VNIVERSIDAD NACIONAL
AVTONOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

MAESTRÍA EN CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN

SOLUCIÓN NUMÉRICA DE FLUJO CONVECTIVO NATURAL USANDO GPUS

SISTEMA ESCOLARIZADO
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIAS

PRESENTA:
ANGEL BERNARDO SANTIAGO ANTONINO

TUTOR:
DR. LUIS MIGUEL DE LA CRUZ SALAS GEOFÍSICA-UNAM

MIEMBROS DEL COMITÉ TUTORAL:
DR. ISMAEL HERRERA REVILLA GEOFÍSICA-UNAM
DRA. GRACIELA HERRERA ZAMARRÓN GEOFÍSICA-UNAM
DR. MARTÍN SALINAS VÁZQUEZ IING-UNAM
DR. DEMETRIO FABIÁN GARCÍA NOCETI IIMAS-UNAM

MÉXICO, D. F. OCTUBRE DEL 2015

Universidad Nacional
Autónoma de México

Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Reconocimientos

Me gustaría expresar un especial gradecimiento a mi asesor de Tesis Luis Miguel de la Cruz Salas por esperar tanto tiempo en el proceso de mi tesis. Y al invaluable apoyo moral de mi madre, sin ella no hubiera llegado hasta aquí.

Además al Dr. Luis Fernando Magaña Solís por tanto apoyo y consejos que me ha dado. Y en general a toda la gente que me ha brindado su confianza.

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado de mi propio trabajo y no incluye nada que sea el resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Angel Bernardo Santiago Antonino. México, D.F., 2015

Resumen

En el presente trabajo se realiza un estudio de rendimiento de tarjetas gráficas ejecutando cálculos para la solución de ecuaciones de flujo (ecuaciones diferenciales parciales de segundo orden, no lineales y acopladas). El objetivo es mostrar la eficiencia de dichas tarjetas trabajando en sistemas que implican varias copias GPU-CPU y CPU-GPU, manejo de sistemas de matrices comprimidas (CRS) y la copia de grandes bloques de información, características de las soluciones de ecuaciones diferenciales que describen a los flujos en varias áreas de las ciencias e ingenierías.

Índice general

1. Introducción. 1
1.1. El cómputo paralelo y la CFD. 2
1.2. Los GPUs y la CFD. 3
1.3. Objetivos y metas. 4
1.4. Estructura de la tesis 5

2. Modelos matemáticos 7
2.1. Convección natural 10
2.1.1. Ecuaciones adimensionales 11

3. Modelo numérico 13
3.1. Volumen finito 13
3.1.1. Discretización de una ecuación general 14
3.2. Aproximación de los términos difusivos 16
3.3. Aproximación de los términos convectivos 17
3.3.1. Condiciones de frontera 18
3.4. Acoplamiento presión-velocidad 20
3.4.1. Mallas desplazadas 21
3.4.2. Ecuación de corrección a la presión 22
3.4.3. SIMPLEC 23
3.5. Solución de los sistemas lineales 25
3.5.1. CGM 25
3.5.2. BiCGStab 27
3.5.3. Compresión de matrices por CRS 28
3.6. Discusión 29

4. Modelos Computacionales 31
4.1. Paralelismo 32
4.1.1. Taxonomía de flynn 33
4.1.2. Memoria compartida 34
4.1.3. Memoria distribuida 34

4.1.4. Memoria híbrida 35
4.1.5. CUDA 35
4.1.6. OpenCL 39
4.1.7. ViennaCL 39
4.2. Medidas de rendimientos 41
4.3. TUNAM 42
4.3.1. BiCGStab adaptado a TUNAM 43
4.4. Discusión 44

5. Resultados numéricos **45**
5.1. Verificación del método de solución 45
5.2. Pruebas de rendimiento 50

6. Conclusiones **61**

A. Especificaciones de tarjetas de video **63**

Bibliografía **65**

Capítulo 1

Introducción.

La dinámica de fluidos computacional o CFD (por sus siglas en inglés) es el proceso de describir el comportamiento de fluidos mediante la solución numérica de sistemas de ecuaciones diferenciales usando computadoras digitales. Ésta es una combinación de física, matemáticas discretas y algunas extensiones de las ciencias computacionales[1].

La CFD es una rama de la dinámica de fluidos que complementa a la parte experimental y la parte teórica, dando una alternativa rentable de investigación, por medio de simulaciones de flujos reales. Así como ofrecer pruebas teóricas para condiciones no disponibles experimentalmente o que son inaccesibles por su tamaño o por su duración, por ejemplo: flujo de galaxias, modelos climáticos, flujos turbulentos, etc.

En la CFD los métodos de solución implican discretización de las ecuaciones diferenciales parciales que describen el problema, mediante el mapeo de la región de interés con un número finito de puntos, y la descripción de dichos puntos interaccionando con sus vecinos cercanos.

El desarrollo de computadoras más eficientes ha generado un creciente interés en la CFD y esto ha producido un dramático incremento en la eficiencia de las técnicas computacionales. Entre las ventajas obtenidas por la dinámica de fluidos por medio de sistemas de cómputo, tenemos:

- Los tiempos de diseño y desarrollo son significativamente reducidos.
- Puede simular condiciones de flujo no reproducibles en modelos de prueba experimentales.
- Provee más detalles e información comprensible.
- Produce un consumo energético bajo.

Además, rediseñar un experimento por medio de la CFD se facilita si se mira el problema mediante componentes que pueden ser reutilizados. De esta manera, un

buen desarrollo permitirá la reutilización de ciertas secciones en el diseño de nuevos experimentos[2][3][4].

En los últimos 50 años las simulaciones numéricas han progresado mucho en términos de robustez, confiabilidad y eficiencia. Y actualmente deben considerar cambios en los procesos de diseño, trabajando en menores tiempo de desarrollo mientras se incluyen más y más disciplinas.

1.1. El cómputo paralelo y la CFD.

A lo largo de los años, el incremento en la demanda del poder de cómputo en la CFD ha hecho indispensable del desarrollo del HPC (Cómputo de alto desempeño). De esta manera se han propuesto muchas tecnologías para reducir los tiempos de computo en la solución de los sistemas de ecuaciones, generados por los modelos de flujo. La primer propuesta fue incrementar las velocidades de los relojes del procesador central, pero dicha alternativa ha quedado estancada, debido principalmente a las restricciones en la disipación de calor. El cómputo paralelo es una buena idea para acelerar los programas, con esto se reduce la carga por procesador cada que se incrementa el número de procesadores[5].

La necesidad de cómputo de alto desempeño ha llevado a los investigadores a trabajar en la paralelización de los procesos, de esta manera se logra resolver en tiempos razonables, algunos problemas como: transferencia de calor, magneto-hidro-dinámica[6], aero-acústica [7], simulación de mantos acuíferos en escalas de kilometros; lo que implica generar mallas con trillones de elementos y la necesidad de cómputo con capacidad de petaflops[8], simulación de yacimientos petroleros [9] [10], modelos de turbulencia aplicados a condiciones aeroe-spaciales; donde para simular un lanzamiento, se requiere más de 93 millones de elementos en una malla [11], etc.

Las formas de paralelismo puede darse como paralelismo de tareas, paralelismo de datos o una combinación de de los dos. Los paradigmas para implementar dichos algoritmos es mediante el paso de mensajes con MPI[1] para sistemas de memoria distribuida y OpenMP[2] para sistemas de memoria compartida[4]. De esta manera se intenta abordar problemas con altas resoluciones, debido principalmente a fenómenos de turbulencia, y reducir los tiempos de cálculo.

[1] http://www.mcs.anl.gov/research/projects/mpi/

[2] http://openmp.org/wp/

1.2. Los GPUs y la CFD.

Los paradigmas de programación y ejecución sobre tarjetas gráficas, han hecho más accesible el uso grandes de capacidades de cómputo en sistemas de menor costo de adquisición y de mantenimiento, ya que anteriormente los únicos sistemas de cómputo de alto rendimiento estaban constituidos por varias computadoras interconectadas, y su mantenimiento es bastante costoso. Por el contrario las tarjetas gráficas han sufrido un incremento en sus capacidades de cálculo con un consumo energético muy eficiente y costos de mantenimiento muy bajos[5].

Desde antes de la aparición de CUDA y OpenCL(De los cuales se hablará más adelante), ya existían artículos que aprovechaban las capacidades de paralelismo de las tarjetas gráficas [12][13][14]. Pero junto con la llegada de CUDA se pudo ver un gran interés en desarrollar software que aprovechara las virtudes de los GPUs[15][11][7][16]. Todo esto utilizando diferentes enfoques, volumen finito, diferencias finitas, métodos iterativos de solución, dominios cúbicos, dominios cilíndricos, mallas estructuradas y no estructuradas, etc. También se adopta OpenCL, aunque en menor medida y mucho tiempo después[17][18][19].

A pesar de que se han obtenido excelentes resultados con las arquiecturas multiprocesador; cluster o Grids[20][8][21], actualmente se hace investigación sobre el uso de las tarjetas gráficas con los sistemas ya existentes para optimizar el trabajo en conjunto, de esta manera se puede encontrar referencia de artículos que se basan en ejecuciones sobre supercomputadoras [16][22][17][23][24], y más recientemente [25][26].

Debido a que CUDA es el principal sistema adoptado para cómputo de propósito general sobre GPUs, existe mucho trabajo basado este con enfoques a CFD, y muchos de ellos se pueden encontrar en Internet con acceso libre, sin embargo una de las desventajas de estos trabajos es que gran parte de la metodología y de las herramientas no se describen a detalle, de esta manera no sabemos hasta dónde son válidas sus mejoras. Además, normalmente se hace una discusión acerca de las virtudes tanto del hardware como de la implementación del software, pero no de sus puntos débiles.

Actualmente están en desarrollo herramientas que permiten la fácil implementación de código sobre GPU, es el caso de OpenACC, sin embargo, no son lo suficientemente eficientes[27].

En las arquitecturas Tesla se contaba con 240 Cuda cores en la siguiente generación Fermi se contaba con 512 Cuda cores y en la versión más reciente hasta el momento Kepler se cuenta con 1536. Pasando de 1063 GFLOPs[1] en la arquitectura Tesla a 3090 GFLOPs en la arquitectura Kepler[28]. En el caso de ATI 48 Stream Cores en su primer generación con 375 GFLOPs a 1600 Stream Cores con 2640 GFLOPs en la cuarta generación[29]. Mientras los MIC de Intel han evolucionado de manera diferente, contando actualmente con 61 cores y 1.2 teraFLOPS[30] a pesar de tener mayor capacidad

[1] 1 GFLOPs son 1000 millones de operaciones de punto flotante por segundo

teórica, en las pruebas, no se nota mucha diferencia, incluso se observa a Xeon Phi con menor rendimiento que la Tesla de NVIDIA[31].

1.3. Objetivos y metas.

El objetivo de esta tesis es estudiar la mejora en rendimiento, generada por el uso de tarjetas gráficas, para resolver problemas de flujo convectivo natural. El planteamiento se hará con la finalidad de tener datos, que puedan ser generalizados o reutilizados para una extensión de este trabajo, con algún grado de complejidad. En este caso requerimos datos que nos permitan observar el comportamiento de los flujos con alta precisión, lo que implica que el dominio sea lo suficientemente fino para ver a detalle los procesos físicos generados. Ésto da como resultado sistemas lineales extremadamente grandes, que a su vez requieren de tiempos muy extensos de cómputo para obtener una solución.

El objetivo desde el punto de vista de la computación es poder optimizar los tiempos de cálculo usando herramientas de última generación, en las áreas donde el cómputo intensivo se vuelve indispensable.

TUNAM es un paquete desarrollado en la UNAM, con la finalidad de resolver de manera eficiente una gran cantidad de problemas de dinámica de fluidos, su código permanece en desarrollo y está completamente disponible de forma libre en Internet. Además es un software ampliamente documentado; y cuyo objetivo desde un principio fue optimizar los tiempos de cálculo, y facilitar su desarrollo mediante una arquitectura bien planeada. Una de las ventajas de usar TUNAM es que está desarrollado con un enfoque a problemas de dinámica de fluidos. Y es que una gran cantidad de investigadores sostienen que un software comercial normalmente tiene bajo rendimiento, debido a la necesidad de hacerlo robusto.

Existen muchas referencias al uso de GPU en CFD(Dinámica de Fluidos Computacional), sin embargo es mucho menor la cantidad de recursos disponibles de manera gratuita y aun menor, de código abierto, es por eso la importancia de trabajar sobre un software, que además de tener código abierto, se sigue desarrollando para cubrir las necesidades en algunos sectores de investigación.

En el presente trabajo se programan módulos que se integran a TUNAM, para trabajar las secciones más paralelizables usando GPUs, ésto a través de las bibliotecas de ViennaCL. La finalidad es extender a TUNAM y de alguna manera obtener una medida en la mejora de dicha extensión. Con ViennaCL se pretende tener una variedad más amplia de recursos de cómputo disponibles, debido a que con el mismos código se puede hacer uso de GPUs ajenos a NVIDIA y dispositivos embebidos que soporten OpenCL.

Con este trabajo se pretende cubrir dos cuestiones; la necesidad de obtener mayor cantidad de recursos disponibles para investigación, mediante el uso de dispositivos como es el caso de GPUs, y obtener un conjunto de medidas de los procesos involucrados

y no mencionados en trabajos similares.

1.4. Estructura de la tesis

En el capítulo dos se planteará un sistema de ecuaciones diferenciales procedentes de la teoría de la dinámica de fluidos, se obtendrán algunas aproximaciones necesarias para el planteamiento de un problema específico, que servirá de referencia para este trabajo.

El capítulo tres se dedicará a los modelos numéricos utilizado en este trabajo, se justificará su uso y se dará una breve descripción de sus características. Aquí se tratará las cuestiones de matemáticas discretas; se abordará temas como volumen finito, condiciones de fronteras términos de convección y difusión en un sistema de ecuaciones diferenciales, desacoplamiento de los sistemas, tratamiento de la no linealidad y los algoritmos de solución.

El capítulo cuatro se dedicará a describir algunos temas referente a recursos computacionales y los modelos, más comunes en estos momentos, de memoria compartida para cómputo concurrente, para luego dar una descripción breve de las bibliotecas utilizadas en este trabajo, se tratan algunos temas importantes como compresión de matrices, CUDA, OpenCL, medidas de rendimiento y las secciones de código utilizadas en este trabajo con una discusión de cada elemento.

El capítulo cinco se dedicará a los resultados obtenidos en las pruebas de rendimiento, se plantea un problema de convección natural y se trabaja sobre un único sistema de cómputo. A partir de esto se realiza un serie de pruebas variando la división del dominio, con el fin de detectar secciones que se puedan optimizar.

En el capítulo seis se presentarán las conclusiones del trabajo, y se hacen sugerencias acerca de trabajos futuros tomando como base el aquí presentado.

Capítulo 2

Modelos matemáticos

La dinámica de fluidos es el estudio del movimiento de los fluidos (líquidos y gases), que son fenómenos macroscópicos. Esto significa que un volumen en el fluido se supone siempre muy grande comparado con las moléculas que lo componen. Al mismo tiempo cuando se considera un volumen infinitesimalmente pequeño lo que realmente se está suponiendo es que ese volumen es muy pequeño comparado con el volumen total bajo estudio, pero grande comparado con las moléculas que lo componen.

La descripción del estado de un fluido en movimiento se hace por medio de la función que describe la distribución de la velocidad del fluido $\mathbf{v} = \mathbf{v}(x, y, z, t)$ y de dos cantidades termodinámicas más, por ejemplo la presión $p = p(x, y, z, t)$ y la densidad $\rho = \rho(x, y, z)$, de esta manera se tiene bien determinado el estado de movimiento de un fluido[32], esta afirmación se justificará a continuación junto con la deducción de un conjunto de ecuaciones denominadas ecuaciones fundamentales de la dinámica de fluidos.

La primer ecuación que será deducida es la que expresa la conservación de la materia. Considérese un volumen V_0 en un espacio, la masa total en este volumen está dada por: $\int \rho dV$ integrada sobre todo el volumen V_0 y donde ρ es la densidad del fluido. De esta manera el decremento en la cantidad de masa por unidad de tiempo en V_0 es:

$$-\frac{\partial}{\partial t}\int_{V_0} \rho dV \tag{2.1}$$

De la misma manera; la masa de fluido que fluye a través de un elemento de superficie que limita el volumen V_0 es $\rho\mathbf{v} \cdot d\mathbf{f}$, donde $\mathbf{v}$ es el vector que representa la velocidad del fluido en cada punto del espacio, la magnitud del vector $\mathbf{f}$ es igual al área de dicha superficie, y la dirección es perpendicular esta superficie en direccion externa a V_0, de esta manera $\rho\mathbf{v} \cdot d\mathbf{f}$ es positivo si el flujo es hacia afuera de V_0 y negativo si entra a dicho volumen. Si se integra a través de toda la superficie del volumen, se tiene:

$$\oint_{\partial V_0} \rho\mathbf{v} \cdot d\mathbf{f} \tag{2.2}$$

que representa el flujo total de masa a través de la frontera de V_0 por unidad de tiempo.

Igualando las ecuaciones 2.2 y 2.1 tenemos:

$$\frac{\partial}{\partial t}\int_{V_0} \rho dV = -\oint_{\partial V_0} \rho \mathbf{v} \cdot d\mathbf{f} \tag{2.3}$$

donde la integral de superficie puede ser transformada una integral de volumen por medio del teorema de Green:

$$\oint_{\partial V_0} \rho \mathbf{v} \cdot d\mathbf{f} = \int_{V_0} \nabla \cdot (\rho \mathbf{v}) dV$$

de esta manera se llega a que:

$$\int_{V_0} \left[\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v})\right] dV = 0 \tag{2.4}$$

Como la ecuación anterior es válida para cualquier volumen, entonces:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{2.5}$$

A esta última igualdad se le conoce como *ecuación de continuidad* y es una de las ecuaciones fundamentales de la dinámica de fluidos.

