Una tesina presentada para el grado de
Ingeniería Informática

Resolución de las ecuaciones de Navier-Stokes utilizando CUDA

Autor: Santiago D. Costarelli

Director: Dr. Rodrigo R. Paz

Co-Director: Dr. Lisandro D. Dalcin

Junio, 2011

Facultad de Ingeniería y Ciencias Hídricas

Quiero expresar mis agradecimientos a las personas que han contribuido en la formación de la persona que hoy en día soy, en especial a:

- mis padres y mi hermano, por su inmenso apoyo a lo largo de estos años,
- mis abuelos, tíos y primos, por muchas charlas interesantes que me ayudaron a abrir los ojos,
- mis amigos, por estar, aún en momento difíciles,
- Rodrigo Paz, por incontables discusiones, siempre atento y predispuesto a responder mis inquietudes,
- Lisandro Dalcin y Mario Storti, también, apoyándome desde un principio,

a todos ustedes, *muchísimas gracias*.

Hay una fuerza motriz más poderosa que el vapor,
la electricidad y la energía atómica: la voluntad.
Albert Einstein

Resumen

En este trabajo se propone una resolución de las ecuaciones de Navier-Stokes para el caso de fluidos Newtonianos incompresibles utilizando la arquitectura CUDA[1] provista por NVIDIA[2]. Se utiliza para la discretización espacial un esquema de diferencias finitas en grillas *staggered* (para evitar el desacople en la presión), y el método de Pasos Fraccionados (Fractional-Step) para la integración temporal. El paso predictor (problema de advección para las ecuaciones de cantidad de movimiento) se resuelve utilizando el esquema de Adams-Bashforth de segundo orden estabilizando los términos convectivos con el método QUICK. Además, el paso de Poisson (para imponer la incompresibilidad) es resuelto iterativamente mediante el método de Gradientes Conjugados precondicionando al sistema utilizando transformadas rápidas de Fourier.

En presente el desarrollo se utilizan librerías estándar de CUDA para el manejo de matrices y vectores, como ser Thrust[3] y CUSP[4], además de CUFFT[5] para las transformadas rápidas de Fourier. De esta forma, mediante las herramientas aportadas por las anteriores se confeccionan los kernels necesarios enfatizando la utilización de memoria shared, accesos fusionados a la memoria global, reduciendo al mínimo la cantidad de registros por thread, entre otros.

A continuación se presenta una serie de casos de estudio con el objetivo de validar el desarrollo y de, posteriormente, comparar las performances obtenidas con implementaciones en otras arquitecturas (CPU, unicore y multicore).

[1] http://developer.nvidia.com/cuda-downloads
[2] http://arg.nvidia.com/page/home.html
[3] http://code.google.com/p/thrust/
[4] http://code.google.com/p/cusp-library/
[5] http://developer.nvidia.com/cufft

Índice general

Capítulo 1

Ecuaciones de Navier-Stokes

Las ecuaciones de *Navier-Stokes* (N-S) modelan el comportamiento de cualquier tipo de fluido mediante la conservación de 3 cantidades, a saber: masa, cantidad de movimiento y energía. A comienzos de la era de la computación estos modelos no podían ser utilizados en simulaciones dada su alta complejidad computacional, en su lugar se utilizaron modelos de *flujo potencial* y posteriormente *descripciones Eulerianas*. Conforme la tecnología de las computadoras fue evolucionando, estos modelos comenzaron a utilizarse cada vez con más frecuencia hasta resultar natural hoy en día simular complejos sistemas en donde los fluidos son resueltos por una formulación de Navier-Stokes.

Así en este capítulo se introducirán los conceptos básicos que posteriormente se utilizarán para el desarrollo que se propone en este PFC (proyecto final de carrera).

1.1. Introducción

Las ecuaciones de N-S conforman un sistema de ecuaciones diferenciales en derivadas parciales (PDE) no lineales, es por ello que (a excepción de casos muy simples) no se tienen soluciones analíticas conocidas. La necesidad de una resolución numérica resulta evidente, luego grandes esfuerzos se han (aún continúan y continuarán) realizado con el objetivo de mejorar las soluciones obtenidas tanto en tiempo como en precisión.

Si bien el tema central de este PFC se basa en formulaciones para fluidos Newtonianos viscosos incompresibles, se dará una breve introducción al desarrollo y obtención de 2 de las 3 leyes de conservación antes mencionadas (la de energía no será introducida pues no será utilizada). Posteriormente se presentará el problema de incompresibilidad y las ventajas y desventajas que se presentan al tratar con dicho caso.

1.2. Las ecuaciones de conservación

La conservación de cierta magnitud física de un fluido dentro de un determinado volumen de control se define como el efecto neto de fuentes internas en adición a la cantidad de dicha magnitud que atraviesa la superficie, esto último denominado *flujo*, cuya expresión resulta de relaciones entre las propiedades mecánicas y termodinámicas del fluido a tratar. Los flujos asociados a una magnitud escalar son vectores, mientras que aquellos asociados a una vectorial resultan tensores.

Es bien sabido actualmente que la evolución de un fluido esta totalmente descripta por la conservación de tres propiedades, masa, cantidad de movimiento y energía. Además se tiene que otras propiedades como presión, temperatura o entropía (por nombrar algunas) no satisfacen leyes de conservación.

Si se considera entonces un volumen de control arbitrario Ω limitado por una superficie de control S y además se tiene una propiedad escalar por unidad de volumen U, se define la cantidad total de esa propiedad dentro del dominio Ω como

$$\int_{\Omega} U d\Omega, \tag{1.1}$$

luego la variación por unidad de tiempo de la cantidad U en el dominio Ω resulta

$$\frac{\partial}{\partial t}\int_{\Omega} U d\Omega. \tag{1.2}$$

Considere algún tipo de fluido atravesando una determinada superficie de control S, luego el flujo $\mathbf{F}$ como cantidad direccional por unidad de tiempo en un elemento de superficie $\mathbf{dS}$, cuya normal apunta en sentido saliente al volumen, queda definido por

$$\mathbf{F}\cdot\mathbf{dS} = F_n dS, \tag{1.3}$$

donde se ve claramente que si el fluido fluye en dirección paralela a la superficie de control, entonces no se tiene flujo por dicha superficie (el producto escalar resulta nulo). Luego sólo aportan a la ecuación (1.3) aquellos términos en dirección a la normal entrante al cuerpo. El total del flujo superficial entrante esta dado por

$$-\oint_S \mathbf{F}\cdot\mathbf{dS} \tag{1.4}$$

donde el signo negativo esta dado porque *se considera al flujo entrante como positivo*, luego como el $\mathbf{dS}$ apunta hacia afuera, el producto escalar de la ecuación (1.4) resulta negativo. Así es necesario el signo aplicado para concordar con el postulado.

Luego se tienen términos fuentes que de alguna manera reproducen el hecho de ciertas modificaciones sobre la cantidad de materia, existen fuentes internas y externas. Las primeras actúan sólo en la superficie mientras que las segundas actúan volumétricamente por fuerzas como la gravedad, la fuerza de sustentación, el electromagnetismo, entre otros. Así la contribución total por las fuentes esta dado por

$$\int_{\Omega} Q_V d\Omega + \oint_S \mathbf{Q}_S\cdot\mathbf{dS} \tag{1.5}$$

donde Q_V representan las fuentes volumétricas y $\mathbf{Q}_S$ las fuentes superficiales.

Se tiene entonces una ley de conservación general integral para una magnitud conservada U dada por

$$\frac{\partial}{\partial t}\int_{\Omega} U d\Omega = -\oint_S \mathbf{F}\cdot\mathbf{dS} + \int_{\Omega} Q_V d\Omega + \oint_S \mathbf{Q}_S\cdot\mathbf{dS}, \tag{1.6}$$

cuyas principales ventajas son

- que se cumple para cualquier volumen Ω y superficie S,
- que las variaciones de la cantidad U, en auscencia de términos volumétricos, están dadas por las contribuciones de los flujos superficiales y no en flujos volumétricos, y
- que no presenta flujos bajo ningún operador de derivada por lo cual espacialmente puede ser discontinuo, como es el caso de las ondas de choque.

Luego un esquema de discretización es conservativo si mantiene especialmente la segunda condición. Así por ejemplo, si los flujos dentro del dominio no se cancelaran luego no podrían distinguirse de aquellos términos de fuentes volumétricas, como resultado cierta cantidad de masa se estaría creando o destruyendo, corrompiendo el conocido postulado de conservación de masa (que será introducido posteriormente).

Utilizando el teorema de Gauss, que establece la relación entre integrales superficiales y volumétricas, se tiene entonces para un determinado flujo F

$$\oint_S \mathbf{F}\cdot\mathbf{dS} = \int_{\Omega} \nabla\cdot\mathbf{F} d\Omega. \tag{1.7}$$

Introduciendo el resultado de la ecuación (1.7) en la (1.6), y considerando que el volumen es fijo, se obtiene entonces

$$\int_{\Omega} \frac{\partial U}{\partial t} d\Omega + \int_{\Omega} \nabla\cdot\mathbf{F} d\Omega = \int_{\Omega} Q_V d\Omega + \int_{\Omega} \nabla\cdot\mathbf{Q}_S d\Omega, \tag{1.8}$$

luego como la ecuación (1.8) se cumple para cualquier volumen arbitrario Ω, debe ser válido locamente en cualquier punto del dominio del fluido, esto es

$$\frac{\partial U}{\partial t} + \nabla \cdot (\mathbf{F} - \mathbf{Q}_S) = Q_V. \tag{1.9}$$

Entonces se tiene que

- los términos de flujo aparecen bajo el operador gradiente,
- las ecuaciones de conservación siempre presentan la forma de la ecuación (1.9), y
- se requiere de al menos existencia C^1 (es decir, continuidad en la primer derivada) para los flujos.

1.2.1. Convección y difusión

Para cualquier variable conservada U existe un flujo que está conformado por dos contribuciones: la contribución dada por el transporte convectivo y aquella difusiva dada por la agitación molecular.

El total de la cantidad U que esta siendo convectada por el fluido esta definida como

$$\mathbf{F}_C = U\mathbf{v}, \tag{1.10}$$

donde $\mathbf{v}$ es el vector velocidad del fluido. Luego el total de flujo por elemento de superficie es

$$\mathbf{F}_C \cdot \mathbf{dS}. \tag{1.11}$$

Si se define $U = \rho$ con ρ representando la densidad del fluido, luego se define la *razón de flujo másico* como

$$\frac{dm}{dt} = \rho\mathbf{v} \cdot \mathbf{dS}, \tag{1.12}$$

representando la cantidad de masa por unidad de tiempo fluyendo por el elemento de superficie dS, expresado según el sistema internacional por las unidades $[Kg/seg]$.

Con respecto al flujo difusivo $\mathbf{F}_D$ se define básicamente como la contribución presente en fluidos en reposo dado por el efecto macroscópico de la agitación termal molecular. Así cualquier fluido que presente alguna perturbación que modifique su homogeneidad, tratará de llegar a un estado de equilibrio dado a que se generan transferencias en espacio con el objeto de reducir la inhomogeneidad. Esta contribución al flujo total es proporcional al gradiente de de la cantidad correspondiente que necesita desaparecer en distribuciones homogéneas.

Entonces con el objetivo de obtener una expresión para el término difusivo, se presentan a continuación diferentes resultados obtenidos de observaciones experimentales

- el flujo difusivo es proporcional al gradiente de la concentración,
- el gradiente como operador direccional necesita estar negado para apuntar en la dirección de minimización de la función, esto con el objetivo de lograr a la uniformidad, y
- debe ser proporcional a un factor de difusividad que expresa la facilidad de movimiento molecular (propiedad inherente a la propiedad considerada y su ambiente).

Con estas consideraciones se presenta la *ley de Fick*, definida para fluido incompresibles como

$$\mathbf{F}_D = -\kappa\rho\nabla u, \tag{1.13}$$

donde κ representa el factor de difusividad cuyas unidades resultan $[m^2/seg]$, ρ es la densidad y u es una magnitud por unidad de masa.

Considerando ahora $U = \rho u$, luego la ecuación (1.9) puede reescribirse como

$$\frac{\partial \rho u}{\partial t} + \underbrace{\nabla \cdot (\rho\mathbf{v}u)}_{\text{convectivo}} = \underbrace{\nabla \cdot (\kappa\rho\nabla u)}_{\text{difusivo}} + Q_V + \nabla \cdot \mathbf{Q}_S, \tag{1.14}$$

una típica ecuación escalar de convección-difusión. Se observa en la ecuación (1.14) que para coeficientes κ y ρ constante del término difusivo se tiene un Laplaciano de la variable conservada, lo cual resulta en una descripción isotrópica de la difusión del fluido (en el sentido de iguales aportes en las distintas direcciones).

Convección	Difusión
- Expresa el transporte de una dada cantidad por el fluido	- Reproduce el efecto de las colisiones moleculares
- No existe en fluidos en reposo	- Existe en fluidos en reposo
- Todas las cantidades son convectadas por el fluido	- No todas las cantidades están sujetas a difusión
- Comportamiento direccional	- Comportamiento isotrópico
- Modelada por derivadas espaciales de primer orden	- Modelada por derivadas espaciales de segundo orden
- Generalmente no lineal (función de la cantidad convectada)	- Generalmente lineal para propiedades constantes de los fluidos

Cuadro 1.1: Comparación convección vs difusión.

En el Cuadro (1.1) se muestra un resumen de las propiedades que se han estudiado en esta sección.

Cuando la cantidad conservada es un vector $\mathbf{U}$, luego los flujos resultan tensores $\mathbf{F}$ y el término de fuente volumétrica un vector $\mathbf{Q}_V$, así entonces se define la ecuación de conservación vectorial en su forma integral como

$$\int_\Omega \frac{\partial \mathbf{U}}{\partial t} d\Omega + \int_\Omega \nabla \cdot \mathbf{F} d\Omega = \int_\Omega \mathbf{Q}_V d\Omega + \int_\Omega \nabla \cdot \mathbf{Q}_S d\Omega, \tag{1.15}$$

y en su forma diferencial como

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{F} - \mathbf{Q}_S) = \mathbf{Q}_V. \tag{1.16}$$

Luego se definen entonces los flujos convectivos como

$$(\mathbf{F}_C)_{ij} = U_i v_j, \tag{1.17}$$

y los difusivos como

$$(\mathbf{F}_D)_{ij} = -\kappa\rho \frac{\partial u_i}{\partial x_j}. \tag{1.18}$$

Así las ecuaciones (1.15) y (1.16) sólo son equivalentes cuando se asumen que las propiedades del fluido varían de forma continua, de otra forma (como en el caso de ondas de choque invíscidas o discontinuidades de contacto) sólo la primera es válida.

1.3. La ecuación de conservación de la masa

La ley de conservación de masa se cumple con independencia de la naturaleza del fluido o de las fuerzas que actúen sobre el, en otras palabras, es de naturaleza puramente cinemática. Así expresa el hecho empírico de que la materia no desaparece ni puede ser creada. Luego no existe flujo difusivo de transporte de masa total, sino que sólo puede ser transportada por convección.

La cantidad U en este caso es la densidad $U = \rho$, cuyas dimensiones son $[Kg/m^3]$. Entonces la ecuación de conservación tiene la forma

$$\frac{\partial}{\partial t} \int_\Omega \rho d\Omega + \oint_S \rho \mathbf{v} \cdot \mathbf{dS} = 0, \tag{1.19}$$

y en forma diferencial

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0. \tag{1.20}$$

La densidad de un fluido (o de cualquier medio) puede enunciarse como la razón entre su masa y su volumen ocupado, en otras palabras, es una medida que indica que cantidad del volumen ocupa la masa que se tiene.

Las ecuaciones (1.19-1.20) se denominan también ecuaciones de continuidad y están escritas en forma conservativa (de acuerdo a la regla derivada para las ecuaciones conservativas). Existe una versión no conservativa que introduce el concepto de derivada material que no será introducida en este documento pero puede revisarse en textos como [Hir07].

En esencia el problema de los esquemas no conservativos es que si no se toman los recaudos necesarios, las discretizaciones pueden derivar situaciones en donde los flujos internos en el dominio Ω no se cancelen de a pares, lo cual genera un flujo que puede no conservar la masa constante.

1.4. La ecuación de conservación de cantidad de movimiento

La cantidad de movimiento es una cantidad vectorial definida como el producto entre la masa y la velocidad. Así cuando se expresa por unidad de volumen tiene la forma

$$\mathbf{U} = \rho \mathbf{v}, \tag{1.21}$$

entonces la ecuación de conservación en su forma integral se expresa como

$$\frac{\partial}{\partial t}\int_\Omega \rho \mathbf{v} d\Omega + \oint_S (\rho \mathbf{v} \otimes \mathbf{v} - \kappa \rho \nabla \mathbf{v}) \cdot \mathbf{dS} = \int_\Omega \mathbf{Q}_V d\Omega + \oint_S \mathbf{Q}_S \cdot \mathbf{dS}. \tag{1.22}$$

Se tiene que en un fluido en reposo no existe difusión de cantidad de movimiento y por lo tanto no existe contribución al tensor de flujos $\mathbf{F}$.

La ecuación de conservación de cantidad de movimiento se deriva de la segunda ley de Newton que relaciona fuerzas, masas y aceleraciones. Así establece que un cambio en la cantidad de movimiento de un sistema esta sujeto a dos tipos de fuerzas aplicadas sobre el: las externas $\mathbf{f}_e$ y las internas $\mathbf{f}_i$, definidas por unidad de masa. Luego el término fuente $\mathbf{Q}_V$ consiste en la suma de esas dos fuerzas, a saber $\rho \mathbf{f}_e$ y $\rho \mathbf{f}_i$ la primera actuando sobre el volumen y la segunda sólo en la superficie (dado por el par acción-reacción).

Considerando a $\boldsymbol{\sigma}$ como el tensor de tensiones del fluido luego se tiene que las fuerzas externas actuando sobre un elemento de superficie $\mathbf{dS}$ están definidas como

$$\mathbf{f}_i = \boldsymbol{\sigma} \cdot \mathbf{n}, \tag{1.23}$$

donde $\mathbf{n}$ representa la normal al elemento de superficie $\mathbf{dS}$. El tensor de tensiones puede definirse en base a dos componentes, uno isotrópico y uno desviatorio, esto es

$$\boldsymbol{\sigma} = -p\mathbf{I} + \boldsymbol{\tau}, \tag{1.24}$$

donde p es la presión, término definido en pocas palabras como la fuerza en dirección a la normal de la superficie donde se calcula, $\mathbf{I}$ es el tensor unitario y $\boldsymbol{\tau}$ es el tensor de tensiones de corte viscoso definido para el caso de fluidos Newtonianos como

$$\tau_{ij} = \mu \left[\left(\frac{\partial v_j}{\partial x_i} + \frac{\partial v_i}{\partial x_j} \right) - \frac{2}{3}(\nabla \cdot \mathbf{v})\delta_{ij} \right], \tag{1.25}$$

donde μ es la viscosidad cinemática. La viscosidad puede enunciarse como la resistencia de un fluido a fluir, que esencialmente esta dada por el rozamiento de las moléculas del fluido generando colisiones y rozamientos. La viscosidad dinámica μ mide la resistencia de un fluido a fluir por un tubo capilar, lo mismo para la viscosidad cinemática ν, sin embargo su relación es $\nu = \frac{\mu}{\rho}$.

Una definición básica del tensor de corte viscoso es la que enuncia que representa las fuerzas de fricciones internas provenientes de las capas del fluido al rozar entre ellas.

Luego se tiene una ecuación para la conservación de la cantidad de movimiento en fluidos Newtonianos como

$$\begin{aligned} \frac{\partial}{\partial t}\int_\Omega \rho \mathbf{v} d\Omega + \oint_S \rho \mathbf{v}(\mathbf{v} \cdot \mathbf{dS}) &= \int_\Omega \rho \mathbf{f}_e d\Omega + \oint_S \sigma \cdot \mathbf{dS}, \\ &= \int_\Omega \rho \mathbf{f}_e d\Omega - \oint_S p \cdot \mathbf{dS} + \oint_S \boldsymbol{\tau} \cdot \mathbf{dS}. \end{aligned} \tag{1.26}$$

Aplicando el teorema de Gauss se obtiene la forma integral de conservación

$$\frac{\partial}{\partial t}\int_\Omega \rho\mathbf{v}d\Omega + \int_\Omega \nabla\cdot(\rho\mathbf{v}\otimes\mathbf{v}) = \int_\Omega \rho\mathbf{f}_e d\Omega + \int_\Omega \nabla\cdot\boldsymbol{\sigma} d\Omega, \tag{1.27}$$

y luego se obtiene aquella en forma diferencial

$$\frac{\partial\rho\mathbf{v}}{\partial t} + \nabla\cdot(\rho\mathbf{v}\otimes\mathbf{v} + p\mathbf{I} - \boldsymbol{\tau}) = \rho\mathbf{f}_e. \tag{1.28}$$

Finalmente si se aplicara la ecuación (1.25) en la ecuación (1.28), se obtendrían las ecuaciones de movimiento de Navier-Stokes. Además si se considerara un fluido sin tensiones internas de corte, las ecuaciones de cantidad de movimiento se reducen a las ecuaciones de movimiento de Euler.

1.5. El modelo incompresible

Las ecuaciones que gobiernan a los fluidos incompresibles, resultado de considerar la densidad ρ como constante, pueden ser escritas en forma conservativa como

$$\begin{aligned}\frac{\partial\mathbf{u}}{\partial t} + \nabla\cdot(\mathbf{u}\otimes\mathbf{u}) &= -\frac{1}{\rho}\nabla p + \nu\Delta\mathbf{u} + \rho\mathbf{f}_e \\ \nabla\cdot\mathbf{u} &= 0\end{aligned} \tag{1.29}$$

teniendo en cuenta que $\mathbf{u}$ es el campo velocidad (en 3D presenta 3 componentes), ρ es la densidad, p es la presión, ν es la viscosidad cinemática y $\mathbf{f}_e$ representa a las fuerzas ejercidas por el exterior sobre el fluido. Se tiene entonces que los campos incógnita son dos, a saber: presión y velocidad.

El sistema (1.29) presenta un desacople de las ecuaciones de continuidad y cantidad de movimiento sobre la de energía (esta última no introducida pues no es utilizado en lo que sigue de este PFC), esto si el flujo permanece isotérmico lo que en general se cumple cuando no se tiene transferencia de calor. Además se observa la ausencia de un término temporal sobre la presión dado al hecho de que la ecuación de continuidad no tiene un carácter transiente, lo que genera ciertas dificultades al momento de la resolución numérica.

Una ecuación para la presión puede ser obtenida tomando la divergencia en las ecuaciones de cantidad de movimiento e introduciendo la condición de incompresibilidad, de esta forma se obtiene

$$\frac{1}{\rho}\Delta p = -\nabla\cdot(\mathbf{u}\otimes\mathbf{u}) + \nabla\cdot\mathbf{f}_e. \tag{1.30}$$

Capítulo 2

Discretización espacial de las ecuaciones de N-S

Para tratar con un sistema de ecuaciones como el de Navier-Stokes que se encuentra expresado en el campo del continuo, es que se utilizan técnicas de discretización espacio-temporal para finalmente obtener el modelo que puede ser tratado por una computadora. Es de importancia notar que este tipo de simplificaciones (discretización) producen errores de truncamiento, por ende deben ser analizados con cautela.

En esta sección se introducirán los conceptos que servirán para llevar una ecuación diferencial parcial (PDE), a una ecuación diferencial ordinaria (ODE), esto es pues de una ecuación con términos espacio-temporales continuos se obtendrá una ecuación discreta en espacio pero continua en tiempo.

2.1. Introducción

Existen varias técnicas de discretización de ecuaciones diferenciales, tanto ordinarias como parciales. En este PFC se expondrá el método de diferencias finitas (FDM), analizando el error de los operadores discretizados e introduciendo el concepto de discretización conservativa.

2.2. Grillas estructuradas y no estructuradas

Las grillas de resolución se definen como el conjunto discreto de puntos pertenecientes al dominio continuo sobre el cual se obtienen las soluciones. A grandes rasgos existen dos tipos de grillas, las estructuradas y las no estructuradas. En las primeras si se supone un espacio n-dimensional, luego se tienen n familias ortogonales de curvas que se intersectan en los denominados nodos. Así dado un nodo P, es trivial obtener algún vecino pues sólo se necesita restar en una unidad alguna de las coordenadas del nodo P, supóngase el caso 2D, si se tiene un nodo en las coordenadas (i, j) luego sus posibles vecinos más cercanos son 4, dígase $(i-1, j), (i+1, j), (i, j-1)$ y $(i, j+1)$. Como contrapartida si se necesitara de mayor densidad de puntos en una determinada zona, debería de introducirse más curvas ortonales a la familia original que incrementa la cantidad de puntos en partes del dominio que no lo requieran (es decir, no existe localidad en la adición de curvas cuando se quiere refinar cierta sección del dominio).

En las segundas, grillas no estructuradas, uno no puede conocer a priori los vecinos de un determinado nodo. Contrario al caso estructurado, aquí se tienen divisiones del dominio que presentan como elemento básico un poliedro. Luego no es posible determinar un nodo vecino dado un nodo particular con coordenadas como en el caso estructurado. Como ventaja las zonas que requieran de mayor resolución de nodos pueden obtenerse sin incrementar la densidad de nodos en lugares no deseados.

2.3. El método de las diferencias finitas unidimensional

El método de las diferencias finitas esta basado en la aproximación de Taylor de una función. Además se tiene que sólo puede ser utilizado para grillas estructuradas. Considerando el caso 1D, sea la función $u(x)$ derivable en el punto $x \in \mathbb{R}$ se tiene entonces que la expansión de Taylor a la función $u(x)$ tiene la forma

$$u(x+\Delta x) = u(x) + \Delta x \frac{du}{dx} + \frac{\Delta x^2}{2!}\frac{d^2u}{dx^2} + \frac{\Delta x^3}{3!}\frac{d^3u}{dx^3} + \cdots, \tag{2.1}$$

$\forall \Delta x \in \mathbb{R}$ y el operador $\frac{d}{dx}$ indicando diferencial total. Esta ecuación indica que uno puede conocer el valor exacto de una función definida en x desplazada en Δx conociendo el valor de la función en dicho punto además de sus infinitas primeras derivadas. Así esta claro que todo intento de reducir la cantidad de términos para la aproximación resultará en un error de truncamiento del orden de coeficiente de la derivada de mayor orden que se deprecie.

Si se supone que se desea conocer una aproximación al operador derivada primera, entonces uno podría simplemente hacer pasaje de términos en la ecuación (2.1) para obtener

$$\frac{du}{dx} = \frac{u(x+\Delta x) - u(x)}{\Delta x} \underbrace{- \frac{\Delta x}{2}\frac{d^2u}{dx^2} - \frac{\Delta x^2}{6}\frac{d^3u}{dx^3} - \cdots}_{\text{error de truncamiento}}, \tag{2.2}$$

en donde puede considerarse que mientras $\Delta x \to 0$ luego $\Delta x \gg \Delta x^2 \gg \cdots$. Así el término englobado por la etiqueta *error de truncado* esta superiormente acotado por Δx, y puede expresarse como $\mathcal{O}(\Delta x)$. Entonces

$$\frac{du}{dx} = \frac{u(x+\Delta x) - u(x)}{\Delta x} + \mathcal{O}(\Delta x). \tag{2.3}$$

Considerando la grilla estructurada y homogénea (en el sentido de que $\Delta x_0 = x_1 - x_0 = \Delta x_1 = x_2 - x_1 = \cdots = \Delta x_{n-1} = x_n - x_{n-1} = \Delta x$), y a u_i como $u_i = u(x_0 + i\Delta x)$, luego se define la *diferencia hacia adelante de primer orden* como

$$\left(\frac{du}{dx}\right)_i = \frac{u_{i+1} - u_i}{\Delta x} + \mathcal{O}(\Delta x), \tag{2.4}$$

mientras que la *diferencia hacia atrás de primer orden* esta definida como

$$\left(\frac{du}{dx}\right)_i = \frac{u_i - u_{i-1}}{\Delta x} + \mathcal{O}(\Delta x). \tag{2.5}$$

Es de interés notar que ambas aproximaciones son de primer orden espacial. Una aproximación de mayor orden puede obtenerse combinando las ecuaciones (2.4) y (2.5), si se suman ambas y se realizan algunos pasajes de términos, obteniendo

$$\left(\frac{du}{dx}\right)_i = \frac{u_{i+1} - u_{i-1}}{2\Delta x} + \mathcal{O}(\Delta x^2), \tag{2.6}$$

la denominada *fórmula de diferencias centradas*. La mejora de las diferencias centradas viene de la mano de que aproxima a la derivada en el nodo i, con una recta secante que pasa por $i+1$ e $i-1$, mientras que para el caso de diferencia hacia adelante se aproxima con una recta secante entre el nodo $i+1$ y el i, y para el caso de diferencia hacia atrás por un recta secante entre el nodo i e $i-1$.

