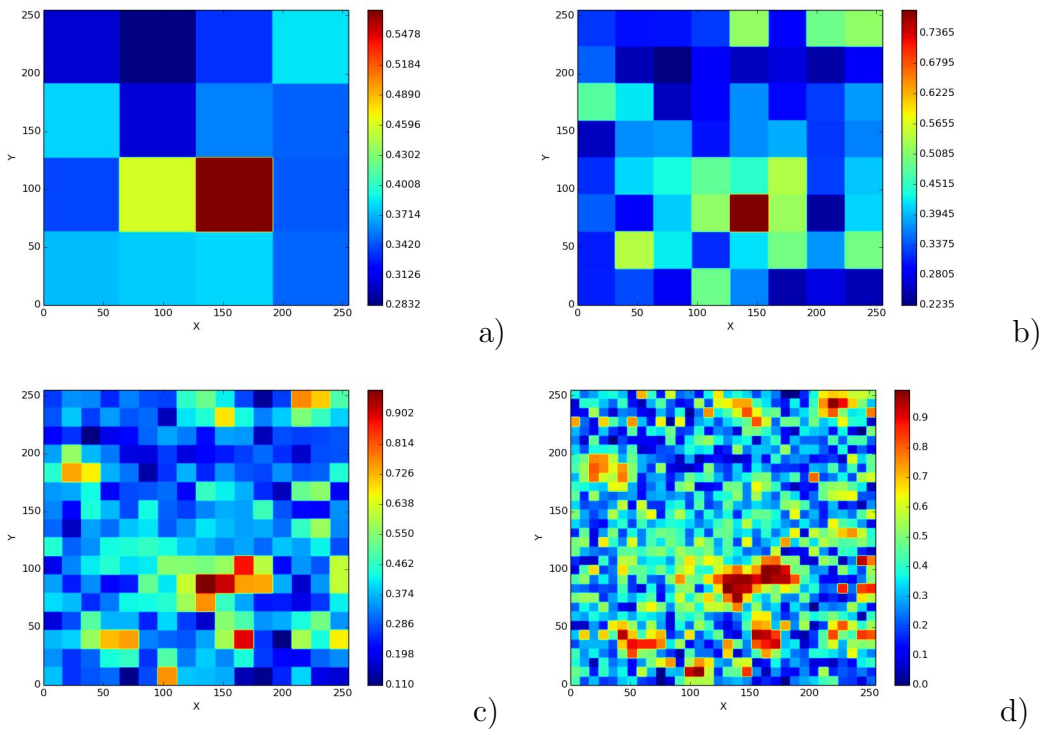


Figura 2.6: Porosidades calculadas con diferentes números de particiones a) 4×4, b) 8×8, c) 16×16 y d) 32×32 particiones.

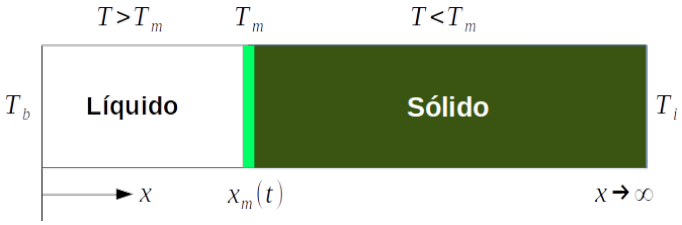


Figura 2.7: Problema 1

2.4. Solución de los problemas

En éste trabajo de tesis se da solución al problema de cambio de fase con convección natural influenciada por la gravedad, de un material inmerso en un medio poroso no Darciano (en éste caso es el Galio). Se considera que el medio poroso está sujeto a condiciones de frontera y además que la porosidad es variable en función del espacio. El problema se resuelve utilizando el método de lattice Boltzmann con tiempos múltiples de relajación (MLB-TRM) implementado en una arquitectura de cómputo paralelo con python-Numba-CUDA. En la solución de éste problema intervienen la transferencia de momento y de energía, por lo que primero se procede a resolver el problema de la transferencia de energía, el cual le llamaré 'problema 1' y posteriormente se resuelve el problema completo que se describió previamente, al cual llamaré 'problema 2'.

2.4.1. Problema 1

Consiste en simular el cambio de fase de una barra sólida semi infinita, como se muestra en la figura 2.7. Al comienzo la barra es completamente sólida a una temperatura T_i tal que $T_i < T_m$, donde T_m es la temperatura de la transición de sólido a líquido del material. En el tiempo $t = 0$ toda la barra se encuentra a una temperatura fija T_b tal que $T_b < T_m$ que se mantiene constante en la frontera izquierda ($x = 0$). En éste problema no se conoce la temperatura del líquido y del sólido cuando $t > 0$, por lo que se trata de un problema de dos regiones pero tiene una solución analítica en el libro de transferencia de calor de Hahn y Ã–Zisik [23]. Donde T_i es la temperatura de la frontera derecha, T_b es la temperatura de la frontera izquierda, las fronteras de arriba y abajo son adiabáticas. Aun cuando no se trata de un problema de medios porosos, la solución de éste problema permite validar el uso del método de lattice Boltzmann para modelar la transferencia de calor y también la implementación paralela en python-CUDA mediante la solución analítica, que también está implementada.

Éste problema se implementó de dos maneras, que se describen a continuación. Los códigos se pueden descargar libremente desde github (`https://github.com/BenjaminNoyola/Melting_Ga`). Ver el anexo B y C para descargar y ejecutar los códigos desde dockerhub y github respectivamente.

1. La primera implementación se lleva a cabo en serie utilizando el lenguaje python 3.8 y la biblioteca numpy junto a numba. Ésta última biblioteca permite generar código máquina muy eficiente, similar al que se obtiene con el lenguaje C.

2. La segunda implementación se hace con python junto a numba-CUDA. Ésta implementación tiene la particularidad de que los cálculos para el paso de colisión, las propiedades macroscópicas y las condiciones de frontera se llevan a cabo en la memoria local de CUDA, mientras que el paso de propagación se lleva a cabo en la memoria global, debido al alto rendimiento obtenido en ésta implementación, ésta técnica también se utilizó en el problema 2.

3. Se implementó la solución analítica, también utilizando el lenguaje python junto a numpy y scipy. Se importa un archivo de texto, que contiene la solución numérica para comparar con la solución analítica obtenida y mide el error porcentual medio cuadrático.

2.4.2. Problema 2

Aquí se implementa la simulación del cambio de fase sólido-líquido del MCF inmerso en un medio poroso. En la figura 2.8 se muestra un esquema del comportamiento del cambio de fase y los parámetros del problema están definidos en la tabla 2.1. Al comienzo de la simulación cuando $t = 0$ el MCF se encuentra en estado sólido pero la matriz porosa se encuentra influenciada por la frontera isotérmica T_h en el lado oeste, por lo que la temperatura comienza a incrementarse hasta alcanzar y superar la temperatura de fusión. Se cumple que $T_h > T_m$ y $T_C < T_m$ donde T_m es la temperatura de fusión y T_C es la temperatura fría de la frontera isotérmica en el lado este. Las fronteras norte y sur son adiabáticas como se muestra en la figura 2.8. Las simulaciones utilizan el MLB-TRM con un estencil D_2Q_5 para modelar la transferencia de calor y D_2Q_9 para modelar la transferencia de momento, la transferencia de momento toma en cuenta la convección natural inducida por la gravedad. Se utiliza el lenguaje python junto a otras bibliotecas que permiten generar cómputo de alto rendimiento. Aquí se implementan 3 casos, los cuales son los siguientes:

- Simulación utilizando una porosidad homogénea de $\phi = 0{,}368$

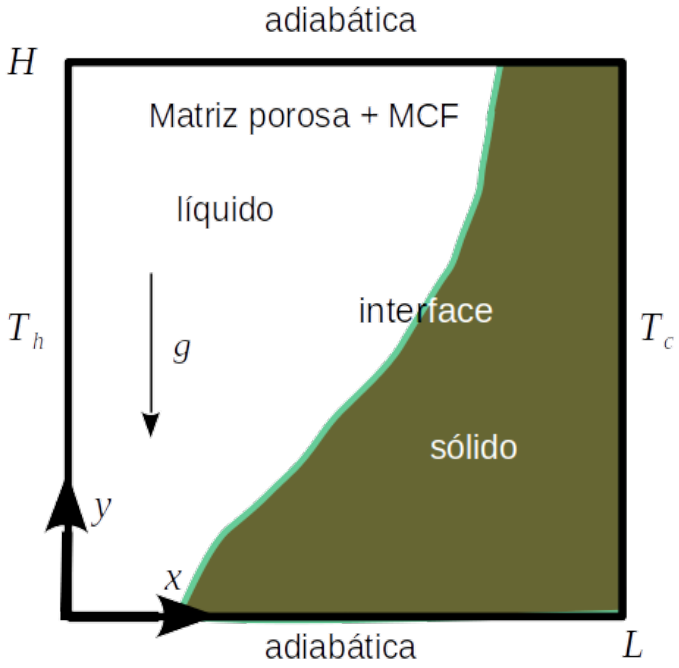


Figura 2.8: Problema 2

- Simulación utilizando una porosidad calculada a partir de una imagen, la cual no es homogénea. La imagen es procesada como se explicó en la sección 2.3. Para calcular la porosidad por regiones más pequeñas se divide la imagen en sub-regiones a las que se le calcula la porosidad, y luego cada valor de la porosidad es inyectado como valor de entrada para iniciar la simulación, en éste proceso se utiliza la ley de Kozeny. De modo que se puede considerar que la porosidad es considerada heterogénea de manera global y homogénea de manera local.

- Adicional al punto anterior, si incluyen fuentes de energía en el medio y distribuidos aleatoriamente, se analiza el comportamiento de las velocidades que forman los vórtices.

2.4.3. Validación

En primer lugar se valida la solución del problema 1 con un margen de error aceptable, éste problema tiene solución analítica descrita en el libro de transferencia

Parámetro	valor	Parámetro	valor
$lattice_x$	256	F_0	1,829
$lattice_y$	256	S_t	0,1241
Ra	$8{,}409e^5$	Ma	0,1
Da	$1{,}37e^{-5}$	σ_l	0,8604
J	1,0	σ_s	0,8352
P_r	0,0208	ρ_0	1,0
λ	0,2719	$\overline{w}$	-2
T_h	45	$\delta x = \delta y = \delta t$	1,0
T_c	20	c	$\frac{\delta_x}{\delta t}$
$Cpl = Cps$	1,0	c_{st}	$\sqrt{1/5}$
T_m	29,78	c_s	$\frac{c}{\sqrt{3}}$
T_0	20,0	$\phi(X, Y)$	En función del espacio
N_x	256	$K(X, Y)$	En función del espacio

Tabla 2.1: Parámetros en la simulación del problema 2

de calor de Ã–zisik [23]. En cuanto al problema 2, la validación se hace únicamente cuando la porosidad es homogénea en todo el dominio, y se valida con resultados experimentales reportados por Liu et al. [8].

