UNIVERSIDAD NACIONAL DEL LITORAL

DOCTORADO EN INGENIERÍA

Modelos numéricos en GPGPU para el tratamiento de fondos móviles erosionables

Lucas Carmelo Bessone Martínez

FICH
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

INTEC
INSTITUTO DE DESARROLLO TECNOLÓGICO PARA LA INDUSTRIA QUÍMICA

CIMEC
CENTRO DE INVESTIGACIÓN DE MÉTODOS COMPUTACIONALES

sinc(*i*)
INSTITUTO DE INVESTIGACIÓN EN SEÑALES, SISTEMAS E INTELIGENCIA COMPUTACIONAL

Tesis de Doctorado **2024**

UNIVERSIDAD NACIONAL DEL LITORAL
Facultad de Ingeniería y Ciencias Hídricas
Instituto de Desarrollo Tecnológico para la Industria Química
Centro de Investigación de Métodos Computacionales
Instituto de Investigación en Señales, Sistemas e Inteligencia Computacional

MODELOS NUMÉRICOS EN GPGPU PARA EL TRATAMIENTO DE FONDOS MÓVILES EROSIONABLES

Lucas Carmelo Bessone Martínez

Tesis remitida al Comité Académico del Doctorado
como parte de los requisitos para la obtención
del grado de
DOCTOR EN INGENIERÍA
Mención Mecánica Computacional
de la
UNIVERSIDAD NACIONAL DEL LITORAL

2024

Comisión de Posgrado, Facultad de Ingeniería y Ciencias Hídricas, Ciudad Universitaria,
Paraje “El Pozo”, S3000, Santa Fe, Argentina.

UNIVERSIDAD NACIONAL DEL LITORAL
Facultad de Ingeniería y Ciencias Hídricas
Instituto de Desarrollo Tecnológico para la Industria Química
Centro de Investigación de Métodos Computacionales
Instituto de Investigación en Señales, Sistemas e Inteligencia Computacional

MODELOS NUMÉRICOS EN GPGPU PARA EL TRATAMIENTO DE FONDOS MÓVILES EROSIONABLES

Lucas Carmelo Bessone Martínez

Lugar de Trabajo:

CIMEC
Centro de Investigación de Métodos Computacionales

Facultad de Ingeniería y Ciencias Hídricas
Universidad Nacional del Litoral

Director:

Mario A. Storti CIMEC, UNL

Co-director:

Pablo Gamazo CENUR LN, UDELAR

Jurado Evaluador:

Carlos Vionnet UNL
César Aguirre UNER
Sergio Preidikman UNC

2024

ACTA DE EVALUACIÓN DE TESIS DE DOCTORADO

En la sede de la Facultad de Ingeniería y Ciencias Hídricas de la Universidad Nacional del Litoral, a los veintisiete días del mes de marzo del año dos mil veinticuatro, se reúnen en forma online sincrónica los miembros del Jurado designado para la evaluación de la Tesis de Doctorado en Ingeniería, Mención Mecánica Computacional, titulada ***"Modelos numéricos en GPGPU para el tratamiento de fondos móviles erosionables"***, desarrollada por el Ing. Lucas Carmelo BESSONE MARTÍNEZ, DNI Nº 32.505.831 bajo la dirección del Dr. Mario Storti y la codirección del Dr. Pablo Gamazo. Ellos son: Dr. Carlos Vionnet, el Dr. César Aguirre y el Dr. Sergio Preidikman.----------

La Presentación oral y defensa de la Tesis se efectúa bajo la modalidad online sincrónica según lo establecido por Resolución CS N° 382/21.

Luego de escuchar la Defensa Pública y de evaluar la Tesis, el Jurado considera:

Que el trabajo realizado presenta un tema de interés muy actual para la simulación en lechos de ríos. Aborda problemas de interacción fluido-lechos erosionables para simular el transporte de sedimentos abordados con estas herramientas de gran poder de cómputo en paralelo como el que propone la arquitectura de Placas Gráficas GPGPU. Los aspectos técnicos y algorítmicos fueron tratados con gran detalle en esta tesis.
La presentación ha sido clara y concisa demostrando el dominio de los temas teóricos y de los desarrollos concretados a lo largo de la tesis
Durante la defensa el tesista respondió con solvencia las preguntas del jurado

Por lo tanto, el Jurado dictamina aprobar la tesis con calificación 10 (diez) Sobresaliente.

Sin más, se da por finalizado el Acto Académico con la firma de los miembros del Jurado al pie de la presente. ----------

Dr. Carlos Vionnet	Dr. César Aguirre	Dr. Sergio Preidikman

FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS * SANTA FE *
DR. JOSÉ LUIS MACOR
SECRETARIO DE POSGRADO
Facultad de Ingeniería y Cs. Hídricas

Universidad Nacional del Litoral
Facultad de Ingeniería y
Ciencias Hídricas

Secretaría de Posgrado

Ciudad Universitaria
C.C. 217
Ruta Nacional Nº 168 – Km. 472,4
(3000) Santa Fe
Tel: (54) (0342) 4575 229
Fax: (54) (0342) 4575 224
E-mail: posgrado@fich.unl.edu.ar

DECLARACIÓN LEGAL DEL AUTOR

Agradecimientos

Quiero expresar mi más sincero agradecimiento a las siguientes personas e instituciones:

A mi Director, Mario Storti, por sus aportes enriquecedores, su confianza, predisposición, dedicación y disponibilidad siempre.

A mi Co-Director, Pablo Gamazo, por confiar en mi capacidad de trabajo y por generar siempre instancias de discusión y análisis de gran valor.

A la Facultad Regional Concordia de la Universidad Tecnológica Nacional, que a través del programa *"Becas de formación de doctores para fortalecer las áreas de I+D+i, Res. C.S. No. 1460/15"*, brindó soporte durante todo el desarrollo de esta tesis.

A la Universidad Nacional del Litoral y Facultad de Ingeniería y Ciencias Hídricas y todos sus docentes que permitieron brindarme la posibilidad de realizar estos estudios de posgrado de primer nivel.

Al Departamento del Agua y el Departamento de Matemática y Estadística del Litoral, pertenecientes al Centro Universitario Regional Litoral Norte de la Universidad de la República y sus respectivos integrantes por brindar su apoyo.

Al grupo de trabajo del CIMEC con quienes pude compartir experiencias durante el cursado y desarrollo de la tesis.

Por último, quiero agradecer a quienes están a mi lado independientemente de lo que me proponga, familiares, amigos y especialmente, Agustina.

Resumen

En la última década, el cómputo científico en GPU ha demostrado ser una excelente alternativa para la computación de alto desempeño. Con una mejora sustancial en términos de rendimiento, bajo costo y consumo de energía en comparación con un clúster mediano, los desarrollos en GPU han producido resultados exitosos en diferentes campos de la ingeniería como la dinámica de fluidos computacional. Sin embargo, para lograr un rendimiento óptimo y aprovechar las capacidades de la GPU, es necesario que las diferentes implementaciones estén diseñadas específicamente según el tipo de hardware.

En esta tesis se desarrollan y adaptan diferentes estrategias de paralelización en GPU, para esquemas numéricos en el contexto del método de los volúmenes finitos que permiten resolver, entre otros, el complejo problema de transporte de sedimentos y erosión localizada. Con los algoritmos y métodos expuestos se desarrolló una librería en GPU que incluye: rutinas para resolver sistemas lineales dispersos, utilizando los métodos de Krylov y técnicas multigrilla, métodos para resolver la ecuación de transporte con esquemas de Variación Total Decreciente (TVD), un solver que resuelve el flujo incompresible utilizando dos algoritmos (SIMPLE o pasos fraccionados), desarrollo de un método de fronteras embebidas (IBM) para tratar geometrías complejas, modelos para calcular descargas de sedimento y evolución temporal del lecho, que se acoplan al modelo hidrodinámico para poder simular problemas de erosión.

Las diferentes componentes desarrolladas se validan mediante la utilización de soluciones analíticas, con el objetivo de comprobar la convergencia numérica de los esquemas espaciales y temporales. También se lleva a cabo una verificación mediante el uso de benchmark, para asegurar la precisión de las implementaciones.

Con el fin de destacar los beneficios de las implementaciones basadas en GPU, se llevan a cabo evaluaciones utilizando diversas métricas para medir el rendimiento en problemas de diferentes tamaños. Se logran abordar simulaciones de problemas en el orden de las 100 millones de celdas, lo que demuestra que la capacidad de una GPU es comparable al rendimiento obtenido mediante el uso de múltiples CPU en paralelo. Esta capacidad para manejar grandes volúmenes de datos confirma la eficiencia y la escalabilidad de las soluciones basadas en GPU en comparación con las alternativas tradicionales.

Los problemas abordados en la tesis incluyen advección difusión utilizando esquemas lineales y no lineales para comparar el desempeño de técnicas implícitas y explícitas, los resultados brindan pautas en cuanto a elección de métodos para resolver estas ecuaciones. Se resolvió un problema de difusión no lineal, mostrando que el método implícito puede tener un buen desempeño, sin embargo el bajo requerimiento en memoria y relativa simplicidad en implementación para los métodos explícito lo hacen más atractivos al resolver este tipo de ecuaciones en GPU. Se expuso la solución de problemas advectivo-dominantes usando esquemas TVD demostrando que puede obtenerse un buen desempeño en GPU, permitiendo resolver problemas que requieren el transporte de gradientes pronunciados. Se analizan dos variantes para el solver de flujo incompresible lo que permite evaluar pros y contras al paralelizarlos en GPU. Se desarrolla un método IBM que combina técnicas de celdas fantasma y una estrategia de interpolación que permite mantener un esquema espacial compacto y la aplicación de diferentes condiciones de borde sobre el sólido inmerso.

Mediante una combinación adecuada de los métodos desarrollados, se logró resolver de manera satisfactoria un desafiante problema tridimensional de erosión localizada alrededor de un objeto

rectangular. Esta resolución se llevó a cabo mediante el uso exclusivo de una sola GPU, logrando completar el cálculo en tan solo 10 horas. Esta marca representa una mejora considerable en el tiempo de ejecución, siendo hasta 36 veces más rápido en comparación con un equipo de cómputo CPU equipado con 32 núcleos.

La solución se obtuvo a utilizando el solver desarrollado para el flujo incompresible acoplado con un modelo de transporte de sedimentos. Esta integración permitió simular y comprender con precisión el proceso de erosión en un entorno tridimensional, proporcionando así una valiosa herramienta para estudios relacionados dentro de esta disciplina.

Esto proporciona una herramienta de bajo costo que resuelve problemas con grandes requisitos computacionales en tiempos reducidos. Los códigos desarrollados son adaptables y pueden ser utilizados en una gama de problemas. Las simulaciones numéricas realizadas con esta herramienta contribuyen a complementar las formulaciones clásicas y ofrecen una perspectiva más completa y mejorada de los fenómenos físicos, contribuyendo así al avance del conocimiento en las respectivas disciplinas.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

La descripción matemática del modo en como se transportan las partículas sólidas en una corriente líquida es sumamente compleja. El transporte sólido en los ríos (ya sea por arrastre de fondo y/o en suspensión) es de difícil determinación porque se tiene gran variabilidad en los fenómenos tanto en el espacio como en el tiempo, un elevado número de variables intervinientes y dificultad de comprobar en la naturaleza los resultados que se obtienen. Entre los problemas vinculados al transporte de sedimentos se encuentra la sedimentación y erosión localizada alrededor de pilas y estribos de puentes. Los daños vinculados a estos casos, se pueden reducir en la medida que se pueda predecir el transporte sólido. Debido al complejo carácter tridimensional del fenómeno de erosión localizada y las formas en el fondo del lecho del río, es necesario el uso de modelos de flujo que sean tridimensionales lo cual puede dar lugar a problemas computacionalmente muy demandantes.