Ahora consideremos la fuerza total que actúa sobre un volumen de fluido V_0:

$$-\oint_{\partial V_0} p d\mathbf{f} \tag{2.6}$$

donde p es la presión punto a punto y $d\mathbf{f}$ es el mismo vector definido anteriormente. Transformando la expresión 2.6 en una integral de volumen, tenemos:

$$-\oint_{\partial V_0} p d\mathbf{f} = -\int_{V_0} \nabla p dV$$

De esta manera tenemos la expresión de una fuerza $-\nabla p$ que actúa por unidad de volumen en el fluido. Entonces tenemos que la ecuación de movimiento de dicho elemento de volumen es:

$$\rho_0 \frac{d\mathbf{v}}{dt} = -\nabla p + \mu \nabla^2 \mathbf{v} + \rho \mathbf{g} \tag{2.7}$$

Donde ∇p es la contribución de la presión a las fuerzas que actúan sobre el elemento de fluido, $\rho \mathbf{g}$ es la contribución de la fuerza de gravedad, y $\mu \nabla^2 \mathbf{v}$ es la contribución de las fuerzas dadas por la viscosidad del fluido, para flujos incompresibles, y donde a μ se le conoce como viscosidad dinámica. En este texto no se hace la deducción de esta última expresión ya que requiere algunos conceptos extra sobre tensores pero se puede consultar en la bibliografía [e.g. 33, p. 45].

Del lado izquierdo de la ecuación 2.7 tenemos; $\rho d\mathbf{v}/dt$ que representa la variación respecto al tiempo, de la velocidad de una partícula cuando se mueve en el espacio. Esta es la derivada total de la velocidad, también conocida como derivada material y se expresa como sigue:

$$\frac{d\mathbf{v}}{dt} = \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} \tag{2.8}$$

Una descripción más detallada, se puede ver la bibliografía [e.g. 32, p.3].

Sustituyendo la expresión 2.8 en 2.7 se tiene:

$$\rho \left[\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} \right] = -\nabla p + \mu \nabla^2 \mathbf{v} + \rho \mathbf{g} \tag{2.9}$$

A la ecuación 2.9 se le conoce como *ecuación de* $Navier - stokes$ para flujos incompresibles y también forma parte del conjunto de ecuaciones fundamentales de la mecánica de fluidos. Cabe señalar que es una ecuación vectorial, donde hay una igualdad por cada componente de las coordenadas espaciales por tanto en muchos textos se le denomina ecuaciones de Navier-Stokes.

La siguiente expresión se obtiene de considerar un balance de energía, los detalles acerca de cada término no se abordan debido a la necesidad de conceptos avanzados de termodinámica, pero se pueden consultar ampliamente en la bilbiografía[e.g. 34].

$$\frac{d}{dt} \int_{V_0} \rho \hat{u} dV = - \oint_{\partial V_0} (\rho \hat{h}) \mathbf{v} \cdot d\mathbf{f} - \oint_{\partial V_0} (-k \nabla T) \cdot d\mathbf{f} + \int_{V_0} \dot{q} dV \tag{2.10}$$

Donde el término a la izquierda de la igualdad es el incremento de la energía interna en un volumen V_0. El primer término a la derecha de la igualdad es el flujo de energía interna y el trabajo en V_0. El segundo término de la derecha es la transferencia de calor a través de las paredes de V_0, y el último término es una contribución dada por radiación externa, o calentamiento por resistencia eléctrica o reacciones químicas.

Dado que el cambio en la energía interna se mide en la misma región cuando transcurre el tiempo, podemos reescribir el primer término de la ecuación 2.10 como:

$$\frac{d}{dt} \int_{V_0} \rho \hat{u} dV = \int_{V_0} \frac{\partial}{\partial t} \rho \hat{u} dV$$

Y las integrales de superficie convertirlas en integrales de volumen:

$$\oint_{\partial V_0} (\rho \hat{h}) \mathbf{v} \cdot d\mathbf{f} = \int_{V_0} \nabla \cdot (\rho \hat{h} \mathbf{v}) dV$$

$$\oint_{\partial V_0} (-k \nabla T) \cdot d\mathbf{f} = - \int_{V_0} \nabla \cdot (k \nabla T) dV$$

Por lo tanto la ecuación 2.10 se puede escribir como:

$$\int_{V_0} \left(\frac{\partial}{\partial t} \rho \hat{u} + \nabla \cdot (\rho \hat{h}) \mathbf{v} + \nabla \cdot (k \nabla T) - \dot{q} \right) dV = 0$$

Como la ecuación anterior es válida para cualquier volumen, entonces:

$$\frac{\partial}{\partial t} \rho \hat{u} + \nabla \cdot (\rho \hat{h} \mathbf{v}) + \nabla \cdot (k \nabla T) - \dot{q} = 0 \tag{2.11}$$

De termodinámica se sabe que $\hat{u} = \hat{h} + p/\rho$ y que $d\hat{h} = c_v dT + (d\hat{h}/dp)_T dp$. Pero las variaciones de la presión en el flujo, no son suficientemente grandes como para afectar sus propiedades termodinámicas. De esta manera se puede despreciar el efecto de las variaciones de presión en la energía interna y en la densidad. Esta aproximación es razonable para la mayoría de los líquidos y gases fluyendo a velocidades menores a 1/3 de la velocidad del sonido[34, p. 292]. Por tanto podemos reescribir la ec. 2.11 como:

$$\rho c_v \frac{\partial}{\partial t} T + \rho c_v \nabla \cdot (T \mathbf{v}) + k \nabla^2 T - \dot{q} = 0 \tag{2.12}$$

A la ecuación 2.12 se le conoce como *ecuación de energía* para flujos incompresibles y también forma parte del conjunto de ecuaciones fundamentales de la mecánica de fluidos.

2.1. Convección natural

La convección natural es un fenómeno que aparece en muchos sistemas naturales y de las ingenierías. A continuación se plantea el sistema de ecuaciones que gobiernan este fenómeno, en donde se toma en cuenta la aproximación de Boussinesq, es decir, la densidad se considera constante excepto en los términos de fuerza de cuerpo. Las ecuaciones son:

$$\nabla \cdot (\mathbf{v}) = 0 \tag{2.13}$$

$$\rho_0 \left[\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla) \mathbf{v} \right] = -\nabla p + \mu \nabla^2 \mathbf{v} + \rho \mathbf{g} \tag{2.14}$$

$$\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T = \alpha \nabla^2 T \tag{2.15}$$

donde ρ es la densidad, ρ_0 es una densidad de referencia, μ es la viscosidad dinámica, α es la difusividad térmica. κ es el coeficiente de conductividad térmica y tanto c_V como κ forman parte de la constante α.

El conjunto de cinco ecuaciones anteriores tiene 6 incógnitas; las tres componentes de la velocidad, la densidad, la presión y la temperatura. Para resolver estas ecuaciones

es necesario contar con una ecuación más que relacione la temperatura con la densidad. En este caso (fluido incompresible) bastará con la siguiente ecuación de estado:

$$\rho = \rho_0[1 - \beta(T - T_0)]$$

donde β es el coeficiente de expansión volumétrica, T_0 es el valor de la temperatura cuando $\rho = \rho_0$ y β está definida como:

$$\beta = \frac{1}{\rho_0}\left(\frac{\partial \rho}{\partial T}\right)_{T=T_0}$$

2.1.1. Ecuaciones adimensionales

Una forma equivalente al conjunto de ecuaciones 2.13, 2.14 y 2.15 se obtiene haciendo un escalamiento, para llevarlas a una forma adimensional. De esta manera aparecen parámetros que permiten hacer estudios a flujos en diferentes estados como flujo laminar o flujo turbulento por ejemplo. En este caso se definen las siguientes relaciones:

$$\mathbf{x} = \frac{\mathbf{x}'}{H} \tag{2.16}$$

$$t = \frac{t'}{H^2/\alpha} \tag{2.17}$$

$$\mathbf{v} = \frac{\mathbf{v}'}{\alpha/H} \tag{2.18}$$

$$p = \frac{p'}{\alpha\nu\rho_0/H^2} \tag{2.19}$$

$$T = \frac{T' - T_C}{\Delta T} - \frac{1}{2} \tag{2.20}$$

aquí se han reemplazado a las variables originales por variables con tilde ('). Donde $\mathbf{x}$ se refiere a las coordenadas x,y y z. H es una longitud característica, ν la viscosidad cinemática($\nu = \mu/\rho_0$) y $\Delta T = T_H - T_C$ representa una diferencia de temperaturas, haciendo la sustitución en las ecuaciones 2.13, 2.14 y 2.15 se llega al siguiente conjunto de ecuaciones:

$$\nabla \cdot (\mathbf{v}) = 0 \tag{2.21}$$

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\nabla p + \mathrm{Pr}\nabla^2\mathbf{v} + \mathbf{b} \tag{2.22}$$

$$\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T = \nabla^2 T \tag{2.23}$$

En este caso $\mathbf{b}$ representa a las fuerzas de cuerpo externas y sólo se toma en cuenta el efecto de la gravedad apuntando en la dirección negativa del eje y. De esta manera $\mathbf{b} = (0, \mathbf{Pr}(\mathbf{Ra})T, 0)$, donde $\mathbf{Pr}$ es el número de Prandlt y $\mathbf{Ra}$ el número de Rayleigh con la siguiente definición:

$$\mathbf{Pr} = \frac{\nu}{\alpha}$$

$$\mathbf{Ra} = \frac{g\beta\Delta L_y^3}{\alpha\nu}$$

g es la constante de aceleración de la gravedad.

El conjunto de ecuaciones 2.21, 2.22 y 2.23 puede escribirse de una forma generalizada como:

$$\frac{\partial\varphi}{\partial t} + \frac{\partial}{\partial x_j}(u_j\varphi) = \frac{\partial}{\partial x_j}\left(\Gamma\frac{\partial\varphi}{\partial x_j}\right) + S, \quad \text{para} \;\; \text{j} = 1, 2, 3 \tag{2.24}$$

donde índices repetidos representan una suma sobre todos los valores que puede tomar dicha variable. El término φ representa a la temperatura (T), la densidad (ρ) o a alguna variable de nuestro modelo, Γ es el coeficiente de difusión y S representa a los terminos fuente.

Capítulo 3

Modelo numérico

Los métodos numéricos permiten encontrar soluciones aproximadas a las ecuaciones presentadas en el capítulo anterior. No se conocen, hasta ahora, soluciones analíticas a dichas ecuaciones, pues ellas son no lineales y acopladas, lo cual las hace muy complicadas.

Existen varios métodos para resolver el conjunto de ecuaciones 2.21, 2.22 y 2.23 de forma numérica, para este trabajo se utiliza el método de Volumen Finito que es un procedimiento de discretización; que integra dichas ecuaciones sobre un volumen finito (1D, 2D o 3D), en este caso se hará sobre dominios cúbicos con mallas estructuradas ortogonales.

3.1. Volumen finito

El método de volumen finito toma su forma de las leyes de conservación en física. Este método está basado en las ecuaciones balance, y esto permite que aun teniendo una malla gruesa podamos tener resultados cualitativamente realistas.

La idea básica es colocar un volumen (intervalo en caso de 1D o superficie en caso de 2D) alrededor de cada vértice generado por la malla, estos volúmenes no se traslapan pero sus fronteras si coinciden a lo largo de todo el dominio (véase Fig. 3.1), luego se hace un balance de la propiedad estudiada en cada volumen como si se tratara del dominio completo.

Por ejemplo, el conjunto de ecuaciones 2.21, 2.22 y 2.23 que competen a este trabajo, se integran en cada volumen para realizar una aproximación a su solución numérica. Dichas ecuaciones son de segundo orden en la derivada, por tanto requerimos de algunas técnicas, de aproximación numéricas, para los términos de las primeras y segundas derivadas. Esto al final nos lleva a un sistema de ecuaciones generado por todos los volúmenes de la malla y sus vecinos inmediatos. El planteamiento de esta manera generará sistemas que tendrán la propiedad de conservación de masa, energía o cantidad

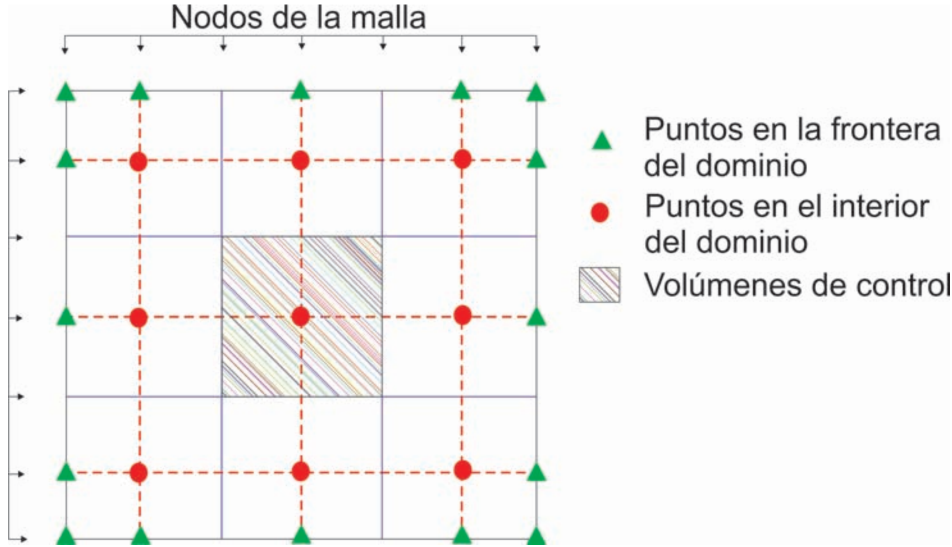

Figura 3.1: Dominio de estudio discretizado usando volúmenes de control en 2D.

de movimiento(según sea el caso).

3.1.1. Discretización de una ecuación general

Dado un dominio en tres dimensiones, considérese una malla rectangular con volúmenes de control como el de la figura 3.2. Los subíndices ***E, W, N, S, F*** y ***B*** indican los puntos vecinos al punto ***P*** de la malla, mientras que las etiquetas ***e, w, n, s, f*** y ***b*** indican las caras del volumen de control. La integración, espacial y temporal de la ecuación 2.24 para una propiedad escalar ϕ produce la siguiente expresión:

$$\left(\phi - \phi^0\right) \frac{\Delta V}{\Delta t} + C = D + S \tag{3.1}$$

donde ϕ^0 representa el valor en el tiempo t y ϕ es el valor de la variable escalar en el tiempo $t + \Delta t$, de esta manera la integración temporal se realiza en el intervalo Δt. La forma de los términos **C, D** y **S**, convectivo, difusivo y fuente respectivamente, dependen del esquema numérico utilizado en la discretización de cada uno de estos términos.

Utilizando la nomenclatura de la figura 3.2, la forma general de los términos de la ecuación 3.1 es como sigue:

$$C = (c_e \phi_e - c_w \phi_w) + (c_n c_n - c_s \phi_s) + (c_f \phi_f - c_b \phi_b) \tag{3.2}$$

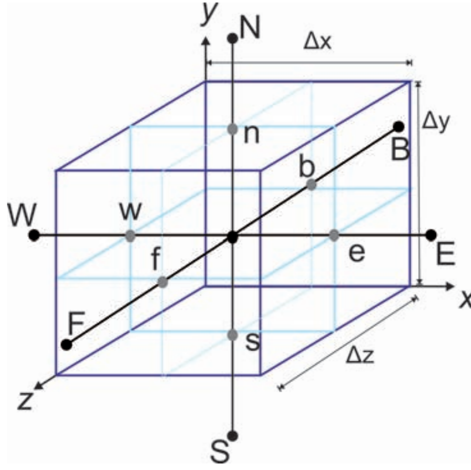

Figura 3.2: Volumen de control alrededor del punto P

$$\begin{aligned} D = \Gamma \left[\left(\frac{\partial \phi}{\partial x} \right)_e - \left(\frac{\partial \phi}{\partial x} \right)_w \right] \Delta y \Delta z \\ + \Gamma \left[\left(\frac{\partial \phi}{\partial y} \right)_n - \left(\frac{\partial \phi}{\partial y} \right)_s \right] \Delta x \Delta z \\ + \Gamma \left[\left(\frac{\partial \phi}{\partial z} \right)_f - \left(\frac{\partial \phi}{\partial z} \right)_b \right] \Delta x \Delta y \end{aligned} \tag{3.3}$$

$$S = S_p \Delta V \tag{3.4}$$

donde los términos c de la ecuación 3.2 están definidos de la siguiente manera:

$$\begin{aligned} c_e = u_e \Delta y \Delta z, \quad c_w = u_w \Delta y \Delta z, \\ c_n = v_n \Delta x \Delta z, \quad c_s = v_s \Delta x \Delta z, \\ c_f = w_f \Delta x \Delta y \quad c_b = w_b \Delta x \Delta y \end{aligned} \tag{3.5}$$

Los términos convectivos y difusivos se pueden aproximar usando diferentes esquemas como se puede ver en la referencia[35], pero independientemente de la aproximación usada, cuando se insertan estos esquemas en las ecuaciones 3.2 y 3.3, y estos a su vez en la ecuación 3.1, se obtienen sistemas lineales como el siguiente:

$$a_P\phi_P = a_E\phi_E + a_W\phi_W + a_N\phi_N + a_S\phi_S + a_F\phi_F + a_B\phi_B + S_P \tag{3.6}$$

donde los coeficientes contienen una parte difusiva (D)y otra convectiva(C):

$$\begin{gathered} a_E = D_E + C_E, \qquad a_W = D_W + C_{W,} \\ a_N = D_N + C_N \qquad a_S = D_S + C_S \\ a_F = D_F + C_F \qquad a_B = D_B + C_B \\ a_P = D_P + C_P + \frac{\Delta V}{\Delta t} \end{gathered} \tag{3.7}$$

En este trabajo se utilizan diferentes aproximaciones para calcular numéricamente los términos convectivos y difusivos definidos en las ecuaciones 3.2 y 3.3. En las secciones que siguen se hace una descripción detallada de dichas aproximaciones.