Problema 1

Para determinar la ubicación de la interfaz sólido-líquido en una barra de 256×8 se utiliza el MLB-TRM implementado en python-CUDA, el software se puede descargar libremente de `https://github.com/BenjaminNoyola/Melting_Ga/tree/master/Case_1`. Al principio de la simulación cuando $t = 0$, la barra se encuentra en estado sólido a una temperatura $T_i = -1$ (temperatura adimensional). Pero luego la barra se encuentra sometida a una temperatura $T_b = 1$ en la frontera izquierda por lo que la temperatura se incrementa desde la frontera izquierda y se propaga a lo largo de la barra, alcanzando la temperatura de fusión $T_b = 0$. Se forma una interfaz sólido-líquido que se está moviendo de izquierda a derecha. La solución analítica para calcular la temperatura de la barra en función del tiempo y de la posición, se encuentra en el libro de transferencia de calor de Ã–zisik [23], cuya solución está descrita en dos partes:

1. Para el líquido:

$$T(x,t) = T_b - \frac{(T_b - T_m)\, erf\left[x/\left(2\sqrt{\alpha_l t}\right)\right]}{erf(\eta)}, \quad 0 < x \leq x_m(t) \tag{2.36}$$

2. Para el sólido:

$$T(x,t) = T_i + \frac{(T_m - T_i)\ erfc\left[x/\left(2\sqrt{\alpha_s t}\right)\right]}{erfc\left(\eta\sqrt{\alpha_l/\alpha_s}\right)}, \quad x > x_m(t) \tag{2.37}$$

Dónde $x_m(t) = 2\eta\sqrt{\alpha_l t}$ es la posición de la interfaz sólido líquido en función del tiempo, *erf*(x) es la función error y *erfc*(x) es la función error complementaria. El parámetro η está dada por la siguiente ecuación trascendental:

$$\frac{e^{-\eta^2}}{erf(\eta)} + \frac{k_s}{k_l}\left(\frac{\alpha_l}{\alpha_s}\right)^{1/2}\frac{(T_i - T_m)}{(T_b - T_m)}\frac{e^{-\eta^2(\alpha_l/\alpha_s)}}{erfc\left(\eta\sqrt{\alpha_l/\alpha_s}\right)} = \frac{\eta L_a\sqrt{\pi}}{C_{pl}(T_b - T_m)}. \tag{2.38}$$

En la ecuación 2.38 se sustituyen los parámetros: α_s=0.02, T_i=-1, T_m=0, T_b=1, ϕ=1, σ=1, Cp_l=Cp_s=1, $St = \frac{C_{pl}\Delta T}{L_a}$=1 luego mediante el método de Brent se encuantra la raíz(η) que posteriormente se sustituye en las ecuaciones 2.36 para hallar la temperatura en la fase líquida y 2.37 para la fase sólida. Las simulaciones fueron calculadas con una relación de difusividad térmica de $\alpha_l/\alpha_s = 10$, pero éste parámetro puede ser cambiado en el código, ver: `https://github.com/BenjaminNoyola/Melting_Ga/tree/master/Case_1`. En cuanto a las simulaciones numéricas fueron calculadas sobre una barra con dimensiones de 256×8 y se aplicaron condiciones de frontera periódicas en la pared superior e inferior y de tipo bounce-back en las fronteras isotérmicas (izquierda y derecha). El tiempo de relajación se define por $\tau = 0{,}5 + \alpha_e/(\sigma C_{sT}^2 \delta_t)$.

Para reproducir la imagen 2.9 desde python se puede lograr descargando el código en la siguiente dirección: `https://github.com/BenjaminNoyola/Melting_Ga/tree/master/Case_1/Results_Analytic_Vs_Numeric`

Los perfiles de temperatura a diferentes pasos de tiempo, (de 1000 hasta 300,000) se pueden observar en la gráfica 2.9, dónde se compara la solución analítica respecto a la solución numérica obtenida de la simulación con el MLB-TRM y python-CUDA.

Como puede observarse en la figura 2.9 aparentemente la solución numérica se encuentra sobrepuesta a la solución analítica (visualmente sin diferencia), pero se calculó el error porcentual medio cuadrático (EPMC) para comparar entre las dos soluciones, se tomó a la solución analítica como la solución verdadera, ver la ecuación 2.39. Donde N_x es la longitud característica, X_i es el resultado de la solución numérica y An_i es la solución analítica. Cabe mencionar que la longitud característica N_x y los pasos de tiempo "t"de la simulación están relacionados con el número de Fourier.

$$EPMC = \left(\frac{\sum_{i=1}^{N_x}\frac{An_i - X_i}{An_i} \times 100}{N_x}\right)^2 \tag{2.39}$$

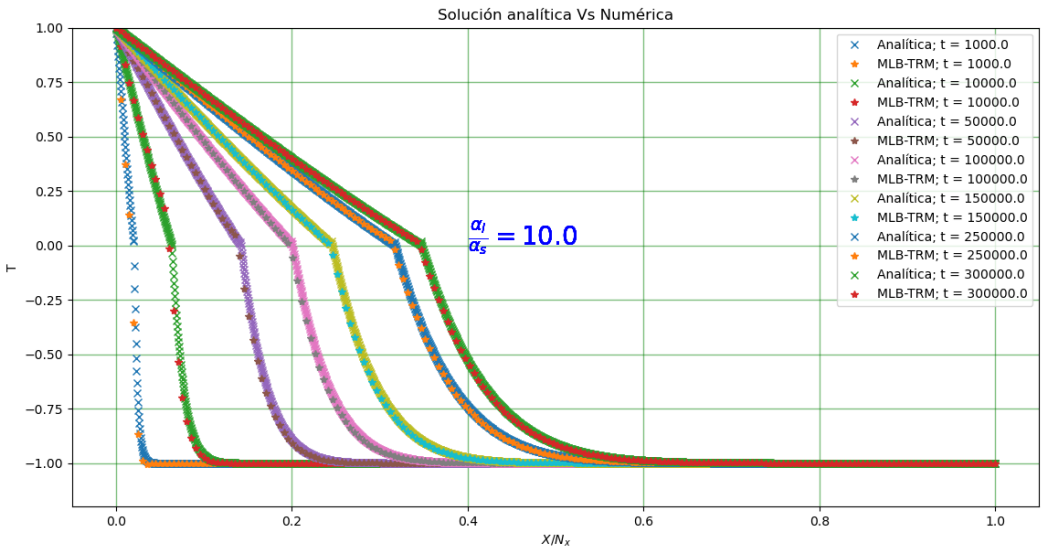


Figura 2.9: Solución numérica en CUDA (MRT LBM) Vs solución analítica para una barra que cambia de fase sólida a líquida a una relación de difusividad térmica $\frac{\alpha_l}{\alpha_s} = 10$.

Los resultados se calcularon modificando el tamaño de la barra (N_x) desde 64 a 2048 con diferentes pasos de tiempo, el error encontrado se reporta en la tabla 2.2.

Tabla 2.2: Error porcentual medio cuadrático

N_x	Pasos de tiempo	EPMC (%)
64	20,480	6.745
128	81,920	0.428
256	327,680	0.985
1024	5,242,880	0.172
2048	20,971,520	0.464

Problema 2

El problema 2 se validó con el trabajo de Liu et al. [8] a una porosidad homogénea de $\phi = 0{,}385$, se encontraron resultados similares que se pueden descargar libremente del repositorio: `https://github.com/BenjaminNoyola/Melting_Ga/tree/master/`

Tabla 2.3: NVIDIA Gforce GTX 750 Ti

CUDA Capability	5.0
Streaming Multiprocessors	5.0
CUDA core per multiprocessor	128
Global memory	2 Gb
Maximum number of resident threads per multiprocessor	2048
Maximum number of threads per block	1024
Warp size	32
Maximum number of resident blocks per multiprocessor	32
Maximum number of resident warps per multiprocessor	64

`Case_2/Poro_homogeneous`, No fue posible reproducir en su totalidad los resultados del artículo [8] ya que no se muestran de manera explicita todos los parámetros necesarios para implementar la simulación, por ejemplo α_l no está reportado, mientras que en éste trabajo se utiliza $\alpha_l = 0{,}11176$ y las dimensiones son de $[256 \times 256]$. Éstas simulaciones numéricas fueron implementadas en python-CUDA utilizando una tarjeta de video NVIDIA con las características de la tabla 2.3:

Capítulo 3

Algoritmo de Solución

3.1. Solución numérica

Los algoritmos implementados para solucionar los problemas planteados en éste trabajo de tesis, en sus diferentes matices, se encuentran todos en el siguiente repositorio de GitHub: `https://github.com/BenjaminNoyola/Melting_Ga`. Los problemas se resuelven tanto en serie como en paralelo utilizando python-CUDA como ya se dijo. En las figuras 3.1 y 3.2 se muestran los diagramas que describen los algoritmos de solución implementados para los problemas 1 y 2 y se describen a continuación:

3.1.1. Problema 1

El problema 1 se implementa tanto en serie (python-Numpy) como en paralelo (python-CUDA) y se descarga libremente desde `https://github.com/BenjaminNoyola/Melting_Ga/tree/master/Case_1`. En esta sección únicamente se describe el algoritmo paralelo, ver figura 3.1, pero el código que resuelve el problema 1 de manera serial también se incluye en este repositorio:

1. Los parámetros se especifican en el archivo `Parameters.json` que contiene todos los parámetros físicos y geométricos que requiere el problema.

2. Inicialización: en éste paso se especifica la función de distribución (g), fracción de líquido (f_l), temperatura inicial de la barra (T)

3. Ciclo: En este paso la solución se encuentra en un proceso iterativo y se detiene hasta que se alcanza el número de pasos buscados.

4. Colisión: en éste paso se resuelve la ecuación 2.23 de manera masivamente paralela, para cada celda en el dominio. Estas operaciones se resuelven en la

memoria local de la GPU (CUDA), lo que permite que sea una operación muy veloz.

5. Propagación: en éste paso se resuelve la ecuación 2.24 de manera paralela pero en la memoria compartida de la GPU (CUDA) ya que en este paso existe un intercambio de información con la celda vecina.

6. Condiciones de frontera: Este paso puede ser calculado en la memoria local o en la memoria compartida. Las condiciones de frontera de tipo periódicas se calculan en la memoria compartida, mientras que las condiciones de frontera de tipo Dirichlet y bounce-back se calculan en la memoria local ya que no requieren intercambio de información con las celdas vecinas, Ver figura 1.3.

7. Cálculo de la temperatura macroscópica: en éste paso se calcula la temperatura en cada celda, ver ecuación 2.27. Luego con ésta nueva temperatura se reinicia el ciclo o termina si es el caso.

3.1.2. Problema 2

El problema 2 se puede descargar libremente del siguiente repositorio en github: `https://github.com/BenjaminNoyola/Melting_Ga/tree/master/Case_2`, en éste repositorio hay dos soluciones, una en donde la porosidad es homogénea y se resuelve en paralelo (python-CUDA) y otra en donde la porosidad es no homogénea y se lee de una imagen a partir de un tratamiento digital. A continuación se describe el algoritmo donde la porosidad es no homogénea y además es calculada a partir de una imagen digital. La descripción está basada en los diagramas 3.2 y 2.4.

1. El algoritmo general mostrado en la figura 3.2 toma los valores de $\phi[X, Y]$ y $K[X, Y]$ provenientes del tratamiendo digital de la imagen (ver diagrama 2.4), y también lee los parámetros físicos y geométricos que se encuentran en el archivo 'Parameters.json' como entrada.

2. Inicialización: Se inicializan las funciones de distribución f, g, la velocidad u, el termino de la fuerza F, la fracción de líquido f_l y la temperatura inicial T, ver paso de 'Inicialización' en el diagrama 3.2.

3. Ciclo: En este paso la solución se encuentra en un proceso iterativo y se detiene hasta que se alcanza el número de pasos buscados.

4. Paso de colisión (fluido): Se resuelve la ecuación 2.12. Esta ecuación es masivamente paralelizable y se resuelve en la memoria local de la GPU (CUDA).

5. Paso de colisión (temperatura): Se resuelve la ecuación 2.23. Esta ecuación es masivamente paralelizable y se resuelve en la memoria local de la GPU (CUDA)

6. Paso de propagación (fluido): Se resuelve la ecuación 2.13 en paralelo pero en la memoria compartida ya que en este paso hay intercambio de información con las celdas vecinas.

7. Paso de propagación (temperatura): Se resuelve la ecuación 2.24 en paralelo pero también en la memoria compartida por el intercambio de información con las celdas vecinas.

8. Condiciones de frontera para los campos de fluido y temperatura: Para el campo de flujo se aplican condiciones de frontera de tipo bounce-back, y para el campo de temperatura se aplican condiciones de frontera adiabáticas en las paredes superior e inferior, mientras que en las paredes este y oeste se aplican condiciones de frontera de tipo dirichlet.