1.2. Uso de la GPU para el cálculo científico

Con el paso de los años los cambios en hardware han permitido el desarrollo de modelos con mayor resolución y complejidad. En la última década las GPUs han surgido como una alternativa para computación de propósito general y su uso para cálculo científico ha ido aumentando [32, 171, 183]. Las arquitecturas GPU se caracterizan por la gran capacidad de cómputo en relación al ancho de banda de memoria. Esto las hace muy buenas para resolver discretizaciones temporalmente explícitas y espacialmente compactas [104]. Por otro lado se deben tener en cuenta los aspectos del modelo de programación CUDA (Computed Unified Device Architecture) que resultan críticos para la naturaleza del ancho de banda en los métodos utilizados para reslver las ecuaciones discretizadas como por ejemplo Navier Stokes (NS). El modelo de ejecución adoptado por la GPU para el cómputo en paralelo es el de Single Instruction Multiple Threads (SIMT) [102], esto implica dividir el procesamiento de una malla en diferentes funciones (CUDA kernels), que agrupan un conjunto de instrucciones que se pasan a la GPU de modo que cada proceso o hilo de la GPU ejecute la misma instrucción pero sobre un gran conjunto de datos [167]. En cuanto a la discretización temporal, en general se prefieren los métodos implícitos debido a su mayor estabilidad a lo largo de la evolución del problema ya que son capaces de utilizar pasos de tiempo mucho más largos que los esquemas explícitos. Sin embargo, las características del modelo SIMT en las GPU, hacen que el uso de métodos explícitos sea atractivo desde el punto de vista computacional, aunque requieren considerar un paso de tiempo mucho más pequeño [104]. Por otra parte la mayoría de los trabajos que resuelven numéricamente ecuaciones diferenciales mediante discretización (por ejemplo FVM), se centran en desarrollar implementaciones eficientes de resolvedores de los sistemas de ecuaciones algebraicas generados por diferentes métodos numéricos [25, 41, 77]. Pero para obtener un buen desempeño en aplicaciones CFD usando GPU es crucial trabajar preferentemente

en códigos basados completamente en GPU. Como las técnicas de discretización y solución afectan los rendimientos de manera considerable, esto motiva la implementación de varios esquemas numéricos en GPU.

1.3. Resolvedores para CFD en GPUs

En base al modelo de programación CUDA, es preferible el uso de grillas cartesianas fijas, dentro de ellas, las grillas colocadas son las más populares. Los métodos más frecuentes encontrados en la literatura para la resolución numérica de las ecuaciones de NS en flujos incompresibles en GPU, son el método de proyección [161, 167, 189] propuesto originalmente por [37], y el algoritmo SIMPLE (Semi-Implicit Method for Pressure Linked Equations)[80, 127, 182] presentado por [136]. No se ha encontrado una comparación del desempeño en GPU para ambos algoritmos, esto motiva a realizar un breve análisis del rendimiento de dos implementaciones basadas en GPU de los mismos.

1.4. Simulación numérica para transporte de sedimentos

La simulación numérica se utiliza como herramienta adicional para reducir los esfuerzos utilizados para el diseño y experimentación. Para ello se requieren resolvedores rápidos tanto para el fluido como para la modelación del sedimento. En general el transporte de sedimentos se trata en dos partes, el transporte por carga de fondo y el transporte por carga en suspensión [15, 48, 49]. El transporte de fondo comprende el material transportado cerca de la superficie del lecho, mientras que el resto del material se transporta en el cuerpo del fluido en forma suspendida. En la ecuación para el transporte de fondo, se determina la superficie del sedimento en base al equilibrio de masa que entra y sale [53]. Similarmente a partir de la conservación de masa se obtiene una ecuación de advección difusión (AD) que permite modelar el transporte de una concentración en suspensión [144]. El parámetro de entrada en ambos sistemas es la velocidad del flujo que corresponde a una velocidad de transporte y de acuerdo al material transportado, la altura de superficie del lecho evoluciona en el tiempo.

1.5. Fórmulas de transporte

El gasto sólido de fondo y gasto sólido en suspensión usualmente se relacionan con el caudal que escurre en el río. Las fórmulas de transporte que determinan la carga de sedimento transportado en la modalidad de fondo a cierta velocidad es un tema extensamente investigado durante décadas [51, 113, 153, 176]. Muchos autores coinciden en que los granos en el fondo comenzarán a moverse cuando la tensión τ_b exceda un valor crítico (τ_c). Por ejemplo una fórmula muy popular, verificada con datos para arena gruesa y grava uniformes es la fórmula de [113], que fue re-analizada por [180] quienes encontraron un mejor ajuste a dichos datos, dado por la siguiente fórmula:

$$q^* = 4{,}93\,(\tau^* - 0{,}047)^{1{,}06}$$

siendo q^* la carga de fondo adimensionalizada y τ^* el esfuerzo de corte adimensionalizado, en el fondo (lecho) generado por el flujo circulante. Nótese que el esfuerzo de corte crítico adimensionalizado τ_c^* adopta el valor $0{,}047$ en esta fórmula.

En la práctica, el esfuerzo en el fondo τ_b no se puede medir, pero en la literatura pueden encontrarse varias fórmulas empíricas para estimarlo. Sin embargo, debido a que en la implementación numérica se dispone de los gradientes de velocidad, el esfuerzo cortante podría calcularse explícitamente como $\tau = \mu \partial u / \partial n$.

1.6. Acoplamiento del flujo con transporte de sedimentos

Existen diferentes enfoques para realizar el acoplamiento de las ecuaciones de NS, la ecuación de conservación de la masa de sedimento en el lecho (ecuación de Exner) y la ecuación de AD para el transporte en suspensión por ejemplo [57, 83, 149]. Los acoplamientos explícitos pueden representar inestabilidades, pero pueden resultar sencillos en cuanto a implementación en GPU.

Por otra parte un acoplamiento fuerte (implícito o semi-implícito) que resulta estable, requiere la solución simultánea de los sistemas, cosa que podría llevarse a cabo en conjunto dentro de la implementación del solucionador de NS en un esquema segregado.

Sin embargo, como se espera que las escalas temporales involucradas en el transporte de sedimentos y en los cambios morfológicos sean muy diferentes respecto de los tiempos de cambio en el flujo. Entonces, puede resultar factible un desacoplamiento entre los modelos dentro del algoritmo conjunto.

1.7. Objetivos

1.7.1. Objetivo general

El objetivo general de esta tesis es estudiar e implementar algoritmos para resolver problemas de dinámica de fluidos computacional (CFD) orientados al transporte de sedimentos, utilizando tarjetas gráficas del tipo GPGPU (General Purpose Graphics Processing Units).

1.7.2. Objetivos específicos

Como objetivos específicos dentro del alcance de esta tesis se propone:

- Implementar métodos del tipo Volúmenes Finitos (FVM) en mallas cartesianas uniformes.
- Implementar un resolvedor para calcular el flujo incompresible a través de un algoritmo que trate el acoplamiento presión-velocidad.
- Resolver el transporte de sedimento usando fórmulas estándar de carga de fondo.
- A partir de las descargas de sedimentos resolver la ecuación de Exner acoplada con las ecuaciones de NS y el modelo de carga de fondo.

1.7.3. Contribuciones esperadas

Como resultado se espera la implementación de un modelo de transporte de sedimentos completo acoplado con un resolvedor para flujos incompresibles. Con los modelos descritos, se podría adaptar el código fácilmente a una gran variedad de problemas y condiciones de campo. Los resultados de las simulaciones numéricas contribuirán a complementar las formulaciones clásicas y permitirán una mejor comprensión de los fenómenos físicos involucrados.

1.8. Estructura de la Tesis

La estructura de la tesis es dividida en un total de siete capítulos. En el Capítulo 1 (Introducción), se exponen las principales motivaciones que movilizan la realización de la presente Tesis y una introducción al estado del arte en el cálculo científico en GPUs y la simulación numérica de flujos y transporte de sedimentos. En el Capítulo 2 se describen las ecuaciones y algoritmos utilizados en la dinámica de fluidos computacional, los esquemas de discretización espacial y temporal en el contexto del método de los volúmenes finitos. En el Capítulo 3 se presentan las principales características del modelo de programación CUDA y se presentan algunos resultados en cuanto a

desempeño de métodos y algoritmos implementados en GPU aplicados a la resolución de diferentes problemas de CFD. En el Capítulo 4 se desarrolla un método multigrilla basado en GPU para reducir el tiempo de cómputo en las ecuaciones de NS. En el Capítulo 5 se presenta y valida una formulación basada en GPU para el método de fronteras embebidas que permite resolver en fronteras complejas. En el Capítulo 6 se describen los modelos utilizados para predecir el transporte de sedimentos, se desarrolla una estrategia para acoplar los modelos hidrodinámico y morfodinámico, y se realiza una aplicación del programa desarrollado en GPU en un caso de estudio de la erosión localizada en sobre un objeto rectangular. Finalmente en el Capítulo 7 se presentan las conclusiones y aportes principales del trabajo realizado en esta Tesis, así como también las líneas abiertas para el desarrollo de trabajos futuros.

1.9. Publicaciones

A continuación se listan las publicaciones que ha realizado el autor de esta tesis, incluyendo solamente aquellas que derivaron directamente de la realización de la misma.

1.9.1. Publicaciones en revistas

- **Bessone, L.**, Gamazo, P., Dentz, M., Storti, M., & Ramos, J. (2022). GPU implementation of Explicit and Implicit Eulerian methods with TVD schemes for solving 2D solute transport in heterogeneous flows. Computational Geosciences, 26(3), 517-543.

1.9.2. Publicaciones y presentaciones en congresos

- **Bessone, L. C.**, Gamazo, P., & Storti, M. A. (2018). Evaluación del Desempeño de Diferentes Esquemas temporales para la Resolución de una Ecuación de Difusión No Lineal en GPGPU. *Mecánica Computacional, 36*(14), 605-625.
- **Bessone, L.**, Gamazo, P., & Storti, M. A. (2019). Evaluación del Desempeño de Dos Métodos para Resolver Flujos Incompresibles en GPGPU. *Mecánica Computacional, 37*(16), 623-623.
- **Bessone, L.**, Gamazo, P., Ramos, J., & Storti, M. (2020, May). Performance Evaluation of different time schemes for a Nonlinear diffusion equation on multi-core and many-core platforms. In *EGU General Assembly Conference Abstracts* (p. 1632).
- **Bessone, L.**, Gamazo, P., Dentz, M., Storti, M., Ezzatti, P., & Ramos, J. (2020, December). An efficient GPU solver for highly heterogeneous flows. In *AGU Fall Meeting Abstracts* (Vol. 2020, pp. H196-0004).
- **Bessone, L.**, Gamazo, P., Storti, M., Saracho, A., Ramos, J., Alvareda, E., ...& Paskosky, P. (2022). Uso de placas de video para predecir la erosión local. *XXX Congreso Latinoamericano de Hidraulica*. Foz do Iguazú. Brasil 2022.
- **Bessone, L.**, Gamazo, P., Storti, M., Dentz M. & Ramos, J. (2023) Implementación en GPU de un método multigrilla centrado en celda. *Mecánica Computacional, 40*(40), 1481-1495.
- **Bessone, L.**, Gamazo, P., Storti, M., Ramos, J., Saracho, A. & Alvareda, E. (2023). Una herramienta basada en GPU para simular procesos de erosión localizada. *Mecánica Computacional, 40*(40), 1497-1497.
- Saracho, A., **Bessone, L.**, Gamazo, P., Storti, M., Paskosky, P. & Navas, R. (2023). Parallel Resolution Techniques for the 2D Transport Equation: Comparison of Explicit and Implicit Methods. *Mecánica Computacional, 40*(40), 1499-1499.

Capítulo 2

Ecuaciones de gobierno y métodos para CFD

Tanto los balances de materia como los principios de la mecánica, son un requisito para los cálculos en la solución de problemas de la física e ingeniería. Las leyes de conservación sirven como una fuerte restricción en cualquier teoría sobre cualquier rama de la ciencia. Las leyes físicas que controlan estos principios se traducen en relaciones matemáticas, escritas en forma de ecuaciones diferenciales, lo cual representa el medio necesario para realizar simulaciones. Los principios fundamentales más conocidos que conforman estas leyes son: conservación de la masa (ecuación de continuidad), conservación del momento (o conservación de la cantidad de movimiento), conservación de la energía total del sistema. Una de las ecuaciones analizadas en esta tesis es la ley de conservación de una propiedad intensiva, esta ecuación general de transporte puede tener carácter escalar o vectorial.

2.1. Ecuación de transporte

La ecuación de transporte escalar en forma diferencial, considerando únicamente los mecanismos de transporte por advección y difusión está representada por la siguiente ecuación en derivadas parciales:

$$\frac{\partial C}{\partial t} + \nabla\cdot(\mathbf{u}C - \mathbf{D}\nabla C) = f \tag{2.1}$$

donde $C(\mathbf{x}, t)$ representa una propiedad intensiva generalizada que varía en forma continua y es función de la posición $\mathbf{x}$ y del tiempo t. Por ejemplo, C representa una concentración en [kg/m^3], $\mathbf{u}(\mathbf{x}) = (u_x, u_y, u_z)$ representaría un campo de velocidades dado en [m/s], $\mathbf{D}$ representa el tensor de difusión [m^2/s] y f un término fuente o sumidero [kg/m^3/s].