3.2. Aproximación de los términos difusivos

De la ecuación 3.3 se ve necesario calcular un conjunto de derivadas parciales, dichas derivadas se deben evaluar en las caras del volumen de control. Éstas pueden aproximarse usando un perfil lineal entre los valores adyacentes de los puntos de la malla; por ejemplo entre **P** y **E**, lo que da como resultado:

$$\begin{aligned} \left(\frac{\partial \phi}{\partial x}\right)_e &\simeq \frac{\phi_E - \phi_P}{\Delta x_e}, \qquad & \left(\frac{\partial \phi}{\partial x}\right)_w &\simeq \frac{\phi_P - \phi_W}{\Delta x_w}, \\ \left(\frac{\partial \phi}{\partial y}\right)_n &\simeq \frac{\phi_N - \phi_P}{\Delta y_n}, \qquad & \left(\frac{\partial \phi}{\partial y}\right)_w &\simeq \frac{\phi_P - \phi_W}{\Delta x_w}, \\ \left(\frac{\partial \phi}{\partial z}\right)_f &\simeq \frac{\phi_F - \phi_P}{\Delta z_f}, \qquad & \left(\frac{\partial \phi}{\partial z}\right)_b &\simeq \frac{\phi_P - \phi_B}{\Delta z_b}, \end{aligned} \tag{3.8}$$

donde Δx_e, Δx_w, Δy_n, Δy_s, Δz_f y Δz_b son definidos como se muestra en la figura 3.3.

Si se hace un desarrollo en series de Taylor para ϕ_E, ϕ_W, ϕ_N, ϕ_S, ϕ_F y ϕ_B y se supone una malla uniforme, las expresiones 3.8 producen una aproximación de orden $\mathcal{O}\left(\Delta x^2\right)$. Con estas aproximaciones la parte difusiva de los coeficientes 3.7 tiene la siguiente forma:

$$D_E = \Gamma\frac{\Delta y \Delta z}{\Delta x_e}, \qquad D_W = \Gamma\frac{\Delta y \Delta z}{\Delta x_w},$$

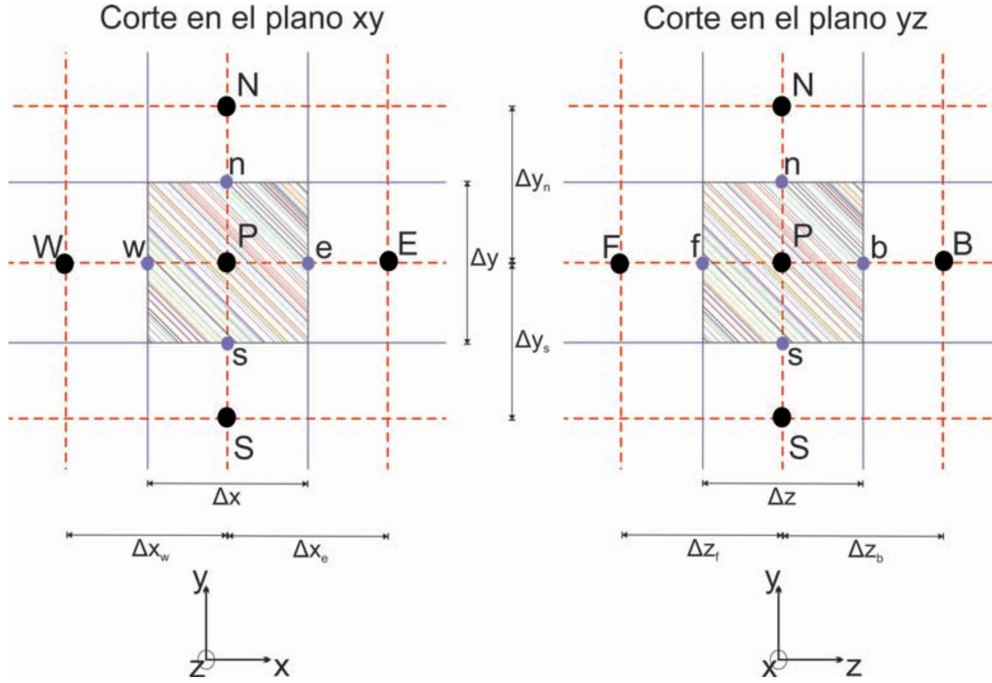

Figura 3.3: Cortes del dominio discreto en los planos xy y yz.

$$D_N = \Gamma \frac{\Delta x \Delta z}{\Delta y_n}, \qquad D_S = \Gamma \frac{\Delta x \Delta z}{\Delta y_s}, \tag{3.9}$$

$$D_F = \Gamma \frac{\Delta x \Delta y}{\Delta z_f}, \qquad D_B = \Gamma \frac{\Delta x \Delta y}{\Delta z_b},$$

$$D_P = D_E + D_W + D_N + D_S + D_F + D_B$$

De esta manera obtenemos expresiones sencillas para la aproximación de los términos difusivos que serán ocupadas a lo largo de todo este trabajo.

3.3. Aproximación de los términos convectivos

Existen varios esquemas para aproximar los términos convectivos del conjunto de ecuaciones 3.7, estos términos son los que aportan la parte no lineal a la ecuación general de transporte (2.24), por tanto es importante utilizar un esquema adecuado en el cálculo de dichos términos.

En la ecuación 3.2 se ve necesario tener los valores de ϕ en las caras del volumen de control. Sin embargo, ϕ representa a una variable escalar definida en los centros de los

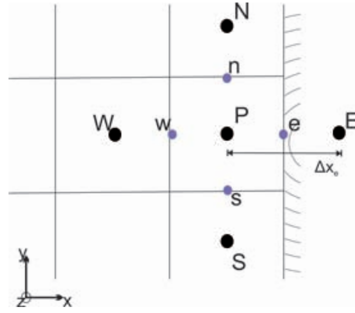

Figura 3.4: Frontera de un dominio Volumen de control en la frontera.

volúmenes, como se muestra en las figuras 3.2 y 3.3. Por tanto es necesario realizar una interpolación entre puntos adyacentes a las caras para obtener los valores necesarios. En este trabajo se utilizan tres diferentes esquemas para aproximar ϕ en las caras de los volúmenes: Upnwid, CDS, QUICK, véase la referencia [36] para más detalles de estos esquemas.

3.3.1. Condiciones de frontera

Las condiciones de frontera y las condiciones iniciales son las que definen a un problema, una vez establecida la naturaleza de éste. Es decir, si se trata de un flujo, la solución al sistema de ecuaciones generado por éste, es única una vez establecidas las condiciones anteriormente mencionadas. Por tanto, es importante tener un buen esquema de aproximación, de los valores en las fronteras, para nuestro modelo numérico. Y para este trabajo básicamente tenemos que modelar dos tipos de condiciones de fronteras:

1. Dirichlet: en donde el valor del campo se define en la frontera, es decir: $\phi = \phi_b$.
2. Neumann: está definido el gradiente del campo normal a la frontera, es decir $\partial\phi/\partial n = \phi'_b$.

En la figura 3.4, por ejemplo, el punto ***E*** cae fuera del dominio y la cara **e** del volumen que rodea a ***P*** cae justo en la frontera. Se puede usar las técnicas de discretización descritas antes, con lo cuál obtenemos una forma similar a la ecuación 3.6 para el punto ***P*** en términos de sus vecinos. Sin embargo la diferencia se ve reflejada en los vecinos que no se encuentran dentro del dominio.

En el caso de la figura 3.4, la condición de frontera de tipo Dirichlet es aproximada de tal manera que $\phi_b = \phi_e$. El valor de la frontera se puede aproximar mediante una interpolación lineal simple:

$$\phi_e = \phi_b \approx \frac{\phi_P + \phi_E}{2} \tag{3.10}$$

de donde se obtiene:

$$\phi_E = 2\phi_b - \phi_P \tag{3.11}$$

Sustituyendo esta última expresión en la ecuación 3.6 y factorizando obtenemos:

$$a_P^* \phi_P = a_E^* \phi_E + a_W \phi_W + a_N \phi_N + a_S \phi_S + a_F \phi_F + a_B \phi_B + S_P^* \tag{3.12}$$

donde

$$a_P^* = a_P + a_E$$

$$S_P^* = S_P + 2a_E \phi_b \; y$$

$$a_E^* = 0$$

Para las condiciones de tipo Neumann, usando diferencias centrales para aproximar el gradiente normal a la superficie, tenemos que:

$$\phi_b' = \left(\frac{\partial \phi}{\partial x}\right)_e \approx \frac{\phi_E - \phi_P}{\Delta x_e} \tag{3.13}$$

de donde obtenemos:

$$\phi_E = \phi_P + \Delta x_e \phi_b' \tag{3.14}$$

Sustituyendo esta expresión en la ecuación 3.6 obtenemos una ecuación similar a 3.12, con los coeficientes definidos como sigue:

$$a_P^* = a_P - a_E,$$

$$S_P^* = S_P + a_E \Delta x_e \phi_b'$$

$$a_E^* = 0$$

De esta manera las condiciones de frontera se integran al sistema de ecuaciones a resolver.

3.4. Acoplamiento presión-velocidad

La componente de la convección de una variable escalar ϕ depende de la magnitud y dirección de la velocidad. En la mayoría de los problemas la velocidad no se conoce y debe calcularse al mismo tiempo que se obtienen las otras variables del flujo. Las ecuaciones para cada componente de la velocidad (ecuaciones de cantidad de movimiento) están descritas en la ecuación general 2.24. En este trabajo sólo se trata con flujos incompresibles, por lo tanto la velocidad debe satisfacer la ecuación de continuidad 2.21.

A partir de la ecuación 2.24 podemos derivar las ecuaciones de cantidad de movimiento y de continuidad como sigue:

$$\begin{aligned}
\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial w}{\partial z} &= -\frac{\partial p}{\partial x} + \Gamma\frac{\partial^2 u}{\partial x^2} + \Gamma\frac{\partial^2 u}{\partial y^2} + \Gamma\frac{\partial^2 u}{\partial z^2} + S_u, \\
\frac{\partial v}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial w}{\partial z} &= -\frac{\partial p}{\partial y} + \Gamma\frac{\partial^2 v}{\partial x^2} + \Gamma\frac{\partial^2 v}{\partial y^2} + \Gamma\frac{\partial^2 v}{\partial z^2} + S_v, \\
\frac{\partial w}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial w}{\partial z} &= -\frac{\partial p}{\partial z} + \Gamma\frac{\partial^2 w}{\partial x^2} + \Gamma\frac{\partial^2 w}{\partial y^2} + \Gamma\frac{\partial^2 w}{\partial z^2} + S_w
\end{aligned} \tag{3.15}$$

$$\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{3.16}$$

Para obtener una solución a este conjunto de ecuaciones primero tendremos que abordar los siguientes problemas:

- Los términos convectivos de las ecuaciones de cantidad de movimiento son cantidades no lineales.
- Las ecuaciones están fuertemente acopladas debido a que cada componente de la velocidad aparece en cada ecuación de cantidad de movimiento.
- No existe explícitamente una ecuación para la presión por tanto se necesita un esquema que la integre al conjunto de ecuaciones existentes.

Existen varios métodos para abordar dichos problemas, pero en este trabajo se utiliza el método SIMPLEC (*Semi-Implicit Method for Pressure Linked Equations - Consistent*) [37] con algunas modificaciones para resolver los problemas planteados en el capítulo anterior.

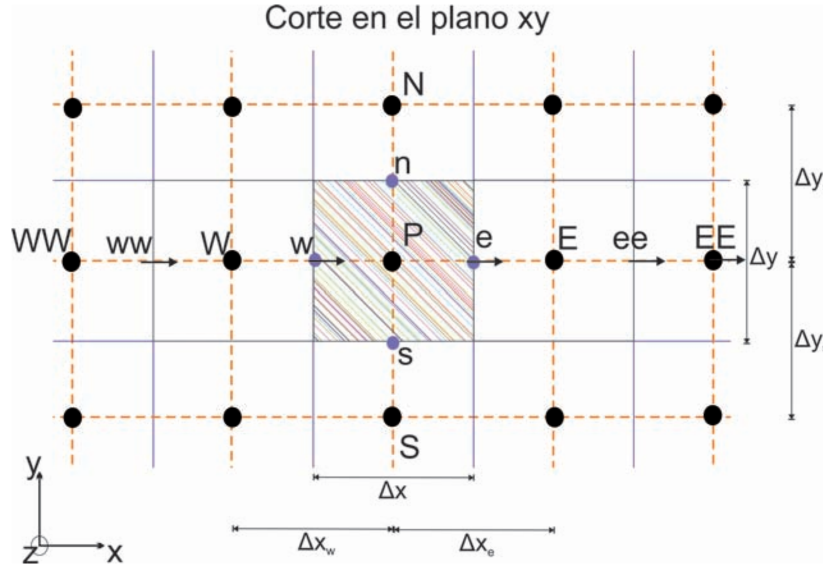

Figura 3.5: Esquema Upwind para el caso: $c_e > 0 \Longrightarrow \phi_e = \phi_P$ y $c_w > 0 \Longrightarrow \phi_w = \phi_W$.

3.4.1. Mallas desplazadas

Para aproximar el gradiente de presiones que aparece en las ecuaciones 3.15, se utilizará una interpolación lineal. Por ejemplo en la dirección x y basándonos en la figura 3.5, tenemos la siguiente expresión para el gradiente de presiones:

$$\left(\frac{\partial p}{\partial x}\right)_P \approx \frac{p_e - p_w}{\Delta x} = \frac{\left(\frac{p_E+p_P}{2}\right) - \left(\frac{p_P+p_W}{2}\right)}{\Delta x} = \frac{p_E - p_W}{\Delta x} \tag{3.17}$$

Con dicha expresión se presenta un nuevo problema ya que no aparece el valor de la presión en el punto central, lo que puede ocasionar oscilaciones no realistas en el campo de presiones, si se definen las velocidades en los centros de las mallas, entonces los valores de la presión no estarán representados correctamente y viceversa, un remedio a este problema es utilizar mallas desplazadas (*staggered grids*) para las componentes de la velocidad. De esta manera se evalúa las variables escalares, tales como la presión y la temperatura en los centros de los volúmenes de control, mientras que las velocidades se evalúan en las caras de los volúmenes. La figura 3.6 muestra de manera gráfica este procedimiento, donde se observa que el punto central $\boldsymbol{P}$, para la componente u de la velocidad, se desplaza a la cara w y lo mismo sucede para las demás direcciones. De esta manera el gradiente de presiones, en la ecuación para u se calcula de la siguiente forma:

$$\left(\frac{\partial p}{\partial x}\right)_w \approx \frac{p_P - p_W}{\Delta x} \tag{3.18}$$

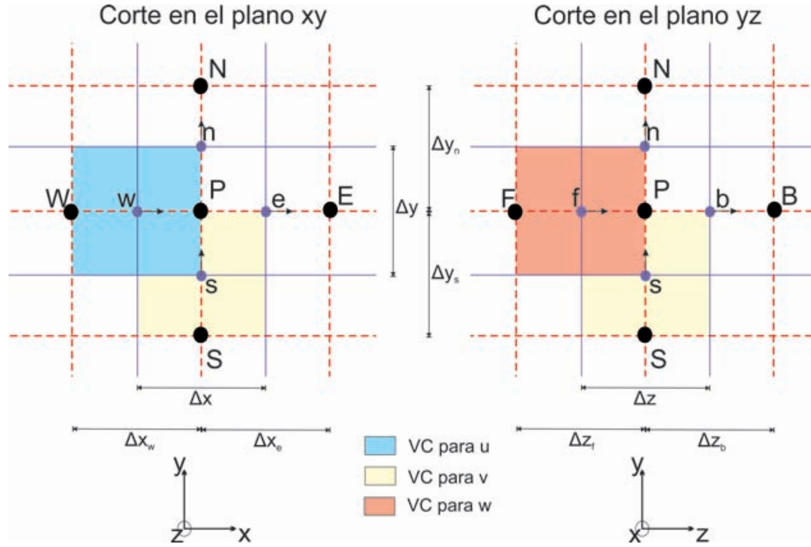

Figura 3.6: Descripción gráfica de volúmenes de control desplazados para las componentes de la velocidad.

También podemos ver que las mallas desplazadas generan velocidades en los lugares exactos donde se requiere para las ecuaciones escalares, por lo tanto no es necesario realizar interpolaciones.

3.4.2. Ecuación de corrección a la presión

El siguiente problema a abordar es el de la expresión para la presión, una relación para ésta se puede obtener a partir de las ecuaciones 3.15 y 3.16. Aplicando el método de volumen finito a la ecuación de cantidad de movimiento para u en una malla desplazada como se muestra en la figura 3.6, obtenemos:

$$a_P u_P = \sum_{nb} a_{nb} u_{nb} + b_u + A_u \left(p_W - p_P \right) \tag{3.19}$$

donde ya se ve el gradiente de presiones. En esta ecuación tenemos que $_{nb} =_E, _W, _N, _S, _F, _B$, $A_u = \Delta y \Delta z$ es el área de la cara w del volumen de control y b_u es el término fuente.

Para realizar la aproximación de la presión, definimos $p*$ como una presión aproximada ésta a su vez produce una velocidad aproximada $u*$, lo que genera la siguiente expresión:

$$a_P u_P^* = \sum_{nb} a_{nb} u_{nb}^* + b_u + A_u \left(p_W^* - p_P^* \right) \tag{3.20}$$

Para obtener u y p correctas se deben corregir los valores aproximados mediante $p = p^* + p'$ y $u = u^* + u'$. Una relación entre p' y u' se obtiene restando las ecuaciones 3.19 y 3.20:

$$a_P u'_P = \sum_{nb} a_{nb} u'_{nb} + A_u \left(p'_W - p'_P \right) \tag{3.21}$$

En los métodos del tipo SIMPLE se encuentra una ecuación discreta ya sea para la presión p o para la corrección a la presión p' y se resuelve dentro del ciclo global del método (véase [35]). En este trabajo se utiliza el método SIMPLEC que es una modificación del SIMPLE con la ventaja de que se obtiene convergencia en menos iteraciones.