9. Paso del cálculo de la densidad y temperatura: Se resuelven las ecuaciones 2.18 y 2.27 respectivamente. Posteriormente se reinicia el ciclo si es el caso o termina el proceso si se completaron los pasos.

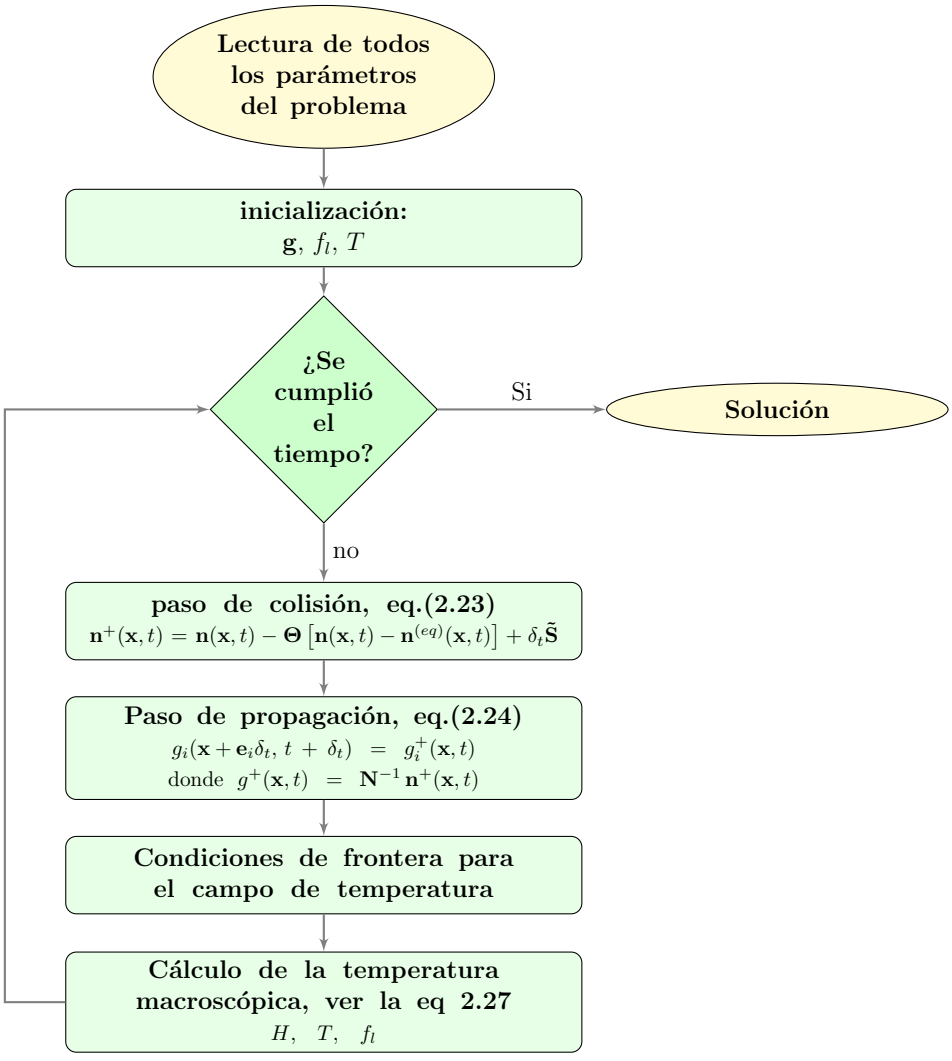


Figura 3.1: Solución paralela para el problema 1.

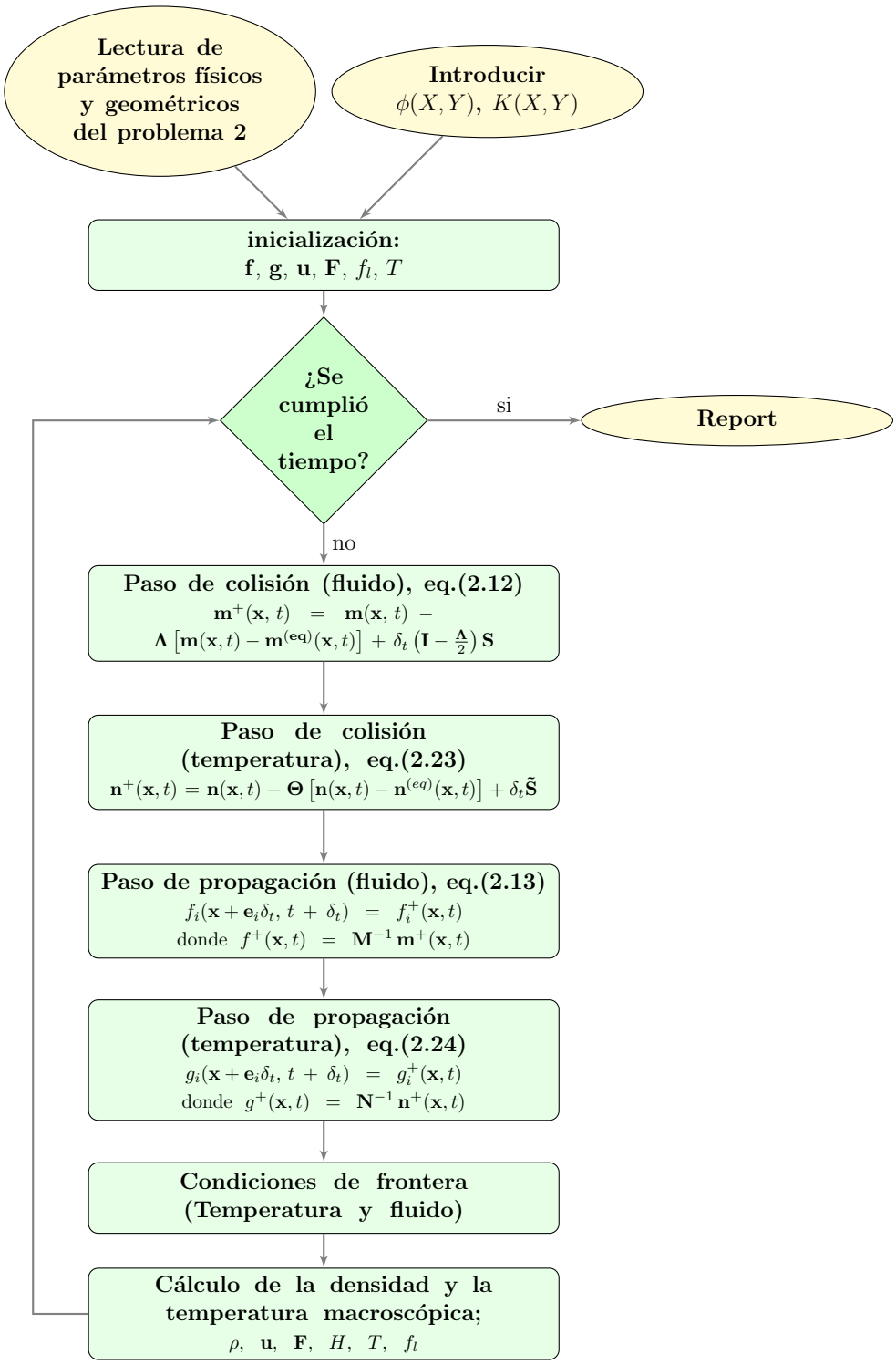


Figura 3.2: Solución paralela con el MLB-TRM aplicado al problema 2.

Capítulo 4

El Software

4.1. Arquitectura de software

El software que se desarrolló en éste trabajo se llama **Melting_Ga** y se encuentra publicado en el siguiente repositorio: `https://github.com/BenjaminNoyola/Melting_Ga` en el cual se resuelven los dos problemas previamente mencionados pero en diferentes escenarios:

- Problema 1 en versión serial (Numpy)
- Problema 1 en versión paralela (Python-CUDA)
- Problema 2 en versión paralela con porosidad homogenea
- Problema 2 en versión paralela con porosidad no homogenea
- Problema 2 en versión paralela con múltiples fuentes y sumideros.

Además en éste repositorio también se incluyen códigos para construir gráficos.

Recordando que el MLB-TRM consta básicamente de 5 pasos: 1) inicialización, 2) paso de colisión, 3) paso de propagación, 4) condiciones de frontera y 5) cálculo de variables macroscópicas, ver Figuras 3.1 y 3.2 y cada paso se puede paralelizar de forma diferente en la GPU a excepción de la inicialización que se opera en la CPU (host). Los otros pasos se operan en la GPU (device) sin salir del 'device' hasta que se completan las iteraciones, ver Figuras 3.1 y 3.2.

El espacio de memoria total en la GPU se clasifica de la siguiente manera; memoria de los registros, memoria compartida, memoria constante, memoria de textura y memoria global como se muestra en la Figura 4.1. Los registros y la memoria compartida son memorias en chip, los registros se utilizan para almacenar variables

locales y tienen la mayor velocidad. La memoria global es abundante pero tiene la velocidad de acceso más lenta, y cada hilo puede acceder a ella durante el tiempo de ejecución del kernel [24]. Las variables que se encuentran en la memoria de los registros se puede acceder a muy alta velocidad y de forma masivamente paralela porque cada hilo accede a sus propios registros [25].

Este algoritmo y en general el método de lattice Boltzmann, se adapta muy bien a la arquitectura CUDA, ya que el MLB-TRM se comporta similar a un autómata celular y permite ser masivamente paralelizable en el paso de colisión, que justamente es donde se llevan a cabo las operaciones más complejas y permite que cada núcleo de la memoria de los registros de la GPU (Ver figura 4.1) se encargue de resolver la operación para una celda en particular. En éste trabajo se resuelve la ecuación 2.23 (campo de temperatura) y 2.12 (campo de flujo) en cada celda bajo éste esquema. Es importante mencionar que la memoria de los registros es la más rápida de la GPU [25].

Las operaciones de propagación y condiciones de frontera se realizan en la memoria global porque necesitan intercambiar información entre celdas. Esta operación no es compleja porque sólo consiste en intercambiar información con las celdas vecinas, aunque la memoria global no es tan rápida como la memoria de los registros, la operación se resuelve bastante rápido por lo simple que es.

El código está diseñado en 3 capas. La primera inicializa las matrices en la CPU y transfiere la información a la GPU. La segunda capa realiza el paso de propagación en la memoria global y vincula la información con la memoria de los registros y la tercera capa es donde se realizan las operaciones más complejas que se llevan a cabo en la memoria de registros y no comparten información de otros núcleos de procesamiento. Cuando un núcleo ha completado su tarea primero, espera hasta que todos los núcleos de procesamiento estén listos para transferir toda la información al CPU (host). Ver los algoritmos de los problemas 1 y 2 en la Figura 3.1 y 3.2.

Un asunto importante a mencionar es la forma en que se realizan las operaciones matriciales en la memoria de registros (operaciones locales), por ejemplo para las operaciones del campo de flujo (ecuación 2.12), en la parte subrayada de la ecuación que se muestra a continuación:

$$\mathbf{m}^{+}(\mathbf{x},\, t) = \mathbf{m}(\mathbf{x},\, t) - \underline{\mathbf{\Lambda}\left[\mathbf{m}(\mathbf{x},t) - \mathbf{m}^{(\mathbf{eq})}(\mathbf{x},t)\right]} + \delta_t \left(\mathbf{I} - \frac{\mathbf{\Lambda}}{2}\right)\mathbf{S}.$$

Se aprovecha la forma de la matriz diagonal Λ para operarla con los vectores $\mathbf{m}$ y $\mathbf{m}^{(\mathbf{eq})}$.

