VNIVERSIDAD NACIONAL AVTNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

APLICACIÓN DE DIFERENTES ESTRATEGIAS DE CÓMPUTO PARALELO EN LA SIMULACIÓN DE PROCESOS DE RECUPERACIÓN SECUNDARIA DE HIDROCARBUROS

T E S I S

Que para obtener el grado de:

MAESTRO EN CIENCIAS
(COMPUTACIÓN)

P R E S E N T A:

DANIEL MONSIVAIS VELÁZQUEZ

DIRECTOR DE TESIS:

DR. LUIS MIGUEL DE LA CRUZ SALAS

MÉXICO, D.F. 2011.

A Carmen, que un día nos dejó, y a Lola que a veces nos quiere dejar.

.

A dos Chelas y a dos Raules, que sé que piensan en mí.

Agradecimientos

Quiero agradecer especialmente al Dr. Luis Miguel de la Cruz Salas, por su apoyo, atención y consejos en la realización de este trabajo, y en las actividades del grupo de desarrollo de los simuladores WAG-Aire. Dentro del mismo proyecto, quiero agradecer a mis seis compañeros *"programadores"* por la convivencia que tuvimos durante más de un año. También quiero agradecer al Doctores Héctor Benítez, Guillermo Hernández, Ismael Herrera y Jorge Ortega, por las correcciones y consejos que me dieron al revisar este trabajo.

Este trabajo fue apoyado por el proyecto IX 101110 de la convocatoria 2010 del Observatorio de Visualización Ixtli.

Resumen

En este documento se presenta un análisis del desempeño de tres bibliotecas paralelas, construidas cada una sobre un esquema de memoria distinto, empleadas para resolver el problema de Buckley-Leverett, el cual es el modelo básico en la recuperación secundaria de hidrocarburos. Se compara el uso de GPU's (CUDA), de memoria compartida (IntelMKL) y de memoria distribuida (PETSc), para resolver los sistemas lineales de presión y saturación (IMPES) generados por una biblioteca computacional (TUNA) que simula problemas de dinámica de fluidos mediante el método de volumen finito. Se obtuvo que para este tipo de problemas, el uso de GPU's tiene la mejor aceleración ($\approx 2.6X$).

Contenido

Lista de Figuras

Lista de Tablas

Lista de Algoritmos

Capítulo 1

Introducción

A finales del siglo pasado, la tecnología de tarjetas para manejo de gráficos se desarrolló enormemente, impulsada principalmente por las exigencias de la creciente industria de videojuegos, pasando en el transcurso de 10 años, de pequeñas unidades para aceleración de gráficos en 2D, a equipos capaces de visualizar simulaciones de fluidos o de exploraciones médicas en tiempo real, aplicaciones 3D, realidad virtual y aumentada, por mencionar algunas. La necesidad de equipos que puedan procesar para visualización grandes cantidades de información en tiempos cortos, ha dejado hoy en día tarjetas que rivalizan en velocidad y memoria, con los mejores procesadores disponibles en mercado (Intel y AMD). Las aplicaciones que explotan el uso de tarjetas gráficas ha revolucionado muchos campos de la ciencia y la tecnología, ya que ahora es posible en una computadora convencional con una tarjeta gráfica relativamente simple visualizar:

- simulaciones de dinámica molecular en bioquímica,
- simulaciones de dinámica de fluidos computacional,
- imágenes 3D de tomografías y resonancias en medicina, así como realizar cirugía asistida por computadora.
- predicciones sobre la propagación de desastres naturales (sismos, tsunamis, erupciones, huracanes, etc)

A pesar del gran desarrollo de dichas tarjetas, éstas no podían ser usadas de manera sencilla para otra cosa que no fuera el manejo de gráficos. Hasta que en el 2005, el fabricante de tarjetas gráficas Nvidia liberó un compilador y varias bibliotecas que permitían la programación de sus tarjetas para propósitos generales (GPGPU o *General Programming Graphics Processor Unit* por sus siglas en inglés). Esto abrió la oportunidad de explotar la naturaleza paralela de estas tarjetas en la optimización de aplicaciones que requieren de técnicas y arquitecturas paralelas para poder ejecutarse en tiempos razonables, y que en general utilizan esquemas de memoria compartida o distribuida. A partir de su aparición, el GPGPU ha sido utilizado en aplicaciones diversas, desde el procesamiento de señales, hasta el plegamiento de proteínas y la alineación de secuencias. Con la aparición formal del GPGPU (hubo algunos intentos de realizarla mediante el mapeo de matrices a pixeles[Kirk10] con cierto éxito), el cómputo paralelo tuvo una nueva técnica para atacar los problemas complejos para el cual es requerido. Las aplicaciones de cómputo paralelo son ubicuas en la ciencia y en la ingeniería, resolviendo problemas de clima y estado del tiempo, predicciones de mercados financieros, cosmología, física de partículas y de altas energías, vulcanología, etc.

Entre las áreas donde la aplicación del cómputo paralelo es una necesidad, está la dinámica de fluidos. Con técnicas de paralelización, la descripción numérica del flujo de líquidos y gases puede ser estudiada a gran escala. Dentro de los estudios de mayor importancia para la dinámica de fluidos, principalmente para la economía mundial, y en particular para México, está el estudio y descripción de la producción de yacimientos petrolíferos. Para empresas como Petróleos Mexicanos (PEMEX), dedicadas a la extracción de hidrocarburos, la descripción del flujo de estas sustancias dentro de un yacimiento es indispensable. La investigación desarrollada sobre este tema es extensa, y sigue en desarrollo. Existen varios tipos de recuperación de hidrocarburos, para los cuales se utilizan diversas técnicas que incluyen la inyección de otras sustancias (gases, agua, surfactantes, etc) e incluso la ignición de frentes de combustión *in situ*. La predicción de la producción de un yacimiento a lo largo de tiempo es una tarea complicada, en la que intervienen múltiples factores, como son las propiedades de las fluidos participantes y del medio, apertura y cierre de pozos, tazas de inyección y extracción, topología de yacimiento, propiedades de interfaz entre sustancias,

por mencionar algunas, e incluso otras más del tipo económico, como puede ser el costo de producción contra el costo cambiante de los hidrocarburos en el mercado, oferta y demanda, etc. Para la descripción física del problema, debido a la diversidad de factores que intervienen en él, un método general que simplifique su modelación matemática y computacional es conveniente. Un candidato natural es la formulación axiomática [Herrera88], en la cual se plantean ecuaciones de balance entre propiedades extensivas e intensivas para describir la dinámica del problema, y establece un marco de desarrollo más general para atacar problemas similares.

En general, la naturaleza complicada de la mayoría de los problemas de la dinámica de fluidos computacional dificulta la predicción ya que se puede requerir de muchos recursos computacionales para realizarla. Para solventar estas necesidades con el desarrollo actual de la tecnología, generalmente se recurre a tres tipos de tecnologías de paralelización: usando memoria compartida, usando memoria distribuida y mediante las tarjetas gráficas mencionadas anteriormente. Con la cantidad de memoria disponible actualmente y la velocidad de los procesadores actuales, es posible hacer simulaciones a escalas relativamente grandes en una computadora de costo no muy elevado. Esto abre la pregunta sobre cuál es la tecnología adecuada para resolver cada tipo específico de problema que requiera paralelización, y si una técnica es igualmente apta para un mismo problema al cambiar este de tamaño.

1.1. Objetivos

La intención principal de este trabajo es comparar el desempeño entre tres formas de paralelización (memoria compartida, distribuida y tarjetas gráficas), para un modelo específico en la recuperación secundaria de hidrocarburos (Buckley-Leverett). Gracias al avance de la tecnología de microprocesadores, éstas tres formas de paralelización pueden evaluarse en una sola máquina de alto desempeño (con pocos procesadores), sin requerir de una cluster de computadoras para analizar un problema con un tamaño relativamente grande. Para esto se utilizará el paquete computacional TUNA, el cual está enfocado a la solución de las ecuaciones diferenciales resultantes de la dinámica de fluidos. Se usará una

interfaz a distintas bibliotecas paralelas para la solución de los sistemas lineales que el TUNA genere. Se plantearon las siguientes metas:

- Dar una descripción general de una metodología de modelación matemática y computacional (formulación axiomática) y aplicarla al modelo de Buckley-Leverett para obtener el modelo numérico.
- Resolver de manera serial el modelo numérico mediante el paquete computacional TUNA.
- Resolver el mismo problema utilizando distintos tipos de paralelización mediante una interfaz al TUNA y comparar su desempeño.

1.2. Descripción de capítulos

En el capítulo 2 se describe la formulación axiomática, su aplicación al problema de Buckley-Leverett y la discretización del problema para resolverlo numéricamente.

En los capítulos 3 y 3.3 se describe los tipos de cómputo paralelo a evaluar, así como las bibliotecas númericas usadas en la programación del modelo numérico sobre cada tipo de paralelización.

En el capítulo 4 se muestra las implementaciones para cada tipo de paralelización, y se hace un análisis de las operaciones involucradas por los métodos de solución empleados para resolver las ecuaciones discretizadas del modelo numérico.

En los capítulo 5 y 6 se muestra los resultados obtenidos del desempeño de cada biblioteca paralela evaluada, el análisis de dichos resultados y las conclusiones finales del trabajo.

Capítulo 2

Modelación matemática y computacional

2.1. Formulación axiomática

2.1.1. Propiedades Intensivas y extensivas

Para obtener las ecuaciones de flujo bifásico en medios porosos, se aplicará la formulación axiomática descrita por Herrera et al [Herrera88]. Esta formulación consiste en identificar un conjunto de propiedades intensivas y extensivas relevantes para el problema bajo estudio y establecer balances de ellas en un volumen representativo $B(t)$ del medio continuo.

Las propiedades intensivas son funciones, escalares o vectoriales, definidas en cada paso de tiempo para cada una de las partículas del volumen representativo. Por ejemplo la velocidad de una de estas partículas es una propiedad intensiva vectorial, mientras que la densidad es una propiedad intensiva escalar. En general, estas funciones hacen corresponder a cada partícula y a cada tiempo un número real o un vector.

Una función que a cada cuerpo (volumen representativo) de un sistema continuo y a cada tiempo le asocia un número real o un vector, se le llama propiedad extensiva cuando está dada por una integral como la que sigue

$$E(B,t) \equiv \int_{B(t)} \psi(\underline{x},t)d\underline{x}$$

donde $\underline{x}$ es la posición en el espacio y t el tiempo. El integrando de la ecuación anterior define una función $\psi(\underline{x},t)$ la cual es una propiedad intensiva Euleriana. Esta ecuación establece una correspondencia biunívoca entre las propiedades intensivas y extensivas: dada la representación Euleriana de cualquier propiedad intensiva, su integral sobre el dominio ocupado por cualquier cuerpo $B(t)$, define una propiedad extensiva correspondiente.

2.1.2. Balance de las propiedades intensivas y extensivas

En general, el cambio de una propiedad extensiva E con el paso del tiempo se debe a que ésta se produce en el interior del sistema o a que entra o sale por la frontera. Matemáticamente este balance se escribe como sigue

$$\frac{dE}{dt} = \frac{d}{dt}\int_{B(t)} \psi(\underline{x},t)d\underline{x} = \int_{B(t)} q(\underline{x},t)d\underline{x} + \int_{\partial B(t)} \underline{\tau}(\underline{x},t)\cdot\underline{n}dS \tag{2.1}$$

donde $q(\underline{x},t)$ y $\underline{\tau}(\underline{x},t)$ representan la *"generación"* y el vector de *"flujo"* de la propiedad extensiva respectivamente. $B(t)$ es un volumen representativo como el que se muestra en la figura 2.1 (véase [Herrera88, Herrera91]), y $\underline{n}$ es el vector normal a la superficie S la cual envuelve a $B(t)$.

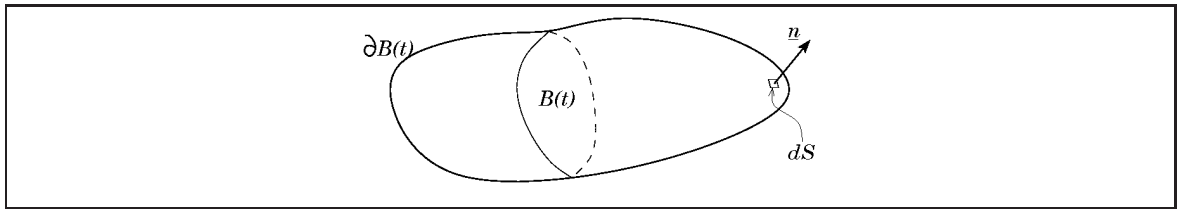

Figura 2.1: Volumen representativo del medio continuo.

Aplicando varios resultados matemáticos, la ecuación (2.1) se puede escribir como

$$\int_{B(t)} \left\{ \frac{\partial\psi}{\partial t} + \nabla\cdot(\underline{v}\psi) \right\} d\underline{x} = \int_{B(t)} q d\underline{x} + \int_{B(t)} \nabla\cdot\underline{\tau}d\underline{x} \tag{2.2}$$

Lo anterior produce la siguiente ecuación diferencial de balance local para la propiedad extensiva ψ

$$\frac{\partial \psi}{\partial t} + \nabla \cdot (\underline{v}\psi) = q + \nabla \cdot \underline{\tau} \tag{2.3}$$

2.1.3. Forma conservativa de las ecuaciones de balance

Definiendo una *"función de flujo"* como $\underline{\mathcal{F}} = \underline{v}\psi - \underline{\tau}$ es posible escribir (2.2) de la siguiente manera:

$$\int_{B(t)} \frac{\partial \psi}{\partial t} d\underline{x} + \int_{B(t)} \nabla \cdot \underline{\mathcal{F}} d\underline{x} = \int_{B(t)} q d\underline{x} \tag{2.4}$$

lo que produce la siguiente ecuación diferencial.

$$\frac{\partial \psi}{\partial t} + \nabla \cdot \underline{\mathcal{F}} = q \tag{2.5}$$

Las ecuaciones (2.4) y (2.5) se conocen como la *"forma conservativa"* de las ecuaciones (2.2) y (2.3) respectivamente, véase [Leveque80].

Aplicando el teorema de Gauss a la ecuación (2.4) se obtiene

$$\int_{B(t)} \frac{\partial \psi}{\partial t} d\underline{x} + \int_{S} \underline{\mathcal{F}} \cdot \underline{n} dS = \int_{B(t)} q d\underline{x} \tag{2.6}$$

2.1.4. Modelo matemático de sistemas multifásicos

En un sistema multifásico dentro de un medio poroso, la masa de fluido de la fase α es una propiedad extensiva que se escribe como:

$$E = \int_{B(t)} \phi \rho_\alpha S_\alpha d\underline{x},$$

donde ϕ es la porosidad del medio, ρ_α y S_α es la densidad y la saturación de la fase α. De acuerdo con la formulación axiomática, la correspondiente propieda intensiva en este caso es:

$$\psi = \phi \rho_\alpha S_\alpha.$$

Suponiendo que no hay difusión, es decir $\tau = 0$, de acuerdo con (2.3), la ecuación de balance local para la masa de fluido de la fase α se escribe como

$$\frac{\partial(\phi \rho_\alpha S_\alpha)}{\partial t} + \nabla \cdot (\underline{v} \phi \rho_\alpha S_\alpha) = \rho_\alpha q_\alpha.$$

Definiendo la velocidad de Darcy de la fase α como $\underline{u}_\alpha = \underline{v} \phi S_\alpha$, la ecuación diferencial de balance de masa para una fase de fluido α, que forma parte de un sistema multifásico, se escribe como

$$\frac{\partial(\phi \rho_\alpha S_\alpha)}{\partial t} + \nabla \cdot (\rho_\alpha \underline{u}_\alpha) = \rho_\alpha q_\alpha. \tag{2.7}$$

La ley de Darcy para un sistema multifásico se expresa como

$$\underline{u}_\alpha = -\frac{\underline{\underline{k}} k_{r\alpha}}{\mu_\alpha} (\nabla p_\alpha - \rho_\alpha \underline{g}), \tag{2.8}$$

donde $\underline{\underline{k}}$ es el tensor de permeabilidad absoluta que depende del medio (la roca) y $k_{r\alpha}$ es la permeabilidad relativa a la fase α. La viscosidad y la presión se denotan por μ_α y p_α respectivamente, mientras que $\underline{g}$ es la fuerza de la gravedad apuntando en dirección negativa del eje y.

Sustituyendo (2.8) en (2.7), y definiendo la movilidad de la fase α como

$$\lambda_\alpha = \frac{k_{r\alpha}}{\mu_\alpha}, \tag{2.9}$$

se obtiene

$$\frac{\partial(\phi \rho_\alpha S_\alpha)}{\partial t} - \nabla \cdot (\rho_\alpha \lambda_\alpha \underline{\underline{k}} (\nabla p_\alpha - \rho_\alpha \underline{g})) = \rho_\alpha q_\alpha. \tag{2.10}$$

La ecuación (2.10) representa un conjunto de ecuaciones diferenciales parciales, una para cada fase. Estas ecuaciones se completan con una serie de fórmulas constitutivas las cuales describen la dependencia de una variable con respecto de las otras e integran el

conocimiento científico y tecnológico del problema que se esté estudiando. En un sistema multifásico totalmente saturado, generalmente se usan las siguientes ecuaciones constitutivas

$$\sum_{\alpha=1}^{N} S_\alpha = 1 \quad \text{y} \tag{2.11}$$

$$p_{c\psi\alpha} = p_\psi - p_\alpha, \alpha \neq \psi \tag{2.12}$$

donde $p_{c\psi\alpha}$es la presión capilar, N es el número de fases, ψ y α representan una fase no mojadora (*non-wetting*) y una fase mojadora (*wetting*), respectivamente.

La combinación de las ecuaciones (2.10), (2.11) y (2.12) produce un sistema de ecuaciones fuertemente acopladas (*fully coupled*). Por ejemplo, para un sistema de dos fases, considerando el agua como la fase mojadora y el aceite como la fase no mojadora, en una formulación presión–saturación, y usando p_o y S_w como variables primarias, se obtiene el siguiente sistema de ecuaciones

$$S_w + S_o = 1, \tag{2.13}$$

$$p_c \equiv p_{cow} = p_o - p_w, \tag{2.14}$$

$$\frac{\partial(\phi\rho_w S_w)}{\partial t} - \nabla \cdot (\rho_w \lambda_w \underline{\underline{k}}(\nabla p_o - \nabla p_c) - \rho_w \underline{g})) = \rho_w q_w, \tag{2.15}$$

$$-\frac{\partial(\phi\rho_o S_w)}{\partial t} - \nabla \cdot (\rho_o \lambda_o \underline{\underline{k}}(\nabla p_o - \rho_o \underline{g})) = \rho_o q_o. \tag{2.16}$$

2.2. Formulación presión–saturación

La formulación de flujo fraccional de un sistema de N fases, desacopla el sistema de ecuaciones diferenciales parciales fuertemente acoplado, por ejemplo (2.15) y (2.16), en una ecuación para la presión y $N-1$ ecuaciones de transporte de saturación. Esto produce ecuaciones débilmente acopladas que pueden ser resueltas de manera separada.

En esta formulación, la ecuación de presión se puede resolver de manera aislada, y los resultados obtenidos se usan para resolver las ecuaciones de transporte para la saturación. Posteriormente, los resultados de la solución de las ecuaciones de saturación, se insertan en la ecuación de presión para obtener la solución en el paso de tiempo siguiente. Este proceso es iterativo y se repite durante un número de pasos de tiempo previamente definidos. En las secciones que siguen se muestra como llegar a la formulación presión – saturación.

2.2.1. Ecuación de presión

Para obtener la ecuación de presión, primero se divide la ecuación (2.7) por ρ_α, con lo que se obtiene

$$\frac{1}{\rho_\alpha}\left[\frac{\partial(\phi\rho_\alpha S_\alpha)}{\partial t} + \nabla\cdot(\rho_\alpha \underline{u}_\alpha) - \rho_\alpha q_\alpha\right] = 0,$$

sumando sobre todas las fases se llega a lo siguiente

$$\sum_\alpha \left\{\frac{1}{\rho_\alpha}\left[\frac{\partial(\phi\rho_\alpha S_\alpha)}{\partial t} + \nabla\cdot(\rho_\alpha \underline{u}_\alpha)\right] - q_\alpha\right\} = 0,$$

desarrollando las derivadas de la ecuación anterior se obtiene

$$\sum_\alpha \left\{\frac{1}{\rho_\alpha}\left[\rho_\alpha S_\alpha\frac{\partial\phi}{\partial t} + \phi S_\alpha\frac{\partial\rho_\alpha}{\partial t} + \rho_\alpha\phi\frac{\partial S_\alpha}{\partial t} + \rho_\alpha\nabla\cdot\underline{u}_\alpha + \underline{u}_\alpha\cdot\nabla\rho_\alpha\right] - q_\alpha\right\} = 0,$$

y finalmente rearreglando y aplicando la relación (2.11) a la ecuación anterior, se puede escribir

$$\frac{\partial\phi}{\partial t} + \sum_\alpha \nabla\cdot\underline{u}_\alpha + \sum_\alpha \frac{1}{\rho_\alpha}\left[\phi S_\alpha\frac{\partial\rho_\alpha}{\partial t} + \underline{u}_\alpha\cdot\nabla\rho_\alpha\right] - \sum_\alpha q_\alpha = 0. \tag{2.17}$$

Se define ahora la velocidad total $\underline{u}$ como

$$\underline{u} = \sum_\alpha \underline{u}_\alpha, \tag{2.18}$$

aplicando la divergencia a esta relación se obtiene

$$\nabla \cdot \underline{u} = \nabla \cdot \sum_{\alpha} \underline{u}_{\alpha} = \sum_{\alpha} \nabla \cdot \underline{u}_{\alpha}. \tag{2.19}$$

Insertando (2.19) en (2.17) se obtiene una ecuación general multifásica para la presión

$$\frac{\partial \phi}{\partial t} + \nabla \cdot \underline{u} + \sum_{\alpha} \frac{1}{\rho_{\alpha}} \left[\phi S_{\alpha} \frac{\partial \rho_{\alpha}}{\partial t} + \underline{u}_{\alpha} \cdot \nabla \rho_{\alpha} \right] - \sum_{\alpha} q_{\alpha} = 0. \tag{2.20}$$

De la definición de la ley de Darcy se tiene que

$$\underline{u} = \sum_{\alpha} \underline{u}_{\alpha} = \sum_{\alpha} \left[-\frac{\underline{\underline{k}} k_{r\alpha}}{\mu_{\alpha}} (\nabla p_{\alpha} - \rho_{\alpha} \underline{g}) \right]. \tag{2.21}$$

Por otro lado, la función de flujo fraccional de la fase α se define como

$$f_{\alpha} = \frac{\lambda_{\alpha}}{\lambda} \Longrightarrow \lambda_{\alpha} = f_{\alpha} \lambda \tag{2.22}$$

donde $\lambda = \sum_{\alpha} \lambda_{\alpha}$ es la movilidad total, por lo tanto $\sum_{\alpha} f_{\alpha} = 1$.

Sustituyendo (2.22) en (2.21) se obtiene

$$\underline{u} = -\lambda \underline{\underline{k}} \left[\sum_{\alpha} f_{\alpha} \nabla p_{\alpha} - \sum_{\alpha} f_{\alpha} \rho_{\alpha} \underline{g} \right] \tag{2.23}$$

Con esta formulación de la velocidad total, la ecuación de presión (2.20) puede ser escrita como sigue

$$\begin{aligned} \frac{\partial \phi}{\partial t} \quad &- \quad \nabla \cdot \lambda \underline{\underline{k}} \left[\sum_{\alpha} f_{\alpha} \nabla p_{\alpha} - \sum_{\alpha} f_{\alpha} \rho_{\alpha} \underline{g} \right] \\ &+ \quad \sum_{\alpha} \frac{1}{\rho_{\alpha}} \left[\phi S_{\alpha} \frac{\partial \rho_{\alpha}}{\partial t} + \underline{u}_{\alpha} \cdot \nabla \rho_{\alpha} \right] - \sum_{\alpha} q_{\alpha} = 0. \end{aligned} \tag{2.24}$$

Presión global

Es posible definir una presión global p_g tal que $\nabla p_g = \sum_{\alpha} f_{\alpha} \nabla p_{\alpha}$ de tal manera que la ecuación (2.23) se escribe como función de p_g

$$\underline{u} = -\lambda \underline{\underline{k}} \left[\nabla p_g - \sum_{\alpha} f_{\alpha} \rho_{\alpha} \underline{g} \right]. \tag{2.25}$$

Usando esta formulación para la velocidad total, se puede escribir una ecuación en términos de la presión global como sigue

$$\begin{aligned} \frac{\partial \phi}{\partial t} \quad - \quad & \nabla \cdot \underbrace{\lambda \underline{\underline{k}} \left[\nabla p_g - \sum_{\alpha} f_\alpha \rho_\alpha \underline{g} \right]}_{\underline{u}} \\ + \quad & \sum_{\alpha} \frac{1}{\rho_\alpha} \left[\phi S_\alpha \frac{\partial \rho_\alpha}{\partial t} + \underline{u}_\alpha \cdot \nabla \rho_\alpha \right] - \sum_{\alpha} q_\alpha = 0. \end{aligned} \tag{2.26}$$

Presión de fase

En esta formulación se usa una presión de fase para obtener la ecuación de presión. El uso de una presión de fase p_α tiene más significado físico que una presión global. Considérese un sistema de dos fases: agua (w) y aceite (o). Para obtener la ecuación de presión en términos de una de las presiones de fase, por ejemplo la presión del agua, primero se hace uso de la ecuación (2.12) para llegar a la siguiente versión modificada de la ecuación (2.23):

$$\underline{u} = -\lambda \underline{\underline{k}} \Big[\nabla p_w + f_o \nabla p_c - (f_w \rho_w + f_o \rho_o) \underline{g} \Big]. \tag{2.27}$$

Sustituyendo esta forma de la velocidad total en la ecuación (2.20) y suponiendo que ambos fluidos son incompresibles y que la porosidad es constante se obtiene:

$$\begin{aligned} \frac{\partial \phi}{\partial t} \quad - \quad & \nabla \cdot \underbrace{\lambda \underline{\underline{k}} \left[\nabla p_w + f_o \nabla p_c - \sum_{\alpha=w,o} f_\alpha \rho_\alpha \underline{g} \right]}_{\underline{u}} \\ + \quad & \sum_{\alpha=w,o} \frac{1}{\rho_\alpha} \left[\phi S_\alpha \frac{\partial \rho_\alpha}{\partial t} + \underline{u}_\alpha \cdot \nabla \rho_\alpha \right] - \sum_{\alpha=w,o} q_\alpha = 0. \end{aligned} \tag{2.28}$$

Una vez conocidas las presiones p_w y p_o, las correspondientes velocidades de fase se pueden calcular usando la ley de Darcy, ecuación (2.8).