Esta claro entonces que si se tiene un error de truncamiento $\mathcal{O}(\Delta x^2)$, luego la aproximación será exacta para funciones lineales. Lo mismo ocurre para truncamientos del orden de la derivada tercera, que será exacta para funciones cuadráticas. Y así sucesivamente.

Un tema que merece ser introducido es que dada una aproximación entre dos puntos, si la misma se aplica a un punto intermedio luego el orden de la misma será de un orden mayor. Por ejemplo, si se tiene la diferencia hacia adelante definida en el nodo i, luego se tiene que

$$\left(\frac{du}{dx}\right)_{i+\frac{1}{2}} = \frac{u_{i+1} - u_i}{\Delta x} + \mathcal{O}(\Delta x^2), \tag{2.7}$$

así se ha ganado un orden de precisión tomando los valores intermedios.

Finalmente el orden del sistema algebraico a resolver es del orden del término con aproximación más gruesa, es decir, el orden global es aquel correspondiente al orden del término del continuo aproximado con mayor error de truncamiento.

Otro punto a tener en cuenta es la condición de consistencia de las fórmulas de diferencias finitas, que establece que la suma de los coeficientes de los nodos en una fórmula de diferencias finitas debe ser cero, asegurando que la aproximación numérica a la derivada en una zona constante sea nula.

Una aproximación de segundo orden a la derivada segunda de u esta definida por

$$\left(\frac{d^2u}{dx^2}\right)_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + \mathcal{O}(\Delta x^2), \tag{2.8}$$

en donde si se tiene en cuenta el carácter isotrópico de la difusión, luego este operador simétrico resulta una buena aproximación (tanto física como matemáticamente).

2.4. El método de las diferencias finitas, extensión multidimensional

El método de las diferencias finitas puede ser extendido fácilmente desde el caso 1D a más dimensiones. Sin pérdida de generalidad se puede decir que consiste en aplicar la discretización en la dirección que indica el operador diferencial, así por ejemplo, se puede discretizar un operador Laplaciano de la forma

$$(\Delta u)_{ij} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}, \tag{2.9}$$

conservando el segundo orden para grillas uniformes, y no uniformes siempre y cuando la distancia entre nodos no cambie significativamente.

De esta forma uno podría utilizar discretizaciones de diferentes ordenes para determinadas direcciones enfatizando aquella que lo requiera.

2.5. El problema de las grillas no uniformes

En general cuando se tienen cuerpos embebidos dentro del dominio computacional que se esta resolviendo uno necesita adaptar la grilla para que esta pueda representar la realidad física acometida en la proximidad de la interfaz. Habitualmente se utilizan conceptos como refinamiento de grilla (incrementando la cantidad de nodos involucrados en la capa límite, o cerca del cuerpo) para evitar el desborde de los gradientes de alguna cantidad en dicha zona. Como el caso de estudio actual trata sobre grillas estructuradas, una mejor resolución (de nodos) espacial requiere de más divisiones (lineas) en una dirección dada, independientemente de los problemas que esto trae (analizado en la sección (2.2)) presenta aún otro problema interesante.

Cuando se ha expandido la serie de Taylor (ecuación (2.1)) se ha considerado una distancia entre nodos Δx uniforme, es decir, la misma entre todos los nodos. Esto en general puede no ser así, indicando que ciertas zonas podrían tener una mayor densidad de nodos que otras. Luego el error de truncamiento del operador del continuo discretizado debe ser reevaluado, pues si las diferencias entre los Δx_i es muy grande entonces el orden de los operadores puede reducirse. Por ejemplo, si se considera el caso de aproximación a una derivada primera por diferencias hacia adelante y hacia atrás, esto es

$$\begin{aligned}\left(\frac{du}{dx}\right)_i =& \frac{u_{i+1} - u_i}{\Delta x_i} - \frac{\Delta x_i}{2}\left(\frac{d^2u}{dx^2}\right)_i - \frac{\Delta x_i^2}{6}\left(\frac{d^3u}{dx^3}\right)_i \\ =& \frac{u_i - u_{i-1}}{\Delta x_{i-1}} - \frac{\Delta x_{i-1}}{2}\left(\frac{d^2u}{dx^2}\right)_i - \frac{\Delta x_{i-1}^2}{6}\left(\frac{d^3u}{dx^3}\right)_i,\end{aligned} \tag{2.10}$$

luego sumando ambas aproximaciones y despejando el término de derivada primera se obtiene

$$\left(\frac{du}{dx}\right)_i = \frac{1}{2}\left[\frac{u_{i+1}-u_i}{\Delta x_i} + \frac{u_i - u_{i-1}}{\Delta x_{i-1}}\right] - \underbrace{\frac{\Delta x_i - \Delta x_{i-1}}{4}}_{*}\left(\frac{d^2u}{dx^2}\right)_i - \frac{\Delta x_i^2 + \Delta x_{i-1}^2}{12}\left(\frac{d^3u}{dx^3}\right)_i, \tag{2.11}$$

observar que en grillas uniforme la ecuación (2.11) se reduce a la (2.6) mientras que en grillas no uniformes si $\Delta x_i - \Delta x_{i-1}$ no esta próximo a cero, luego el término englobado por (*) no puede ser depreciado y el el operador discretizado (que antes era de segundo orden) ahora resulta de primer orden.

2.6. Fórmulas conservativas

La conservación al nivel de discretización es de vital importancia en los desarrollos numéricos pues impone una condición necesaria para obtener resultados congruentes con la situación física simulada. Sin entrar en detalles en el Capítulo (1) se han introducido las ecuaciones de conservación, especialmente la de la masa, y además se ha presentado una fórmula general de conservación. Recordando se tenían términos fuentes volumétricos y superficiales, y términos de flujo, estos dos últimos actuando únicamente superficialmente. Entonces necesariamente dentro del dominio estos términos deberían de ser cero (dado en principio por pares acción-reacción), así se define que una ecuación es conservativa cuando la ecuación para un celda esta definida como una combinación de las variables definidas en sus celdas adyacentes. De esta forma, cuando se ensamblen las ecuaciones para los nodos interiores, sólo aquellas variables en el extremo permanecerán en las ecuaciones, mientras que las internas desaparecerán en el acople. Finalmente, cuando se haya ensamblado todo el dominio, sólo los términos extremos permanecerán, cumpliendo la condición impuesta por la ecuación de conservación general.

Como ejemplo se puede presentar el caso de la derivada primera unidimensional de la cantidad u, un esquema de segundo orden tiene la forma

$$\left(\frac{du}{dx}\right)_i = \frac{1}{2}\frac{u_{i+\frac{1}{2}} - u_{i-\frac{1}{2}}}{x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}}, \tag{2.12}$$

si para simplificar se consideran grillas uniformes y si se realizara la sumatoria de todas las aproximaciones en cada nodo se obtendría

$$\frac{du}{dx} = \sum_{i=0}^{n-1}\left(\frac{du}{dx}\right)_i = \frac{u_{n-\frac{1}{2}} - u_{-\frac{1}{2}}}{\Delta x}, \tag{2.13}$$

considerando a n como el total de nodos en una distribución unidimensional.

2.7. El teorema de Lax para diferencias finitas

Sea $u_{ex,i}$ el valor nodal de la solución exacta $u(x_i)$, y además se considera un operador Laplaciano de 3 puntos con un error de truncamiento $\mathcal{O}(\Delta x^2)$ obtenido expandiendo Taylor por derecha y por izquierda de la componente i-ésima, esto es,

$$\begin{aligned} u_{i+1} &= u_i + \Delta x\frac{du_i}{dx} + \frac{\Delta x^2}{2}\frac{d^2u_i}{dx^2} + \frac{\Delta x^3}{6}\frac{d^3u_i}{dx^3} + \frac{\Delta x^4}{24}\frac{d^4u_{i_1}}{dx^4} \\ u_{i-1} &= u_i - \Delta x\frac{du_i}{dx} + \frac{\Delta x^2}{2}\frac{d^2u_i}{dx^2} - \frac{\Delta x^3}{6}\frac{d^3u_i}{dx^3} + \frac{\Delta x^4}{24}\frac{d^4u_{i_2}}{dx^4} \\ \rightarrow \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} &= \frac{d^2u_i}{dx^2} + \frac{\Delta x^2}{12}\left[\frac{d^4u_{i_1}}{dx^4} + \frac{d^4u_{i_2}}{dx^4}\right] = \frac{d^2u_i}{dx^2} + \frac{\Delta x^2}{12}\frac{d^4u_{i_3}}{dx^4}, \end{aligned} \tag{2.14}$$

en donde $u_{i_1} \in [(i-1)\Delta x; i\Delta x]$, $u_{i_2} \in [i\Delta x; (i+1)\Delta x]$ y $u_{i_3} \in [(i-1)\Delta x; (i+1)\Delta x]$. Luego se tiene que el error de truncamiento e esta acotado, esto es

$$|e_i| \leq \frac{\Delta x^2}{12} \max_{[x_{i-1}, x_{i+1}]} \frac{d^4 u_{i_3}}{dx^4} \leq C\Delta x^2. \tag{2.15}$$

En notación matricial se tiene

$$\mathbf{K}\mathbf{u}_{ex} = \mathbf{f} + \mathbf{e}, \tag{2.16}$$

y considerando que, además del error de truncamiento en el operador diferencial existe otro error en la evaluación de la función en los nodos. Denotando E a estos valores, cuya componente j-ésima se relaciona al nodo j, luego se tiene que

$$\begin{aligned} \mathbf{K}\mathbf{u} &= \mathbf{f}, \\ \mathbf{K}\mathbf{u}_{ex} &= \mathbf{f} + \mathbf{e}, \\ \mathbf{K}\mathbf{E} &= -\mathbf{e}, \\ \rightarrow \mathbf{E} &= -\mathbf{K}^{-1}\mathbf{e}, \end{aligned} \tag{2.17}$$

donde $\mathbf{E} = \mathbf{u} - \mathbf{u}_{ex}$. Al hecho de que el error de truncamiento $\mathbf{e} \rightarrow 0$ conforme $\Delta x \rightarrow 0$ se denomina *consistencia* del método, indica básicamente que incluyendo más términos de la expansión de Taylor a la aproximación del diferencial continuo el error de truncamiento se reducirá, y en el límite con la inclusión de infinitos términos de Taylor el error de truncamiento se hace cero.

Se necesita conocer además bajo que condiciones $\mathbf{E} \rightarrow 0$. De la ecuación (2.17) se observa que este hecho debe estar relacionado de alguna manera a $\mathbf{K}$, es decir, que conforme $\Delta x \rightarrow 0$, $\mathbf{K}^{-1}$ se mantenga acotada.

Tomando normas en la ecuación (2.17) (y considerando la desigualdad triangular) se obtiene

$$||\mathbf{E}|| \leq ||\mathbf{K}^{-1}||\, ||\mathbf{e}||, \tag{2.18}$$

considerando sólo normas-p tanto para vectores como para matrices y recordando que la norma-p con $p = 2$ sobre matrices denominada (norma espectral) es igual al mayor valor propio de la matriz sobre la que se calcula, luego en la inversa de una matriz el mayor autovalor será $1/\lambda_{min}$, con lo cual

$$\begin{aligned} \sqrt{\sum_{\forall i}(u_i - u_{ex,i})^2} &\leq \frac{1}{\lambda_{min}} \sqrt{\sum_{\forall i} e_i^2} \\ &\leq \frac{1}{\lambda_{min}} \sqrt{\sum_{\forall i} L\Delta x^2} \\ &\leq \frac{1}{\lambda_{min}} \sqrt{L-1} C\Delta x^2, \end{aligned} \tag{2.19}$$

es decir que

$$\sqrt{\frac{1}{L-1}\sum_{i=1}^{L-1}(u_i - u_{ex,i})^2} \leq \frac{1}{\lambda_{min}} C\Delta x^2, \tag{2.20}$$

en donde el LHS (left hand side, o miembro izquierdo) de la ecuación (2.20) representa al error cuadrático medio de la solución numérica, y es del mismo orden que el error de truncamiento siempre y cuando λ_{min} se mantenga acotado cuando $\Delta x \rightarrow 0$. Si se cumple esta condición se dice que el esquema es estable.

Finalmente un esquema se considera *convergente* cuando garantiza tanto la condición de consistencia como la de estabilidad, y este es el teorema de Lax, base del análisis de error para el método de diferencias finitas.

Capítulo 3

Discretización temporal de las ecuaciones de N-S

Luego de la discretización espacial el sistema resultante es una ODE, en este sólo los términos temporales aparecen descriptos por operadores continuos. Es por esto que en este capítulo se introducirán los conceptos necesarios para la discretización temporal, requerida para una completa formulación discreta.

3.1. Introducción

Considerando una ODE de la forma

$$\mathbf{C}\frac{d\mathbf{u}}{dt} + \mathbf{K}\mathbf{u} = \mathbf{f}, \tag{3.1}$$

donde $\mathbf{C}$ y $\mathbf{K}$ son las matrices con los coeficientes de los nodos incógnitas $\mathbf{u}$ en dependencia temporal y espacial respectivamente y $\mathbf{f}$ un término forzante que no es función de $\mathbf{u}$; resulta claro que el operador diferencial temporal actuando sobre ϕ necesita ser discretizado para finalmente obtener una sistema de ecuaciones procesable por una computadora.

Sin entrar en detalles dos familias de métodos se utilizan para discretizar el término temporal, los métodos de un paso y aquellos multipasos. Los primeros realizan una aproximación a la derivada primera utilizando información de los nodos en una iteración previa, esto es, un paso atrás, de allí el nombre de métodos de un paso. Entre ellos podemos nombrar los métodos de Euler y de Runge-Kutta. Con respecto a los segundos, utilizan la información de más de un paso previo para realizar la aproximación, de esta forma se tiene historia del término y en general resulta en un aproximación de mayor precisión. Por nombrar un ejemplo se tienen los métodos de Adams-Bashforth y Adams-Moulton.

La ventaja de los métodos de un paso en comparación a aquellos multipasos es que requieren de menor memoria y tiempo de cálculo, puesto que toman decisiones (o en otras palabras, realizan la aproximación) con menor cantidad de información que aquellos multipasos. Como desventaja estos últimos presentan un error menor a los primeros.

3.2. Discretizaciones temporales básicas

Considerando n y $n+1$ dos valores temporales consecutivos cuyo espaciamiento temporal esta dado por Δt y además un escalar $\theta \in \mathbb{R}[0;1]$, luego uno podría realizar las siguientes expansiones de Taylor considerando un $\Delta t \in \mathbb{R}$

$$\mathbf{u}^n = \mathbf{u}^{n+\theta} - \theta\Delta t \left.\frac{d\mathbf{u}}{dt}\right|^{n+\theta} + \frac{(\theta\Delta t)^2}{2} \left.\frac{d^2\mathbf{u}}{dt^2}\right|^{n+\theta} - \frac{(\theta\Delta t)^3}{6} \left.\frac{d^3\mathbf{u}}{dt^3}\right|^{n+\theta_1}, \tag{3.2}$$

y

$$\mathbf{u}^{n+1} = \mathbf{u}^{n+\theta} + (1-\theta)\Delta t \left.\frac{d\mathbf{u}}{dt}\right|^{n+\theta} + \frac{[(1-\theta)\Delta t]^2}{2} \left.\frac{d^2\mathbf{u}}{dt^2}\right|^{n+\theta} + \frac{[(1-\theta)\Delta t]^3}{6} \left.\frac{d^3\mathbf{u}}{dt^3}\right|^{n+\theta_2}, \tag{3.3}$$

donde $\theta_1 \in [n; n+\theta\Delta t]$ y $\theta_2 \in [n+\theta\Delta t; n+1]$ son dos valores utilizados por las expansiones de Taylor y tienen un carácter de representación de los restantes términos de mayor orden truncados. Si uno restara las ecuaciones (3.3) y (3.2) obtendría una aproximación a la derivada primera centrada en el tiempo $n+\theta$

$$\begin{aligned} \left.\frac{d\mathbf{u}}{dt}\right|^{n+\theta} = \frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t} + \frac{\Delta t}{2}[(1-\theta)^2-\theta^2] \left.\frac{d^2\mathbf{u}}{dt^2}\right|^{n+\theta} \\ + \frac{\Delta t_n^2}{6}\left[(1-\theta)^3 \left.\frac{d^3\mathbf{u}}{dt^3}\right|^{n+\theta_1} - \theta^3 \left.\frac{d^3\mathbf{u}}{dt^3}\right|^{n+\theta_2}\right]. \end{aligned} \tag{3.4}$$

En la ecuación (3.4) se observa como con un valor de $\theta = 1/2$ el error de truncamiento es $\mathcal{O}(\Delta t^2)$, mientras que si $\theta = 0,1$ luego el error de truncamiento es $\mathcal{O}(\Delta t)$.

Considerando aproximaciones a $\mathbf{u}^{n+\theta}$ como

$$\mathbf{u}^{n+\theta} = (1-\theta)\mathbf{u}^n + \theta\mathbf{u}, \tag{3.5}$$

y similar para el caso de $\mathbf{f}^{n+\theta}$, se obtiene un esquema discreto genérico que presenta la forma

$$\left\{\frac{\mathbf{C}}{\Delta t} + \theta\mathbf{K}\right\}\mathbf{u}^{n+1} + \left\{-\frac{\mathbf{C}}{\Delta t} + (1-\theta)\mathbf{K}\right\}\mathbf{u}^n = (1-\theta)\mathbf{f}^n + \theta\mathbf{f}^{n+1}. \tag{3.6}$$

Finalmente de esta última expresión se obtienen los siguientes esquemas de discretización temporal básicos:

1- Euler hacia adelante (o Forward Euler -FE-): consiste en utilizar $\theta = 0$ con lo cual la ecuación (3.6) resulta

$$\left\{\frac{\mathbf{C}}{\Delta t}\right\}\mathbf{u}^{n+1} + \left\{-\frac{\mathbf{C}}{\Delta t} + \mathbf{K}\right\}\mathbf{u}^n = \mathbf{f}^n, \tag{3.7}$$

de esta forma el sistema resultante presenta un error de truncamiento $\mathcal{O}(\Delta t)$, y además es explícito, en el sentido de que la matriz LHS resultante es diagonal con lo cual la inversión para resolver el sistema es trivial.

2- Euler hacia atrás (o Backward Euler): consiste en utilizar $\theta = 1$ con lo cual la ecuación (3.6) resulta

$$\left\{\frac{\mathbf{C}}{\Delta t} + \mathbf{K}\right\}\mathbf{u}^{n+1} + \left\{-\frac{\mathbf{C}}{\Delta t}\right\}\mathbf{u}^n = \mathbf{f}^{n+1}, \tag{3.8}$$

luego el sistema resultante ya no es diagonal (pues términos de $\mathbf{K}$ introducen estructuras no diagonales en la matriz LHS final) y surge la necesidad de invertir el sistema para obtener $\mathbf{u}^{n+1}$. El error de truncamiento es $\mathcal{O}(\Delta t)$. A estos métodos se los denomina implícitos.

3- Cranck-Nicholson: consiste en utilizar $\theta = 1/2$ con lo cual la ecuación (3.6) resulta

$$\left\{\frac{\mathbf{C}}{\Delta t} + \frac{1}{2}\mathbf{K}\right\}\mathbf{u}^{n+1} + \left\{-\frac{\mathbf{C}}{\Delta t} + \frac{1}{2}\mathbf{K}\right\}\mathbf{u}^n = \frac{1}{2}\left(\mathbf{f}^n + \mathbf{f}^{n+1}\right), \tag{3.9}$$

otro método implícito que, a diferencia de Euler hacia atrás, presenta un error de truncamiento $\mathcal{O}(\Delta t_n^2)$.

3.3. La condición CFL

En las discretizaciones explícitas la variable actual (LHS) por calcular depende sólo de los valores previos de esa variable (RHS, o miembro derecho), de esta manera el sistema es diagonal y las soluciones se obtienen, trivialmente, resolviendo la inversa de una matriz diagonal. Como ventajas las soluciones se obtienen con pocas cuentas aritméticas, como desventaja el paso de tiempo (Δt) se ve altamente afectado por este esquema o dicho de otra manera, debe satisfacer la condición CFL. La condición CFL es una relación entre la velocidad actual del fluido u, el paso de tiempo Δt y el tamaño de elemento Δx; en 1D queda definido como

$$Courant = \frac{u\Delta t}{\Delta x} \leq 1, \tag{3.10}$$

lo que en problemas estacionarios puede resultar desventajoso, pues se requerirá de más pasos de tiempo para lograr el estado estacionario.

En las implícitas la variable en un tiempo dado depende de más variables en el mismo tiempo. Como desventaja la matriz resultante ya no es diagonal y su inversa no es trivial. En general las estructura que presenta es banda, simétrica, entre otras propiedades, y se conocen algoritmos muy eficientes para resolverla. Como ventaja no es necesario que cumpla con la condición CFL, por lo que los pasos de tiempos pueden ser mayor y por ende lograr el estado estacionario en menos pasos.

3.4. El método de Adams-Bashforth

Como se ha mencionado anteriormente la ventaja de los métodos multipasos radica en la utilización de más de un valor solución precedente y de sus derivadas. De esta forma la aproximación tiene memoria lo que conlleva en una mejora notable en términos de exactitud.

Suponiendo una PDE de la forma

$$\frac{d\mathbf{u}}{dt} = f(t, \mathbf{x}) = f(t, \mathbf{u}(t)), \tag{3.11}$$

donde $t \in [a; b]$ y la solución inicial es $\mathbf{u}(a) = \boldsymbol{\alpha}$, un método multipasos de m pasos queda definido por

$$\begin{aligned} \mathbf{u}^{n+1} =& a_{m-1}\mathbf{u}^n + a_{m-2}\mathbf{u}_{n-1} + \cdots + a_0\mathbf{u}_{n+1-m} \\ &+ \Delta t \left[b_m f(t_{n+1}, \mathbf{u}^{n+1}) + b_{m-1} f(t_n, \mathbf{u}^n) + \cdots + b_0 f(t_{n+1-m}, \mathbf{u}^{n+1-m}) \right], \end{aligned} \tag{3.12}$$

en donde se utilizó $\mathbf{u}^n = \mathbf{u}(t_n)$, Δt el paso de tiempo (uniforme) y $a_k, b_k \ \forall k = 0, 1, ..., m$ coeficientes constantes. Además se definen sus condiciones iniciales (m en total) como

$$\mathbf{u}^0 = \boldsymbol{\alpha}_0, \ \ \mathbf{u}^1 = \boldsymbol{\alpha}_1, \ \ \cdots \ \ , \mathbf{u}^{m-1} = \boldsymbol{\alpha}_{m-1}. \tag{3.13}$$

Así el método puede ser explícito cuando $b_m = 0$, o implícito de otra forma.

Observar que la ecuación (3.11) puede resolverse en los instantes de tiempo t_n y t_{n+1} como

$$\mathbf{u}^{n+1} - \mathbf{u}^n = \int_{t_n}^{t_{n+1}} f(t, \mathbf{u}(t))dt \quad \rightarrow \quad \mathbf{u}^{n+1} = \mathbf{u}^n + \int_{t_n}^{t_{n+1}} f(t, \mathbf{u}(t))dt. \tag{3.14}$$

Luego como la función $f(t, \mathbf{u}(t))$ es la incógnita uno podría aproximar el término del integrando por un polinomio de grado $m-1$ determinado con los pares $(t_n, f(t_n, \mathbf{u}(t_n))), (t_{n-1}, f(t_{n-1}, \mathbf{u}(t_{n-1}))), \cdots, (t_{n+1-m}, f(t_{n+1-m}, \mathbf{u}(t_{n+1-m})))$, con lo cual

$$\mathbf{u}^{n+1} \approx \mathbf{u}^n + \int_{t_n}^{t_{n+1}} \mathbf{P}_{m-1}(t)dt. \tag{3.15}$$

Un polinomio típico para este tipo de interpolaciones resulta ser el *Polinomio Regresivo de Newton*, de esta forma y considerando que existe un escalar $\xi^n \in [t_{n-1+m}; t_n]$ luego

$$f(t,\mathbf{u}(t)) = \mathbf{P}_{m-1}(t) + \underbrace{\frac{f^m(\xi^n,\mathbf{u}(\xi^n))}{m!}(t-t_n)(t-t_{n-1})\cdots(t-t_{n+1-m})}_{\text{error}}, \tag{3.16}$$

puesto que el polinomio es de orden $m-1$ existirá un error de truncamiento de orden m contemplado por aquel término etiquetado *error*.

Sea $s \in \mathbb{R}$ tal que $s \in [0;1]$, y definiendo $t = t_n + s\Delta t$ con lo cual tomando diferenciales a ambos miembros se obtiene $dt = \Delta t\, ds$ luego

$$\begin{aligned}\int_{t_n}^{t_{n+1}} f(t,\mathbf{u}(t))dt &= \int_{t_n}^{t_{n+1}} \underbrace{\sum_{k=0}^{m-1}(-1)^k\binom{-s}{k}\nabla^k f(t_n,\mathbf{u}^n)}_{\text{Polinomio regresivo de Newton}}\, dt \\ &\quad + \int_{t_n}^{t_{n+1}} \frac{f^m(\xi^n,\mathbf{u}(\xi^n))}{m!}(t-t_n)(t-t_{n-1})\cdots(t-t_{n+1-m})dt \\ &= \sum_{k=0}^{m-1}\nabla^k f(t_n,\mathbf{u}^n)\Delta t\,\underbrace{(-1)^k\int_0^1\binom{-s}{k}ds}_{*} \\ &\quad + \frac{\Delta t^{m+1}}{m!}\int_0^1 f^m(\xi^n,\mathbf{u}(\xi^n))s(s+1)\cdots(s+m-1)ds,\end{aligned} \tag{3.17}$$

donde la etiqueta (*) engloba a una integral polinómica función de k que, por ejemplo, para $k=0$ vale 1, para $k=1$ vale $1/2$ y para $k=2$ vale $5/12$.

Considerando lo anterior se tiene entonces

$$\begin{aligned}\int_{t_n}^{t_{n+1}} f(t,\mathbf{u}(t))dt =& \Delta t\left[f(t_n,\mathbf{u}^n) + \frac{1}{2}\nabla f(t_n,\mathbf{u}^n) + \frac{5}{12}\nabla^2 f(t_n,\mathbf{u}^n) + \cdots\right] \\ &+ \frac{\Delta t^{m+1}}{m!}\int_0^1 f^m(\xi^n,\mathbf{u}(\xi^n))s(s+1)\cdots(s+m-1)ds.\end{aligned} \tag{3.18}$$

Considerando además que $s(s+1)\cdots(s+m-1)$ no cambia de signo en $[0;1]$, aplicando el teorema de valor medio ponderado para integrales para algún valor $\mu_n \in [t_{i+1-m};t_{i+1}]$ luego

$$\begin{aligned}&\frac{\Delta t^{m+1}}{m!}\int_0^1 f^m(\xi^n,\mathbf{u}(\xi^n))s(s+1)\cdots(s+m-1)ds \\ &\qquad = \frac{\Delta t^{m+1}f^m(\mu_n,\mathbf{u}(\mu_n))}{m!}\int_0^1 s(s+1)\cdots(s+m-1)ds \\ &\qquad = \Delta t^{m+1}f^m(\mu_n,\mathbf{u}(\mu_n))(-1)^m\int_0^1\binom{-s}{m}ds,\end{aligned} \tag{3.19}$$

así finalmente introduciendo lo presentado en la ecuación (3.19) en la (3.15) se obtiene

$$\begin{aligned}\mathbf{u}^{n+1} = \mathbf{u}^n &+ \Delta t\left[f(t_n,\mathbf{u}^n) + \frac{1}{2}\nabla f(t_n,\mathbf{u}^n) + \frac{5}{15}\nabla^2 f(t_n,\mathbf{u}^n) + \cdots\right] \\ &+ \Delta t^{m+1}f^m(\mu_n,\mathbf{u}(\mu_n))(-1)^m\int_0^1\binom{-s}{m}ds.\end{aligned} \tag{3.20}$$

Con respecto al error de truncamiento, puede decirse que en la n-ésima iteración esta definido como

$$\begin{aligned}\boldsymbol{\tau}^{n+1} =& \frac{\mathbf{u}^{n+1} - a_{m-1}\mathbf{u}^n - \cdots - a_0\mathbf{u}^{n+1-m}}{\Delta t} \\ &- [b_m f(t_{n+1},\mathbf{u}^{n+1}) + \cdots + b_0 f(t_{n+1-m},\mathbf{u}^{n+1-m})].\end{aligned} \tag{3.21}$$

Considerando que el término k en la combinatoria de la ecuación (3.17) representa a las normalizaciones por Δt cuando se efectúa una operación de discretización de algún término del continuo, el esquema de Adams-Bashforth de segundo orden (AB2) que resulta de utilizar $k = 2$ es

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \frac{\Delta t}{2}\left[3f(t_n, \mathbf{u}^n) - f(t_{n-1}, \mathbf{u}^{n-1})\right], \tag{3.22}$$

con un error de truncamiento $\mathcal{O}(\Delta t^2)$.