Se aplica el mismo principio para operar:

$$\delta_t \left(\mathbf{I} - \frac{\mathbf{\Lambda}}{2}\right)\mathbf{S}$$

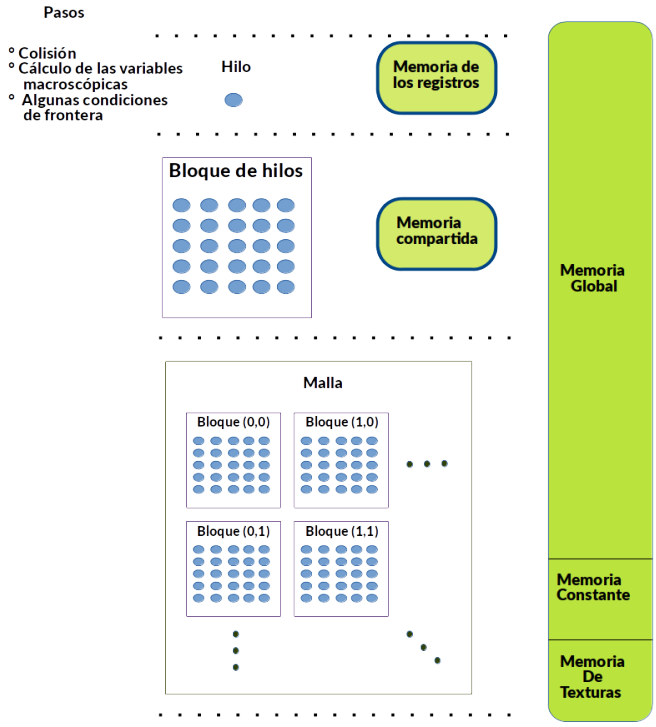


Figura 4.1: Estructura de la memoria de la GPU.

Luego para realizar la transformación lineal $\mathbf{m} = \mathbf{M}\,\mathbf{f}$, se opera en la memoria de los registros de la siguiente manera:

```
mloc[0] = floc[0] + floc[1] + floc[2] + floc[3] + floc[4] + floc[5] + floc[6] + floc[7] + floc[8]
mloc[1] = -4.0*floc[0] - floc[1] - floc[2] - floc[3] - floc[4] + 2.0*floc[5] + 2.0*floc[6]
        + 2.0*floc[7] + 2.0*floc[8]
mloc[2] = 4.0*floc[0] - 2.0*floc[1] - 2.0*floc[2] - 2.0*floc[3] - 2.0*floc[4] + floc[5] + floc[6]
        + floc[7] + floc[8]
mloc[3] = floc[1] - floc[3] + floc[5] - floc[6] - floc[7] + floc[8]
mloc[4] = -2.0*floc[1] + 2.0*floc[3] + floc[5] - floc[6] - floc[7] + floc[8]
mloc[5] = floc[2] - floc[4] + floc[5] + floc[6] - floc[7] - floc[8]
mloc[6] = -2.0*floc[2] + 2.0*floc[4] + floc[5] + floc[6] - floc[7] - floc[8]
mloc[7] = floc[1] - floc[2] + floc[3] - floc[4]
mloc[8] = floc[5] - floc[6] + floc[7] - floc[8]
```

Aquí `mloc` y `floc` son matrices de dimensión 9, ver matriz 1.12, como se observa, no se están operando los ceros de la matriz. Se usa el mismo principio para operar la ecuación 2.23.

Y luego también se usa el mismo procedimiento para operar la transformación lineal: $\mathbf{f} = \mathbf{M^{-1}}\,\mathbf{m}$ para encontrar f.

4.2. Partes Principales del Software

A continuación se describen los principales componentes del software:

1. Archivo de parámetros: 'Parameters.json' en éste código se encuentran los parámetros físicos y geométricos del problema

2. Programa principal: 'Main_change_phase_poro_image.py' este es el código principal, se encarga de inicializar arreglos numéricos que se utilizan en la simulación y coordina la información entre el CPU (host) y la GPU (device).

3. Programa que coordina la memoria global de la GPU: La información proveniente del CPU se coordina entre la memoria global y la memoria local el código se llama "Global_memory.py". Las tareas más complejas y que pueden ser masivamente paralelizables, son enviadas a la memoria de los registros, y el resto de tareas se ejecuta en la memoria global.

4. Programa que ejecuta operaciones en la memoria local: éste programa se llama 'local.py' el cual ejecuta las tareas que son masivamente paralelizables, ver el algoritmo de la figura 3.2.

Al final el código entrega archivos de texto que contienen variables como temperaturas, velocidades, fracciones de líquido, densidades, rendimiento, etc.

También se incluye un código que permite graficar los resultados de temperatura o velocidades, se llama 'Grafico_2D_STR.py'.

4.3. Rendimiento del software

Las simulaciones se llevaron a cabo en una tarjeta de vídeo NVIDIA con las características mostradas en la tabla 2.3 y las métricas utilizadas para medir el rendimiento fueron: los SpeedUP y los MLUPS (Millones de celdas actualizadas por segundo) que sus siglas en inglés significa 'Million Lattice Updates Per Second'. La formula para encontrar el SpeedUP que se implementa es la siguiente:

$$S_p = \frac{\Upsilon_{serie}}{\Upsilon_{paralelo}}$$

donde Υ_{serie} es el tiempo de ejecución en serie y $\Upsilon_{paralelo}$ es el tiempo de ejecución en paralelo.

Problema 1

En el problema 1 el rendimiento se mide en SpeedUP. Los códigos implementados en paralelo se comparan con un código implementado en serie en 'python - Numpy' y los resultados se muestran en la tabla 4.1. Todas las pruebas se ejecutaron sobre un problema con una malla de 256×8 y 3×10^6 pasos de tiempo.

Tabla 4.1: Performance of problem 1 with mesh size of 256×8, and 3×10^5 time steps

Code	time (s)	SpeedUP
Python-CUDA mejorado	69.12	236.259
Python-CUDA básico	151.312	113.85
Python-Numba	7869.9	2.19
Python-Numpy	16331.5	1

Como pudo observarse en la tabla 4.1 se logró tener un SpeedUp de hasta 236 pero estos mismos códigos podrían ejecutarse en tarjetas más modernas y con mejores prestaciones respecto a mi GPU, cuyas especificaciones se muestran en la tabla 2.3, sin embargo los códigos son modernos a la fecha de los últimos ajustes de esta tesis en diciembre del 2020.

Problema 2

En el **Problema 2** el rendimiento se mide en MLUPS, el problema se ejecutó sobre mallas de diferentes tamaños y en todos los casos 2×10^6 pasos de tiempo. Como puede observarse en la figura 4.2 se obtuvo un mejor rendimiento con una malla de 256×256 ya que se actualizan más 30 millones de celdas por segundo y la simulación completa concluyó en 2 horas y 5 minutos. Los datos a la izquierda del gráfico en azul, muestran el número de celdas actualizadas por segundo, y la escala a la derecha, en rojo, muestra el tiempo que tardó la simulación, mientras que en el eje x, con letras color negro se muestra el tamaño del problema.

Estos experimentos no fueron factibles de ejecutar en serie, ya que cada una de estas corridas puede durar más de 2 semanas corriendo y en la literatura es más común la métrica de MLUPS al operar con el MLB-TRM [**?**].

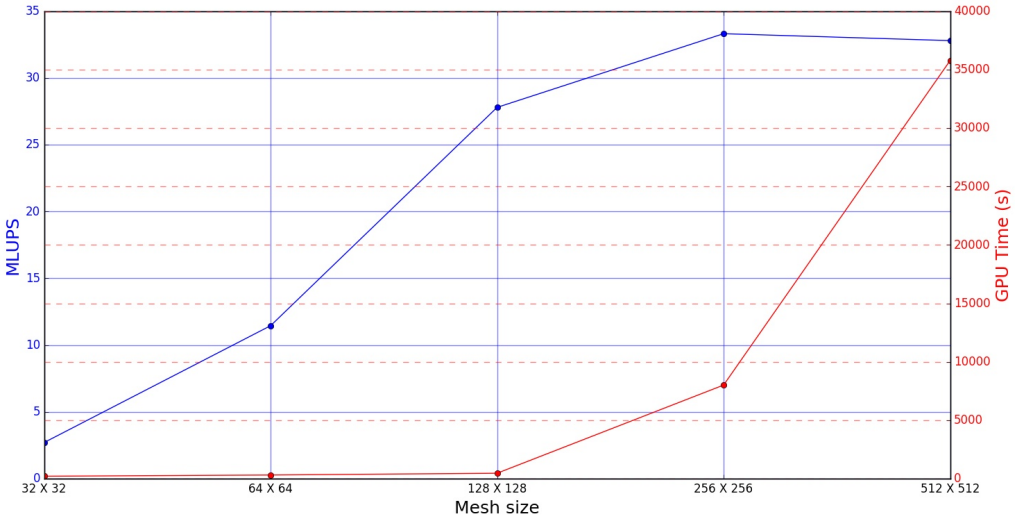


Figura 4.2: Problema 2, Pruebas de rendimiento.

4.4. Funcionalidades del software

Como se mencionó en la introducción de ésta tesis, es de gran interés estudiar el cambio de fase de materiales que se encuentran inmersos en medios porosos, en la industria hay varias aplicaciones, por ejemplo: congelamiento y derretimiento de suelos, almacenamiento de energía, etc.

El software 'Melting_Ga' que aquí se presenta es libre, se encuentra alojado en GitHub y se puede descargar en zip o clonando el repositorio. Éste software le permite a los usuarios estudiar el cambio de fase de un material inmerso en un medio poroso, el cual cuenta con un archivo donde están contenidos todos los parámetros que pueden ser modificados para adaptarlos a otras propiedades físicas del medio poroso o del MCF, como en éste caso el Galio. Por ejemplo en éste software se pueden cambiar parámetros físicos del problema, tamaño del dominio, condiciones de frontera, la imagen del medio poroso, tamaño de la matriz porosa, las técnicas de paralelización, etc.

Capítulo 5

Resultados

Los resultados aquí mostrados fueron publicados en el artículo: "Simulations of Ga melting based on multiple-relaxation time lattice Boltzmann method performed with CUDA in Python" [2]. El problema 1 se validó con la solución analítica, mientras que en el problema 2 se validó comparando con los resultados de Beckermann y Viskanta [9] a una porosidad homogénea de $\phi = 0{,}385$ para todo el medio poroso. Los resultados se muestra en las figuras 5.1 y 5.2, donde se puede observar que la porosidad del medio no tiene impacto que modifique el patrón de comportamiento de las lineas de corriente. En la naturaleza existen diversos factores que modifican el comportamiento de las líneas de corriente, como la porosidad y la permeabilidad. De manera que hacer experimentos donde la porosidad es totalmente homogénea, se encuentran distantes a lo que sucede en la realidad, ya que la porosidad y la permeabilidad cambian en el medio y modifican el rumbo de las líneas de corriente, como se observa en las figuras 5.1 y 5.2 que se encuentran sobre un dominio de 256×256 del MLB-TRM.