2.1.1. Discretización espacial

La mayoría de métodos utilizados para transformar (2.1) en un sistema de ecuaciones algebraicas son el método de los elementos finitos (FEM), el método de diferencias finitas (FDM) y el método de los volúmenes finitos (FVM). En esta tesis se resuelven las ecuaciones usando FVM centrado en celdas considerando grillas cartesianas uniformes, resultando equivalente al FDM. Sin pérdida de generalidad, se consideran siempre dominios prismáticos cuadrangulares $\Omega \subset \mathbb{R}^3$ (o rectangulares en $\mathbb{R}^2$) discretizados con paso de malla constante ($\Delta x = \Delta y = \Delta z = h$). En lo que sigue, la celda central y sus celdas vecinas se designan con el subíndice C y E, W, N, S, T, B respectivamente, haciendo alusión a *Center, East, West, North, South, Top, Bottom* correspondientes a vecinos en las 3 direcciones (x-dir, y-dir y z-dir). El subíndice en minúscula indica los centros de caras de las celdas, la figura 2.1 presenta la molécula computacional para el caso 2D y 3D.

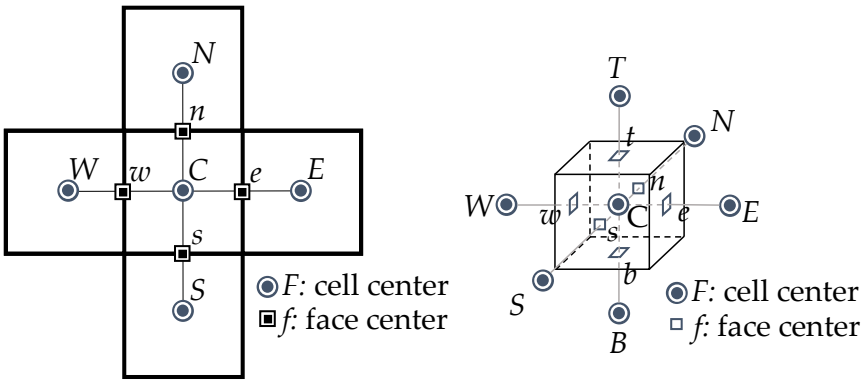

Figura 2.1: Molécula computacional para el caso 2D.

Si se integra miembro a miembro (2.1) en la celda C, se aplican el teorema del valor medio, el teorema de la divergencia y se re ordenan los términos se obtiene:

$$\begin{aligned}
\int_{V_C} \frac{\partial C}{\partial t} dV + \int_{V_C} \nabla\cdot(\mathbf{u}C - \mathbf{D}\nabla C) dV &= \int_{V_C} f dV \\
\int_{V_C} \frac{\partial C}{\partial t} dV + \int_{\partial V_C} (\mathbf{u}C - \mathbf{D}\nabla C)\cdot \mathbf{dS} &= \int_{V_C} f dV \\
\left(\frac{\partial C}{\partial t}\right)_C V_C - \sum_{f\sim nb(C)} \mathbf{D}_f \nabla C_f \cdot \mathbf{S}_f + \sum_{f\sim nb(C)} \dot{m}_f C_f &= f_C V_C
\end{aligned} \tag{2.2}$$

donde $V_C = h^3$ (h^2 en caso 2D) es el volumen de la celda y $\dot{m}_f = \mathbf{u}_f \cdot \mathbf{S}_f$ es la componente del flujo de masa que cruza en dirección perpendicular la cara f de la celda medido en volumen, esto es, sin considerar la densidad del fluido para simplificar en el caso de flujos incompresibles donde ($\rho \sim$ const) y $\mathbf{S}_f$ es el vector normal saliente a la celda C de magnitud igual al área de la cara (h^2 en 3D y h en 2D). La discretización espacial consiste en aproximar el operador gradiente y los valores de las incógnitas en las caras C_f, de esta manera, al considerar todas las celdas del dominio, el problema diferencial (2.1) se transforma en un sistema de ecuaciones algebraicas donde las incógnitas son los valores en cada centro de celda.

El primer término se tratará más adelante en la discretización temporal, para el segundo término, término difusivo, es suficiente usar el esquema de diferencias centradas (CD) $\nabla C_f = (C_F - C_C)/h$, si se requiere una precisión de segundo orden (orden $\sim O\left(h^2\right)$), donde el subíndice F indica alguno de los centros de celda vecinos ($F = E, W, N, S, T, B$, indicación abreviada en lo que sigue como $F \sim NB(C)$). Mientras que para el tercer término se requieren buenas estimaciones para obtener estabilidad numérica y precisión en los resultados, especialmente en problemas advectivo-dominantes. Así por ejemplo, los dos esquemas más simples son $\dot{m}_f C_f = \dot{m}_f (C_F + C_C)/2$ (esquema CD, orden $\sim O\left(h^2\right)$) o el esquema *upwind* (esquema UD, orden $\sim O\left(h\right)$), que en el caso de un campo de velocidades no uniforme se expresa como:

$$\dot{m}_f C_f = ||\dot{m}_f, 0|| C_C - || - \dot{m}_f, 0|| C_F \tag{2.3}$$

donde usamos $||a, b||$ para indicar $\max(a, b)$.

Estos esquemas consideran una combinación lineal de los valores en centros de celdas y además como en el segundo caso se elige de acuerdo a la dirección del flujo, esto lo hace un esquema estable en problemas advectivo-dominantes, sin embargo es una aproximación de primer orden y suaviza la solución excesivamente al generar una difusión numérica igual a $|\mathbf{u}_f|h/2$.

Por otro lado el conocido teorema de Godunov establece que: *Los esquemas numéricos lineales para resolver ecuaciones en derivadas parciales, que tienen la propiedad de no generar nuevos*

extremos (esquema monótono), pueden tener, como mucho, precisión de primer orden. Esto es, si se necesita un esquema de segundo orden o más de precisión, para que se conserve la monotonía, el mismo debe ser no lineal. Luego, la solución de C_f depende de los valores de C_C y sus vecinos C_F (con $F \sim NB(C)$) no linealmente como mostraremos más adelante. En la próxima subsección se repasará brevemente los esquemas de alta resolución para aproximar el término advectivo en la ecuación (2.2).

2.1.2. Esquemas TVD o de variación total decreciente

Las interpolaciones de alto orden (HO) que consideran la dirección contra corriente (upwind) llenan los vacíos en la precisión del esquema UD. Pero tales enfoques, de acuerdo con el teorema de Godunov [65], generalmente proveen soluciones no acotadas, las cuales pueden ser particularmente problemáticas en campos de velocidad no uniformes o flujos altamente heterogéneos. Un esquema HO acotado, conocido como esquema de alta resolución (HR), se obtiene imponiendo la condición TVD (*Total Variation Dimishing*) que se explica a continuación.

Considerando por simplicidad una grilla unidimensional, la variación total (TV) se define como $TV(C) = \sum_i |C_{i+1} - C_i|$ donde i es el índice de un nodo de la grilla. Se dice que un esquema es TVD si TV no crece a lo largo del tiempo:

$$TV(C^{t+\Delta t}) \leq TV(C^t) \tag{2.4}$$

Un esquema HO monótono es TVD y a su vez un esquema TVD preserva la monotonía [71]. Se pueden usar las funciones limitadoras (limitador o limitador de flujo) para construir un esquema TVD. Tales limitadores previenen la aparición de oscilaciones no físicas. Si se utiliza el enfoque de Sweby [162], llamando al limitador $\psi(r)$ donde r indica la relación entre dos gradientes consecutivos, el valor en el centro de cara puede obtenerse de una manera simple:

$$C_f = C_C + \frac{1}{2}\psi(r_f)\,(C_D - C_C) \;\; ; \text{ with } r_f = \frac{C_C - C_U}{C_D - C_C} \tag{2.5}$$

donde el subíndice D y U indica los nodos *upwind* y *downwind*, por lo tanto deben elegirse de acuerdo a la dirección en que la velocidad $\mathbf{u}_f$ atraviesa el centro de la cara f de la celda C. Por ejemplo, los flujos correspondientes a las caras *east* y *west* resultan respectivamente en:

$$\dot{m}_e C_e = \left[C_C + \frac{1}{2}\psi(r_e^+)\,(C_E - C_C)\right] ||\dot{m}_e, 0|| - \left[C_E + \frac{1}{2}\psi(r_e^-)\,(C_C - C_E)\right] || - \dot{m}_e, 0||;$$
$$\text{con } r_e^+ = \frac{C_C - C_W}{C_E - C_C},\; r_e^- = \frac{C_E - C_{EE}}{C_C - C_E} \tag{2.6}$$

$$\dot{m}_w C_w = \left[C_C + \frac{1}{2}\psi(r_w^+)\,(C_W - C_C)\right] ||\dot{m}_w, 0|| - \left[C_W + \frac{1}{2}\psi(r_w^-)\,(C_C - C_W)\right] || - \dot{m}_w, 0||;$$
$$\text{con } r_w^+ = \frac{C_C - C_E}{C_W - C_C},\; r_w^- = \frac{C_W - C_{WW}}{C_C - C_W} \tag{2.7}$$

donde el doble subíndice C_{EE} (respectivamente C_{WW}) se utiliza para indicar el centro de celda vecino a dos celdas de distancia de la celda C en esa dirección, esto es, la celda EE se ubica al este de la celda E (resp. WW al oeste de la celda W).

Se han desarrollado muchos esquemas TVD de esta forma, eligiendo apropiadamente el limitador $\psi(r_f)$ (OSHER [31], MUSCL [175], SUPERBEE y MINMOD [145] entre otros). Nótese que siempre que $\psi(r_f)$ permite generar un esquema TVD, C_f será una combinación no lineal de los valores en los centros de celda, ya que r_f es una función de C_C y sus vecinos C_F. En otros casos, el esquema será lineal, pero no TVD, como es el caso de los esquemas de alto orden o

HO, por ejemplo: esquema CD ($\psi(r_f) = 1$), esquema *second order upwind*- SOU ($\psi(r_f) = r_f$) o incluso el esquema QUICK [99] ($\psi(r_f) = (r_f + 3)/4$). Esto crea una complicación para los métodos implícitos debido a que requerirían una estrategia iterativa para resolver el sistema no lineal que se genera.

2.1.3. Enfoque de Corrección Diferida (DC)

A lo largo de esta tesis, se utilizan esquemas temporales explícitos e implícitos. Para los métodos explícitos, la función limitadora para la linealización del esquema advectivo $\psi(r_f)$ puede calcularse usando los valores de la solución temporal previa C_C^t y C_F^t. Para los esquemas implícitos, esto implica que los coeficientes fuera de la diagonal en el sistema de ecuaciones tienen signos opuestos, violando así una regla básica para la estabilidad de un algoritmo iterativo [121]. Aunque lo anterior se puede evitar si se utiliza un método directo para la solución del sistema, tal situación no es nada práctica en discretizaciones con millones de celdas.

Para tratar el problema mencionado, en los algoritmos implícitos se utiliza la técnica de corrección diferida [82]. La estrategia es tratar el término advectivo como sigue:

$$\dot{m}_f C_f^{HR} = \underbrace{\dot{m}_f C_f^U}_{\text{implicit}} + \underbrace{\dot{m}_f \left(C_f^{HR} - C_f^U\right)}_{\text{explicit}} \tag{2.8}$$

donde los supraíndices U y HR indican el uso de los esquemas upwind y TVD respectivamente (es decir, usar (2.3) para U y (2.5) para HR). La idea entonces es que el valor en centro de cara se trata implícitamente considerando el esquema de primer orden. La diferencia entre los esquemas upwind y TVD se trata de forma explícita, calculando los valores basándose en la última solución disponible, esto es, la obtenida en la iteración previa dentro de un proceso iterativo. La ventaja de este enfoque es que se obtiene una matriz diagonalmente dominante en el esquema temporal implícito (cosa deseable en los resolvedores iterativos, como los métodos de Krylov), mientras que la no linealidad del esquema TVD es tratada explícitamente.

2.2. Discretización temporal

En esta sección se describen los principales esquemas temporales utilizados en la tesis. Para ello, se considerará sin pérdida de generalidad el sistema de ODEs:

$$\frac{\partial \mathbf{C}}{\partial t} = \mathbf{f}(\mathbf{C}, t), \text{ con } \mathbf{f}(\mathbf{C}, t) : \mathbb{R}^{m+1} \to \mathbb{R}^m \text{ una función lineal o no lineal de } \mathbf{C} \tag{2.9}$$

siendo $\mathbf{C} \in \mathbb{R}^m$ el vector solución para todo el dominio computacional y $\mathbf{f}$ representa la expresión resultante luego de aplicar una discretización espacial en el contexto de alguno de los métodos usuales (FEM, FDM o FVM), que para esquemas lineales se puede expresar como el producto de una matriz por el vector solución $\mathbf{f} = \mathbf{BC}$. En lo que sigue se aplicará la notación utilizando supraíndices de forma que C_C^n corresponde a la variable evaluada en el centro de celda C en tiempo actual $t = n\Delta t$.

2.2.1. Métodos explícitos

A continuación, se resumen los esquemas de discretización explícitos más conocidos con sus respectivos errores de truncamiento y algunas consideraciones a tener en cuenta en el contexto de esta tesis.