3.4.3. SIMPLEC

En el método SIMPLEC[37] se realiza una aproximación que nos lleva a una expresión sencilla para p'. En este algoritmo se resta el termino $\sum a_{nb} u'_P$ en ambos lados de la ecuación 3.21 para obtener una expresión de la siguiente forma:

$$\left(a_P - \sum_{nb} a_{nb} \right) u'_P = \underbrace{\sum_{nb} a_{nb} \left(u'_{nb} - u'_b \right)}_{\approx 0} + A_u \left(p'_W - p'_P \right) \tag{3.22}$$

Suponiendo que $(u'_{nb} - u'_P)$ es aproximadamente igual a cero, para todo $_{nb}$ y mallas relativamente finas, se elimina este término de la ecuación, con lo que llegamos a que la expresión para la velocidad corregida tiene la siguiente forma:

$$u_P = u^*_P + u'_P = u^*_P + d_u \left(p'_W - p'_P \right) \tag{3.23}$$

donde

$$d_e = A_e / \left(a_P - \sum a_{nb} \right) \tag{3.24}$$

De la misma manera se obtienen expresiones para v y w teniendo:

$$v_P = v^*_P + v'_P = v^*_P + d_v \left(p'_S - p'_P \right) \tag{3.25}$$

$$w_P = w^*_P + w'_p = w^*_P + d_w \left(p'_B - p'_P \right) \tag{3.26}$$

Para obtener la expresión para p' se sustituyen las ecuaciones 3.23, 3.25 y 3.26 en la ecuación 3.16 y esto nos lleva a:

$$a_P p'_P = a_E p'_E + a_W p'_W + a_N p'_N + a_S p'_S + a_F p'_F + a_B p'_B + b_p \tag{3.27}$$

donde los coeficientes tienen la siguiente forma:

$$a'_E = d_u A_u, \quad a'_W = d_u A_u, \quad a'_N = d_v A_v,$$

$$a'_S = d_v A_v, \quad a'_F = d_w A_w, \quad a'_B = d_w A_W,$$

$$b_p = -\left(u_e^* - u_w^*\right)\Delta y \Delta z - \left(v_n^* - v_s^*\right)\Delta x \Delta z - \left(w_f^* - w_b^*\right)\Delta x \Delta y \tag{3.28}$$

De esta manera la definición de b_p es la forma discreta de la ecuación de continuidad para u^*, v^* y w^*, que produce el método de volumen finito, el objetivo es reducir esta cantidad hasta aproximarla a cero, con esto tenemos un criterio de convergencia que se utilizará a lo largo de este trabajo.

En los problemas de convección natural la diferencia de temperaturas es la que genera el movimiento, por tanto primero se resuelve ésta y luego las ecuaciones de cantidad de movimiento y la de corrección a la presión. De esta manera el método SIMPLEC para nuestro sistema de ecuaciones se compone de los siguientes pasos:

1. Se inicia con campos aproximados: T^*,p^*, u^*, v^* y w^*
2. Se resuelve la ecuación de energía para obtener T.
3. Se resuelven las ecuaciones de cantidad de movimiento usando los campos de presión y velocidad iniciales aproximados (p^*, u^*,v^*, w^*) y el campo de temperaturas T.
4. Se calculan los coeficientes de la ecuación de presión p_{nb} usando los coeficientes de las ecuaciones de cantidad de movimiento.
5. Se resuelve la ecuación de corrección a la presión.
6. Se corrige la presión mediante $p = p^* + p$.
7. Se corrige la velocidad mediante ecuaciones 3.23, 3.25 y 3.26.
8. Se verifica el criterio de convergencia:
   a) Si $b_p \leq \epsilon_s$ ir al paso 9.
   b) Si $b_p > \epsilon_s$ regresar al paso 2.
   donde ϵ_s es la tolerancia especificada.
9. Fin.

Cuando los problemas tienen dependencia temporal, los pasos para encontrar la solución son:

1. Inicializar la malla(dx, dy,dz) y definir el paso de tiempo (dt).

2. Inicializar T, u,v,w y p.
3. Definir $t = t + dt$, $u^0 = u$,$u^0 = u$,$v^0 = v$,$w^0 = w$,$p^0 = p$,$T^0 = T$.
4. Aplicar el método SIMPLEC al conjunto de ecuaciones actualizadas.
5. Si $t > t_{max}$ se termina, sino regresar al paso 3.

3.5. Solución de los sistemas lineales

El objetivo de discretizar las ecuaciones como se ha hecho anteriormente, es obtener la aproximación a su solución por medio de un sistema de cómputo, donde únicamente se trabaja con sistemas discretos. Una de las consecuencias de dicho procedimiento es contar con sistemas de ecuaciones como el mostrado en la ecuación 3.6, un sistema de ecuaciones lineales cuya solución se puede encontrar por diferentes métodos. Los métodos directos son una opción, sin embargo conforme crece el número de ecuaciones en este sistema, los tiempos de cómputo requeridos son mucho mayores y por tanto es necesario buscar otras opciones para acelerar dicho cálculo. Esto nos lleva a otro grupo de métodos conocido como iterativos, que permiten una alta paralelización y una convergencia a la solución del sistema en una cantidad considerablemente menor de iteraciones.En este trabajo, se usa el método de BiCGStab que es un método ampliamente adoptado por las bibliotecas que incluyen métodos de solución a sistemas lineales.

3.5.1. CGM

El método del gradiente conjugado (Conjugate Gradient Method) o CGM es un método iterativo muy popular para resolver sistemas de ecuaciones grandes. Es efectivo para sistemas de la forma:

$$\mathbf{A}x = b \tag{3.29}$$

donde se quiere obtener a x a partir de los valores conocidos de $\mathbf{A}$ y b, y donde $\mathbf{A}$ es una matriz cuadrada, simétrica, y positiva definida. Si $\mathbf{A}$ es densa[1] el tiempo requerido para factorizar dicha matriz es rigurosamente equivalente al tiempo requerido para resolver el sistema de forma iterativa, y una vez que $\mathbf{A}$ es factorizada el sistema puede ser resuelto para múltiples valores de b, lo que no sucede con el método iterativo. Sin embargo el CGM permite una mejor administración de la memoria y es más rápido en la solución con matrices dispersas[2]. Este método forma parte de un conjunto de técnicas

[1] Densa se refiere a que la mayoría de sus elementos tienen valores distintos de 0.

[2] Una matriz dispersa tiene en su mayoría elementos que valen 0.

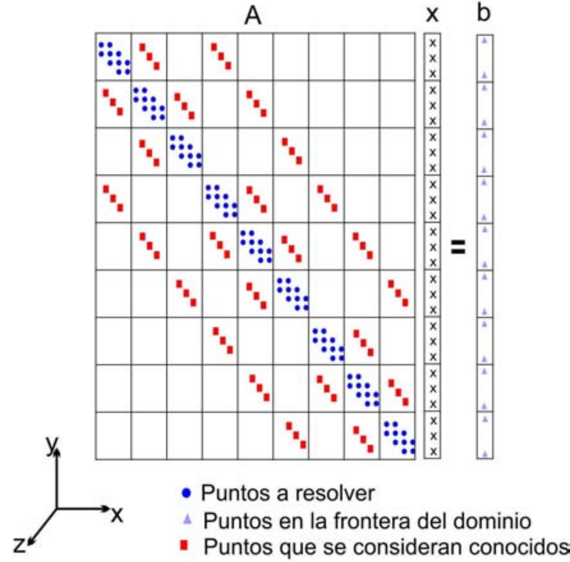

Figura 3.7: Sistema lineal generado por una malla en 3D con volumenes cúbicos.

de proyección ortogonal dentro de un subespacio de Krylov, los detalles pueden verse en la bibliografía[38, e.g].

Para problemas tridimensionales la ecuación discretizada toma la forma siguiente:

$$-a_S\phi_S - a_W\phi_W + a_P\phi_P - a_E\phi_E - a_N\phi_N - a_F\phi_F - a_B\phi_B = S_P \tag{3.30}$$

como se bosqueja en la figura 3.7. Para resolver el sistema usando CGM se usa el siguiente algoritmo:

1. Se selecciona un vector inicial x_0 que puede ser un vector nulo.
2. Se establece un valor máximo de iteraciones en caso de no convergencia y un valor límite de convergencia.
3. Se definen dos vectores $d_0 = r_0 = b - \mathbf{A}x_0$
4. Se define $\alpha_k = \frac{r_k^T r_k}{d_k^T \mathbf{A} d_k}$
5. Se asigna $x_{k+1} = x_k + \alpha_k d_k$.
6. Se asigna $r_{k+1} = r_k - \alpha_k \mathbf{A} d_k$.
7. Se define $\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$.

8. Se asigna $d_{k+1} = r_{k+1} + \beta_{k+1} d_k$.

9. Se revisa, si $\|r\| > tol$ y si $k < k_{max}$ se regresa al paso 4.

10. Si $\|r\| < tol$ o si $k < k_{max}$ se termina el proceso.

Como se ve del algoritmo anterior, los pasos son relativamente pocos y en cada paso se puede implementar una paralelización, sobre todo en sistemas de memoria compartida (en el siguiente capítulo se describe este tipo de sistemas), es por eso que este método es ampliamente ocupado, pero sólo funciona con matrices simétricas. Por tanto en la siguiente sección se abordará el método de BiCGStab que requiere una mayor cantidad de iteraciones, pero funciona con matrices no simétricas.

3.5.2. BiCGStab

El método de CGM no puede resolver los sistemas si la matriz **A** de la ecuación 3.29 no es simétrica, por tanto en esta sección se abordará de manera breve otro método iterativo llamado BiCGStab que es una combinación de BiCG y GMRES [39], y forma parte de un conjunto de técnicas de proyección ortogonal dentro de un subespacio de Krylov al igual que CGM[38, e.g]. Entre sus ventajas están la convergencia más rápida y suave que BiCG y GMRES, y la resolución numérica de los sistemas de ecuaciones lineales no simétricos, es decir, sistemas de la forma 3.29 donde la matriz de valores no es simétrica.

Este método es implementado por la mayoría de las bibliotecas con soporte a soluciones de sistemas de ecuaciones lineales y el procedimiento es el siguiente:

1. Se selecciona un vector inicial x_0 que puede ser un vector nulo.

2. Se calcula $r_0 = b - \mathbf{A}x_0$ y se selecciona un vector r_0^* tal que $r_0 \cdot r_0^* \neq 0$.

3. Se define un vector $p_0 = r_0$.

4. Se establece un valor máximo de iteraciones en caso de no convergencia y un valor límite de convergencia.

5. Se define $\alpha_j = \frac{r_j^T r_0^*}{(\mathbf{A}p_j) r_0^*}$.

6. Se define $s_j = r_j - \alpha_j \mathbf{A} p_j$.

7. Se define $w_j = \frac{(\mathbf{A}s_j)\cdot s_j}{(\mathbf{A}s_j)\cdot(\mathbf{A}s_j)}$.

8. Se asigna $x_{j+1} = x_j + \alpha_j p_j + w_j s_j$.

9. Se asigna $r_{j+1} = s_j + w_j \mathbf{A} s_j$.

10. Se define $\beta_j = \frac{r_{j+1} \cdot r_0^*}{r_j \cdot r_0^*} \times \frac{\alpha_j}{w_j}$.
11. Se asigna $p_{j+1} = r_{j+1} + \beta_j \left(p_j - w_j \mathbf{A} p_j\right)$.
12. Se revisa, si $\|r\| > tol$ y si $k < k_{max}$ se regresa al paso 5.
13. Si $\|r\| < tol$ o si $k < k_{max}$ se termina el proceso.

Con este algoritmo se resolverán los sistemas lineales no simétricos generados a lo largo de este trabajo, debido también a su alta capacidad de paralelización en las secciones de multiplicación de matriz-vector, vector-vector y suma de vectores.

3.5.3. Compresión de matrices por CRS

Debido al límite en la memoria de los sistemas de cómputo, se han desarrollado métodos de almacenar matrices de una forma más eficiente. Por ejemplo, el formato CRS (Compressed Row Storage) nos permite almacenar de manera eficiente, en términos de consumo de memoria, las matrices dispersas. Donde las matrices dispersas son un tipo de matrices donde la mayoría de sus elementos tienen valor 0. Por ejemplo; los sistemas generados por la discretización por medio de Volumen finito son matrices dispersas como la que se mostró en la figura 3.7.

La forma de almacenar una matriz en formato CRS es por medio de tres arreglos unidimensionales que se describen a continuación:

- *values*: Es un arreglo de valores reales o complejos que contiene los elementos diferentes de cero, pertenecientes a la matriz dispersa.
- *columns*: Indica el número de la columna donde se encuentra el correspondiente valor diferente de cero. Este arreglo contiene sólo números enteros.
- *row_ptr*: Este es un arreglo de números enteros, que indica donde empieza un renglón de la matriz, dentro del arreglo *values*.

La longitud de los arreglos *values* y *columns* es igual al número de elementos diferentes de cero en la matriz.

Por ejemplo; tomemos la siguiente matriz:

$$A = \begin{bmatrix} 1 & -1 & 0 & -3 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

Su representación en CRS tiene la siguiente forma:

values	=	1	-1	-3	-2	5	4	6	4	-4	2	7	8	-5
columns	=	0	1	3	0	1	2	3	4	0	2	3	1	4

row_ptr	=	0	3	5	8	11	13

Donde se ve que el último valor del arreglo renglones contiene la cantidad de valores diferentes de cero en la matriz A, que es igual a la cantidad de valores contenidos en los arreglos *values* y *columns*.

3.6. Discusión

En este capítulo se vieron los métodos de discretización para el conjunto de ecuaciones 2.21, 2.22 y 2.23, con dichos procedimientos se vio como llegar a sistema de ecuaciones lineales y los métodos de solución a dichos sistemas; se abordó problemas como acoplamientos, y los modelos para condiciones de frontera, términos convectivos y difusivos. Al final se tiene un conjunto reducido de ecuaciones que permiten obtener soluciones discretas al conjunto de ecuaciones mencionadas.

Los temas abordados en este capítulo nos proporcionan las herramientas necesarias para programar un conjunto de rutinas, que nos permitan obtener la solución de una amplia variedad de problemas relacionados con flujo. Sin embargo este tipo de problemas ya está bastante estudiado, es por eso que el objetivo de este trabajo es realizar un estudio de los tiempos de solución. La descripción de los sistemas de cómputo se hará en el siguiente capítulo, así como un planteamiento más preciso de los problemas abordados.

Capítulo 4

Modelos Computacionales

Para mejorar el rendimiento de los procesadores, desde mediados de los 80's se hizo lo posible por aumentar la frecuencia. Sin embargo este incremento en la frecuencia generaba un mayor consumo energético por parte del procesador, lo que frenó el crecimiento de frecuencias en los procesadores alrededor del 2004.

En los últimos años la necesidad de realizar cómputo más rápido, particularmente en las áreas de ciencias, nos ha llevado al desarrollo del cómputo paralelo, a pesar de que la idea de paralelizar procesos es tan antigua como las computadoras mismas[40].

En 1968 E.W. Dijkstra propone el primer modelo matemático para el manejo de procesos secuenciales co-operantes[41], de esta manera pretende resolver el problema de "la respuesta automatizada de una computadora ante la llegada de una gran variedad de mensajes en algún momento impredecible", con lo que se inicia una serie de estudios acerca de la programación de procesos concurrentes y con esto el cómputo paralelo a nivel de software.

Por otro lado la investigación sobre el uso de procesadores especializados (embebidos) a ido creciendo desde los inicios de la computación, pero la fabricación de aceleradores modernos para el uso en cómputo de alto rendimiento (HPC-High Performance Computing) surge hasta después de los 2000. Alrededor de estas fechas los procesadores especializados llamados Unidades gráficas de procesamiento (Graphics Processing Units-GPUs), producidos por NVIDIA y ATI/AMD, alcanzan capacidades de cómputo suficientes para soportar cargas de trabajo para HPC, mediante arquitecturas como la que se muestra en el esquema de la figura 4.1. Con este nuevo enfoque en el uso de los GPUs, NVIDIA libera su primer beta de CUDA en el 2007 y AMD libera un paquete de herramientas (no propietarias) llamadas OpenCL en 2008. En la figura 4.1 se observa una diferencia principal entre los CPUs y los GPUs. Esta diferencia es el área verde, la cual está dedicada a las operaciones aritméticas de punto flotante. Es notable como en el GPU esta área es mucho mayor por lo que esta arquitectura permite acelerar cálculos numéricos por varios órdenes de magnitud.

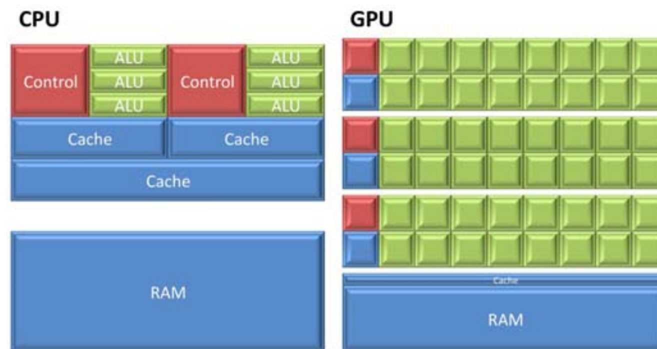

Figura 4.1: Esquema CPU vs GPU.

4.1. Paralelismo

En cómputo, el paralelismo se refiere al uso simultáneo de un conjunto de recursos enfocados en la solución de un problema específico. La forma de hacer dicha paralelización se basa en el principio de que todo problema se puede dividir en secciones más pequeñas que trabajan de manera concurrente, además, depende fuertemente del problema y de los recursos de cómputo disponibles. El objetivo de paralelizar una tarea es obtener resultados en una menor cantidad de tiempo y una mejor relación costo/-beneficio, esto último se ve desde la perspectiva de trabajar con elementos de bajo costo en la solución de un problema complejo. Aunado a esto, el desarrollo de esta área implica investigación en cuestiones como; algoritmos, planeación del software, sistemas operativos, compiladores y hardware [42, p. 1].