2.2.2. Ecuación de saturación

La solución de la ecuación de presión produce la velocidad que se requiere para resolver la ecuación de saturación. En la **formulación de presión de fase**, la saturación puede calcularse directamente de la ecuación de balance de masa (2.7). En un sistema de dos fases agua y aceite, la ecuación para la saturación del agua se escribe como

$$\frac{\partial(\phi\rho_w S_w)}{\partial t} + \nabla\cdot(\rho_w \underline{u}_w) - \rho_w q_w = 0, \tag{2.29}$$

Para la **formulación de presión global**, que produce la velocidad total $\underline{u}$ pero no la velocidad de fase, se debe derivar una ecuación especial para la saturación. Igual que en la formulación de presión de fase, esta ecuación se obtiene de la ecuación de balance de masa (2.7). Para la fase agua se tiene

$$\frac{\partial(\phi\rho_w S_w)}{\partial t} + \nabla\cdot\left[\rho_w\left(f_w\underline{u} + \bar{\lambda}\underline{\underline{k}}(\nabla p_c + (\rho_w - \rho_o)\underline{g})\right)\right] = \rho_w q_w. \tag{2.30}$$

donde se ha definido $\bar{\lambda} = \lambda_w\lambda_o/\lambda$. En esta última ecuación se observa la inclusión de la velocidad total $\underline{u}$, la cual puede obtenerse a partir de la presión global p_g.

2.3. Presión–saturación para flujo bifásico

En esta sección se supone que se tiene un sistema de dos fases, agua y aceite, y además que los flujos son incompresibles y porosidad de la roca es constante. Con estas suposiciones, y a partir de la ecuación (2.24) se llega a

$$-\nabla\cdot\lambda\underline{\underline{k}}\Big[f_w\nabla p_w + f_o\nabla p_o - (f_w\rho_w + f_o\rho_o)\underline{g}\Big] - (q_w + q_o) = 0. \tag{2.31}$$

Dado que el aceite es una fase continua y consecuentemente su presión es bien comportada, se obtendrá una ecuación en términos de la presión del aceite.

Usando $p_c \equiv p_{cow} = p_o - p_w$ en la ecuación (2.31) se obtiene

$$\begin{aligned}
-\nabla\cdot\lambda\underline{\underline{k}}\Big[f_w\nabla(p_o-p_c)+f_o\nabla p_o-(f_w\rho_w+f_o\rho_o)\underline{g}\Big]-(q_w+q_o) &= 0,\\
\Longrightarrow -\nabla\cdot\lambda\underline{\underline{k}}\Big[\underbrace{(f_w+f_o)}_{1}\nabla p_o-f_w\nabla p_c-(f_w\rho_w+f_o\rho_o)\underline{g}\Big]-(q_w+q_o) &= 0,\\
\Longrightarrow -\nabla\cdot\big(\underline{\underline{k}}\lambda\nabla p_o\big)+\nabla\cdot\big(\underline{\underline{k}}\lambda_w\nabla p_c\big)+\nabla\cdot\big(\underline{\underline{k}}(\lambda_w\rho_w+\lambda_o\rho_o)\underline{g}\big)-(q_w+q_o) &= 0.
\end{aligned}$$

En muchas ocasiones, la presión capilar depende de la saturación del agua, de tal manera que el gradiente de la presión capilar se puede expresar como

$$\nabla p_c(S_w)=\frac{dp_c}{dS_w}\nabla S_w. \tag{2.32}$$

Con esta forma para la presión capilar, se obtiene la siguiente ecuación para la presión

$$-\nabla\cdot\big(\underline{\underline{k}}\lambda\nabla p_o\big)+\nabla\cdot\left(\underline{\underline{k}}\lambda_w\frac{dp_c}{dS_w}\nabla S_w\right)+\nabla\cdot\big(\underline{\underline{k}}(\lambda_w\rho_w+\lambda_o\rho_o)\underline{g}\big)=q_w+q_o. \tag{2.33}$$

Para obtener una ecuación para la saturación se parte de la ecuación (2.29). Tomando en cuenta la incompresibilidad de las fases y que la porosidad es constante se tiene que

$$\begin{aligned}
\phi\frac{\partial S_w}{\partial t}+\nabla\cdot\underline{u}_w &= q_w,\\
\Longrightarrow \phi\frac{\partial S_w}{\partial t}-\nabla\cdot\big(\lambda_w\underline{\underline{k}}(\nabla p_w-\rho_w\underline{g})\big) &= q_w,\\
\Longrightarrow \phi\frac{\partial S_w}{\partial t}-\nabla\cdot\big(\lambda_w\underline{\underline{k}}(\nabla p_o-p_c)-\rho_w\underline{g}\big) &= q_w,\\
\Longrightarrow \phi\frac{\partial S_w}{\partial t}-\nabla\cdot\big(\underline{\underline{k}}\lambda_w\nabla p_o\big)+\nabla\cdot\left(\underline{\underline{k}}\lambda_w\frac{dp_c}{dS_w}\nabla S_w\right)+\nabla\cdot\big(\underline{\underline{k}}\lambda_w\rho_w\underline{g}\big) &= q_w.
\end{aligned} \tag{2.34}$$

En los casos de estudio que se tratarán aquí, no se toman en cuenta los efectos de la fuerza de gravedad, por lo tanto, la ecuación de presión (2.33) y la ecuación de saturación (2.34) se transforman como sigue

$$\nabla\cdot\left(-\underline{\underline{k}}\lambda\nabla p_o+\underline{\underline{k}}\lambda_w\frac{dp_c}{dS_w}\nabla S_w\right)=q_w+q_o \tag{2.35}$$

y

$$\phi\frac{\partial S_w}{\partial t} + \nabla\cdot\left(-\underline{\underline{k}}\lambda_w\nabla p_o + \underline{\underline{k}}\lambda_w\frac{dp_c}{dS_w}\nabla S_w\right) = q_w. \tag{2.36}$$

Estas dos últimas ecuaciones son las que se usarán en el caso de las subsección siguiente.

2.3.1. Caso: Buckley-Leverett

En este problema se considera un medio tridimensional homogéneo de longitud L, con una sección transversal **a**, inicialmente saturado con aceite [Buckley41]. Se inyecta agua a una razón de flujo constante en el extremo izquierdo del dominio, la cual desplazará el aceite hacia el extremo derecho donde la presión se mantiene constante, véase figura 2.2.

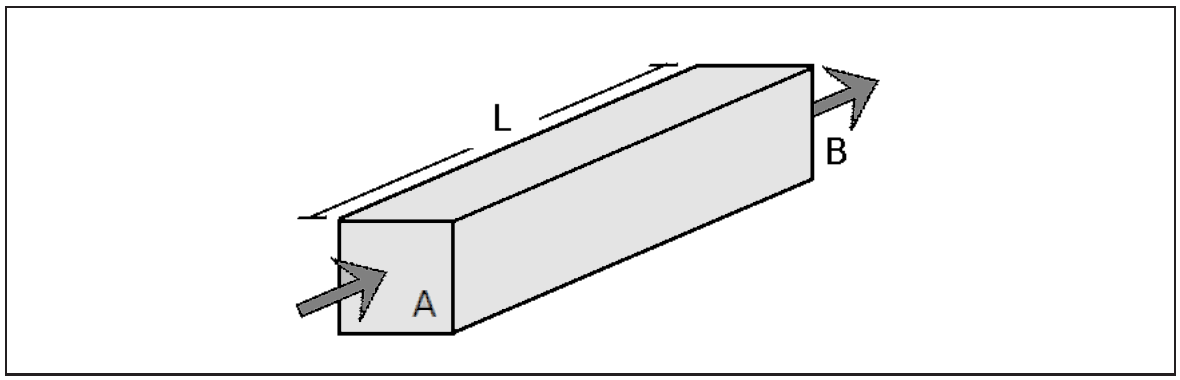

Figura 2.2: Dominio de estudio y condiciones de frontera. Inyección en la cara A y extracción por la cara B

Para este problema se considera además que $p_c \equiv q_w \equiv q_o \equiv 0$. Entonces las ecuaciones (2.35) y (2.36) se transforman como sigue (usando $p_o \equiv p$ y $S_w \equiv S$)

$$-\nabla\cdot\left(\underline{\underline{k}}\lambda\nabla p\right) = 0 \tag{2.37}$$

y

$$\phi\frac{\partial S}{\partial t} - \nabla\cdot\left(\underline{\underline{k}}\lambda_w\nabla p\right) = 0. \tag{2.38}$$

Las condiciones iniciales son las siguientes

$$p(t_0) = p_{t_0} \text{ Pa}; \quad S(t_0) = 0,$$

mientras que las condiciones de frontera se expresan como sigue

$$(S^{in})_A = s_0; \qquad (g_p^{in})_A = g_A \text{ m/s};$$
$$(S^{out})_B = 0; \qquad (p^{out})_B = p_{out} \text{ Pa}.$$

Nótese que las condiciones para la saturación son de tipo Dirichlet, mientras que para la presión se tiene una condición Dirichlet a la salida, y una condición tipo Neumann a la entrada dada en términos de la velocidad de inyección.

Para este caso se usarán las siguientes relaciones constitutivas:

$$\text{Permeabilidad relativa}: \quad k_{rw} = S_e^{\sigma}; k_{ro} = (1 - S_e)^{\sigma}. \tag{2.39}$$

$$\text{Saturación efectiva}: \quad S_{ef} = \frac{S - S_{rw}}{1 - S_{rw} - S_{ro}}. \tag{2.40}$$

En (2.39) el exponente σ puede ser 1 o 2 para los casos lineal y cuadrático respectivamente. S_{rw} y S_{ro} son las saturaciones residuales del agua y del aceite respectivamente.

El modelo de Buckley-Leverett presentado en esta subsección ha sido estudiado por varios autores, véase [Buckley41, Welge52, Lake96, Wolff08]. Es bien conocido que este problema presenta discontinuidades, conocidas también como choques. El tratamiento de este tipo de problemas se puede realizar de varias maneras.

Observando la ecuación 2.38 se puede ver que es de tipo hiperbólica, véase [Leveque80]. Esto significa que dicha ecuación, como se dijo antes presenta choques. Una manera de tratar estos choques es la siguiente. Dado que la ecuación es continua lejos de los choques, un enfoque para obtener la solución numérica es combinar el MVF en las regiones suaves de la solución, con algún procedimiento explícito para seguir los choques o discontinuidades que se puedan presentar. Este tratamiento numérico es similar a la forma matemática en donde las EDPs se completan con condiciones de salto en las discontinuidades, véase [Herrera10],[Herrera91],[Datta-Gupta07]. Este enfoque es conocido como *"shock tracking"*

o *"front tracking"*. Cuando se tiene más de una dimension, las discontinuidades típicamente caen en curvas, en 2D, o en superficies en 3D, y los algoritmos son bastante complicados. Además, en problemas realistas, pueden existir muchas superficies que interactuan de manera muy complicada conforme el tiempo evoluciona. Estos métodos no serán discutidos en este trabajo. Para ejemplos y discusión acerca de estos temas véase [Herrera91],[Chen07],[Leonard90],

En este trabajo el enfoque se basará en métodos conocidos como *"shock-capturing"*, donde el objetivo es capturar las discontinuidades en la solución de manera automática, sin usar estrategias explícitas para seguirlas. Las discontinuidades serán suavizadas sobre una o más celdas de la malla. El éxito de esta estrategia requiere que el método incorpore correcta e implícitamente las condiciones de salto, que reduzca la difusión numérica al mínimo y no introducir oscilaciones no realistas cerca de las discontinuidades.

Los esquemas numéricos clásicos tales como Upwind de primer orden, upstream de alto orden, Lax-Friedrichs, Lax-Wendroff y Warming-Beam [Leveque80], son conocidos como métodos de diferencias finitas por que las derivadas se aproximan usando diferencias de valores discretos. Es bien conocido que los métodos de alto orden ocasionan oscilaciones en los choques, donde los gradientes son altos, mientras que los de bajo orden producen difusión numérica.

Por ejemplo, el esquema Upwind para una dimensión se puede escribir como sigue (véase [Leonard90]):

Considerese la ecuación de convección-difusión:

$$\frac{\partial \Phi}{\partial t} = -u\frac{\partial \Phi}{\partial x} + D\frac{\partial^2 \Phi}{\partial x^2}, \tag{2.41}$$

para u, D positivas. Usando diferencias centrales para el espacio, y hacia adelante en el tiempo para discretizar la ecuación 2.41, y suponiendo una malla uniforme, tenemos:

$$\frac{\Phi^{n+1}}{\Delta t} = -u\frac{\Phi^n_{i+1} - \Phi^n_{i-1}}{2\Delta x} + D\frac{\Phi^n_{i+1} - 2\Phi^n_i + \Phi^n_{i-1}}{\Delta x^2}. \tag{2.42}$$

La ecuación anterior puede ser rearreglada de la siguiente manera:

$$\frac{\Phi^{n+1}}{\Delta t} = \Phi^n_i - \frac{c}{2}(\Phi^n_{i+1} - \Phi^n_{i-1}) + \frac{|c|}{P_\Delta}(\Phi^n_{i+1} - 2\Phi^n_i + \Phi^n_{i-1}), \tag{2.43}$$

con $c = u\Delta t/\Delta x$, $P_\Delta = |u|\Delta x/D$ los números de Courant y de celda de Peclet respectivamente.

Sin embargo, la ecuación 2.41 también puede ser discretizada despreciando el término difusivo, y utilizando el esquema Upwind de primer orden,

1. para $u > 0$
$$\frac{\Phi^{n+1}}{\Delta t} = \Phi_i^n - c(\Phi_i^n - \Phi_{i-1}^n), \tag{2.44}$$
similar a diferencias finitas hacia atrás.

2. para $u < 0$
$$\frac{\Phi^{n+1}}{\Delta t} = \Phi_i^n - c(\Phi_{i+1}^n - \Phi_i^n), \tag{2.45}$$
similar a diferencias finitas hacia delante.

Observemos que la ecuación 2.44, puede ser rearreglada de la siguiente manera,

$$\frac{\Phi^{n+1}}{\Delta t} = \Phi_i^n - \frac{c}{2}(\Phi_{i+1}^n - \Phi_{i-1}^n) + \frac{|c|}{P_\Delta^*}(\Phi_{i+1}^n - 2\Phi_i^n + \Phi_{i-1}^n), \tag{2.46}$$

con (P_Δ^*), la cual es una forma similar a la de la ecuación 2.43. Por lo tanto, al emplear el esquema Upwind de primer orden y los choques son atenuados por la presencia de difusión numérica (véase el segundo término de la ecuación).

Como se ha mencionado, existe una vasta cantidad de técnicas numéricas las cuales se pueden usar para aproximar el tipo de ecuaciones del modelo de Buckley-Leverett. Sin embargo, en esta tesis, dado que el objetivo principal es la paralelización de los métodos de solución de los sistema de ecuaciones, solo se usará el esquema Upwind de primer orden, pues otros esquemas pueden ser incorporados posteriormente sin cambiar sustancialmente los resultados de la paralelizacion. La implementación usando el MVF correspondiente al esquema Upwind se describe en la siguiente sección.

2.4. Modelo numérico

La solución numérica de las ecuaciones (2.35) y (2.36) se obtendrá aplicando el método de volumen finito estándar [Patankar92, Malalasekera95].

2.4.1. Método de Volumen Finito

El método de volumen finito (MVF) se deriva a partir de la forma conservativa de las ecuaciones de balance. En este método, el dominio de estudio se divide en un número de volúmenes de control que no se traslapan, de tal manera que hay un volumen rodeando a cada punto de la malla, véase figura 2.3. Luego, se integra la ecuación de balance (2.5) sobre cada volumen, lo cual es equivalente a aplicar la ecuación de balance (2.4) tomando a $B(t)$ igual a cada volumen de control. Los flujos a través de las caras del volumen de control se aproximan usando esquemas numéricos apropiados, y esto da como resultado un conjunto de ecuaciones discretas, una para cada volumen de control, las cuales deben resolverse para obtener una solución numérica aproximada

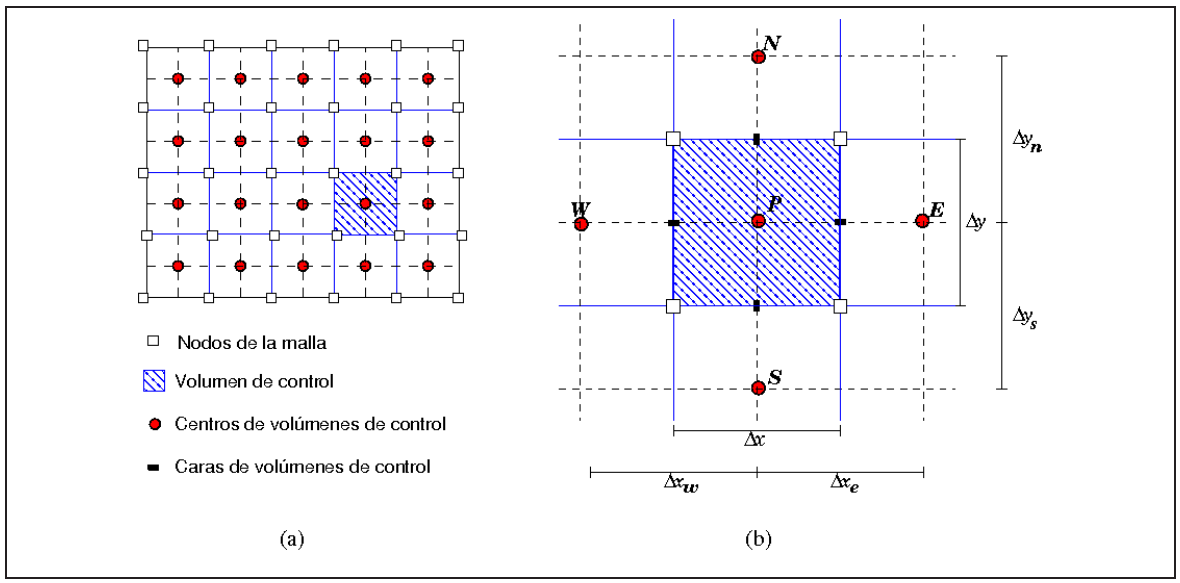

Figura 2.3: (a) Dominio de estudio discretizado usando volúmenes de control en 2D. (b) Volumen de control y sus vecinos en 2D.

Las ecuaciones discretas que resultan usando esta estrategia expresan el principio de conservación para la propiedad extensiva correspondiente, en cada volumen de control, de la misma forma en que la ecuación diferencial expresa el mismo principio para un volumen de control infinitesimal. Esta característica es válida para cualquier número de volúmenes sobre la malla y no solamente para un número grande de ellos. Por lo tanto, aún una solución en una malla gruesa exhibirá un balance exacto, aunque para obtener buena precisión se

requiere de un número grande de volúmenes o esquemas de alto orden. Esta clara relación entre el algoritmo numérico y el principio físico de conservación es una de las mayores atracciones del MVF.

Por comodidad, en las secciones que siguen se usará la notación $p_o \equiv p$ y $S_w \equiv S$. Además se supondrá que el tensor de permeabilidad $\underline{\underline{k}}$ tiene la forma

$$\underline{\underline{k}} = \begin{pmatrix} k_{11} & 0 & 0 \\ 0 & k_{22} & 0 \\ 0 & 0 & k_{33} \end{pmatrix} \tag{2.47}$$

donde k_{11}, k_{22} y k_{33} son las permeabilidades en las direcciones x, y y z, las cuales se consideran constantes.

2.4.2. Discretización de la ecuación de presión

Para discretizar la ecuación de presión (2.35), se define la siguiente *"función de flujo"*:

$$\underline{\mathcal{F}} = -\underline{\underline{k}}\lambda\nabla p + \underline{\underline{k}}\lambda_w \frac{dp_c}{dS}\nabla S \tag{2.48}$$

Con esta definición las componentes de $\underline{\mathcal{F}}$ se escriben como sigue

$$\begin{aligned} \mathcal{F}_x &= -k_{11}\left(\lambda\frac{\partial p}{\partial x} - \lambda_w\frac{dp_c}{dS}\frac{\partial S}{\partial x}\right) \\ \mathcal{F}_y &= -k_{22}\left(\lambda\frac{\partial p}{\partial y} - \lambda_w\frac{dp_c}{dS}\frac{\partial S}{\partial y}\right) \\ \mathcal{F}_z &= -k_{33}\left(\lambda\frac{\partial p}{\partial z} - \lambda_w\frac{dp_c}{dS}\frac{\partial S}{\partial z}\right) \end{aligned} \tag{2.49}$$

De esta manera, la ecuación (2.35) se transforma en

$$\nabla\cdot\underline{\mathcal{F}} = \frac{\partial\mathcal{F}_x}{\partial x} + \frac{\partial\mathcal{F}_y}{\partial y} + \frac{\partial\mathcal{F}_z}{\partial z} = q_w + q_o \tag{2.50}$$

En términos de la ecuación de balance (2.4) la ecuación anterior se puede escribir como

$$\int_{\Delta V} \nabla \cdot \underline{\mathcal{F}} dV = \int_{\Delta V} \left(\frac{\partial \mathcal{F}_x}{\partial x} + \frac{\partial \mathcal{F}_y}{\partial y} + \frac{\partial \mathcal{F}_z}{\partial z} \right) dV = \int_{\Delta V} (q_w + q_o) dV \tag{2.51}$$

donde $dV = dxdydz$ y la integración se hace sobre un volumen de control como el que se muestra en la figura 2.4, es decir, en este caso $B(t) \equiv \Delta V$.

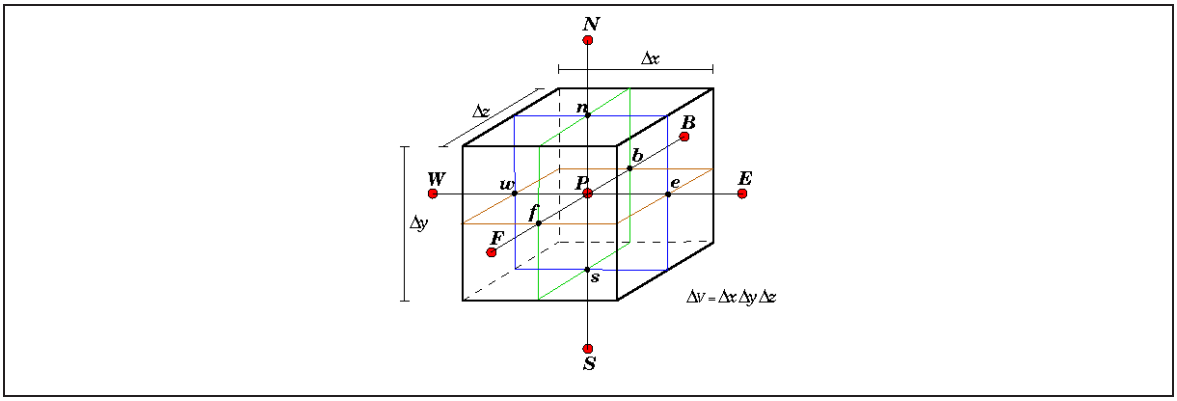

Figura 2.4: Volumen de control en 3D alrededor del punto P. Los volúmenes vecinos están representados por $\boldsymbol{E}$, $\boldsymbol{W}$, $\boldsymbol{N}$, $\boldsymbol{S}$, $\boldsymbol{F}$, $\boldsymbol{B}$. Las caras del volumen de control se etiquetan con $\boldsymbol{e}$, $\boldsymbol{w}$, $\boldsymbol{n}$, $\boldsymbol{s}$, $\boldsymbol{f}$, $\boldsymbol{b}$.

Nótese que el volumen representado en la figura 2.4 es un hexaedro cuyos ejes son paralelos a los ejes coordenados de un sistema Cartesiano. Esta forma de los volúmenes simplifica la aproximación de las integrales en la ecuación (2.51), aunque limita el tipo de dominos a estudiar, ya que no es adaptable.

Usando como referencia el volumen de control de la figura 2.4, las integrales de la ecuación (2.51) se calculan como sigue

$$\int_{\boldsymbol{b}}^{\boldsymbol{f}} \int_{\boldsymbol{s}}^{\boldsymbol{n}} \int_{\boldsymbol{w}}^{\boldsymbol{e}} \frac{\partial \mathcal{F}_x}{\partial x} dxdydz = \Big((\mathcal{F}_x)_{\boldsymbol{e}} - (\mathcal{F}_x)_{\boldsymbol{w}} \Big) A_x \tag{2.52}$$

donde $A_x = \Delta y \Delta z$ representa el área de las caras del volumen de control paralelas al plano yz. La notación $(\mathcal{F}_x)_{\boldsymbol{e}}$ significa que $\mathcal{F}_x$ se debe evaluar en la cara $\boldsymbol{e}$ del volumen de control.

Realizando el mismo tratamiento para los dos términos restantes de la integral de la izquierda de la ecuación (2.51) se obtiene

$$\Big((\mathcal{F}_x)_{\boldsymbol{e}} - (\mathcal{F}_x)_{\boldsymbol{w}} \Big) A_x + \Big((\mathcal{F}_y)_{\boldsymbol{n}} - (\mathcal{F}_y)_{\boldsymbol{s}} \Big) A_y + \Big((\mathcal{F}_z)_{\boldsymbol{f}} - (\mathcal{F}_z)_{\boldsymbol{b}} \Big) A_z = (\bar{q}_w + \bar{q}_o) \Delta V \tag{2.53}$$

donde $A_y = \Delta x \Delta z$, $A_z = \Delta x \Delta y$ y $\Delta V = \Delta x \Delta y \Delta z$. En esta última ecuación $\bar{q}_w$ y $\bar{q}_o$ representan un promedio de q_w y q_o dentro del volumen de control, respectivamente.

La evaluación de $\mathcal{F}_x$ en la cara $\boldsymbol{e}$ se hace como sigue

$$\begin{aligned}(\mathcal{F}_x)_{\boldsymbol{e}} &= -k_{11}\left(\lambda\frac{\partial p}{\partial x} - \lambda_w\frac{dp_c}{dS}\frac{\partial S}{\partial x}\right)_{\boldsymbol{e}} \\ &= -k_{11}\left((\lambda)_{\boldsymbol{e}}\left(\frac{\partial p}{\partial x}\right)_{\boldsymbol{e}} - \left(\lambda_w\frac{dp_c}{dS}\right)_{\boldsymbol{e}}\left(\frac{\partial S}{\partial x}\right)_{\boldsymbol{e}}\right) \\ &= -k_{11}\left((\lambda)_{\boldsymbol{e}}\frac{p_{\boldsymbol{E}} - p_{\boldsymbol{P}}}{\Delta x_{\boldsymbol{e}}} - \left(\lambda_w\frac{dp_c}{dS}\right)_{\boldsymbol{e}}\frac{S_{\boldsymbol{E}} - S_{\boldsymbol{P}}}{\Delta x_{\boldsymbol{e}}}\right)\end{aligned}$$

donde las derivadas parciales se han aproximado usando diferencias centrales. Los subíndices $\boldsymbol{E}$ y $\boldsymbol{P}$ indican que la variable se evalúa en el centro del volumen de control correspondiente. La evaluación de $\mathcal{F}_x$ en la cara $\boldsymbol{w}$ se hace de forma similar, de tal manera que el primer término de la ecuación (2.53) queda como sigue

$$\begin{aligned}\Big((\mathcal{F}_x)_{\boldsymbol{e}} - (\mathcal{F}_x)_{\boldsymbol{w}}\Big)A_x &= -k_{11}\left((\lambda)_{\boldsymbol{e}}\frac{p_{\boldsymbol{E}} - p_{\boldsymbol{P}}}{\Delta x_{\boldsymbol{e}}} - \left(\lambda_w\frac{dp_c}{dS}\right)_{\boldsymbol{e}}\frac{S_{\boldsymbol{E}} - S_{\boldsymbol{P}}}{\Delta x_{\boldsymbol{e}}}\right)A_x + \\ &\quad k_{11}\left((\lambda)_{\boldsymbol{w}}\frac{p_{\boldsymbol{P}} - p_{\boldsymbol{W}}}{\Delta x_{\boldsymbol{w}}} - \left(\lambda_w\frac{dp_c}{dS}\right)_{\boldsymbol{w}}\frac{S_{\boldsymbol{P}} - S_{\boldsymbol{W}}}{\Delta x_{\boldsymbol{w}}}\right)A_x\end{aligned}$$

donde $\Delta x_{\boldsymbol{w}} = x_{\boldsymbol{P}} - x_{\boldsymbol{W}}$ y $\Delta x_{\boldsymbol{e}} = x_{\boldsymbol{E}} - x_{\boldsymbol{P}}$, véase figura 2.3.