- **Euler hacia adelante - FE**. Este esquema de diferencias finitas es de primer orden en el tiempo (orden $\sim O(\Delta t)$):

$$\frac{\mathbf{C}^{n+1} - \mathbf{C}^n}{\Delta t} = \mathbf{f}(\mathbf{C}^n, t) + O(\Delta t) \tag{2.10}$$

constituye uno de los métodos más simples y adecuado para implementar en GPU debido a que solo se requieren almacenar dos vectores en memoria $\mathbf{C}^{n+1}$ y $\mathbf{C}^n$. La parte principal del código consiste en una función que actualiza las variables al avanzar cada paso de tiempo, sin embargo como se mencionó, es de primer orden de precisión.

- **Adams-Bashforth - AB**. Este esquema es parte de la familia de métodos multipaso y es de segundo orden en el tiempo:

$$\frac{\mathbf{C}^{n+1}-\mathbf{C}^n}{\Delta t} = \frac{3}{2}\mathbf{f}(\mathbf{C}^n,t) - \frac{1}{2}\mathbf{f}(\mathbf{C}^{n-1},t) + O(\Delta t^2) \tag{2.11}$$

como se mostrará mas adelante, en un problema de difusión pura resuelto con un esquema temporal explícito, no tiene mucho sentido mejorar el orden de precisión temporal, la fórmula resulta útil en otro tipo de problemas como ser: advección pura, advección difusión, o aplicado únicamente a uno de los términos de la ecuación diferencial. En cuando al contexto GPU, se requiere almacenar en memoria tres vectores, correspondientes a las soluciones futura, presente y pasada. La parte principal del código consistirá en una función que lee los dos vectores solución (n y $n-1$) y actualiza la solución.

- **Runge-Kutta 2 - RK2**. Otra familia de métodos explícitos de mayor orden la constituye los métodos tipo Runge-Kutta. Este esquema es de segundo orden en el tiempo y consiste en realizar las siguientes operaciones:

$$\begin{aligned}
\frac{\mathbf{C}^{n+1}-\mathbf{C}^n}{\Delta t} &= \mathbf{f}(\mathbf{C}^n,t) && \text{1er paso tipo FE}\\
\frac{\mathbf{C}^{n+2}-\mathbf{C}^{n+1}}{\Delta t} &= \mathbf{f}(\mathbf{C}^{n+1},t) && \text{2do paso tipo FE}\\
\mathbf{C}^{n+1} &= \frac{1}{2}\mathbf{C}^{n+2}+\frac{1}{2}\mathbf{C}^n && \text{etapa de corrección}
\end{aligned} \tag{2.12}$$

este método también se conoce como método de Heun o método predictor-corrector. Requiere almacenar una solución más que el esquema FE. Es muy fácil de implementar debido a que se aplica directamente la programación del método FE como *"caja negra"*.

- **Runge-Kutta 3 - RK3**. Este esquema es de tercer orden en el tiempo y consiste en realizar las siguientes operaciones:

$$\begin{aligned}
\frac{\mathbf{C}^{n+1}-\mathbf{C}^n}{\Delta t} &= \mathbf{f}(\mathbf{C}^n,t) && \text{1er paso tipo FE}\\
\frac{\mathbf{C}^{n+2}-\mathbf{C}^{n+1}}{\Delta t} &= \mathbf{f}(\mathbf{C}^{n+1},t) && \text{2do paso tipo FE}\\
\mathbf{C}^{n+1/2} &= \frac{3}{4}\mathbf{C}^n+\frac{1}{4}\mathbf{C}^{n+2} && \text{primer etapa de promediado}\\
\frac{\mathbf{C}^{n+3/2}-\mathbf{C}^{n+1/2}}{\Delta t} &= \mathbf{f}(\mathbf{C}^{n+1/2},t) && \text{avance a tiempo } t+3/2\Delta t \text{ (FE)}\\
\mathbf{C}^{n+1} &= \frac{1}{3}\mathbf{C}^n+\frac{2}{3}\mathbf{C}^{n+3/2} && \text{segunda etapa de promediado}
\end{aligned} \tag{2.13}$$

es el método usado mayoritariamente en los trabajos académicos debido a su precisión, fácil implementación y además es un método tipo TVD (en el tiempo) en el caso de advección pura. Se puede implementar como RK2 usando las rutinas FE como *"caja negra"* almacenando solamente una solución más extra respecto del método FE, es decir se requieren en total 3 vectores almacenados en memoria.

Cabe aclarar que el orden de precisión del esquema temporal que se menciona en los métodos surge al considerar el desarrollo de Taylor en el término transiente de (2.2), por ejemplo, el esquema FE para la derivada temporal en la celda C es:

$$\begin{aligned} C_C(t+\Delta t) &= C_C(t) + \frac{\partial C_C(t)}{\partial t}\Delta t + \frac{\partial^2 C_C(t)}{\partial t^2}\frac{\Delta t^2}{2!} + \dots \\ \Rightarrow \left(\frac{\partial C}{\partial t}\right)_C &= \frac{\partial C_C(t)}{\partial t} = \frac{C_C(t+\Delta t) - C_C(t)}{\Delta t} + O(\Delta t) = \frac{C_C^{n+1} - C_C^n}{\Delta t} + O(\Delta t) \end{aligned} \tag{2.14}$$

2.2.2. Métodos implícitos

A continuación, se resumen los esquemas implícitos considerados en este trabajo.

- **Euler hacia atrás - BE**. Este esquema de diferencias hacia atrás es de primer orden en el tiempo (orden $\sim O(\Delta t)$):

$$\frac{\mathbf{C}^{n+1} - \mathbf{C}^n}{\Delta t} = \mathbf{f}(\mathbf{C}^{n+1}, t) + O(\Delta t) \tag{2.15}$$

  debe notarse que la determinación del vector solución a tiempo futuro $\mathbf{C}^{n+1}$ surge de resolver un sistema de ecuaciones ya que la variable a tiempo futuro se encuentra dentro de la función $\mathbf{f}$. Es un método que en general resulta incondicionalmente estable, luego el paso de tiempo se puede elegir tan grande como se quiera, pero suele elegir de forma tal que el error temporal pese menos que la precisión del esquema espacial que se utilice.

- **Crank Nicolson - CN**. Este esquema de diferencias centradas es de segundo orden en el tiempo (orden $\sim O\left(\Delta t^2\right)$):

$$\frac{\mathbf{C}^{n+1} - \mathbf{C}^n}{\Delta t} = \frac{1}{2}\mathbf{f}(\mathbf{C}^{n+1}, t) + \frac{1}{2}\mathbf{f}(\mathbf{C}^n, t) + O(\Delta t^2) \tag{2.16}$$

  como en el método anterior, luego de ordenar de acuerdo a los valores disponibles (colocándolos en el lado derecho o RHS) y los valores incógnita (colocándolos en el lado izquierdo, LHS), debe resolverse el sistema de ecuaciones en cada paso de tiempo. Es un método marginalmente estable, puede presentar alguna inestabilidad cuando hay incopatibilidad en la condición inicial y la condición de borde a tiempo $t > 0$, sin embargo, se mantiene acotada (es estable). El costo computacional es el mismo que BE pero al tener segundo orden de precisión, $O(\Delta t^2)$, es el método implícito más utilizado. En el caso de difusión pura, se gana mucho repecto a un esquema explícito (que posee una fuerte restricción de estabilidad) ya que la precisión de este método permite elegir pasos temporales relativamente grandes manteniendo el error de truncamiento del esquema temporal por debajo del que presenta el esquema espacial.

Los métodos FE, BE y CN se pueden agrupar en un método conocido como **Método** θ.

$$\frac{\mathbf{C}^{n+1} - \mathbf{C}^n}{\Delta t} = (1-\theta)\mathbf{f}(\mathbf{C}^n, t) + \theta\mathbf{f}(\mathbf{C}^{n+1}, t) \tag{2.17}$$

de esta forma, cuando $\theta = 0$ corresponde al método explícito puro (FE), para $\theta = 1$ es un método implícito puro (BE) y para $\theta = 1/2$ es un método explícito-implícito (CN)[1].

[1] El término explícito-implícito es simplemente para indicar la contribución de ambas partes al ensamblar el sistema, sin embargo al momento de resolver se trata de un método implícito como BE.

2.2.3. Sobre la estabilidad de los esquemas temporales

Los métodos explícitos son más fáciles de implementar, además de que son más baratos computacionalmente (en cuanto a tiempo de cómputo por paso de tiempo), pero están sujetos a inestabilidades numéricas y son condicionalmente estables o inestables. El método clásico para determinar las condiciones que debe cumplir el paso de tiempo para que la evolución del problema se mantenga estable a lo largo del tiempo, es el método de Von Neuman[74].

Algunas definiciones que vale la pena mencionar son:

- **Consistencia:** es una condición sobre el *esquema numérico*, a saber, que el esquema numérico debe tender a la ecuación diferencial, cuando $(\Delta t, h) \to (0, 0)$ [74].

- **Estabilidad:** es una condición sobre la *solución numérica*, dígase, todos los errores, tales como los de redondeo por la aritmética finita de la computadora, deben mantenerse acotados al avanzar en el proceso iterativo. Esto es, para valores finitos de Δt y h, el error definido como la solución numérica y la solución exacta del esquema numérico debe mantenerse acotado cuando $n \to \infty$ (siendo n el número de pasos de tiempo). Esto involucra una condición sobre el esquema numérico y no sobre la ecuación diferencial [74].

- **Convergencia:** es una condición sobre la *solución numérica*, debe garantizarse que la salida de la simulación es una correcta representación del modelo que se está resolviendo, esto es, la solución numérica debe tender a la solución exacta del modelo matemático cuando $(\Delta t, h) \to (0, 0)$ [74].

Finalmente, el teorema de equivalencia de Lax[96] establece que:

Para un problema a valores iniciales bien planteado (bien puesto) y un esquema de discretización ***consistente****, la* ***estabilidad*** *es una condición necesaria y suficiente para la* ***convergencia****.*

En la práctica se puede obtener la condición de *estabilidad local* vía un análisis de Von Neumann, sin embargo muchas veces se debe obtener experimentalmente mediante pruebas numéricas, principalmente porque si la ecuación diferencial involucra muchos términos o se empiezan a combinar esquemas de discretización temporal, el método analítico se vuelve imposible de manipular.

Condiciones de estabilidad más conocidas

A continuación se repasan los criterios de estabilidad y números adimensionales usados frecuentemente en la combinación de esquemas.

- **Número de Fourier F_O:** el número adimensional que gobierna la estabilidad en el método explícito FE para el caso de difusión transiente (esquema espacial CD) para el caso unidimensional es:

$$F_O = \frac{2\kappa \Delta t}{h^2} \leq 1 \tag{2.18}$$

  donde κ es el coeficiente de difusividad [m^2/s]. Muchas veces se define sin el factor de 2 y en cuyo caso la condición de estabilidad es 1/2. Conceptualmente en un problema de calor, el número F_O es la relación dentre la velocidad de conducción del calor en el medio y la velocidad de almacenamiento de energía. Para el caso de difusión pura en 2D y 3D el número contiene un factor de 4 y 6 respectivamente, $F_O = 4\kappa\Delta t/h^2$ y $F_O = 6\kappa\Delta t/h^2$. Notar que el método FE y CD en conjunto es orden $O(h^2, \Delta t)$, con lo que habría que ver el mayor de ambos. Sin embargo como la condición $F_O \leq 1$ implica que $\Delta t \sim h^2$, no tendrían sentido los estudios de convergencia temporal. Por otro lado, al usar un esquema de Heun o RK2, la condición de estabilidad es similar a la anterior y utilizar dicho esquema temporal no aporta precisión ya que $O(\Delta t^2) \sim h^4 \ll h^2$ para $h \to 0$.

- **Número de Courant C_O:** el número adimensional que gobierna la estabilidad en el método explícito FE para el caso de advección pura transiente (esquema espacial UD) para el caso unidimensional es:

$$C_O = \frac{|v|\Delta t}{h} \leq 1 \tag{2.19}$$

donde v es la velocidad de advección [m/s]. El número de Courant establece que el paso de tiempo a usar no puede ser mayor al tiempo en que la concentración tarda en atravesar una celda a velocidad v. Observe que en este caso los dos esquemas están en el mismo orden de aproximación $\Delta t \sim h$.

- **Condición CFL:** la condición C_O también se conoce como condición CFL (Courant-Friedrich-Lewy), solo que esta última es más general, no solo para advección pura. Por ejemplo, para el problema de advección difusión transiente usando esquema FE, UD (advección) y CD (difusión) la condición CFL luce como:

$$CFL = \frac{|v|\Delta t}{h} + \frac{2\kappa\Delta t}{h^2} \leq 1 \tag{2.20}$$

Cabe mencionar que en caso de usar un esquema CD para la advección, además de la condición CFL, que es algo diferente a la presentada, hay que verificar que el número de Pèclet de grilla sea menor a 1, siendo $P_e = \frac{|v|h}{2\kappa}$ para el caso unidimensional y relaciona el transporte por advección y difusión.