En la actualidad podemos ver la incursión del cómputo en cualquier actividad, desde algo tan simple como checar un correo electrónico hasta generar un modelo numérico para la evolución de galaxias, y de la misma manera estamos viendo cómo sucede lo mismo con el supercómputo en estas mismas áreas, es cada vez más común ver simulaciones más complejas en áreas de ciencias como; Predicción atmosférica, medio ambiente. Física; aplicada, nuclear, partículas, materia condensada, altas presiones, fusión, fotónica. Biociencias; biotecnología, genética. Química; ciencias moleculares. Geología; sismología. Ingeniería mecánica. Ingeniería eléctrica. Ciencias computacionales. Defensa y armamento, etc.

En aplicaciones comerciales existe una creciente demanda en procesamiento de grandes cantidades de datos a altas velocidades; "Big Data", bases de datos, minería de datos, exploración petrolera, buscadores de internet, servicios web para negocios, imágenes médicas y diagnóstico, modelos económicos y finanzas, administración de empresas multinacionales, realidad virtual, tecnologías multimedia y streming, ambientes de trabajo colaborativo, etc. [43] [44] [40].

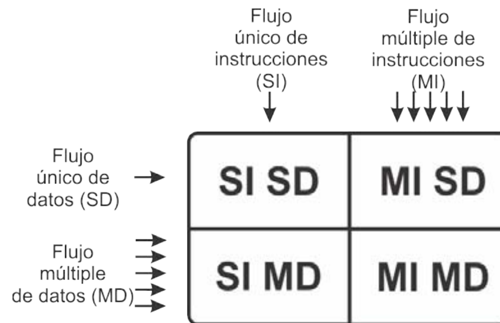

Figura 4.2: Clasificación de Flynn para la interaccion de datos y procesos.

4.1.1. Taxonomía de flynn

Los sistemas de cómputo pueden ser clasificados por varios criterios, por ejemplo; basadas en el flujo de instruciones y datos (taxonomía de Flynn), también pueden ser clasificados basados en la estructura de las computadoras, por ejemplo, múltiples procesadores teniendo memoria separada o una memoria global. Los niveles de procesamiento paralelo pueden también ser definidos basados en el tamaño de instrucciones en un programa, llamado tamaño de grano, etc.

En 1972 Micheal J. Flynn propone una clasificación para los sistemas de cómputo[45], basado en la magnitud de interacciones entre secuencias de instrucciones y datos, proponiendo cuatro categorías:

1.SISD (Single Instruction, Sigle Data). Se refiere a la capacidad de una computadora de procesar un dato con una instrucción como se hacía tradicionalmente con las computadoras caseras de los 90s, es la clasificación más pobre ya que no explota ningún tipo de paralelismo.

2.SIMD (Single Instruction, Multiple Data). Se refiere a la capacidad de procesar múltiples datos con un único flujo de instrucciones, este tipo de procesadores son los que actualmente se están usando para procesamiento gráfico y sus respectivas aplicaciones a propósito general.

3.MISD (Multiple instruction, Sigle Data). Se refiere a la capacidad de procesar un solo dato con diferentes instrucciones, este tipo de paralelismo se observa en los sistemas donde se requiere redundancia, es decir varios sistemas de respaldo que aseguren un resultado preciso.

4.MIMD (Multiple Instruction, Multiple Data) Se refiere a la capacidad de procesar muchos tipos de datos con múltiples instrucciones, los sistemas distribuidos entran en esta clasificación, pero también las computadoras multi-procesadores de memoria compartida.

En la figura 4.2 se esquematiza esta clasificación como una combinación de tipos de flujo de datos y tipos de flujo de procesos.

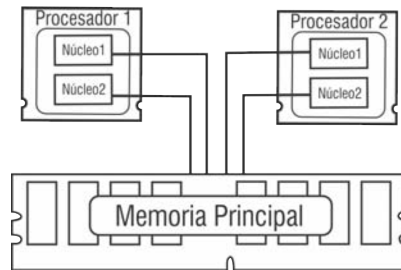

Figura 4.3: Modelo de memoria compartida, la memoria pricncipal es compartida y también es el medio de comunicación entre CPUs.

En las siguientes secciones se abordará la clasificación de sistemas de memoria compartida, memoria distribuida y memoria mixta que es una clasificación exclusiva de sistemas de cómputo en paralelo, basada precisamente en la organización de la memoria.

4.1.2. Memoria compartida

En un modelo de memoria compartida, los sistemas multiprocesadores comparten una memoria en común esto a través de una red de interconexión (bus), es un modelo para interacciones entre procesadores dentro de un sistema paralelo. Donde un valor escrito en una sección de memoria compartida puede ser accedido directamente por algún otro procesador. Físicamente la memoria compartida puede tener gran *ancho de banda*(cantidad de datos transferidos por ciclo) y baja *latencia*(tiempo de respuesta entre el tiempo de solicitud y la recepción de un dato). Sin embargo,la cantidad de procesadores compartiendo memoria puede impactar seriamente al rendimiento, debido a los accesos simultáneos a secciones de memoria.

La comunicación entre procesadores se hace a través de la memoria, véase fig. 4.3, por lo que se necesita organizar desde la programación para evitar conflictos de acceso a una misma localidad. El objetivo del rendimiento en este modelo es tener los datos en las localidades de memoria más cercanas al procesador, para que a la hora de requerirlos, el acceso sea lo más rápido posible.

Los sistemas de memoria compartida tienen la desventaja de no escalar mucho, sin embargo se pueden integrar a sistemas de memoria distribuida y mejorar el desempeño general, ya que la comunicación en cada subproceso (en memoria compartida) es mucho más rápida.

4.1.3. Memoria distribuida

Este modelo permite escalar muy fácilmente si se requiere mayor capacidad de cómputo, sin embargo la programación resulta más complicada que en memoria com-

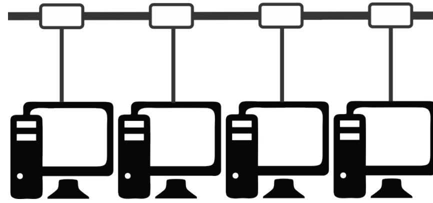

Figura 4.4: Modelo de memoria compartida, cada computadora cuenta con su procesador y su propia memoria.

partida.

Es un modelo alternativo para cómputo paralelo donde la idea principal es duplicar todas las unidades de cómputo, de esta manera tenemos múltiples computadoras en el sistema. Así, nada es compartido en este modelo. La comunicación se realiza mediante una red que interconecta a los procesadores directamente, a través del envío de mensajes.

Este modelo también es conocido como modelo multicomputadora[46] véase4.4. La memoria es estrictamente local para cada procesador, de esta manera si un CPU requiere datos que se encuentran en una sección de memoria de otro procesador, es necesario hacer una solicitud al procesador que administra esa sección de memoria. El rendimiento de este tipo de sistemas depende fuertemente de la capacidad de la red de comunicación, pero tiene la ventaja de escalar fácilmente.

4.1.4. Memoria híbrida

Este modelo es una combinación de los modelos anteriores, Dónde cada computadora cuenta con múltiples procesadores que comparten memoria localmente y a su vez se comunican con otras computadoras para realizar una tarea a fin.

Las super-computadoras más grandes del mundo actuales manejan esquemas de memoria híbrida `http://www.top500.org/`, interconectando múltiples computadoras (nodos) para trabajar como un sistema con grandes capacidades(*clusters*), por ejemplo, "Tianhe-2" en china, es un cluster que cuenta con 16,000 nodos de cómputo, cada nodo cuenta con dos procesadores Intel Ivy Bridge Xeon 12C y tres Intel Xeon Phi, además de 88 gigabytes de memoria RAM. Véase figura 4.5.

4.1.5. CUDA

Los GPUs (Unidades Gráficas de Procesamiento) están presentes en la mayoría de las computadoras actuales, en éstos se pueden realizar algunos procesos con mejor rendimiento que el CPU, por ejemplo procesar una imagen y luego desplegarla en

Figura 4.5: La supercomputadora "Tianhe-2" en china (China's National University of Defense Technology) es la número uno en el top de las computadoras más rápidas del mundo. Cuenta con cerca de 3,120,000 núcleos de procesamiento. Su uso es principalmente para simulaciones, análisis y aplicaciones de seguridad gubernamental. www.top500.org/system/177999 (fecha de actualización: 2 de junio del 2015)

pantalla, esto debido a su capacidad intrínseca de procesamiento en paralelo. Los GPU procesan polígonos, mediante sus elementos de procesamiento *shaders*, aplicando a dichos polígonos; texturas y luego realizando cálculos de sombreado e iluminación, a este proceso se le conoce como *renderizado*.

Historicamente uno de los pasos más importantes fue el desarrollo de *shaders* programables, donde éstos tuvieron la capacidad de calcular diferentes efectos directamente en el GPU. De esta manera, el renderizado pasó rápidamente a ser una tarea del GPU y no del CPU; marcando el nacimiento de las Unidades de Procesamiento Gráfico con Propósito General(GPGPU). A partir de entonces surgieron algunos grupos que trataron de acelerar el cómputo de propósito general, usando técnicas de simulación de polígonos dentro del GPU, teniendo como resultado un procesamiento altamente paralelo.

Muchas de las iniciativas dirigidas a GPGPU(e.g. BrookGPU, Cg, CTM, etc.)[47] no tuvieron tanto éxito por su difícil aprendizaje e implementación. Con la llegada de CUDA (Arquitectura Unificada de Dispositivo de Cómputo) se empezó a disminuir los efectos del cambio de paradigma en la programación de procesos en paralelo.

CUDA es una plataforma de cómputo paralelo y un modelo de programación desarrollado por NVIDIA®[48], fue desarrollada pensando en facilitar la programación directa sobre GPUs, desde un lenguaje de alto nivel extendido de C y un conjunto de bibliotecas. Los modelos de programación sobre GPUs tienen algunas abstracciones

clave; cooperación de threads organizados en grupos, memoria compartida y barreras de sincronización. De esta manera un programador de CUDA debe particionar el programa en bloques grandes que puedan ser ejecutados en paralelo, cada bloque es particionado en threads que pueden cooperar usando memoria compartida y barreras de sincronización[49].

Los diseños de GPUs son optimizados para la computación en renderizado de gráficos[1][48], pero son suficientemente generales para ser usados en muchas aplicaciones de cómputo intensivo con alto paralelismo de datos.

Un GPU consiste en un número de bloques clave;

- Memoria(global, constante, compartida).
- Streaming Mutiprocesadores(SMs)
- Streaming Procesadores(SPs)

Físicamente un GPU es un arreglo de SMs, cada uno de éstos tiene N núcleos (SP), este es el aspecto clave que permite el escalamiento de los procesadores. Un GPU consiste en uno o más SMs. Entre más SM tenga, mayor cantidad de tareas simultáneas podrán realizarse. Una de las principales formas de incrementar el rendimiento de los GPGPUs es aumentando el número de SMs y el número de SP por SM. Es por eso que cuando se diseña software para estos dispositivos, es importante considerar que la siguiente generación puede incrementar el número de SMs y SPs.

Cada SM tiene acceso a los llamados registros, que son secciones de memoria que corren a la misma velocidad que los SP, lo que genera un acceso inmediato a los datos contenidos en esas secciones. Existe también un bloque de memoria compartida accesible únicamente a los elementos de cada SM, semejante a la memoria cache en un CPU. Cada SM tiene un bus separado dentro de la memoria de texturas, la memoria constante y la memoria global. Donde la memoria de texturas es un a sección especial dentro de la memoria global, que es útil para datos donde se requiere interpolación. La memoria constante es usada para datos de sólo lectura; de igual manera, es una sección dentro de la memoria global. Por último; la memoria global es una sección de memoria disponible a todos los SM, normalmente del tipo GDDR(Doble tasa de trasferencia gráfica), que es un tipo de memoria con un ancho de banda 5 a 10 veces mayor que los encontrados el los CPU actuales[47].

Cada SM tiene dos o más unidades de propósito general (SPUs), que realizan funciones especiales como seno, coseno, operaciones de potencias, etc.

En la figura 4.6 se muestra un diagrama perteneciente a la arquitectura G80 de NVIDIA®, en este esquema se encuentran presentes los elementos más comunes los GPGPU actuales.

[1] Para referencia más extensa acerca del desarrollo histórico de CUDA, véase [e.g. 50, p.13][e.g. 47, p. 4]

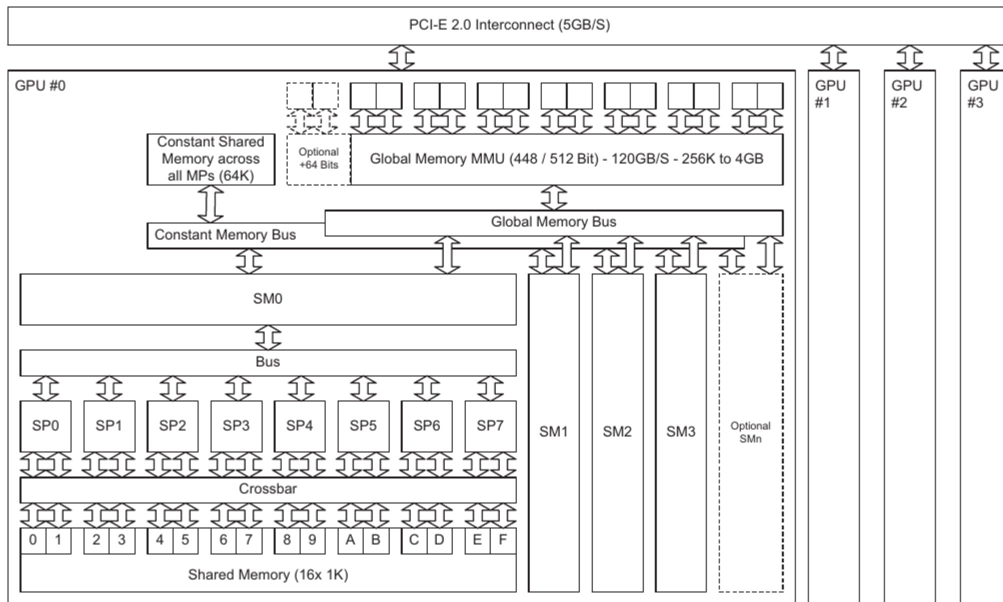

Figura 4.6: Diagrama de GPU (G80/GT200) tomado de la referencia [47, p. 44].

Para desarrollar software en CUDA, es necesario contar con los siguientes elementos como mínimo:

- Un GPU certificado para CUDA.
- Un controlador para dispositivos NVIDIA®.
- Un kit de desarrollo específico de CUDA (Compilador, bibliotecas, etc.).
- Un compilador estándar de C.

Todo esto se encuentra disponible de manera gratuita a través de la red, para sistemas operativos como Windows, Linux y MacOS.

Los GPUs tienen una pequeña diferencia con los sistemas SIMD. Recordemos que en una arquitectura del tipo SIMD, se aplica una función fija a un conjunto de datos, mientras que; en el modelo implementado por NVIDIA®, llamado SIMT(instrucción única-multiples hilos), las instrucciones no son funciones fijas, en su lugar el programador define, por medio de un *kernel*[1] qué es lo que se debe hacer con cada uno de los datos. Así, el *kernel* debe leer los datos uniformemente y el código del *kernel* debe ejecutar las transformaciones a los datos como sea necesario [47].

[1] Un kernel es la sección de código que será ejecutada sobre el dispositivo GPGPU.

4.1.6. OpenCL

OpenCL(Lenguaje de Computación Abierto) es una API libre y de código abierto, basada en el lenguaje de programación C. Es financiada por los principales fabricantes de microchips del mundo y desarrollada para éstos, lo que le permite tener una gran portabilidad. OpenCL es la principal alternativa a CUDA y a pesar de soportar una gama más amplia de Hardware, el desarrollo a partir de OpenCL es menor, debido principalmente a una mayor complejidad en la programación y a que CUDA fue presentada antes[47].

Para resolver problemas de propósito general OpenCL simula fragmentos de código con sombreado de pixeles en el contexto de gráficos, como se hace en CUDA. De la misma manera; uno de los conceptos clave es el kernel. Las instancias del kernel son ejecutadas concurrentemente sobre un Grid virtual y desde el código se define las dimensiones del particionamiento. Durante la ejecución, se agrupan las unidades de cómputo de acuerdo al tamaño de las unidades de trabajo. Donde una unidad de cómputo es un CPU mutltinúcleo, un GPU o cualquier dispositivo especializado para el que exista un driver OpenCL. Y donde las unidades de trabajo son el equivalnete a los SM en CUDA.

El surgimiento de herramientas como CUDA u OpenCL impulsó el desarrollo de muchas técnicas y recursos para cómputo acelerado de alto nivel (CUSP, CUBLAS, ViennaCL, etc) que facilita a los programadores el crear aplicaciones con tecnologías de GPUs.

4.1.7. ViennaCL

ViennaCL es un conjunto de bibliotecas de alto nivel, desarrolladas sobre el lenguaje de programación C++, que permiten seleccionar una compilación basada en CUDA, OpenCL o el estándar OpenMP[51][52], sin modificar el código. De esta manera facilita realizar programas que ejecuten procesos sobre aceleradores gráficos (GPUs), aceleradores multinúcleos como Xeon phi de Intel (MIC) o un CPU multinúcleo.

Estas bibliotecas están enfocadas a soluciones de sistemas de álgebra lineal; cuenta con funciones de BLAS, nivel 1,2 y 3. Además cuenta con subrutinas para la solución de sistemas matriciales por métodos iterativos y problemas de eigenvalores; esto a través de diferentes tipos de matrices comprimidas, además de integrar código de más bajo nivel.

Sus autores sostienen que ViennaCL fue diseñada para ser una biblioteca de fácil uso, ya que se diseñó para simplificar tanto como sea posible la transición a cómputo sobre GPU.