Las componentes $\mathcal{F}_y$ y $\mathcal{F}_z$ se evalúan de la misma manera en las caras correspondientes. Sustituyendo estas evaluaciones en la ecuación (2.53) y rearreglando se obtiene la siguiente ecuación discreta para el volumen de control $\boldsymbol{P}$

$$\boldsymbol{a}_{\boldsymbol{P}}p_{\boldsymbol{P}} = \boldsymbol{a}_{\boldsymbol{E}}p_{\boldsymbol{E}} + \boldsymbol{a}_{\boldsymbol{W}}p_{\boldsymbol{W}} + \boldsymbol{a}_{\boldsymbol{N}}p_{\boldsymbol{N}} + \boldsymbol{a}_{\boldsymbol{S}}p_{\boldsymbol{S}} + \boldsymbol{a}_{\boldsymbol{F}}p_{\boldsymbol{F}} + \boldsymbol{a}_{\boldsymbol{B}}p_{\boldsymbol{B}} + q_{\boldsymbol{P}} \tag{2.54}$$

donde los coeficientes están definidos como sigue

$$\boldsymbol{a_E} = \frac{k_{11}(\lambda)_{\boldsymbol{e}} A_x}{\Delta x_{\boldsymbol{e}}}; \qquad \boldsymbol{a_W} = \frac{k_{11}(\lambda)_{\boldsymbol{w}} A_x}{\Delta x_{\boldsymbol{w}}}; \qquad \boldsymbol{a_N} = \frac{k_{22}(\lambda)_{\boldsymbol{n}} A_y}{\Delta y_{\boldsymbol{n}}};$$

$$\boldsymbol{a_S} = \frac{k_{22}(\lambda)_{\boldsymbol{s}} A_y}{\Delta y_{\boldsymbol{s}}}; \qquad \boldsymbol{a_F} = \frac{k_{33}(\lambda)_{\boldsymbol{f}} A_z}{\Delta z_{\boldsymbol{f}}}; \qquad \boldsymbol{a_B} = \frac{k_{33}(\lambda)_{\boldsymbol{b}} A_z}{\Delta z_{\boldsymbol{b}}}; \tag{2.55}$$

$$\boldsymbol{a_P} = \boldsymbol{a_E} + \boldsymbol{a_W} + \boldsymbol{a_N} + \boldsymbol{a_S} + \boldsymbol{a_F} + \boldsymbol{a_B}$$

$$\begin{aligned} q_{\boldsymbol{P}} \;=\; & \Big((L)_{\boldsymbol{e}} + (L)_{\boldsymbol{w}} + (L)_{\boldsymbol{n}} + (L)_{\boldsymbol{s}} + (L)_{\boldsymbol{f}} + (L)_{\boldsymbol{b}}\Big) S_{\boldsymbol{P}} - \\ & \Big((L)_{\boldsymbol{e}} S_{\boldsymbol{E}} + (L)_{\boldsymbol{w}} S_{\boldsymbol{W}} + (L)_{\boldsymbol{n}} S_{\boldsymbol{N}} + (L)_{\boldsymbol{s}} S_{\boldsymbol{S}} + (L)_{\boldsymbol{f}} S_{\boldsymbol{F}} + (L)_{\boldsymbol{b}} S_{\boldsymbol{B}}\Big) + \\ & (\bar{q}_w + \bar{q}_o)\Delta V \end{aligned} \tag{2.56}$$

$$(L)_{\boldsymbol{e}} = k_{11}\left(\lambda_w \frac{dp_c}{dS}\right)_{\boldsymbol{e}} \frac{A_x}{\Delta x_{\boldsymbol{e}}}; \quad (L)_{\boldsymbol{w}} = k_{11}\left(\lambda_w \frac{dp_c}{dS}\right)_{\boldsymbol{w}} \frac{A_x}{\Delta x_{\boldsymbol{w}}};$$

$$(L)_{\boldsymbol{n}} = k_{22}\left(\lambda_w \frac{dp_c}{dS}\right)_{\boldsymbol{n}} \frac{A_y}{\Delta y_{\boldsymbol{n}}}; \quad (L)_{\boldsymbol{s}} = k_{22}\left(\lambda_w \frac{dp_c}{dS}\right)_{\boldsymbol{s}} \frac{A_y}{\Delta y_{\boldsymbol{s}}}; \tag{2.57}$$

$$(L)_{\boldsymbol{f}} = k_{33}\left(\lambda_w \frac{dp_c}{dS}\right)_{\boldsymbol{f}} \frac{A_z}{\Delta z_{\boldsymbol{f}}}; \quad (L)_{\boldsymbol{b}} = k_{33}\left(\lambda_w \frac{dp_c}{dS}\right)_{\boldsymbol{b}} \frac{A_z}{\Delta z_{\boldsymbol{b}}};$$

Nótese que los coeficientes antes descritos dependen de la movilidad total y de la del agua, las cuales en general dependen de la saturación. En este escrito, se usarán valores de la saturación en un paso anterior al actual, con lo que la ecuación (2.54) queda desacoplada de la saturación y linealizada.

Escribiendo las ecuaciones discretas para todos los volúmenes de la malla, se obtiene una sistema lineal de ecuaciones el cual consta de siete diagonales para el caso tridimensional. Además, dada la forma de los coeficientes, el sistema es diagonal dominante. Para resolver este sistema y obtener la presión en todos los puntos de la malla se usará un algoritmo iterativo.

2.4.3. Discretización de la ecuación de saturación

La ecuación de saturación (2.36), a diferencia de la ecuación de presión, será resuelta de manera explícita. Para discretizar esta ecuación se define la siguiente *"función de flujo"*:

$$\underline{\mathcal{F}} = -\underline{\underline{k}}\lambda_w \nabla p + \underline{\underline{k}}\lambda_w \frac{dp_c}{dS}\nabla S \tag{2.58}$$

Nótese que la única diferencia con la ecuación (2.48) es que aquí aparece la movilidad del agua λ_w en lugar de la movilidad total en el primer término. Todo el tratamiento de las componentes de $\underline{\mathcal{F}}$ en este caso, es similar a como se hizo para la ecuación de presión.

Usando la definición (2.58), la ecuación (2.36) se transforma en

$$\phi\frac{\partial S}{\partial t} + \nabla \cdot \underline{\mathcal{F}} = \phi\frac{\partial S}{\partial t} + \frac{\partial \mathcal{F}_x}{\partial x} + \frac{\partial \mathcal{F}_y}{\partial y} + \frac{\partial \mathcal{F}_z}{\partial z} = q_w \tag{2.59}$$

En términos de la ecuación de balance (2.4), la ecuación anterior se puede escribir como

$$\frac{\partial}{\partial t}\int\limits_{\Delta V}(\phi S)dV + \int\limits_{\Delta V}\left(\frac{\partial \mathcal{F}_x}{\partial x} + \frac{\partial \mathcal{F}_y}{\partial y} + \frac{\partial \mathcal{F}_z}{\partial z}\right)dV = \int\limits_{\Delta V} q_w dV \tag{2.60}$$

donde ΔV es el volumen de control representado en la figura 2.4, y nuevamente se tiene que $B(t) \equiv \Delta V$. En esta ecuación se usó el hecho de que la densidad de las fases y la porosidad son constantes.

La forma de manejar derivadas con respecto al tiempo en el MVF, es mediante una integración de toda la ecuación diferencial en el intervalo $[t, t+\Delta t]$. Entonces, integrando la ecuación (2.60) en dicho intervalo, y denotando con n el instante de tiempo t y con $n+1$ el instante de tiempo $t+\Delta t$, se obtiene

$$\int\limits_{n}^{n+1}\left[\frac{\partial}{\partial t}\int\limits_{\Delta V}(\phi S)dV + \int\limits_{\Delta V}\left(\frac{\partial \mathcal{F}_x}{\partial x} + \frac{\partial \mathcal{F}_y}{\partial y} + \frac{\partial \mathcal{F}_z}{\partial z} - q_w\right)dV\right]dt = 0 \tag{2.61}$$

El primer término de esta última ecuación se aproxima como sigue

$$\int_n^{n+1}\left[\frac{\partial}{\partial t}\int_{\Delta V}(\phi S)dV\right]dt = \int_{\Delta V}\int_n^{n+1}\phi\frac{\partial S}{\partial t}dtdV$$
$$= \phi\int_{\Delta V}\Big(S^{n+1}-S^n\Big)dV$$

Para aproximar esta última integral se necesita conocer la forma de S como función de $\underline{x}$. Es posible usar funciones lineales o de más alto orden, sin embargo en MVF se supone la función es constante dentro del volumen de control ΔV, es decir se toma el valor del centro $S_{\boldsymbol{P}}$, por lo tanto se tiene lo siguiente

$$\int_n^{n+1}\left[\frac{\partial}{\partial t}\int_{\Delta V}(\phi S)dV\right]dt = \phi\Big(S_{\boldsymbol{P}}^{n+1}-S_{\boldsymbol{P}}^n\Big)\Delta V \tag{2.62}$$

El segundo término de la ecuación (2.61) se aproxima de manera similar a como se hizo para la ecuación de presión, en este caso se obtiene lo siguiente

$$\int_n^{n+1}\int_{\Delta V}\left(\nabla\cdot\underline{\mathcal{F}}-q_w\right)dVdt = \int_n^{n+1}\Big(\boldsymbol{a}_{\boldsymbol{P}}^w p_{\boldsymbol{P}} - \big(\boldsymbol{a}_{\boldsymbol{E}}^w p_{\boldsymbol{E}}+\boldsymbol{a}_{\boldsymbol{W}}^w p_{\boldsymbol{W}}+\boldsymbol{a}_{\boldsymbol{N}}^w p_{\boldsymbol{N}}+$$
$$\boldsymbol{a}_{\boldsymbol{S}}^w p_{\boldsymbol{S}}+\boldsymbol{a}_{\boldsymbol{F}}^w p_{\boldsymbol{F}}+\boldsymbol{a}_{\boldsymbol{B}}^w p_{\boldsymbol{B}}+q_{\boldsymbol{P}}\big)\Big)dt \tag{2.63}$$

donde los coeficientes $\boldsymbol{a}^w$ son similares a los coeficientes $\boldsymbol{a}$, ecuaciones (2.55), excepto que se usa la movilidad del agua λ_w en lugar de la movilidad total λ (el superíndice w indica este hecho). También, el coeficiente $q_{\boldsymbol{P}}$ es similar al definido en (2.56) excepto que en el último término solo se tiene a la fuente del agua $\bar{q}_w$.

La integral integral temporal se aproxima usando el esquema θ

$$\int_n^{n+1} f dt = \left[\theta f^{n+1}+(1-\theta)f^n\right]\Delta t,$$

donde para $\theta = 0$ se tiene un esquema explícito ($f^n\Delta t$), para $\theta = 1$ se tiene un esquema implícito ($f^{n+1}\Delta t$), y para $\theta = \frac{1}{2}$ se tiene el esquema conocido como de Crank-Nicolson ($\left[f^{n+1}+f^n\right]\frac{\Delta t}{2}$), véase [Crank96]. En el método IMPES la ecuación de saturación se resuelve explícitamente, por lo que la ecuación (2.63) se transforma en

$$\int_{n}^{n+1}\int_{\Delta V}(\nabla\cdot\underline{\mathcal{F}}-q_w)\,dV\,dt = \Big(\boldsymbol{a}_{P}^{w,n}p_{P}^{n}-\big(\boldsymbol{a}_{E}^{w,n}p_{E}^{n}+\boldsymbol{a}_{W}^{w,n}p_{W}^{n}+\boldsymbol{a}_{N}^{w,n}p_{N}^{n}+ \boldsymbol{a}_{S}^{w,n}p_{S}^{n}+\boldsymbol{a}_{F}^{w,n}p_{F}^{n}+\boldsymbol{a}_{B}^{w,n}p_{B}^{n}+q_{P}^{n}\big)\Big)\Delta t \tag{2.64}$$

donde el superíndice n indica que los coeficientes se calculan en el instante n.

Sustituyendo (2.62) y (2.64) en (2.61), y rearreglando se obtiene la siguiente relación explícita para la saturación

$$S_{P}^{n+1}=S_{P}^{n}-\boldsymbol{b}_{P}^{n}p_{P}^{n}+\boldsymbol{b}_{E}^{n}p_{E}^{n}+\boldsymbol{b}_{W}^{n}p_{W}^{n}+\boldsymbol{b}_{N}^{n}p_{N}^{n}+\boldsymbol{b}_{S}^{n}p_{S}^{n}+\boldsymbol{b}_{F}^{n}p_{F}^{n}+\boldsymbol{b}_{B}^{n}p_{B}^{n}+q_{P}^{n} \tag{2.65}$$

donde los coeficientes están definidos como

$$\boldsymbol{b}_{E}^{n}=\frac{k_{11}(\lambda_w)_{\boldsymbol{e}}^{n}}{\phi\Delta x_{\boldsymbol{e}}\Delta x};\qquad \boldsymbol{b}_{W}^{n}=\frac{k_{11}(\lambda_w)_{\boldsymbol{w}}^{n}}{\phi\Delta x_{\boldsymbol{w}}\Delta x};\qquad \boldsymbol{b}_{N}^{n}=\frac{k_{22}(\lambda_w)_{\boldsymbol{n}}^{n}}{\phi\Delta y_{\boldsymbol{n}}\Delta y};$$

$$\boldsymbol{b}_{S}^{n}=\frac{k_{22}(\lambda_w)_{\boldsymbol{s}}^{n}}{\phi\Delta y_{\boldsymbol{s}}\Delta y};\qquad \boldsymbol{b}_{F}^{n}=\frac{k_{33}(\lambda_w)_{\boldsymbol{f}}^{n}}{\phi\Delta z_{\boldsymbol{f}}\Delta z};\qquad \boldsymbol{b}_{B}^{n}=\frac{k_{33}(\lambda_w)_{\boldsymbol{b}}^{n}}{\phi\Delta z_{\boldsymbol{b}}\Delta z}; \tag{2.66}$$

$$\boldsymbol{b}_{P}^{n}=\boldsymbol{b}_{E}^{n}+\boldsymbol{b}_{W}^{n}+\boldsymbol{b}_{N}^{n}+\boldsymbol{b}_{S}^{n}+\boldsymbol{b}_{F}^{n}+\boldsymbol{b}_{B}^{n}$$

$$q_{P}^{n} = \Big((L)_{\boldsymbol{e}}^{n}+(L)_{\boldsymbol{w}}^{n}+(L)_{\boldsymbol{n}}^{n}+(L)_{\boldsymbol{s}}^{n}+(L)_{\boldsymbol{f}}^{n}+(L)_{\boldsymbol{b}}^{n}\Big)S_{P}^{n}- \Big((L)_{\boldsymbol{e}}^{n}S_{E}^{n}+(L)_{\boldsymbol{w}}^{n}S_{W}^{n}+(L)_{\boldsymbol{n}}^{n}S_{N}^{n}+(L)_{\boldsymbol{s}}^{n}S_{S}^{n}+(L)_{\boldsymbol{f}}^{n}S_{F}^{n}+(L)_{\boldsymbol{b}}^{n}S_{B}^{n}\Big)+ (\bar{q}_w)^{n}\Delta V \tag{2.67}$$

$$(L)_{\boldsymbol{e}}^{n}=\left(\lambda_w\frac{dp_c}{dS}\right)_{\boldsymbol{e}}^{n}\frac{k_{11}}{\phi\Delta x_{\boldsymbol{e}}\Delta x};\quad (L)_{\boldsymbol{w}}^{n}=\left(\lambda_w\frac{dp_c}{dS}\right)_{\boldsymbol{w}}^{n}\frac{k_{11}}{\phi\Delta x_{\boldsymbol{w}}\Delta x};$$

$$(L)_{\boldsymbol{n}}^{n}=\left(\lambda_w\frac{dp_c}{dS}\right)_{\boldsymbol{n}}^{n}\frac{k_{22}}{\phi\Delta y_{\boldsymbol{n}}\Delta x};\quad (L)_{\boldsymbol{s}}^{n}=\left(\lambda_w\frac{dp_c}{dS}\right)_{\boldsymbol{s}}^{n}\frac{k_{22}}{\phi\Delta y_{\boldsymbol{s}}\Delta x}; \tag{2.68}$$

$$(L)_{\boldsymbol{f}}^{n}=\left(\lambda_w\frac{dp_c}{dS}\right)_{\boldsymbol{f}}^{n}\frac{k_{33}}{\phi\Delta z_{\boldsymbol{f}}\Delta x};\quad (L)_{\boldsymbol{b}}^{n}=\left(\lambda_w\frac{dp_c}{dS}\right)_{\boldsymbol{b}}^{n}\frac{k_{33}}{\phi\Delta z_{\boldsymbol{b}}\Delta x};$$

Estos coeficientes dependen de la movilidad del agua, y en general de la saturación. Una ecuación del tipo (2.65) se deriva para cada volumen de control, sin embargo no se requiere de la solución de ningún sistema lineal, pues todos los términos de la derecha son conocidos.

2.4.4. IMPES

El problema representado por las ecuaciones (2.35) y (2.36) es no lineal y está fuertemente acoplado, por lo tanto se debe usar una estrategia de linealización. En la formulación presentada aquí, se usará el método IMPES (*IMplicit Pressure Explicit Saturation*), el cual se describe en el algoritmo 1. Este algoritmo será empleado para resolver los ejemplos de la sección 2.5.

Algoritmo 1 IMPES

1: Definir condiciones iniciales y de frontera del problema $S^0, p^0, T_{max}, \Delta t, \Delta x, \ldots$
2: **while** $t < T_{max}$ **do**
3: Calcular los coeficientes de la ecuación de presión usando (2.55), (2.56) y (2.57).
4: Resolver la ecuación de presión (2.54) de manera implícita usando un algoritmo iterativo.
5: Calcular los coeficientes de la ecuación de saturación usando (2.66), (2.67) y (2.68).
6: Resolver la ecuación de saturación (2.65) de manera explícita.
7: $t \leftarrow t + \Delta t$
8: **end while**

2.5. Ejemplo Buckley-Leverett

La ecuación discreta de la presión para este ejemplo es como la mostrada en (2.54) con $q_P = 0$. Los coeficientes (2.55) requieren de obtener el valor de λ en las caras del volumen de control $\boldsymbol{nb} = \boldsymbol{e}, \boldsymbol{w}, \boldsymbol{n}, \boldsymbol{s}, \boldsymbol{f}, \boldsymbol{b}$. Para encontrar estos valores se usan las relaciones (2.39) y (2.40) como sigue

$$\begin{aligned}
\lambda &= \lambda_w + \lambda_o = \frac{k_{rw}}{\mu_w} + \frac{k_{ro}}{\mu_o}, \\
\Rightarrow \lambda &= \frac{S_{ef}^{\sigma}}{\mu_w} + \frac{(1-S_{ef})^{\sigma}}{\mu_o} = \frac{1}{\mu_w}\left(\frac{S-S_{rw}}{1-S_{rw}-S_{ro}}\right)^{\sigma} + \frac{1}{\mu_o}\left(1-\frac{S-S_{rw}}{1-S_{rw}-S_{ro}}\right)^{\sigma}, \\
\Rightarrow \lambda &= \frac{1}{(1-S_{rw}-S_{ro})^{\sigma}}\left(\frac{(S-S_{rw})^{\sigma}}{\mu_w} + \frac{(1-S_{ro}-S)^{\sigma}}{\mu_o}\right).
\end{aligned}$$

Ahora, suponiendo que se conocen los valores de S en el tiempo anterior n, entonces, para calcular λ en el instante n en las caras del volumen de control se usa

$$(\lambda)_{\boldsymbol{nb}}^{n} = \frac{1}{(1-S_{rw}-S_{ro})^{\sigma}}\left(\frac{\left(S_{\boldsymbol{nb}}^{n}-S_{rw}\right)^{\sigma}}{\mu_w} + \frac{\left(1-S_{ro}-S_{\boldsymbol{nb}}^{n}\right)^{\sigma}}{\mu_o}\right). \tag{2.69}$$

La ecuación discreta de la saturación para este ejemplo es como la mostrada en (2.65) con $q_{\boldsymbol{P}}^{n} = 0$. Los coeficientes (2.66) requieren de obtener el valor de λ_w en las caras del volumen de control $\boldsymbol{nb} = \boldsymbol{e}, \boldsymbol{w}, \boldsymbol{n}, \boldsymbol{s}, \boldsymbol{f}, \boldsymbol{b}$. Para encontrar estos valores se usan las relaciones (2.39) y (2.40) como sigue

$$\lambda_w = \frac{k_{rw}}{\mu_w} = \frac{S_{ef}^{\sigma}}{\mu_w} = \frac{1}{\mu_w}\left(\frac{S-S_{rw}}{1-S_{rw}-S_{ro}}\right)^{\sigma}.$$

Entonces, el cálculo de λ_w en las caras en el tiempo n se hace con

$$(\lambda_w)_{\boldsymbol{nb}}^{n} = \frac{1}{\mu_w}\left(\frac{S_{\boldsymbol{nb}}^{n}-S_{rw}}{1-S_{rw}-S_{ro}}\right)^{\sigma}, \tag{2.70}$$

Cálculo de la saturación en las caras

En las ecuaciones (2.69) y (2.70) se observa que es necesario obtener los valores $S_{\boldsymbol{nb}}^{n}$, es decir la saturación en las caras de los volúmenes de control. Si se supone que el valor de S^n es conocido en todos los centros de los volúmenes, y estos valores provienen de la solución de la ecuación de saturación (2.38), entonces se requiere un esquema adecuado para aproximar el valor de S^n en las caras. Como se mencionó en la sección 2.3.1 existen diversos esquemas con los que se puede hacer esta aproximación. El utilizado en este trabajo fue el Upwind, el cual se explica a continuación:

Upwind: se hace una comparación entre los valores de la presión en los puntos vecinos a la cara del volumen de control donde se desea evaluar S^n y se le asigna el valor de donde la presión es mayor. Por ejemplo, para la cara $\boldsymbol{e}$ esto se hace como sigue

if $(p_{\boldsymbol{E}}^n >= p_{\boldsymbol{P}}^n)$ **then**
$S_{\boldsymbol{e}}^n = S_{\boldsymbol{E}}^n$
else
$S_{\boldsymbol{e}}^n = S_{\boldsymbol{P}}^n$
end if

Esta aproximación es de orden lineal y produce difusión numérica en la solución.

Este esquema, es estable, aunque presenta difusión numérica, de tal manera que para obtener un resultado con alta precisión se requiere de un número de volúmenes relativamente grande.

Condiciones de frontera para la presión

Para describir como se insertan las condiciones de frontera en la solución numérica de las ecuaciones se hará referencia a la figura 2.5, en donde se representan los volúmenes vecinos a las fronteras izquierda y derecha del dominio de estudio.

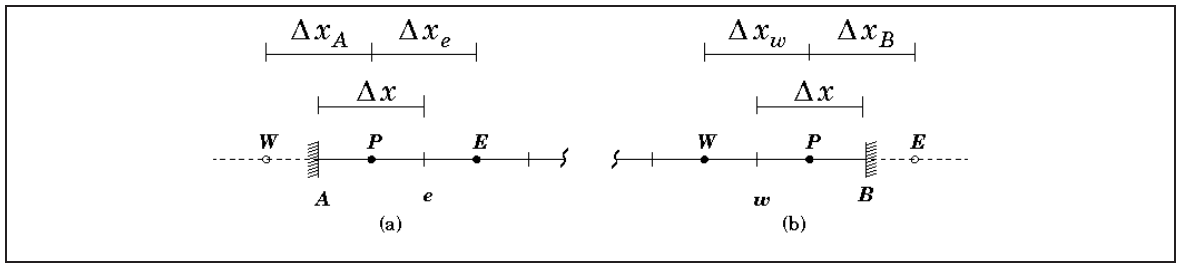

Figura 2.5: Volúmenes de control en las fronteras izquierda (a) y derecha (b).

La ecuación discreta de la presión para los volúmenes P junto a las fronteras, figura 2.5, es de la forma

$$\boldsymbol{a}_{\boldsymbol{P}}^n p_{\boldsymbol{P}}^{n+1} = \boldsymbol{a}_{\boldsymbol{E}}^n p_{\boldsymbol{E}}^{n+1} + \boldsymbol{a}_{\boldsymbol{W}}^n p_{\boldsymbol{W}}^{n+1} + \boldsymbol{a}_{\boldsymbol{N}}^n p_{\boldsymbol{N}}^{n+1} + \boldsymbol{a}_{\boldsymbol{S}}^n p_{\boldsymbol{S}}^{n+1} + \boldsymbol{a}_{\boldsymbol{F}}^n p_{\boldsymbol{F}}^{n+1} + \boldsymbol{a}_{\boldsymbol{B}}^n p_{\boldsymbol{B}}^{n+1}. \tag{2.71}$$

Los superíndices en esta ecuación indican el instante de tiempo en que se calcula cada cantidad.