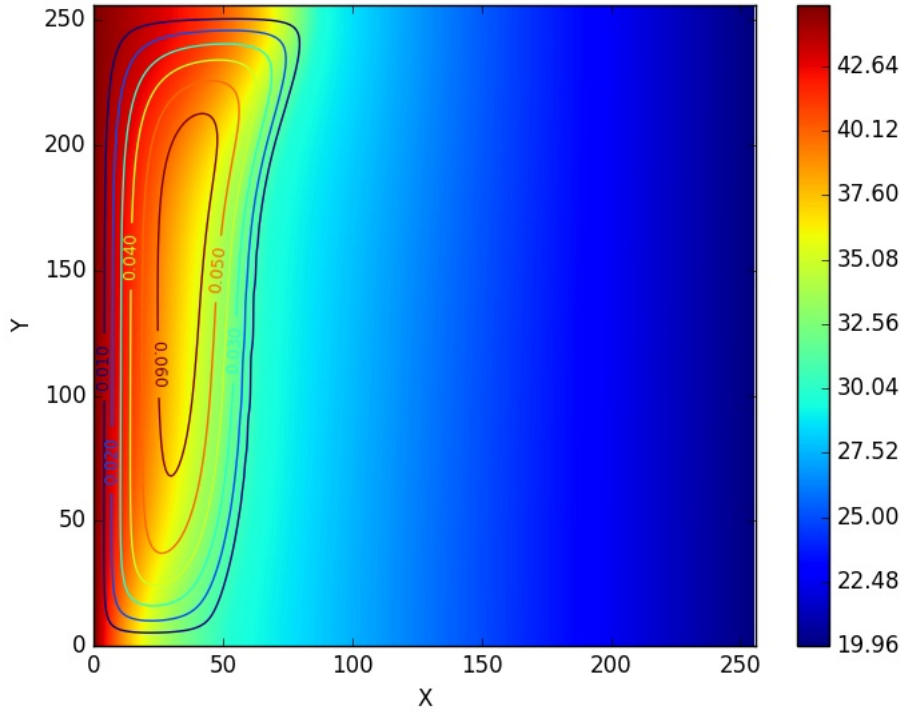


Figura 5.1: Simulación con porosidad homogénea de $\phi = 0{,}385$ y 5×10^5 pasos de tiempo, aquí la escala térmica se muestra en grados Celsius.

La dirección de una línea de corriente se da por el vector resultante de las componentes de la velocidad. Cuando el medio tiene porosidad no homogénea como en la figuras 2.6, en la simulación se obtienen líneas de corriente que no siguen un patrón bien definido con curvas suaves, como se puede observar en las figuras 5.5 y 5.6. Donde la mayor velocidad se encuentra en la región con un rojo mas intenso, mientras que cuando la velocidad tiende a 0, se puede observar en un azul más intenso. A la derecha del mapa de colores se observa una escala de velocidad adimensional, la cual se ejecutó con 5×10^5 pasos de tiempo sobre un dominio de 256×256 celdas que se utilizaron para modelar con el MLB-TRM.

Si se incrementa el tiempo de simulación a 2×10^6 pasos de tiempo, se observa que la velocidad es más grande en las regiones de mayor permeabilidad, como se muestra en la figura 5.4. Se puede apreciar que la velocidad es mayor en las regiones donde hay mayor permeabilidad, esto se puede observar en rojo intenso. Mientras que cuando la velocidad tiende a cero, el color azul se hace más intenso. Cómo se puede leer en la barra lateral.

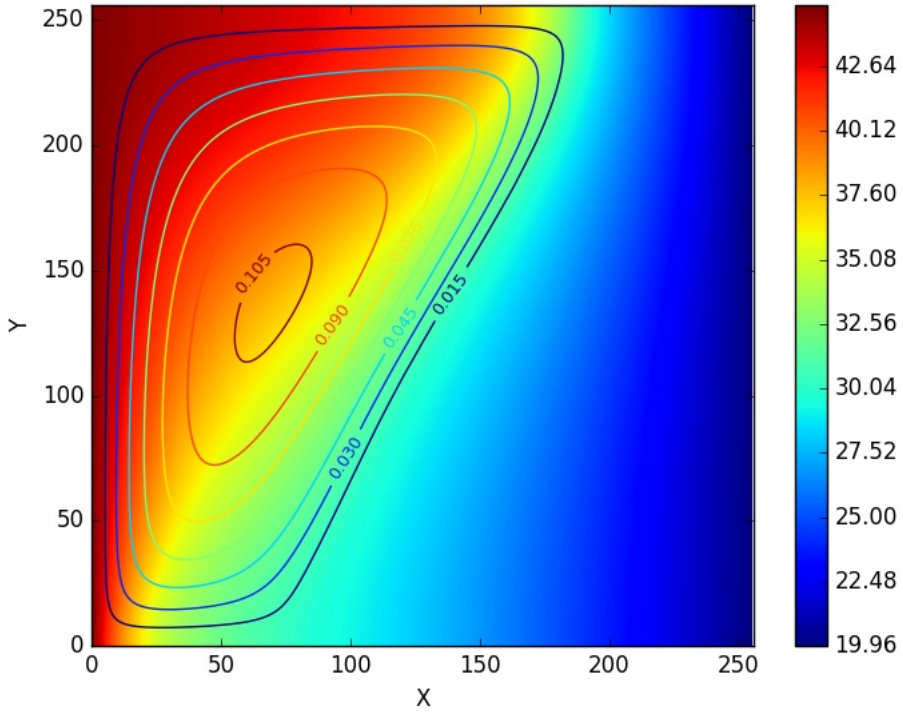


Figura 5.2: Simulación con porosidad homogénea de $\phi = 0{,}385$ y 2×10^6 pasos de tiempo, aquí la escala térmica se muestra en grados Celsius.

A continuación se repite el experimento anterior pero ahora sobre una matriz porosa de 32×32 como se observa en la figura 2.6 c) y Los resultados se muestran en las figuras 5.5 y 5.6. Se observa cómo la velocidad se concentra en las zonas de más alta permeabilidad logrando obtener velocidades más altas en estos casos. Es decir si comparamos las magnitudes de la velocidad obtenida en la figura 5.4 y 5.6 se puede observar que en la figura 5.6 la velocidad es aproximadamente hasta el doble que la obtenida en la figura 5.4 en pequeñas regiones, mientras que también se observa que en regiones de baja permeabilidad, la velocidad se ve fuertemente disminuida.

Luego se analiza el impacto de la permeabilidad sobre la temperatura en el mismo experimento. Los resultados se observan en las imágenes 5.7, 5.8, 5.9, 5.10. En los mapas térmicos de las figuras, la temperatura se encuentra en grados Celsius y se indica en la barra lateral. La frontera isotérmica izquierda se encuentra a 45 °C, las fronteras inferior y superior son adiabáticas y la frontera isotérmica derecha se encuentra a 20 °C.

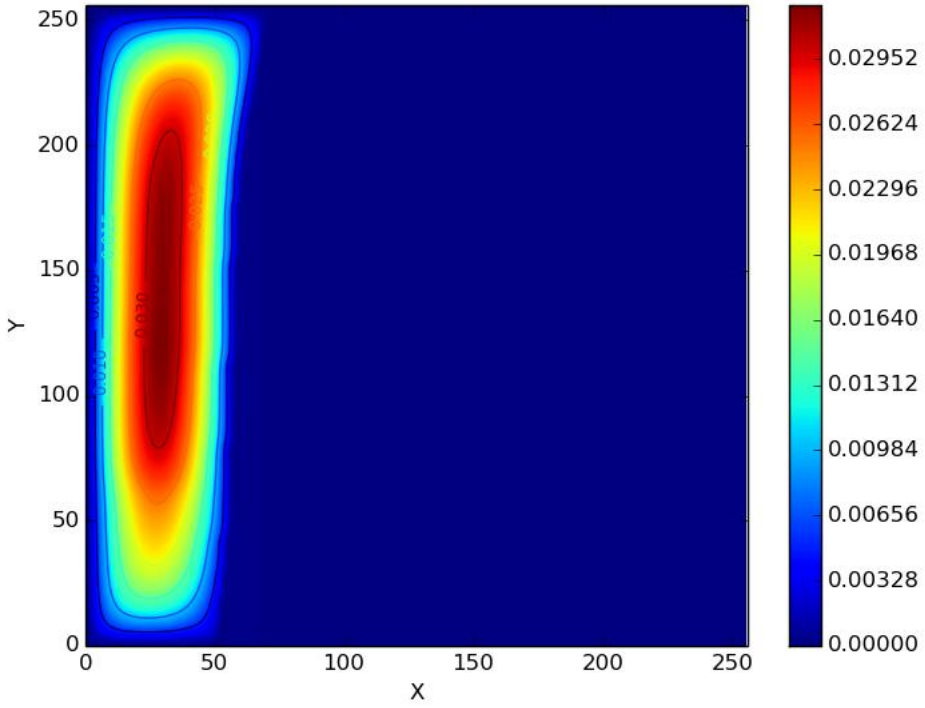


Figura 5.3: Velocidad en un medio no homogéneo con 5×10^5 pasos de tiempo, en un medio poroso partido en 4×4 y un dominio de 256×256.

El desarrollo de éste software es importante porque permite realizar estudios de cambio de fase a partir de una imagen de un medio poroso, que en éste trabajo se utilizó el Ga, pero aun así el software permite que los parámetros sean modificados para tomar las propiedades de otras sustancias, y de esta manera tener simulaciones diferentes.

En éste trabajo se desarrolló una doble multi relajación, una para modelar el campo de temperaturas con el MLB-TRM y otra para modelar el campo de momentos también utilizando el MLB-TRM.

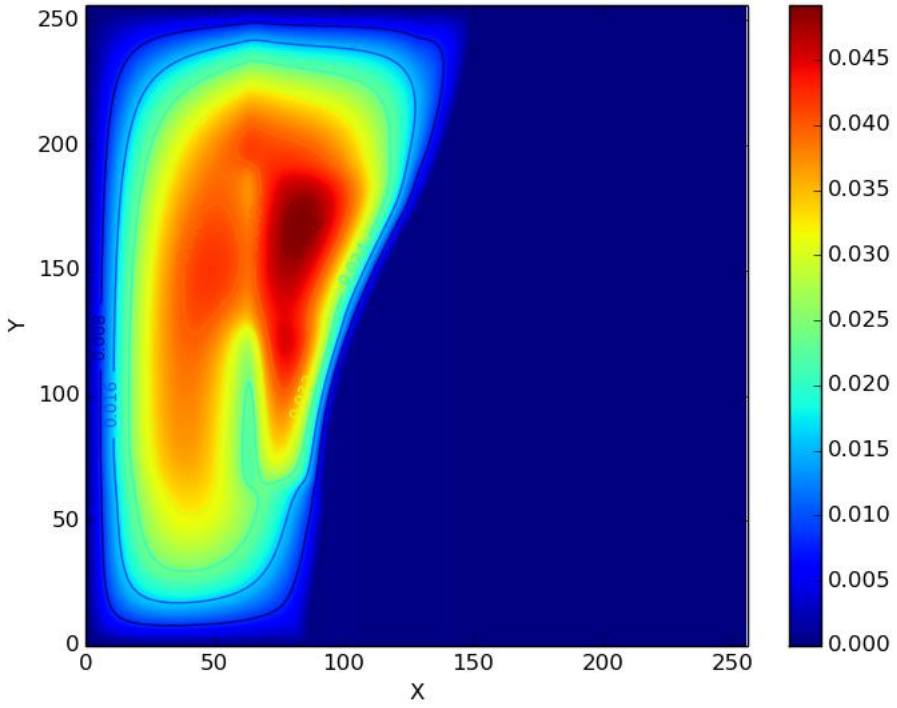


Figura 5.4: Velocidad en un medio no homogéneo con 2×10^6 pasos de tiempo, en un medio poroso partido en 4×4 y un dominio de 256×256.