Por ejemplo, si se tiene una matriz $\mathbf{A}$ en formato CRS de tamaño N y n valores distintos de 0, y se quiere encontrar la solución al sistema $\mathbf{A}x = b$, el algoritmo mediante ViennaCL tiene la siguiente forma:

```
1 #include "viennacl/compressed_matrix.hpp"
```

```
 2 #include "viennacl/linalg/bicgstab.hpp"
 3 #include "viennacl/vector.hpp"
 4 void solve_viennacl(int *row_ptr,int *columns,double *values,
 5        std::vector<double> &b_cpu,std::vector<double> &x_cpu,
 6                        double tol, int max_iter,int N,int nnz)
 7 {   //Inicializacion de vectores y matrices
 8     viennacl::compressed_matrix<double>    matrizA_gpu(N,N);
 9     viennacl::vector<double>       x_gpu(N),       b_gpu(N);
10     //Construccion en GPU de matriz A
11     matrizA_gpu.set(row_ptr,columns,&values[0],N,N,nnz);
12     //Copia a GPU de vector b
13     viennacl::copy(b_cpu.begin(), b_cpu.end(), b_gpu.begin());
14     // Solucion del sistema mediante BiCGStab
15     viennacl::linalg::jacobi_precond< MatVCL> vcl_jacobi(A_gpu,
16                                  viennacl::linalg::jacobi_tag());
17     viennacl::linalg::bicgstab_tag custom_bicgstab(tol, max_iter);
18     x_gpu = viennacl::linalg::solve(A_gpu, b_gpu,
19                        custom_bicgstab,vcl_jacobi);
20     //Copia del resultado a la memoria principal
21     viennacl::copy(x_gpu, x_cpu);
22 }
```

Donde `row_ptr` es el arreglo que almacena los puntos donde comienzan los renglones, `columns` es el arreglo que almacena las posiciones de las columnas y `values` almacena los valores de dicha matriz, como se describió en la sección 3.5.3. La variable `nnz` es la cantidad de valores diferentes de 0, `tol` es la tolerancia que se le pide al método como condición de convergencia y `max_iter` es la cantidad máxima de iteraciones en caso de no obtener convergencia. En este caso se usa un método de BiCGStab y el resultado final se coloca en el arreglo `x_cpu`.

Este código se puede dividir en un par de funciones que servirán de referencia para un estudio de rendimiento en este mismo trabajo;

- **Transferencia_CPU – GPU**: Las líneas 11 y 13 constituyen la transferencia a GPU de los datos del sistema de ecuaciones almacenados en CPU, para su posterior solución.

- **Precondicionamiento**: Las líneas 15 y 16 constituyen el precondicionamiento, en este caso *precondicionamiento de jacobi*, que junto con las líneas 18 y 19

resuelven el sistema de ecuaciones. En caso de resolver sin precondicionamiento; se omiten las líneas 15 y 16, y el último argumento de la línea 19 (`vcl_jacobi`).

- **BiCGStab_GPU**: Formalmente la implementación del método de solución por medio de BiCGStab lo constituyen las líneas 18 y 19, todos los demás pasos contenidos en esta sección de código son innecesarios si el sistema se resolviera sobre CPU.

- **Transferencia CPU – GPU**: Una vez resuelto el sistema lineal dentro de la memoria del GPU es necesario colocarla en la memoria principal para su posterior manipulación, esto se hace mediante la línea 21.

4.2. Medidas de rendimientos

El concepto de rapidez se define de manera cualitativa según las necesidades del usuario, por ejemplo; para un usuario común la rapidez de una computadora se mide en términos del menor tiempo que un tarda una tarea en completarse, sin embargo para un administrador de un centro de cómputo la idea de rapidez de una computadora se mide en términos de la mayor cantidad de tareas que termina dicha computadora en intervalo de tiempo. De esta manera el usuario común quiere reducir el tiempo de respuesta mientras el administrador del centro de cómputo requiere incrementar el ancho de banda[53].

Es por eso que para medir rendimiento, de un sistema de cómputo, se definen métricas. A continuación se describen algunas métricas que se usarán para este trabajo:

- Tiempo de ejecución:

  Para un programa ejecutándose en serial, el tiempo de ejecución T_s es el tiempo que tarda en completarse todo el programa, desde que se inicia el primer proceso hasta que se detiene el programa, con el último proceso también finalizado.

  Para un programa en paralelo, la definición es equivalente, es decir; el tiempo de ejecución T_p es el tiempo que tarda en completarse todo el programa, desde que se inicia el primer proceso hasta que se detiene el programa, con el último proceso también finalizado. En este caso de manera intrínseca se toman en cuenta el tiempo de cálculo, el tiempo de inactividad y el tiempo de comunicación.

- Aceleración:

  Esta es una métrica enfocada a los sistemas en paralelo, su objetivo es medir el beneficio, en tiempo de ejecución, de un programa en paralelo contra su versión serial. Se define como la razón que hay entre el tiempo de ejecución del programa

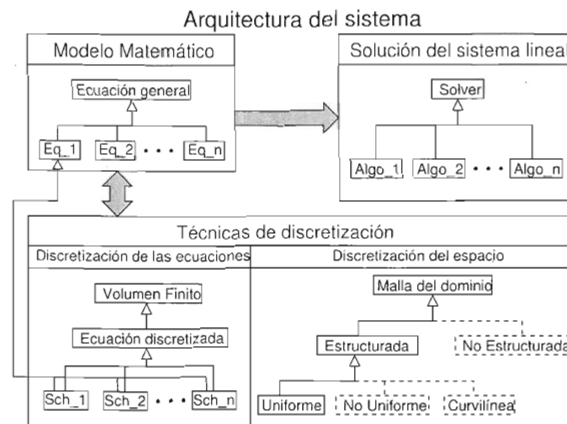

Figura 4.7: Diagrama de la arquitectura sobre la que se desarrolló TUNAM

serial T_s entre el tiempo de ejecución de la versión en paralelo T_p con n_p procesadores idénticos, de esta manera $S = T_s/T_p$. La aceleración ideal está definida como $S = n_p$.

4.3. TUNAM

TUNAM es una biblioteca que permite resolver las ecuaciones del problema de convección natural, en mallas ortogonales, usando el método de volumen finito, y los algoritmos descritos en el capítulo 3. Para mayores detalles de esta biblioteca véase [54][32].

En el proceso de desarrollo del software es importante iniciar con el diseño y la descripción de una arquitectura central, a partir de la cual se determine las componentes o módulos que deben ser construidos; los módulos deben tener baja dependencia entre ellos para permitir un desarrollo y mantenimiento simple.

La arquitectura de un software es la forma en que se organizan los diferentes componentes que lo integran. Una buena arquitectura permite que cada componente pueda ser reutilizada, modificada o sustituida sin afectar otras partes. Bajo esta filosofía se desarrolló TUNAM y en la figura 4.7 se observa el diagrama original de la arquitectura de este sistema.

EL `modelo matemático` describe el fenómeno de convección mediante un conjunto de ecuaciones diferenciales de segundo orden, parciales, no lineales y acopladas.

Las `técnicas de discretización` son las técnicas con las que se genera la malla del dominio y la posterior traducción de las ecuaciones continuas, a su versión discreta. Dada la malla, es posible elegir un esquema numérico que nos convenga, según el

problema a resolver.

La `solución del sistema de ecuaciones` ; después de la discretización obtenemos un sistema de ecuaciones algebraicas, donde existe un gran variedad de bibliotecas que se enfocan en resolver este tipo de sistemas, por ejemplo; ViennaCL, eigen, cusp, etc. Se requiere que este módulo trabaje con diferente tipos de datos y que sea independiente del método de discretización y la forma de la malla.

El método de solución para el sistema $\mathbf{A}x = b$, generado por la discretización a partir de volumen finito en TUNAM, es TDMA; un método optimizado para ejecución en serial, véase [e.g. 54, p.52].

4.3.1. BiCGStab adaptado a TUNAM

Para solucionar el sistema $\mathbf{A}x = b$ en TUNAM se utiliza un función de nombre `TDMA3D` y su equivalente para este trabajo es la función `solver_bicgstab_3D` que entre sus funciones agrega algunos ajustes necesarios para la manipulación y solución mediante GPU. Al hacer referencia a cualquiera de estas dos funciones en la sección de resultados, se hará con el sobrenombre de *Solver Principal*.

A continuación se coloca la sección de código correspondiente a `solver_bicgstab_3D`, y una descripción de las principales secciones de dicho código.

```
1  int solver_bicgstab_3D(ecuacionGeneral,ValueType tol,int max_iter,
2                         int N, int nnz, int nx, int ny, int nz)
3  { std::vector<double>       b_cpu(N),   x_cpu(N);
4    int       *row_ptr = NULL,   *columns = NULL;
5    double      *values = NULL;
6    int           ii,i,j,k,iter;
7    ii = 0;
8    for( i = 1; i <= nx; i++)
9     for( k = 1; k <= nz; k++)
10     for( j = 1; j <= ny; j++, ii++)
11               b_cpu[ii]=-ecuacionGeneral.sp(i,j,k);
12   csr_3D(row_ptr,columns,values,ecuacionGeneral, N, nnz, nx, ny,nz);
13   solve_viennacl(row_ptr,columns,values,b_cpu,x_cpu,
14                                       tol,max_iter,N,nnz);
15   ii = 0;
16   for( i = 1; i <= nx; i++)
17    for( k = 1; k <= nz; k++)
18     for( j = 1; j <= ny; j++, ii++)
```

```
19              ecuacionGeneral.phi(i,j,k) = x_cpu[ii];
20 }
```

- **Compresión a CSR** : Mediante esta función se pasa de un formato de almacenamiento por diagonales de la matriz **A** al formato CSR, el tiempo requerido para esta función es considerable, como se verá en la sección de resultados, y está implementado en la línea 12 mediante la función `CSR_3D` que no se anexa debido a su gran extensión en líneas de código.
- **Solución con ViennaCL**: Esta función permite una solución del sistema $\mathbf{A}x = b$ y está aislada para reutilizarla en sistemas de 1D y 2D, está implementada en la línea 13 y su contenido se encuentra en la sección 4.1.7.

4.4. Discusión

La llegada de los GPUs al cómputo de propósito general implicó adoptar en principio un modelo de memoria híbrida, donde se maneja una parte distribuida; es decir, se dividen tareas que son enviadas a un dispositivo con memoria independiente y luego dicho dispositivo regresa el resultado del proceso mediante la sincronización de sus respectivas secciones de memoria. Pero una vez que el GPU cuenta con los datos necesarios para empezar a procesar; tenemos que utilizar un enfoque de memoria compartida, esto entre otras cosas implica que a la hora de trabajar en optimización de código ejecutándose sobre GPU, debemos tomar en cuenta los tiempos de sincronización entre secciones de memoria(Memoria de CPU $\leftrightarrow$ Memoria de GPU), que a su vez deriva en que es necesario decidir dónde se obtendrá mayor rendimiento dependiendo del tipo de tarea; CPU ó GPU.

Actualmente uno de los principales problemas en las tarjetas de video es la poca memoria con la que cuentan, ya que en promedio se tiene 2GB, que fue una limitate en nuestros casos de estudio(véase la sección de resultados), sin embargo NVIDIA$^{\text{TM}}$ya está poniendo énfasis en mejorar esta capacidad, y actualmente existen tarjetas con 24 GB de RAM, que aunque los precios aun no son accesibles tendremos estas capacidadaes al alcance muy pronto.

Capítulo 5

Resultados numéricos

En este capítulo se presentará un conjunto de tablas y gráficas, con los resultados de rendimiento arrojados por el cálculo de la solución de sistemas de flujo convectivo natural.

Cómo se verá más adelante, la sección que más tiempo consume de todo el proceso es la que soluciona el sistema lineal generado por la discretización de las ecuaciones de flujo, es por eso que se enfocó en esa sección, y en particular se seleccionó el método de BiCGStab que es el más adecuado para el tipo de sistemas generados, ya que se tienen matrices dispersas y no simétricas, y se requiere de un método iterativo dado el tamaño de los sistemas a resolver (mayor a 10^5 elementos).

Las pruebas se hacen con un enfoque en el rendimiento del GPU y del método de solución (en este caso BiCGStab). De esta manera las pruebas se realizan sobre mallas con intervalos de distintos tamaños. Todo esto sobre un mismo sistema de cómputo, cuyas características se enumeran a continuación:

Sistema de pruebas
Procesador Phenom 955 AM3 3.2 Ghz
16GB RAM DDR3 1333MHz
Tarjeta de Video GTX 680 1 GB de memoria (Más especificaciones en el apéndice)
Disco duro 7200RPM 8MB de cache

5.1. Verificación del método de solución

El primer paso fue verificar los sistemas de solución para comprobar que se está solucionando el mismo problema, dicha verificación se hizo mediante el planteamiento de un problema de flujo convectivo con un dominio cúbico tridimensional de 48 puntos por cada lado, un paso de tiempo de 10^{-4}, el error permitido fue de 10^{-6} para BiCGStab, con un máximo de iteraciones de 5000 iteraciones, y con los extremos de la coordenada "y" colocados a distintas temperaturas como se muestra en la figura 5.1.

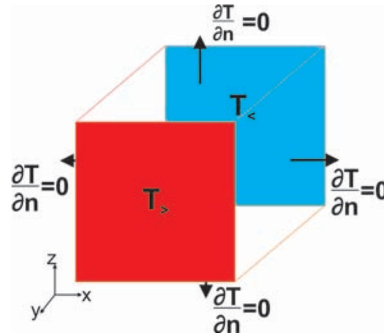

Figura 5.1: Dominio de solución para problema de calibración.

Dicho estudio de calibración se realizó comparando los métodos TDMA_3D y BiCGStab. El primero es el que viene implementado originalmente en TUNAM, mientras que el segundo es el que se implementó en este trabajo mediante la biblioteca ViennaCL. La idea es obtener los mismos resultados numéricos en ambos casos, y posteriormente comparar su rendimiento.

Como resultado se obtuvo un conjunto de gráficas que nos muestran las diferentes variables calculadas por el software ejecutándose sobre GPUs; mediante ViennaCL con CUDA, y sobre CPUs con la versión original; [54] mediante TDMA. En este caso son los mismos resultados si se usa el método de BiCGStab simple o si se aplica precondicionamiento al sistema, es por eso que sólo se coloca un conjunto de datos representativos de ambos.

La figura 5.2 muestra la evolución temporal del error en la solución del sistema para la variable temperatura, la escala es logarítmica ya que es la más indicada para ver variaciones en los resultados arrojados por los diferentes métodos. En este caso se ve que los dos sistemas calculan los mismos resultados con una diferencia entre ellos insignificante, incluso se calculó un valor RMS para la diferencia entre todos los valores, y se obtuvo un RMS para los errores en la temperatura de 0.000001422(TDMA vs BiCGStab-ViennaCL). En este caso el "error" es una medida de cuanto cambia una varible en el instante n, con respecto al instante n+1. Si este error tiende a cero, conforme avanza el tiempo, entonces significa que estamos llegando a un estado estacionario.

La figura 5.6 representa el flujo de calor a través de la pared con la temperatura más baja en nuestro dominio de estudio, lo más reelevante de estas gráficas es el hecho de que desde un principio coinciden a la perfección, de nuevo se tomó como medida de igualdad el valor RMS y se obtuvo 0.002687489 (TDMA vs BiCGStab-ViennaCL).

La figura 5.7 representa el flujo de calor a través de la pared con la temperatura más alta en nuestro dominio de estudio, y se observa que desde un principio coinciden a la perfección, de nuevo se tomó como medida de igualdad el valor RMS y se obtuvo 0.003202506 (TDMA vs BiCGStab-ViennaCL).

La figura 5.8 muestra el valor obtenido por la variable b_p en la ecuación 3.28 y entre

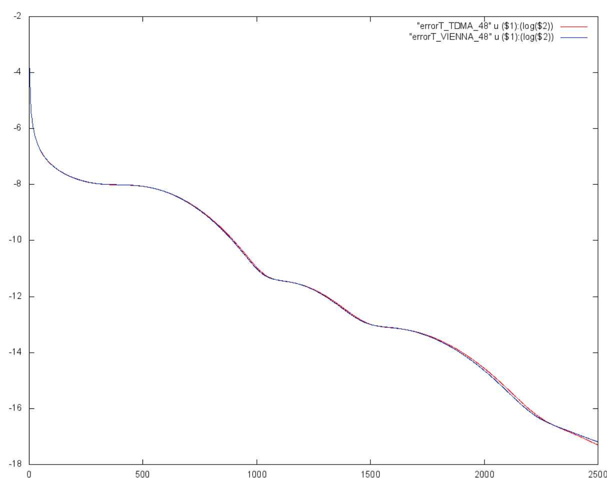

Figura 5.2: Errores en la variable temperatura calculadas a través del tiempo comparando la ejecución serial con la versión sobre GPUs, la escala es logarítmica.

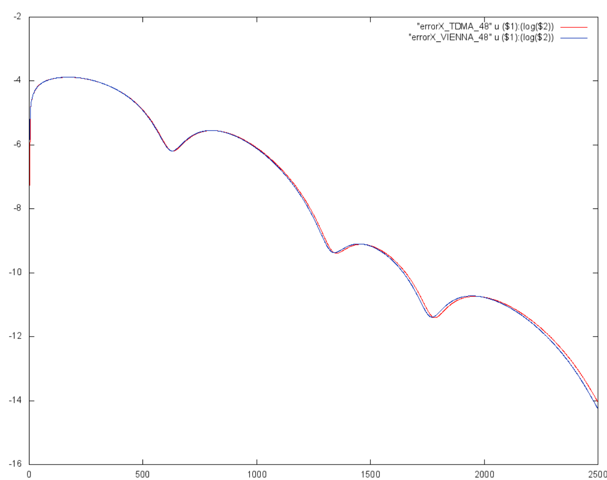

Figura 5.3: Errores en la componente X de la velocidad, calculados a través del tiempo comparando la ejecución serial con la versión sobre GPUs, la escala es logarítmica.

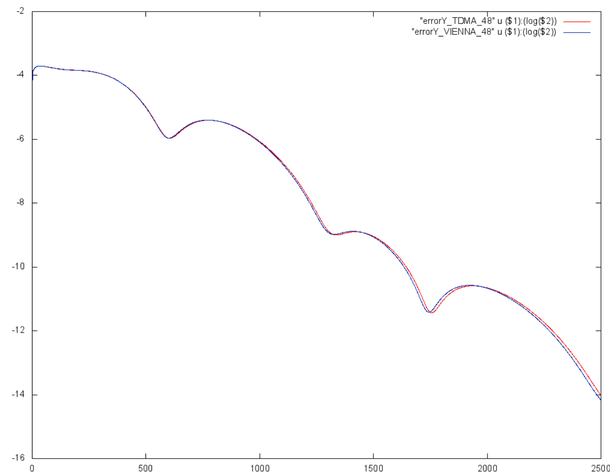

Figura 5.4: Errores en la componente Y de la velocidad, calculados a través del tiempo, comparando la ejecución serial con la versión sobre GPUs, la escala es logarítmica.