Dado que se impone una razón de inyección constante en la frontera izquierda se tiene que

$$\begin{aligned}
-\underline{n}\cdot\underline{u} = -\underline{n}\cdot\left(-\lambda\underline{\underline{k}}\cdot(f_w\nabla p_w + f_o\nabla p_o - (f_w\rho_w + f_o\rho_o)\underline{g})\right) &= g_p^{in}\\
\Rightarrow \underline{n}\cdot\left(\underline{\underline{k}}\cdot(\lambda f_w\nabla(p_o - p_c) + \lambda f_o\nabla p_o - (\lambda f_w\rho_w + \lambda f_o\rho_o)\underline{g})\right) &= g_p^{in}\\
\Rightarrow \underline{n}\cdot\left(\underline{\underline{k}}\cdot(\lambda\nabla p_o - \lambda_w\nabla p_c - (\lambda_w\rho_w + \lambda_o\rho_o)\underline{g})\right) &= g_p^{in}\\
\Rightarrow \underline{n}\cdot\left(\lambda\underline{\underline{k}}\cdot\nabla p_o - \lambda_w\frac{dp_c}{dS_w}\underline{\underline{k}}\cdot\nabla S_w - (\lambda_w\rho_w + \lambda_o\rho_o)\underline{\underline{k}}\cdot\underline{g}\right) &= g_p^{in} \qquad (2.72)
\end{aligned}$$

En este caso la presión capilar es cero y no se toman en cuenta los efectos de la fuerza de gravedad, por lo tanto la ecuación anterior se reduce a

$$\underline{n}\cdot\left(\lambda\underline{\underline{k}}\cdot\nabla p_o\right) \equiv \underline{n}\cdot\left(\lambda\underline{\underline{k}}\cdot\nabla p\right) = g_p^{in}.$$

Tomando en cuenta que la dirección de $\underline{n}$ es opuesta a la dirección de g_p^{in} en la frontera A del dominio, se tiene lo siguiente

$$\begin{aligned}
-\left(\lambda k\frac{\partial p}{\partial x}\right)_{\boldsymbol{A}} &= g_p^{in},\\
\Longrightarrow -\frac{p_{\boldsymbol{P}} - p_{\boldsymbol{W}}}{\Delta x_{\boldsymbol{A}}} &= \frac{g_p^{in}}{k\lambda_{\boldsymbol{A}}},\\
\Longrightarrow p_{\boldsymbol{W}} &= p_{\boldsymbol{P}} + \frac{\Delta x_{\boldsymbol{A}}}{k\lambda_{\boldsymbol{A}}}g_p^{in}. \qquad (2.73)
\end{aligned}$$

Sustituyendo (2.73) en (2.71) se obtiene:

$$\begin{aligned}
\boldsymbol{a}_{\boldsymbol{P}}^n p_{\boldsymbol{P}}^{n+1} &= \boldsymbol{a}_{\boldsymbol{E}}^n p_{\boldsymbol{E}}^{n+1} + \boldsymbol{a}_{\boldsymbol{W}}^n\left(p_{\boldsymbol{P}} + \frac{\Delta x_{\boldsymbol{A}}}{k\lambda_{\boldsymbol{A}}}g_p^{in}\right)^{n+1} +\\
&\quad \boldsymbol{a}_{\boldsymbol{N}}^n p_{\boldsymbol{N}}^{n+1} + \boldsymbol{a}_{\boldsymbol{S}}^n p_{\boldsymbol{S}}^{n+1} + \boldsymbol{a}_{\boldsymbol{F}}^n p_{\boldsymbol{F}}^{n+1} + \boldsymbol{a}_{\boldsymbol{B}}^n p_{\boldsymbol{B}}^{n+1},\\
(\boldsymbol{a}_{\boldsymbol{P}}^n - \boldsymbol{a}_{\boldsymbol{W}}^n)p_{\boldsymbol{P}}^{n+1} &= \boldsymbol{a}_{\boldsymbol{E}}^n p_{\boldsymbol{E}}^{n+1} + \boldsymbol{a}_{\boldsymbol{N}}^n p_{\boldsymbol{N}}^{n+1} + \boldsymbol{a}_{\boldsymbol{S}}^n p_{\boldsymbol{S}}^{n+1} + \boldsymbol{a}_{\boldsymbol{F}}^n p_{\boldsymbol{F}}^{n+1} + \boldsymbol{a}_{\boldsymbol{B}}^n p_{\boldsymbol{B}}^{n+1} + \qquad (2.74)\\
&\quad \boldsymbol{a}_{\boldsymbol{W}}^n\left(\frac{\Delta x_{\boldsymbol{A}}}{k\lambda_{\boldsymbol{A}}}g_p^{in}\right)^{n+1}.
\end{aligned}$$

El valor de la presión en la frontera A se calcula como sigue

$$\begin{aligned} p_{\boldsymbol{A}} &= f p_{\boldsymbol{P}} + (1-f) p_{\boldsymbol{W}}, \\ \Longrightarrow p_{\boldsymbol{A}} &= f p_{\boldsymbol{P}} + (1-f)\left(p_{\boldsymbol{P}} + \frac{\Delta x_{\boldsymbol{A}}}{k\lambda_{\boldsymbol{A}}} g_p^{in} \right), \\ \Longrightarrow p_{\boldsymbol{A}} &= p_{\boldsymbol{P}} + (1-f)\frac{\Delta x_{\boldsymbol{A}}}{k\lambda_{\boldsymbol{A}}} g_p^{in}. \end{aligned} \tag{2.75}$$

donde $f = (x_{\boldsymbol{P}} - x_{\boldsymbol{w}})/\Delta x_{\boldsymbol{w}}$. Para mallas uniformes $f = 1/2$.

Para la frontera derecha, se impone una presión constante p^{out}. De la figura 2.5 (b) se tiene lo siguiente:

$$\begin{aligned} f p_{\boldsymbol{E}} + (1-f) p_{\boldsymbol{P}} &= p^{out} \\ \Longrightarrow p_{\boldsymbol{E}} &= \frac{p^{out} - (1-f) p_{\boldsymbol{P}}}{f} \end{aligned} \tag{2.76}$$

Sustituyendo (2.76) en (2.71) se obtiene:

$$\begin{aligned} \boldsymbol{a}_{\boldsymbol{P}}^n p_{\boldsymbol{P}}^{n+1} &= \boldsymbol{a}_{\boldsymbol{E}}^n \left(\frac{p^{out} - (1-f) p_{\boldsymbol{P}}}{f} \right)^{n+1} + \boldsymbol{a}_{\boldsymbol{W}}^n p_{\boldsymbol{W}}^{n+1} + \\ &\quad \boldsymbol{a}_{\boldsymbol{N}}^n p_{\boldsymbol{N}}^{n+1} + \boldsymbol{a}_{\boldsymbol{S}}^n p_{\boldsymbol{S}}^{n+1} + \boldsymbol{a}_{\boldsymbol{F}}^n p_{\boldsymbol{F}}^{n+1} + \boldsymbol{a}_{\boldsymbol{B}}^n p_{\boldsymbol{B}}^{n+1}, \\ (\boldsymbol{a}_{\boldsymbol{P}}^n + \frac{1-f}{f}\boldsymbol{a}_{\boldsymbol{E}}^n) p_{\boldsymbol{P}}^{n+1} &= \boldsymbol{a}_{\boldsymbol{W}}^n p_{\boldsymbol{W}}^{n+1} + \boldsymbol{a}_{\boldsymbol{N}}^n p_{\boldsymbol{N}}^{n+1} + \boldsymbol{a}_{\boldsymbol{S}}^n p_{\boldsymbol{S}}^{n+1} + \boldsymbol{a}_{\boldsymbol{F}}^n p_{\boldsymbol{F}}^{n+1} + \boldsymbol{a}_{\boldsymbol{B}}^n p_{\boldsymbol{B}}^{n+1} + \\ &\quad \boldsymbol{a}_{\boldsymbol{E}}^n \left(\frac{p^{out}}{f} \right)^{n+1}. \end{aligned} \tag{2.77}$$

Las ecuaciones (2.74) y (2.77) son las que deben incluirse en el sistema lineal a resolver para los volúmenes cercanos a las fronteras A y B, respectivamente.

Condiciones de frontera para la saturación

Para la saturación se imponen condiciones de frontera tipo Dirichlet. Primero se escribe la ecuación discreta de la saturación para este problema como sigue:

$$S_{\boldsymbol{P}}^{n+1} = S_{\boldsymbol{P}}^n - \boldsymbol{b}_{\boldsymbol{P}}^n p_{\boldsymbol{P}}^n + \boldsymbol{b}_{\boldsymbol{E}}^n p_{\boldsymbol{E}}^n + \boldsymbol{b}_{\boldsymbol{W}}^n p_{\boldsymbol{W}}^n + \boldsymbol{b}_{\boldsymbol{N}}^n p_{\boldsymbol{N}}^n + \boldsymbol{b}_{\boldsymbol{S}}^n p_{\boldsymbol{S}}^n + \boldsymbol{b}_{\boldsymbol{F}}^n p_{\boldsymbol{F}}^n + \boldsymbol{b}_{\boldsymbol{B}}^n p_{\boldsymbol{B}}^n \tag{2.78}$$

En la frontera izquierda se tiene que $p_{\boldsymbol{W}}$ esta dada por la ecuación (2.73), entonces la ecuación discreta de (2.78) se transforma en

$$\begin{aligned} S_{\boldsymbol{P}}^{n+1} &= S_{\boldsymbol{P}}^{n} - \boldsymbol{b}_{\boldsymbol{P}}^{n} p_{\boldsymbol{P}}^{n} + \boldsymbol{b}_{\boldsymbol{E}}^{n} p_{\boldsymbol{E}}^{n} + \boldsymbol{b}_{\boldsymbol{W}}^{n} \left(p_{\boldsymbol{P}} + \frac{\Delta x_{\boldsymbol{A}}}{k\lambda_{\boldsymbol{A}}} g_p^{in} \right)^n + \\ & \quad \boldsymbol{b}_{\boldsymbol{N}}^{n} p_{\boldsymbol{N}}^{n} + \boldsymbol{b}_{\boldsymbol{S}}^{n} p_{\boldsymbol{S}}^{n} + \boldsymbol{b}_{\boldsymbol{F}}^{n} p_{\boldsymbol{F}}^{n} + \boldsymbol{b}_{\boldsymbol{B}}^{n} p_{\boldsymbol{B}}^{n} \\ \Longrightarrow S_{\boldsymbol{P}}^{n+1} &= S_{\boldsymbol{P}}^{n} - (\boldsymbol{b}_{\boldsymbol{P}}^{n} - \boldsymbol{b}_{\boldsymbol{W}}^{n}) p_{\boldsymbol{P}}^{n} + \boldsymbol{b}_{\boldsymbol{E}}^{n} p_{\boldsymbol{E}}^{n} + \boldsymbol{b}_{\boldsymbol{N}}^{n} p_{\boldsymbol{N}}^{n} + \boldsymbol{b}_{\boldsymbol{S}}^{n} p_{\boldsymbol{S}}^{n} + \boldsymbol{b}_{\boldsymbol{F}}^{n} p_{\boldsymbol{F}}^{n} + \boldsymbol{b}_{\boldsymbol{B}}^{n} p_{\boldsymbol{B}}^{n} + \\ & \quad \boldsymbol{b}_{\boldsymbol{W}}^{n} \left(\frac{\Delta x_{\boldsymbol{A}}}{k\lambda_{\boldsymbol{A}}} g_p^{in} \right) \end{aligned} \tag{2.79}$$

En la frontera derecha $p_{\boldsymbol{E}}$ está dada por la ecuación (2.76), entonces

$$\begin{aligned} S_{\boldsymbol{P}}^{n+1} &= S_{\boldsymbol{P}}^{n} - \boldsymbol{b}_{\boldsymbol{P}}^{n} p_{\boldsymbol{P}}^{n} + \boldsymbol{b}_{\boldsymbol{E}}^{n} \left(\frac{p^{out} - (1-f) p_{\boldsymbol{P}}}{f} \right) + \\ & \quad \boldsymbol{b}_{\boldsymbol{W}}^{n} p_{\boldsymbol{W}}^{n} + \boldsymbol{b}_{\boldsymbol{N}}^{n} p_{\boldsymbol{N}}^{n} + \boldsymbol{b}_{\boldsymbol{S}}^{n} p_{\boldsymbol{S}}^{n} + \boldsymbol{b}_{\boldsymbol{F}}^{n} p_{\boldsymbol{F}}^{n} + \boldsymbol{b}_{\boldsymbol{B}}^{n} p_{\boldsymbol{B}}^{n} \\ S_{\boldsymbol{P}}^{n+1} &= S_{\boldsymbol{P}}^{n} - \left(\boldsymbol{b}_{\boldsymbol{P}}^{n} + \frac{1-f}{f} \boldsymbol{b}_{\boldsymbol{E}}^{n} \right) p_{\boldsymbol{P}}^{n} + \boldsymbol{b}_{\boldsymbol{W}}^{n} p_{\boldsymbol{W}}^{n} + \boldsymbol{b}_{\boldsymbol{N}}^{n} p_{\boldsymbol{N}}^{n} + \boldsymbol{b}_{\boldsymbol{S}}^{n} p_{\boldsymbol{S}}^{n} + \\ & \quad \boldsymbol{b}_{\boldsymbol{F}}^{n} p_{\boldsymbol{F}}^{n} + \boldsymbol{b}_{\boldsymbol{B}}^{n} p_{\boldsymbol{B}}^{n} + \boldsymbol{b}_{\boldsymbol{E}}^{n} \left(\frac{p^{out}}{f} \right) \end{aligned} \tag{2.80}$$

Las ecuaciones (2.79) y (2.80) se usan en los volúmenes vecinos a las fronteras A y B, respectivamente.

2.6. Modelo computacional

En esta sección, se dará una breve explicación del método BiCGStab, el cual fue utilizado para resolver el sistema de ecuaciones lineales para la presión obtenido de la discretización del módelo de Buckley-Leverett. Además, se incluirá una descripción del cálculo explícito de la saturación, la cual puede ser representada en términos de operaciones vectoriales.

2.6.1. Gradiente Biconjugado Estabilizado (BICGSTAB)

Es un método iterativo del subespacio de Krylov para la solución de sistemas lineales, basado en el gradiente conjugado (CG). Fue creado para sortear la limitante que

tiene el CG al aplicarse únicamente a matrices simétricas, y a que otro método similar, el gradiente conjugado cuadrado (CGS), tiene problemas de sobreflujo y errores de redondeo si la convergencia es irregular [Saad00]. El BiCGStab es usado comúnmente debido a su gran velocidad de convergencia, a su facilidad de implementación, así como a su flexibilidad en el uso de precondicionadores si la matriz está mal condicionada. Una posible implementación se muestra en el pseudocódigo mostrado en el algoritmo 2.

Algoritmo 2 BiCGStab

1: $r_0 \leftarrow b - Ax_0$
2: r_0^* arbitrario
3: $p_0 \leftarrow r_0$
4: **for** $j = 0, 1, 2 \ldots$ **until convergence do**
5: $\quad \alpha_j \leftarrow \frac{(r_j, r_0^*)}{(Ap_j, r_0^*)}$
6: $\quad s_j \leftarrow r_j - \alpha_j A p_j$
7: $\quad \omega_j \leftarrow (As_j, s_j)/(As_j, As_j)$
8: $\quad x_{j+1} \leftarrow x_j + \alpha_j p_j + \omega_j s_j$
9: $\quad r_{j+1} \leftarrow s_j - \omega_j A s_j$
10: $\quad \beta_j \leftarrow \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
11: $\quad p_{j+1} \leftarrow r_{j+1} + \beta_j (p_j + \omega_j A p_j)$
12: **end for**

En la implementación del BiCGStab se realizan, dentro del ciclo de solución diversos tipos de operaciones de álgebra lineal, los cuales se muestran en la tabla 2.1. La mayor parte del tiempo dentro de cada paso del BiCGStab se realiza en las multiplicaciones matriz por vector $\underline{\underline{A}}\ \underline{s}$ y $\underline{\underline{A}}\ \underline{p}$, las cuales se asignan a dos vectores temporales evitando la repetición de cálculos, realizando únicamente dos multiplicaciones matriz-vector en cada paso. Como veremos en la sección 2.6.3, en el caso de matrices dispersas, estas operaciones presentan problemas de optimización difíciles de resolver, debido principalmente a la no coalescencia (adyacencia en la memoria) de los elementos que intervienen en las operaciones vectoriales.

Escala	Contracción	axpy	Duplicación	Matriz por Vector
$\underline{y} \leftarrow \alpha \underline{y}$	$c \leftarrow \underline{x} \cdot \underline{y}$	$\underline{y} \leftarrow \underline{y} + \alpha \underline{x}$	$\underline{y} \leftarrow \underline{x}$	$\underline{y} \leftarrow \underline{\underline{A}}\ \underline{x}$
1	6	6	2	2

Tabla 2.1: Operaciones vectoriales realizadas en cada iteración del BiCGStab.

2.6.2. Saturación explícita

A cada uno de los N nodos de la malla, digamos el I, le corresponde valores del campo de presión P_I, saturación S_I, etc. Para cada campo C, podemos juntar los valores de los N nodos en un vector $\underline{C}$ de N entradas. Entonces, definiendo $\underline{S}^{n+1}$, $\underline{S}^n$, $\underline{P}^n$ y $\underline{q}^n$, los vectores correspondientes para la saturación al tiempo $n+1$, la saturación al tiempo n, la presión al tiempo n y el término fuente al tiempo n respectivamente, tenemos:

$$\begin{aligned}
\underline{S}^{n+1} &= \{S_1^{n+1}, S_2^{n+1}, ..., S_I^{n+1}, ..., S_N^{n+1}\} \\
\underline{S}^{n} &= \{S_1^{n}, S_2^{n}, ..., S_I^{n}, ..., S_N^{n}\} \\
\underline{P}^{n} &= \{p_1^{n}, p_2^{n}, ..., p_I^{n}, ..., p_N^{n}\} \\
\underline{q}^{n} &= \{S_1^{n}, S_2^{n}, ..., S_I^{n}, ..., S_N^{n}\}, .
\end{aligned}$$

Con estas definciones, a partir de la ecuación 2.65,

$$S_{\boldsymbol{P}}^{n+1} = S_{\boldsymbol{P}}^{n} - \boldsymbol{b}_{\boldsymbol{P}}^{n} p_{\boldsymbol{P}}^{n} + \boldsymbol{b}_{\boldsymbol{E}}^{n} p_{\boldsymbol{E}}^{n} + \boldsymbol{b}_{\boldsymbol{W}}^{n} p_{\boldsymbol{W}}^{n} + \boldsymbol{b}_{\boldsymbol{N}}^{n} p_{\boldsymbol{N}}^{n} + \boldsymbol{b}_{\boldsymbol{S}}^{n} p_{\boldsymbol{S}}^{n} + \boldsymbol{b}_{\boldsymbol{F}}^{n} p_{\boldsymbol{F}}^{n} + \boldsymbol{b}_{\boldsymbol{B}}^{n} p_{\boldsymbol{B}}^{n} + q_{\boldsymbol{P}}^{n}, \tag{2.81}$$

vista en secciones anteriores de este capítulo, el cálculo explícito de la saturación al tiempo $n+1$ se puede expresar como una operación de álgebra lineal, si se identifica a cada elemento P (volumen de control) de la malla con el correspondiente elemento I de los vectores que representan cada uno de los campos.

Para esto, se define una matriz cuadrada $\underline{\underline{B}}^n$ con 7 diagonales distintas de cero, cada una con los correspondientes valores $\boldsymbol{b}_\alpha^n$ de la ecuación 2.81 , con $\alpha = \{1, 2..., N\}$, colocados de la siguiente manera

$$\underline{\underline{B}} = \begin{vmatrix} -b_1^n & b_2^n & . & b_{bf}^n & . & . & b_{sn}^n & & & & & & & & & \\ . & . & . & . & . & . & . & . & . & & & & & & & \\ & . & . & . & . & . & . & . & . & . & . & & & & & \\ & b_{I-sn}^n & . & . & b_{I-bf}^n & . & b_{I-1}^n & -b_I^n & b_{I+1}^n & . & b_{I+bf}^n & . & . & b_{p+sn}^n & & \\ & & & . & . & . & . & . & . & . & . & . & . & . & & \\ & & & & & & . & . & . & . & . & . & . & . & . & \\ & & & & & & & & b_{N-sn}^n & . & . & b_{N-bf}^n & . & b_{N-1}^n & -b_N^n \end{vmatrix},$$

con sn, bf el número de elementos (fijo) desde cualquier elemento I del vector $\vec{S^n}$, a los elementos al norte o sur (N o S), y al frente o detrás (F o B) respectivamente sobre la malla. Los elementos $I+1$, $I-1$ corresponden con los elementos al este y al oeste (E y W) respectivamente, y el elemento I al elemento al centro del volumen de control (P).

Con estas definiciones, el cálculo de la saturación queda expresado en términos de una multiplicación matriz por vector y dos axpys, como puede observarse en la ecuación 2.82

$$\underline{S}^{n+1} = \underline{S}^n + \underline{\underline{B}}^n \underline{P}^n + \underline{q}^n \tag{2.82}$$

2.6.3. Limitantes en la optimización

Formato de almacenamiento por renglón comprimido o CRS (Compressed Row Storage)

Cuando una matriz tiene una gran cantidad de entradas fijas con valor igual cero a lo largo de toda una simulación, por ejemplo la matriz obtenida en un problema de diferencias finitas sobre una malla cartesiana, es conveniente no almacenar los elementos nulos. Esto ahorra memoria, y permite estudiar problemas de mayor tamaño sin perder información. Existen diversos formatos de almacenamiento, los cuales tienen ventajas y desventajas entre sí, y en general, el tipo de problema a estudiar es el que determina cual es el mejor formato a utilizar. En el siguiente listado se nombran algunos de los más usados (véase [Saad00]) y una breve descripción de cada uno:

1. Formato de almacenamiento por coordenadas: este formato almacena tres arreglos, cada uno con dimensión igual al número N_z de entradas distintas de cero. El primer arreglo contiene los coeficientes no nulos de la matriz, el segundo arreglo contiene el renglón al que pertenece cada coeficiente no nulo almacenado, y el tercer arreglo contiene la columna a la que pertenece dicho coeficiente.

2. Formato de almacenamiento por renglón (columna) comprimido: en este formato se elimina la redundancia existente en el formato de almacenamiento por coordenadas, al evitar almacenar, para todos los elementos de un mismo renglón (columna), la información del renglón (columna) al que pertenecen, pues es el mismo valor para los elementos de dicho renglón (columna). Almacena un arreglo de mucho menor tamaño, el cual contiene el índice de inicio y finalización de los elementos no nulos de un mismo renglón (columna) extraídos sobre el arreglo de coeficientes.

3. Formato de almacenamiento diagonal: este formato extrae completas todas las diagonales de la matriz que tengan elementos distintos de cero. Almacena cada diagonal como una columna de una matriz rectangular, y almacena la distancia que hay de cada diagonal extraída a una diagonal de referencia (en general la principal). Si la matriz tiene una clara estructura bandada, como las matrices de diferencias finitas en mallas cartesianas, este formato es el que obtiene una mayor compresión. Por el contrario si su estructura es irregular, su compresión puede ser nula e incluso puede almacenar una cantidad mayor de elementos que la matriz original, al almacenar elementos no existentes en ellas (todos los arreglos que contienen las diagonales extraídas son del mismo tamaño que la diagonal principal).

4. Formato de almacenamiento ELLPACK: en este formato, se crea un arreglo bidimensional, con tantos renglones como tenga la matriz original, y de N_{zr} columnas, donde N_{zr} es el número máximo de elementos distintos de cero en los renglones de la matriz original. Este formato es similar al de compresión diagonal, pues existen solo dos arreglos, uno con los coeficientes extraídos de la matriz, y otro en el que se guarda la columna en la que se encuentra cada elemento.

El formato más común es el CRS y casi cualquier biblioteca de álgebra lineal (serial o paralela) maneja dicho formato, y fue el formato elegido en este trabajo. Daremos una descripción general de este formato.

A partir de una matriz dispersa de tamaño $N \times N$, con una cantidad N_z de elementos distintos de cero, se generan los siguientes tres arreglos:

1. A^*: contiene los N_z coeficientes distintos de cero, colocados en el mismo orden que van apareciendo en la matriz original (avanzando sobre las columnas).

2. J: debido a que el arreglo A^*, no contiene información alguna de la ubicación sobre los renglones o las columnas de cada coeficiente almacenado, se define el arreglo J (del mismo tamaño N_z que el arreglo A^*), el cual guarda la columna sobre la que se localizaba el coeficiente almacenado en la matriz original.

3. I: este arreglo completa la información sobre la ubicación de cada elemento extraído en A^*, ya que el arreglo J no contiene información sobre el renglón en el que se encontraba dicho elemento. Para obtener dicha información, no es necesario indicar en que renglón se localiza cada elemento extraído, solo basta indicar cual es la partición del vector A^*, es decir, el índice (del vector A^*) donde está ubicado el primer elemento de cada renglón de la matriz original. Este arreglo es de tamaño $N+1$ y en la última entrada se guarda el número de elementos distintos de cero $N_z + 1$.

Para dar una idea más clara de este formato, veamos el siguiente ejemplo, en el cual una matriz $\underline{\underline{A}}$ de tamaño $N \times N$ se comprime a tres arreglos A^*, I y J:

$$A = \begin{vmatrix} a & 0 & 0 & b & 0 \\ c & 0 & 0 & d & e \\ 0 & 0 & f & g & 0 \\ 0 & h & 0 & 0 & 0 \\ 0 & i & 0 & j & 0 \end{vmatrix}$$

El arreglo A^* se llena extrayendo, renglón por renglón, todos los elementos distintos de cero. El arreglo J guarda la colmna de cada elemento extraído en A^*. El arreglo I guarda la ubicación en el arreglo A^*, del primer elemento de cada renglón en la matriz original $\underline{\underline{A}}$.

$$A^* = [a, b, c, d, e, f, g, h, i, j] \tag{2.83}$$
$$J = [1, 4, 1, 4, 5, 3, 4, 2, 2, 4]$$
$$I = [1, 3, 6, 7, 8, 11]$$

Desventajas en el acceso a memoria

La multiplicación Matriz-Vector es muy ineficiente si la matriz usada está en algún formato de compresión (para el caso de Buckley-Leverett es dispersa), debido a que no existe coalescencia entre los datos necesarios para obtener cada una de las entradas del vector resultante. Para ver esto, representemos dicha operación en forma tensorial, i.e. $y_i = A_{ij}x_i$. Cuando se calcula el elemento I del vector resultante y, se obtiene la suma

$$y_I = \sum_{j=1}^{N} A_{Ij}x_j. \tag{2.84}$$

Para una matriz densa (sin formato de compresión) almacenada por renglones, cada elemento A_{Ij+1} en los bancos de memoria es contigüo al elemento A_{Ij}. De igual manera, para cualquier vector, el elemento i y el $i+1$ residen en localidades vecinas en la memoria. Un procesador moderno, al realizar una operación como en la ec. 2.84, cada vez que solicita de los bancos de memoria los elementos necesarios para realizar la multiplicación de la pareja $A_{IJ}x_J$, trae consigo dos bloques de memoria, los cuales además de contener a cada elemento de la pareja a multiplicar, incluyen a elementos vecinos ($\approx$ 32 Kbytes por bloque en una procesador Intel i5). Esto reduce el tráfico de información entre el procesador y los distintos niveles de memoria, pues al terminar la carga del los elementos $A_{IJ}x_J$, puede realizar la carga inmediata de los elementos $A_{IJ+1}x_{J+1}$ ya que estos se encuentran dentro del bloque de memoria que está actualmente en el cache. Para el caso de matrices dispersas en algún formato de compresión, existe un problema al realizar la suma de la ecuación 2.84. Cuando se calcula cada elemento y_I, los sumandos que participan en general utilizan elementos no consecutivos en los respectivos arreglos (existían ceros entre ellos antes de comprimir la matriz). Esto puede generar que al cargarse todo un bloque, éste contenga pocos elementos que intervengan en la suma de la ecuación 2.84, ya que la separación entre

ellos en la memoria puede ser mayor que el tamaño de bloque subido al caché. Como la cantidad de elementos utilizados por cada bloque cargado puede ser muy pequeña, se genera un desperdicio en el tiempo de carga y uso de información por el procesador. Esto se muestra de forma sencilla con el siguiente ejemplo. Calculemos la primer entrada de un vector y, obtenido al multiplicar la matriz A del ejemplo anterior por el vector $x = [A, B, C, D, E]$. El resultado es el siguiente:

$$y_1 = A_{11} * x_1 + A_{14} * x_4 \tag{2.85}$$

$$a * A + b * D \tag{2.86}$$

$$A_1^* * A + A_2^* * D \tag{2.87}$$

$$A_1^* * x_1 + A_2^* * x_4 \tag{2.88}$$

Podemos observar que, para calcular la primera entrada y_1, se requieren dos datos contiguos de A^* (elementos A_1^*, A_2^*), pero fue necesario el acceso a otros dos elementos no contigüos de x, (elementos x_1, x_4), los cuales si el problema es relativamente grande, seguramente no podrán ser cargados en el mismo bloque. Para matrices dispersas muy grandes y/o sin regularidad, esto ocasiona un tráfico muy grande entre los distintos niveles de memoria, llevando a grandes ineficiencias. Sin embargo, comparada con el tiempo y memoria que se requeriría realizar la misma operación con una matriz densa, estas ineficiencias son despreciables.