2.2.4. Comentarios generales

Los métodos explícitos son más rápidos, principalmente para cómputo en paralelo, pero el criterio de estabilidad hace que la rapidez del método no compense la cantidad de pasos de tiempo (muy pequeños) requeridos para llegar a cierto instante de simulación.

Los métodos implícitos son muy costosos y complejos pero se compensan con el hecho de permitir pasos de tiempo grandes.

La mayoría de veces, para ver la ventaja real de los métodos explícitos se debe pensar en paralelismo de grano fino, esto es, número de procesadores de cómputo en el orden de 1000 o más como es el caso de las GPU de cómputo científico.

2.3. Ecuaciones no lineales - Método de Newton Raphson

El método de Newton Raphson (NR) es una de las estrategias más utilizadas para resolver numéricamente ecuaciones no lineales. El caso particular que interesa para este trabajo es cuando uno de los términos de la ecuación diferencial es no lineal y se resuelve numéricamente utilizando un esquema temporal implícito. Considere entonces, la ecuación (2.9) con $\mathbf{f}(\mathbf{C}, t)$ no lineal en las componentes de $\mathbf{C}^{n+1}$. El método de NR consiste en resolver iterativamente el sistema escrito en forma de residuo y linealizado:

$$\mathbf{R}\left(\mathbf{C}^{n+1}\right) \approx \mathbf{R}\left(\mathbf{C}^{(k)}\right) + \mathbf{J}\left(\mathbf{C}^{(k)}\right)\left(\mathbf{C}^{(k+1)} - \mathbf{C}^{(k)}\right) = 0 \tag{2.21}$$

definiendo $\Delta\mathbf{C}^{(k+1)} = \mathbf{C}^{(k+1)} - \mathbf{C}^{(k)}$, (2.21) puede expresarse como

$$\begin{aligned} \mathbf{R}\left(\mathbf{C}^{(k)}\right) + \mathbf{J}\left(\mathbf{C}^{(k)}\right)\Delta\mathbf{C}^{(k+1)} &= 0 \\ \Leftrightarrow \mathbf{C}^{(k+1)} = \mathbf{C}^{(k)} - \left(\mathbf{J}\left(\mathbf{C}^{(k)}\right)\right)^{(-1)} &\mathbf{R}\left(\mathbf{C}^{(k)}\right) \end{aligned} \tag{2.22}$$

donde $\mathbf{J}\left(\mathbf{C}^{(k)}\right)$ es el Jacobiano del residuo $\mathbf{R}$ evaluado en $\mathbf{C}^{(k)}$ que corresponde a la variable en tiempo futuro $\mathbf{C}^{n+1}$ en la k-ésima iteración.

Algoritmo 1: Algoritmo para el método de Newton Raphson con un esquema temporal implícito

1 $\mathbf{C}^{(0)} \leftarrow \mathbf{C}^n$; // $\mathbf{C}^n$ solución a tiempo pasado;
2 **para** $i = 1, \ldots$ *hasta la convergencia* **hacer**
3 computar $\mathbf{J}\left(\mathbf{C}^{(k)}\right)$ y ensamblar el sistema lineal (2.22);
4 resolver el sistema lineal para $\Delta\mathbf{C}^{(k+1)}$;
5 $\mathbf{C}^{(k+1)} \leftarrow \mathbf{C}^{(k)} + \Delta\mathbf{C}^{(k+1)}$;
6 **fin**
7 $\mathbf{C}^{n+1} \leftarrow \mathbf{C}^{(k+1)}$;

El algoritmo para resolver numéricamente la ecuación con un esquema temporal implícito, en cada paso de tiempo es el siguiente:

Es decir, que para cada paso de tiempo se deben ensamblar y resolver un sistema lineal K veces, uno por cada iteración NR, para poder obtener la solución en el paso de tiempo siguiente $\mathbf{C}^{n+1}$. Sin embargo, un detalle a tener en cuenta es que resolver el algoritmo iterativo de NR hasta la convergencia puede no ser necesario como se mostrará a continuación.

Orden de aproximación de la iteración NR al usar el esquema temporal CN

En vías de optimizar las implementaciones que se realicen, se mostrará que el orden de aproximación temporal para una iteración NR al usar el esquema de diferencias centradas CN es al menos $O(\Delta t^2)$. Esto implica que el nivel de precisión de la solución no cambia de orden con las sucesivas iteraciones de NR.

Para probar esto, considere el sistema (2.9) para el caso no lineal. Definiendo $\mathbf{z}^n = \mathbf{C}^{n+1} - \mathbf{C}^n$ y reemplazando en (2.16) se obtiene

$$\mathbf{z}^n = \frac{\Delta t}{2}\left[\mathbf{f}\left(\mathbf{C}^n + \mathbf{z}^n, t\right) + \mathbf{f}(\mathbf{C}^n, t)\right] + O(\Delta t^3) \tag{2.23}$$

escribiendo (2.23) de manera truncada y en forma de residuo:

$$\mathbf{R}(\mathbf{z}^n) = \mathbf{z}^n - \frac{\Delta t}{2}\left[\mathbf{f}\left(\mathbf{C}^n + \mathbf{z}^n, t\right) + \mathbf{f}(\mathbf{C}^n, t)\right] = 0 \tag{2.24}$$

de donde si se calcula el jacobiano del residuo queda:

$$\frac{\partial \mathbf{R}}{\partial \mathbf{z}^n} = \mathbf{I} - \frac{\Delta t}{2}\mathbf{J}_f\left(\mathbf{C}^n + \mathbf{z}^n, t\right) \tag{2.25}$$

siendo $\mathbf{I}$ la matriz identidad y $\mathbf{J}_f$ el jacobiano de la función no lineal $\mathbf{f}$ de (2.9). El esquema iterativo de NR luce así

$$\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)} - \left(\mathbf{J}_f(\mathbf{z}^{(k)})\right)^{-1}\mathbf{R}(\mathbf{z}^{(k)}) \tag{2.26}$$

Para evaluar el error de aproximación temporal cometido para una iteración de NR, se considerará $\mathbf{z}^{(0)} = 0$, obteniéndose los siguientes residuo y jacobiano:

$$\begin{aligned} \mathbf{R}^{(0)} &= -\Delta t \mathbf{f}(\mathbf{C}^n, t) \\ \frac{\partial \mathbf{R}}{\partial \mathbf{z}^n}(0) &= \mathbf{I} - \frac{\Delta t}{2}\mathbf{J}_f(\mathbf{C}^n, t) \end{aligned} \tag{2.27}$$

reemplazando en (2.26) se obtiene:

$$\begin{aligned} \mathbf{z}^{(1)} &= 0 - \left(I - \frac{\Delta t}{2}\mathbf{J}_f(\mathbf{C}^n, t)\right)^{-1}\left(-\Delta t \mathbf{f}(\mathbf{C}^n, t)\right) \\ &= \left(I - \frac{\Delta t}{2}\mathbf{J}_f(\mathbf{C}^n, t)\right)^{-1}\Delta t \mathbf{f}(\mathbf{C}^n, t) \end{aligned} \tag{2.28}$$

Considerando la serie geométrica $\frac{1}{1-x} = 1 + x + x^2 + x^3 + \ldots$, se puede escribir la inversa que aparece en (2.28) como:

$$\left(I - \frac{\Delta t}{2}\mathbf{J}_f(\mathbf{C}^n, t)\right)^{-1} = I + \frac{\Delta t}{2}\mathbf{J}_f(\mathbf{C}^n, t) + \frac{\Delta t^2}{4}\left(\mathbf{J}_f(\mathbf{C}^n, t)\right)^2 + O(\Delta t^3) \tag{2.29}$$

combinando (2.28), (2.29) y usando $\mathbf{C}^{n+1} = \mathbf{C}^n + \mathbf{z}^n$:

$$\begin{aligned} \mathbf{C}^{n+1} &= \mathbf{C}^n + \Delta t\mathbf{f}(\mathbf{C}^n, t) + \frac{\Delta t^2}{2}\mathbf{J}_f(\mathbf{C}^n, t)\mathbf{f}(\mathbf{C}^n, t) + O(\Delta t^3) \\ \Leftrightarrow \mathbf{C}^{n+1} &= \mathbf{C}^n + \frac{\Delta t}{2}\mathbf{f}(\mathbf{C}^n, t) + \frac{\Delta t}{2}\left[\mathbf{f}(\mathbf{C}^n, t) + \Delta t\mathbf{J}_f(\mathbf{C}^n, t)\mathbf{f}(\mathbf{C}^n, t)\right] + O(\Delta t^3) \end{aligned} \tag{2.30}$$

como se probará más adelante, la expresión entre corchetes en (2.30) se puede aproximar como:

$$\mathbf{f}(\mathbf{C}^n, t) + \Delta t\mathbf{J}_f(\mathbf{C}^n, t)\mathbf{f}(\mathbf{C}^n, t) = \mathbf{f}(\mathbf{C}^{n+1}, t) + O(\Delta t^2) \tag{2.31}$$

reemplazando ahora (2.31) en (2.30) se obtiene:

$$\begin{aligned} \mathbf{C}^{n+1} &= \mathbf{C}^n + \frac{\Delta t}{2}\mathbf{f}(\mathbf{C}^n, t) + \frac{\Delta t}{2}\left[\mathbf{f}(\mathbf{C}^{n+1}, t) + O(\Delta t^2)\right] + O(\Delta t^3) \\ \mathbf{C}^{n+1} &= \mathbf{C}^n + \frac{\Delta t}{2}\mathbf{f}(\mathbf{C}^n, t) + \frac{\Delta t}{2}\mathbf{f}(\mathbf{C}^{n+1}, t) + O(\Delta t^3) \\ \mathbf{C}^{n+1} &= \mathbf{C}^n + \frac{\Delta t}{2}\left(\mathbf{f}(\mathbf{C}^n, t) + \mathbf{f}(\mathbf{C}^{n+1}, t)\right) + O(\Delta t^3) \end{aligned} \tag{2.32}$$

Puede observarse que la última expresión de la ecuación (2.32) es equivalente a (2.16). En la práctica al implementar el método de NR se resuelve el sistema lineal que equivale a obtener la inversa del jacobiano en (2.26). Considerando la ecuación (2.32) se puede observar que al resolver el sistema linealizado de NR una única vez, se obtiene una solución con un error de truncamiento proporcional a Δt^3. Dicho error es del mismo orden que el del esquema temporal CN. Por lo tanto, si bien sucesivas iteraciones de NR producirían soluciones con un error numéricamente menor, el mismo mantendría el mismo orden. Expresado de otra manera, una única iteración en el esquema NR entrega una solución con un error del mismo orden que el de la solución de convergencia.

Para finalizar se demostrará la equivalencia considerada en la ecuación (2.31). Usando el polinomio de Taylor se puede escribir:

$$\mathbf{f}(\mathbf{C}^{n+1}, t) = \mathbf{f}(\mathbf{C}^n, t) + \mathbf{J}_f(\mathbf{C}^n, t)\left(\mathbf{C}^{n+1} - \mathbf{C}^n\right) + \frac{1}{2}\frac{\partial^2\mathbf{f}}{\partial\mathbf{C}^{n2}}\left(\mathbf{C}^{n+1} - \mathbf{C}^n\right)^2 \tag{2.33}$$

usando (2.10) puede verse que:

$$\begin{aligned} \mathbf{C}^{n+1} - \mathbf{C}^n &= \Delta t\mathbf{f}(\mathbf{C}^n, t) + O(\Delta t^2) \\ \left(\mathbf{C}^{n+1} - \mathbf{C}^n\right)^2 &= \Delta t^2\left(\mathbf{f}(\mathbf{C}^n, t)\right)^2 + O(\Delta t^3) \end{aligned} \tag{2.34}$$

reemplazando las últimas dos en la expresión (2.33) queda:

$$\begin{aligned} \mathbf{f}(\mathbf{C}^{n+1}, t) &= \mathbf{f}(\mathbf{C}^n, t) + \mathbf{J}_f(\mathbf{C}^n, t)\left[\Delta t\mathbf{f}(\mathbf{C}^n, t) + O(\Delta t^2)\right] \\ &\quad + \frac{\partial^2\mathbf{f}}{\partial\mathbf{C}^{n2}}\left[\frac{\Delta t^2}{2}\left(\mathbf{f}(\mathbf{C}^n, t)\right)^2 + O(\Delta t^3)\right] \\ \Leftrightarrow \mathbf{f}(\mathbf{C}^{n+1}, t) &= \mathbf{f}(\mathbf{C}^n, t) + \Delta t\mathbf{J}_f(\mathbf{C}^n, t)\mathbf{f}(\mathbf{C}^n, t) + O(\Delta t^2) \end{aligned} \tag{2.35}$$

finalmente esto valida la aproximación usada en (2.31). Con esto se prueba que para sistemas no lineales en los que no se tenga grandes inestabilidades debidas a la no linealidad, el algoritmo 1 puede modificarse realizando una sola iteración NR sin perder precisión, esto implica la solución de **un único** sistema lineal por cada paso de tiempo como sucede los esquemas implícitos en las ecuaciones lineales.