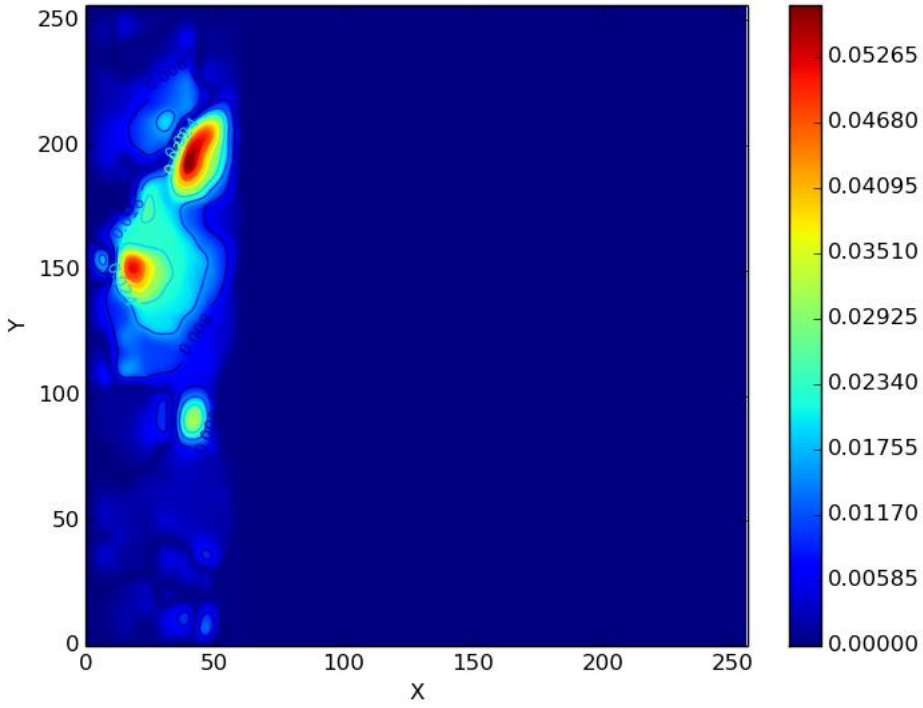


Figura 5.5: Velocidad en un medio no homogéneo con 5×10^5 pasos de tiempo, en un medio poroso partido en 32×32 y un dominio de 256×256.

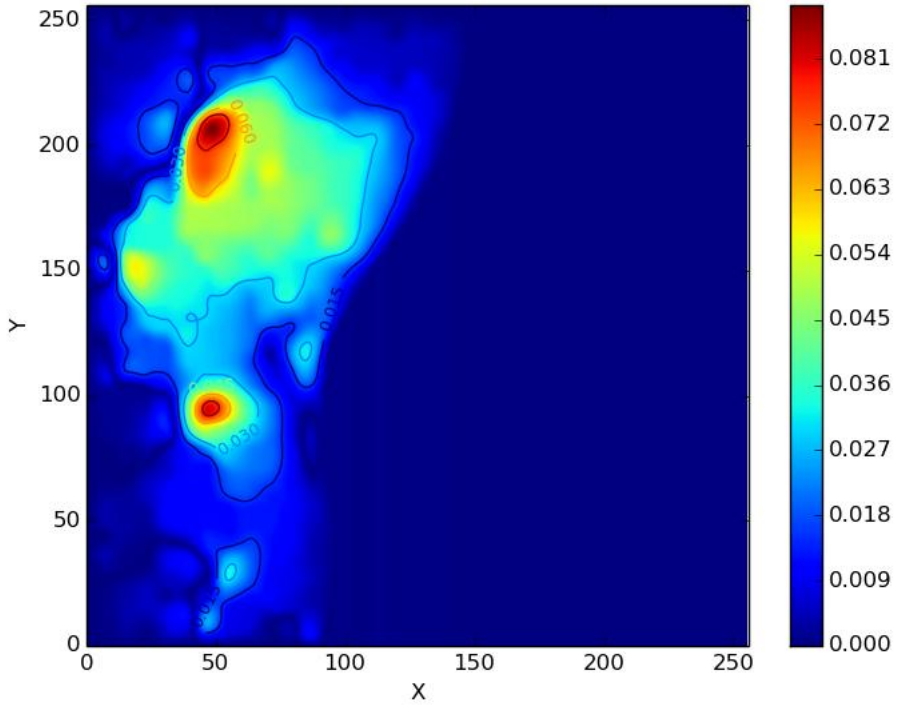


Figura 5.6: Velocidad en un medio no homogéneo con 2×10^6 pasos de tiempo, en un medio poroso partido en 32×32 y un dominio de 256×256.

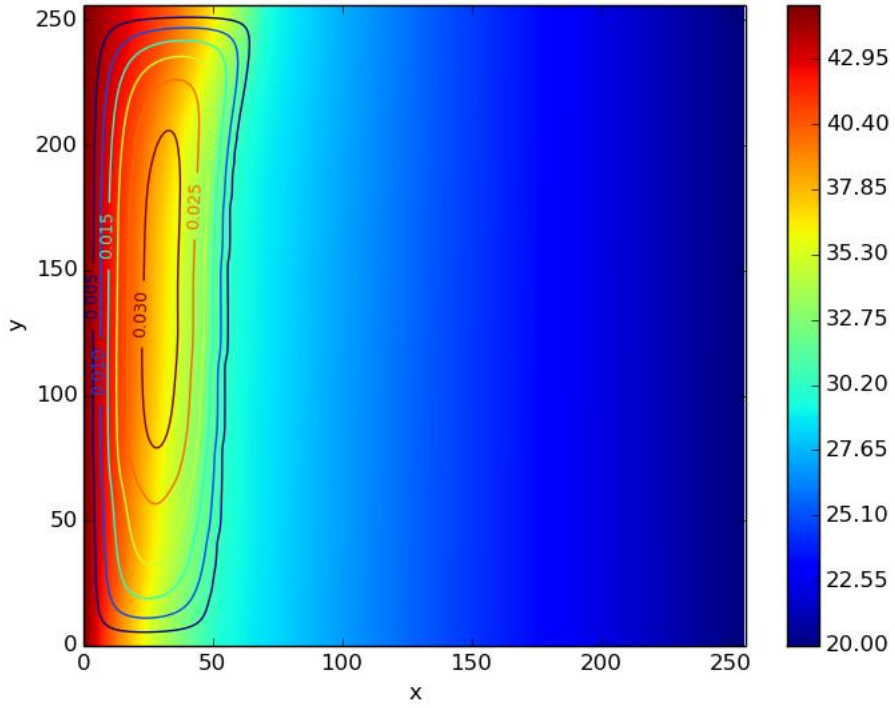


Figura 5.7: Mapa térmico (°C) de una simulación con 5×10^5 pasos de tiempo, sobre un medio poroso no homogéneo partido en 4×4 y un dominio de 256×256.

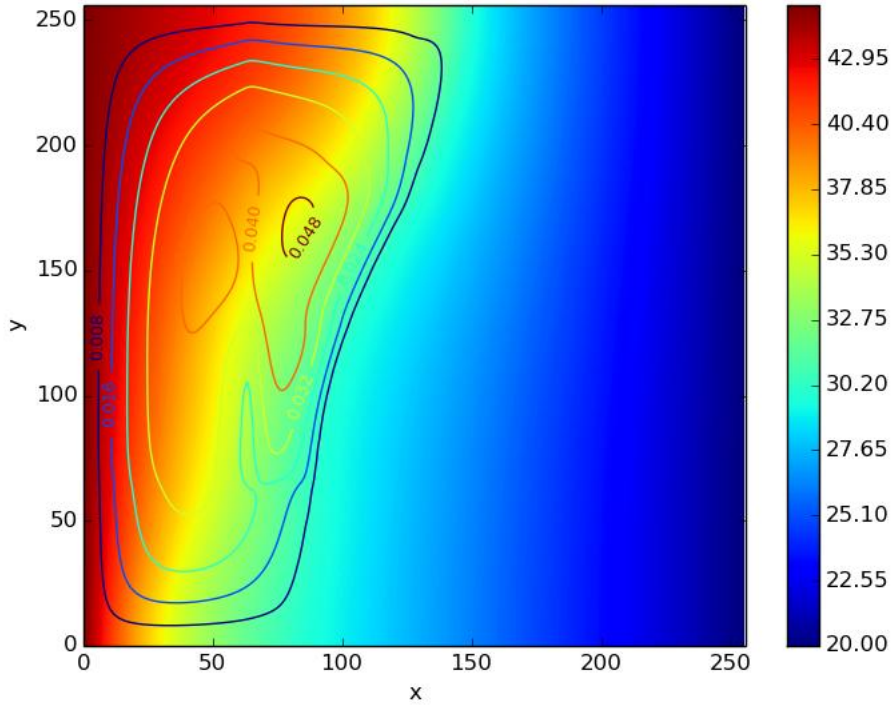


Figura 5.8: Mapa térmico (°C) de una simulación con 2×10^6 pasos de tiempo, sobre un medio poroso no homogéneo partido en 4×4 y un dominio de 256×256.

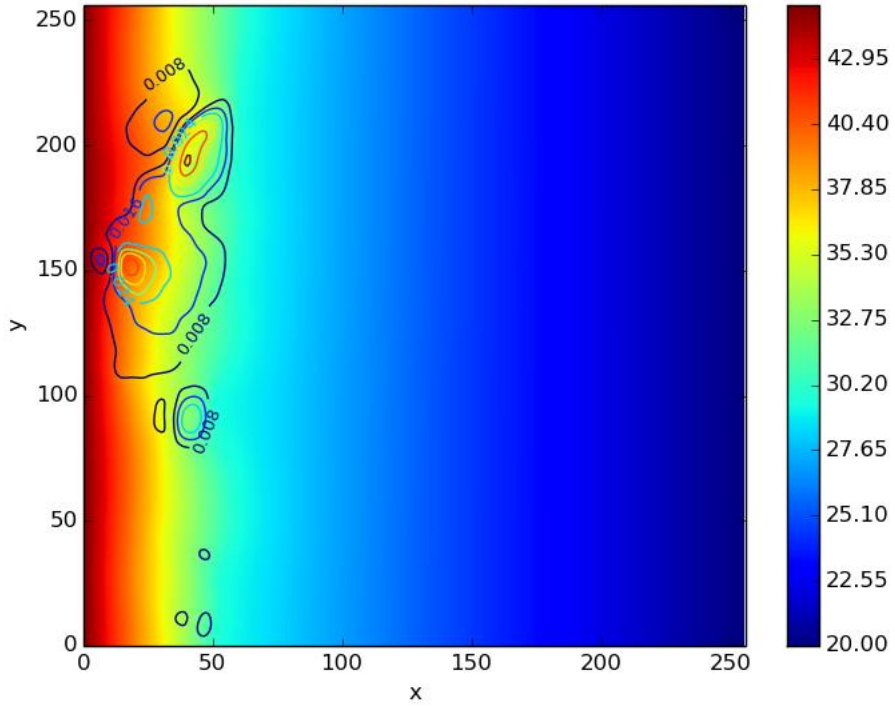


Figura 5.9: Mapa térmico (°C) de una simulación con 5×10^5 pasos de tiempo, sobre un medio poroso no homogéneo partido en 32×32 y un dominio de 256×256.

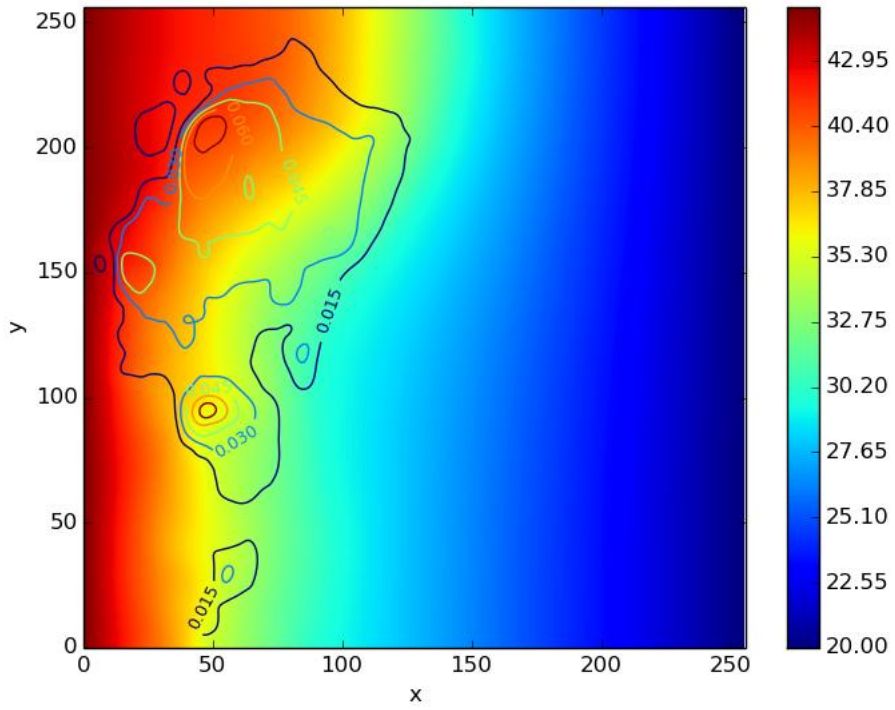


Figura 5.10: Mapa térmico (°C) de una simulación con 2×10^6 pasos de tiempo, sobre un medio poroso no homogéneo partido en 32×32 y un dominio de 256×256.