Hasta este punto hemos discutido que el usar formatos de compresión tiene un acceso ineficiente de memoria, pero que esa ineficiencia es despreciable comparado con la ventaja de ahorro de memoria y tiempo de ejecución que tienen estos formatos con respecto a las operaciones con matrices densas. Entonces, ¿porqué hablar de ineficiencias si estas son despreciables?. Recordemos que estamos comparando distintas arquitecturas de cómputo paralelo, y que se quiere conocer el desempeño que tiene una u otra. En sistemas paralelos, principalmente en los distribuidos, el manejo ineficiente de intercambio de información entre bancos de memoria puede llevar a consecuencias desastrosas en el desempeño de un sistema. En problemas que requieren operaciones de álgebra lineal con matrices e incluso vectores dispersos, el manejo y la velocidad de la memoria son los que marcan el límite en

el desempeño, independientemente de la arquitectura en la que se ejecuten.

2.7. Conclusiones del capítulo

A lo largo de este capítulo se desarrollo todo el modelo maematico, numérico y computacional alrededor del problema de Buckley-Leverett. A partir del modelo matemático, se generó la discretización de las ecuaciones diferenciales gobernantes, y la solución de los sistemas lineales que arrojó dicha discretización, será el material empleado en las secciones siguientes para comparar cual tecnología paralela puede ser más adecuada (en términos de desempeño) para resolver dichos sistemas. La forma de almacenamiento de dichos sitemas y su rapidez de acceso, serán parte crucial en las métricas realizadas.

Capítulo 3

Arquitecturas de cómputo paralelo y bibliotecas empleadas

Con el desarrollo de los lenguajes de programación y de las computadoras durante la década de los 70, se expandieron las posibilidades de la ciencia y la tecnología para utilizar métodos numéricos en la resolución de problemas complejos. Sin embargo esto planteó un nuevo problema, pues si bien es cierto que se podía simular sistemas físicos, el tamaño de estos (medido por ejemplo, en el número de elementos del sistema), era demasiado pequeño como para poder poder extender las conclusiones a sistemas de mayor tamaño, más cercanos a la realidad y cuyo tamaño requeriría de recursos, principalmente de memoria y tiempo, que ningún sistema por si mismo podría ofrecer. La forma en que se resolvió esta limitante, fue reformulando los problemas de mayor tamaño, segmentándolos de alguna manera, para que cada una de sus partes pudiera ser asignada y procesada en una computadora distinta, e ir conjuntando cada procesamiento para construir la solución final. Para esto, también se tuvieron que reformular y/o extender los lenguajes de programación y los algoritmos de solución. Aquel nuevo esquema de cómputo, sirvió la base de lo que hoy se conoce como Cómputo de Alto Rendimiento (HPC por las siglas en inglés de *High Performance Computing*). El tener que separar el sistema generó nuevas dificultades, ya que al tener que dividir el problema en varias partes, cada una tenía que obtener, procesar y regresar cierta parte del total de información, la cual a su vez, podía ser necesitada por otros procesos. Esto

requirió de la definición de un esquema en la interacción de los procesos con la memoria, en el que se establecieron los modos de acceso, uso y escritura de información entre los bancos de memoria. A partir de esto, se crearon modelos de manejo de memoria [Kaufmann03], en los cuales se especifica la distribución y acceso a la información (memoria) entre el conjunto de procesos que ejecutan algún programa paralelo. Cuando toda la información es accesible a todos los procesos, se tiene un modelo de memoria compartida. Si la memoria está segmentada y distribuida entre varios sistemas, y cada parte es asignada a un proceso distinto, solo este tendrá acceso directo a su segmento de memoria. Este segundo modelo se designa de memoria distribuida. Existe un tercer modelo, el cual emplea los dos esquemas anteriores, estableciendo en general una jerarquía de manejo de memoria, empleando un esquema u otro en cada nivel de la jerarquía. Este esquema distribuido de memoria compartida, puede presentarse, por ejemplo, en un cluster(sistema distribuido) de multi-procesadores simétricos (memoria compartida).

3.1. Modelos de memoria

3.1.1. Esquema de Memoria Distribuida

La forma más sencilla de resolver las limitantes de memoria y/o tiempo de ejecución, es la repartición de actividades a un conjunto de computadoras conectadas por una red, cada una corriendo su propio proceso, los cuales se comunican entre si a través de alguna red [Kaufmann03]. Este funcionamiento básico dió origen a un modelo de programación, basado en el envío de mensajes. Este modelo originalmente fue el utilizado en las arquitecturas paralelas de memoria distribuida, aunque hoy existen bibiliotecas estandarizadas de comunicación sobre modelos de memoria distribuida como MPI, que empiezan a manejar de manera básica el concepto de memoria compartida[Groop98], lo cual permite mezclar los dos modelos de programación. Un esquema básico de la configuración general de un sistema con arquitectura de memoria distribuida, se muestra en la figura 3.1.

Dentro del HPC, sistemas paralelos basados en memoria distribuida son los más comunes, dada su facilidad de montaje, y en general, la parte más compleja es el establecimiento de una red de comunicación eficiente y con gran ancho de banda, para evitar

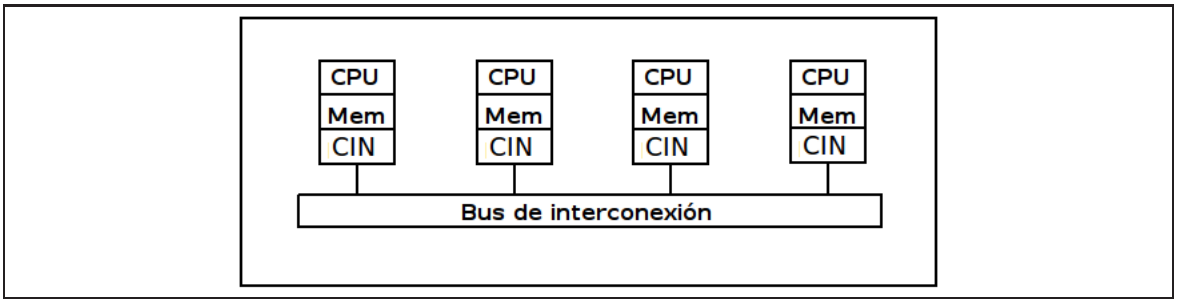

Figura 3.1: Configuración básica de un sistema de memoria distribuida. Cada procesador tiene su propio banco de memoria, invisible directamente para los demás procesadores, y la comunicación se realiza mediante controladores de interfaces(CIN) sobre una red con cierta topología

tiempos muertos en la ejecución al realizar la comunicación de mensajes. De ahí que el el principal problema al utilizar este modelo de programación, esté en la dependencia de datos(información) que pueda existir entre procesos. Si cada proceso necesita mucha información de otro banco de memoria, como en el caso de simulaciones de N cuerpos con interacciones entre todos los pares de partículas [Liu00], el tiempo de transmisión-recepción puede ser igual e incluso mayor que el tiempo de procesamiento, lo cual destruye cualquier ventaja obtenida al haber distribuido el tiempo de cálculo. Por el contrario, si el problema (y el algoritmo) puede ser fácilmente distribuido, como en el cifrado de claves encriptadas, la comunicación es casi nula, lo cual permite, al trabajar independientemente cada proceso, reducir los tiempos de ejecución drásticamente (proporcionalmente al número de procesos ejecutados simultáneamente).

Otra ventaja de los sistemas distribuidos, es que pueden ser ejecutados sobre sistemas heterogéneos, permitiendo la creación de un sistema con un número realmente grande de procesadores. Un ejemplo reciente del poder de cómputo de estos sistemas heterogéneos es el proyecto Folding@home, el cual en el 2010, alcanzó el record mundial de operaciones de punto flotante por segundo (6.3 PetaFLOPS[Fol]).

3.1.2. Memoria Compartida

El modelo de memoria compartida tiene la enorme ventaja con respecto a su similar distribuido, de la accesibilidad de todos los bancos de memoria a todos los procesadores,

pudiendo así cualquier procesador obtener cualquier dato almacenado en la memoria. Para evitar problemas de inconsistencia de memoria, el acceso a regiones de la memoria pasa por un proceso de confirmación, en el cual se garantiza que ningún procesador lea o escriba en alguna unidad de memoria, mientras otro haya iniciado un proceso similar. El evitar comunicar grandes cantidades de información, hace que las arquitecturas con memoria compartida puedan alcanzar una mayor cantidad de FLOPS, que los sistemas basados en memoria distribuida (si estos tienen el mismo número de procesadores, con la misma velocidad y con la misma cantidad de memoria). Entre sus desventajas está la poca escalabilidad, principalmente en la complejidad de la red de conexión a la memoria y del algoritmo de acceso a dicha memoria, y al costo que el sistema de comunicación (bus de interconexión) tiene[Kaufmann03].

Debido al avance de la tecnología de microchips y del estudio de los semiconductores, en las dos últimas décadas del siglo pasado, el tamaño del chip del procesador disminuyó de forma drástica, llevándolo a escalas en las cuales los efectos cuánticos influyen en las compuertas lógicas. Además, debido a la enorme frecuencia de estos dispositivos ($\approx GHz$), la cantidad de energía liberada en forma de calor por el procesador es demasiada, por lo que mantenerlo a temperaturas de operación se volvió cada más difícil. Esto, sumado a la predicción de la ley de Moore [Kaufmann03], y al retraso que existe en el desarrollo de la memoria RAM (las velocidades de lectura-escritura son aproximadamente un cuarto del reloj del procesador), y a la velocidad de entrada-salida del bus de datos del procesador, los diseñadores de procesadores se vieron forzados a buscar alternativas para ofrecer procesadores más veloces. Esto llevo a desarrollar los multi-procesadores simétricos, los cuales son procesadores (núcleos) con su propia memoria local(cache), pero que comparten una memoria global. Cada núcleo tiene acceso directo a la memoria global, y existen mecanismos para garantizar la coherencia en la lectura-escritura de datos realizada por distintos núcleos a las mismas unidades de memoria. Un esquema básico de un multi-procesador simétrico se muestra en la figura 3.2.

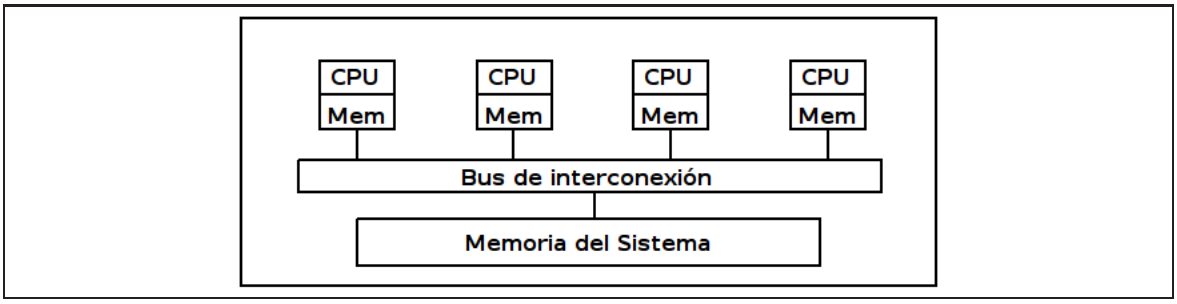

Figura 3.2: Configuración básica de un sistema de memoria compartida con acceso uniforme. Cada procesador tiene acceso a cualquier banco de memoria vía el bus.

3.2. Conjunto de instrucciones

El modelo de memoria usado, no determina completamente el tipo de arquitectura paralela que se tiene. Además se tiene que definir que tipo de instrucción es la que el procesador realiza. Según la clasificación de Flynn[Flynn72], existen 4 tipos de flujo de instrucción, las cuales indican el tipo de operación que ejecuta un sistema sobre un conjunto de datos. Las relaciones entre operación-conjunto de datos son:

1. SISD (Single Instruction stream, Single Data stream): En general, es realizado por una máquina secuencial, en la cual se ejecuta un solo flujo de instrucciones a un solo dato.

2. SIMD (Single Instruction stream, Multiple Data stream): Se ejecuta una solo flujo de instrucciones pero a un conjunto de datos. Procesadores con este tipo de relación, son conocidos como vectoriales.

3. MISD (Multipe Instruction stream, Multiple Data stream): Se ejecuta un conjunto de flujo de instrucciones sobre un solo conjunto de datos. En general son máquinas paralelas, las cuales controlan un solo dispositivo. Son diseñadas para control de fallas.

4. MIMD (Multiple Instruction stream, Multiple Data stream): Se ejecuta un conjunto de flujos de instrucciones sobre conjuntos independientes de datos. Este es el modelo mas común en máquinas paralelas de muchos procesadores, como un cluster de procesadores tipo SISD.

   Un esquema simplificado de la clasificación de Flynn se muestra en la figura 3.3

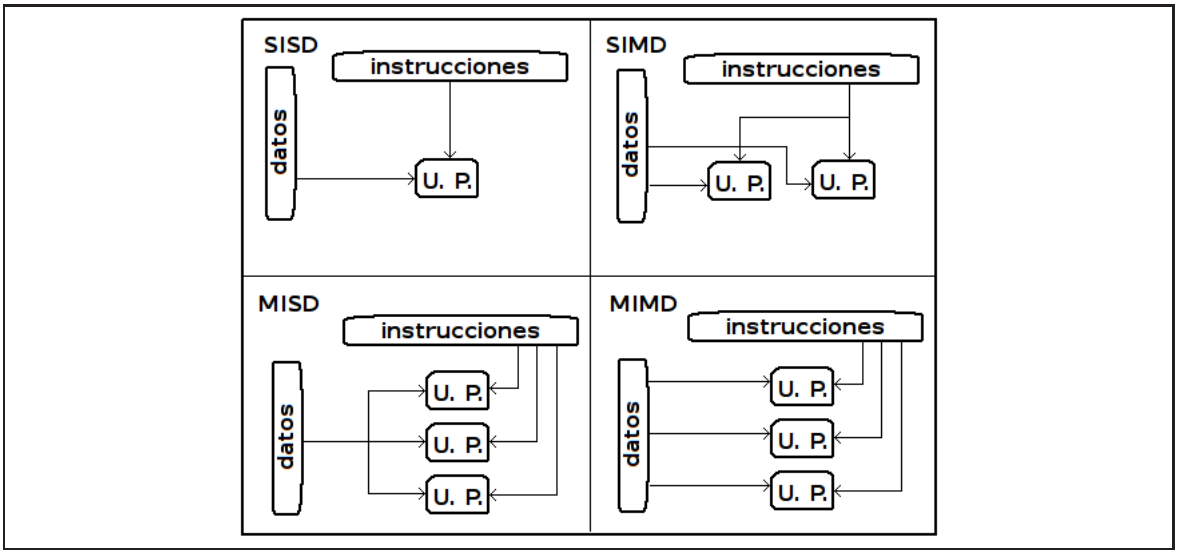

Figura 3.3: Modelos básicos de arquitecturas con distinto conjunto de instrucciones. (U. P. Unidad de Procesamiento).

3.2.1. Arquitecturas a utilizar

En el capítulo 5 de este trabajo, son mostradas métricas de rendimiento, realizadas sobre arquitecturas paralelas de varios tipos posibles, combinando modelos de memoria y conjunto de instrucciones. Se eligieron tres sistemas, dos de memoria compartida y uno de memoria distribuida, descritos en el siguiente listado:

1. Sistema tipo SPMD (Single Program, Multiple Data), ejecutado sobre un modelo de memoria compartida. El conjunto de instrucciones SPMD es una variación del sistema SIMD, en el cual se ejecuta un mismo conjunto de instrucciones (programa) sobre distintos conjuntos de datos. Este tipo de procesadores (vectoriales) han tenido un resurgimiento, a partir del uso de tarjetas gráficas como procesadores de propósito general, principalmente para cómputo numérico[Sanders10].

2. Sistema tipo MIMD, ejecutado sobre un modelo de memoria compartida. Estos sistemas han sido desarrollados fuertemente en los últimos diez años, al aumentar el número de núcleos un solo chip, los cuales comparten el mismo banco de memoria (Multi-procesadores simétricos como los de *Intel* o *AMD*).

3. Sistema tipo MIMD, ejecutado sobre un modelo de memoria distribuida. Es un cluster

de procesadores tipo (SISD), conectados por algún tipo de red (puede ser virtual).

3.3. Bibliotecas numéricas de cómputo paralelo

Como vimos en el capítulo 2, la solución al problema de Buckley-Leverett puede ser aproximada utilizando el método de Volumen Finito sobre una malla cartesiana. Dado que éste es un método numérico, la utilización de paquetes y/o bibliotecas de alto nivel que implementan dicho método es un recurso muy utilizado en la comunidad científica, ya que esto permite ahorrar una cantidad considerable de tiempo, al ser necesario únicamente conocer y manipular el conjunto de funciones de dicho paquete, sin tener que implementar desde cero toda una simulación en algún lenguaje de programación. Entre dichos paquetes podemos nombrar OpenFVM ([Ope]), FEniCS Project ([Fen]), PETSc ([PET]), y TUNA ([TUN]), por mencionar algunos, y existen paquetes para cualquier lenguaje de alto nivel común, como C/C++, Fortran, Java, Python, etc.

Para este trabajo, se eligió TUNA para resolver el problema de Buckley-Leverett, el cual puede requerir de la solución de grandes sistemas de ecuaciones lineales. Se eligió principalmente por su facilidad de implementación, por la cercanía del autor con el desarrollador de TUNA, y por que en el proyecto en el que se desarrolló tesis, TUNA ha servido como herramienta en varias ocasiones. Dado que en la implementación de una simulación utilizando TUNA se puede mezclar el código con el de diversas bibliotecas de solución de sistemas lineales, es posible evaluar el rendimiento entre distintas bibliotecas, al someter todas a las mismas condiciones (tamaño, tipo y contenido de datos, condiciones iniciales, etc). A continuación se dará una breve descripción del funcionamiento del TUNA, así como de las 3 bibliotecas numéricas utilizadas dentro del TUNA para resolver los sistemas lineales que se producen durante su ejecución. Las tres bibliotecas que se usarán para resolver los sistemas lineales están construidas sobre distintos modelos de memoria y orientadas a distintas arquitecturas, lo cual nos permitirá evaluar cual es el rendimiento de una misma simulación cuando es ejecutada sobre distintas arquitecturas, y sacar conclusiones acerca de cual es la mejor opción al momento de resolver problemas similares al de Buckley-Leverett.

La solución de sistemas lineales en paralelo puede ser llevado a cabo en distintas

arquitecturas y modelos de memoria. La forma en que cada una de las bibliotecas que se evaluarán realiza la paralelización se explica en el siguiente listado:

1. PETSc. Es una biblioteca que requiere ejecutarse sobre el modelo de memoria distribuida, usando MPI para comunicar la información necesaria. El conjunto de instrucciones sobre el que trabaja eficientemente es de tipo MIMD[PET].

2. IntelMKL. Esta biblioteca está construida sobre la tecnología *Multithread-OpenMP*, lo cual requiere una arquitectura con memoria compartida, con un conjunto de instrucciones tipo MIMD[Int].

3. CUDA. Este es un conjunto de compilador y bibliotecas que utiliza el modelo de memoria compartida de una tarjeta de gráficos, ejecutándose sobre un conjunto de instrucciones tipo SPMD(Single Program, Multiple Data).

A groso modo, en la figura 3.4 se muestra el esquema sobre el cual se desarrolló la comparación de desempeño de entre las tres bibliotecas. En la parte superior se encuentra la misma aplicación (TUNA), la cual se comunica mediante distintas bibliotecas con un mismo sistema operativo. Sin embargo, al requerir distintos dispositivos de comunicación y de cómputo para cada biblioteca, en general, ocupará distintas partes del sistema operativo. Estó implica que la comparación es a nivel de aplicación, y la eficiencia que tenga sobre cada biblioteca, estará ligada la respectiva tecnología que ejecute las instrucciones establecidas por cada biblioteca.

3.3.1. TUNA

TUNA es un conjunto de funciones, clases y espacios de nombres parametrizadas (templates) para resolver numéricamente las ecuaciones gobernantes del flujo de fluidos usando el método de Volumen Finito. El propósito inicial fue simular fenómenos de convección natural en cavidades rectangulares, pero debido a su construcción genérica y al uso de programación orientada a objetos, nuevas características pueden agregarse fácilmente [TUN]. Está escrita en C++, hace uso intensivo de templates y utiliza la biblioteca Blitz++[OOS]

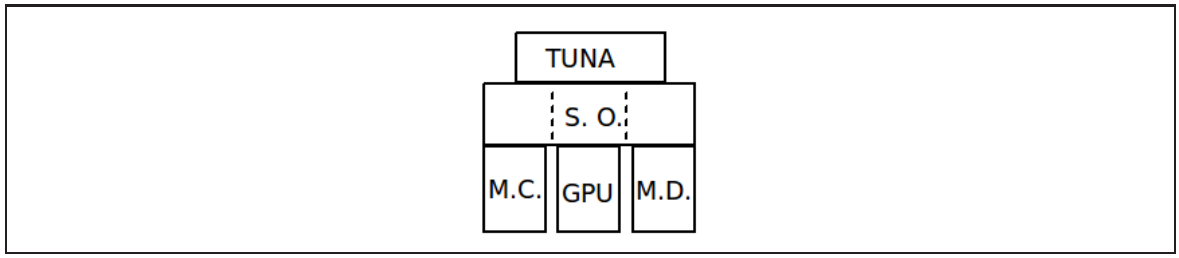

Figura 3.4: Esquema de comparación: Sobre el sistema operativo (S.O.) se ejecuta TUNA, el cual mediante la interfaz a cada biblioteca, emplea distintas partes del sistema operativo para realizar la comunicación y cómputo en las diferentes tecnologías (memoria compartida (M.C.), tarjeta gráfica (GPU), memoria distribuida (M.D.))

para el manejo de arreglos. Para más detalles acerca del funcionamiento de TUNA véase [TUN]. En esta biblioteca se abstraen los conceptos matemáticos del problema a resolver, y se asocia un objeto a dicha abstracción, definiendo por ejemplo, la ecuación diferencial que gobierna el sistema a simular, y asociándola a una clase parametrizada (template) que describe dicha ecuación y tiene cierta funcionalidad (como por ejemplo, la generación, cálculo y actualización de coeficientes, condiciones de frontera, etc., de la matriz resultante al discretizar la ecuación diferencial). Para resolver el problema de Buckley-Leverett descrito en el capítulo 2, usando TUNA, se hace lo siguiente: Primero se define la malla y los campos donde se guardará la solución, de la siguiente forma:

Código (C++) 3.1: Generación de mallas y campos

```
StructuredMesh<Uniform<double, 3> > mesh(length_x, num_nodes_x,
                                         length_y, num_nodes_y,
                                         length_z, num_nodes_z);
ScalarField3D p ( mesh.getExtentVolumes() );
ScalarField3D Sw ( mesh.getExtentVolumes() );
ScalarField3D p_n( mesh.getExtentNodes()   );
ScalarField3D Sw_n( mesh.getExtentNodes()   );
```

Haciendo uso de los objetos *mesh, p, Sw, p_n y Sw_n* definidos anteriormente, se pueden crear objetos para manejar las ecuaciones de presión y saturación. Por ejemplo, para el cálculo de la presión se realiza lo siguiente:

Código (C++) 3.2: definición y configuración del objeto pressure

```
    TwoPhaseEquation< BLIP1<double, 3> >
            pressure(p, A, b, mesh.getDeltas());
pressure.setDeltaTime(dt);
pressure.setPermeability(permeability);
pressure.setPorosity(porosity);
pressure.setSrw(Srw);
pressure.setSro(Sro);
pressure.setViscosity_w(mu_w);
pressure.setViscosity_o(mu_o);
pressure.setNeumann   (LEFT_WALL, -injection * mu_w / permeability);
pressure.setDirichlet(RIGHT_WALL);
pressure.setNeumann   (TOP_WALL);
pressure.setNeumann   (BOTTOM_WALL);
pressure.setNeumann   (FRONT_WALL);
pressure.setNeumann   (BACK_WALL);
pressure.setSaturation(Sw);
```

Observese que el objeto *pressure* contiene toda la información necesaria para controlar el esquema numérico de discretización, la malla, las condiciones iniciales y de frontera, entre otras. Un código similar se realiza para definir un objeto *saturation*, el cual manejará la ecuación de saturación.

El método IMPES descrito en el algoritmo 1, se implementa como sigue:

Código (C++) 3.3: IMPES

```
while (t <= Tmax) {
            pressure.calcCoefficients();
            Solver::TDMA3D(pressure, tolerance, 20, 1);
            pressure.update();
            saturation.calcCoefficients();
            Solver::solExplicit3D(saturation);
            saturation.update();
    }
```

Dentro del ciclo IMPES, el TUNA utiliza los métodos *TDMA* y *solExplicit3D*, en los cuales se resuelve los sistema de ecuaciones lineales para la presión y para la saturación

variable	valor	unidades
L	256	m
a	16	m^2
g_P^{in}	4.4722×10^{-7}	$\frac{m^3}{s}$
p_o^{out}	1×10^7	Pa
$\mu_w = \mu_o$	0.001	$Pa \cdot s$
δt	6000	s
c. f. cara a :	Dirichlet	–
c. f. otras caras :	Neumann	–
S_{rw}	0	–
S_{r0}	0.2	–
k	1.15×10^{-15}	m^2
θ	1	–
esquema sat. :	Upwind	–
$nx = ny$	16	–
nz	1024	–
$dx = dy = dz$	0.25	m
TIEMPO TOTAL	1251	*días*

Tabla 3.1: Valores utilizados en la simulación del problema de Buckley-Leverett usando TUNA.

respectivamente (el *TDMA* es una variación del método de Thomas de matrices tridiagonales). Esas líneas son las que pueden ser modificadas para utilizar cualquier biblioteca de solución de sistemas lineales, únicamente extrayendo los campos contenidos en los objetos *pressure* y *saturation*, y pasarlos como argumentos a la biblioteca que se desee utilizar, garantizando únicamente que los arreglos extraídos estén en el formato adecuado que requiera cada biblioteca. Las modificaciones realizadas para conectar el TUNA con cualquiera de las 3 bibliotecas evaluadas (IntelMKL, CUDA, PETSc), serán descritas en el capítulo 4. En las siguientes secciones se dará cada una visión global de estas bibliotecas a evaluar.