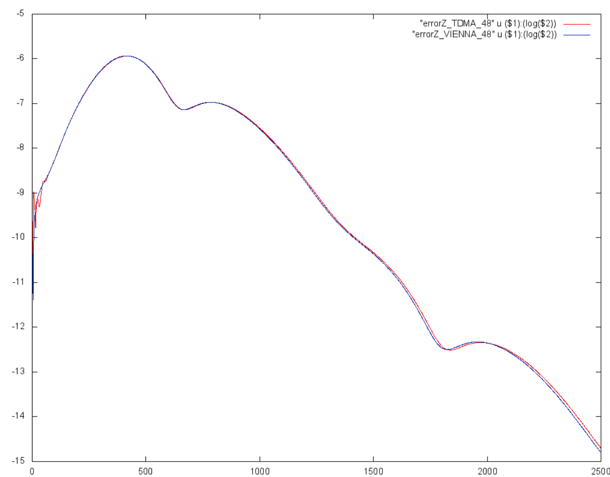

Figura 5.5: Errores en la componente Z de la velocidad, calculados a través del tiempo comparando la ejecución serial con la versión sobre GPUs, la escala es logarítmica.

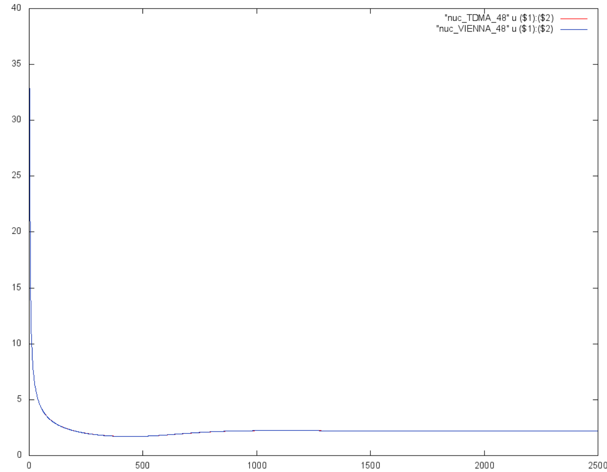

Figura 5.6: Errores en las variables de los flujos de calor en la pared con la temperatura más baja, calculados a travéz del tiempo, comparando la ejecución serial con la versión sobre GPUs, la escala es logarítmica.

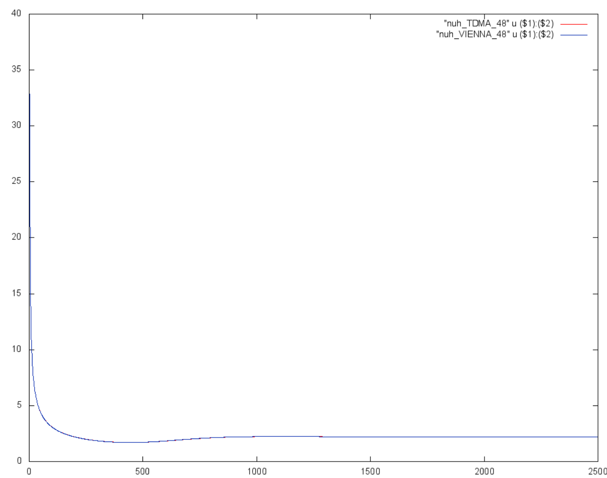

Figura 5.7: Errores en las variables de los flujos de calor en la pared con la temperatura más alta, calculadas a travéz del tiempo, comparando la ejecución serial con la versión sobre GPUs, la escala es logarítmica.

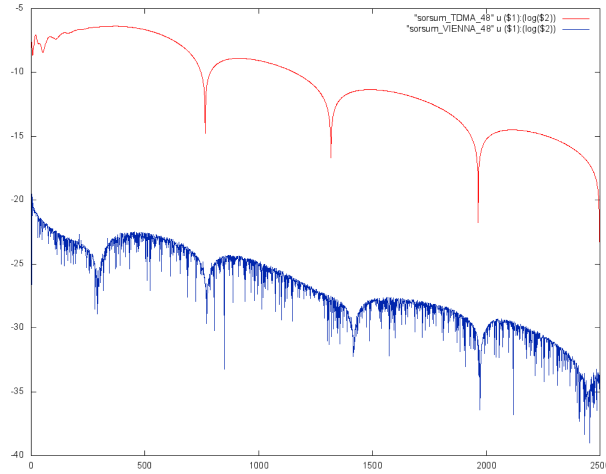

Figura 5.8: Errores en la variable que mide el error en la función de conservsación de masa en el dominio, calculadas a travéz del tiempo comparando la ejecución serial con la versión sobre GPUs, la escala es logarítmica.

otras cosas se observa su tendencia a cero lo que nos da confiabilidad en el sistema utilizado. A pesar de que no coinciden los valores con ambos métodos de solución, el más reciente (BiCGStab) es el que da una mejor solución ya que cuenta con valores más pequeños lo que nos permite tomar como confiable esta nueva herramienta.

La figura 5.9 muestra una representación de la variable temperatura para tres instantes diferentes; el estado inicial (t=0), un estado intermedio (t=500), un segundo estado intermedio (t=1000) y un estado donde ya es estacionario el flujo(t=1500). La figura 5.10 muestra una representación de las velocidades sobre el mismo dominio de estudio y con el mismo perfil de la figura 5.9, para los mismos tiempos. Por último, la figura 5.11 muestra la presión para los mismos tiempos y el mismo dominio. El mapeo de colores, tanto en la temperatura como en la presión, va del color azúl para valores menos intensos al color rojo para los valores más intensos.

5.2. Pruebas de rendimiento

El objetivo de este trabajo, como se comentó en la introducción; es medir de alguna manera la mejora en rendimiento al utilizar GPU contra la versión ya implementada sobre CPU. De esta manera se toma como base el problema ya descrito para la calibración y se miden los tiempos de ejecución de las diferentes funciones que constituyen el método de solución, las funciones reportadas son las que se describen en la sección 4.3.1. Estos tiempos se toman para mallas de diferentes tamaños, pero conservando el

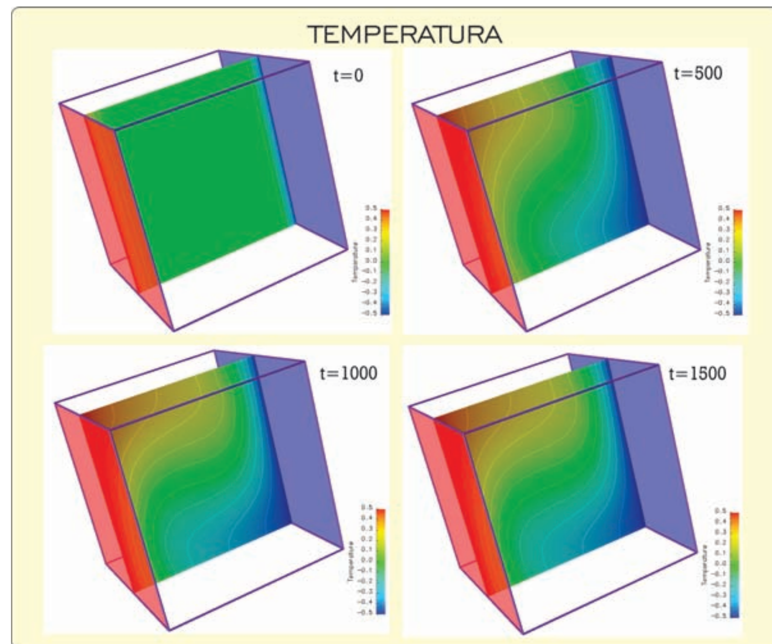

Figura 5.9: Resultado final en la simulación de flujo convectivo, mapeando la temperatura sobre un plano YZ

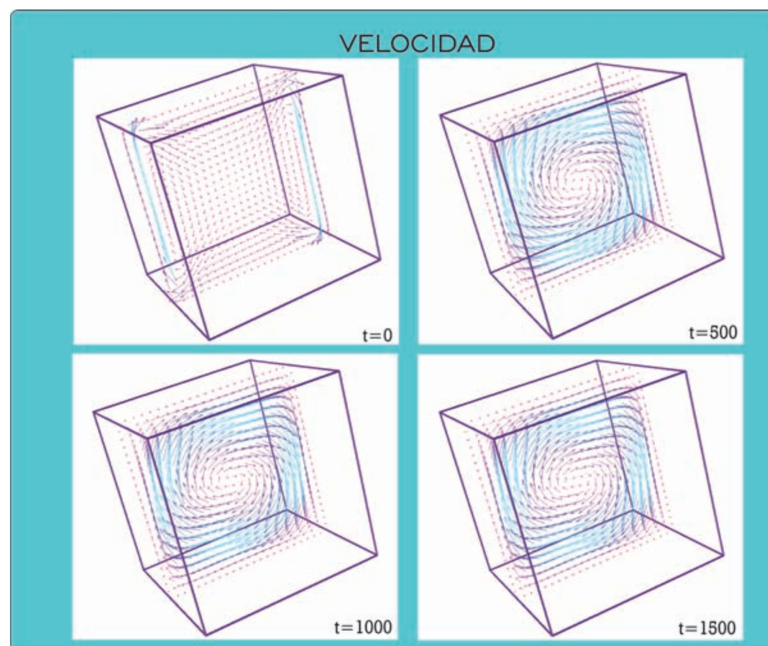

Figura 5.10: Resultado final en la simulación de flujo convectivo, mapeando la velocidad sobre un plano YZ

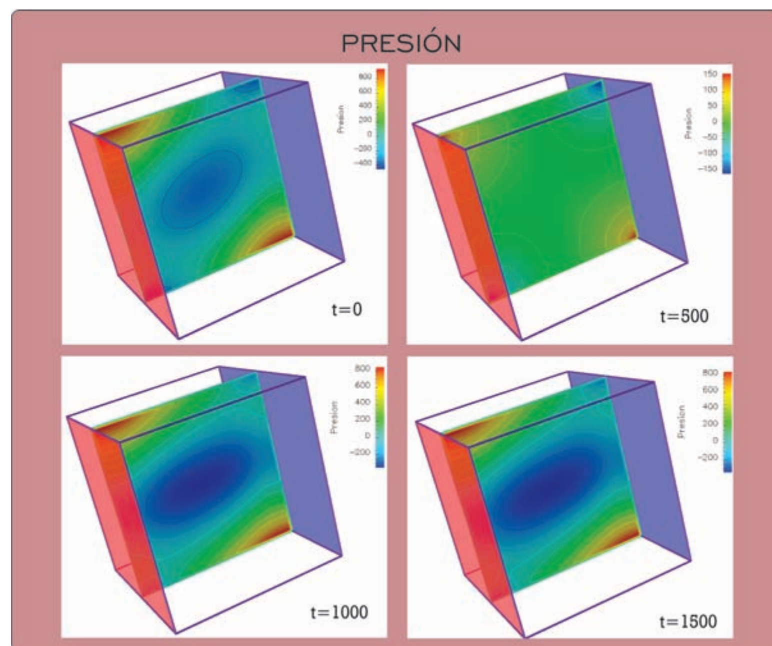

Figura 5.11: Resultado final en la simulación de flujo convectivo, mapeando la presión sobre un plano YZ

mismo planteamiento físico y las mismas condiciones para los métodos de solución.

Además de BiCGStab, también se realizaron mediciones utilizando precondicionamiento, en este caso mediante el precondicionamiento de Jacobi.

En los resultados que se presentan a continuación tomamos en cuenta lo siguiente. Dado que TDMA3D se ejecuta sobre CPU, sólo se toma el tiempo de ejecución de la función completa (*Solver Principal*). En el caso de ViennaCL (BiCGSTAB) se toman en cuenta los tiempos de sus principales componentes con la finalidad de proponer alguna optimización al final de este trabajo. Las principales componentes del solver BiCGSTAB son las siguientes:

1) Compresión a CRS: Originalmente, la matriz generada por TUNAM, cuando se aplica el método de volumen finito, se almacena en arreglos que representan las diagonales de la matriz en donde están los elementos diferentes de cero, véase figura 3.7. Este almacenamiento no es óptimo para el método BiCGSTAB, por lo que se construyó una función de transformación al formato CRS.

2) Transferencia CPU-GPU: La matriz y el lado derecho (RHS) del sistema se deben transferir a la memoria del GPU. Dado que la matriz puede ser muy grande, estos tiempos de transferencia pueden ser muy grandes.

3) Precondicionamiento: En esta parte se construye una matriz M tal que al multiplar el sistema por M, el condicionamiento del sistema disminuye con lo que en principio el número de iteraciones del algoritmo de solución debe reducirse. En nuestro caso usamos el precondicionador de Jacobi, que consiste en una matriz cuya diagonal es la inversa de la diagonal de la matriz original, y los elementos fuera de la diagonal son todos cero. Este precondicionador es muy fácil de construir, sin embargo puede que no reduzca sustancialmente las iteraciones. Otros precondicionadores son más complicados y toman tiempo adicional para su construcción, por lo que se decidió para este trabajo no estudiarlos. Además, el precondicionador de Jacobi ya viene implementado en ViennaCL.

4) BiCGStab_GPU: Algortimo BiCGStab.

5) Transferencia GPU-CPU: Una vez obtenida la solución, ésta se debe transmitir al memoria del CPU para su posterior procesamiento. Dado que es un solo vector, esta transferencia es mucho menor que del inciso 2).

En la tabla 5.1 se presentan los tiempos en segundos requeridos para resolver los sistemas lineales. Se muestran los tiempos para TDMA3D y ViennaCL (BiCGSTAB) con y sin precondicionamiento. En este caso por cada variable se genera un sistema lineal en cada paso de tiempo. De esta manera si tenemos 5 variables y en todas las pruebas se tomaron 2500 pasos de tiempo, tanto para TDMA3D como para ViennaCL (BiCGSTAB) se ejecutaron 12500 veces a lo largo de cada solución.

Observamos que el tiempo total del programa con TDMA es de 455.81 segs, de estos 239.88 se usaron en la solución de los sistemas, es decir el más del 50 % del proceso total. En el caso de BiCGSTAB, el tiempo del solver principal fue de 409.55 segs, lo cual es más del 62 % del tiempo total de procesamiento (622.65 segs). De los 409.55segs.

Función	**TDMA**	**ViennaCL GPU**	**ViennaCL GPU Precondicionado**
Solver Principal	239.88	409.55	456.25
Compresión a CSR	_	249.56	249.00
Transferencia CPU-GPU	_	82.58	83.80
Precondicionamiento	_	_	0.10
BiCGStab_GPU	_	31.37	77.76
Transferencia GPU-CPU	_	0.14	0.17
Tiempo total	455.81	622.65	681.08

Tabla 5.1: Tiempos obtenidos para una malla de 48x48x48 nodos.

Función	**TDMA**	**ViennaCL GPU**	**ViennaCL GPU Precondicionado**
Solver Principal	717.29	979.70	991.65
Compresión a CSR	_	606.27	600.03
Transferencia CPU-GPU	_	199.10	201.95
Precondicionamiento	_	_	0.01
BiCGStab_GPU	_	63.34	76.32
Transferencia GPU-CPU	_	0.24	0.17
Tiempo total	1583.49	1842.57	1894.82

Tabla 5.2: Tiempos obtenidos para una malla de 64x64x64 nodos.

Función	TDMA	ViennaCL GPU	ViennaCL GPU Precondicionado
Solver Principal	2263.68	3451.30	3439.80
Compresión a CSR	_	2184.94	2122.98
Transferencia CPU-GPU	_	696.43	697.36
Precondicionamiento	_	_	0.01
BiCGStab_GPU	_	183.59	231.68
Transferencia GPU-CPU	_	0.57	0.31
Tiempo total	5321.08	6418.31	6384.03

Tabla 5.3: Tiempos obtenidos para una malla de 96x96x96 nodos.

Función	TDMA	ViennaCL GPU	ViennaCL GPU Precondicionado
Solver Principal	10236.54	9900.30	10089.65
Compresión a CSR	_	6313.12	6857.32
Transferencia CPU-GPU	_	1721.91	1726.15
Precondicionamiento	_	_	0.10
BiCGStab_GPU	_	428.29	497.12
Transferencia GPU-CPU	_	0.14	0.53
Tiempo total	17860.90	17210.42	17682.68

Tabla 5.4: Tiempos obtenidos para una malla de 128x128x128 nodos.

Función	**TDMA**	**ViennaCL GPU**	**ViennaCL GPU Precondicionado**
Solver Principal	25057.00	20902.90	20955.67
Compresión a CSR	_	13908.72	13945.47
Transferencia CPU-GPU	_	4038.98	1097.53
Precondicionamiento	_	_	0.31
BiCGStab_GPU	_	1711.24	1879.63
Transferencia GPU-CPU	_	2.61	1.39
Tiempo total	46579.14	41246.01	42054.81

Tabla 5.5: Tiempos obtenidos para una malla de 192x192x192 nodos.

el 60 % se usó en la compresión CRS (249.56 segs.); un poco más del 20 % fue para la transferencia CPU-GPU; y sólo el 7.7 % se usó en el algoritmo BiCGSTAB (31.37segs.). La transferencia de información de GPU a CPU fue mínima e insignificante (0.14segs.). Los resultados usando el precondicionador de Jacobi fueron muy similares, aunque dicha técnica en vez de reducir los tiempos de ejecución pasó lo contrario, es decir el programa tardó un poco más. Por lo tanto, para este caso, de una malla de 48^3, el precondicionador de Jacobi no es conveniente.

De estas mediciones, se puede concluir que se requiere de una mejor estrategia para almacenar las matrices provenientes de la discretización de volumen finito. Se debe buscar una manera de evitar el doble almacenamiento: primero en diagonales, y después su transformación a CRS, pues este trabajo toma el 60 % del tiempo.