2.4. Ecuaciones de Navier Stokes

Si la propiedad intensiva que se conserva es la cantidad de movimiento, la ecuación general de transporte tiene carácter vectorial y luce así:

$$\frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla\cdot(\rho\mathbf{u}\mathbf{u}) = -\nabla p + \nabla\cdot(\mu\nabla\mathbf{u}) + \mathbf{f}_b \tag{2.36}$$

donde ρ es la densidad del fluido [kg/m^3], p representa la presión [N/m^2], μ la viscosidad dinámica o absoluta [kg/m/s^2] y $\mathbf{f}_b$ representa las fuerzas de cuerpo por unidad de volumen [N/m^3]. Ecuación simplificada que vale para fluidos newtonianos incompresibles, a la cual se agrega la ley de conservación de masa (ecuación de continuidad) o también conocida como condición de incompresibilidad:

$$\nabla\cdot\mathbf{u} = 0 \tag{2.37}$$

El conjunto de cuatro ecuaciones (1 ecuación vectorial + 1 ecuación escalar) (2.36) y (2.37) se conoce como ecuaciones de Navier-Stokes (NS), donde las incógnitas son (u_x, u_y, u_z) y p todas dependientes de $\mathbf{x}$ y t.

A diferencia de la ecuación de transporte escalar, las ecuaciones de NS agregan cierto grado de complejidad debido a:

- En el sistema no se dispone de una ecuación explícita independiente para la presión, sino que aparece solamente en las ecuaciones de momento como una variable secundaria en forma de gradiente y en general se lo trata como un término fuente (fuerza de cuerpo no conservativa) o como una fuerza superficial (tratamiento conservativo)[61].
- Las ecuaciones son no lineales debido al término convectivo en las ecuaciones de momento.
- Para flujos incompresibles, la velocidad y presión están fuertemente acopladas [121] a través de la ecuación de continuidad y la conservación de masa se transforma en una restricción cinemática al campo de velocidades en lugar de ser una ecuación dinámica.
- Finalmente, como la presión aparece en forma de gradiente, la solución del campo de presiones no es única y se determina a menos de una constante aditiva, en otras palabras el flujo solo se ve afectado por la diferencia de presiones de un punto a otro (presión relativa) y no por la presión absoluta [61].

Para sortear estas dificultades, se han diseñado algoritmos que linealizan la ecuación de momento, permiten resolver el acoplamiento presión-velocidad y finalmente combinar las ecuaciones para obtener una ecuación explícita para la presión a partir de la ecuación de continuidad. Las dos familias de métodos más utilizados en el contexto FVM, FDM y FEM, para resolver las ecuaciones de flujo incompresible son: métodos implícitos o semi-implícitos de corrección de presión y los métodos de proyección (o método de pasos fraccionados). Los primeros, conocidos en la literatura como métodos tipo SIMPLE (*Semi-Implicit Method for Pressure-Linked Equation*) se basan en un procedimiento del tipo predictor-corrector donde la presión se utiliza para imponer la continuidad, tratándola de forma explícita en las ecuaciones de momento, mientras que los segundos, son métodos no iterativos con base en la fundamentación matemática y la evolución de la velocidad se calcula en dos subpasos por cada paso de tiempo (predicción y proyección). Más aún, dado que el último enfoque es más general, algunos métodos directamente no incluyen la presión en el paso predictor y utilizan la misma como una variable puramente matemática en lugar de física (por ej. multiplicador de Lagrange)[38, 61, 85].

Otro aspecto a considerar al momento de la discretización es el ordenamiento de las variables en el dominio computacional. Al guardar las variables en centros de celda (*"collocated grid"*), las velocidades quedan desacopladas de la presión [121]. Para remediar esto, las dos alternativas más utilizadas son:

- Usar un arreglo en forma desparramada (*"staggered grid"*)

- Usar un ordenamiento *"colocado"* pero calculando las velocidades en centros de cara mediante la interpolación propuesta por Rhie-Chow [143].

En esta tesis se utiliza un ordenamiento colocado debido a las ventajas que supone para la GPU en términos de memoria y por otro lado como se utilizan métodos de fronteras embebidas (IBM) para resolver geometrías complejas, un arreglo desparramado implica que el obstáculo inmerso queda con una superficie difusa. La figura 2.2 presenta un ejemplo de los dos tipos de arreglos para el caso 2D.

En cuanto a la interpolación de Rhie-Chow para obtener $\mathbf{u}_f = (u_f, v_f, w_f)$ en la cara f, en [143] proponen una interpolación especial ponderada por presión conocida como PWIM (*Pressure-Wighted Interpolation Method*) que mantiene el acoplamiento presión-velocidad mediante el cálculo de las velocidades en caras como la media ponderada de los valores en centros de celda adyacentes más un término adicional que es una función del campo de presión. En las dos subsecciones siguientes se presenta la estrategia de interpolación que se utilizará para ambos algoritmos, no obstante en [133] puede encontrarse una descripción detallada del método original y algunas variantes del mismo.

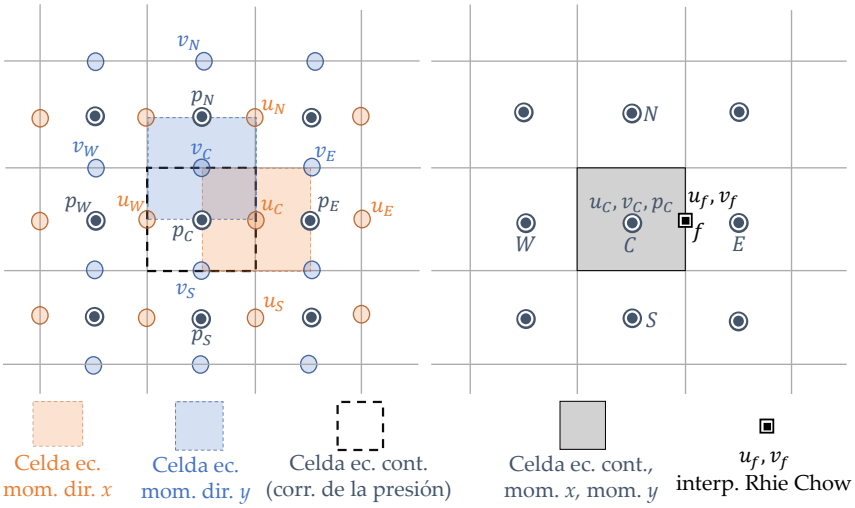

Figura 2.2: Tipos de arreglo para discretizar las ecuaciones de NS: *desparramado* (izquierda) y *colocado* (derecha).

A continuación se describen los dos métodos utilizados en esta tesis, para el lector interesado, en [124] se puede encontrar un análisis sobre la relación entre los métodos tipo SIMPLE y los métodos de proyección.

2.4.1. Método SIMPLE

El método SIMPLE fue desarrollado originalmente por Patankar y Spalding [134–136] y fue pensado para que sea bastante eficiente en la solución de problemas en estado estacionario, pero puede usarse indistintamente en problemas transientes. Entre los trabajos que realizan una implementación del algoritmo basada en GPU se pueden destacar [40, 80, 109, 127, 182, 184]. Cabe mencionar que la mayoría de trabajos en general diseñan aceleraciones en GPU del algoritmo, ejecutando una porción del mismo en GPU (por ejemplo la solución del sistema de ecuaciones) y el resto de pasos en CPU.

Discretización de la ecuación de momento

Sin pérdida de generalidad, considere la ecuación (2.36) para flujos incompresibles luego de dividir miembro a miembro por ρ. Integrando sobre la celda C cada término de la ecuación, aplicando el teorema del valor medio, el teorema de la divergencia si corresponde y aproximando los términos, se obtienen:

- **Término transiente:**

$$\int_{V_C} \frac{\partial(\mathbf{u})}{\partial t} dV = \left(\frac{\partial \mathbf{u}}{\partial t}\right)_C V_C = \frac{\mathbf{u}_C^{n+1} - \mathbf{u}_C^n}{\Delta t} V_C = \frac{\mathbf{u}_C^{n+1} - \mathbf{u}_C^n}{\Delta t} h^3 \tag{2.38}$$

- **Término convectivo:**

$$\int_{V_C} \nabla\cdot(\mathbf{uu}) dV = \int_{\partial V_C} \mathbf{u}(\mathbf{u}\cdot\mathbf{n})\mathbf{dS} = \sum_{f\sim nb(C)} \mathbf{u}_f\,(\mathbf{u}_f\cdot\mathbf{S}_f) = \sum_{f\sim nb(C)} \dot{m}_f \mathbf{u}_f \tag{2.39}$$

cabe mencionar que este término se lo trata mediante una linealización dentro del algoritmo, esto es, la incógnita es $\mathbf{u}_f$ discretizada mediante algún esquema espacial o considerando directamente el esquema UD y el enfoque de corrección diferida DC mientras que para el otro factor de flujo se utiliza la última solución disponible $\dot{m}_f^{(k)} = \mathbf{u}_f^{(k)}\cdot\mathbf{S}_f$ o se lo trata almacenado como una variable en caras $\dot{m}_f$.[2] Por ejemplo si se aplica el esquema UD:

$$\sum_{f\sim nb(C)} \dot{m}_f \mathbf{u}_f = \sum_{\substack{f\sim nb(C)\\ F\sim NB(C)}} (||\dot{m}_f, 0||\mathbf{u}_C - ||-\dot{m}_f, 0||\mathbf{u}_F) \tag{2.40}$$

- **Término de presión:**

$$-\int_{V_C} \nabla\hat{p}\, dV = -\int_{\partial V_C} \hat{p}\,\mathbf{dS} = -\sum_{f\sim nb(C)} \hat{p}_f \mathbf{S}_f \tag{2.41}$$

donde $\hat{p} = p/\rho$. Expandiendo la suma sobre las caras considerando un esquema espacial centrado, en el caso 3D:

$$\begin{aligned} -\sum_{f\sim nb(C)} \hat{p}_f\mathbf{S}_f = -\Bigg(&\frac{\hat{p}_E+\hat{p}_C}{2}\mathbf{S}_e + \frac{\hat{p}_W+\hat{p}_C}{2}\mathbf{S}_w + \frac{\hat{p}_N+\hat{p}_C}{2}\mathbf{S}_n \\ &+\frac{\hat{p}_S+\hat{p}_C}{2}\mathbf{S}_s + \frac{\hat{p}_T+\hat{p}_C}{2}\mathbf{S}_t + \frac{\hat{p}_B+\hat{p}_C}{2}\mathbf{S}_b\Bigg) = -\frac{h^2}{2}\begin{bmatrix} \hat{p}_E-\hat{p}_W \\ \hat{p}_N-\hat{p}_S \\ \hat{p}_T-\hat{p}_B \end{bmatrix} \end{aligned} \tag{2.42}$$

Otra alternativa es, en lugar de aplicar primero el teorema de la divergencia, aplicar el teorema de valor medio, en este caso:

$$-\int_{V_C} \nabla\hat{p}\, dV = -\left(\nabla\hat{p}\right)_C V_C = -\begin{bmatrix} \frac{\hat{p}_E-\hat{p}_W}{2h} \\ \frac{\hat{p}_N-\hat{p}_S}{2h} \\ \frac{\hat{p}_T-\hat{p}_B}{2h} \end{bmatrix} h^3 = -\frac{h^2}{2}\begin{bmatrix} \hat{p}_E-\hat{p}_W \\ \hat{p}_N-\hat{p}_S \\ \hat{p}_T-\hat{p}_B \end{bmatrix} \tag{2.43}$$

[2] En esta tesis se almacenan 3 vectores que contienen los flujos en caras $\dot{m}_e, \dot{m}_n, \dot{m}_t$ debido a que con esta estrategia se encontraron los mejores desempeños medidos en tiempos de cómputo en GPU.