Conclusiones

- En éste trabajo se proporciona un algoritmo para generar simulaciones donde la porosidad no se considera homogénea en todo el medio, esto permite estudiar el comportamiento de las propiedades como la temperatura, la velocidad, líneas de corriente en función de una permeabilidad local. Todo el procedimiento incicia con la imagen de un medio poroso al que se le calcula la porosidad y permeabilidad mediante un tratamiento digital de la imagen, luego estos datos son proporcionados al software como matrices para luego ejecutar la simulación. En éste trabajo de tesis se muestra que la porosidad y la permeabilidad no homogénea juegan un papel muy importante en el estudio del comportamiento de las propiedades macroscópicas, ya que al comparar su comportamiento con el obtenido en una simulación con porosidad homogénea, tienen importantes diferencias en las magnitud y distribuciones de las mismas.

- Se desarrolló un software paralelo en python con CUDA, que es altamente eficiente, hace uso de bibliotecas modernas tales como numpy y numba, siendo ésta última la que permite generar código máquina muy eficiente y además permite el uso de los núcleos de CUDA. Este algoritmo hace uso del MLB-TRM y se muestra que se adapta muy bien a la arquitectura CUDA.

- El software aquí desarrollado es libre y cualquier persona puede hacer uso de él y modificarlo para otros propósitos, se encuentra en el repositorio: `https://github.com/BenjaminNoyola/Melting_Ga`.

- El software es altamente parametrizable, se pueden ajustar las variables físicas del problema, por ejemplo en este trabajo se tomaron las del Galio, pero pueden ajustarse a otra sustancia (Temperatura de fusión, Capacidad calorífica, número de Darcy, viscosidad, etc) también se pueden ajustar las condiciones de frontera, La imagen de entrada puede ser diferente a la que se usa actualmente, los pasos de tiempo pueden variar, el tamaño del dominio también se puede modificar. etc.

Bibliografía

[1] Benjamin Noyola. BenjaminNoyola/Melting_ga, November 2020. original-date: 2018-05-19T06:36:18Z.

[2] Benjamín Salomón Noyola-García and Suemi Rodriguez-Romo. Simulations of Ga melting based on multiple-relaxation time lattice Boltzmann method performed with CUDA in Python. *Mathematics and Computers in Simulation*, 181:170–191, March 2021.

[3] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-Gas Automata for the Navier-Stokes Equation. *Physical Review Letters*, 56(14):1505–1508, April 1986.

[4] Shiyi Chen and Gary D. Doolen. Lattice Boltzmann Method for Fluid Flows. *Annual Review of Fluid Mechanics*, 30(1):329–364, 1998.

[5] P. L. Bhatnagar, E. P. Gross, and M. Krook. A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. *Physical Review*, 94(3):511–525, May 1954.

[6] Xiaoyi He and Li-Shi Luo. Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation. *Physical Review E*, 56(6):6811–6817, December 1997.

[7] Jia Wang, Donghai Wang, Pierre Lallemand, and Li-Shi Luo. Lattice Boltzmann simulations of thermal convective flows in two dimensions. *Computers & Mathematics with Applications*, 65(2):262–286, January 2013.

[8] Qing Liu and Ya-Ling He. Double multiple-relaxation-time lattice Boltzmann model for solid–liquid phase change with natural convection in porous media. *Physica A: Statistical Mechanics and its Applications*, 438(Supplement C):94–106, November 2015.

[9] C. Beckermann and R. Viskanta. Natural convection solid/liquid phase change in porous media. *International Journal of Heat and Mass Transfer*, 31(1):35–46, January 1988.

[10] Piyasak Damronglerd and Yuwen Zhang. Numerical Simulation of Melting in Porous Media via a Modified Temperature-Transforming Model. *Journal of Thermophysics and Heat Transfer*, 24(2):340–347, 2010.

[11] Guy R. McNamara and Gianluigi Zanetti. Use of the Boltzmann Equation to Simulate Lattice-Gas Automata. *Physical Review Letters*, 61(20):2332–2335, November 1988.

[12] Juan P. Giovacchini, Omar E. Ortiz, and Carlos Sacco. El Método de Lattice Boltzmann con Condiciones de Borde en Geometrías Arbitrarias No Regulares. *Mecánica Computacional*, 31(2):165–183, 2012.

[13] A. A. Mohamad. *Lattice Boltzmann Method: Fundamentals and Engineering Applications with Computer Codes*. Springer-Verlag, London, 2011.

[14] Jesús Ojeda Contreras. Aceleración de simulaciones de fluidos Lattice Boltzmann utilizando CUDA. 2009.

[15] Feng Chen, Ai-Guo Xu, Guang-Cai Zhang, and Yong-Long Wang. Two-dimensional MRT LB model for compressible and incompressible flows. *Frontiers of Physics*, 9(2):246–254, April 2014.

[16] Dominique d'Humières. Multiple–relaxation–time lattice Boltzmann models in three dimensions. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 360(1792):437–451, March 2002.

[17] Qing Liu, Ya-Ling He, Qing Li, and Wen-Quan Tao. A multiple-relaxation-time lattice Boltzmann model for convection heat transfer in porous media. *International Journal of Heat and Mass Transfer*, 73(Supplement C):761–775, June 2014.

[18] Michael E. McCracken and John Abraham. Multiple-relaxation-time lattice-Boltzmann model for multiphase flow. *Physical Review E*, 71(3):036701, March 2005.

[19] Zhaoli Guo and T. S. Zhao. A Lattice Boltzmann Model for Convection Heat Transfer in Porous Media. *Numerical Heat Transfer, Part B: Fundamentals*, 47(2):157–177, January 2005.

[20] Grashof-Denkmünze. Zeitschrift des Vereines Deutscher Ingenieure. page 64 v., 1857.

[21] Und Energiesysteme and Werner Schmidt. Influence of Multidimensionality and Interfacial Friction on the Coolability of Fragmented Corium. January 2004.

[22] R. D. Barree and M. W. Conway. Beyond Beta Factors: A Complete Model for Darcy, Forchheimer, and Trans-Forchheimer Flow in Porous Media. Society of Petroleum Engineers, January 2004.

[23] David W. Hahn and M. Necati Ã–zisik. *Heat Conduction*. John Wiley & Sons, August 2012.

[24] Wei Gong, Kévyn Johannes, and Frédéric Kuznik. Numerical Simulation of Melting with Natural Convection Based on Lattice Boltzmann Method and Performed with CUDA Enabled GPU. *Communications in Computational Physics*, 17(5):1201–1224, May 2015.

[25] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors, Second Edition: A Hands-on Approach*. Morgan Kaufmann, Amsterdam, edición: 2 edition, 2012.

[26] Luc Alberts. *Initial porosity of random packing: Computer simulation of grain rearrangement*. PhD thesis, 10 2005.

Apéndice A

Instalación del software Melting_Ga

El software se descarga libremente del repositorio git:

`https://github.com/BenjaminNoyola/Melting_Ga.git`

Es necesario tener instalado 'git' para clonar el repositorio alojado en github. Con los siguientes comandos se descarga el software del repositorio desde una terminal para Linux o Mac:

```
git clone https://github.com/BenjaminNoyola/Melting_Ga.git
```

Para clonar éste repositorio con Windows seguir las instrucciones de la siguiente liga: `https://git-scm.com/book/en/v2/Getting-Started-Installing-Git` apuntando al repositorio: `https://github.com/BenjaminNoyola/Melting_Ga.git`

Después de haber descargado el repositorio, se observan 2 directorios adentro:

- Case_1
- Case_2

En primer lugar se deben instalar las bibliotecas requeridas mediante el siguiente comando:

```
pip install -r requirements.txt
```

Problema 1

Luego si se desea ejecutar el Case_1 entonces al acceder a éste directorio donde se encuentran 3 directorios:

- Numpy: resuelve el problema 1 planteado en la sección 2.4 utilizando cómputo en serie, en este caso se ejecuta el código con el siguiente comando:

```
python MRT-LB_serial_Numpy.py
```

- Improved_Paralel: también se resuelve el problema 1 planteado en la sección 2.4 utilizando cómputo paralelo, en este caso se ejecuta con el siguiente comando:

```
python MRT-LB_parallel_CUDA_2.py
```

- Results_Analytic_Vs_Numeric: En ésta sección se resuelve el problema 1 planteado en la sección 2.4 mediante una solución analítica y se compara con el resultado de la solución numérica.

En los directorios que contienen la solución del problema 1 hay un archivo Parameters.json donde se ajustan los parámetros del problema como sea requerido.

Problema 2

Para resolver el problema 2 se debe acceder al directorio Case_2, que contiene los siguientes directorios:

- Poro_homogeneous: En éste directorio se encuentra la solución paralela del problema 2 con porosidad homogénea, para ejecutar el programa se utiliza el siguiente comando:

```
python Main_change_phase_poro_homo.py
```

- Poro_from_image: En éste directorio se resuelve el problema 2 partiendo de una imagen porosa, mediante el siguiente comando:

```
python Main_change_phase_poro_image.py
```

El archivo 'Parameters.json' permite ajustar los parámetros del problema como sea requerido.

Apéndice B

Convección natural utilizando el método de lattice Boltzmann

Convección natural

En esta sección se muestra la simulación de la convección natural de un fluido en un dominio rectangular, la implementación está hecha en Fortran 90 y también en python 3.8. Se puede descargar libremente del repositorio de github: `https://github.com/BenjaminNoyola/Tesis/tree/main/natural_convection`. Y éste tema está extensamente explicado en la sección 6.6.1 del libro [A. A. Mohamad, Lattice Boltzmann Method, Springen, 2011] [13]. Esta implementación se utilizó para estudiar en primer lugar el fenómeno de la convección natural, que sirve como base para el software 'Melting_Ga' que en esta tesis se presenta.

La simulación implementada en el lenguaje python está repartida en dos carpetas, las cuales contienen diferentes implementaciones pero al fin se obtienen los mismos resultados, las cuales son: 'python' y 'python-nopy'. Para ejecutar los códigos hay que acceder a cualquiera de ellas y ejecutar:

```
python natural_convection.py
```

En cada carpeta se encuentra el código principal con los parámetros del problema y un código (modulos_nat.py) con todos los módulos y el decorador @jit (just in time) de numba, que permite hacer de python un lenguaje máquina más eficiente.

Para ejecutar la misma simulación implementada en Fortran, se debe acceder a la carpeta 'fortran' del repositorio y ejecutar de la siguiente manera:

```
gfortran -o ejecutable natural_convection.f90
./ejecutable
```

Eficiencias:

C, C++ y Fortran son considerados los lenguajes de programación mas eficientes para ejecutar cálculos científicos complejos, pero recientemente python permite generar código máquina muy eficiente bajo la biblioteca 'numba' [https://numba.pydata.org/]. En este documento se compara el código implementado en Fortran y python donde se obtienen los mismos resultados en las simulaciones de convección natural de un fluido, El código en fortran aparece en el apéndices de este libro. Para fines de evaluar la eficiencia computacional de python numba, se implementó dicho código en python-numba, los resultados fueron los siguientes en la misma computadora:

- Tiempo Fortran: real 9m42.625s user 9m42.088s sys 0m0.032s
- Tiempo python: real 2m52.063s user 2m51.656s sys 0m0.048s

En éste problema en específico se encontró que 'python-numba' fue 3.38 veces más rápido que Fortran.