3.5. El método de Pasos Fraccionados

El método de Pasos Fraccionados (o Fractional Step, de ahora en adelante F-S) es en esencia un esquema de integración temporal que propone un desacople en las ecuaciones de velocidad y presión resolviendo en primer instancia las ecuaciones de cantidad de movimiento obteniendo un campo de velocidad intermedio (paso predictor) que no cumple con la condición de incompresibilidad en el paso siguiente, por lo que debe ser corregido. A continuación se resuelve una ecuación de Poisson para la presión teniendo en cuenta que el campo de velocidad final debe de cumplir con la ecuación de continuidad (o la incompresibilidad). Finalmente con el campo presión resultante se corrige el campo velocidad obtenido anteriormente (paso corrector) y así se impone la condición de incompresibilidad que se necesitaba.

Reescribiendo las ecuaciones de N-S incompresibles en su forma dimensional conservativa vistas en el Capítulo (1), sin considerar fuerzas externas, se tiene el siguiente sistema a resolver

$$\begin{gathered}\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) = -\frac{1}{\rho}\nabla p + \nu \Delta \mathbf{u}, \\ \nabla \cdot \mathbf{u} = 0,\end{gathered} \tag{3.23}$$

considerando un esquema de integración temporal del tipo Forward Euler (es decir, explícito y de un paso) se define la primer etapa de F-S como

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -\nabla \cdot (\mathbf{u} \otimes \mathbf{u})^n + \nu \Delta \mathbf{u}^n, \tag{3.24}$$

donde se observa que sólo se resuelven los términos de convección y de tensiones. No está demás decir que se ha escogido Forward Euler como integración temporal sólo en aras de simplicidad, recordando el problema de estabilidad (condicionalmente estable para FE) inherente de los métodos explícitos. En la práctica métodos semi-implícitos o implícitos son utilizados, en especial en este PFC se ha utilizado Adams-Bashforth de segundo orden. Luego el campo de velocidades obtenido no cumple con la condición de incompresibilidad, por lo que debe ser sometido a algún tipo de corrección. Se tiene además que debe aplicarse algún criterio para la primer iteración, puesto que no se tiene información de una iteración previa. Es así que se propone aplicar Forward Euler para la primer iteración y posteriormente aplicar Adams-Bashforth.

Una vez resuelta las ecuaciones de cantidad de movimiento obteniendo el campo de velocidades intermedio, se tiene una ecuación que incorpora los términos de presión, esta es

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^*}{\Delta t} = -\frac{1}{\rho}\nabla p^{n+1}, \tag{3.25}$$

donde se observa que para obtener el valor del campo de velocidades en el paso de tiempo $n+1$ debe de tenerse el campo de presiones en ese mismo paso, lo cual introduce la idea que de alguna manera el campo de presiones en $n+1$ aplica una corrección al campo de velocidades en el paso de tiempo $*$. Es así que aplicando divergencia a ambos miembros de la ecuación (3.25) e imponiendo la condición de incompresibilidad en el paso $n+1$ se obtiene

$$\begin{aligned} \mathbf{u}^{n+1} &= \mathbf{u}^* - \frac{\Delta t}{\rho}\nabla p^{n+1}, \\ \underbrace{\nabla \cdot \mathbf{u}^{n+1}}_{=0 \text{ por incomp}} &= \nabla \cdot \mathbf{u}^* - \frac{\Delta t}{\rho}\nabla \cdot \left(\nabla p^{n+1}\right), \\ \Delta p^{n+1} &= \frac{\rho}{\Delta t}\left(\nabla \cdot \mathbf{u}^*\right), \end{aligned} \tag{3.26}$$

es decir, una ecuación de Poisson para la presión.

Una vez resuelta la ecuación (3.26), se procede con la ecuación (3.25) para obtener el campo de velocidad buscado.

Finalmente se presenta la ecuación para los pasos de tiempo $[1; N]$ que se ha seguido en este PFC, es decir, las ecuaciones de cantidad de movimiento con la integración temporal de Adams-Bashforth de segundo orden

$$\mathbf{u}^* = \mathbf{u}^n + \frac{\Delta t}{2}\left[3R(\mathbf{u}^n) - R(\mathbf{u}^{n-1})\right], \tag{3.27}$$

en donde $R(\mathbf{u}^n)$ esta definido como

$$R\left(\mathbf{u}^n\right) = -\nabla \cdot (\mathbf{u} \otimes \mathbf{u})^n + \nu \Delta \mathbf{u}^n, \tag{3.28}$$

y lo mismo para $R\left(\mathbf{u}^{n-1}\right)$. Finalmente se requiere de historia del campo de velocidades en un paso anterior.

Capítulo 4

Formulación discreta de N-S incompresible

En los capítulos previos se han introducido todas las herramientas, tanto físicas como matemáticas, que permitirán derivar una formulación discreta para el problema abordado. En este capítulo se presentarán otras herramientas utilizadas y finalmente la formulación completa del sistema de ecuaciones que será resuelto posteriormente.

4.1. Grillas staggered (desparramadas)

En general las técnicas de discretización de operadores diferenciales son centrados o upwind, los primeros relacionados al fenómenos de difusión (por ser de carácter isotrópico) mientras que los segundos relacionados (generalmente, no siempre) a fenómenos de convección. Sin embargo una discretización centrada para la convección es posible pero necesita de términos de amortiguamientos para errores numéricos de alta frecuencia.

Además se tiene que la falta del término de derivada de densidad (como en el caso compresible) y la discretización centrada (es decir, de segundo orden espacial) en las ecuaciones de cantidad de movimiento genera un desacople entre presión y velocidad en los métodos de corrección de presión. Suponiendo el caso 1D de las ecuaciones de N-S para fluidos incompresibles y una discretización de la presión de segundo orden, se tiene entonces que

$$\begin{aligned} \frac{u_i^{n+1} - u_i^*}{\Delta t} &= -\frac{1}{\rho}\frac{p_{i+1} - p_{i-1}}{2\Delta x}, \\ \rightarrow u_i^{n+1} &= u_i^n - \frac{\Delta t}{\rho}\frac{p_{i+1} - p_{i-1}}{2\Delta x}, \end{aligned} \tag{4.1}$$

e introduciéndolos en una discretización centrada para la ecuación de continuidad de la forma

$$\frac{u_{i+1}^{n+1} - u_{i-1}^{n+1}}{2\Delta x} = 0, \tag{4.2}$$

se obtiene una ecuación para la presión que presenta la forma

$$\frac{p_{i+2} - 2p_i + p_{i-2}}{4\Delta x^2} = \frac{\rho}{\Delta t}\frac{u_{i+1}^* - u_{i-1}^*}{2\Delta x}, \tag{4.3}$$

donde se puede observar que dado a su stencil $(i-2, i, i+2)$ luego sólo presenta nodos pares o impares, lo cual deriva en un desacople para nodos pares e impares que resultan en errores para la presión con componentes oscilatorios de alta frecuencia [Hir07, FP02, HC00]. Se tiene además que la velocidad en el nodo i no se relaciona con la presión en ese mismo nodo ni viceversa. Por lo tanto, velocidad y presión están desacoplados (situación que no ocurre en el caso compresible dado por la relación entre densidad y velocidad en la ecuación de continuidad).

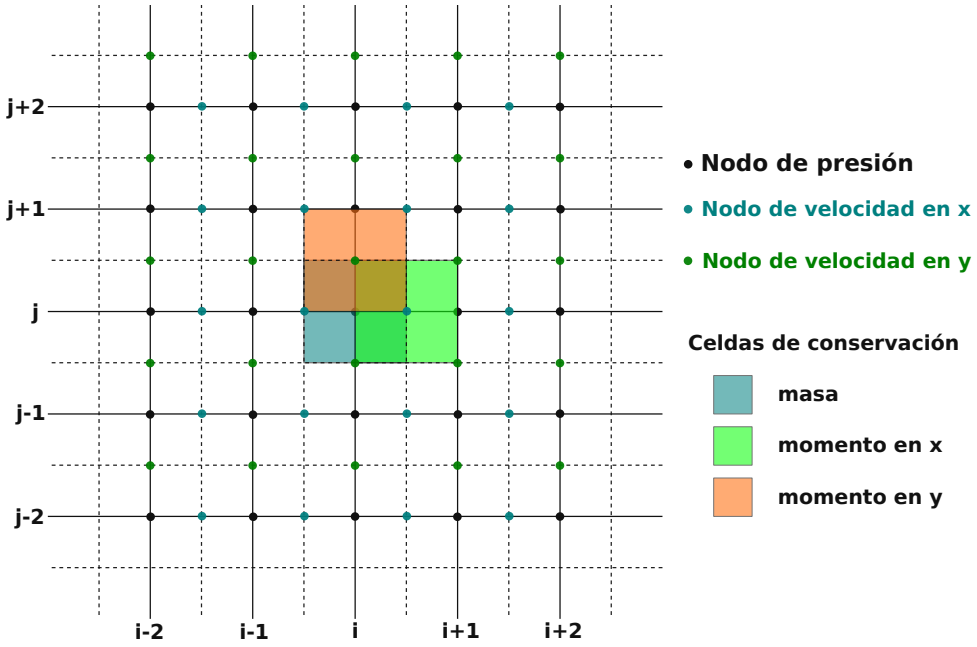


Figura 4.1: Ejemplo de grilla staggered 2D.

Una solución al problema del desacople consiste en utilizar grillas distintas para presión y velocidades. De esta forma, una posible elección son grillas para velocidades desplazadas en $\delta/2$, donde δ simboliza a Δx, Δy y Δz según direcciones x, y y z para el caso 3D. Luego la conservación se realiza sobre celdas cuyo centro se encuentra ubicado en el nodo donde se están calculando y su tamaño esta definido por el h de cada dimensión, de esta forma, y continuando con el caso 1D, se define nuevamente la ecuación de continuidad (centrada en celdas de presión) como

$$\frac{u^{n+1}_{i+\frac{1}{2}} - u^{n+1}_{i-\frac{1}{2}}}{\Delta x} = 0, \tag{4.4}$$

con lo cual la ecuación de Poisson para la presión resulta

$$\frac{p_{i+1} - 2p_i + p_{i-1}}{\Delta x^2} = \frac{\rho}{\Delta t}\frac{u^*_{i+\frac{1}{2}} - u^*_{i-\frac{1}{2}}}{2\Delta x}, \tag{4.5}$$

eliminando completamente el desacople de nodos pares e impares para la presión y los campos de velocidad y presión están totalmente acoplados (balances sobre el centro y caras de la celda).

En la Figura (4.1) se muestra un caso 2D de grillas staggered introducidas en este Capítulo, las líneas continuas pertenecen a la grilla estructurada homogénea del campo presión, mientras que las líneas discontinuas pertenecen a aquellas pertenecientes al campo velocidad.

4.2. Interpolación espacial por el método QUICK

De acuerdo a lo introducido en la sección de grillas staggered, las componentes del campo de velocidad están desplazadas en media unidad de longitud de acuerdo a la dirección que le corresponda. Como será visto más adelante será necesario obtener valores de presiones y velocidades descentradas, es decir, en otros valores nodales que no sean donde están definidos. Esta descentralización se realiza justo sobre media unidad de longitud, por ende es necesario un esquema de interpolación sobre esa posición que garantice un término de error de al menos el error de truncamiento del esquema de integración temporal. Existen varias formas de interpolar un valor entre nodos, considerando un dominio de interpolación de tres nodos uno puede definir un polinomio de segundo orden definido por las coordenadas de los tres nodos.

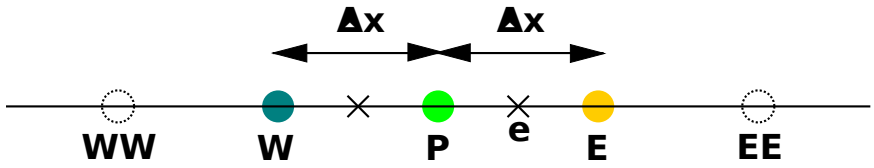


Figura 4.2: Nodos necesarios para el cálculo en 1D del esquema de interpolación QUICK cuando $u_x > 0$.

Considerando la topología de los nodos presentados en la Figura (4.2), uno podría realizar una expansión de Taylor sobre u_e de acuerdo a

$$u_P = u_e - \frac{\Delta x}{2}u_e^{'} + \frac{\Delta x^2}{4}u_e^{''} - \mathcal{O}(\Delta x^3), \tag{4.6}$$

la misma expansión podría tomar la forma

$$u_E = u_e + \frac{\Delta x}{2}u_e^{'} + \frac{\Delta x^2}{4}u_e^{''} + \mathcal{O}(\Delta x^3), \tag{4.7}$$

o

$$u_W = u_e - \frac{3\Delta x}{2}u_e^{'} + \frac{9\Delta x^2}{4}u_e^{''} - \mathcal{O}(\Delta x^3). \tag{4.8}$$

El objetivo es tomar una combinación lineal de las ecuaciones (4.6-4.8) que conserve un error de truncamiento $\mathcal{O}(\Delta x^3)$, así todos los términos en derivadas primeras y segundas deberían anularse.

Sumando la ecuación (4.6) con la (4.7) se obtiene

$$u_P + u_E = 2u_e + \frac{\Delta x^2}{2}u_e^{''} - \mathcal{O}(\Delta x^3), \tag{4.9}$$

mientras que multiplicar por 3 a la ecuación (4.6) y restarle la (4.8) resulta en

$$3u_P - u_W = 2u_e - \frac{3\Delta x^2}{2}u_e^{''} + \mathcal{O}(\Delta x^3). \tag{4.10}$$

Finalmente multiplicando por 3 la ecuación (4.9) y sumándole la (4.10) se obtiene

$$\begin{aligned} 6u_P + 3u_E - u_W &= 8u_e + \mathcal{O}(\Delta x^3) \\ \rightarrow u_e &= \frac{3}{8}u_E + \frac{6}{8}u_P - \frac{1}{8}u_W - \mathcal{O}(\Delta x^3), \end{aligned} \tag{4.11}$$

con lo cual se obtiene una expresión para la interpolación de nodos desplazados en media unidad por dirección cuyo error de truncamiento es proporcional a $\mathcal{O}(\Delta x^3)$. Un proceso similar se utiliza para encontrar la expresión cuando se tiene $u_x < 0$, que resulta ser

$$u_e = \frac{3}{8}u_P + \frac{6}{8}u_E - \frac{1}{8}u_{EE} - \mathcal{O}(\Delta x^3), \tag{4.12}$$

y similarmente en las direcciones y y z.

Lo que no se ha introducido al momento es que este esquema de interpolación en realidad actúa como estabilización para el término convectivo en las ecuaciones de cantidad de movimiento. Su asimetría con respecto al nodo central y su forma cuadrática le brindan conservación de masa y otras características de estabilidad que, si bien no van a ser introducidas en este texto, pueden ser vistan en [Leo79].

4.3. Discretización de las ecuaciones de cantidad de movimiento

Finalmente luego de la introducción del problema en el continuo, las discretizaciones espaciales y temporales y posteriormente el planteo del método de Pasos Fraccionados y grillas staggered, es que se presentan las ecuaciones resultantes que se han utilizado en este PFC.

Desagregando la ecuación (1.29) en sus respectivas direcciones se obtiene para el caso 3D el sistema

$$\begin{aligned}
\frac{\partial u}{\partial t}+\frac{\partial\left(uu\right)}{\partial x}+\frac{\partial(uv)}{\partial y}+\frac{\partial(uw)}{\partial z} &= -\frac{1}{\rho}\frac{\partial p}{\partial x}+\nu\Delta u,\\
\frac{\partial v}{\partial t}+\frac{\partial(vu)}{\partial x}+\frac{\partial\left(vv\right)}{\partial y}+\frac{\partial(vw)}{\partial z} &= -\frac{1}{\rho}\frac{\partial p}{\partial y}+\nu\Delta v,\\
\frac{\partial w}{\partial t}+\frac{\partial(wu)}{\partial x}+\frac{\partial(wv)}{\partial y}+\frac{\partial\left(ww\right)}{\partial z} &= -\frac{1}{\rho}\frac{\partial p}{\partial z}+\nu\Delta w,\\
\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z} &= 0.
\end{aligned} \tag{4.13}$$

Considerando la definición del residuo del campo de velocidades $\mathbf{u}$ dada por la ecuación (3.28), es que se tiene que el residuo aplicado a la componente u del campo de velocidades (esto es, velocidad según la dirección x) como

$$\begin{aligned}
\left(R^{*}\left(u^{n}\right)\right)_{i+\frac{1}{2},j,k} = &-\left\{\frac{(uu^{Q})^{n}_{i+1,j,k}-(uu^{Q})^{n}_{i,j,k}}{\Delta x}\right\}\\
&-\left\{\frac{(vu^{Q})^{n}_{i+\frac{1}{2},j+\frac{1}{2},k}-(vu^{Q})^{n}_{i+\frac{1}{2},j-\frac{1}{2},k}}{\Delta y}\right\}\\
&-\left\{\frac{(wu^{Q})^{n}_{i+\frac{1}{2},j,k+\frac{1}{2}}-(wu^{Q})^{n}_{i+\frac{1}{2},j,k-\frac{1}{2}}}{\Delta z}\right\}\\
&+\nu\left\{\frac{u^{n}_{i+\frac{3}{2},j,k}-2u^{n}_{i+\frac{1}{2},j,k}+u^{n}_{i-\frac{1}{2},j,k}}{\Delta x^{2}}\right.\\
&+\frac{u^{n}_{i+\frac{1}{2},j+1,k}-2u^{n}_{i+\frac{1}{2},j,k}+u^{n}_{i+\frac{1}{2},j-1,k}}{\Delta y^{2}}\\
&\left.+\frac{u^{n}_{i+\frac{1}{2},j,k+1}-2u^{n}_{i+\frac{1}{2},j,k}+u^{n}_{i+\frac{1}{2},j,k-1}}{\Delta z^{2}}\right\},
\end{aligned} \tag{4.14}$$

donde se observa que el residuo se toma en la iteración n-ésima, nodo $(i+\frac{1}{2},j,k)$ de la componente u del campo de velocidades $\mathbf{u}$ (claramente realizando el balance sobre la celda de cantidad de movimiento según x), y el superíndice Q haciendo referencia QUICK.

Se tiene además que algunos nodos que aparecen en la ecuación (4.14) no se encuentran en la grilla de velocidades de u, por lo cual requieren de aproximaciones que definan a los mismos. Un esquema simple para solucionar lo anterior consiste en promediar el valor mediante los dos nodos más próximos, según la dirección que se requiera. Entonces considerando esto se obtiene

$$
\begin{aligned}
u^n_{i+1,j,k} &= \frac{1}{2}\left(u^n_{i+\frac{3}{2},j,k} + u^n_{i+\frac{1}{2},j,k}\right),\\
u^n_{i,j,k} &= \frac{1}{2}\left(u^n_{i+\frac{1}{2},j,k} + u^n_{i-\frac{1}{2},j,k}\right),\\
v^n_{i+\frac{1}{2},j+\frac{1}{2},k} &= \frac{1}{2}\left(v^n_{i+1,j+\frac{1}{2},k} + v^n_{i,j+\frac{1}{2},k}\right),\\
v^n_{i+\frac{1}{2},j-\frac{1}{2},k} &= \frac{1}{2}\left(v^n_{i+1,j-\frac{1}{2},k} + v^n_{i,j-\frac{1}{2},k}\right),\\
w^n_{i+\frac{1}{2},j,k+\frac{1}{2}} &= \frac{1}{2}\left(w^n_{i+1,j,k+\frac{1}{2}} + w^n_{i,j,k+\frac{1}{2}}\right),\\
w^n_{i+\frac{1}{2},j,k-\frac{1}{2}} &= \frac{1}{2}\left(w^n_{i+1,j,k-\frac{1}{2}} + w^n_{i,j,k-\frac{1}{2}}\right).
\end{aligned}
\tag{4.15}
$$

Mientras que aquellos nodos que requieren de la aproximación QUICK se definen como

$$
\begin{aligned}
\left(u^Q\right)^n_{i+1,j,k} &= \begin{cases} c_0u^n_{i+\frac{3}{2},j,k} + c_1u^n_{i+\frac{1}{2},j,k} + c_2u^n_{i-\frac{1}{2},j,k},\ si\ \ u^n_{i+1,j,k} \geq 0\\ c_0u^n_{i+\frac{1}{2},j,k} + c_1u^n_{i+\frac{3}{2},j,k} + c_2u^n_{i+\frac{5}{2},j,k},\ si\ \ u^n_{i+1,j,k} < 0 \end{cases}\\
\left(u^Q\right)^n_{i,j,k} &= \begin{cases} c_0u^n_{i+\frac{1}{2},j,k} + c_1u^n_{i-\frac{1}{2},j,k} + c_2u^n_{i-\frac{3}{2},j,k},\ si\ \ u^n_{i,j,k} \geq 0\\ c_0u^n_{i-\frac{1}{2},j,k} + c_1u^n_{i+\frac{1}{2},j,k} + c_2u^n_{i+\frac{3}{2},j,k},\ si\ \ u^n_{i,j,k} < 0 \end{cases}\\
\left(u^Q\right)^n_{i+\frac{1}{2},j+\frac{1}{2},k} &= \begin{cases} c_0u^n_{i+\frac{1}{2},j+1,k} + c_1u^n_{i+\frac{1}{2},j,k} + c_2u^n_{i+\frac{1}{2},j-1,k},\ si\ \ v^n_{i+\frac{1}{2},j+\frac{1}{2},k} \geq 0\\ c_0u^n_{i+\frac{1}{2},j,k} + c_1u^n_{i+\frac{1}{2},j+1,k} + c_2u^n_{i+\frac{1}{2},j+2,k},\ si\ \ v^n_{i+\frac{1}{2},j+\frac{1}{2},k} < 0 \end{cases}\\
\left(u^Q\right)^n_{i+\frac{1}{2},j-\frac{1}{2},k} &= \begin{cases} c_0u^n_{i+\frac{1}{2},j,k} + c_1u^n_{i+\frac{1}{2},j-1,k} + c_2u^n_{i+\frac{1}{2},j-2,k},\ si\ \ v^n_{i+\frac{1}{2},j-\frac{1}{2},k} \geq 0\\ c_0u^n_{i+\frac{1}{2},j-1,k} + c_1u^n_{i+\frac{1}{2},j,k} + c_2u^n_{i+\frac{1}{2},j+1,k},\ si\ \ v^n_{i+\frac{1}{2},j-\frac{1}{2},k} < 0 \end{cases}\\
\left(u^Q\right)^n_{i+\frac{1}{2},j,k+\frac{1}{2}} &= \begin{cases} c_0u^n_{i+\frac{1}{2},j,k+1} + c_1u^n_{i+\frac{1}{2},j,k} + c_2u^n_{i+\frac{1}{2},j,k-1},\ si\ \ w^n_{i+\frac{1}{2},j,k+\frac{1}{2}} \geq 0\\ c_0u^n_{i+\frac{1}{2},j,k} + c_1u^n_{i+\frac{1}{2},j,k+1} + c_2u^n_{i+\frac{1}{2},j,k+2},\ si\ \ w^n_{i+\frac{1}{2},j,k+\frac{1}{2}} < 0 \end{cases}\\
\left(u^Q\right)^n_{i+\frac{1}{2},j,k-\frac{1}{2}} &= \begin{cases} c_0u^n_{i+\frac{1}{2},j,k} + c_1u^n_{i+\frac{1}{2},j,k-1} + c_2u^n_{i+\frac{1}{2},j,k-2},\ si\ \ w^n_{i+\frac{1}{2},j,k-\frac{1}{2}} \geq 0\\ c_0u^n_{i+\frac{1}{2},j,k-1} + c_1u^n_{i+\frac{1}{2},j,k} + c_2u^n_{i+\frac{1}{2},j,k+1},\ si\ \ w^n_{i+\frac{1}{2},j,k-\frac{1}{2}} < 0 \end{cases}
\end{aligned}
\tag{4.16}
$$

Continuando con las discretizaciones de las ecuaciones de cantidad de movimiento, se presentan ahora aquellas afectadas en dirección y. El balance se realiza sobre la celda centrada en $(i, j+\frac{1}{2}, k)$, entonces

$$
\begin{aligned}
\left(R^*\left(v^n\right)\right)_{i,j+\frac{1}{2},k} = &-\left\{\frac{(uv^Q)^n_{i+\frac{1}{2},j+\frac{1}{2},k} - (uv^Q)^n_{i-\frac{1}{2},j+\frac{1}{2},k}}{\Delta x}\right\}\\
&-\left\{\frac{(vv^Q)^n_{i,j+1,k} - (vv^Q)^n_{i,j,k}}{\Delta y}\right\}\\
&-\left\{\frac{(wv^Q)^n_{i,j+\frac{1}{2},k+\frac{1}{2}} - (wv^Q)^n_{i,j+\frac{1}{2},k-\frac{1}{2}}}{\Delta z}\right\}\\
&+\nu\left\{\frac{v^n_{i+1,j+\frac{1}{2},k} - 2v^n_{i,j+\frac{1}{2},k} + v^n_{i-1,j+\frac{1}{2},k}}{\Delta x^2}\right.\\
&+\frac{v^n_{i,j+\frac{3}{2},k} - 2v^n_{i,j+\frac{1}{2},k} + v^n_{i,j-\frac{1}{2},k}}{\Delta y^2}\\
&\left.+\frac{v^n_{i,j+\frac{1}{2},k+1} - 2v^n_{i,j+\frac{1}{2},k} + v^n_{i,j+\frac{1}{2},k-1}}{\Delta z^2}\right\},
\end{aligned}
\tag{4.17}
$$

donde el campo de velocidades centradas se define como

$$
\begin{aligned}
u^n_{i+\frac{1}{2},j+\frac{1}{2},k} &= \frac{1}{2}\left(u^n_{i+\frac{1}{2},j+1,k} + u^n_{i+\frac{1}{2},j,k}\right),\\
u^n_{i-\frac{1}{2},j+\frac{1}{2},k} &= \frac{1}{2}\left(u^n_{i-\frac{1}{2},j+1,k} + u^n_{i-\frac{1}{2},j,k}\right),\\
v^n_{i,j+1,k} &= \frac{1}{2}\left(v^n_{i,j+\frac{3}{2},k} + v^n_{i,j+\frac{1}{2},k}\right),\\
v^n_{i,j,k} &= \frac{1}{2}\left(v^n_{i,j+\frac{1}{2},k} + v^n_{i,j-\frac{1}{2},k}\right),\\
w^n_{i,j+\frac{1}{2},k+\frac{1}{2}} &= \frac{1}{2}\left(w^n_{i,j+1,k+\frac{1}{2}} + w^n_{i,j,k+\frac{1}{2}}\right),\\
w^n_{i,j+\frac{1}{2},k-\frac{1}{2}} &= \frac{1}{2}\left(w^n_{i,j+1,k-\frac{1}{2}} + w^n_{i,j,k-\frac{1}{2}}\right),
\end{aligned}
\tag{4.18}
$$

y aquellas utilizando QUICK como

$$
\begin{aligned}
\left(v^Q\right)^n_{i+\frac{1}{2},j+\frac{1}{2},k} &= \begin{cases} c_0 v^n_{i+1,j+\frac{1}{2},k} + c_1 v^n_{i,j+\frac{1}{2},k} + c_2 v^n_{i-1,j+\frac{1}{2},k}, u^n_{i+\frac{1}{2},j+\frac{1}{2},k} \geq 0\\ c_0 v^n_{i,j+\frac{1}{2},k} + c_1 v^n_{i+1,j+\frac{1}{2},k} + c_2 v^n_{i+2,j+\frac{1}{2},k}, u^n_{i+\frac{1}{2},j+\frac{1}{2},k} < 0 \end{cases}\\
\left(v^Q\right)^n_{i-\frac{1}{2},j+\frac{1}{2},k} &= \begin{cases} c_0 v^n_{i,j+\frac{1}{2},k} + c_1 v^n_{i-1,j+\frac{1}{2},k} + c_2 v^n_{i-2,j+\frac{1}{2},k}, u^n_{i-\frac{1}{2},j+\frac{1}{2},k} \geq 0\\ c_0 v^n_{i-1,j+\frac{1}{2},k} + c_1 v^n_{i,j+\frac{1}{2},k} + c_2 v^n_{i+1,j+\frac{1}{2},k}, u^n_{i-\frac{1}{2},j+\frac{1}{2},k} < 0 \end{cases}\\
\left(v^Q\right)^n_{i,j+1,k} &= \begin{cases} c_0 v^n_{i,j+\frac{3}{2},k} + c_1 v^n_{i,j+\frac{1}{2},k} + c_2 v^n_{i,j-\frac{1}{2},k}, v^n_{i,j+1,k} \geq 0\\ c_0 v^n_{i,j+\frac{1}{2},k} + c_1 v^n_{i,j+\frac{3}{2},k} + c_2 v^n_{i,j+\frac{5}{2},k}, v^n_{i,j+1,k} < 0 \end{cases}\\
\left(v^Q\right)^n_{i,j,k} &= \begin{cases} c_0 v^n_{i,j+\frac{1}{2},k} + c_1 v^n_{i,j-\frac{1}{2},k} + c_2 v^n_{i,j-\frac{3}{2},k}, v^n_{i,j,k} \geq 0\\ c_0 v^n_{i,j-\frac{1}{2},k} + c_1 v^n_{i,j+\frac{1}{2},k} + c_2 v^n_{i,j+\frac{3}{2},k}, v^n_{i,j,k} < 0 \end{cases}\\
\left(v^Q\right)^n_{i,j+\frac{1}{2},k+\frac{1}{2}} &= \begin{cases} c_0 v^n_{i,j+\frac{1}{2},k+1} + c_1 v^n_{i,j+\frac{1}{2},k} + c_2 v^n_{i,j+\frac{1}{2},k-1}, w^n_{i,j+\frac{1}{2},k+\frac{1}{2}} \geq 0\\ c_0 v^n_{i,j+\frac{1}{2},k} + c_1 v^n_{i,j+\frac{1}{2},k+1} + c_2 v^n_{i,j+\frac{1}{2},k+2}, w^n_{i,j+\frac{1}{2},k+\frac{1}{2}} < 0 \end{cases}\\
\left(v^Q\right)^n_{i,j+\frac{1}{2},k-\frac{1}{2}} &= \begin{cases} c_0 v^n_{i,j+\frac{1}{2},k} + c_1 v^n_{i,j+\frac{1}{2},k-1} + c_2 v^n_{i,j+\frac{1}{2},k-2}, w^n_{i,j+\frac{1}{2},k-\frac{1}{2}} \geq 0\\ c_0 v^n_{i,j+\frac{1}{2},k-1} + c_1 v^n_{i,j+\frac{1}{2},k} + c_2 v^n_{i,j+\frac{1}{2},k+1}, w^n_{i,j+\frac{1}{2},k-\frac{1}{2}} < 0. \end{cases}
\end{aligned}
\tag{4.19}
$$