- **Término difusivo:**

$$\int_{V_C} \nabla\cdot\left(\nu\nabla\mathbf{u}\right) dV = \int_{\partial V_C} \nu\left(\nabla\mathbf{u}\right)\cdot\mathbf{dS} = \nu \sum_{f\sim nb(C)} \nabla\mathbf{u}_f\cdot\mathbf{S}_f \tag{2.44}$$

donde $\nu = \mu/\rho$ es la viscosidad cinemática. Considerando el esquema CD para el gradiente en caras, queda:

$$\nu \sum_{f\sim nb(C)} \nabla\mathbf{u}_f\cdot\mathbf{S}_f = \nu \sum_{F\sim NB(C)} \begin{bmatrix} \frac{u_F - u_C}{h} \\ \frac{v_F - v_C}{h} \\ \frac{w_F - w_C}{h} \end{bmatrix} h^2 \tag{2.45}$$

- **Término de fuerzas de cuerpo:**

$$\int_{V_C} \mathbf{f}_b\, dV = \left(\mathbf{f}_b\right)_C V_C = \begin{bmatrix} f_C^x \\ f_C^y \\ f_C^z \end{bmatrix} h^3 \tag{2.46}$$

Agrupando todos los términos que surgen de aplicar los esquemas espacial y el esquema temporal FE, la ecuación de momento discretizada escrita en forma vectorial luce como:

$$a_C^{\mathbf{u}}\mathbf{u}_C + \sum_{F\sim NB(C)} a_F^{\mathbf{u}}\mathbf{u}_F = \mathbf{b}_C^{\mathbf{u}} \tag{2.47}$$

donde el coeficiente principal $a_C^{\mathbf{u}}$ contiene la contribución de los términos transiente, convectivo (usando enfoque DC) y difusivo:

$$a_C^{\mathbf{u}} = \frac{h^3}{\Delta t} + \sum_{f\sim nb(C)} \left(||\dot{m}_f, 0|| + \nu h\right) \tag{2.48}$$

los coeficientes que multiplican los valores de la variable en celdas vecinas:

$$a_F^{\mathbf{u}} = -||-\dot{m}_f, 0|| - \nu h \tag{2.49}$$

finalmente el lado derecho contiene los aportes del término transiente, del término fuerza de presión, las fuerzas de cuerpo y la contribución del esquema HR aplicado con el enfoque DC:

$$\mathbf{b}_C^{\mathbf{u}} = \frac{h^3}{\Delta t}\mathbf{u}_C^n - \sum_{f\sim nb(C)} \dot{m}_f\left(\mathbf{u}_f^{HR} - \mathbf{u}_f^U\right) - \left(\nabla\hat{p}\right)_C h^3 + \left(\mathbf{f}_b\right)_C h^3 \tag{2.50}$$

Note que los coeficientes del sistema dependen de la velocidad (originados por el término no lineal), pero como se mencionó antes, esto se maneja mediante una linealización dentro del un proceso iterativo, sin embargo, el cambio de los coeficientes para cambios grandes en $\mathbf{u}$ puede afectar la tasa de convergencia o incluso provocar la divergencia del método, por ello, se aplica un proceso de relajación implícita del sistema mediante el enfoque propuesto por Patankar [121, 135], con esto la ecuación relajada luce así:

$$\frac{a_C^{\mathbf{u}}}{\lambda^{\mathbf{u}}}\mathbf{u}_C + \sum_{F\sim NB(C)} a_F^{\mathbf{u}}\mathbf{u}_F = \mathbf{b}_C^{\mathbf{u}} + \frac{1-\lambda^{\mathbf{u}}}{\lambda^{\mathbf{u}}} a_C^{\mathbf{u}}\mathbf{u}_C^{(k)} \tag{2.51}$$

para facilitar la notación redefinimos las variables y se separa el término de presión del RHS:

$$\begin{aligned} a_C^{\mathbf{u}} &\leftarrow \frac{a_C^{\mathbf{u}}}{\lambda^{\mathbf{u}}} \\ \mathbf{b}_C^{\mathbf{u}} &\leftarrow \hat{\mathbf{b}}_C^{\mathbf{u}} + \frac{1-\lambda^{\mathbf{u}}}{\lambda^{\mathbf{u}}} a_C^{\mathbf{u}}\mathbf{u}_C^{(k)} - \left(\nabla\hat{p}\right)_C h^3 \end{aligned} \tag{2.52}$$

finalmente reescribimos la ecuación de momento de la siguiente forma:

$$\begin{aligned}\mathbf{u}_C + \sum_{F\sim NB(C)} \frac{a_F^{\mathbf{u}}}{a_C^{\mathbf{u}}}\mathbf{u}_F &= \frac{\mathbf{b}_C^{\mathbf{u}}}{a_C^{\mathbf{u}}} - (\nabla\hat{p})_C \frac{h^3}{a_C^{\mathbf{u}}} \\ \mathbf{u}_C + \mathbf{H}_C(\mathbf{u}) &= \mathbf{B}_C^{\mathbf{u}} - \mathbf{D}_C^{\mathbf{u}} (\nabla\hat{p})_c\end{aligned} \tag{2.53}$$

donde hemos definido los siguientes operadores:

$$\mathbf{H}_C(\mathbf{u}) = \sum_{F\sim NB(C)} \frac{a_F^{\mathbf{u}}}{a_C^{\mathbf{u}}}\mathbf{u}_F; \qquad \mathbf{B}_C^{\mathbf{u}} = \frac{\mathbf{b}_C^{\mathbf{u}}}{a_C^{\mathbf{u}}}; \qquad \mathbf{D}_C^{\mathbf{u}} = \frac{h^3}{a_C^{\mathbf{u}}} \tag{2.54}$$

Discretización de la ecuación de continuidad. Derivación de una ecuación para la presión

Considerando una solución inicial, por ejemplo la del paso de tiempo presente $\mathbf{u}^n$, $\dot{m}_f^n$ y $\hat{p}^n$, se resuelve la ecuación de momento para el campo $\mathbf{u}^*$ mediante la ecuación de momento relajada (2.53):

$$\mathbf{u}_C^* + \mathbf{H}_C(\mathbf{u}^*) = \mathbf{B}_C^{\mathbf{u}} - \mathbf{D}_C^{\mathbf{u}} (\nabla\hat{p}^n)_c \tag{2.55}$$

nótese que la solución de convergencia de esta ecuación linealizada en la velocidad y presión satisface la ecuación de momento pero no la ecuación de continuidad, por ello, se deben corregir los campos para forzar la conservación de la masa, estas correcciones se pueden escribir como:

$$\mathbf{u} = \mathbf{u}^* + \mathbf{u}'; \qquad \hat{p} = \hat{p}^n + p'; \qquad \dot{m}_f = \dot{m}_f^* + \dot{m}_f' \tag{2.56}$$

aplicando FVM a la ecuación de continuidad y reemplazando se obtiene:

$$\begin{aligned}\int_{V_C} \nabla\cdot\mathbf{u} = \int_{\partial V_C} \mathbf{u}\cdot\mathbf{dS} = \sum_{f\sim nb(C)} \mathbf{u}_f\cdot\mathbf{S}_f = \sum_{f\sim nb(C)} \dot{m}_f = 0 \\ \Leftrightarrow \sum_{f\sim nb(C)} \dot{m}_f' = -\sum_{f\sim nb(C)} \dot{m}_f^*\end{aligned} \tag{2.57}$$

aquí, si la suma de flujos másicos $\dot{m}_f^* = \mathbf{u}_f^*\cdot\mathbf{S}_f$ no es conservativa, lleva a un desbalance de masa en el RHS que se corregirá con los flujos $\dot{m}_f'$. Por otro lado, $\mathbf{u}_f^*$ se debe computar utilizando la interpolación de Rhie Chow [143] que para este caso luce así:

$$\mathbf{u}_f^* = \overline{\mathbf{u}_f^*} - \overline{\mathbf{D}_f^{\mathbf{u}}} \left(\nabla\hat{p}_f^n - \overline{\nabla\hat{p}_f^n}\right) \tag{2.58}$$

donde la barra superior indica promedio ponderado geométricamente, que para grillas cartesianas de paso constante h es directamente:

$$\overline{\mathbf{u}_f^*} = \frac{\mathbf{u}_C^* + \mathbf{u}_F^*}{2}; \qquad \overline{\mathbf{D}_f^{\mathbf{u}}} = \frac{\mathbf{D}_C^{\mathbf{u}} + \mathbf{D}_F^{\mathbf{u}}}{2} \tag{2.59}$$

Note además que para el cómputo de $\dot{m}_f^*$ solo se requiere una componente de $\mathbf{u}$ según la orientación del vector $\mathbf{S}_f$, por ejemplo, para las caras e, n, t se requieren respectivamente $u_{C,E}$ para $\mathbf{S}_e$, $v_{C,N}$ para $\mathbf{S}_n$ y $w_{C,T}$ para $\mathbf{S}_t$. En el caso del gradiente de presión ambos términos, gradiente en cara usual o compacto y gradiente en cara promediado o expandido, difieren como se muestra a continuación:

$$\begin{aligned}\nabla\hat{p}_f^n &= \frac{\hat{p}_F^n - \hat{p}_C^n}{h} \\ \overline{\nabla\hat{p}_f^n} &= \frac{\nabla\hat{p}_C^n + \nabla\hat{p}_F^n}{2}\end{aligned} \tag{2.60}$$

entonces para la cara *east* quedarían respectivamente:

$$\begin{aligned} \nabla \hat{p}_e^n &= \frac{\hat{p}_E^n - \hat{p}_C^n}{h} \\ \overline{\nabla \hat{p}_e^n} &= \frac{\dfrac{\hat{p}_E - \hat{p}_W}{2h} + \dfrac{\hat{p}_{EE} - \hat{p}_C}{2h}}{2} = \frac{\hat{p}_{EE} + \hat{p}_E - \hat{p}_C - \hat{p}_W}{4h} \end{aligned} \tag{2.61}$$

Restando (2.55) de (2.53) se obtienen ecuaciones para las variables en centros de celda $\mathbf{u}'_C$ y $\mathbf{u}'_F$

$$\mathbf{u}'_{C,F} + \mathbf{H}_{C,F}(\mathbf{u}') = -\mathbf{D}^{\mathbf{u}}_{C,F} \left(\nabla p'\right)_{C,F} \tag{2.62}$$

si se aplica la interpolación de Rhie-Chow se obtiene la correción de velocidades en caras:

$$\mathbf{u}'_f = \overline{\mathbf{u}'_f} - \overline{\mathbf{D}^{\mathbf{u}}_f} \left(\nabla p'_f - \overline{\nabla p'_f}\right) \tag{2.63}$$

que reemplazada en (2.57) permite obtener una ecuación de corrección de la presión:

$$\underbrace{\sum_{f\sim nb(C)} \overline{\mathbf{u}'_f \cdot \mathbf{S}_f} + \sum_{f\sim nb(C)} \overline{\mathbf{D}^{\mathbf{u}}_f}\, \overline{\nabla p'_f} \cdot \mathbf{S}_f}_{I} - \sum_{f\sim nb(C)} \overline{\mathbf{D}^{\mathbf{u}}_f} \nabla p'_f \cdot \mathbf{S}_f = -\sum_{f\sim nb(C)} \dot{m}^*_f \tag{2.64}$$

los términos sobre I representan el efecto de las correcciones de velocidad de los vecinos a celda C sobre la corrección de la misma. El tratamiento de dichos términos resulta crítico para poder arribar a un sistema lineal resoluble, en el algoritmo SIMPLE original este término se desprecia bajo la hipótesis de que al avanzar en el proceso iterativo las correcciones de los campos tienden a 0 y no afecta a la solución final, sin embargo, si las correcciones son grandes, ese término puede pesar lo suficiente para afectar la tasa de convergencia o provocar la divergencia. Entonces, para mejorar la tasa de convergencia del algoritmo, se realiza una relajación explícita del término corrección de presión como se indica más abajo.

Se han propuesto diferentes mejoras al algoritmo, por ejemplo aproximando de alguna manera los términos sobre I de forma que el sistema lineal quede resoluble, tratando la ecuación de momento de forma explícita entre otras estrategias, de allí se pueden derivar diferentes algoritmos de la familia de métodos tipo SIMPLE, por ejemplo SIMPLEC (SIMPLE Consistent) [173], SIMPLER (SIMPLE-Revised) [134], PRIME (Pressure Implicit Momentum Explicit) [108], PISO (Pressure-Implicit Split Operator) [78] entre otros.