A continuación se presenta el código que se utilizó para graficar los resultados, obtenidos de las simulaciones con los códigos de este mismo apéndice.

```
# Autor: Benjamín Salomón Noyola García
# Tema: Gráfico 2D.

import numpy as np
from StringIO import StringIO
import matplotlib.pyplot as plt

pfile=open('strf.txt','r')
data=pfile.read()
pfile.close()
data=np.genfromtxt(StringIO(data))
n=len(data)
##print n
data_c = np.copy(data)
for i in range(n):
    cont=0
    for j in range(n):
        data[i,j]=data_c[i,n-1-cont]
        cont+=1
fig = plt.figure(1)
plt.title('Streamlines')
```

```
plt.xlabel('Y')
plt.ylabel('X')
cs1 = plt.contourf((data), 50) # Pintamos 100 niveles con relleno
plt.colorbar()
plt.savefig("STR.png")
plt.show()
```

El resultado obtenido se muestra en la figura B.1

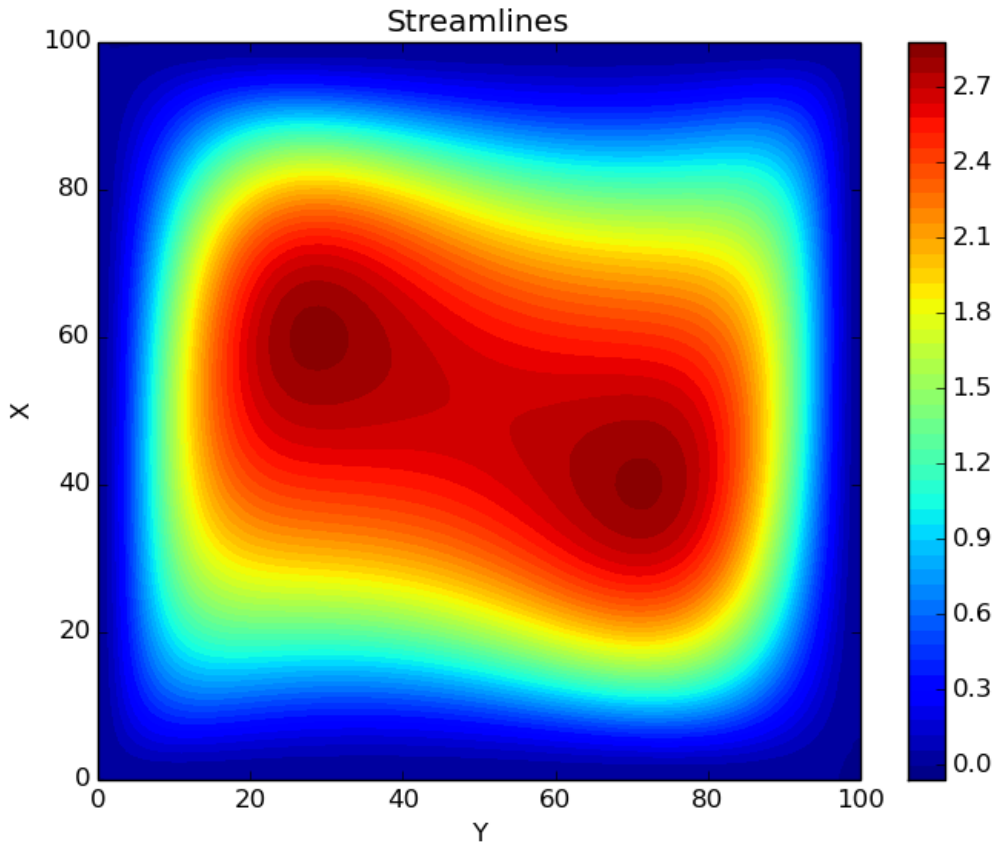


Figura B.1: Líneas de corriente en una cavidad cuadrada, Ra $=5^5$; Pr $= 0.71$

Difusión

También se encuentra implementado el problema de la difusión, el cual se explica detalladamente en el capítulo 3 del libro [A. A. Mohamad, Lattice Boltzmann Method,

Springen, 2011] y su implementación fue hecha en python 3.8 y Fortran 90, se puede localizar en el repositorio `https://github.com/BenjaminNoyola/Tesis/tree/main/difusion_mohamad`.

Para ejecutar el código en python se debe acceder a la carpeta 'python' y ejecutar el siguiente comando:

```
python difusion.py
```

y para ejecutar en Fortran, acceder a la carpeta 'fortran':

```
gfortran -o ejecutable codigo.f90
./ejecutable
```

Los resultados obtenidos se muestran en las figuras B.2 y B.3

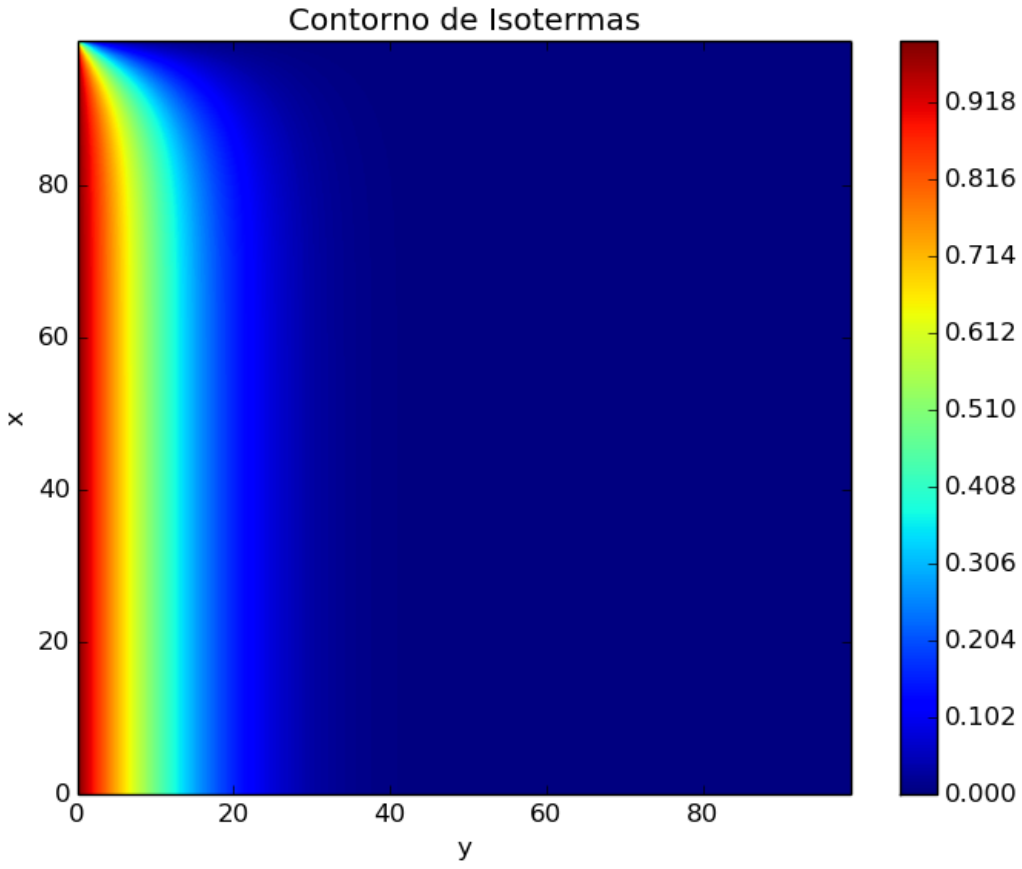


Figura B.2: Isotermas

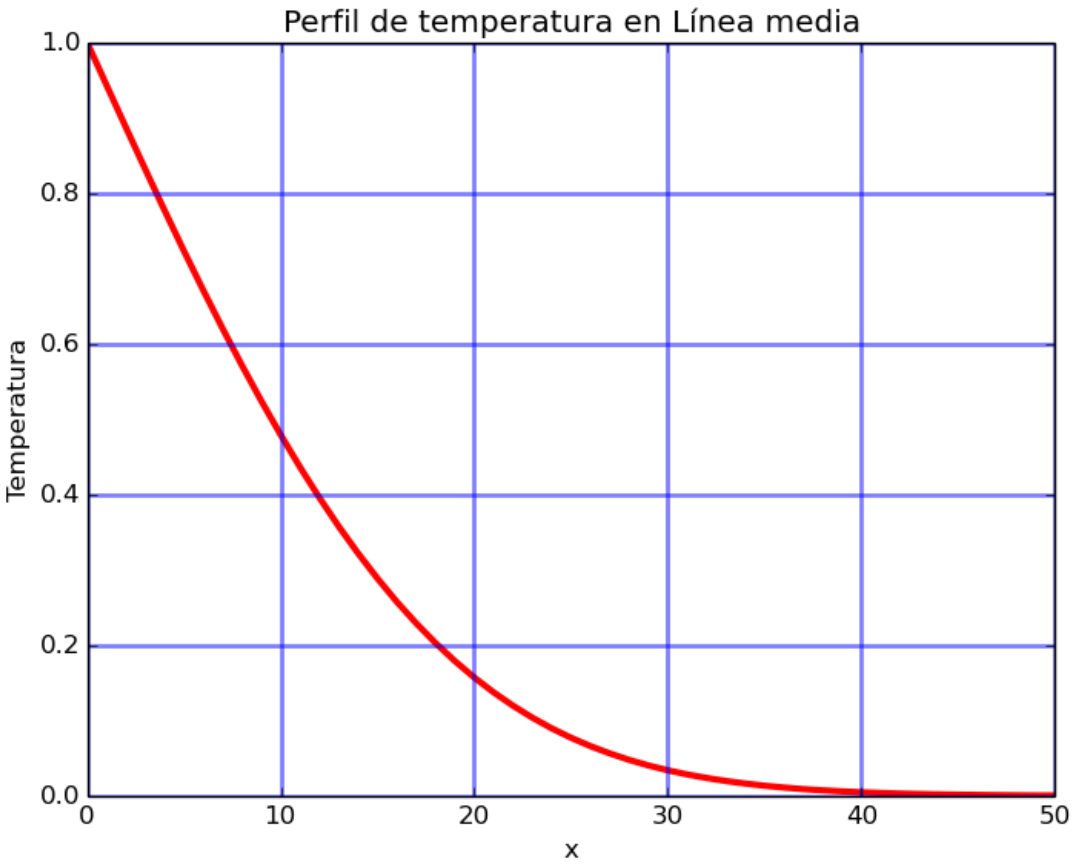


Figura B.3: perfil de temperatura de una líne que pasa por mitad

Apéndice C

Generación de imágenes con porosidad no homogénea

Las imágenes se realizaron utilizando un software que genera porosidades empacadas aleatoriamente, descrito por Luc Alberts [26], el software se puede obtener del repositorio de github: `https://github.com/BenjaminNoyola/Tesis/tree/main/Porosity`.

Para ejecutar el código se hace con el siguiente comando:

```
python Calculate_Porosity.py
```

Si se requieren obtener nuevas simulaciones, entonces modificar los parámetros:

```
a = rndp(lx=128., ly=128., rmin=0.5, rmax=2.0,
target_porosity = 0.368, packing='rnd')
```

donde 'lx' es la dimensión x de la resolución de la imagen, 'ly' es la dimensión y de la imagen, 'rmin' es el radio mínimo de la partícula, al generarse aleatoriamente, 'rmax' es el radio máximo. 'target_porosity' es la porosidad buscada en la imagen, 'packing' es la técnica de generación de la porosidad, 'rnd' se refiere a que se hará un empacado aleatorio para obtener la imagen porosa.

Al ejecutar éste código se obtienen imágenes como la mostrada en la figura 2.5.

UNAM, 2021