Finalmente se presentan aquellas ecuaciones resultantes obtenidas mediante el balance sobre la celda centrada en $(i, j, k+\frac{1}{2})$, obteniendo

$$
\begin{aligned}
\left(R^*\left(w^n\right)\right)_{i,j,k+\frac{1}{2}} = &-\left\{\frac{(uw^Q)^n_{i+\frac{1}{2},j,k+\frac{1}{2}} - (uw^Q)^n_{i-\frac{1}{2},j,k+\frac{1}{2}}}{\Delta x}\right\}\\
&-\left\{\frac{(vw^Q)^n_{i,j+\frac{1}{2},k+\frac{1}{2}} - (vw^Q)^n_{i,j-\frac{1}{2},k+\frac{1}{2}}}{\Delta y}\right\}\\
&-\left\{\frac{(ww^Q)^n_{i,j,k+1} - (ww^Q)^n_{i,j,k}}{\Delta z}\right\}\\
&+\nu\left\{\frac{w^n_{i+1,j,k+\frac{1}{2}} - 2w^n_{i,j,k+\frac{1}{2}} + w^n_{i-1,j,k+\frac{1}{2}}}{\Delta x^2}\right.\\
&+\frac{w^n_{i,j+1,k+\frac{1}{2}} - 2w^n_{i,j,k+\frac{1}{2}} + v^n_{i,j-1,k+\frac{1}{2}}}{\Delta y^2}\\
&\left.+\frac{w^n_{i,j,k+\frac{3}{2}} - 2w^n_{i,j,k+\frac{1}{2}} + w^n_{i,j,k-\frac{1}{2}}}{\Delta z^2}\right\},
\end{aligned}
\tag{4.20}
$$

donde el campo de velocidades centradas se define como

$$
\begin{aligned}
u^n_{i+\frac{1}{2},j,k+\frac{1}{2}} &= \frac{1}{2}\left(u^n_{i+\frac{1}{2},j,k+1} + u^n_{i+\frac{1}{2},j,k}\right),\\
u^n_{i-\frac{1}{2},j,k+\frac{1}{2}} &= \frac{1}{2}\left(u^n_{i-\frac{1}{2},j,k+1} + u^n_{i-\frac{1}{2},j,k}\right),\\
v^n_{i,j+\frac{1}{2},k+\frac{1}{2}} &= \frac{1}{2}\left(v^n_{i,j+\frac{1}{2},k+1} + v^n_{i,j+\frac{1}{2},k}\right),\\
v^n_{i,j-\frac{1}{2},k+\frac{1}{2}} &= \frac{1}{2}\left(v^n_{i,j-\frac{1}{2},k+1} + v^n_{i,j-\frac{1}{2},k}\right),\\
w^n_{i,j,k+1} &= \frac{1}{2}\left(w^n_{i,j,k+\frac{3}{2}} + w^n_{i,j,k+\frac{1}{2}}\right),\\
w^n_{i,j,k} &= \frac{1}{2}\left(w^n_{i,j,k+\frac{1}{2}} + w^n_{i,j,k-\frac{1}{2}}\right),
\end{aligned}
\tag{4.21}
$$

y finalmente aquellas utilizando QUICK como

$$
\begin{aligned}
\left(w^Q\right)^n_{i+\frac{1}{2},j,k+\frac{1}{2}} &= \begin{cases} c_0 w^n_{i+1,j,k+\frac{1}{2}} + c_1 w^n_{i,j,k+\frac{1}{2}} + c_2 w^n_{i-1,j,k+\frac{1}{2}}, u^n_{i+\frac{1}{2},j,k+\frac{1}{2}} \geq 0\\ c_0 w^n_{i,j,k+\frac{1}{2}} + c_1 w^n_{i+1,j,k+\frac{1}{2}} + c_2 w^n_{i+2,j,k+\frac{1}{2}}, u^n_{i+\frac{1}{2},j,k+\frac{1}{2}} < 0 \end{cases}\\
\left(w^Q\right)^n_{i-\frac{1}{2},j,k+\frac{1}{2}} &= \begin{cases} c_0 w^n_{i,j,k+\frac{1}{2}} + c_1 w^n_{i-1,j,k+\frac{1}{2}} + c_2 w^n_{i-2,j,k+\frac{1}{2}}, u^n_{i-\frac{1}{2},j,k+\frac{1}{2}} \geq 0\\ c_0 w^n_{i-1,j,k+\frac{1}{2}} + c_1 w^n_{i,j,k+\frac{1}{2}} + c_2 w^n_{i+1,j,k+\frac{1}{2}}, u^n_{i-\frac{1}{2},j,k+\frac{1}{2}} < 0 \end{cases}\\
\left(w^Q\right)^n_{i,j+\frac{1}{2},k+\frac{1}{2}} &= \begin{cases} c_0 w^n_{i,j+1,k+\frac{1}{2}} + c_1 w^n_{i,j,k+\frac{1}{2}} + c_2 w^n_{i,j-1,k+\frac{1}{2}}, v^n_{i,j+\frac{1}{2},k+\frac{1}{2}} \geq 0\\ c_0 w^n_{i,j,k+\frac{1}{2}} + c_1 w^n_{i,j+1,k+\frac{1}{2}} + c_2 w^n_{i,j+2,k+\frac{1}{2}}, v^n_{i,j+\frac{1}{2},k+\frac{1}{2}} < 0 \end{cases}\\
\left(w^Q\right)^n_{i,j-\frac{1}{2},k+\frac{1}{2}} &= \begin{cases} c_0 w^n_{i,j,k+\frac{1}{2}} + c_1 w^n_{i,j-1,k+\frac{1}{2}} + c_2 w^n_{i,j-2,k+\frac{1}{2}}, v^n_{i,j-\frac{1}{2},k+\frac{1}{2}} \geq 0\\ c_0 w^n_{i,j-1,k+\frac{1}{2}} + c_1 w^n_{i,j,k+\frac{1}{2}} + c_2 w^n_{i,j+1,k+\frac{1}{2}}, v^n_{i,j-\frac{1}{2},k+\frac{1}{2}} < 0 \end{cases}\\
\left(w^Q\right)^n_{i,j,k+1} &= \begin{cases} c_0 w^n_{i,j,k+\frac{3}{2}} + c_1 w^n_{i,j,k+\frac{1}{2}} + c_2 w^n_{i,j,k-\frac{1}{2}}, w^n_{i,j,k+1} \geq 0\\ c_0 w^n_{i,j,k+\frac{1}{2}} + c_1 w^n_{i,j,k+\frac{3}{2}} + c_2 w^n_{i,j,k+\frac{5}{2}}, w^n_{i,j,k+1} < 0 \end{cases}\\
\left(w^Q\right)^n_{i,j,k} &= \begin{cases} c_0 w^n_{i,j,k+\frac{1}{2}} + c_1 w^n_{i,j,k-\frac{1}{2}} + c_2 w^n_{i,j,k-\frac{3}{2}}, w^n_{i,j,k} \geq 0\\ c_0 w^n_{i,j,k-\frac{1}{2}} + c_1 w^n_{i,j,k+\frac{1}{2}} + c_2 w^n_{i,j,k+\frac{3}{2}}, w^n_{i,j,k} < 0. \end{cases}
\end{aligned}
\tag{4.22}
$$

4.4. Discretización de la ecuación de continuidad

La ecuación de continuidad se define mediante una simple discretización centrada realizando el balance sobre la celda centrada en el nodo de presión (i, j, k) obteniendo de esta manera

$$
\frac{u^{n+1}_{i+\frac{1}{2},j,k} - u^{n+1}_{i-\frac{1}{2},j,k}}{\Delta x} + \frac{v^{n+1}_{i,j+\frac{1}{2},k} - v^{n+1}_{i,j-\frac{1}{2},k}}{\Delta y} + \frac{w^{n+1}_{i,j,k+\frac{1}{2}} - w^{n+1}_{i,j,k-\frac{1}{2}}}{\Delta z} = 0. \tag{4.23}
$$

4.5. Discretización de los operadores divergencia y gradiente

En la segunda etapa del desarrollo se debe calcular una ecuación de Poisson para la presión que presenta la forma

$$
\left(\Delta p^{n+1}\right)_{i,j,k} = \frac{\Delta t}{\rho}\left(\nabla \cdot \mathbf{u}^*\right)_{i,j,k}, \tag{4.24}
$$

notar que el balance se realiza en la celda de presión, es decir, centrada en el nodo (i,j,k). Entonces como primer paso se calcula la divergencia y luego se aplica el factor de escala. La divergencia se calcula como

$$
\left(\nabla \cdot \mathbf{u}^*\right)_{i,j,k} = \frac{u^*_{i+\frac{1}{2},j,k} - u^*_{i-\frac{1}{2},j,k}}{\Delta x} + \frac{v^*_{i,j+\frac{1}{2},k} - v^*_{i,j-\frac{1}{2},k}}{\Delta y} + \frac{w^*_{i,j,k+\frac{1}{2}} - w^*_{i,j,k-\frac{1}{2}}}{\Delta z}, \tag{4.25}
$$

donde se observa que el balance también se hace en la celda centrada en el nodo de presión (i, j, k) y el truncamiento resulta $\mathcal{O}(\Delta x^2)$.

Además en la tercer etapa se necesitan los gradientes de presión para la corrección del campo de velocidad $\mathbf{u}^*$, luego se calculan como

$$\nabla p^{n+1} = \left\{ \begin{array}{l} \left(\dfrac{\partial p}{\partial x}\right)^{n+1}_{i+\frac{1}{2},j,k} \\ \left(\dfrac{\partial p}{\partial y}\right)^{n+1}_{i,j+\frac{1}{2},k} \\ \left(\dfrac{\partial p}{\partial z}\right)^{n+1}_{i,j,k+\frac{1}{2}} \end{array} \right\} = \left\{ \begin{array}{l} \dfrac{p^{n+1}_{i+1,j,k} - p^{n+1}_{i,j,k}}{\Delta x} \\ \dfrac{p^{n+1}_{i,j+1,k} - p^{n+1}_{i,j,k}}{\Delta y} \\ \dfrac{p^{n+1}_{i,j,k+1} - p^{n+1}_{i,j,k}}{\Delta z} \end{array} \right\}, \tag{4.26}$$

donde se observa el balance de los gradientes de presión sobre las celdas de velocidad y también presentan un error de truncamiento $\mathcal{O}(\Delta x^2)$. Finalmente se obtiene el campo de velocidades $\mathbf{u}^{n+1}$ mediante

$$\mathbf{u}^{n+1}_{i,j,k} = \left\{ \begin{array}{l} u^{n+1}_{i+\frac{1}{2},j,k} \\ v^{n+1}_{i,j+\frac{1}{2},k} \\ w^{n+1}_{i,j,k+\frac{1}{2}} \end{array} \right\} = \left\{ \begin{array}{l} u^{*}_{i+\frac{1}{2},j,k} - \dfrac{\Delta t}{\rho}\left(\dfrac{\partial p}{\partial x}\right)^{n+1}_{i+\frac{1}{2},j,k} \\ v^{*}_{i,j+\frac{1}{2},k} - \dfrac{\Delta t}{\rho}\left(\dfrac{\partial p}{\partial y}\right)^{n+1}_{i,j+\frac{1}{2},k} \\ w^{*}_{i,j,k+\frac{1}{2}} - \dfrac{\Delta t}{\rho}\left(\dfrac{\partial p}{\partial z}\right)^{n+1}_{i,j,k+\frac{1}{2}} \end{array} \right\}. \tag{4.27}$$

Capítulo 5

Resolución de sistemas de ecuaciones lineales

En resumen de lo realizado hasta este punto, se tiene que de un modelo analítico en ecuaciones diferenciales parciales (PDE's) se procedió a la discretización de las ecuaciones en el continuo obteniendo como producto intermedio un sistema de ecuaciones diferenciales ordinarias (ODE). Luego, se aplicó la discretización temporal obteniendo finalmente un sistema de ecuaciones lineales.

Este capítulo tiene como objetivo principal introducir el método gracias al cual se ha resuelto dicho sistema, denominado *Gradientes Conjugados*.

5.1. Introducción

Cualquier problema que necesite ser procesado por una computadora necesariamente debe atravesar por sucesivos pasos para finalmente obtener un sistema de ecuaciones, lineales o no lineales, que requieren de una resolución. Como es costumbre, se utilizarán letras mayúsculas en negrita para hacer referencia a las matrices, mientras que las minúsculas en negritas para referenciar vectores, además de minúsculas para escalares.

Sea el sistema a resolver de la forma

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{5.1}$$

donde $A \in \mathbb{R}^{n\times n}$ y $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$ para resolver un sistema de ecuaciones, en principio lineales, como aquel presentado por la ecuación (5.1), básicamente se opera con dos estrategias bien conocidas. En primer lugar existe una clase de métodos denominados *directos*, que mediante alguna estrategia resuelven el sistema exactamente. Como principal ventaja de estos, es su entendimiento y su facilidad de programación. En segundo lugar se tienen los métodos iterativos los cuales en base a una solución inicial $\mathbf{x}^0$ y aplicando sistemáticas correcciones siguiendo algún criterio, tienen como objetivo ir aproximando la solución exacta del sistema.

No está de más decir que de acuerdo a la aplicación, una u otra clase de métodos tendrá ventaja sobre la otra. En general, dado la estructura de los problemas de diferencias finitas o elementos finitos, donde se tiene matrices con propiedades como ser *sparse*, *banda*, *simétricas*, *definidas positivas*, entre otras tantas, se utilizan frecuentemente métodos iterativos, como ser el de *Gradientes Conjugados*, que será introducido en esta sección. En general este método es utilizado en grandes sistemas donde la matriz LHS resulta simétrica y definida positiva.

5.2. Gradientes Conjugados

Supongamos que se tiene un sistema lineal $\mathbf{A} \in \mathbb{R}^{n\times n}$, es decir, n ecuaciones con n incógnitas, en principio con todas las filas de la matriz linealmente independientes (o de rango $R(\mathbf{A}) = n$), además $\mathbf{A}$

presenta la propiedad de simetría $\mathbf{A} = \mathbf{A^T}$ y también de definida positiva, es decir que para cualquier vector $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x^T A x} > 0$ a menos que $\mathbf{x} = \mathbf{0}$.

En aras de simplicidad, se presenta a continuación un teorema que relaciona operaciones vectoriales y matriciales.

Teorema 5.2.1. *Si se consideran los vectores $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$, y además un escalar $\alpha \in \mathbb{R}$, entonces se tiene*

(i) $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$;

(ii) $\langle \alpha\mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, \alpha\mathbf{y} \rangle = \alpha \langle \mathbf{x}, \mathbf{y} \rangle$;

(iii) $\langle \mathbf{x} + \mathbf{z}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{z}, \mathbf{y} \rangle$;

(iv) $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$;

(v) $\langle \mathbf{x}, \mathbf{x} \rangle = 0 \iff \mathbf{x} = \mathbf{0}$.

en donde el operador $< ., . >$ representa un producto interno. Si se supone que la matriz $\mathbf{A}$ es simétrica luego $\langle \mathbf{x}, \mathbf{Ay} \rangle = \langle \mathbf{Ax}, \mathbf{y} \rangle$.

Con los resultados recién introducidos, se tiene el siguiente teorema básico para el desarrollo del método de Gradientes Conjugados.

Teorema 5.2.2. *El vector $\mathbf{x}^* \in \mathbb{R}^n$ es una solución del sistema lineal presentado en la ecuación (5.1) si y sólo si $\mathbf{x}^*$ minimiza*

$$g(\mathbf{x}) = \langle \mathbf{x}, \mathbf{Ax} \rangle - 2\langle \mathbf{x}, \mathbf{b} \rangle.$$

Demostración. Sean $\mathbf{x}$ y $\mathbf{v} \neq \mathbf{0}$ dos vectores fijos en $\mathbb{R}^n$ y $t \in \mathbb{R}$ un escalar variable, se tiene

$$\begin{aligned} g(\mathbf{x} + t\mathbf{v}) &= \langle \mathbf{x} + t\mathbf{v}, \mathbf{Ax} + t\mathbf{Av} \rangle - 2\langle \mathbf{x} + t\mathbf{v}, \mathbf{b} \rangle \\ &= \langle \mathbf{x}, \mathbf{Ax} \rangle + t\langle \mathbf{v}, \mathbf{Ax} \rangle + t\langle \mathbf{x}, \mathbf{Av} \rangle + t^2\langle \mathbf{v}, \mathbf{Av} \rangle - 2\langle \mathbf{x}, \mathbf{b} \rangle - 2t\langle \mathbf{v}, \mathbf{b} \rangle \\ &= \underbrace{\langle \mathbf{x}, \mathbf{Ax} \rangle - 2\langle \mathbf{x}, \mathbf{b} \rangle}_{\mathbf{g(x)}} + \underbrace{2t\langle \mathbf{v}, \mathbf{Ax} \rangle - 2t\langle \mathbf{v}, \mathbf{b} \rangle}_{2t\langle \mathbf{v}, \mathbf{Ax} - \mathbf{b} \rangle} + t^2\langle \mathbf{v}, \mathbf{Av} \rangle. \end{aligned}$$

Como se tiene que $\mathbf{x}$ e $\mathbf{v}$ son fijos, luego se puede definir una función $h(t)$ como

$$h(t) = g(\mathbf{x} + t\mathbf{v}).$$

Se conoce del cálculo diferencial, que la minimización de la función h ocurre cuando es diferenciable y se busca $h'(t) = 0$. Entonces se tiene que

$$h'(t) = 2\langle \mathbf{v}, \mathbf{Ax} - \mathbf{b} \rangle + 2t_{min}\langle \mathbf{v}, \mathbf{Av} \rangle = 0,$$

el mínimo t_{min} ocurre entonces cuando

$$t_{min} = \frac{\langle \mathbf{v}, \mathbf{b} - \mathbf{Ax} \rangle}{\langle \mathbf{v}, \mathbf{Av} \rangle},$$

con lo que se obtiene entonces

$$\begin{aligned} g(\mathbf{x} + t_{min}\mathbf{v}) &= g(\mathbf{x}) - 2\frac{\langle \mathbf{v}, \mathbf{b} - \mathbf{Ax} \rangle}{\langle \mathbf{v}, \mathbf{Av} \rangle}\langle \mathbf{v}, \mathbf{b} - \mathbf{Ax} \rangle + \left(\frac{\langle \mathbf{v}, \mathbf{b} - \mathbf{Ax} \rangle}{\langle \mathbf{v}, \mathbf{Av} \rangle} \right)^2 \langle \mathbf{v}, \mathbf{Av} \rangle \\ &= g(\mathbf{x}) - \frac{(\langle \mathbf{v}, \mathbf{b} - \mathbf{Ax} \rangle)^2}{\langle \mathbf{v}, \mathbf{Av} \rangle} \end{aligned}$$

De esta última ecuación se tiene que para cualquier $\mathbf{v} \neq \mathbf{0}$ se cumple $g(\mathbf{x} + t_{min}\mathbf{v}) < g(\mathbf{x})$ a menos que $\langle \mathbf{v}, \mathbf{b} - \mathbf{Ax} \rangle = 0$, pues es este caso la inecuación se transforma a igualdad.

Luego si $\mathbf{x}^*$ es un vector que satisface $\mathbf{Ax}^* = \mathbf{b}$ entonces $\langle \mathbf{v}, \mathbf{b} - \mathbf{Ax}^* \rangle = 0$ independientemente del valor del vector $\mathbf{v}$. Luego dado el cuadrado del numerador (lo que garantiza un término positivo que siempre decrementa a $g(\mathbf{x})$), se ve que $g(\mathbf{x})$ no puede bajo ninguna circunstancia ser menor a $g(\mathbf{x}^*)$. Luego, $\mathbf{x}^*$ minimiza a la función g.

Ahora suponiendo que $\mathbf{x}^*$ minimiza a g, luego para todo vector $\mathbf{v}$ se cumple que $g(\mathbf{x}^* + t_{min}\mathbf{v}) > g(\mathbf{x}^*)$. Luego necesariamente $\langle \mathbf{v}, \mathbf{b} - \mathbf{Ax}^* \rangle = 0$, con lo cual $\mathbf{Ax}^* = \mathbf{b}$. □

La idea del método de *Gradientes Conjugados* entonces es proponer una solución inicial al sistema de ecuaciones (5.1) y un vector dirección $\mathbf{v}$ el cual oriente al moverse de la localidad de la solución actual para obtener la correcta. Definiendo el residuo del sistema lineal como $\mathbf{r} = \mathbf{A}\mathbf{x}^* - \mathbf{b}$ entonces el parámetro t de la Demostración (5.2.2) puede reescribirse como

$$t = \frac{\langle \mathbf{v}, \mathbf{r} \rangle}{\langle \mathbf{v}, \mathbf{A}\mathbf{v} \rangle}, \tag{5.2}$$

donde claramente se observa que si $\mathbf{r} \neq \mathbf{0}$ y $\langle \mathbf{v}, \mathbf{r} \rangle \neq \mathbf{0}$, luego $\mathbf{x} + t\mathbf{v}$ da un valor menor de la función g y en general estará mas cerca de la solución $\mathbf{x}^*$.

Existe un método conocido como el *método de máximo descenso*, que se basa actualizar el vector dirección $\mathbf{v}$ en cada paso de tiempo en sentido contrario al gradiente de la función g, esto es sea $\mathbf{x}^0$ una aproximación inicial al sistema (5.1), y sea $\mathbf{v}^1$ una dirección inicial no nula, luego la actualización tiene la forma

$$\mathbf{v}^{n+1} = \mathbf{r}^n = \mathbf{b} - \mathbf{A}\mathbf{x}^n. \tag{5.3}$$

El principal problema del método anterior es su convergencia lenta en problemas lineales, un método alternativo propone actualizar el campo $\mathbf{v}$ de direcciones teniendo en cuenta lo que se denomina una *A-ortogonalidad*, esto es que

$$\langle \mathbf{v}^i, \mathbf{A}\mathbf{v}^j \rangle = 0, \qquad si\ i \neq j. \tag{5.4}$$

luego los vectores asociados a esta *A-ortogonalidad* son linealmente independientes, por ende forman una base en $\mathbb{R}^n$. Con este esquema de selección de los vectores dirección $\mathbf{v}$ se tiene entonces que

$$t^n = \frac{\langle \mathbf{v}^n, \mathbf{r}^{n-1} \rangle}{\langle \mathbf{v}^n, \mathbf{A}\mathbf{v}^n \rangle}, \tag{5.5}$$

$$\mathbf{x}^n = \mathbf{x}^{n-1} + t^n \mathbf{v}^n. \tag{5.6}$$

Se tiene la particularidad que con la elección de los vectores dirección $\mathbf{v}$ de esta manera se logra la convergencia del método en a lo más n pasos de tiempo, y además se converge a la solución exacta si se utiliza precisión exacta.

Teorema 5.2.3. *Sea* $\mathbf{A}$ *una matriz definida positiva con un conjunto A-ortogonal de vectores* $\{\mathbf{v}^0, \mathbf{v}^1, ..., \mathbf{v}^n\}$ *y un vector* $\mathbf{x}^0$ *arbitrario, se tiene que*

$$t_n = \frac{\langle \mathbf{v}^n, \mathbf{b} - \mathbf{A}\mathbf{x}^{n-1} \rangle}{\langle \mathbf{v}^n, \mathbf{A}\mathbf{v}^n \rangle},$$
$$\mathbf{x}^n = \mathbf{x}^{n-1} + t_k \mathbf{v}^n,$$

para $n = 1, 2, ..., k$. *Finalmente suponiendo aritmética exacta* $\mathbf{A}\mathbf{x}^n = \mathbf{b}$.

La demostración de este hecho puede encontrarse en [BF00, Capítulo 7, sección 5].

El uso del conjunto *A-ortogonal* $\{\mathbf{v}^0, \mathbf{v}^1, ..., \mathbf{v}^n\}$ de vectores dirección da lugar a lo que se denomina un método de dirección conjugada, puesto que se tiene ortogonalidad entre los vectores residuales $\mathbf{r}^n$ y los vectores dirección $\mathbf{v}^k$.

Teorema 5.2.4. *Los vectores residuales* $\mathbf{r}^n$ *con* $n = 1, 2, ..., k$ *para un método de direcciones conjugadas satisfacen*

$$\langle \mathbf{r}^n, \mathbf{v}^j \rangle = 0, \qquad para\ cada\ j = 1, 2, ..., n.$$

La idea entonces es encontrar iterativamente vectores dirección $\mathbf{v}^n$ tal que produzcan residuos $\mathbf{r}^n$ mutuamente ortogonales.

Para el cálculo de los sucesivos $\{\mathbf{v}^0, \mathbf{v}^1, ..., \mathbf{v}^n\}$ y $\{\mathbf{x}^0, \mathbf{x}^1, ..., \mathbf{x}^n\}$ se procede inicialmente al cómputo de $\mathbf{r}^0$ vía el método del máximo descenso, es decir, $\mathbf{r}^0 = \mathbf{b} - \mathbf{A}\mathbf{x}^0$ como la primera dirección de búsqueda.

Procediendo, suponga que se tienen las direcciones conjugadas $\mathbf{v}^0, \mathbf{v}^1, ..., \mathbf{v}^{n-1}$ y las soluciones aproximadas $\mathbf{x}^0, \mathbf{x}^1, ..., \mathbf{x}^{n-1}$ con

$$\mathbf{x}^n = \mathbf{x}^{n-1} + t_{n-1}\mathbf{v}^{n-1},$$

cumpliendo las restricciones comentadas en Teoremas (5.2.3) y (5.2.4). Si se tiene que $\mathbf{x}^{n-1}$ no es la solución al sistema representado por la ecuación (5.1) entonces necesariamente $\mathbf{r}^{n-1} = \mathbf{b} - \mathbf{A}\mathbf{x}^{n-1} \neq \mathbf{0}$ y según teorema (5.2.4) $\langle \mathbf{r}^{n-1}, \mathbf{v}^j \rangle = 0$ para $j = 1, 2, ..., n-1$.