Se realizó una simulación en TUNA del ejemplo de la sección 2.5. En las figuras 3.5 y 3.6 se muestra el avance de la interfaz agua-aceite (choque) lo cual va desplazando el aceite a lo largo del volumen. Los parámetros de la simulación fueron

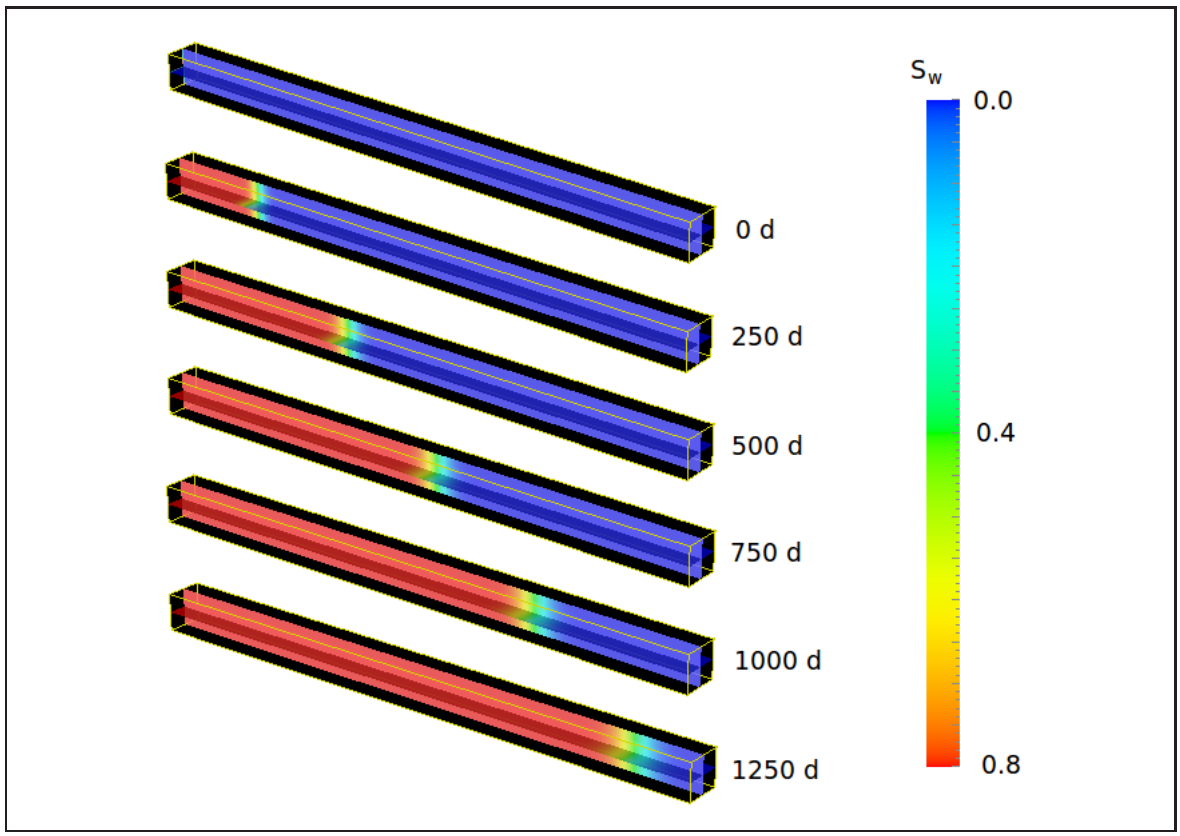

Figura 3.5: Simulación del problema Buckley-Leverett mediante el TUNA, utilizando los parámetros de la tabla 3.1. Se muestra la evolución de la interfaz agua-aceite (choque) cada 250 días, durante 1250 días. El color rojo representa la máxima saturación de agua y el azul la menor. Inicialmente está saturado de aceite. La malla en este problema contenía 262144 celdas.

3.3.2. IntelMKL

En el 2003, el fabricante de microprocesadores, Intel, sacó a la venta la Intel Math Kernel Library, la cual inicialmente era un conjunto de rutinas optimizadas para el manejo de operaciones de álgebra lineal, similares a las implementadas en BLAS y LAPACK. Explotaba la arquitectura de los procesadores Intel, haciéndoles tener un rendimiento mayor que cualquier otra implementación de dichas bibliotecas. Hoy en día, tiene un gran número de funciones, no solo de álgebra lineal, sino de otras áreas de las matemáticas como la estadística(VSL) y el análisis (FFT), entre otras aplicaciones. A la fecha, también optimiza códigos ejecutados en procesadores de otros fabricantes (AMD). Entre las herramientas que utiliza para mejorar el desempeño, está el uso de la memoria compartida de los procesadores

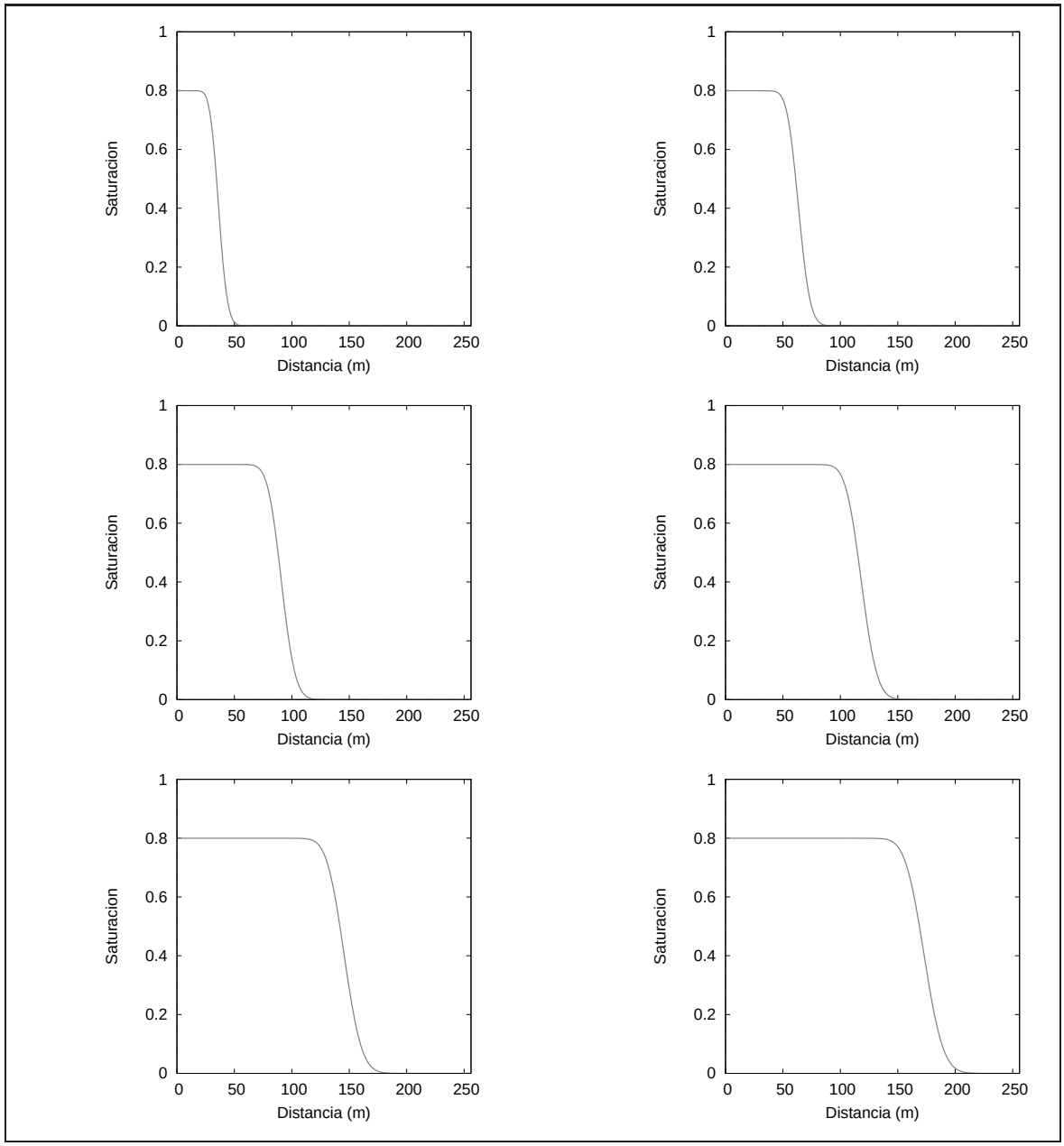

Figura 3.6: Evolución del frente de agua a lo largo de 6 tiempos (200 , 350, 500, 650, 800 y 950 días). Se tomo como línea de referencia el eje central en la dirección z.

con más de un núcleo por chip. Dicho manejo de la memoria lo realiza mediante la interfaz de programación de aplicaciones (API) OpenMP , la cual permite la paralelización de un código ejecutado sobre una máquina de memoria compartida con llamadas a directivas de dicha API. La paralelización se realiza en segmentos específicos del código a ejecutarse, dividiendo dicho segmento entre varias hebras, las cuales existen únicamente durante la ejecución de dicho segmento.

Tiene como gran ventaja, la facilidad de implementación, así como el aprovechar al máximo los recursos de una máquina multi-core, sin embargo, está limitada únicamente a este tipo de máquinas (memoria compartida). Debido al costo que implica una máquina con muchos núcleos por chip, esta tecnología no es muy recurrida en aplicaciones que requieren HPC.

3.3.3. CUDA

Con el desarrollo de la tecnología de tarjetas para el manejo de gráficos, la compañía Nvidia, liberó, en el 2007, un compilador y varias bibliotecas para utilizar dichas tarjetas como procesadores de propósito general. Debido al tipo de operaciones realizadas por dichas tarjetas, las unidades de procesamiento son mucho más sencillas y menos demandantes de energía que sus similares SISD. Esto permitió a los fabricantes de tarjetas de gráficos colocar una gran cantidad de núcleos en una sola tarjeta, a un costo muy bajo, dando como resultado un sistema de decenas e incluso centenas de núcleos instalados en una máquina convencional. Estas pequeñas computadoras con grán cantidad de procesadores, para muchas aplicaciones, obtienen un desempeño mucho mayor que otras máquinas con número similar de procesadores. Su programación está basado en modelo de memoria compartida, pero a diferencia de los procesadores SISD, estas máquinas vectoriales emplean una extensión del conjunto de instrucciones SIMD, el SPMD (single program, multiple data), en el cual, cada procesador vectorial, ejecuta el mismo programa sobre un conjunto distinto de datos. A todo este conjunto de compilador y herramientas se le denominó CUDA (Compute Unified Device Architecture), el cual aporta extensiones al lenguaje C, para la manipulación numérica de datos en el GPU, un mejor control y acceso de los bancos de memoria y poder utilizar los núcleos de la tarjeta para propósito general.

Tiene todas las ventajas de las tarjetas de gráficos (baja latencia y concurrencia), sin embargo el intercambio de memoria entre los bancos de la computadora y los bancos de la tarjeta, son en general, los cuellos de botellas en aplicaciones sobre esta arquitectura, pués actualmente no existe un mecanismo de comunicación GPU-CPU que tenga una velocidad suficiente para despreciar los tiempos de transmisión. Además, estos sistemas están muy limitados de memoria, actualmente, la tarjeta con mayor cantidad de memoria alcanza los 4 GigaBytes, excluyéndola de aplicaciones de memoria masiva. Se le puede dar vuelta al problema, utilizándolas en un cluster, pero la comunicación entre tarjetas se vuelve más lenta, al tener que transferir toda la información a las memoria de la máquina y utilizar alguna biblioteca de envió y recepción de mensajes, como MPI, para realizar la comunicación entre máquinas, y transferir la nueva información a las tarjetas. Sin embargo, computadoras de alto desempeño que utilizan este tipo de configuración hoy en día ya se pueden contar entre las más rápidas del mundo. La lista actual del top500 (mayo de 2011)[top], reporta 3 supercomputadoras entre las primeras 10 que usan este esquema (1^o Tianhe-1A, 3^o Nebulae-Dawning y 4^o Tsubame 2.0-HP).

3.3.4. PETSc

Consiste de toda una recopilación de estructuras de datos y funciones para resolver ecuaciones diferenciales. Su intención es dar todos los elementos para escribir aplicaciones científicas, de forma paralela, escalable, y ocultando al programador la programación explícita de la paralelización. Ha sido desarrollado por la Mathematics and Computer Science Division del Argonnne National Laboratory, y sus siglas vienen de Portable, Extensible Toolkit for Scientific Computation (PETSc). Esta recopilación está diseñada para ser multiplataforma, y utiliza el esquema de memoria distribuida para hacer la paralelización mediante MPI, aunque en la última versión incluye una interfaz para la utilización de tarjetas de gráficos. Posee todo un conjunto de estructuras para describir matrices y vectores, tanto densos como dispersos, en formato serial o paralelo. Tiene diversos métodos del subespacio de Krylov para la solución de sistemas de ecuaciones, varios precondicionadores, métodos de solución no lineales, entre otras características. A su vez, permite elegir la configuración de la aplicación desde la línea de comandos, permitiendo cambiar de modo paralelo a secuen-

cial, de un método de solución a otro, de un precondicionador a otro, etc, solo cambiando un parámetro en la ejecución. Como está construido sobre MPI y su enfoque es la escalabilidad, un mismo programa se puede correr en casi cualquier computadora, sin tener que modificar el código, pues PETSc genera archivos de configuración al instalarse que manejan la interacción con el sistema que lo ejecuta.

3.4. Conclusiones del capítulo

En este capítulo se explicó a grandes rasgos los distintos tipos de esquemas de memoria, y el conjunto de flujo de instrucciones que definen una arquitectura computacional, con la intención de dar elementos extra para entender las mètricas a evaluarse sobre cada distintia arquitectura. Las bibliotecas aquí mencionadas son de las más eficientes en su respectivo esquema de memoria, por lo que se puede hacer una relación directa entre cada biblioteca y la respectiv tecnología donde se ejecuta. Para equipos con muchas computadoras, o incluso para una sola con más de una capacidad paralela, conocer cuál es la mejor opción a emplear para cada situación, permité explotarlos al máximo y eficientar su uso.

Capítulo 4

Implementación

En este capítuloe describirá la implementación de cada una de las tres versiones del BiCGStab y del cálculo de la saturación, una por cada biblioteca descrita en el capítulo anterior.

4.1. Implementaciones

Cada biblioteca analizada tiene su propia definición de operaciones de álgebra lineal. La de Intel (MKL) y la de Nvidia (CUDA), son similares a las definiciones contenidas en BLAS. PETSc maneja definiciones propias. En está sección se mostrará la forma en que se implementó la parte central del BiCGStab en CUDA, PETSc e IntelMKL, mostrando las operaciones correspondientes principales del pseudocódigo del algoritmo 2.1, con la respectiva programación en las bibliotecas mencionadas. Dado que PETSc maneja internamente la implementación, no se tuvo que programar el método, solo fue necesario crear el entorno necesario para su ejecución. Sin embargo, el código de PETSc es abierto (a diferencia de MKL y CUDA), por lo que, se incluirá también el fragmento de código correspondiente. Además, para comparar contra un programa serial, se elaboró un programa escrito en una biblioteca serial eficiente para manejo de álgebra lineal (SELDON), el cual servirá como base de comparación con las bibliotecas paralelas. Una descripción de la biblioteca SELDON se puede ver en [Sel].

Todas las operaciones de álgebra lineal que aparecen en el BiCGStab son generadas

a partir de combinaciones de las operaciones mostradas en la tabla 2.1. A continuación se mostrarán fragmentos de la implementación del BiCGStab, en los que se realizan dichas operaciones, para mostrar la posible claridad, practicidad, similitudes y diferencias, entre cada una de las bibliotecas analizadas.

Entre las condiciones de paro del BiCGStab, existen dos de convergencia, cada una de las cuales se obtiene de las contracción de dos vectores $c \leftarrow \mathbf{r} \cdot \tilde{\mathbf{r}}$. Para cada biblioteca, siendo *r, rtilde*, dos estructuras de datos, cada una representando un vector del BiCGStab, y c el escalar correspondiente, se tiene las siguientes implementaciones (N el número de elementos en cada vector):

Código (C++) 4.1: Contracción vectorial

```
    /*************************************/
    /**************  CUDA ****************/

c = cublasDdot( N , rtilde , 1 , r , 1 );

    /*************************************/
    /**************  IntelMKL  ***********/

c = ddot( &N , rtilde , &inc , r , &inc );

    /*************************************/
    /**************  PETSc  **************/

      ierr = VecDot( r, rtilde, &c );

    /*************************************/
    /**************  SELDON **************/

      c = DotProd( rtilde , r );
```

En la parte final del ciclo del BiCGStab, se requiere hacer la actualización $p_{j+1} \leftarrow r_{j+1} + \beta_j(p_j + \omega_j A_{ij} p_i)$. Esta parte se puede realizar en términos de operaciones axpy's y de un escalamiento, mediante la siguiente secuencia (definiendo previamente el producto

$\underline{\underline{A}} \times \underline{p}$, como un vector temporal $\underline{Ap}$):

1. $p_{j+1} \leftarrow p_j + \omega_j Ap_j$
2. $p_{j+1} \leftarrow \beta_j p_{j+1}$
3. $p_{j+1} \leftarrow r_{j+1} + p_{j+1}$

Las implementaciones para este pseudocódigo, definiendo *p,r, Ap* las estructuras de datos correspondientes, y *omega* y *beta* los respectivos escalares, queda de la siguiente forma (N el número de elementos en cada vector)

Código (C++) 4.2: Parte final del BIiCGStab

```
/*************************************/
/**************  CUDA ****************/

cublasDaxpy( N , -omega , Ap , 1 , p , 1);

cublasDscal( N, beta , p , 1 );

cublasDaxpy( N , 1 , r , 1 , p , 1 );

/*************************************/
/**************  IntelMKL  ************/

daxpy( N , -omega , Ap , 1 , p , 1);

dscal( N , beta , p , 1);

daxpy( N , 1 , r , 1 , p , 1);

/*************************************/
/**************  PETSc  **************/

ierr = VecAXPBYPCZ( p , 1 , -omega*beta , beta , r , Ap );

                /// las operaciones axpy y escalar están definidas
```

```
                    /// en PETSc, de la siguiente manera
                    /// ierr = VecAXPY(p,beta,Ap);
                    /// ierr = VecScale (p, beta);

    /*************************************/
    /**************  SELDON **************/

    Add( -omega , Ap , P);

    Mlt( beta , P );

    Add( 1 , R , P );
```

Como se mencionó anteriormente, en cada paso del BiCGStab, se realizan 2 operaciones matriz-vector, las cuales, consumen la mayor parte del tiempo de cálculo de cada iteración. Para cada biblioteca, se utilizó el mismo formato de compresión matricial CRS (Compressed Row Storage, véase [Saad00]), el cual es uno de los formatos estándar a utilizar en este tipo de problemas. Debido a la utilización de este formato, aparece una pérdida de rendimiento, sobre la cual se hablará en la sección 2.6.3. Así, para una estructura de datos *val* que representa una matriz en formato de compresión CRS, usando dos arreglos *row*, *col* para representar su ubicación en la matriz original (en el caso de la implentación en Seldon la matriz está descrita por una única estructura *VAL*), 2 estructuras de datos *s,As* representando 2 vectores, la implementación de la multiplicación Matriz-Vector $\underline{y} = \underline{\underline{A}}\underline{x}$ en cada biblioteca se muestra en el siguiente listado:

Código (C++) 4.3: matriz por vector usando CRS

```
    /*************************************/
    /**************  CUDA ****************/

cusparseDcsrmv( handle , CUSPARSE_OPERATION_NON_TRANSPOSE
                    , N , N ,  1 , descr , val , row , col
                    , s , 0 , As );

                    /// handle,descr son descriptores de matriz
```

```
/*************************************/
/**************   IntelMKL   ************/


  mkl_dcsrgemv( transa , N , val, row , col , s , As );

                /// transa es un descriptor de matriz

/*************************************/
/**************   PETSc   **************/

  ierr = KSP_PCApplyBAorAB(ksp,s,As,T);

                /// Ksp es una entrono que contiene
                /// a la matriz VAL y T es un arreglo
                /// temporal

/*************************************/
/**************   SELDON **************/

  Mlt(VAL,s,As);
```

La última operación por mostrar es la duplicación de vectores $\underline{y} \leftarrow \underline{x}$, para la cual, definiendo las estructuras de datos x, y como los vectores a duplicar, se tiene la siguiente implementación.

Código (C++) 4.4: Duplicación de vectores

```
/*************************************/
/**************   CUDA ****************/

dcopy( N , x , 1 , y , 1 );

/*************************************/
/**************   IntelMKL   ************/
```

```
    dcopy( N, x , 1 , y , 1 );

  /*************************************/
  /**************  PETSc  **************/

    ierr = VecCopy( r, rtilde );

  /*************************************/
  /**************  SELDON **************/

    Copy( x , y );
```

Para el cálculo de la saturación, según la ecuación 2.82, se requieren dos axpys y una operación matriz vector, las cuales ya han sido descritas dentro de las operaciones del BiCGStab.

4.1.1. Modificaciones específicas a cada biblioteca

Debido a los distintos enfoques de manejo de memoria y tipos de arquitectura de cada biblioteca, en cada simulación se tuvo que agregar código específico para cada una y así asegurar la correcta comunicación con la respectiva interfaz. Estas modificaciones al programa original (escrito únicamente con rutinas del TUNA), no cambiaron su flujo natural ni las condiciones del problema. Estas modificaciones se explican a continuación:

1. En los casos de Seldon e IntelMKL, no fue necesario agregar mucho código , ya que solo era necesario cambiar las llamadas al método de solución respectivo para la presión y la saturación, sin tener que modificar las estructuras de datos que contenían la información del problema (la matriz y los dos vectores del sistema lineal $\underline{\underline{A}}\underline{x} = \underline{b}$ a resolver), pues ambas bibliotecas trabajan directamente con dichas estructuras, una por ser serial (Seldon) y la otra por manejar memoria compartida (IntelMKL).

2. En el caso de CUDA, la información contenida en cada estructura del sistema lineal, reside originalmente en la memoria del CPU, y debe ser enviada a la memoria del GPU para ser procesada (de forma paralela) y enviada de vuelta a los bancos de memoria

del CPU, para que los datos operados sigan con la ejecución del IMPES en el contexto del TUNA. Para esto se agregaron funciones para portar la estructura de datos a un formato manejable por CUDA, y para el envió y recepción correspondiente.

3. En el caso de PETSc, la solución del sistema lineal se realiza de forma paralela sobre un esquema de memoria distribuida, lo cual requiere que el sistema lineal sea segmentado y distribuido a los procesadores que intervengan en la simulación. Además, toda la simulación se realiza de forma paralela (existe una copia del programa por cada procesador usado). Esto crea un problema con el enfoque serial del TUNA, ya que este realiza la creación, ensamblado y actualizaciones de todas sus estructuras de datos de forma serial, y dicha forma queda oculta al programador, impidiendo que cada procesador genere de forma explícita la parte respectiva de cada estructura global (se tendría que reescribir completamente el TUNA para lograr esto). La manera en que se resolvió esto, fue imponiendo que solo un procesador (raíz) generará las estructuras de datos, y una vez listas para ser utilizadas en la solución de la presión y de la saturación, dichas estructuras se dividieran y fueran enviadas a los demás procesadores para utilizar las rutinas de PETSc. Una vez que la biblioteca resolvía (de forma paralela) el sistema, este era devuelto al procesador raíz, y se seguía con el código relativo al TUNA.

Debido a que al utilizar CUDA y PETSc fue necesaria la introducción de operaciones extra (cambios de formato, envío y recepción de información), no puede compararse el desempeño de las bibliotecas midiendo el tiempo total de ejecución de la simulación, pues cada ejecución requirió de un tiempo distinto, específico a cada biblioteca. El objetivo de este trabajo es comparar el rendimiento de dichas bibliotecas en condiciones similares. Al utilizar el TUNA se garantizó que el problema fuera el mismo (dimensiones, tiempo, condiciones iniciales, discretización, etc), pero esto requirió distintas implementaciones con distintos tiempos de configuración. La forma en que se midió el rendimiento fue tomando únicamente el tiempo empleado en resolver el sistema de ecuaciones para la presión y la saturación, desde que se inicia la comunicación (para el caso de CUDA y PETSc), hasta que se regresa dicha información a la secuencia original del TUNA. Una vez conocida cuál bib-

lioteca es más conveniente, se podría escribir toda una aplicación (TUNA por ejemplo) con base en dicha biblioteca para evitar cambios de formato y demás configuraciones relativas a la interfaz.

4.1.2. Métricas

Definiremos de forma sencilla FLOP como la cantidad de operaciones algebraicas mínimas (suma y multiplicaciones) que se realizan para completar alguna operación vectorial de algebra lineal, dividida entre el tiempo tomado en realizarlas. Por ejemplo en una contracción de dos vectores de tamaño N, si se realizan N multiplicaciones $+N-1$ sumas, dando un total de $2N-1 \approx 2N$ operaciones de punto flotante en T segundos, este cálculo empleó $2N/T$ FLOPS.

En general, la aceleración (S) o *speedup* se define como

$$S = \frac{T^{ser}}{T^{par}}, \tag{4.1}$$

con T^{ser} y T^{par} los tiempo totales de ejecución serial y paralelo respectivamente. Para este caso, se definirá una aceleración S^* similar a la anterior, pero los tiempos T^{ser} y T^{par} ya no serán los totales, sino solo los tiempos empleados en calcular la presión (vía el BiCGStab) y la saturación (explícita). Definiendo el tiempo de cálculo serial (paralelo) de la presión como t^{ser}_{pres} (t^{par}_{pres}), y el tiempo de cálculo serial (paralelo) de la saturación como t^{ser}_{sat} (t^{par}_{sat}), la aceleración S^* queda

$$S^* = \frac{T_{ser}}{T_{par}} = \frac{t^{ser}_{pres} + t^{ser}_{sat}}{t^{par}_{pres} + t^{par}_{sat}}. \tag{4.2}$$

Capítulo 5

Resultados y Análisis

En este capítulo se presenta los resultados obtenidos en la evaluación de las métricas de desempeño realizadas a las tres bibliotecas paralelas descritas en el capítulo 3.3, usando como referencia una biblioteca serial optimizada (Seldon). Este capítulo está divido en dos secciones. La primera muestra los resultados obtenidos para el desempeño, ya sea en FLOPS o en segundos, de las implementaciones de las 4 bibliotecas, para:

1. las operaciones vectoriales que componen el BiCGStab,
2. el tiempo promedio de cálculo para la presión (BiCGStab) en cada paso del IMPES,
3. el tiempo promedio del cálculo explicito de la saturación en cada paso del IMPES,
4. la aceleración asociada a cada biblioteca paralela.

Los resultados mostrados son del cálculo de la presión y la saturación utilizando el TUNA como programa principal, haciendo la correspondiente interfaz a cada biblioteca para resolver los sistemas lineales generados por el IMPES, garantizando así las mismas condiciones iniciales en la evolución de cada simulación.

5.1. Resultados

Se utilizó una máquina con procesador Intel Xeon de 64 bits con 8 núcleos a 3.7 GHz cada uno, con 4 GB de memoria RAM. La tarjeta de gráficos empleada (montada en

la misma máquina) fue una NVIDIA TESLA C1060, con 4 GB de memoria RAM, con 30 Multiprocesadores (MP), cada uno de estos con 8 núcleos, dando un total de 240 CUDA cores.

Dada las limitantes establecidas por la memoria RAM (4GB) y el número de núcleos por procesador (8 en el caso de IntelMKL en el esquema de memoria compartida), se limitó la ejecución de la simulación utilizando la biblioteca PETSc (la cual puede ser ejecutada en supercomputadoras con mas 100000 núcleos) a únicamente los 8 núcleos de la máquina utilizada. Esto para comparar el desempeño en igualdad de circunstancias (al menos en el tamaño de memoria).

Para cada operación vectorial, se realizaron mediciones para dos tamaños distintos de vectores, 262144 ó 524288 entradas. Para el caso de la operación matriz por vector, la matriz fue de 262144×262144 ó 524288×524288 entradas.

En el caso de las bibliotecas evaluadas, tanto PETSc como IntelMKL, permiten variar el número de núcleos empleados en la ejecución de un programa. Para ambas bibliotecas, se midió el rendimiento obtenido al realizar cada una de las operaciones vectoriales del BiCGStab (ver tabla 2.1), utilizando distintas cantidades de núcleos (1, 2, 4 y 8), cada núcleo se asignó una hebra, de modo que, de aquí en adelante, cada vez que se refiera a que una simulación fue ejecutada sobre N núcleos, el numéro de hebras ejecutado será también N.