Las tablas 5.2, 5.3 y 5.4 presentan resultados en tiempo similares a los que se muestran en la tabla 5.1 pero para diferentes tamaños de mallas. Se observa en estos resultados que el método BiCGStab va mejorando con respecto de TDMA conforme se aumenta el número de nodos. En el caso de la malla de 48^3 la aceleración es de 239.88 segs. / 409.55 segs. = 0.59, es decir BiCGSTAB en GPU es casi dos veces más lento que el uso de TDMA en el CPU. Sin embargo, para una malla de 128^3 los tiempos en ambos casos son casi los mismos; pero aún no se ve un beneficio del uso de los GPUs. Cabe destacar que se intentó hacer el experimento con una malla de 256^3 pero la tarjeta con la que contamos ya no tuvo la capacidad en memoria para realizar este cálculo.

En este punto, la pregunta es: ¿Qué pasaría si pudiéramos evitar la transformación a CRS, y en vez de ello se almacenaran las matrices directamente en ese formato?. La respuesta se obtiene midiendo la aceleración obtenida usando GPUs, con y sin CRS. Esta aceleración se midió para todos los casos mostrados en las tablas 5.1 a 5.4 y los

Puntos de la malla	**ViennaCL GPU**	**ViennaCL GPU Precondicionado**
48^3	0.59	0.53
64^3	0.73	0.72
96^3	0.66	0.66
128^3	1.03	1.01
192^3	1.13	1.11

Tabla 5.6: Aceleración total.

Puntos de la malla	**ViennaCL GPU**	**ViennaCL GPU Precondicionado**
48^3	1.50	1.16
64^3	1.92	1.83
96^3	1.79	1.71
128^3	2.85	3.17
192^3	3.58	3.57

Tabla 5.7: Aceleración sin tomar en cuenta CRS.

resultado se muestra en las tablas 5.6 y 5.7. Observamos que evitar la transformación CRS es conveniente para obtener un beneficio del uso de los GPUs. En especial, si las mallas son cada vez más finas, los beneficios pueden ser mayores aún. Por ejemplo, en problemas de turbulencia donde se deben resolver escalas muy finas, las mallas deben contar con varios millones de incógnitas para obtener buenos resultados.

Regresando a la arquitectura del CPU y del GPU, como se muestra en la figura 4.1, observamos que el GPU tiene mayor capacidad para el cálculo de operaciones de punto flotante. Entonces decidimos medir solamente los tiempos de procesamiento de punto flotante. Los tiempos para el TDMA3D son los del *Solver Principal*. Para el BiCGStab, las operaciones de punto flotante sólo son las que se ejecutan en BiCGStab_GPU. La tabla 5.8 muestra la aceleración para cada tamaño de malla.

Puntos de la malla	**ViennaCL GPU**	**ViennaCL GPU Precondicionado**
48^3	7.65	3.08
64^3	11.32	9.40
96^3	12.33	9.77
128^3	23.90	20.59
192^3	14.64	13.33

Tabla 5.8: Aceleración comparando sólo las operaciones de punto flotante.

Capítulo 6

Conclusiones

En este trabajo se realizó un estudio del desempeño de la solución del problema de convección natural usando varias implementaciones. Las soluciones se obtuvieron mendiante la aplicación del método de volumen finito sobre mallas ortogonales. Se utilizó el sistema TUNAM que ya tiene implementado el método y que utiliza originalmente el algoritmo TDMA3D para resolver los sistemas lineales resultantes. Nuestro objetivo fue comparar este algoritmo, el cual se ejecuta en CPU, contra una implementación del método BiCGStab el cual se ejecuta en GPUs. Esta implementación se realizón utilizando la biblioteca ViennaCL.

De los resultados obtenidos sacamos las siguientes conclusiones:1) La implementación original de TUNAM es conveniente cuando las mallas no son muy finas, hasta 96^3; 2) Cuando las mallas comienzan a refinarse, se comienza a ver los beneficios de algoritmos iterativos, como el BiCGStab, pues los sistemas además de ser dispersos, son muy grandes, y los algoritmos de tipo subespacio de Krylov son convenientes en esos casos; 3) TUNAM almacena originalmente las matrices de los sistemas en arreglos que representan las diagonales diferentes de cero, sin embargo este almacenamiento está orientado a optimizar los procesos del algoritmo TDMA3D; 4) Para hacer un uso óptimo del algoritmo BiCGSTAB se requiere que la matriz del sistema se almacene en algún formato más general como CRS (o Compressed Column Storage, CCS), por lo tanto se requiere de una transformación previa al uso del algoritmo BiCGStab; 5) Observamos que dicha transformación requirió de un tiempo excesivo, lo cual ocasionó que los tiempos de cómputo no tuvieran beneficio en GPUs; 6) Si hacemos la medición de la aceleración sin tomar en cuenta la trasnformación a CRS observamos que si hay beneficio en el uso de los GPUs, por lo tanto se sugiere realizar un estudio de ingeniería de software para modificar un poco la arquitectura de TUNAM y que sea posible almacenar las matrices de los sistemas directamente en el formato CRS; 7) Con el avance constante de la tecnología, creemos que con tarjetas con más memoria y mayores velocidades de transferencia CPU-GPU, los beneficios pueden ser bastante notables. Sobre todo en problemas donde se requieran mallas muy finas, tal es el caso de problemas de

flujo turbulento donde se deben resolver muchas escalas; 8) Si uno mide solamente los tiempos de las operaciones de punto flotante en ambas implementaciones observamos una aceleración de hasta 23x para el caso de una malla de 128^3 . Pero en la práctica, al menos por ahora, no es posible llegar a esta tasa de aceleración.

Finalmente, la biblioteca ViennaCL puede ser muy conveniente para realizar unas primeras pruebas de aceleración de códigos en GPUs. Pero una vez realizada la primera implementación, se deben buscar alternativas a ciertas implementaciones para lograr una mayor eficiencia. Por ejemplo, el uso de los diferentes tipos de memoria dentro de ViennaCL no es tan directo y eso puede limitar los resultados finales.

Apéndice A

Especificaciones de tarjetas de video

NVIDIA GeForce GTX 680

Especificaciones de la GPU	Valor
Núcleos CUDA	1536
Frecuencia de reloj normal	1006
Frecuencia acelerada	1058
Tasa de rrelleno de texturas	128.8
Frecuencia de la memoria (Gbps)	6.0
Cantidad de memoria	2048 MB
Interfaz de la memoria	256-bit GDDR5
Ancho de banda máx.	192.2

Tabla A.1: Especificaciones técnicas de la tarjeta grágica con chip GTX 680 de NVIDIA

Bibliografía

[1] J. Blazek. *Computational Fluid Dynamics: Principles and Applications*. Elsevier Science, 2015. 1

[2] C. Fletcher. *Computational Techniques for Fluid Dynamics 1*. Computational Techniques for Fluid Dynamics. Springer Berlin Heidelberg, 1991. 2

[3] Pilsung Kang, Eli Tilevich, Srinidhi Varadarajan, and Naren Ramakrishnan. **Maintainable and Reusable Scientific Software Adaptation: Democratizing Scientific Software Adaptation**. In *Proceedings of the Tenth International Conference on Aspect-oriented Software Development*, AOSD '11, pages 165–176, New York, NY, USA, 2011. ACM. 2

[4] Stan Posey. **Considerations for GPU Acceleration of Parallel CFD**. *Procedia Engineering*, **61**:388 – 391, 2013. 25th International Conference on Parallel Computational Fluid Dynamics. 2

[5] Xiaofeng He, Zheng Wang, and Tiegang Liu. **Solving Two-dimensional Euler Equations on GPU**. *Procedia Engineering*, **61**(0):57 – 62, 2013. 25th International Conference on Parallel Computational Fluid Dynamics. 2, 3

[6] Hao Zhang, F. Xavier Trias, Andrey Gorobets, Yuanqiang Tan, and Assensi Oliva. **Direct numerical simulation of a fully developed turbulent square duct flow up to Re=1200**. *International Journal of Heat and Fluid Flow*, **54**(0):258 – 267, 2015. 2

[7] Shuming Miao, Xin Zhang, Oswald G. Parchment, and Xiaoxian Chen. **A fast GPU based bidiagonal solver for computational aeroacoustics**. *Computer Methods in Applied Mechanics and Engineering*, **286**:22 – 39, 2015. 2, 3

[8] Carsten Burstedde, Omar Ghattas, Michael Gurnis, Georg Stadler, Eh Tan, Tiankai Tu, Lucas C. Wilcox, and Shijie Zhong. **Scalable Adaptive Mantle Convection Simulation on Petascale Supercomputers**. In

Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC '08, pages 62:1–62:15, Piscataway, NJ, USA, 2008. IEEE Press. 2, 3

[9] Mohamad Sindi, Majdi Baddourah, and M. Ehtesham Hayder. **Poster: High Performance Computing Reservoir Simulation in the Oil Industry**. In *Proceedings of the 2011 Companion on High Performance Computing Networking, Storage and Analysis Companion*, SC '11 Companion, pages 7–8, New York, NY, USA, 2011. ACM. 2

[10] Yaakoub el Khamra, Shantenu Jha, and Christopher D. White. **Modelling Data-driven CO2 Sequestration Using Distributed HPC Cyberinfrastructure**. In *Proceedings of the 2010 TeraGrid Conference*, TG '10, pages 6:1–6:8, New York, NY, USA, 2010. ACM. 2

[11] G. Sudhakaran, Thomas C. Babu, and V. Ashok. **A GPU Computing Platform (SAGA) and a CFD CODE on GPU for Aerospace Applications**. In *Proceedings of the ATIP/A*CRC Workshop on Accelerator Technologies for High-Performance Computing: Does Asia Lead the Way?*, ATIP '12, pages 3:1–3:5, Singapore, Singapore, 2012. A*STAR Computational Resource Centre. 2, 3

[12] Stephan Nowatschin, Martin Bertram, and Christoph Garth. **GPU-based simulation of cold air flow for environmental planning**. In *Proceedings of the 22nd Spring Conference on Computer Graphics, SCCG 2006, Casta-Papiernicka, Slovakia, April 20-22, 2006*, pages 191–198, 2006. 3

[13] Mark J. Harris, Greg Coombe, Thorsten Scheuermann, and Anselmo Lastra. **Physically-based Visual Simulation on Graphics Hardware**. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '02, pages 109–118, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association. 3

[14] Nolan Goodnight, Cliff Woolley, Gregory Lewin, David Luebke, and Greg Humphreys. **A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware**. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '03, pages 102–111, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. 3

[15] Dennis C. Jespersen. **Acceleration of a CFD Code with a GPU**. *Sci. Program.*, **18**(3-4):193–201, August 2010. 3

[16] Julien C Thibault and Inanc Senocak. **CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompres-**

sible flows. In *Proceedings of the 47th AIAA aerospace sciences meeting*, pages 2009–758, 2009. 3

[17] A.V. Gorobets, F.X. Trias, and A. Oliva. **A parallel MPI + OpenMP + OpenCL algorithm for hybrid supercomputations of incompressible flows**. *Computers & Fluids*, **88**:764 – 772, 2013. 3

[18] Andrey Gorobets, F. Xavier Trias, and Assensi Oliva. **An OpenCL-based Parallel {CFD} Code for Simulations on Hybrid Systems with Massively-parallel Accelerators**. *Procedia Engineering*, **61**:81 – 86, 2013. 25th International Conference on Parallel Computational Fluid Dynamics. 3

[19] S.A. Soukov, A.V. Gorobets, and P.B. Bogdanov. **OpenCL Implementation of Basic Operations for a High-order Finite-volume Polynomial Scheme on Unstructured Hybrid Meshes**. *Procedia Engineering*, **61**:76 – 80, 2013. 25th International Conference on Parallel Computational Fluid Dynamics. 3

[20] S. Han and Hyoung G. Choi. **Investigation of the Parallel Efficiency of a PC Cluster for the Simulation of a CFD Problem**. *Personal Ubiquitous Comput.*, **18**(6):1303–1314, August 2014. 3

[21] Iván Bermejo-Moreno, Julien Bodart, Johan Larsson, Blaise M. Barney, Joseph W. Nichols, and Steve Jones. **Solving the Compressible Navier-stokes Equations on Up to 1.97 Million Cores and 4.1 Trillion Grid Points**. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 62:1–62:10, New York, NY, USA, 2013. ACM. 3

[22] G. Oyarzun, R. Borrell, A. Gorobets, O. Lehmkuhl, and A. Oliva. **Direct Numerical Simulation of Incompressible Flows on Unstructured Meshes Using Hybrid CPU/GPU Supercomputers**. *Procedia Engineering*, **61**:87 – 93, 2013. 25th International Conference on Parallel Computational Fluid Dynamics. 3

[23] J. Appleyard and D. Drikakis. **Higher-order CFD and interface tracking methods on highly-Parallel MPI and GPU systems**. *Computers & Fluids*, **46**(1):101 – 105, 2011. 10th {ICFD} Conference Series on Numerical Methods for Fluid Dynamics (ICFD 2010). 3

[24] Robert Strzodka, Jonathan Cohen, and Stan Posey. **GPU-Accelerated Algebraic Multigrid for Applied CFD**. *Procedia Engineering*, **61**:381 – 387, 2013. 25th International Conference on Parallel Computational Fluid Dynamics. 3

[25] Konstantinos I. Karantasis, Eleftherios D. Polychronopoulos, and John A. Ekaterinaris. **High order accurate simulation of compressible flows on GPU clusters over Software Distributed Shared Memory**. *Computers & Fluids*, **93**:18 – 29, 2014. 3

[26] Chuanfu Xu, Xiaogang Deng, Lilun Zhang, Jianbin Fang, Guangxue Wang, Yi Jiang, Wei Cao, Yonggang Che, Yongxian Wang, Zhenghua Wang, Wei Liu, and Xinghua Cheng. **Collaborating CPU and GPU for large-scale high-order CFD simulations with complex grids on the TianHe-1A supercomputer**. *Journal of Computational Physics*, **278**:275 – 297, 2014. 3

[27] Brent P. Pickering, Charles W. Jackson, Thomas R.W. Scogland, Wu-Chun Feng, and Christopher J. Roy. **Directive-based GPU programming for computational fluid dynamics**. *Computers & Fluids*, **114**(0):242 – 253, 2015. 3

[28] Desconocido. **NVIDIA GeForce GTX 680 The fastest, most efficient GPU ever built.** 3

[29] Desconocido. **Advanced Micro Devices Heterogeneous Computing OpenCL™ and the ATI Radeon™ HD 5870**. 3

[30] Desconocido. **Intel Xeon Phi Product Family**. 3

[31] T. Liu, X.G. Xu, and C.D. Carothers. **Comparison of two accelerators for Monte Carlo radiation transport calculations, Nvidia Tesla {M2090} {GPU} and Intel Xeon Phi 5110p coprocessor: A case study for X-ray {CT} imaging dose calculation**. *Annals of Nuclear Energy*, **82**(0):230 – 239, 2015. Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, {SNA} + {MC} 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms. 4

[32] E.M. Lifshits and L.D. Landau. *Mecánica de fluidos*. Curso de Física Teórica. Reverté, 1985. 7, 9, 42

[33] T. Kambe. *Elementary Fluid Mechanics*. World Scientific, 2007. 8

[34] J.H. Lienhard. *A Heat Transfer Textbook*. Dover Books on Engineering. Dover Publications, 2011. 9, 10

[35] S.V. Patankar. *Numerical Heat Transfer and Fluid Flow*. McGraw-Hill, 1980. 15, 23

[36] M.J.S. de Lemos. *Turbulence in Porous Media: Modeling and Applications*. Elsevier insights. Elsevier Science, 2012. 18

[37] J.V. Doormal and G.D. Raithby. **Enhancements of the simple method for predicting incompressible fluid flow**. *Num, Heat Transfer*, **7**:147–163, 1984. 20, 23

[38] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003. 26, 27

[39] H. A. Van Der Vorst. **Bi-CGStab: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems**. *SIAM Journal on Scientific and Statistical Computing*, **12**:631–644, 1992. 27

[40] T.J. Fountain. *Parallel Computing: Principles and Practice*. Cambridge University Press, 2006. 31, 32

[41] E.W. Dijkstra. **Co-operating Sequential Processes**. *ed. Genuys*, **In Programming Languages**:43–112, 1968. 31

[42] F. Gebali. *Algorithms and Parallel Computing*. Wiley Series on Parallel and Distributed Computing. Wiley, 2011. 32

[43] Blaise Barney. **Introduction to Parallel Computing**. 32

[44] C. Bischof. *Parallel Computing: Architectures, Algorithms, and Applications*. Advances in parallel computing. IOS Press, 2008. 32

[45] M. J. Flynn. **Some computer organizations and effectiveness**. *IEEE Transactions on computers*, **C-21 Issue:9**:948–960, 1966. 33

[46] P.R. PRAKASH and D. SHIKHARE. *INTRODUCTION TO PARALLEL PROCESSING*. PHI Learning, 2006. 35

[47] S. Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Applications of GPU computing series. Morgan Kaufmann, 2013. 36, 37, 38, 39

[48] Desconocido. **CUDA Parallel Computing Platform**. 36, 37

[49] Desconocido. **CUDA Fortran Programing Guide and Reference-Versión 2015**. 37

[50] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Pearson Education, 2010. 37

[51] M.S. Müller, B.R. de Supinski, and B. Chapman. *Evolving OpenMP in an Age of Extreme Parallelism: 5th International Workshop on OpenMP, IWOMP 2009, Dresden, Germany, June 3-5, 2009 Proceedings*. LNCS sublibrary: Programming and software engineering. Springer, 2009. 39

[52] B. Chapman, G. Jost, and R. van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. Number v. 10 in Scientific Computation Series. MIT Press, 2008. 39

[53] Hennessy John L. y David A. Patterson, trad. Juan Manuel Sanchez et. al. *Arquitectura de Computadores, Un enfoque cualitativo*. McGraw-Hill, California, 1993. 41

[54] Luis Miguel de la Cruz Salas. *Cómputo paralelo en la solución numérica de las ecuaciones de balance en flujo turbulento*. PhD thesis, IIMAS-UNAM, 2005. 42, 43, 46