A diferencia del método de máximo descenso, ahora se busca una nueva dirección dada por

$$\mathbf{v}^n = \mathbf{r}^{n-1} + s_{n-1}\mathbf{v}^{n-1}, \tag{5.7}$$

con la restricción que la elección del escalar s_{n-1} produzca una nueva dirección que satisfaga la condición de A-ortogonalidad. Se derivan entonces las siguientes relaciones

$$\begin{aligned} \mathbf{A}\mathbf{v}^n &= \mathbf{A}\mathbf{r}^{n-1} + s_{n-1}\mathbf{A}\mathbf{v}^{n-1} \\ \langle \mathbf{v}^{n-1}, \mathbf{A}\mathbf{v}^n \rangle &= \langle \mathbf{v}^{n-1}, \mathbf{A}\mathbf{r}^{n-1} \rangle + s_{n-1}\langle \mathbf{v}^{n-1}, \mathbf{A}\mathbf{v}^{n-1} \rangle \end{aligned} \tag{5.8}$$

con lo cual, imponiendo la A-ortogonalidad en el miembro izquierdo de la última ecuación

$$s_{n-1} = -\frac{\langle \mathbf{v}^{n-1}, \mathbf{A}\mathbf{r}^{n-1} \rangle}{\langle \mathbf{v}^{n-1}, \mathbf{A}\mathbf{v}^{n-1} \rangle} \tag{5.9}$$

se obtiene el valor de s_{n-1} que se buscaba, que satisface la condición de $\langle \mathbf{v}^n, \mathbf{A}\mathbf{v}^i \rangle = 0$ para $i = 1, 2, ..., n-1$.

A continuación se requiere calcular el valor del escalar t_n como se ha presentado en el Teorema 1.2.2, entonces se tiene que

$$\begin{aligned} t_n &= \frac{\langle \mathbf{v}^n, \mathbf{r}^{n-1} \rangle}{\langle \mathbf{v}^n, \mathbf{A}\mathbf{v}^n \rangle} \\ &= \frac{\langle \mathbf{r}^{n-1} + s_{n-1}\mathbf{v}^{n-1}, \mathbf{r}^{n-1} \rangle}{\langle \mathbf{v}^n, \mathbf{A}\mathbf{v}^n \rangle} \\ &= \frac{\langle \mathbf{r}^{n-1}, \mathbf{r}^{n-1} \rangle}{\langle \mathbf{v}^n, \mathbf{A}\mathbf{v}^n \rangle} + s_{n-1}\frac{\langle \mathbf{v}^{n-1}, \mathbf{r}^{n-1} \rangle}{\langle \mathbf{v}^n, \mathbf{A}\mathbf{v}^n \rangle} \end{aligned} \tag{5.10}$$

y utilizando las restricciones del Teorema (5.2.4) se obtiene finalmente

$$t_n = \frac{\langle \mathbf{r}^{n-1}, \mathbf{r}^{n-1} \rangle}{\langle \mathbf{v}^n, \mathbf{A}\mathbf{v}^n \rangle}. \tag{5.11}$$

Luego, se tiene que la solución al paso de tiempo n se obtiene como

$$\mathbf{x}^n = \mathbf{x}^{n-1} + t_n\mathbf{v}^n. \tag{5.12}$$

Finalmente para el cálculo del $\mathbf{r}^n$ necesario para la próxima búsqueda de vector dirección se procede multiplicando ambos miembros de la ecuación (5.12) por la matriz $\mathbf{A}$ para posteriormente restarle el vector $\mathbf{b}$ obteniendo

$$\begin{aligned} \mathbf{x}^n &= \mathbf{x}^{n-1} + t_n\mathbf{v}^n \\ \mathbf{A}\mathbf{x}^n - \mathbf{b} &= \mathbf{A}\mathbf{x}^{n-1} - \mathbf{b} + t_n\mathbf{A}\mathbf{v}^n \\ \mathbf{r}^n &= \mathbf{r}^{n-1} + t_n\mathbf{A}\mathbf{v}^n \end{aligned} \tag{5.13}$$

tomando el producto interno con $\mathbf{r}^n$ en la ecuación (5.13) se obtiene

$$\langle \mathbf{r}^n, \mathbf{r}^n \rangle = \underbrace{\langle \mathbf{r}^{n-1}, \mathbf{r}^n \rangle}_{\text{cero}} + t_n\langle \mathbf{A}\mathbf{v}^n, \mathbf{r}^n \rangle = t_n\langle \mathbf{A}\mathbf{v}^n, \mathbf{r}^n \rangle \tag{5.14}$$

por la condición de exclusión mutua. Considerando ahora la ecuación (5.9), y teniendo en cuenta la ecuación (5.11) se obtiene

$$s_n = \frac{\langle \mathbf{r}^n, \mathbf{r}^n \rangle}{\langle \mathbf{r}^{n-1}, \mathbf{r}^{n-1} \rangle}. \tag{5.15}$$

Finalmente y para resumir, inicialmente se procede con el método de máximo descenso (hallando $\mathbf{r}^0$ y $\mathbf{v}^1$), luego en cada iteración se computan en orden las ecuaciones (5.11), (5.12), (5.13), (5.15) y (5.7).

5.3. Gradientes Conjugados precondicionado

Dada una matriz $\mathbf{A} \in \mathbb{R}^{n \times n}$ se define su número de condición $\kappa(\mathbf{A}) \in \mathbb{R}$ como

$$\kappa(\mathbf{A}) = \lambda_{max}/\lambda_{min}, \tag{5.16}$$

donde λ_{max} y λ_{min} representan el máximo y el mínimo autovalor de la matriz $\mathbf{A}$ respectivamente. Este escalar adimensional es de vital importancia pues esta directamente relacionado con el condicionamiento de la matriz.

El problema principal del método de Gradientes Conjugados recientemente introducido es que para matrices mal condicionadas ($\kappa(\mathbf{A}) >> 1$) presenta sensibilidad a errores de redondeos, lo que provoca que a los n pasos de tiempo la solución exacta no pueda ser alcanzada (generalmente).

Para un sistema bien condicionado, Gradientes Conjugados como método iterativo converge a una solución lo suficientemente buena en cerca de $\sqrt{n}$ pasos.

El proceso de condicionar un sistema comienza proponiendo una matriz de condicionamiento no singular $\mathbf{P}$ tal que

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{P}^{-1}\mathbf{A}\mathbf{x} &= \mathbf{P}^{-1}\mathbf{b} \\ \mathbf{P}^{-1}\mathbf{A}(\mathbf{P}\mathbf{P}^{-1})^T\mathbf{x} &= \mathbf{P}^{-1}\mathbf{b} \\ (\mathbf{P}^{-1}\mathbf{A}(\mathbf{P}^{-1})^T)(\mathbf{P}^T\mathbf{x}) &= \mathbf{P}^{-1}\mathbf{b}, \end{aligned} \tag{5.17}$$

para obtener

$$\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}, \tag{5.18}$$

donde

$$\tilde{\mathbf{A}} = \mathbf{P}^{-1}\mathbf{A}(\mathbf{P}^{-1})^T, \tag{5.19}$$

$$\tilde{\mathbf{x}} = \mathbf{P}^T\mathbf{x} \;\; y \tag{5.20}$$

$$\tilde{\mathbf{b}} = \mathbf{P}^{-1}\mathbf{b}. \tag{5.21}$$

Finalmente el sistema gobernado por la ecuación (5.18) se resuelve para $\tilde{\mathbf{x}}$ y luego se obtiene $\mathbf{x}$ premultiplicando la ecuación (5.20) por $(\mathbf{P}^{-1})^T$.

Se obtendrán entonces las nuevas ecuaciones incorporando implícitamente la matriz de condicionamiento. Para ello en primera instancia considerando las ecuaciones (5.19 - 5.21) se define el residuo como

$$\tilde{\mathbf{r}}^n = \tilde{\mathbf{b}} - \tilde{\mathbf{A}}\tilde{\mathbf{x}}^n = \mathbf{P}^{-1}\mathbf{r}^n, \tag{5.22}$$

luego se define el escalar s_n como

$$\tilde{\mathbf{s}}_n = \frac{\langle \tilde{\mathbf{r}}^n, \tilde{\mathbf{r}}^n \rangle}{\langle \tilde{\mathbf{r}}^{n-1}, \tilde{\mathbf{r}}^{n-1} \rangle} = \frac{\langle \mathbf{P}^{-1}\mathbf{r}^n, \mathbf{P}^{-1}\mathbf{r}^n \rangle}{\langle \mathbf{P}^{-1}\mathbf{r}^{n-1}, \mathbf{P}^{-1}\mathbf{r}^{n-1} \rangle}. \tag{5.23}$$

Si se define $\tilde{\mathbf{v}}_n = \mathbf{P}^T\mathbf{v}^n$ entonces $\tilde{t}_n$ resulta ser

$$\begin{aligned}\tilde{\mathbf{t}}_n = \frac{\langle \tilde{\mathbf{r}}^{n-1}, \tilde{\mathbf{r}}^{n-1} \rangle}{\langle \tilde{\mathbf{v}}^n, \tilde{\mathbf{A}}\tilde{\mathbf{v}}^n \rangle} &= \frac{\langle \mathbf{P}^{-1}\mathbf{r}^{n-1}, \mathbf{P}^{-1}\mathbf{r}^{n-1} \rangle}{\langle \mathbf{P}^T\mathbf{v}^n, \mathbf{P}^{-1}\mathbf{A}(\mathbf{P}^{-1})^T\mathbf{P}^T\mathbf{v}^n \rangle}, \\ &= \frac{\langle \mathbf{P}^{-1}\mathbf{r}^{n-1}, \mathbf{P}^{-1}\mathbf{r}^{n-1} \rangle}{\langle \mathbf{P}^T\mathbf{v}^n, \mathbf{P}^{-1}\mathbf{A}\mathbf{v}^n \rangle}, \\ &= \frac{\langle \mathbf{P}^{-1}\mathbf{r}^{n-1}, \mathbf{P}^{-1}\mathbf{r}^{n-1} \rangle}{\langle \mathbf{v}^n, \mathbf{A}\mathbf{v}^n \rangle}\end{aligned} \tag{5.24}$$

la última ecuación es cierta pues si se descompone el denominador quedando sólo los términos en P resulta $\langle \mathbf{P}^T, \mathbf{P}^{-1} \rangle = (\mathbf{P}^{-1})^T\mathbf{P}^T = (\mathbf{P}\mathbf{P}^{-1})^T = \mathbf{I}$ donde $\mathbf{I}$ es la identidad.

Continuando con el desarrollo se tiene que

$$\tilde{\mathbf{x}}^n = \tilde{\mathbf{x}}^{n-1} + \tilde{t}_n\tilde{\mathbf{v}}^n \Longleftrightarrow \mathbf{x}^n = \mathbf{x}^{n-1} + \tilde{t}_n\mathbf{v}^n, \tag{5.25}$$

y

$$\tilde{\mathbf{r}}^n = \tilde{\mathbf{r}}^{n-1} + \tilde{t}_n\tilde{\mathbf{A}}\tilde{\mathbf{v}}^n \Longleftrightarrow \mathbf{r}^n = \mathbf{r}^{n-1} + \tilde{t}_n\mathbf{A}\mathbf{v}^n. \tag{5.26}$$

Finalmente falta definir el vector dirección de la iteración actual, entonces

$$\tilde{\mathbf{v}}^{n+1} = \tilde{\mathbf{r}}^n + \tilde{s}_n\tilde{\mathbf{v}}^n \Longleftrightarrow \mathbf{v}^{n+1} = (\mathbf{P}^{-1})^T\mathbf{P}^{-1}\mathbf{r}^{n-1} + \tilde{s}_n\mathbf{v}^n. \tag{5.27}$$

Cuando se utiliza la matriz $\mathbf{P}$ de precondicionamiento, en general esta presenta una forma similar a la matriz $\mathbf{L}$ obtenida en la factorización de Cholesky. En la práctica, aquellos valores de $\mathbf{A}$ próximos a cero se deprecian (pues son ellos principalmente quienes introducen errores de redondeo) y se aplica Cholesky para obtener una factorización incompleta. Luego la matriz resultante puede actuar como precondicionador.

5.4. Precondicionamiento FFT (transformada rápida de Fourier)

Una vez introducidos los conceptos básicos del método iterativo *Gradientes Conjugados Precondicionado*, en esta sección se tratarán aquellos que derivan de precondicionar al sistema obtenido de la discretización de la ecuación de Poisson para la presión en el segundo paso de *Fractional-Step* utilizando para ello una *Transformada Rápida de Fourier* (de ahora en más FFT), dando lugar al método conocido como *Resolvedor rápido de Poisson.*

Recordando se tiene

$$\underbrace{\mathbf{P}^{-1}\mathbf{A}}_{*}\mathbf{x} = \mathbf{P}^{-1}\mathbf{b}, \tag{5.28}$$

en donde $\mathbf{P}$ hace referencia a la matriz de precondicionamiento, lo ideal sería que el término sobre la etiqueta (*) resultara en la identidad, o bien diagonal, con el objetivo de tener una inversa trivial. Considerando el problema de Poisson para la presión requerido por la segunda etapa de F-S en donde la discretización sobre grillas homogéneas del operador lineal Laplaciano (forzando la propiedad de positivo definido mediante el signo introducido) determina una matriz simétrica $\mathbf{A}$ de la forma

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \dots & 0 & 0 & -1 \\ -1 & 2 & -1 & \dots & 0 & 0 & 0 \\ 0 & -1 & 2 & \dots & 0 & 0 & 0 \\ 0 & 0 & -1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -1 & 2 & -1 \\ -1 & 0 & 0 & \dots & 0 & -1 & 2 \end{bmatrix}. \tag{5.29}$$

se tiene entonces que es deseable encontrar una matriz de precondicionamiento que, actuando sobre la matriz Laplaciana, reproduzca una matriz diagonal tal y como se ha introducido anteriormente. Considere la transformada discreta de Fourier unidimensional definida como

$$X_k = \sum_{j=1}^{N} x_j e^{i\frac{2\pi kj}{N}}, \tag{5.30}$$

donde X_k es la componente frecuencial k-ésima, x_j es el valor del nodo j, y la exponencial es la base de la transformada discreta de Fourier (periódica, de período N). Sea la matriz $\mathbf{F}$ el resultado de evaluar dicha base exponencial, se puede demostrar que sus columnas representan los autovectores de la matriz $\mathbf{A}$. De esta forma [Loa92, Sección 4.5.3], se tiene

$$\mathbf{D} = \mathbf{F}^{-1}\mathbf{A}\mathbf{F} \tag{5.31}$$

donde $\mathbf{D}$ es una matriz diagonal cuyas componentes resultan los N autovalores de $\mathbf{A}$. Considerando que $\mathbf{A} = \mathbf{F}\mathbf{D}\mathbf{F}^{-1}$ luego se obtiene finalmente

$$\mathbf{P}^{-1} = \mathbf{A}^{-1} = \mathbf{F}\mathbf{D}^{-1}\mathbf{F}^{-1}, \tag{5.32}$$

con lo cual

$$\mathbf{x} = \mathbf{F}\mathbf{D}^{-1}\mathbf{F}^{-1}\mathbf{b}, \tag{5.33}$$

de esta forma el CG converge en una iteración aplicando sólo tres cálculos, dos transformadas de Fourier (una hacia delante y otra inversa) con $\mathcal{O}(Nlog_2(N))$ operaciones para cada transformación, y una multiplicación de N elementos con $\mathcal{O}(N)$ operaciones. Esta precondición puede extender a otras condiciones de borde, como ser, las fijas, o bien en grillas no homogéneas. La extensión 2D y 3D sigue el mismo camino.

Cuando se tienen cuerpos embebidos en el dominio la solución del precondicionador es tan buena que el CG converge en pocas iteraciones, es por ello que anteriormente se ha introducido dicho método en conjunto con la precondición. Si bien en este PFC no se tratan casos de estudio en esta temática, un análisis profundo puede ser encontrado en [MASC10].

Capítulo 6

Introducción a CUDA

Una vez introducido matemáticamente el modelo discreto y su resolución, esta sección abordará los detalles de la arquitectura sobre la cual se han realizado las implementaciones, sin más la arquitectura CUDA.

6.1. Introducción

En esencia la arquitectura CUDA fue introducida al mercado a inicios de 2007 por la empresa NVIDIA y básicamente se ideó con el objeto de facilitar la programación de componentes GPU (graphics processing units). Dado que la GPU fue concebida para un cálculo intensivo (principalmente dado a que libera al CPU de la tarea de la renderización de los objetos en la pantalla) es que al ver la creciente curva de rendimiento obtenida a lo largo de estos últimos años, se comenzó a utilizar la misma para cálculos de propósito general.

Internamente una GPU implementa un enorme número de unidades aritmético lógicas (UAL) que son las que realizan las operaciones elementales, mientras que presenta un escaso número de unidades de control de flujo, lo contrario sucede en una CPU. Como resultado, la GPU presenta más características de procesamiento masivo que una CPU.

6.2. Elementos de una arquitectura CUDA

En la jerga GPU una GPU/GPGPU (general purpose graphics processing units) se identifica como un device, mientras que el hardware que la soporta como un host. Un host puede soportar varios devices (en principio tantos como puertos PCI-Express se tengan). Cada GPU implementa básicamente 4 tipos de componentes: los *streaming multiprocesors* (SM), las *unidades especiales* (SFU), las de *doble precisión* y el *caché*. Sin entrar en detalles los SM presentan un conjunto de 8 procesadores escalares denominados SP (scalar processors) capaces de procesar evaluandos de precisión simple. Las unidades SFU implementan esquemas de evaluación optimizados (a veces por hardware, otras por software) para funciones trascendentales y recíprocos (entre otros). Las unidades de doble precisión realizan operaciones con evaluandos de doble precisión (siguiendo los estándares del IEEE 754). Finalmente el caché realiza operaciones análogas a aquel presente en un CPU, en pocas palabras, preserva un conjunto de datos e instrucciones (siguiendo una cierta lógica) cuya lectura presenta un rendimiento mayor a aquella obtenida al efectuarse sobre otro tipo de memoria.

6.3. Modelo de organización y ejecución

Una función que se ejecuta en un device se denomina *kernel*. Para su procesamiento CUDA conforma lo que se denomina grilla (o *grid*). La grilla presenta en cada nodo un bloque de threads (o *threadblock*). Un thread se define como la mínima unidad de ejecución posible. Un threadblock entonces es un conjunto de threads.

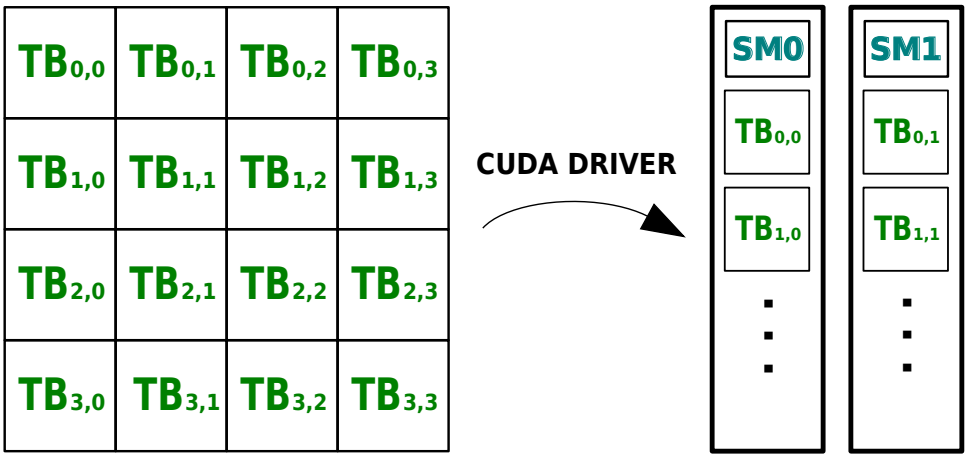


Figura 6.1: Descomposición del dominio de resolución en threadblocks (TB) y posterior asignación a los SM. En este ejemplo se tienen sólo 2 SMs, y la distribución de los threadblock se hace sobre estos. Es de interés mencionar que el orden no esta definido, y el presentado para esta imagen tiene sólo fines prácticos.

El fundamento por detrás de estas subdivisiones se encuentra de la mano de como la arquitectura distribuye estos threadblocks sobre los SM para que sean procesados. Si bien el orden de reparto es desconocido, un ejemplo de ello se puede observar en la Figura (6.1).

Existe una (al menos en Tesla[1] -arquitectura GPU utilizada en este PFC- y anteriores) unidad que se encarga de la organización de los threadblocks dentro de cada SM, denominada unidad SIMT. Ella es la encargada de mantener, siempre que se pueda, a los SP activos. Para ello cada threadblock es subdividido en *warps*, un conjunto de 32 threads. Cada thread esta identificado por un número unidimensional que lo diferencia de los restantes, es más, los threads dentro de un mismo warp presentan identificadores sucesivos y crecientes. La Figura (6.2) presenta la subdivisión de un threadblock en warps.

Así la unidad SIMT desglosa estos threadblocks y distribuye warps sobre los SP. La cantidad de warps activos por SM es dependiente del *Compute Capability* del dispositivo, para la Tesla (con Compute Capability 1.3) se tiene que se puede mantener 32 warps activos por SM. A su vez la unidad administra múltiples colas de warps, por ejemplo, se tienen colas de warps en espera, listos para ejecución, entre otras. El objetivo de tener dicha cantidad de warps activos es que cuando los threads de un determinado halfwarp (conjunto de 16 threads, división de un warp, es la unidad de ejecución por SP) realizan operaciones de alta latencia (como ser la lectura/escritura sobre memoria local) luego dicho warp es enviado a la cola de espera y un nuevo warp toma el hardware para ser procesado. De esta forma el hardware evita la desocupación de sus SM. Lo anterior se conoce como *zero overheading schedule*.

Si bien el intercambio de procesos en CPU es una tarea costosa, dado a que requiere almacenar el estado del programa para que en el próximo quantum de tiempo de procesador que le sea asignado pueda conocer el punto desde donde debe continuar (pila de instrucciones) y los datos que tenia disponible (así también los recursos a los que poseía acceso); contrariamente en una GPU los recursos son asignados por thread, por lo cual no hay intercambio de estados entre ellos. Así el intercambio de threads se hace sin una carga adicional de rendimiento. El resultado entonces es una unidad que mediante el intercambio masivo de threads puede reducir notablemente la latencia de operaciones, en comparación a otras arquitecturas (p.e. la PC).

El modelo de ejecución de la arquitectura CUDA se basa en lo que se conoce como SIMT *single instruction multiple thread*), es decir, la idea es que todos los threads dentro de un mismo halfwarp ejecuten la misma instrucción cada uno a sus respectivos datos, y de esta forma se amortiza la búsqueda de la instrucción en memoria.

[1] http://www.nvidia.com/object/tesla_computing_solutions.html

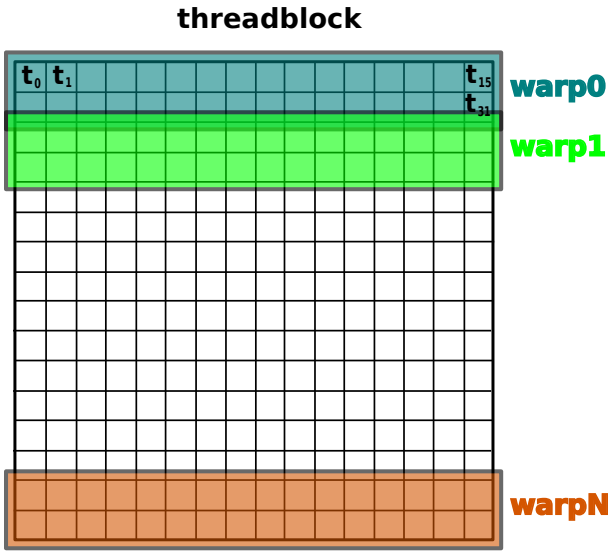


Figura 6.2: Subdivisión de un threadblock en warps. Observar que cada thread se identifica en base a dos índices, tx y ty, que lo identifican en el grupo bidimensional (aunque el threadblock puede ser 1D o 3D también). Luego la idea de IDs contiguos y ascendentes dentro de un warp viene dado por un mapeo 1D utilizando estas dos coordenadas.

6.4. Tipos de memorias

La arquitectura CUDA presenta actualmente 5 tipos distintos de memorias, en las secciones que siguen se introducirán los conceptos básicos que rigen el funcionamiento de cada una.

6.4.1. Memoria global y accesos fusionados

La memoria global (global memory) es una vasta área de memoria que permite tanto lectura como escritura, con una latencia media de 400-600 ciclos de reloj. Si bien no presenta accesos vía caché (al menos no en la tecnología Tesla), su gran capacidad la hace fundamental cuando se requiere de gran cantidad de datos en la GPU. La unidad Tesla C1060 que se dispone presenta un total de 4 Gbytes de memoria global.

Para reducir su gran latencia, se implementan lecturas/escrituras optimizadas en caso de cumplir ciertos requisitos, denominadas *coalesce readings/writings* o lecturas/escrituras fusionadas.

Suponiendo que se tiene un determinado warp en ejecución asociado a una operación de lectura o escritura, entonces se tienen diferentes escenarios donde el acceso fusionado puede ocurrir dependiendo del valor de *Compute Capability* que se disponga. Para el caso de la Tesla C1060, una acceso fusionado tendrá lugar cuando todos los threads de un mismo halfwarp accedan a posiciones de memoria global que se encuentren dentro un bloque alineado a 32-64-128 bytes (de acuerdo a si el dato a leer presenta 1,2-4-8 bytes). Además, no importa si dos o más threads dentro del mismo halfwarp requieren del mismo dato, o bien si existen posiciones de memoria que no se utilizan (en este caso se estaría desperdiciando la capacidad de lectura).

El acceso fusionado es fundamental en cualquier desarrollo en CUDA, puesto que de utilizarse se estaría aprovechando la capacidad máxima de lectura otorgada por este tipo de memoria. Una discusión más extensa se accesos fusionados puede encontrarse en [KmWH10, SK10].

6.4.2. Memoria constante

La memoria constante (constant memory) es una memoria (de 64 Kbytes para la Tesla) que presenta caché (de 8 Kbytes para la Tesla) y tiene la particularidad que sólo permite lecturas. Presenta accesos optimizados de acuerdo a como se efectúen las lecturas sobre esta.

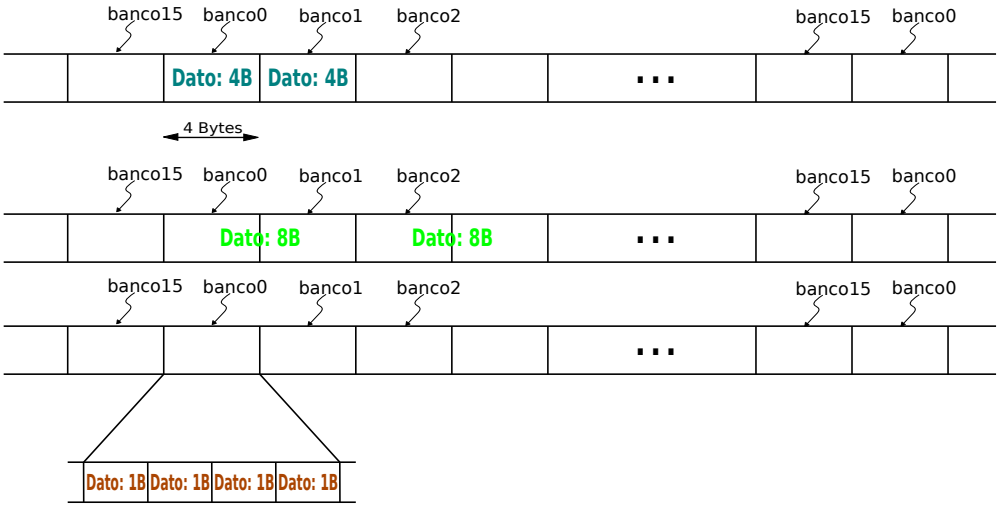


Figura 6.3: Estructura de la memoria compartida y problemas con conflictos de bancos.

En particular si todos los threads dentro de un mismo halfwarp acceden a la misma posición de memoria, luego sólo una lectura es realizada (con lo cual en principio se evitarían 15 lecturas extra) y es replicada vía broadcast a todos los threads de ese halfwarp. Además, en caso de que en el próximo halfwarp se requiera de la misma posición de memoria, el acceso se realizaría sobre el caché. Como resultado, estos accesos resultan (en general) casi tan rápido como el acceso a un registro.