A continuación se muestra un listado con las gráficas de los resultados obtenidos

1. Los resultados da las operaciones axpy, contracción y escala para la biblioteca IntelMKL, utilizando 1, 2, 4 y 8 núcleos son mostrados en la figura 5.1.

2. Los resultados para el caso de la biblioteca PETSc (axpy, contracción y escala), utilizando 1, 2, 4 y 8 núcleos, son mostrados en la figura 5.2.

3. Una vez obtenida la cantidad de núcleos óptima para las bibliotecas IntelMKL y PETSc, se comparara en las figuras 5.3 y 5.4 dichos valores (con número óptimo de núcleos) con los resultados de las bibliotecas CUDA y Seldon, para las operaciones evaluadas en las figuras 5.1 y 5.2.

4. La duplicación vectorial no realiza operaciones de punto flotante, sin embargo, su realización en dos ocasiones dentro del ciclo principal del BiCGStab, requiere de un tiempo considerable. Los resultados obtenidos en el tiempo de ejecución de esta instrucción para las cuatro bibliotecas analizadas, son mostrados en las figuras 5.5.

Una vez evaluado el desempeño para cada operación vectorial dentro del BiCGStab, se procedió a medir el tiempo empleado en cada llamada al BiCGStab y al cálculo explícito de la saturación por cada paso del IMPES, empleando las distintas implementaciones de estas rutinas. Se calculó el tiempo promedio por cada iteración del IMPES.

1. Los resultados obtenidos en el tiempo promedio de cálculo de la presión (BiCGStab) por iteración del IMPES para dominios problemas con 262144 y 524288 celdas son mostrados en la figura 5.6.

2. Los resultados para el tiempo de cálculo de la saturación por iteración del IMPES son mostrados en la figura 5.7.

3. Para cada ciclo del IMPES, la suma de los tiempos de cálculo de la presión t_{pres}^{par} y la saturación t_{sat}^{par} (véase subsección 4.1.1), permiten calcular la aceleración S^*, definida en la ecuación 4.2, tomando como T^{ser} el tiempo secuencial de la biblioteca Seldon. Los resultados son mostrados en la figura 5.8.

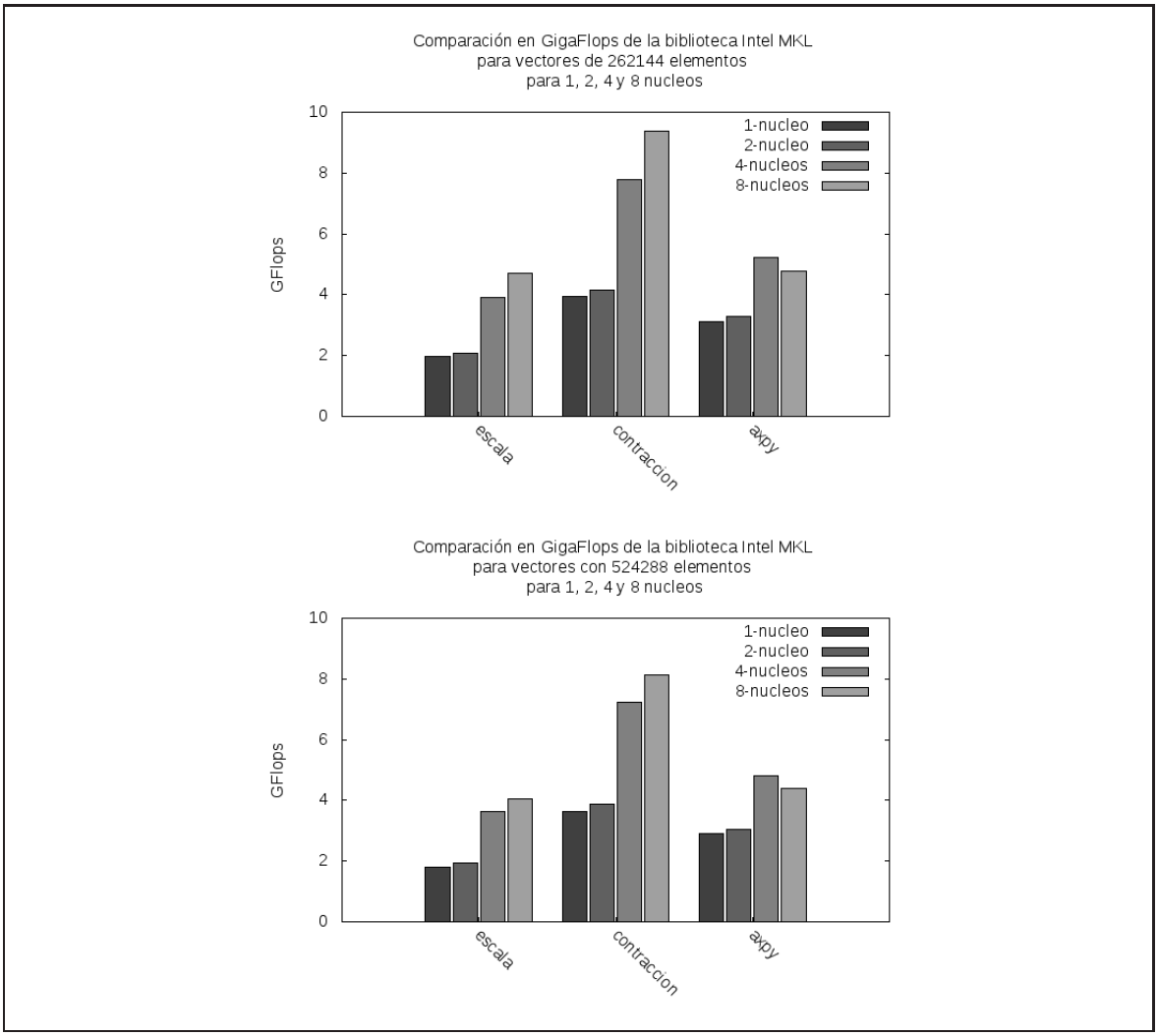

Figura 5.1: Desempeño en FLOPS obtenido por la biblioteca IntelMKL en las operaciones axpy, contracción y escala. Se evaluó cada operación para 1, 2 , 4 y 8 núcleos, realizando 1000 promedios para cada una. No se obtuvo un resultado general sobre el número óptimo de núcleos a emplear. Para el axpy el número óptimo es 4 núcleos, para contracción y escala, el óptimo fue 8 núcleos.

5.2. Análisis

Para las bibliotecas en las cuales se puede variar el número de núcleos (IntelMKL y Petsc), los resultados muestran el cambio de rendimiento al variar dicho número, lo cual tiene consecuencias importantes en la optimización de aplicaciones, al no ser necesaria en todos los casos la utilización de todo un procesador (varios núcleos) para obtener mejor

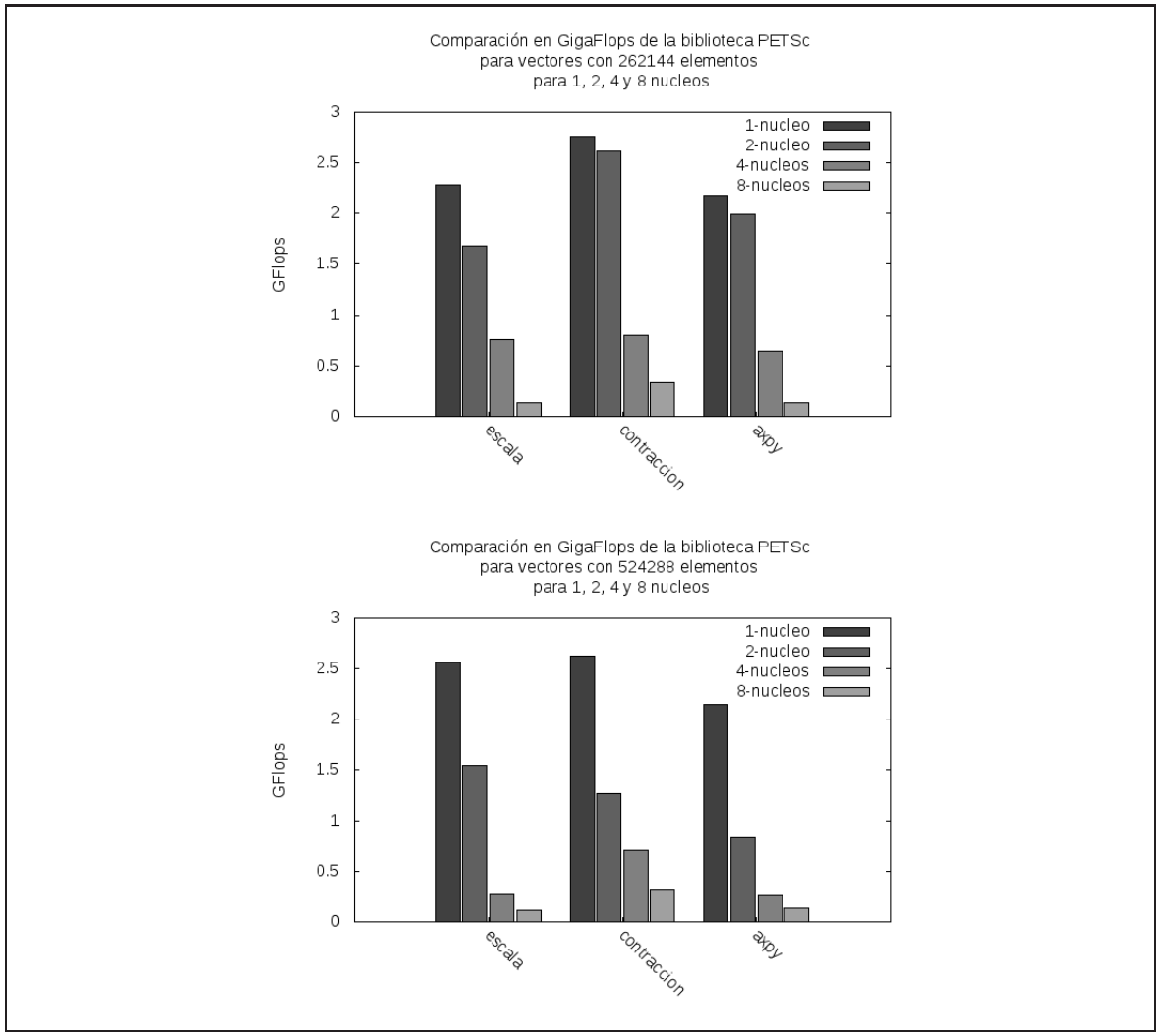

Figura 5.2: Desempeño en GFLOPS obtenido por la biblioteca PETSc en las operaciones axpy, contracción y escala. Se evaluó cada operación para 1, 2 , 4 y 8 núcleos, realizando 1000 promedios para cada una. El desempeño de está biblioteca decae conforme el número de núcleos utilizados crece.

desempeño. Además, el número óptimo de núcleos puede cambiar de una operación vectorial a otra, por ejemplo, el número óptimo de núcleos para el axpy (4) y para la contracción (8) difieren (figura 5.1).

A diferencia de sistemas de memoria compartida, como los que explota IntelMKL, en sistemas distribuidos la comunicación de datos es un cuello de botella, y en general el rendimiento disminuye conforme aumenta el número de núcleos si la operación requiere del

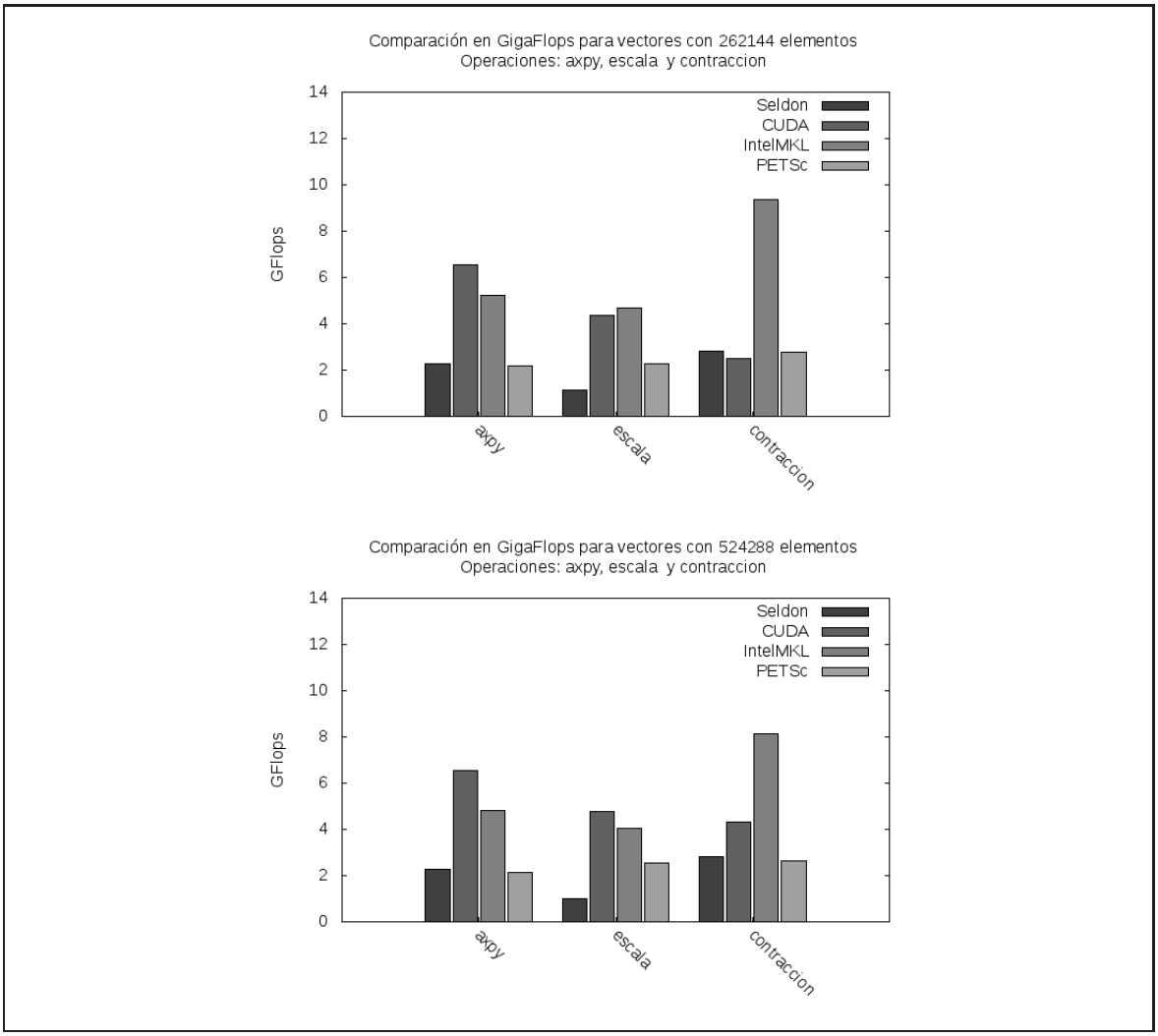

Figura 5.3: Comparación en el desempeño (GFLOPS) de las 4 bibliotecas, para las operaciones axpy, contracción y escala. Para axpy el mejor desempeño lo obtuvo CUDA, y para las otras dos operaciones fue IntelMKL la de mejor resultado. Para las bibliotecas PETSc e IntelMKL, se utilizaron los mejores resultados obtenidos en las gráficas 5.1 y 5.2. Se realizaron 1000 promedios por cada resultado.

envió e intercambio de información. Incluso, para un mismo número de núcleos, el aumentar el tamaño del conjunto de datos a operar, puede disminuir el rendimiento del sistema, como puede observarse en la relación entre el desempeño de uno y dos núcleos en el axpy de la figura 5.2.

No hubo un resultado general para todas las operaciones, aunque los mejores de-

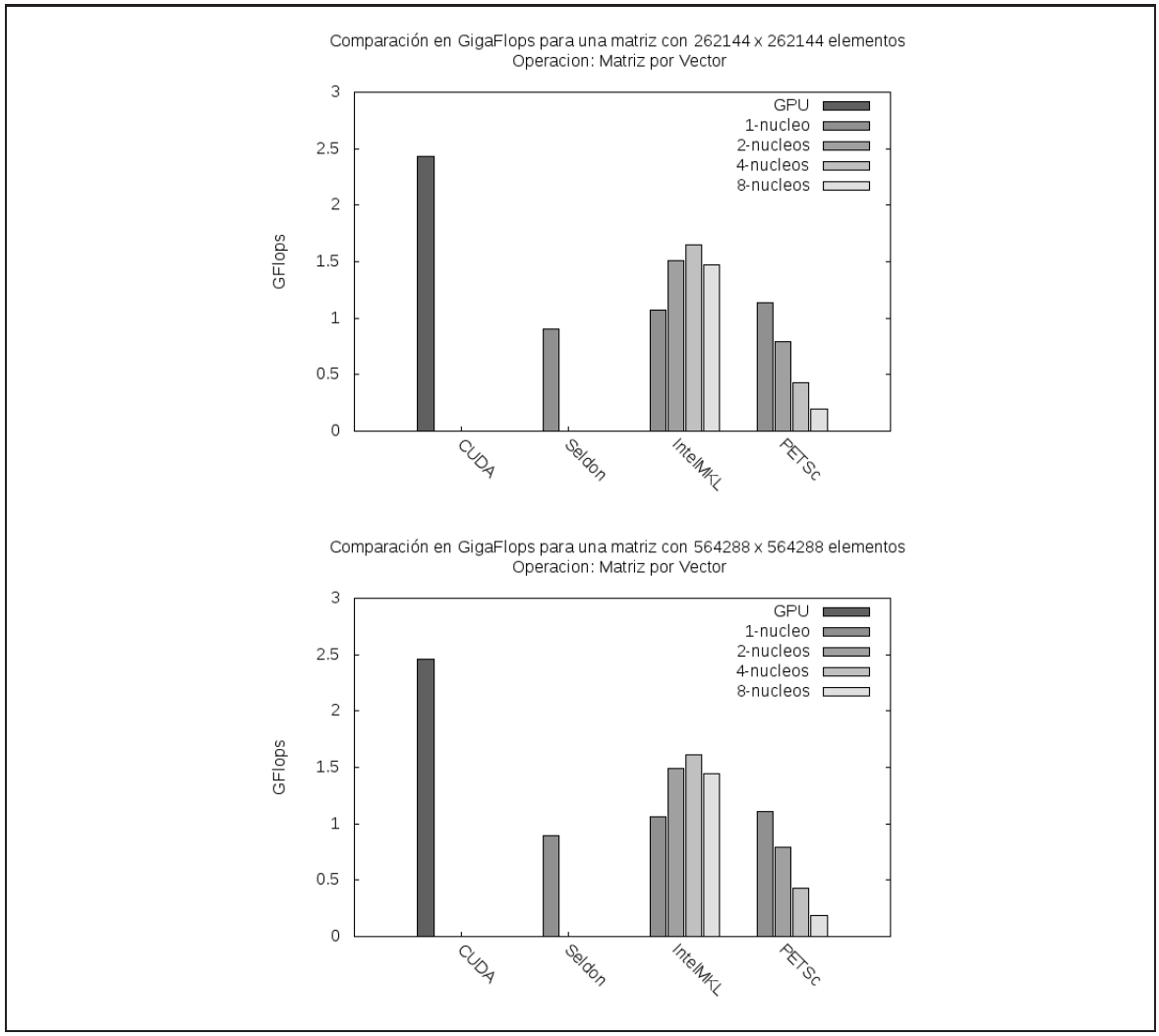

Figura 5.4: Comparación en el desempeño (GFLOPS) de las 4 bibliotecas, para la operación matriz por vector. Para esta operación, la cual es más demandante de tiempo, fue CUDA la biblioteca con mejor desempeño. Para las bibliotecas PETSc e IntelMKL, se utilizaron los mejores resultados obtenidos en las gráficas 5.1 y 5.2. Se realizaron 1000 promedios por cada resultado.

sempeños fueron obtenidos por las bibliotecas CUDA e IntelMKL. Para las operaciones matriz-vector y axpy, CUDA obtuvo el mejor desempeño, siendo más notorio para la primera. Este resultado es importante, ya que dentro de cada ciclo del BiCGStab, las dos multiplicaciones matriz por vector son las que consumen el mayor tiempo de cálculo. Además, dentro de cada ciclo se realizan 6 axpy's (ver tabla 2.1), los cuales también requieren una cantidad importante de tiempo al realizarse. Esto muestra la conveniencia

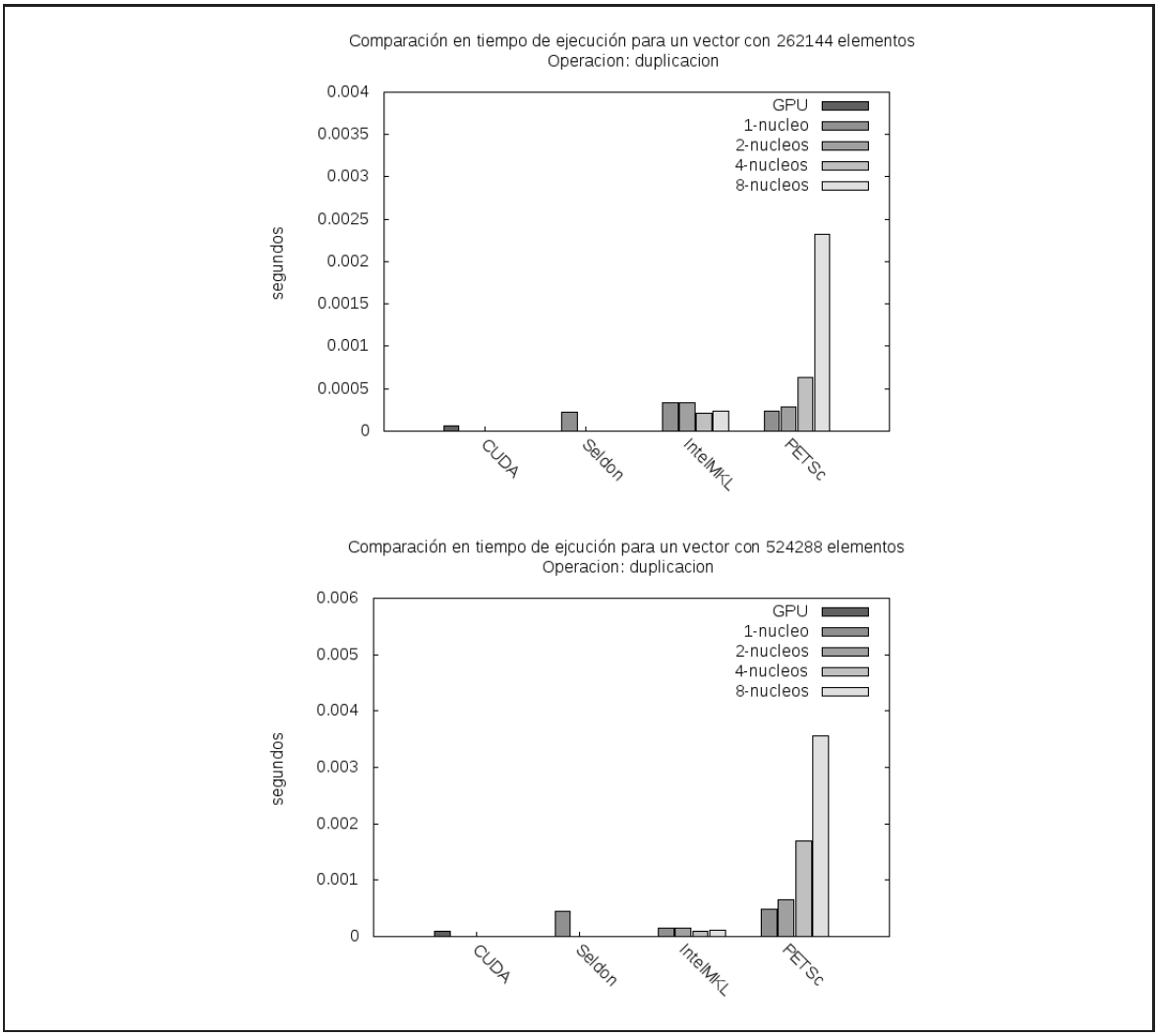

Figura 5.5: Comparación en el tiempo (segundos) de las 4 bibliotecas, para la duplicación vectorial. No hubo una diferencia notoria entre los tiempos para las bibliotecas IntelMKL y CUDA, las cuales reportaron los menores tiempos. Se realizaron 1000 promedios por cada resultado.

del uso de GPU's en problemas similares al descrito en este trabajo (ver capítulo 2).

Para la operación vectorial escala, las bibliotecas CUDA e IntelMKL tuvieron desempeños similares, y solo para la operación de contracción, IntelMKL tuvo una diferencia considerable. Esto sugiere el uso de distintas bibliotecas dentro de cada ciclo, lo cual podría mejorar el desempeño al explotar las operaciones óptimas de cada biblioteca. Sin embargo, se debe tomar en cuenta que los datos que opera el GPU deben estar en la memoria RAM del GPU, lo cual plantea un problema al momento de querer realizar operaciones del BiCGStab

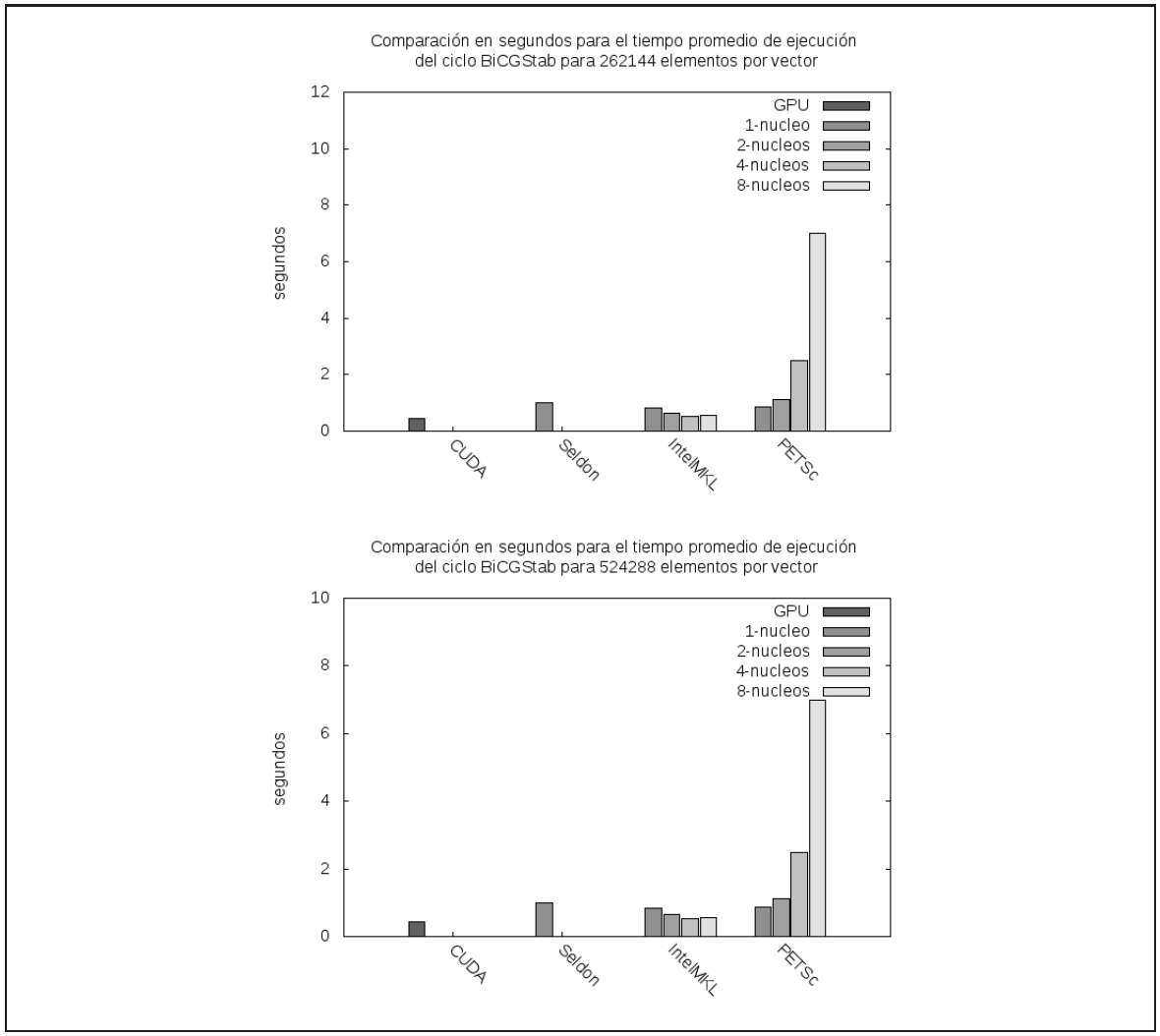

Figura 5.6: Comparación en el tiempo (segundos) de las 4 bibliotecas, del tiempo promedio empleado en el cálculo de la presión (BiCGStab) en cada paso del IMPES. Para este caso, fue la biblioteca CUDA la de mejor desempeño. Se realizó toda una simulación del problema de Buckley-Leverett utilizando TUNA y se promedió durante 1000 iteraciones del IMPES.

con otras bibliotecas (las cuales emplean datos de la memoria RAM en la computadora).