Sin embargo si los threads dentro de un halfwarp requieran de accesos a distintas porciones de memoria, los accesos reducirían notablemente su rendimiento, puesto que el broadcast ya no se realizaría a nivel de los 16 threads, sino al número de threads que accedan a la misma posición. El peor caso es cuando todos los threads acceden a posiciones distintas con lo cual el rendimiento se verá seriamente afectado, pudiendo resultar en un acceso con menor rendimiento que uno sobre memoria global (recordando que sobre este último se tiene la capacidad del acceso fusionado).

6.4.3. Memoria compartida

Puede ser considerada como la memoria de mayor importancia dentro de la arquitectura CUDA. Su profundo entendimiento y posterior utilización puede resultar en enormes incrementos de rendimiento en comparación al uso de la memoria global. En esencia la memoria compartida (shared memory) es una memoria de alto rendimiento tanto para lecturas como para escrituras, con visibilidad para todos los threads de un mismo threadblock. La latencia que tienen operaciones sobre ellas resultan 100-200 veces menores a las mismas realizadas sobre la memoria global. En las Tesla se disponen de 16 Kbytes de memoria disponible por SM.

Se utiliza generalmente cuando cada thread dentro de un mismo threadblock requiere de valores que pueden ser cargados por otro thread, así cada uno carga su propio conjunto de datos y lo almacena en memoria compartida, para que luego no haya accesos redundantes a memoria global por cada thread.

6.4.3.1. Bancos de memoria y su organización

La organización de la memoria compartida se desprende del hecho de que esta conformada por palabras de 4 bytes. Se tiene además una subdivisión de 16 bancos de memoria, donde palabras sucesivas pertenecen a bancos sucesivos. Es importante notar que la cantidad de bancos coincide con la cantidad de threads en un halfwarp, puesto que están íntimamente ligados como se mostrará a continuación.

En la Figura (6.3) se muestra como bloque unidimensional la memoria compartida, y la asignación de cada palabra a un y sólo un banco de memoria. En especial muestra lo que se denomina conflicto de bancos (o *bank conflicts*), situación a darse en caso de ocurrir alguno de los siguientes escenarios:

- Threads dentro de un mismo halfwarp acceden a las mismas palabras, en este caso se producen accesos serializados al banco.
- Tipos de datos con alineamiento distinto a 4 bytes pueden producir que o bien un mismo dato ocupe palabras de distintos bancos (como ser el caso de alineamientos del tipo de datos de doble precisión), o bien varios datos se encuentren almacenados en la misma palabra (el caso del tipo de datos caracter). Como resultado en el primer caso se tendrán conflictos de 2 bancos (puesto que cada thread necesitará leer además de una palabra en su propio banco, otra del banco contiguo), mientras que en el segundo caso se tendrán conflictos de 4 bancos (dado que cada thread necesitará un dato de la misma palabra que otros 3 threads más).

Como resultado ocurrirán accesos serializados cuya principal desventaja es la reducción de la eficiencia de la lectura/escritura sobre la memoria compartida.

6.4.4. Memoria local

La memoria local (local memory) es una memoria dedicada por thread y se utiliza para almacenar datos privados. Los datos que allí residen presentan visibilidad sólo al thread que pertenecen (no así la memoria global, la constante y la compartida -sólo para threads en un threadblock-). En esencia en una porción de memoria global que actúa como soporte a la memoria de registros. Cuando uno define variables en un kernel de CUDA estas serán depositadas en memoria local cuando: sean arreglos, existan demasiados registros utilizados o bien la estructura completa pueda utilizar demasiada cantidad de espacio de registros.

6.4.5. Memoria de texturas

La memoria de texturas (texture memory) es una memoria con visibilidad para todos los threads sin importar a que threadblock pertenezcan. Es de sólo lectura y cacheada. La disposición de los datos dentro del hardware es 1D o 2D, con lo cual si la naturaleza del problema presenta cierta localidad en la utilización de los datos (p.e. un stencil de diferencias finitas en mallas estructuradas) luego el acceso a esos datos se realiza con lecturas muy eficientes.

6.4.6. Registros

Memoria dedicada por thread de rápido acceso. La GPGPU Tesla C1060 dispone de 16384 registros por SM, lo cual permite concluir que en promedio cada thread dispone de $16384/(32 \times 32) = 16$ registros, considerando que la GPGPU en cuestión puede disponer de 32 warps activos por SM, cada uno con 32 threads.

6.5. Divergencia de threads en un warp

Como se ha presentado anteriormente la arquitectura CUDA se basa en el esquema SIMT, y como punto crítico se tiene que, en el mejor de los casos, todos los threads de un halfwarp *deberían* ejecutar la misma instrucción para el conjunto de datos que tienen cargados. Esto en la práctica no resulta generalmente así puesto que los threads de un mismo halfwarp pueden seguir caminos distintos en condicionales (p.e. IF-THEN), el resultado son varias cargas de instrucciones (tantas como caminos condicionales se tengan) y la serialización de los caminos de ejecución. Entonces, suponiendo que se tienen dos caminos de ejecución y que de los 16 threads de un halfwarp los primeros 8 siguen un camino, mientras que los restantes 8 siguen el otro; así en primer instancia se atenderán a los primeros, y una vez finalizada su operación se procederán con los segundos.

6.6. Recursos de la GPU: consideraciones

Como se ha comentado anteriormente, los recursos de una GPGPU pueden listarse básicamente como la cantidad de registros y de memoria compartida que dispone. La cantidad de threads que pueda ejecutar

en un determinado tiempo esta en completa relación con la cantidad de recursos que se estén utilizando. Inicialmente la unidad SIMT desglosa los threadblocks en warps que son asignados a un determinado SP para su ejecución, así la cantidad total de warps que puede tener asignado un SM esta determinado por la cantidad de recursos que consuma un solo warp. Luego como se conoce que a mayor cantidad de warps activos mayor posibilidad de esconder la latencia de operaciones lentas, es que se recomienda disponer de la mayor cantidad de warps activos por SM.

En general la cantidad máxima esta determinada por el hardware que se dispone, en especial por el computeCapability del mismo, que para el caso de la Tesla es 1.3, y finalmente puede contener hasta 32 warps activos por SM (es decir, 1024 threads).

Lo mismo ocurre con la cantidad de threadblocks asociados por SM, para un computeCapability de 1.3 el máximo es de 8 siempre considerando el límite actual para la tecnología Tesla de a lo más 512 threads por threadblock. Además una consideración merece ser introducida cuando la cantidad de recursos resulta utilizada con abuso. Como los recursos son asociados a threads individuales dentro de un mismo threadblock, luego sólo se tendrán disponibles para su ejecución una cantidad determinada de threadblocks cuya suma de registros sea menor a la cantidad total disponible, siempre considerando threadblocks completos. Es por esto que se dice que las reducciones en la cantidad de threadblocks esta dada en múltiplos del tamaño del mismo, hasta que finalmente la cantidad de recursos utilizados por los threadblocks no exceda la cantidad disponible.

Capítulo 7

Detalles de la implementación

Se han adquirido a lo largo de los capítulos anteriores las herramientas que sirven de base para el desarrollo del tema principal de este PFC. Con las formulaciones discretas presentadas y la arquitectura CUDA introducida, esta sección abordará todo lo relacionado con el desarrollo.

7.1. Introducción

Como bien fue argumentado en el Capítulo (6) los objetivos principales a tener en cuenta para un desarrollo eficiente de un kernel en CUDA, y teniendo en cuenta las características del problema planteado, son:

1- Accesos fusionados,

2- utilización de memoria compartida y

3- minimización de la cantidad de registros por thread.

Si bien existen muchas otras ventajas de la arquitectura CUDA, estas resultan ser las que se pueden utilizar en el planteo propuesto.

7.2. Las estructuras de datos y las condiciones de borde

Es sabido que los campos incógnitas en el problema estudiado son dos: un escalar *presión*, y un vector *velocidad*. Una vez introducido el concepto de grillas staggered y el problema de desacople para nodos de presión, es directo notar que los cálculos en 3D tendrán que efectuarse sobre 4 grillas independientes (en principio). Luego una formulación sencilla de estructura de datos consistiría en disponer de dos grillas, una para presión y otra velocidad.

Si se considerara a M, N y P como las dimensiones del dominio computacional según direcciones x, y y z, una estructura de datos para la presión consistirá en una arreglo unidimensional con $M \times N \times P$ valores, cada uno haciendo referencia a un valor de presión en una determinada coordenada (i, j, k). La segunda estructura consistirá en otro arreglo unidimensional con con $M \times N \times P$ valores donde cada valor es una subestructura con componentes u, v y w, o componentes del campo velocidad según direcciones x, y y z.

A su vez los valores de las grillas se almacenarán según z, y y x, es decir, las primeras P posiciones de ambas estructuras de datos contienen los valores de los nodos $(0, 0, k)$ con $k = 0, 1, \cdots, P-1$, los siguientes P valores son aquellos con coordenadas $(0, 1, k)$ con $k = 0, 1, \cdots, P-1$, y así sucesivamente hasta los primeros $N \times P$ valores. Luego se tiene $(1, 0, k)$ con $k = 0, 1, \cdots, P-1$, posteriormente $(1, 1, k)$ con $k = 0, 1, \cdots, P-1$, hasta que se carguen los respectivos $M \times N \times P$ valores.

Finalmente con este tipo de estructura de datos uno esperaría los tipos de alineamientos de lectura y escritura que se presentan en el Cuadro (7.1).

Entonces se observa que para cualquier operación en la grilla de presión, esta será utilizada al máximo y no existirán pérdidas de ancho de banda por no utilizar la totalidad el alineamiento correspondiente.

	Presión		Velocidad	
	8x8	16x16	8x8	16x16
	[Bytes]	[Bytes]	[Bytes]	[Bytes]
Simple	32 (1 - 32)	64 (1 - 64)	96 (2 - 64+32)	192 (2 - 128+64)
Doble	64 (1 - 64)	128 (1 - 128)	192 (2 - 128+64)	384 (3 - 128)

Cuadro 7.1: Alineamientos realizados por la arquitectura para la lectura/escritura de datos a la memoria global. Entre paréntesis la cantidad de lecturas/escrituras realizadas y el alineamiento de la misma.

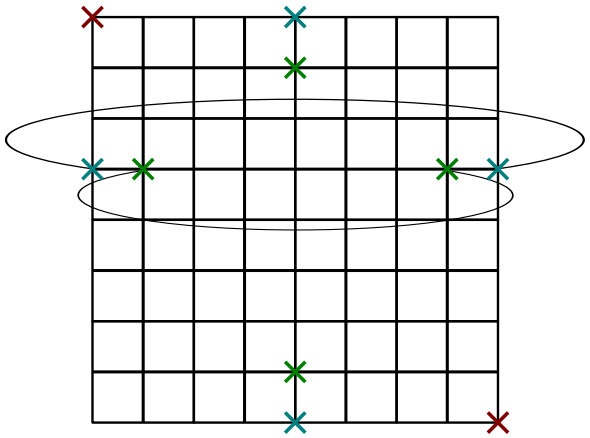


Figura 7.1: Topología de los nodos requeridos por aquellos threads en frontera.

Mientras que para el caso de la velocidad existirá un factor de 25 % de pérdida de rendimiento para alineamientos de 32 Bytes y de 6.25 % con 64 y 128 Bytes para el tipo de datos a simple precisión. Cuando se tienen datos en doble precisión estos porcentajes resultan en 25 % para 32 y 64 Bytes y de 6.25 % con 128 Bytes. Por lo que este tipo de estructuras de datos en principio deberían de ser evitados, luego una mejor selección consistiría en tener las 4 grillas por separado de esta forma se utilizaría, teóricamente, al máximo las capacidades de acceso la memoria global.

En la práctica el disponer de 4 grillas por separado no ha rendido como se esperaba, mientras que multiplicaba el código innecesariamente. Finalmente se optó por la resolución inicial, considerando sólo dos grillas, una de presión y otras para las velocidades.

La utilización de grillas staggered requiere, por definición, de una reducción de un nodo por dirección sobre uno de los campos (presión o velocidad) dado a que sus nodos estarán centrados en la celda conformada por 4 nodos en 2D, o por 8 en 3D (velocidad o presión). El disponer de la misma cantidad de nodos para ambos campos evita bifurcaciones de predicados (condicionales IF-THEN-ELSE) pero, sin los recaudos necesarios, sólo funciona para condiciones de borde periódicas. De otra forma, es necesario determinar cual de los campos será el que tenga una capa de nodos menos, puesto que las condiciones de borde depende de ello. El cálculo de los flujos es independiente de como se tomen los datos, aunque cada condición de borde deberá de cargar los mismos de manera consistente. El caso de estudio (y por tanto, la carga de datos) que se propone en este PFC utiliza condiciones de borde periódicas, de esta forma y sin falta de generalidad, lo que sigue del texto trata con este tipo de condiciones.

El patrón de lectura de nodos frontera se observa en la Figura (7.1), los nodos vértices (cruces en rojo) acceden a aquellos opuestos en diagonal, mientras que el resto (verde y azul) en su misma dirección, bien en los primeros o últimos. Cabe aclarar que en esa figura se muestran threads particulares ubicados en el borde, y con el mismo color de cruz aquel nodo que necesariamente debe cargar para la confección de su stencil. La lectura de los nodos será introducido en secciones siguientes.

7.3. Metodología de resolución

Considerando que se dispone de todo aquello anterior al lazo temporal, se presenta en el pseudocódigo (7.1) la metodología propuesta según Pasos Fraccionados.

Código 7.1: Lazo temporal y las etapas requeridas por la integración temporal de Pasos Fraccionados.

```
for(int n=0;n<nsteps;n++){
  \\1- Resolucion de las ecuaciones de cantidad de movimiento. u^n → R(u^n)
  \\2- Integracion temporal.
    \\Iteracion 0, FE. u^n + Δt R(u^n) → u*
    \\Iteracion 1 a nsteps-1, AB. u^n + (Δt/2)[3R(u^n) - R(u^{n-1})] → u*
  \\3- Calculo de divergencia de velocidad. ∇·u*
  \\4- CG+FFT. -Δp^{n+1} = -(ρ/Δt) ∇·u*
  \\5- Computo de gradientes de presion y correccion de velocidad. u* - (Δt/ρ) ∇p^{n+1} → u^{n+1}
}
```

Las etapas 1 y 4 son las que más tiempo de cómputo requieren y serán descriptas en secciones subsecuentes. La etapa 2 es realizada mediante operaciones algebraicas de nivel 1 (vector-vector) provistas por la API Thrust. La etapa 3, el cálculo de la ecuación Laplaciana en la 4 y el cómputo de gradientes en la 5 se realizan de manera similar y seguirán un esquema simplificado de la etapa 1. Mientras que la corrección de velocidad de la etapa 5 será realizada en un kernel actuando conjuntamente por los gradientes de presión para aprovechar las lecturas sobre la memoria global que ya se han realizado, aunque puede haberse optado por utilizar una actualización vectorial vía Thrust como ha sido utilizado en la etapa 2.

7.4. Resolución de las ecuaciones de cantidad de movimiento

Con los recursos de la GPGPU Tesla C1060 en mente, se decidió proceder con una resolución sobre el dominio computacional considerando sucesivos planos del mismo. Esto es, si se supone el esquema de almacenado de campos como ha sido introducido en la sección anterior, es decir, según direcciones z, y y x, luego resulta natural la división en sucesivos planos YZ, confeccionando stencils con información de planos contiguos, y una vez finalizados los cálculos, tomar el siguiente plano YZ hasta que finalmente se hayan procesados todos. Entonces se tendrán tantos ¨planos de avance¨ como nodos en la dirección x.

En la Figura (7.2) se presenta la forma de procesar el plano completo mediante la subdivisión del mismo, y los nodos requeridos para la conformación de stencils. Cada nodo esta asociado con un thread, y cada uno de ellos lee al menos información de 3 nodos, el valor actual del campo de velocidad en la posición (gx, gy, gz) y los nodos pertenecientes a los planos en dirección x contiguos, es decir, con $gx \pm 1$.

Considerando las ecuaciones derivadas en la sección (4.3) se tiene que son necesarios tres planos en dirección x, a su vez de acuerdo a los requeridos para el cómputo de las ecuaciones de cantidad de movimiento por cada thread como se muestra en la Figura (7.4), luego estos planos tienen distintas dimensiones, a saber, $(1+\mathrm{BY}+1, 1+\mathrm{BZ}+1)$ para el plano del medio, $(\mathrm{BY}+1, \mathrm{BZ}+1)$ para el plano de atrás y $(1+\mathrm{BY}, 1+\mathrm{BZ})$ para el plano de adelante. La posición de los 1s dentro de las dimensiones hace referencia para donde se expande esa fila o columna considerando un bloque genérico cuyas dimensiones serían $(\mathrm{BY}, \mathrm{BZ})$. Así el plano de atrás por ejemplo tendrá una fila arriba y una columna a la derecha del bloque genérico, mientras que el plano de adelante tendrá un fila abajo y una columna a la izquierda del bloque genérico. El plano central por su parte, tendrá 2 filas (una arriba y otra abajo) y 2 columnas (una a izquierda y otra a derecha) considerando el bloque genérico. Esta situación se observa en la Figura (7.3).

En el Código (7.2) se exponen tanto las variables requeridas como finalmente las etapas involucradas en el cómputo de las ecuaciones de cantidad de movimiento, a saber, la etapa de carga de datos, la convección y la difusión (recordando que no se tienen en cuenta términos de fuerzas externas).

Código 7.2: Kernel de cantidad de movimiento definicion e inicializacion de variables.

```
//Planos en memoria compartida.
```

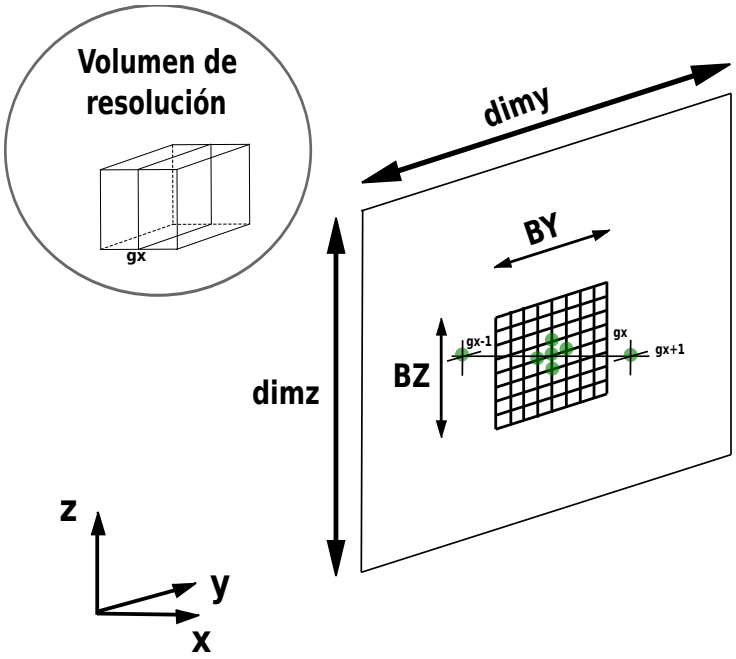


Figura 7.2: Rodaja número gx del dominio de resolución, se observa un stencil con sus 3 nodos tanto en la dirección y como en la z, y luego los 3 nodos en la dirección x. Además se presenta la subdivisión del plano YZ que es procesada por un SM en particular, o en otras palabras, la subdivisión en theadblocks de tamaños (BY, BZ).

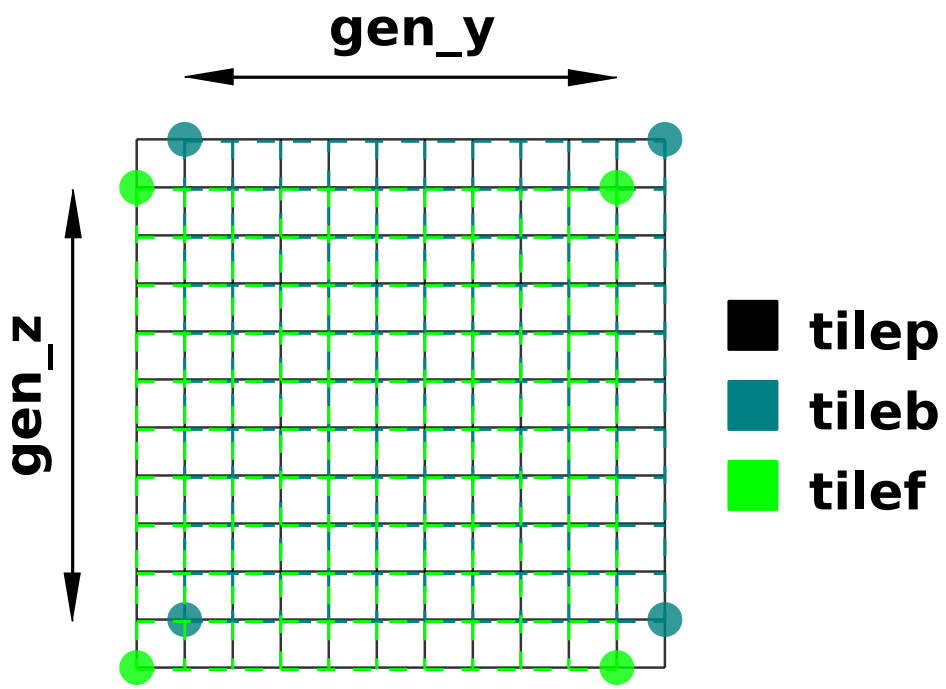


Figura 7.3: Arreglos bidimensionales necesarios para la confección de los stencils. Se observa que tilep, el plano del medio, tiene como dimensiones (1+BY+1,1+BZ+1), mientras que tileb, el plano de atrás, (BY+1,BZ+1) y tilef, el de adelante, (1+BY,1+BZ).

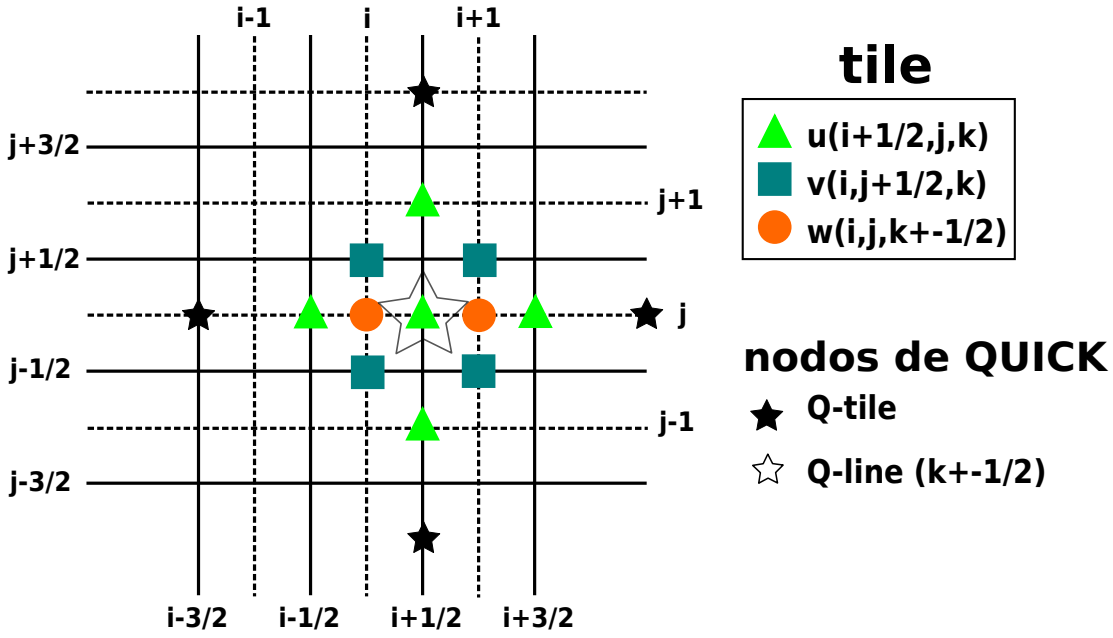


Figura 7.4: Nodos requeridos para el cálculo del residuo en u. La expresión *TILE* hace referencia al threadblock (o división del plano YZ), mientras que *LINE* a los nodos según la dirección de avance (x).

```
__shared__ velocity<scalar> tileb [BY+1][BZ+1]; //ty+1, tz+1
__shared__ velocity<scalar> tilep [1+BY+1][1+BZ+1];
__shared__ velocity<scalar> tilef [1+BY][1+BZ]; //ty-1, tz-1

//Indices globales
const int gy = blockIdx.y * blockDim.y + threadIdx.y;
const int gz = blockIdx.x * blockDim.x + threadIdx.x;

//Indices del bloque (BY,BZ) actual
const int bty = threadIdx.y;
const int btz = threadIdx.x;

//Macro para indexacion
#define grid(i,j,k) (dimy*dimz*(i) + dimz*(j) + 1*(k))

//Auxiliares.
velocity<scalar> rpdaux = {0,0,0}, qvel = {0,0,0};
scalar velprom;

//Lazo en direccion x.
for(int gx=0;gx<dimx;gx++){
  //1-Carga de datos.
  //2-Conveccion + QUICK.
  //3-Difusion.
}
```

Las variables globales gx, gy y gz se utilizan para acceder a cualquier elemento del dominio computacional. Las variables locales ty y tz sólo sirven para acceder a posiciones dentro de un threadblock en particular, observar que existen sólo dos registros para acceder a los 3 planos en memoria compartida, luego el centrado y el de adelante aplican un incremento sobre ese registro cuando efectúan indexaciones, es decir, se utilizan como tx+1 y ty+1 respectivamente. Esto es así puesto que se considera la coordenada $(0,0)$ como el nodo ubicado en $(0,0)$ para el plano de atrás y $(1,1)$ para el plano del medio y el de adelante.

Luego se tiene una macro, y varias variables auxiliares. Si se utilizara CUDA-GDB para saber en que memoria residen dichas variables, uno obtendría que cuando se tienen vectores o estructuras (como es el caso de rpdaux y qvel) luego estas estarán definidas en la memoria local y no en registros, estos es realizado por el compilador como ha sido introducido anteriormente. Sin embargo parece ser que realiza

algún tipo de optimización y las trata como registros (puede ser considerado como que conoce que se utilizarán en reiteradas oportunidades), ya que si uno creara variables que no sean arreglos (como por ejemplo los índices, que según CUDA-GDB sí residen en memoria de registros), operara sobre ellas y luego finalmente confeccionara las estructuras con estas variables no vería mejora alguna.

7.4.1. Carga de datos

La carga de datos tendrá lugar en básicamente dos secciones del kernel, en la primera se leerán los datos de la memoria global y se asignarán a los planos según corresponda. En esta etapa no se leerán valores nodales en las fronteras requeridos para el cómputo de QUICK, estos serán cargados en la segunda etapa y a medida que se vayan utilizando. De esta forma sólo es necesario un registro que guarde la lectura, y no memoria compartida o lecturas adicionales.

Inicialmente cada thread debe leer los valores del nodo (gx, gy, gz) según corresponda, y además los nodos pertenecientes a planos anteriores y posteriores $gx \mp 1$, situación que se observa en el Código (7.3).

Código 7.3: Lectura de los planos sin considerar valores en frontera.

```
tileb[bty][btz] = (gx==0) ? udev[grid(dimx-1,gy,gz)]
                          : udev[grid(gx-1,gy,gz)];
tilep[(bty+1)][(btz+1)] = udev[grid(gx,gy,gz)];
tilef[(bty+1)][(btz+1)] = (gx==dimx-1) ? udev[grid(0,gy,gz)]
                                       : udev[grid(gx+1,gy,gz)];
```

Donde se observa que dado a la condiciones periódicas que fueron asumidas, como el plano de atrás consiste en el plano YZ en posición $gx - 1$, si el plano del medio actual es $gx = 0$ luego el plano de atrás consistirá en el plano YZ con $gx = dimx - 1$, y eso es lo que se observa en la primer linea de código. Una situación similar sucede con el plano hacia adelante.

Los términos de las fronteras son cargados por aquellos threads más externos del threadblock, el Código (7.4) refleja este hecho.