Dentro de los esquemas de memoria, solo la memoria compartida evita el traslado de información entre procesadores. Para la biblioteca PETSc la cual utiliza memoria distribuida, se midieron los tiempos de envío desde el núcleo principal (el cual generaba los vectores y matrices originales) a los demás núcleo. Este tiempo debe ser tomado en cuenta ya que se realiza dos veces en cada paso del IMPES, al inicio y fin ciclo del BicGStab. Sin

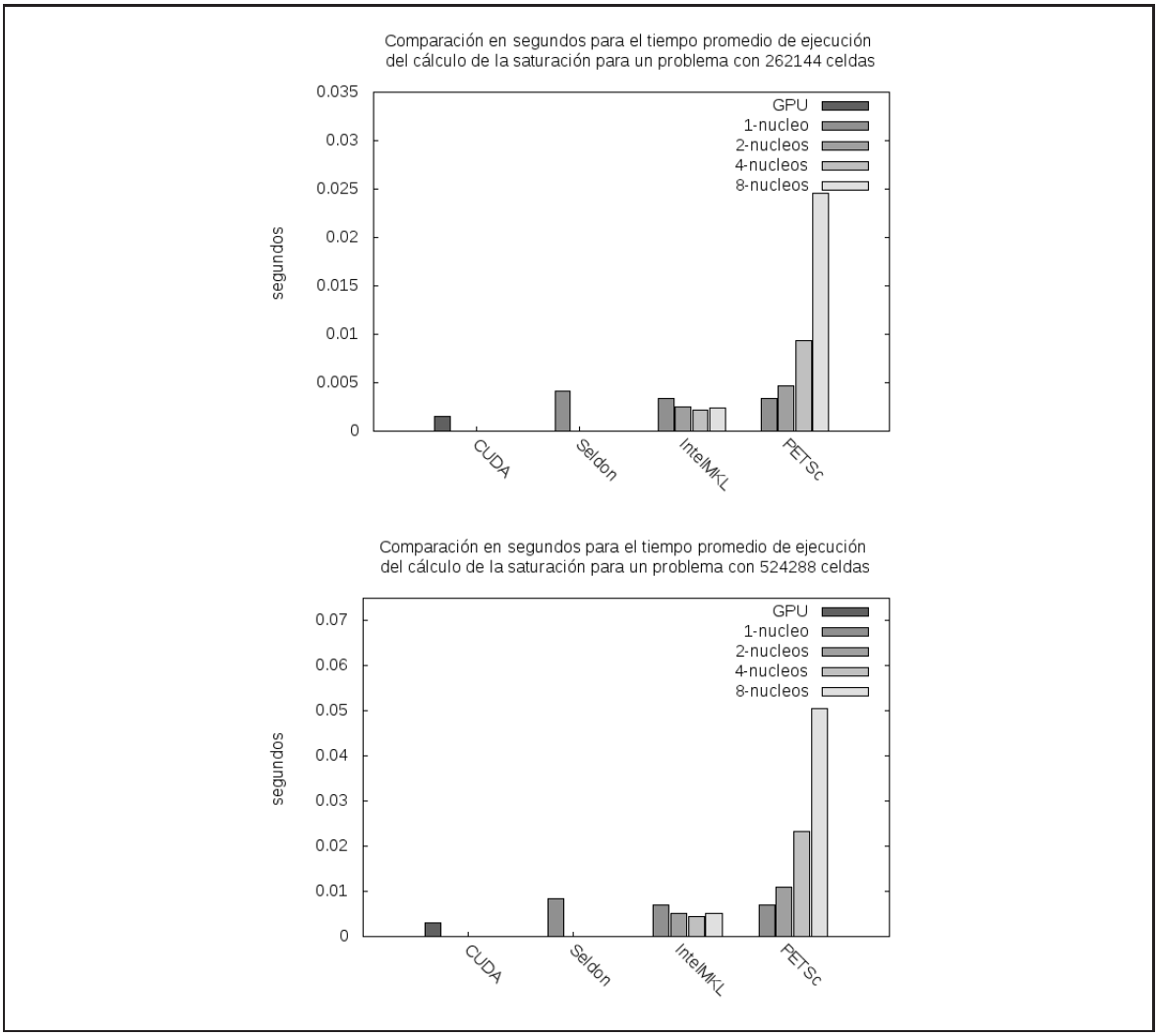

Figura 5.7: Comparación en el tiempo (segundos) de las 4 bibliotecas, del tiempo promedio empleado en el cálculo explícito de la saturación en cada paso del IMPES. Para este caso, fue la biblioteca CUDA la de mejor desempeño. Se realizó toda una simulación del problema de Buckley-Leverett utilizando TUNA y se promedió durante 1000 iteraciones del IMPES.

embargo, el tiempo obtenido de transmisión de datos para está biblioteca fue muy pequeño ($\approx 10^{-6}$ segundos), de modo que se puede considerar despreciable. En los casos de Seldon (secuencial) e IntelMKL (memoria compartida) la información necesaria es accesible a los núcleos, de modo que no es necesario el intercambio de información.

Para la biblioteca CUDA, al principio y al final de cada llamada al BiCGStab se debe transferir estructuras de datos entre la memoria RAM del procesador y la memoria

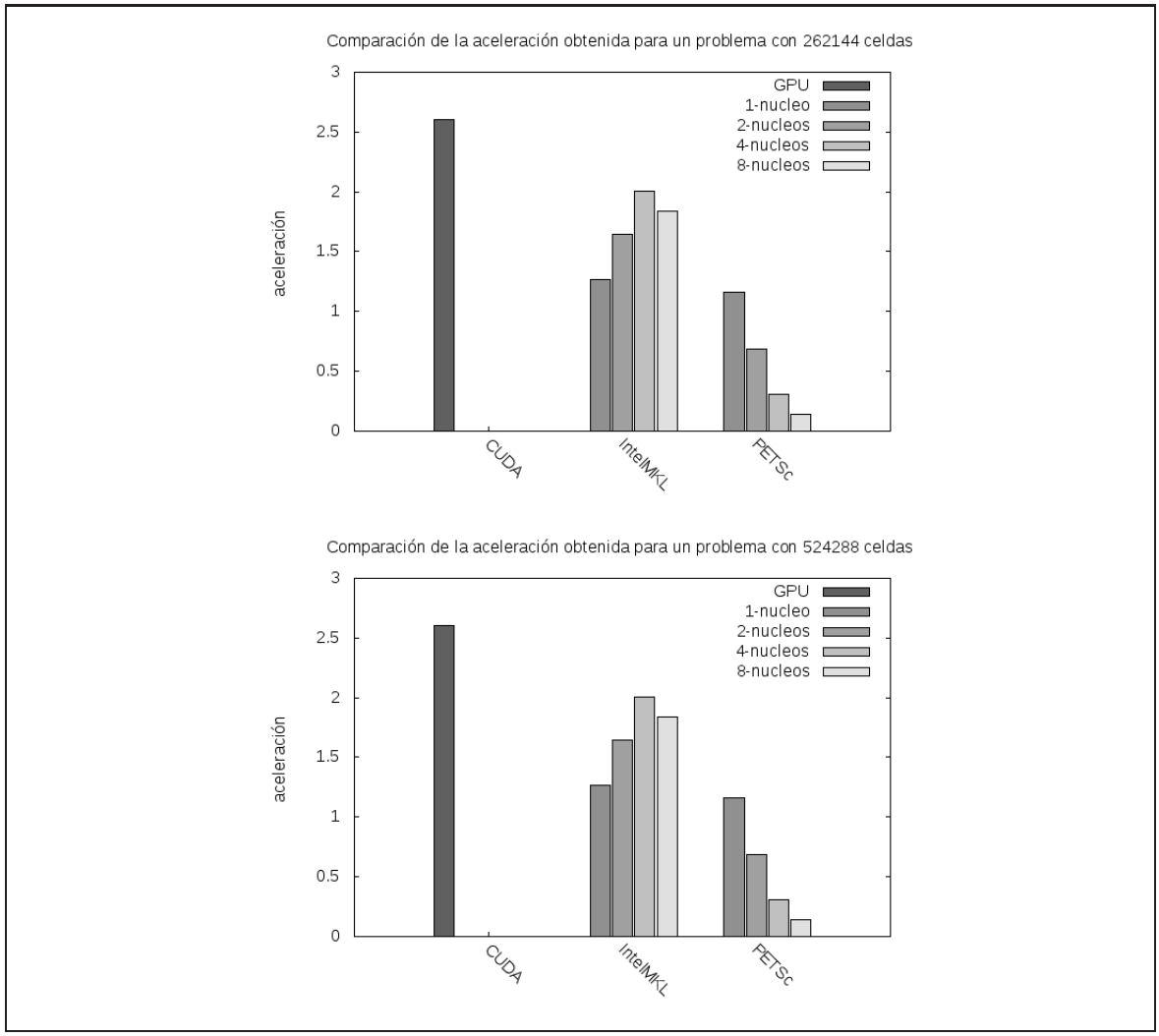

Figura 5.8: Aceleración de las tres bibliotecas paralelas, usando como referencia la biblioteca serial Seldon, a partir de la suma de los tiempos promedio del cálculo de la saturación y de la presión en cada paso del IMPES. La biblioteca CUDA tuvo la mejor aceleración. Se realizó toda una simulación del problema de Buckley-Leverett utilizando TUNA y se promedió durante 1000 iteraciones del IMPES.

RAM del GPU. En este caso, el tiempo ya no fue despreciable. Una transferencia de 262144 elementos desde los bancos de memoria del CPU a los del GPU , tomó 0.0005 segundos. Con la misma cantidad de elementos, el tiempo medido para enviar la información del GPU al CPU tomó 0.006 segundos.

Para este caso, el ancho de banda $B_w^{\rightarrow}$ del CPU al GPU fue:

$$
\begin{aligned}
B_w^{\rightarrow} =& \frac{\text{bytes transmitidos al GPU}}{\text{tiempo de transmisión}} \\
=& \frac{262144 \text{ elementos} * \frac{8 \text{ bytes}}{\text{elemento}}}{0.0005\text{seg}} \\
=& 4194304000 \frac{\text{bytes}}{\text{seg}} = 4.194 \frac{\text{Mbytes}}{\text{seg}}.
\end{aligned} \tag{5.1}
$$

y para el ancho de banda $B_w^{\leftarrow}$ del GPU al CPU se obtuvo:

$$
\begin{aligned}
B_w^{\leftarrow} =& \frac{\text{bytes transmitidos del GPU}}{\text{tiempo de transmisión}} \\
=& \frac{262144 \text{ elementos} * \frac{8 \text{ bytes}}{\text{elemento}}}{0.0006\text{seg}} \\
=& 3.495 \frac{\text{Mbytes}}{\text{seg}}.
\end{aligned} \tag{5.2}
$$

Los anchos de banda medidos $B_w^{\rightarrow}$ y $B_w^{\leftarrow}$ fueron cercanos a los anchos de banda máximos indicados por el fabricante (4946 Mbytes/s y 4231 Mbytes/s respectivamente). El tiempo de transmisión entre los dos dispositivos (CPU y GPU) en general es omitido, y sin este, la aceleración que comúnmente se reporta en la literatura por el uso de GPU's puede ser mucho mayor de lo que en realidad se obtiene, si dichas aplicaciones requieren constantes intercambios de grandes cantidades de información entre dispositivos. Para el caso estudiado, el tiempo de transmisión entre los dispositivos fue del orden de la operación matriz por vector (0.00123 segundos para la multiplicación y $0.0005 + 0.0006 = 0.0011$ segundos para la transmisión de un vector de 262144 elementos de ida y vuelta del CPU al GPU).

Para cada iteración del IMPES, los tiempos de ejecución en los cálculos de la presión y de la saturación mostrados en las gráficas 5.6 y 5.7 arrojaron que la biblioteca CUDA tuvo el mejor desempeño. Es importante recalcar que esta biblioteca tiene la desventaja de requerir cierto tiempo $t^{\leftrightarrow}$ para intercambiar 2 veces información entre el CPU y el GPU en cada paso del IMPES, una por el cálculo de la presión y una por el de la saturación. En el caso de la presión, fue necesario subir a la tarjeta gráfica dos vectores y una matriz, lo cual tomó un tiempo $t^{\leftrightarrow}$, de 0.0099 segundos para el problema de 262144 celdas, y de

0.0187 segundos para el de 524288 celdas. En el caso de la saturación, se requirió transmitir 3 vectores y una matriz, tomando un tiempo $t^{\leftrightarrow}$ de 0.011 segundos para el problema de 262144 celdas y 0.021 segundos para el de 524288 celdas.

Si el tiempo que permanece el programa dentro del es mucho mayor que el tiempo de transmisión $t^{\leftrightarrow}$, este no disminuirá mucho la aceleración S^*. En la solución de la presión, el tiempo de cálculo $t^{par}_{pres} = 0.443seg \approx 44.8t^{\leftrightarrow}$, lo cual no repercute mucho en su aceleración. Sin embargo, para la saturación, el tiempo de cálculo fue menor que el de transmisión ($t^{par}_{sat} = 0.00151seg \approx 0.14t^{\leftrightarrow}$), lo cual no puede ser despreciado. A pesar de esto, el tiempo de cálculo de la saturación es muy pequeño con respecto al tiempo tomado en el cálculo de la presión, por lo que la aceleración total (cálculo de presión y saturación) está determinada casi totalmente por la aceleración que en el cálculo de la presión se obtenga. Así, las aceleraciones mostradas en la figura 5.8, determinaron los resultados finales de todo esté trabajo, siendo CUDA la biblioteca de mayor desempeño, seguida por IntelMKL, y finalmente PETSc.

Capítulo 6

Conclusiones

Este trabajo se realizó principalmente con la intención de comparar el uso de GPU's con las tecnologías paralelas actuales para fenómenos relacionados a la dinámica de fluidos en medios porosos (problema de Buckley-Leverett). Para este problema, se generan matrices dispersas cuya representación comprimida, almacenando únicamente elementos no nulos, es conveniente para el ahorro de recurso computacionales. Sin embargo, esta representación genera que la velocidad de lectura límite el desempeño de los equipos (paralelos o no). Con el desarrollo actual de la tecnología de unidades de memoria y procesamiento, en una estación de trabajo de costo moderado, se pueden estudiar problemas de gran tamaño, por lo que trabajos como éste, permiten conocer cuál es la tecnología más conveniente a usar si se dispone de un equipo de alto desempeño.

La tendencia actual de la tecnología de microprocesadores se enfoca a la inclusión de más núcleos en un solo procesador. Sin embargo, ésto no necesariamente lleva a una mejora en el desempeño de una aplicación paralela, ya que existe un número óptimo de núcleos para cada operación. En los resultados para la comparación del desempeño de las bibliotecas IntelMKL y PETSc en función del número de núcleos (véase figuras 5.1 y 5.2 respectivamente), es notoria la variación en el desempeño, y no necesariamente crece al aumentar el número de núcleos. Esto es algo que debe tomarse en cuenta en la utilización de supercomputadoras tipo cluster (las cuales poseen el mejor desempeño dentro del cómputo científico paralelo), ya que estas generalmente consisten en una colección de nodos (proce-

sadores), cada uno con cierta cantidad de núcleos, y conocer el desempeño de cada nodo en función del respectivo número de núcleos, se puede traducir en una mejora al desempeño de una aplicación paralela a gran escala.

Dentro de cada paso del IMPES, la mayor parte del tiempo de ejecución se realiza dentro del cálculo implícito de la presión. En la solución iterativa de la presión se utilizó el BiCGStab, y en éste, las 2 operaciones matriz por vector, los 6 axpys y las 6 contracciones vectoriales(tabla 2.1) son las que requieren más tiempo en ser completadas. Para la operación matriz vector y el axpy, la biblioteca CUDA (GPU's) obtuvo el mejor desempeño, lo cual se ve reflejado en el tiempo promedio por paso del BiCGStab y la aceleración obtenida en el ciclo IMPES (véase figuras 5.3 y 5.4 respectivamente). Sin embargo, para la contracción vectorial, fue el uso de memoria compartida (IntelMKL) la que obtuvo el mejor desempeño. Para la solución de la saturación, debido a su forma explícita, el tiempo de cálculo es muy pequeño comparado con el de la presión. Si no se toma en cuenta la transmisión de datos entre el GPU y el CPU, el tiempo de cálculo obtenido para CUDA fue superior nuevamente. Sin embargo, el tiempo de transmisión para este caso (≈ 0.011 seg) fue mayor al tiempo de cálculo (≈ 0.0015), de modo que tomando en cuenta estos dos factores, el tiempo total empleado en obtener la solución explícita de la saturación fue mayor al de la biblioteca IntelMKL (≈ 0.0022 seg). Esto muestra que la transmisión de datos entre el GPU y el CPU es un factor que puede afectar de manera importante al desempeño, si el tiempo de cálculo en el GPU es del orden del tiempo de transmisión. Los resultados para la presión y la saturación nos permiten plantear una regla para la utilización de GPU's en simulaciones de problemas parecidos al aquí analizado:

Para obtener en un GPU un desempeño mayor que el de un sistema de memoria compartida, se debe realizar la mayor cantidad posible de procesamiento con la información almacenada en los bancos de memoria del GPU, de modo que el tiempo de transmisión de información sea despreciable con respecto al tiempo empleado en dicho procesamiento.

Dado que en cada iteración del BiCGStab la contracción vectorial es una parte importante (empleada 6 veces por cada iteración), esto sugiere la posibilidad de utilizar una combinación de tecnologías dentro del BiCGStab, empleando CUDA (GPU) para las operaciones matriz-vector y axpy, e IntelMKL (memoria compartida) para la contracción

vectorial, buscando disminuir el tiempo de cálculo de la presión, siempre y cuando el tiempo combinado de transmisión de los vectores entre el GPU y el CPU, y su contracción vectorial dentro del CPU usando memoria compartida (IntelMKL), cumplan la regla anterior.

En general, de los resultados del capítulo 5, sobre el desempeño de las tres bibliotecas evaluadas, para el caso de Buclkely-Leverett, y parecería indicar que en general para problemas que involucren matrices dispersas, se puede concluir que:

- el uso de GPU's mejora el desempeño con respecto a los otros esquemas de memoria evaluados (paralelos o no), obteniéndose aceleraciones S^* alrededor de $2.6X$ para el tiempo total de cálculo de cada paso del IMPES (véase figura 5.8),

- la biblioteca IntelMKL, la cual explota el esquema de memoria compartida, tuvo la segunda mejor aceleración ($\approx 2X$), y para algunas operaciones (contracción vectorial y duplicación) el mejor desempeño. Es importante señalar que el número óptimo de núcleos para cada operación vectorial fue distinto.

- PETSc, mostró un desempeño menor con respecto a las otras bibliotecas, debido principalmente a su enfoque de memoria. Está diseñado para facilitar y optimizar la comunicación de grandes cantidades de información entre un gran número de procesadores, y comparada en aplicaciones locales (en un mismo CPU), no puede competir mucho con otras tecnologías que explotan dicha localidad.

Trabajo futuro

1. Programar un nuevo TUNA que corra principalmente en GPU's , para evitar tiempos de transmisión.

2. Dentro del modelo de Buckley- Leveret, utilizar otros esquemas y técnicas para atacar los choques.

3. Comparar con otros formatos de compresión matricial.

4. Aumentar la complejidad del problema a resolver, como puede ser el modelo de petróleo negro y otros múltifásicos.

5. Comparar el desempeño en equipos más grandes, tanto de memoria compartida, clusters de memoria distribuida, clusters híbridos (con dos e incluso tres de las tecnologías aquí evaluadas), para obtener más información sobre la escalabilidad.

Referencias

[Buckley41] Buckley, S. y Leverett, M. Mecanism of lfuid displacement in sands. *American Institue for Mining and Metallurgical Engineeers*, 1337, 1941.

[Chen07] Chen, Z. *Reservoir simulation. Mathematical techniques in oil recovery.* SIAM, 2007.

[Crank96] Crank, N. P., J. A practical method for numerical evaluation of solutions of partial differetnial equations of the heat conduction type. *Advances in computational mathematics*, 6 : 207-226, 1996.

[Datta-Gupta07] Datta-Gupta, A. y King, M. J. *Streamline simulation: theory and practice.* SPE Textbooks Series, 2007.

[Fen] Fenics project. http://www.fenicsproject.org/.

[Flynn72] Flynn, M. J. Some computer organizations and their effectiveness. *IEEE TRANSACTIONS ON COMPUTERS*, c-21, Septiembre 1972.

[Fol] Folding@home. http://folding.stanford.edu/. Pestaña de Estadísticas.

[Groop98] Groop, W. *MPI: The Complete Reference. Volume 2: the MPI-2 Extensions.* The MIT Press, 1998.

[Herrera88] Herrera, I., Pinder, G., y Allen, M. *Numerical Modelling in science and engineering.* Wyley, 1988.

[Herrera91] Herrera, I. Unified formulation of enhanced oil-recovery methods. *Computational Modelling of Free and Moving Boundary Problems*, 1:399-415, 1991.

[Herrera10] Herrera, I., Galindo, A., y Camacho, R. Shock modelling in variable bubble point problems of petroleum engineering. *Geofísica Internacional*, 50-1:85-98, 2010.

[Int] Intel. *Intel Math Kernel Library for Linux OS*. Http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation/.

[Kaufmann03] Kaufmann, M. *Sourcebook of Parallel Computing*. Morgan Kaufmann Publishers, 2003.

[Kirk10] Kirk, D. y Wen-mei, W. H. *Programming Massively Parallel Processors. A hands-on approach*. Elsevier, 2010.

[Lake96] Lake, L. W. *Enhanced oil recovery*. Prentice Hall, 1996.

[Leonard90] Leonard, H. P. y Mokhtari, S. Beyond first order upwinding: The ultra-sharp alternative for no-oscillatory steady-state simulation of convection. *International Journal for numerical methods in engineering*, 30 : 729-766, 1990.

[Leveque80] Leveque, J. R. *Numerical Methods for Conservation Laws*. A, 1980.

[Liu00] Liu, P. Experiences with parallel n-body simulation. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, 11, Diciembre 2000.

[Malalasekera95] Malalasekera y W. Veersteg, H. K. *An introduction to Computational Fluid Dynamics*. Longman Scientific and technical, 1995.

[OOS] OOSC. *Blitz++ Documentation*. Http://www.oonumerics.org/blitz/docs/.

[Ope] Openfvm. http://sourceforge.net/projects/openfvm/.

[Patankar92] Patankar, S. *Numerical Heat Transfer and Fluid Flow*. Birkhauser verlag, 1992.

[PET] *PETSc User's Manual*, revision 3.1 edón. Http://www.mcs.anl.gov/petsc/petsc-as/documentation/index.html.

[Saad00] Saad, Y. *Iterative Methods for Sparse Linear Systems*. A, 2000.

[Sanders10] Sanders, J. *CUDA by Example. An introduction to General-Purpose GPU Programming*. Addison-Wesley, 2010.

[Sel] *Seldon Documentation*. Http://seldon.sourceforge.net/.

[top] top500 supercomputing sites. www.top500.org.

[TUN] Tuna. http://132.248.182.189/luiggi/soft_S.html. En desarrollo.

[Welge52] Welge, H. J. A simplified method for computing oil recovery by gas or water drive. *Transactions of the American Institue for Mining and Metallurgical Engineeers*, 195 : 91-98, 1952.

[Wolff08] Wolff, M. *Comparison of mathematical and numerical models for two-phase flow in porous media*. Master Thesis, International Research Training Group, 2008.

Glosario

Blitz++ Biblioteca numérica de la OOSC (Object Oriented Scientific Computing). 40

cluster sistema de cómputo distribuido, formado por varios procesadores conectados por una red. 32, 36

CUDA *Compute Unified Device Architecture*, motor de computo paralelo, para ejecutar programas de cómputo numérico sobre tarjetas de gráficos usando tarjetas NVIDIA. 40

FLOPS *Floating Point operations by Second*-Operaciones de punto flotante por segundo. 33, 34

HPC Cómputo de Alto Rendimiento (por las siglas en ingles de High Performance Computing). 31

IntelMKL *Intel Math Kernel Libray*, Biblioteca numérica para optimizar programas que se ejecutan sobre arquitecturas de memoria compartida, usando la tecnología *Multithread-OpenMp*. 40

MIMD *Multiple Instruction stream, Multiple Data stream*, flujo múltiple de instrucciones, flujo múltiple de datos. 35, 36, 40

MISD *Multiple Instruction stream, Single Data stream*, flujo múltiple de instrucciones, flujo simple de datos. 35

MPI Librería para manejo de comunicación mediante mensaje (por las siglas en ingles de Message Passing Interface). 32, 40, 84

PETSc *Portable Extensible Toolkit for Scietnific Computation*, Software para la solucion de ecuaciones diferenciales de manera paralela, usando MPI y un esquema de memoria distribuida. 40

RAM *Random Access Memory*-Memoria de Acceso Aleatorio. 34

SIMD *Single Instruction stream, Multiple Data stream*, flujo simple de instrucciones, flujo múltiple de datos. 35, 36

SISD *Single Instruction stream, Single Data stream*, flujo simple de instrucciones, flujo simple de datos. 35

SPMD *Single Program stream, Multiple Data stream*, flujo simple de conjunto de instrucciones(programas), flujo múltiple de datos. 36

TUNA *Template Units for Numerical Applications, Finite Volume*, Software para la solución de ecuaciones diferenciales usando el método de volumen finito, escrito en C++ sobre la base de templates. 39, 40