Código 7.4: Lectura de nodos frontera.

```
if(gy==0){
  tilep[(bty+1)-1][(btz+1)] = udev[grid(gx,dimy-1,gz)];
  tilef[(bty+1)-1][(btz+1)] = (gx==dimx-1) ? udev[grid(0,dimy-1,gz)]
                                           : udev[grid(gx+1,dimy-1,gz)];
}
else if(gy==dimy-1){
  tileb[bty+1][btz] = (gx==0) ? udev[grid(dimx-1,0,gz)]
                              : udev[grid(gx-1,0,gz)];
  tilep[(bty+1)+1][(btz+1)] = udev[grid(gx,0,gz)];
}
else if((bty+1)==1){
  tilep[(bty+1)-1][(btz+1)] = udev[grid(gx,gy-1,gz)];
  tilef[(bty+1)-1][(btz+1)] = (gx==dimx-1) ? udev[grid(0,gy-1,gz)]
                                           : udev[grid(gx+1,gy-1,gz)];
}
else if ((bty+1)==BY){
  tileb[bty+1][btz] = (gx==0) ? udev[grid(dimx-1,gy+1,gz)]
                              : udev[grid(gx-1,gy+1,gz)];
  tilep[(bty+1)+1][(btz+1)] = udev[grid(gx,gy+1,gz)];
}

if(gz==0){
  tilep[(bty+1)][(btz+1)-1] = udev[grid(gx,gy,dimz-1)];
  tilef[(bty+1)][(btz+1)-1] = (gx==dimx-1) ? udev[grid(0,gy,dimz-1)]
                                           : udev[grid(gx+1,gy,dimz-1)];
}
else if(gz==dimz-1){
  tileb[bty][btz+1] = (gx==0) ? udev[grid(dimx-1,gy,0)]
                              : udev[grid(gx-1,gy,0)];
  tilep[(bty+1)][(btz+1)+1] = udev[grid(gx,gy,0)];
}
```

```
else if((btz+1)==1){
  tilep[(bty+1)][(btz+1)-1] = udev[grid(gx,gy,gz-1)];
  tilef[(bty+1)][(btz+1)-1] = (gx==dimx-1) ? udev[grid(0,gy,gz-1)]
                                           : udev[grid(gx+1,gy,gz-1)];
}
else if ((btz+1)==BZ){
  tileb[bty][btz+1] = (gx==0) ? udev[grid(dimx-1,gy,gz+1)]
                              : udev[grid(gx-1,gy,gz+1)];
  tilep[(bty+1)][(btz+1)+1] = udev[grid(gx,gy,gz+1)];
}
```

Se puede observar entonces como primero se realiza la consulta de si está en una frontera del dominio computacional, pues si así resultara debería leer los datos de la frontera contraria. De otra forma, conoce que no esta en ninguna frontera por ende esta necesariamente en el interior, es más, el rango de los índices globales resulta $gy \in [BY-1; dimy-(BY-1)]$ y $gz \in [BZ-1; dimz-(BZ-1)]$.

Al momento se han cargado totalmente aquellos elementos a compartir, pero necesita de alguna forma asegurar que todos los threads que están colaborando terminaron de cargar los datos, pues de otra forma cuando se calculen los flujos se podrían estar leyendo datos que no sean los correctos (pues no habían sido cargados). Esto se realiza mediante barreras de sincronización que establece una marca en dicha línea de código y devuelve el control de ejecución a la aplicación sólo si todos los threads de un mismo threadblock llegaron hasta allí.

Código 7.5: Sincronización.

```
__syncthreads(); //Sincronizando carga de memoria local.
```

Finalmente para terminar de cargar los datos necesarios para la confección de stencils se realizan dos lecturas más, fuera de la barrera de sincronización porque son utilizados por el thread actual. Además se tiene que los nodos en los vértices del plano central no han sido cargados, y sólo dos de ellos se utilizan. Si uno revisara las ecuaciones, notaría que estos nodos corresponden a los nodos cuyas coordenadas son $(0, BZ-1)$ y $(BY-1, 0)$. De esta forma las cargas resultante se muestran en el Código (7.6).

Código 7.6: Carga de datos en vértices.

```
if((bty+1)==1 && (btz+1)==BZ){
  if(gy==0 && gz==dimz-1){
    tilep[(bty+1)-1][(btz+1)+1] = udev[grid(gx,dimy-1,0)];
  }
  else if(gy==0 && gz!=dimz-1){
    tilep[(bty+1)-1][(btz+1)+1] = udev[grid(gx,dimy-1,gz+1)];
  }
  else if(gy!=0 && gz==dimz-1){
    tilep[(bty+1)-1][(btz+1)+1] = udev[grid(gx,gy-1,0)];
  }
  else{
    tilep[(bty+1)-1][(btz+1)+1] = udev[grid(gx,gy-1,gz+1)];
  }
}
else if((bty+1)==BY && (btz+1)==1){
  if(gy==dimy-1 && gz==0){
    tilep[(bty+1)+1][(btz+1)-1] = udev[grid(gx,0,dimz-1)];
  }
  else if(gy==dimy-1 && gz!=0){
    tilep[(bty+1)+1][(btz+1)-1] = udev[grid(gx,0,gz-1)];
  }
  else if(gy!=dimy-1 && gz==0){
    tilep[(bty+1)+1][(btz+1)-1] = udev[grid(gx,gy+1,dimz-1)];
  }
  else{
    tilep[(bty+1)+1][(btz+1)-1] = udev[grid(gx,gy+1,gz-1)];
  }
}
```

7.4.2. Términos convectivos y la estabilización vía QUICK

Con los datos requeridos por los stencils de cada nodo cargados en memoria compartida tenemos garantizado un acceso con ancho de banda superior al acceso al de la memoria global de entre 150-200 veces más eficiente. Además, el problema de lecturas fusionadas ya no tiene sentido y sólo ha sido tenido en cuenta en el proceso de carga, donde en general los nodos de borde realizaban accesos sin fusión.

No es la intención del presente documento explicar todas las expresiones presentes en el kernel de cantidad de movimiento, más sí la mecánica de indexación que se tiene para llevar el término de la ecuación discreta a un valor de un plano en particular.

Considerando el cálculo de convección según x y sólo un término en dirección x, y además recordando la ecuación que lo gobierna como

$$\frac{(uu^Q)^n_{i+1,j,k} - (uu^Q)^n_{i,j,k}}{\Delta x}, \tag{7.1}$$

donde

$$u^n_{i+1,j,k} = \frac{1}{2}\left(u^n_{i+\frac{3}{2},j,k} + u^n_{i+\frac{1}{2},j,k}\right), \tag{7.2}$$

y

$$\left(u^Q\right)^n_{i+1,j,k} = \begin{cases} c_0 u^n_{i+\frac{3}{2},j,k} + c_1 u^n_{i+\frac{1}{2},j,k} + c_2 u^n_{i-\frac{1}{2},j,k}, & si \;\; u^n_{i+1,j,k} \geq 0 \\ c_0 u^n_{i+\frac{1}{2},j,k} + c_1 u^n_{i+\frac{3}{2},j,k} + c_2 u^n_{i+\frac{5}{2},j,k}, & si \;\; u^n_{i+1,j,k} < 0 \end{cases}, \tag{7.3}$$

entonces se tiene que de acuerdo al signo de la velocidad $\left(u^Q\right)^n_{i+1,j,k}$ podría como no ser necesario un nodo desplazado en dos posiciones en dirección x. De esta forma existen dos formas de cargar dicho valor: en la primera se carga sin más, como desventaja podría tenerse una lectura inútil, que considerando la latencia de las operaciones sobre la memoria global, en principio la situación debería de ser evitada; en la segunda la lectura se es realizada sólo si fuera necesario, que como principal problema introducirá divergencia de threads y según las pruebas que se han realizado presenta un rendimiento peor al primer caso. Finalmente se optó por realizar la lectura, independientemente de su utilización. De esta forma la sección del kernel que realiza la operación descripta por las ecuaciones (7.1-7.3) se presenta en el Código (7.7),

Código 7.7: Cómputo de un término de las ecuaciones de cantidad de movimiento - conveccion.

```
qvel = ((gx==dimx-1) || (gx==dimx-2)) ? udev[grid(gx-(dimx-2),gy,gz)]
                                      : udev[grid(gx+2,gy,gz)];

velprom = 0.5*(tilef[(bty+1)][(btz+1)].u+tilep[(bty+1)][(btz+1)].u);

if(velprom >= 0.){
  rpdaux.u = -velprom*(c0*tilef[(bty+1)][(btz+1)].u+
                       c1*tilep[(bty+1)][(btz+1)].u+
                       c2*tileb[bty][btz].u)/deltax;
}
else{
  rpdaux.u = -velprom*(c0*tilep[(bty+1)][(btz+1)].u+
                       c1*tilef[(bty+1)][(btz+1)].u+
                       c2*qvel.u)/deltax;
}
```

en donde se observa como en la primer linea se carga en un auxiliar el valor del nodo desplazado en dos planos según dirección x, posteriormente se realiza el promedio de velocidades como es propuesto en la ecuación (7.2) y finalmente se hace el cálculo de QUICK según ecuación (7.3). Los coeficientes c0, c1 y c2 son los coeficientes de QUICK y se encuentran almacenados en la memoria constante con el objetivo de (1) reducir la cantidad de registros y (2) utilizar el caché que dispone; de esta forma sólo una lectura sobre la memoria constante es requerida y luego se tienen replicaciones vía broadcast de lecturas al caché. Esto se observa en el Código (7.8).

Código 7.8: Coeficientes de QUICK en memoria constante.

```
//Quick coefficients.
__device__ __constant__ float c0 = 0.375, c1 = 0.75, c2 = -0.125;
```

Se analizará entonces la indexación utilizada. Los nodos $(i+\frac{1}{2},j,k)$, $(i,j+\frac{1}{2},k)$ e $(i,j,k+\frac{1}{2})$ son accedidos bajo la misma secuencia tilep[(bty + 1)][(btz + 1)], sólo diferenciando la disposición de las grillas por los campos u, v y w de la estructura adoptada para el campo de velocidades. De otra forma podría decirse que la grillas staggered son accedidas como si fueran una misma grilla, centradas en los nodos de presión.

De esta forma por ejemplo uno accede a la posición de $u^n_{i+\frac{3}{2},j,k}$ como tilef[(bty + 1)][(btz + 1)].u, tilef pues esta leyendo un nodo desplazado en una unidad del actual desplazamiento de 1/2 en dirección x. Una situación similar ocurriría si se deseara acceder a los datos del nodo $u^n_{i-\frac{1}{2},j,k}$, solamente que el plano de lectura resultaría tileb, dado a que se alinea con el campo de presión en $p_{i-1,j,k}$. Finalmente los nodos en otras direcciones, o bien cruzados, siguen una lógica similar incrementando o decrementando alguna de las dimensiones de los bloques.

7.4.3. Términos difusivos

Con respecto a los términos difusivos, siguen el esquema de discretización del operador Laplaciano en 3D que se muestra en el Código (7.9).

Código 7.9: Cómputo de un término de las ecuaciones de cantidad de movimiento - difusion.

```
rpdaux.u += nu * ((tilef[(bty+1)][(btz+1)].u
                  -2.*tilep[(bty+1)][(btz+1)].u
                  +tileb[bty][btz].u)/(deltax*deltax)
                  +
                  (tilep[(bty+1)+1][(btz+1)].u
                  -2.*tilep[(bty+1)][(btz+1)].u
                  +tilep[(bty+1)-1][(btz+1)].u)/(deltay*deltay)
                  +
                  (tilep[(bty+1)][(btz+1)+1].u
                  -2.*tilep[(bty+1)][(btz+1)].u
                  +tilep[(bty+1)][(btz+1)-1].u)/(deltaz*deltaz));
```

7.5. El problema de Poisson para la presión

Para la resolución del problema de Poisson para la presión se implementó un gradiente conjugado como el introducido en el Capítulo (5), y se utilizó la FFT (transformada rápida de Fourier) provista por la API CUFFT. El concepto es sencillo, resolver el sistema considerando cuerpos inmersos en el fluido utilizando la FFT [MASC10] y con su solución precondicionar al Laplaciano que será introducido en el CG. En general la solución obtenida con la FFT es tan buena, que el CG alcanza a converger en pocas iteraciones (si sólo se tiene fluido, en una sola iteración). Cuando existen cuerpos, estos últimos introducen errores que luego el CG deberá amortiguar.

Como se sabe el RHS de la ecuación de Poisson es una divergencia. Esta se calcula como un esquema simplificado de la resolución presentada para las ecuaciones de cantidad de movimiento. En general hay una etapa de carga de memoria, y posteriormente una etapa de cálculo de divergencia. Considerando dominios de resolución cúbicos con N^3 elementos, luego se necesitan leer $3N^2$ Bytes, dado que cada nodo presenta tres componentes. Luego se tienen 6 operaciones por nodo, con lo cual se tiene un total de $6N^3$ operaciones. Finalmente el cociente de operaciones y lecturas resulta $\mathcal{O}(1)$, con lo cual un procesamiento en GPGPU no brindaría ninguna ventaja extra comparada con una implementación en CPU.

Con el objeto de resolver todas las etapas de Pasos fraccionado en GPU se optó por una implementación sencilla de cálculo de divergencias. Recordando, se tiene que la divergencia se calcula como un balance sobre la celda de presión, es decir, centrada en el nodo (i,j,k)

$$(\nabla \cdot \mathbf{u}^*)_{i,j,k} = \frac{u^*_{i+\frac{1}{2},j,k} - u^*_{i-\frac{1}{2},j,k}}{\Delta x} + \frac{v^*_{i,j+\frac{1}{2},k} - v^*_{i,j-\frac{1}{2},k}}{\Delta y} + \frac{w^*_{i,j,k+\frac{1}{2}} - w^*_{i,j,k-\frac{1}{2}}}{\Delta z}, \tag{7.4}$$

así considerando sólo un plano con dimensiones $(1 + \mathrm{BY}, 1 + \mathrm{BZ})$ se tiene

Código 7.10: Cálculo de la divergencia del campo velocidad.

```
udivdev[index] = (tileu[ty][tz].u-lineu.u)/(deltax)+
                 (tileu[ty][tz].v-tileu[ty-1][tz].v)/(deltay)+
                 (tileu[ty][tz].w-tileu[ty][tz-1].w)/(deltaz);
```

donde lineu es una variable que almacena el campo de velocidades en un plano anterior, esto es

Código 7.11: Lectura de una posición por detrás.

```
lineu = (gx==0) ? udev[grid(dimx-1,gy,gz)]
                : udev[grid(gx-1,gy,gz)];
```

Finalmente se aplica el factor de escala $-\rho/\Delta t$ vía Thrust, el signo negado pues porque se resuelve el negado del Laplaciano de la presión, dado a que el Laplaciano es un operador definido negativo y el CG necesita que la matriz LHS sea definida positiva y simétrica. Luego, una discretización que preserve la condición de isotropía sobre la difusión resulta simétrico, y así se garantiza la convergencia del CG.

Los detalles de la implementación del CG y la precondición vía FFT no serán introducidos en este documento. Pero en esencia lo que se hace es aplicar tanto al inicio del CG como por cada iteración sobre el sistema, el operador de precondición obtenido vía FFT.

7.6. Corrección del campo de velocidades

Con respecto a este último paso ocurre una situación similar al caso de la divergencia, la ganancia de realizar esta operación sobre la GPU no resulta en una ventaja sobre otra calculada en CPU. Obviamente, como aquí se trata con valores que residen en la GPU conviene realizar dichas operaciones sobre la misma (para evitar el traspaso de datos vía el bus PCI-Express), pero si se dispusieran en la memoria principal el hecho de enviar los datos a través del PCI-Express sólo para calcular en GPU y luego devolver los datos podría resultaría en una pérdida de rendimiento que, en principio, debería ser evitada.

Recordando las ecuaciones para el cálculo de gradientes de presión como

$$\nabla p^{n+1} = \left\{ \begin{array}{l} \left(\dfrac{\partial p}{\partial x}\right)^{n+1}_{i+\frac{1}{2},j,k} \\ \left(\dfrac{\partial p}{\partial y}\right)^{n+1}_{i,j+\frac{1}{2},k} \\ \left(\dfrac{\partial p}{\partial z}\right)^{n+1}_{i,j,k+\frac{1}{2}} \end{array} \right\} = \left\{ \begin{array}{l} \dfrac{p^{n+1}_{i+1,j,k} - p^{n+1}_{i,j,k}}{\Delta x} \\ \dfrac{p^{n+1}_{i,j+1,k} - p^{n+1}_{i,j,k}}{\Delta y} \\ \dfrac{p^{n+1}_{i,j,k+1} - p^{n+1}_{i,j,k}}{\Delta z} \end{array} \right\}. \tag{7.5}$$

y la posterior actualización como

$$\mathbf{u}^{n+1}_{i,j,k} = \left\{ \begin{array}{l} u^{n+1}_{i+\frac{1}{2},j,k} \\ v^{n+1}_{i,j+\frac{1}{2},k} \\ w^{n+1}_{i,j,k+\frac{1}{2}} \end{array} \right\} = \left\{ \begin{array}{l} u^{*}_{i+\frac{1}{2},j,k} - \dfrac{\Delta t}{\rho}\left(\dfrac{\partial p}{\partial x}\right)^{n+1}_{i+\frac{1}{2},j,k} \\ v^{*}_{i,j+\frac{1}{2},k} - \dfrac{\Delta t}{\rho}\left(\dfrac{\partial p}{\partial y}\right)^{n+1}_{i,j+\frac{1}{2},k} \\ w^{*}_{i,j,k+\frac{1}{2}} - \dfrac{\Delta t}{\rho}\left(\dfrac{\partial p}{\partial z}\right)^{n+1}_{i,j,k+\frac{1}{2}} \end{array} \right\}, \tag{7.6}$$

luego un esquema de resolución constaría de un plano con dimensiones $(\mathrm{BY} + 1, \mathrm{BZ} + 1)$, cuya carga de datos sería similar (pero reducido) al presentado en la resolución de las ecuaciones de cantidad de movimiento. Finalmente el código que procesa los gradientes y aplica las correcciones se presenta en el Código (7.12).

Código 7.12: Cálculo de gradientes de presión y corrección de velocidad.

```
pu = (gx==dimx-1) ? pdev[grid(0,gy,gz)] : pdev[grid(gx+1,gy,gz)];

//Se actualiza la velocidad con los gradientes de presion.

aux = udev[grid(gx,gy,gz)];

aux.u += -dT/rho*((pu-tile[ty][tz])/deltax);
aux.v += -dT/rho*((tile[ty+1][tz]-tile[ty][tz])/deltay);
aux.w += -dT/rho*((tile[ty][tz+1]-tile[ty][tz])/deltaz);

udev[grid(gx,gy,gz)] = aux;
```

Capítulo 8

Resultados

Esta sección presenta los resultados, tanto visuales como numéricos, obtenidos por el desarrollo en CUDA del modelo discreto que se ha resuelto. Es de interés mencionar que la visualización se ha realizado con VTK[1], una herramienta OpenSource de visualización científica ampliamente aceptada por la comunidad que, integrada con en el código de resolución de las ecuaciones de N-S, permite visualización en tiempo real (es decir, se puede visualizar mientras se calcula).

8.1. Caso de estudio: La inestabilidad de Kelvin-Helmholtz

El problema de Kelvin-Helmholtz surge con la necesidad de una explicación para la formación de olas bajo la acción del viento. Sin embargo el mismo fenómeno tiene lugar en un sin fin de fenómenos físicos, como por ejemplo, en la formación de nubes con estructuras como se observan en la Figura (8.1).

Sus fundamentos físicos serán introducidos a continuación. Considere el caso particular de dos fluidos inmiscibles dentro de un recinto ubicados espacialmente según $z > 0$ y $z < 0$, con densidades y presiones ρ_1, p_1 y ρ_2, p_2 y finalmente con velocidades $U_1\vec{i}$ y $U_2\vec{i}$ respectivamente. Considerando la gravedad este problema se denomina *la inestabilidad de Kelvin-Helmholtz*, pero en aras de simplicidad, no se tendrá en cuenta la misma. Con el objetivo de estudiar la inestabilidad, se introducirá una perturbación $\zeta(x,t)$ que actúa como término de desestabilización por acción de gravedad.

Sin entrar en detalles (que si pueden ser encontrados en [Dra02, Capítulo 3]), puede considerarse que la velocidad para cada fluido surge con la aplicación de un potencial sobre ciertas funciones, Φ_1 y Φ_2, respectivamente.

De esta manera se considera que

- lejos de la interfaz donde actúa la perturbación las velocidades resultan $U_1\vec{i}$ y $U_2\vec{i}$ respectivamente,
- las perturbaciones $\phi_1 \ll \Phi_1$ y $\phi_2 \ll \Phi_2$,
- la perturbación presenta carácter senoidal de la forma $\zeta = Ae^{i(kx-wt)}$ donde A es la amplitud de la perturbación, $k = 2\pi/\lambda$ es el número de onda con λ la longitud del período y w es la frecuencia angular,
- para satisfacer la condición de Laplace (de linealidad) sobre ϕ_1 y ϕ_2 se impone

$$\begin{aligned} \phi_1 &= \bar{\phi_1} e^{-kz} e^{i(kx-wt)} \\ \phi_2 &= \bar{\phi_2} e^{+kz} e^{i(kx-wt)} \end{aligned}, \tag{8.1}$$

para finalmente obtener un sistema determinado con 3 ecuaciones cuyas incógnitas son A, $\bar{\phi_1}$ y $\bar{\phi_2}$.

Resolviendo dicho sistema se obtiene que la frecuencia angular obedece

$$w = k\frac{\rho_1 U_1 + \rho_2 U_2}{\rho_1 + \rho_2} \pm ik\frac{\sqrt{\rho_1\rho_2}|U_1 - U_2|}{\rho_1 + \rho_2}, \tag{8.2}$$

[1] http://www.vtk.org/

Figura 8.1: La inestabilidad de Kelvin-Helmholtz en los cielos de Australia.

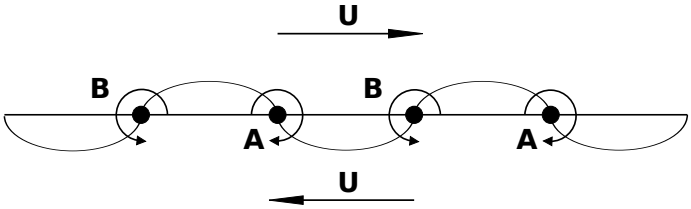


Figura 8.2: Esquema de vorticidad según z=0, se observan los puntos de máximo incremento y decremento de vorticidad total. Las flechas curvas indican el sentido de circulación de la perturbación.

sustituyendo esta última expresión en ϕ_1 y ϕ_2 se observa que si se considera la ecuación (8.2) con signo positivo, luego la exponencial presenta su carácter de factor de amortiguamiento (es decir, es una exponencial negativa). De otra forma, si se considera el signo negativo, la exponencial actúa como un término de desestabilización (o divergencia).

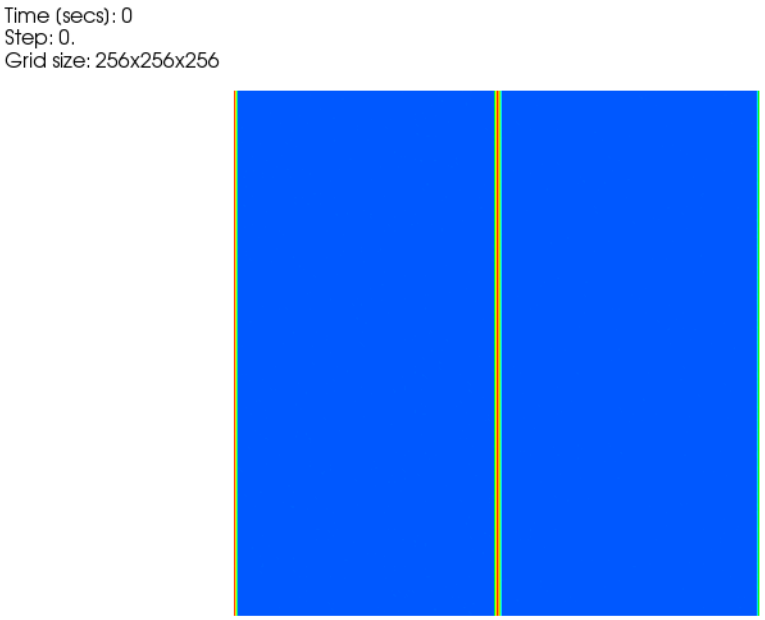
Considerando además obtener la vorticidad total en dirección y como una integral sobre una capa infinitesimal en $z = 0$, se tiene que la perturbación de vorticidad varía en forma cosenoidal. Considerando la Figura (8.2), se tiene que la vorticidad total es incrementada al máximo en puntos como el A, en dirección horaria, mientras que es decrementada al máximo en puntos como el B, en dirección antihoraria, siendo cero es los valles y crestas (dado por el coseno).

Los resultados obtenidos en las simulaciones se presentan en las Figuras (8.1).

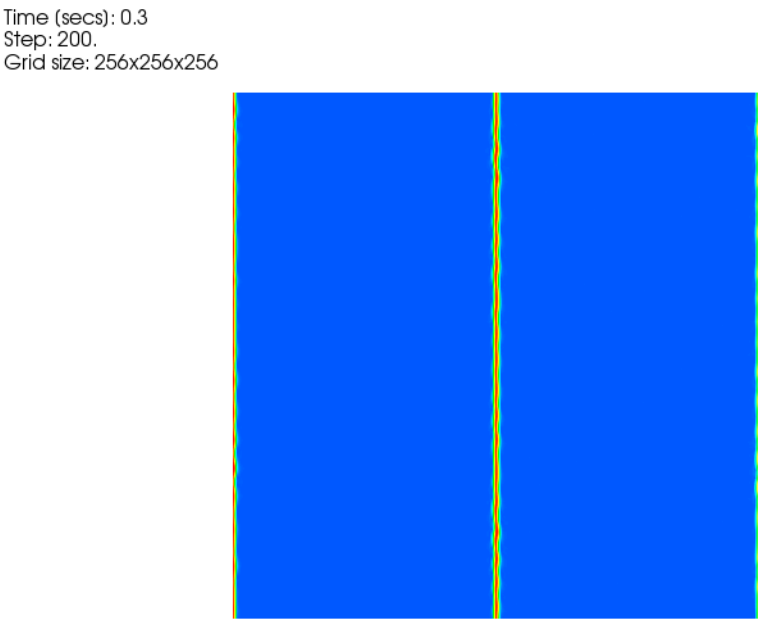
8.2. Cálculos de ancho de banda

Una medida importante es el ancho de banda alcanzado por los distintos kernels que se han implementado, en especial el de cantidad de movimiento. Para ello debe tenerse primero una definición de *ancho de banda*. En esencia el ancho de banda de un determinado componente de hardware se calcula como el producto entre la frecuencia de su reloj (medida en Hertz), y la cantidad de lineas por donde tiene la oportunidad de inyectar los bits (medido en bits), también denominado interfaz (pues representa el lugar en donde ocurre la comunicación con otros dispositivos).

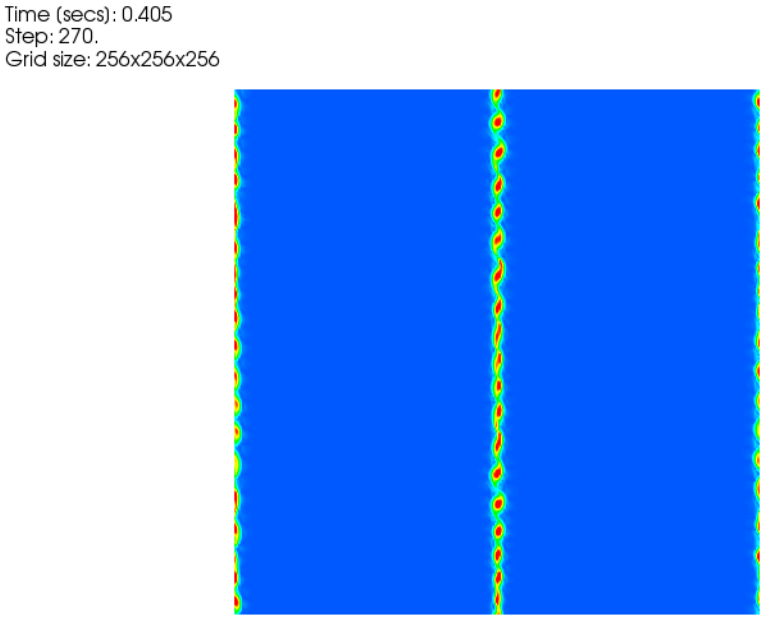
En general se tienen dos tipos distintos de cálculos de ancho de banda, el *teórico* y el *efectivo*. El primer tipo se calcula en base a la frecuencia y la longitud de palabra de la interfaz que se obtienen de la hoja de especificaciones del producto, en general son medidas logradas en condiciones óptimas que, a fines prácticos, resultan imposibles de alcanzar. El segundo tipo se calcula teniendo en cuenta la cantidad de Bytes leídos por el kernel, o B_R, la cantidad de Bytes escritos, o B_W, y el tiempo de procesamiento



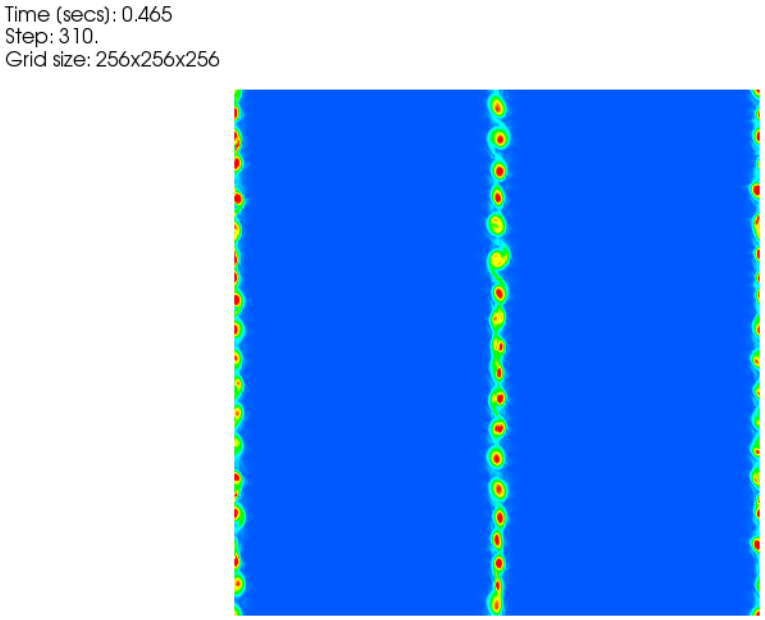
(a) Condiciones iniciales.



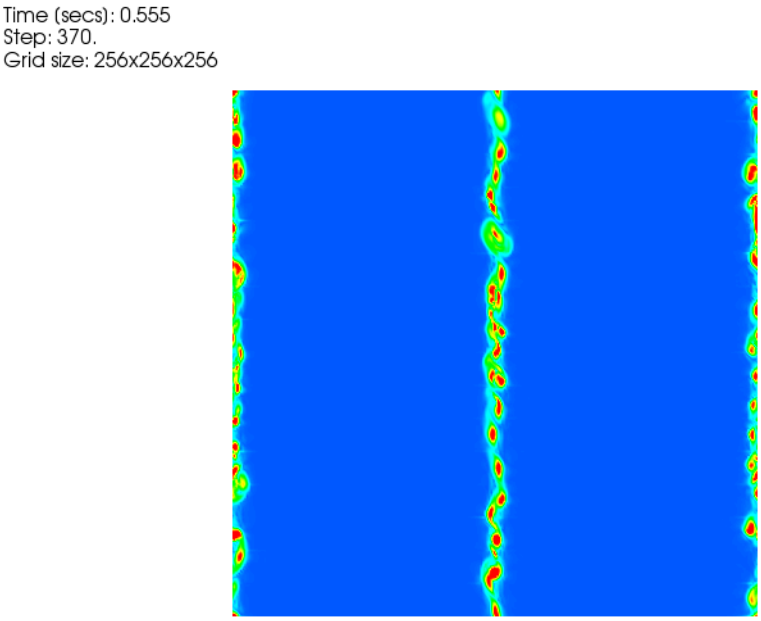
(b) Comienzan a notarse las inestabilidades.



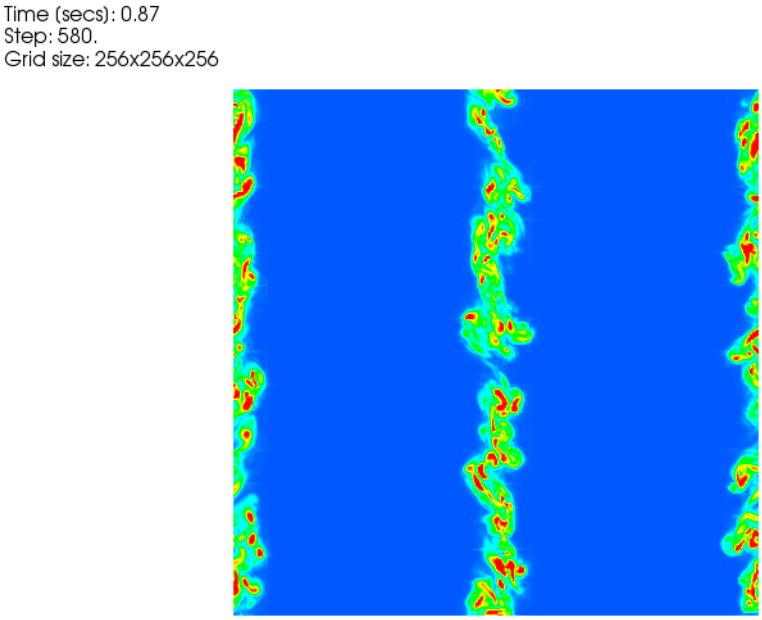
(c) La capa vorticosa va tomando forma.



(d) Ya se tienen vórtices bien definidos.



(e) La convección empieza a desprender los vórtices de la capa central.



(f) Los vórtices empiezan a desparramarse por el resto del dominio.

Figura 8.1: Los parámetros que se han utilizado fueron: dimensiones 256^3, $\nu = 1.004 \cdot 10^{-6}$, $\Delta t = 3.1 \cdot 10^{-3}$. En la simulación se utilizo una tangente hiperbólica para simular una capa de corte continua sobre una de las componentes del campo de velocidad, y además se introdujeron perturbaciones del orden de $1 \cdot 10^3$ que recreaban la inestabilidad. Además la vorticidad (de segundo orden) se calcula con un kernel en CUDA, y resulta como un postproceso del cálculo de la velocidad. Las gráficas presentan un poco de ruido dado al método utilizado para obtener las imágenes.

total, es decir, es una medida real de rendimiento.

Considerando que un kernel en general presenta 4 etapas, esto es, (1) la inicialización de variables, (2) la carga de datos, (3) el cómputo y (4) la escritura, entonces el cálculo de ancho de banda tendrá en cuenta las etapas 1, 2 y 4, sin tener en cuenta la escritura de los datos leídos sobre los registros (en otras palabras, no se tienen en cuenta las asignaciones en memoria compartida). De esta forma se mide el rendimiento en la carga de memoria global, y posteriormente la escritura sobre la misma. Eso es lo que se quiere comparar, en que medida el kernel actual esta aprovechando el máximo rendimiento efectivo otorgado por la memoria global.

De las especificaciones de la Tesla $C1060$ se tiene que su reloj trabaja a 1600 [MHz] y su interfaz es de 512 [bits], por ende el rendimiento teórico máximo resulta

$$1600 \cdot (1 \cdot 10^6)[hz] \cdot 512[b] \cdot \frac{1[GB]}{1024^3 8[b]} = 102.4[GBps], \tag{8.3}$$

donde b se utiliza en representación de los bits, B de los bytes, GB Gigabytes y $GBps$ Gigabytes por segundo.

Utilizando las herramientas otorgadas por el SDK de CUDA se obtiene que el rendimiento efectivo de la memoria global resulta de 74 [GBps], y este es el número contra el cual se harán las comparaciones.

Continuando se tiene entonces una fórmula para el cálculo del ancho de banda de un determinado kernel como

$$[(B_R + B_W) \cdot 1024^{-3}]/t_{total}, \tag{8.4}$$

donde t_{total} es el tiempo calculado como la resta del tiempo tomado inmediatamente tanto antes de la primer como después de la cuarta etapa y el factor $1/1024^3$ es un factor de escala a GBps.

Con respecto al kernel de cantidad de movimiento se necesitan N^3 lecturas sobre la memoria global, cada una accediendo a un campo de velocidad. En 3D se tienen 3 componentes de velocidad que bien podrían ser flotantes de simple o doble precisión, de esta forma se tienen $3N^3 \cdot B$ lecturas, donde B hace referencia al alineamiento en Bytes requerido por ambos tipos de datos, 4 para simple y 8 para doble. Asimismo la cantidad de escrituras es necesariamente la misma que la de lecturas, con lo cual $B_R + B_W = 6N^3 \cdot B$.

Finalmente el Cuadro (8.1) muestra los resultados obtenidos.

	Doble	Simple	
	8x8	8x8	16x16
	[GBps]	[GBps]	[GBps]
32x32x32	31.45 (42.50 %)	21.29 (28.77 %)	16.45 (22.29 %)
64x64x64	45.39 (60.53 %)	37.08 (50.37 %)	40.38 (54.56 %)
128x128x128	50.12 (67.72 %)	43.14 (58.29 %)	47.59 (64.31 %)

Cuadro 8.1: Cuadro comparativo presentando los rendimientos del kernel de cantidad de movimiento. Los valores en rojo representan los GBps obtenidos, mientras que los valores entre paréntesis son los porcentajes de utilización del ancho de banda efectivo.

Se observa en general una buena performance, tanto para simple como para doble precisión, a partir de un dominio de 64^3. El hecho de no disponer del 100 % de eficiencia reduce el rendimiento general del kernel, por lo que en un futuro un punto de ataque para optimizar el código sería mejorar el proceso de carga de datos desde la memoria global. Esta caída de rendimiento se debe principalmente a las condiciones de borde y la forma en que los nodos del dominio de resolución se mapean a un arreglo 1D.

8.3. Comparaciones GPU vs CPU

Con el objetivo de tener una referencia del rendimiento obtenido en el desarrollo propuesto, se procederá a comparar los rendimientos contra una versión en CPU del mismo, considerando versiones unicore y multicore utilizando para ello OpenMP, un estándar de programación multithread en memoria

compartida. La CPU utilizada en las pruebas es un intel i7 950 que presenta 4 núcleos y 12 MB de caché, mientras que la GPU fue una Tesla C1060 con 240 SP.

Es de interés mencionar que en las pruebas

- la optimización con OpenMP sólo es aplicada a la resolución de las ecuaciones de cantidad de movimiento, y
- para el cálculo de las FFT se utilizaron dos versiones, la FFT provista por FFTW (sin threads), y CUFFT (NVIDIA).

8.3.1. Resultados en simple precisión

Los resultados obtenidos al efectuar los cálculos en simple precisión son presentados a continuación. El Cuadro (8.2) presenta los rendimientos en [segs/MCels] obtenidos para un paso de tiempo completo con datos en simple precisión.

	CPU OpenMP (1 thread)	CPU OpenMP (1 thread) +FFT(GPU)	CPU OpenMP (4 threads)	CPU OpenMP (4 threads) +FFT(GPU)	GPU
	[segs/MCels]	[segs/MCels]	[segs/MCels]	[segs/MCels]	[segs/MCels]
32x32x32	0.6391	0.5627	0.5085	0.4329	0.1173
64x64x64	0.7242	0.5852	0.5887	0.4477	0.0294
128x128x128	1.2815	0.5712	1.2107	0.4389	0.0184

Cuadro 8.2: Rendimiento obtenido por paso de tiempo. Tipo de dato simple precisión.

Donde se observa que a medida que las dimensiones del problema crecen la opción de CUDA resulta favorable. Los resultados de speedups se observan en el Cuadro (8.3), teniendo como referencia la versión en GPU, es decir, un factor $2x$ hace referencia a una versión de GPU dos veces más rápida que aquella en CPU.

	CPU OpenMP (1 thread)	CPU OpenMP (1 thread) +FFT(GPU)	CPU OpenMP (4 threads)	CPU OpenMP (4 threads) +FFT(GPU)
32x32x32	5.44x	4.79x	4.33x	3.69x
64x64x64	24.63x	19.90x	20.02x	15.22x
128x128x128	69.64x	31.04x	64.79x	23.85x

Cuadro 8.3: Speedups obtenidos en la resolución de un paso de tiempo completo, GPU vs los distintos desarrollos en CPU. Tipo de dato simple precisión.

Es necesario introducir el hecho de que el calcular la FFT en GPU y el resto en CPU requiere de traspasos de datos por el bus PCI-Express lo que, dado a su escaso ancho de banda (de alrededor de 8 GBps para la versión 2.0), limita el rendimiento general de la implementación.

Con respecto a la toma de tiempos en las ecuaciones de cantidad de movimiento se consideraron dos etapas, la resolución y la integración temporal. Finalmente los resultados son presentados en el Cuadro (8.4).

El Cuadro (8.5) recoge los resultados en speedup, considerando como referencia los resultados obtenidos en GPU.

Para obtener el porqué de los resultados obtenidos se utiliza una herramienta provista por NVIDIA denominada *computeprof*, ella otorga un perfil completo del código CUDA que se este testeando incluyendo tiempos de ejecución, utilización de memoria compartida, registros por threads, información de accesos y escrituras fusionados (y aquellos que no logran la fusión), divergencia de threads, entre otros. De esta forma se obtiene una cantidad de 41 registros requeridos por thread y considerando que el tamaño de bloque escogido para el kernel de cantidad de movimiento fue de 16×16 (superior en rendimiento a bloques con dimensiones menores), se tiene que un sólo bloque activo puede residir por SM. En términos

	CPU OpenMP (1 thread)	CPU OpenMP (4 threads)	GPU
	[segs/MCels]	[segs/MCels]	[segs/MCels]
32x32x32	0.1769	0.0592	0.0341
64x64x64	0.1905	0.0541	0.0104
128x128x128	0.1917	0.0603	0.0093

Cuadro 8.4: Rendimiento obtenido en la resolución de las ecuaciones de cantidad de movimiento. Tipo de dato simple precisión.

	CPU OpenMP (1 thread)	CPU OpenMP (4 threads)
32x32x32	5.18x	1.73x
64x64x64	18.31x	5.20x
128x128x128	20.61x	6.48x

Cuadro 8.5: Speedups obtenidos en la resolución de las ecuaciones de cantidad de movimiento, GPU vs los distintos desarrollos en CPU. Tipo de dato simple precisión.

de warps resultan 8 con lo cual se obtiene una ocupación (cociente entre warps activos y warps disponibles, 32 para la Tesla C1060) de 0.25.

Cabe aclarar que con las dimensiones tratadas sólo un pequeña cantidad de los 30 SMs de los que se disponen son utilizados, entonces si se consideraran problemas con mayores dimensiones se podrían distribuir los threadblocks a mayor cantidad de SM con lo cual una mejora de rendimiento resulta aparente.

8.3.2. Resultados en doble precisión

Los resultados utilizando el tipo de dato doble precisión se muestran en los Cuadros (8.6 y 8.7), en donde los tiempos se toman teniendo en cuenta todas la operaciones requeridas por un paso de tiempo completo.

	CPU OpenMP (1 thread)	CPU OpenMP (1 thread) +FFT(GPU)	CPU OpenMP (4 threads)	CPU OpenMP (4 threads) +FFT(GPU)	GPU
	[segs/MCels]	[segs/MCels]	[segs/MCels]	[segs/MCels]	[segs/MCels]
32x32x32	0.6464	0.5939	0.5252	0.4645	0.2310
64x64x64	0.7894	0.6411	0.6594	0.5109	0.0901
128x128x128	1.5012	0.6384	1.3437	0.5086	0.0602

Cuadro 8.6: Rendimiento obtenido por paso de tiempo. Tipo de dato, doble precisión.

Los Cuadros (8.8) y (8.9) muestran los resultados obtenidos en la resolución de las ecuaciones de cantidad de movimiento.

El porqué de los resultados similares para la resolución de las ecuaciones de cantidad de movimiento en las implementaciones GPU y CPU multithread está en que, la GPU como dispositivo de cálculo masivo de propósito general, no tiene buenos resultados cuando opera en doble precisión. Esto es pues la arquitectura (al menos la Tesla y anteriores) está ideada para cómputos en simple precisión, luego tiene más unidades de simple precisión que de doble. La CPU en cambio, que tiene un grado de evolución para los cálculos de propósito general superior a la GPU, presenta más cantidad de registros de 64 bits, es por ello que la utilización de datos en simple o doble precisión en general no tienen gran diferencia de rendimiento.

El principal problema del kernel en GPU para este caso particular no resulta ni la utilización de memoria compartida, ni su eficiencia de acceso, sino la cantidad de registros por thread requeridos. Utilizando *computeprof* se tiene que cada thread requiere de 88 registros. Considerando threadblocks de

	CPU OpenMP (1 thread)	CPU OpenMP (1 thread) +FFT(GPU)	CPU OpenMP (4 threads)	CPU OpenMP (4 threads) +FFT(GPU)
32x32x32	2.79x	2.57x	2.27x	2.01x
64x64x64	8.76x	7.11x	7.31x	5.67x
128x128x128	24.93x	10.60x	22.32x	8.44x

Cuadro 8.7: Speedups obtenidos en la resolución de un paso de tiempo completo, GPU vs los distintos desarrollos en CPU. Tipo de dato doble precisión.

	CPU OpenMP (1 thread)	CPU OpenMP (4 threads)	GPU
	[segs/MCels]	[segs/MCels]	[segs/MCels]
32x32x32	0.1854	0.0598	0.1071
64x64x64	0.1921	0.0673	0.0602
128x128x128	0.1930	0.0739	0.0435

Cuadro 8.8: Rendimiento obtenido en la resolución de las ecuaciones de cantidad de movimiento. Tipo de dato, doble precisión.

8×8 (cuyo rendimiento resultó superior a threadblocks con dimensiones de 2×2 y 4×4, y además que los bloques de 16×16 no son válidos pues exceden los recursos disponibles) luego necesita 5632 registros, con lo cual sólo dos bloques por SM pueden ser desglosados por la unidad SIMT (pues con 3 bloques se supera la cantidad de registros asociados a SM que son 16384). Luego en términos de warps activos como se tienen threadblocks de 8×8, sólo se disponen de dos warps, considerando entonces que se tienen dos bloques por SM, la cantidad de warps activos por SM resultan 4, de esta forma la ocupación es de 0.125. Un factor bajo, que únicamente podría ser incrementado con la adición de más registros asociados a un SM. La arquitectura Fermi, con 32768 registros por SM, dispondría de 5 threadblocks por SM elevando la cantidad de warps activos a 10, aunque igualmente para esta arquitectura la ocupación resultaría igualmente baja puesto que tiene mayor cantidad de warps disponibles.

El problema de disponer de pocos warps activos ya fue introducido anteriormente, pero básicamente se puede decir que la unidad SIMT no tiene los recursos suficientes como para mantener a los SM activos cuando operaciones de gran latencia (como ser la gran cantidad de operaciones de lecturas requeridas para la confección de stencils) se llevan a cabo.

Se puede decir entonces que una mejor utilización del ancho de banda efectivo no resultaría en un incremento notable puesto que el kernel de cantidad de movimiento se estaría ejecutando sobre la misma cantidad de SMs. Una verdadera mejora consistiría en reducir la cantidad de registros por thread (del lado del programador, o desde el planteo matemático), o bien mediante la introducción de más registros en las GPUs (del lado del proveedor).

8.4. Consideraciones de los esquemas temporales

Como bien es introducido en [JMN08] los esquemas de integración temporal adoptados para el primer paso de F-S resultan incondicionalmente inestable para FE mientras que condicionalmente estable para AB2 [CF92], siempre teniendo en cuenta el esquema de estabilización QUICK para el término convectivo de las ecuaciones de cantidad de movimiento.

Considere una ecuación 1D de advección pura no estacionaria

$$\begin{aligned} &\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0, \\ &\rightarrow \frac{\partial u}{\partial t} = -a\frac{\partial u}{\partial x} = R(u), \end{aligned} \tag{8.5}$$

	CPU OpenMP (1 thread)	CPU OpenMP (4 threads)
32x32x32	1.73x	0.55x
64x64x64	3.19x	1.11x
128x128x128	4.43x	1.69x

Cuadro 8.9: Speedups obtenidos en la resolución de las ecuaciones de cantidad de movimiento, GPU vs los distintos desarrollos en CPU. Tipo de dato, doble precisión.

con integración temporal vía Adams-Bashforth

$$u_{i+\frac{1}{2}}^{n+1} = u_{i+\frac{1}{2}}^{n} + \frac{\Delta t}{2}\left[3R(u)_{i+\frac{1}{2}}^{n} - R(u)_{i+\frac{1}{2}}^{n-1}\right], \tag{8.6}$$

espacial por QUICK

$$\left(\frac{\partial u}{\partial x}\right)_{i+\frac{1}{2}}^{n} \approx \frac{(u^Q)_{i+1}^{n} - (u^Q)_{i}^{n}}{\Delta x}, \tag{8.7}$$

en donde

$$\begin{aligned}(u^Q)_{i+1}^{n} &= \frac{3u_{i+\frac{3}{2}}^{n} + 6u_{i+\frac{1}{2}}^{n} - u_{i-\frac{1}{2}}^{n}}{8},\\ (u^Q)_{i}^{n} &= \frac{3u_{i+\frac{1}{2}}^{n} + 6u_{i-\frac{1}{2}}^{n} - u_{i-\frac{3}{2}}^{n}}{8},\end{aligned} \tag{8.8}$$

con el objetivo de determinar el CFL crítico se hará un análisis típico de estabilidad de Von Neumann, en donde se asume que la solución se compone de ondas planas, esto es

$$u_{i+\frac{1}{2}}^{n} = \lambda^n e^{ikx_{i+\frac{1}{2}}}, \tag{8.9}$$

donde $\lambda \in \mathbb{C}$ es un factor de amplificación temporal, y k el número de onda. La solución sera estable sólo si $|\lambda| \leq 1$. Así se obtiene una ecuación de segundo grado sobre λ de la forma

$$\lambda^2 + \left(\frac{3\beta}{16} - 1\right)\lambda + \frac{\beta}{16} = 0, \tag{8.10}$$

donde

$$\begin{aligned}\beta &= \mathrm{Co}(3 + 6\xi^{-1} - \xi^{-2})(\xi - 1),\\ \mathrm{Co} &= a\frac{\Delta t}{\Delta x},\\ \xi &= e^{ik\Delta x}.\end{aligned} \tag{8.11}$$

De esta forma se proponen Courants entre 0.3 y 0.9 con incrementos de 0.05 y se evalúan para cada uno distintos parámetros ξ sujetos a diferentes números de onda, con lo cual se obtienen ecuaciones cuadráticas. Luego se grafican sus raíces, tanto parte real como imaginaria, que conforman dos lóbulos por Courant como se observa en la Figura (8.2). Para que un esquema sea estable sus curvas deben residir dentro de la circunferencia unitaria.

Finalmente se observa como la curva para Co = 0.6 corta a la circunferencia unitaria, luego el valor límite resulta entre 0.55 y 0.6, más específicamente, 0.588.

Si bien el CFL límite predicho por este análisis no es directamente aplicable al problema resuelto en este PFC, demuestra que la combinación AB+QUICK es condicionalmente estable, mientras que FE+QUICK es incondicionalmente inestable. En las pruebas este mismo comportamiento ha sido observado, con un CFL crítico para AB+QUICK de 0.634.

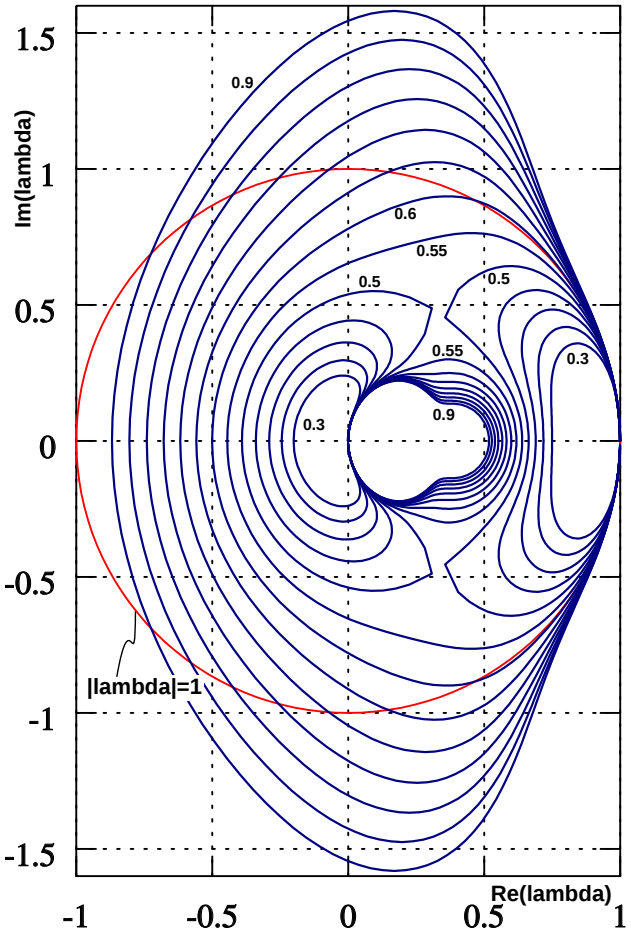


Figura 8.2: En la figura se observan las curvas del factor de amplificación temporal λ obtenidas para una ecuación de advección pura no estacionaria en 1D utilizando integración temporal por Adams-Bashforth de segundo orden y espacial por QUICK, variando el Courant con incrementos de 0.05. Para que el esquema sea estable, la curva de un determinado Courant debe residir en el interior de la circunferencia unitaria.

8.5. Conclusiones

Se puede concluir, en base a los resultados obtenidos, que la tecnología CUDA

- para problemas con dimensiones relativamente pequeñas y utilizando datos en doble precisión, no otorga mejora alguna con respecto a un desarrollo en CPU utilizando múltiples threads. Cuando las dimensiones comienzan a crecer la opción en CUDA resulta favorable dado por su escalabilidad natural;

- con respecto a los desarrollos en simple precisión, presenta mejoras más pronunciadas superando en general a aquellos resultados obtenidos por una versión con múltiples threads;

- presenta el problema fundamental de tener que adaptar el problema a la arquitectura (esto es con el objetivo de utilizar eficientemente la bondades de la misma), luego requiere de un gran esfuerzo para el logro de tareas similares en CPU (a excepción de operaciones triviales);

- dada por la escasez de registros por thread, posee una baja ocupación (en términos de warps activos por SM) que esta directamente relacionado con operaciones de gran latencia, limitando así su capacidad de dispositivo de cálculo masivo. Es por ello que la confección de stencils resulta más costosa que la posterior etapa de cálculos.

Finalmente

- considerando propuestas de esquemas sencillos como ser diferencias finitas en grillas estructuradas, cuya formulación dada por la localidad de los datos y el trivial acceso entre vecinos contiguos (al contrario que con esquemas no estructurados, clásicos de elementos finitos) debiera tener la posibilidad de explotar al máximo el procesamiento masivo de una GPU, se ha visto sin embargo que la escasez de registros impide ese resultado y pone en duda la utilización de esta arquitectura para la mecánica de fluidos computacional;
- extrapolando a esquemas más complejos (como podría ser la utilización de elementos finitos con datos en doble precisión), no se esperan resultados satisfactorios (en comparación a los obtenidos por un CPU), al menos con la arquitectura Tesla, aunque se espera que en un futuro esta tecnología madure y pueda sobrepasar aquellas dificultades expuestas a lo largo de este PFC...

Bibliografía

[BF00] Richard L. Burden and J. Douglas Faires. *Numerical analysis*. Thomson learning, 7th edition, 2000.

[CF92] Yiping Chen and Roger A. Falconer. Advection-diffusion modelling using the modified quick scheme. *International journal for numerical methods int fluids*, 1992.

[Dra02] Philip Drazin. *Introduction to hydrodynamic stability*. Cambridge, 2002.

[FP02] J. H. Ferziger and M. Perić. *Computational methods for fluid dynamics*. Springer, 3rd edition, 2002.

[HC00] Klaus A. Hoffmann and Steve T. Chiang. *Computational fluid dynamics : Volume I*. EESbooks, 4th edition, 2000.

[Hir07] Charles Hirsch. *Numerical computation of internal & external flows : Volume I*. Elsevier, 2th edition, 2007.

[JMN08] Sanjit Patel Jeroen Molemaker, Jonathan M. Cohen and Jonyong Noh. Low voscosity flow simulations for animation. *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2008.

[KmWH10] David B. Kirk and Wen mei W. Hwu. *Programming massively parallel processors: A hands-on approach*. Elsevier, 2010.

[Leo79] B. P. Leonard. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Computer Methods in Applied Mechanics and Engineering*, 1979.

[Loa92] Charles Van Loan. *Computational frameworks for the fast Fourier transform*. Siam, 1992.

[MASC10] Lisandro D. Dalcín Mario A. Storti, Rodrigo R. Paz and Santiago D. Costarelli. A fft preconditioning technique for the solution of incompressible flow with fractional step methods on graphic processing units. *Mecánica Computacional*, 2010.

[SK10] Jason Sanders and Edward Kandrot. *CUDA by example: An introduction to general-purpose GPU programming*. Addison-Wesley, 2010.