Considerando el algoritmo SIMPLE original, con los términos restantes en la ecuación (2.64), se formula la ecuación de corrección para la presión:

$$\begin{aligned} -\sum_{f\sim nb(C)} \overline{\mathbf{D}^{\mathbf{u}}_f} \nabla p'_f \cdot \mathbf{S}_f &= -\sum_{F\sim NB(C)} \left(\frac{\mathbf{D}^{\mathbf{u}}_C + \mathbf{D}^{\mathbf{u}}_F}{2}\right) \left(\frac{p'_F - p'_C}{h}\right) h^2 = -\sum_{f\sim nb(C)} \dot{m}^*_f \\ \Leftrightarrow -\sum_{F\sim NB(C)} \frac{1}{2} \left(\frac{V_C}{a^{\mathbf{u}}_C} + \frac{V_F}{a^{\mathbf{u}}_F}\right) \left(\frac{p'_F - p'_C}{h}\right) h^2 &= -\sum_{f\sim nb(C)} \dot{m}^*_f \\ \Leftrightarrow -\sum_{F\sim NB(C)} \frac{h^3 h^2}{2h} \left(\frac{1}{a^{\mathbf{u}}_C} + \frac{1}{a^{\mathbf{u}}_F}\right) (p'_F - p'_C) &= -\sum_{f\sim nb(C)} \dot{m}^*_f \end{aligned} \tag{2.65}$$

donde se consideró grillas cartesianas de paso constante $\Delta x = \Delta y = \Delta z = h$, donde para el caso 3D resulta $V_C = V_F = h^3$, $|\mathbf{S}_f| = A_f = h^2$ y $d_{CF} = h$. Luego, la ecuación escrita en forma compacta análoga a (2.47) luce como:

$$a^{p'}_C p'_C + \sum_{F\sim NB(C)} a^{p'}_F p'_F = b^{p'}_C \tag{2.66}$$

donde los coeficientes se computan como:

$$a_C^{p'} = -\sum_{F\sim NB(C)} a_F^{p'}; \qquad a_F^{p'} = -\frac{h^3h^2}{2h}\left(\frac{1}{a_C^{\mathbf{u}}}+\frac{1}{a_F^{\mathbf{u}}}\right); \qquad b_C^{p'} = -\sum_{f\sim nb(C)} \dot{m}_f^* \tag{2.67}$$

Una vez calculado el campo de corrección para la presión, se corrigen los campos y se pasa a la siguiente iteración del lazo SIMPLE:

$$\mathbf{u}_C^{**} = \mathbf{u}_C^* - \mathbf{D}_C^{\mathbf{u}}\left(\nabla p'\right)_C; \qquad \dot{m}_f^{**} = \dot{m}_f^* - \overline{\mathbf{D}_f^{\mathbf{u}}}\nabla p'_f \cdot \mathbf{S}_f; \qquad \hat{p}_C^* = \hat{p}_C^n + \lambda^p p'_C \tag{2.68}$$

Para mejorar la convergencia, se utilizó la relajación implícita para la ecuación de momento y una relajación explícita para la corrección de la presión utilizando los factores $\lambda^{\mathbf{u}}$ y λ^p, luego, la evolución de la tasa de convergencia es función de dichos valores, en [121] se muestra que los valores óptimos para estos factores guardan la relación $\lambda^p \approx 1-\lambda^{\mathbf{u}}$. En esta tesis se utilizaron los siguientes valores $\lambda^{\mathbf{u}} = 0{,}7$ y $\lambda^p = 0{,}3$.

Hay que comentar también, que el uso de grillas colocadas e interpolación de Rhie-Chow resultan en soluciones que dependen del factor de relajación, para eliminar dicha dependencia, se requieren realizar modificaciones en dicha interpolación teniendo en cuenta siempre, que se busca imitar la formulación que se obtendría en una grilla desparramada, formando una ecuación de pseudo-momentum en las caras de las celdas [121, 133]. Ese tratamiento también debe tenerse en cuenta cuando se utiliza un esquema temporal diferente al FE y al tratar fuerzas de cuerpo como por ejemplo la fuerza gravitatoria, tarea que requiere almacenar variables extras en la implementación que son utilizadas al ensamblar el RHS de la ecuación de momento. En [121] se repasan algunos ejemplos para los 3 casos mencionados (relajación implícita, término transiente y término de fuerzas de cuerpo).

Algoritmo SIMPLE en grillas colocadas

Finalmente el algoritmo SIMPLE a implementar consiste en las siguientes operaciones:

Algoritmo 2: Algoritmo SIMPLE para flujos incompresibles en grillas colocadas.

mientras $t < T_{final}$ **hacer**
  setear $\dot{m}^{(0)}, \mathbf{u}^{(0)}, \hat{p}^{(0)} \leftarrow \dot{m}^n, \mathbf{u}^n, \hat{p}^n$// (solución a tiempo $t = n\Delta t$ disponible);
  para $k = 0, \ldots$ *hasta la convergencia* **hacer**
    ensamblar el sistema lineal (2.55) y resolver para $\mathbf{u}^*$ // (ecuación de momento);
    computar $\dot{m}_f^*$ usando la interpolación de Rhie-Chow (2.58) ;
    ensamblar el sistema lineal (2.66) y resolver para p'// (ecuación de corrección de la presión);
    corregir los campos $\dot{m}^{**}, \mathbf{u}^{**}, \hat{p}^*$ usando (2.68);
    setear $\dot{m}^{(k+1)}, \mathbf{u}^{(k+1)}, \hat{p}^{(k+1)} \leftarrow \dot{m}^{**}, \mathbf{u}^{**}, \hat{p}^*$;
  fin
  setear $\dot{m}^{n+1}, \mathbf{u}^{n+1}, \hat{p}^{n+1} \leftarrow \dot{m}^{**}, \mathbf{u}^{**}, \hat{p}^*$;
fin

2.4.2. Métodos de proyección, pasos fraccionados

El método de proyección conocido también como método de pasos fraccionados (*fractional step method*-FSM) fue desarrollado originalmente por Chorin [38] y Temam [165, 166]. Los diferentes métodos radican en la discretización temporal elegida para los diferentes términos, desde formulaciones completamente implícitas no lineales donde se requiere aplicar el método NR

[35] o aplicar alguna linealización del término convectivo, formulaciones semi-implícitas donde el término convectivo se trata explícitamente y el término difusivo implícitamente [85, 174], hasta formulaciones explícitas en la velocidad [37, 165, 166]. Entre los trabajos que implementan alguna variante del FSM en GPU se destacan [88, 97, 161, 167, 189]. La mayoría de trabajos revisados implementan variantes del método en grillas desparramadas, sin embargo, como se mencionó en la subsección anterior, para esta tesis busca el uso de grillas colocadas.

A continuación se describirá el método desarrollado originalmente por Van-Kan [174] con las modificaciones propuestas por Zang et al. [187] para adaptarlo en grillas irregulares colocadas y por Ye et al. [186] para el caso de grillas cartesianas colocadas, dichas modificaciones consisten en que el cálculo de las velocidades en centros de cara se realiza de forma separada de las velocidades en centros de celda, aplicando una interpolación del tipo Rhie-Chow. El método tipo FSM consiste en cuatro pasos principales. En el primer paso (predicción) se resuelve la ecuación de momento obteniendo una velocidad intermedia $\mathbf{u}^*$, en el segundo paso se interpolan las velocidades en caras $\mathbf{U}_f$ (Rhie-Chow) con las cuales se computa la divergencia $\nabla\cdot\mathbf{u}^*$, en el tercer paso se resuelve una ecuación de corrección para la presión p'_C (problema tipo Poisson) donde el lado derecho es la divergencia del campo de velocidades y en el último paso (proyección) se actualizan los campos de velocidad en centros de celda $\mathbf{u}_C^{n+1}$ y centros de cara $\mathbf{U}_f^{n+1}$ y campo de presión p_C^{n+1} utilizando el campo obtenido en el tercer paso. Este método de pasos fraccionados es formalmente de segundo orden de precisión en el tiempo [10, 138].

Paso 1. Solución de la ecuación de momento (paso de predicción)

Considerando la ecuación (2.36) sin el término de fuerzas de cuerpo, esta ecuación será discretizada utilizando FVM centrado en celda (ordenamiento colocado) donde se almacenarán el campo de velocidades $\mathbf{u}_C$ y el campo de presiones como se definió en el algoritmo SIMPLE $\hat{p}_C = p_C/\rho$ para flujos incompresibles, adicionalmente se almacenarán las velocidades en caras $\mathbf{U}_f$ normales a ellas, esto es un campo equivalente a $\dot{m}_f$ utilizando en SIMPLE. Aplicando el esquema AB y CN para el término convectivo y difusivo respectivamente queda:

$$\begin{aligned}\frac{\mathbf{u}_C^* - \mathbf{u}_C^n}{\Delta t}h^3 + \sum_{f\sim nb(C)}\left(\frac{3}{2}\mathbf{u}_f^n\mathbf{U}_f^n\cdot\mathbf{S}_f - \frac{1}{2}\mathbf{u}_f^{n-1}\mathbf{U}_f^{n-1}\cdot\mathbf{S}_f\right) &= -\left(\nabla\hat{p}\right)_C^n h^3 \\ + \sum_{f\sim nb(C)}\frac{\mu}{\rho}\left(\frac{1}{2}\nabla\mathbf{u}_f^*\cdot\mathbf{S}_f + \frac{1}{2}\nabla\mathbf{u}_f^n\cdot\mathbf{S}_f\right)&\end{aligned} \tag{2.69}$$

note que en el segundo término se trata explícitamente (esto facilita el tratamiento de la no linealidad) y se expresaron las velocidades en caras como $\mathbf{U}_f$ en lugar de $\mathbf{u}_f$ debido a que se tratan por separado, es decir, mientras que $\mathbf{u}_f$ se computa según algún esquema de discretización espacial (U, CD o esquema HR), $\mathbf{U}_f$ se obtiene mediante interpolación de RC y se actualizará en el cuarto paso por separado. El término de presión, se discretiza como en (2.42) o (2.43), en cuanto al término de difusión, se aproxima de la misma forma que en (2.45), esto es, en ambos casos se utiliza un esquema CD.

Considerando todas las celdas del dominio computacional, se arriba a tres sistemas lineales para las incógnitas $\mathbf{u}^* = (u^*, v^*, w^*)$, las matrices son del tipo simétrica definida positiva (SPD), con lo que puede resolverse mediante el método de Gradientes Conjugados.

El sistema (2.69) puede escribirse de forma compacta como en la subsección anterior:

$$a_C^{\mathbf{u}}\mathbf{u}_C^* + \sum_{F\sim NB(C)} a_F^{\mathbf{u}}\mathbf{u}_F^* = \mathbf{b}_C^{\mathbf{u}} \tag{2.70}$$

en donde, si se elige un esquema HR[3] para el término convectivo los coeficientes valen:

$$
\begin{aligned}
a_C^{\mathbf{u}} &= \frac{h^3}{\Delta t} + \frac{1}{2}\frac{6\nu h^2}{h}; \qquad a_F^{\mathbf{u}} = -\frac{1}{2}\frac{\nu h^2}{h} \\
\mathbf{b}_C^{\mathbf{u}} = \begin{bmatrix} b_C^u \\ b_C^v \\ b_C^w \end{bmatrix} &= -\frac{h^2}{2}\begin{bmatrix} \hat{p}_E - \hat{p}_W \\ \hat{p}_N - \hat{p}_S \\ \hat{p}_T - \hat{p}_B \end{bmatrix}^n + \frac{1}{2}\frac{\nu h^2}{h}\sum_{F\sim NB(C)}\begin{bmatrix} u_F - u_C \\ v_F - v_C \\ w_F - w_C \end{bmatrix}^n \\
&-\frac{3}{2}\sum_{f\sim nb(C)}(\mathbf{U}_f\cdot\mathbf{S}_f)\begin{bmatrix} u_f^{HR} \\ v_f^{HR} \\ w_f^{HR} \end{bmatrix}^n + \frac{1}{2}\sum_{f\sim nb(C)}(\mathbf{U}_f\cdot\mathbf{S}_f)\begin{bmatrix} u_f^{HR} \\ v_f^{HR} \\ w_f^{HR} \end{bmatrix}^{n-1}
\end{aligned}
\tag{2.71}
$$

Paso 2. Interpolación de Rhie-Chow para computar $\mathbf{U}_f^*$

En este paso intermedio, se calculan las velocidades en centros de cara (normales a ella) para permitir el acoplamiento presión-velocidad que se obtendría utilizando un arreglo desparramado, este paso consiste en:

$$
\begin{aligned}
\tilde{\mathbf{u}}_C &= \mathbf{u}_C^* + \Delta t\nabla\hat{p}_C^n = \mathbf{u}_C^* + \frac{\Delta t}{2h}\begin{bmatrix} \hat{p}_E^n - \hat{p}_W^n \\ \hat{p}_N^n - \hat{p}_S^n \\ \hat{p}_T^n - \hat{p}_B^n \end{bmatrix} \\
\tilde{\mathbf{U}}_f &= \frac{\tilde{\mathbf{u}}_C + \tilde{\mathbf{u}}_F}{2} \\
\mathbf{U}_f^* &= \tilde{\mathbf{U}}_f - \Delta t\nabla\hat{p}_f^n = \tilde{\mathbf{U}}_f - \Delta t\left(\frac{\hat{p}_F^n - \hat{p}_C^n}{h}\right)
\end{aligned}
\tag{2.72}
$$

nótese que en el proceso de promediado se utiliza primero los gradientes de presión en centros de celda y luego los gradientes de presión en centros de cara, además de ello, al realizar el paso intermedio para obtener $\tilde{\mathbf{U}}_f$, se estaría utilizando el gradiente de presión en centros de cara promediando, $\overline{\nabla\hat{p}_f}$. Observe además aplicar este proceso es equivalente a aplicar la interpolación (2.58) y solamente difieren en el multiplicador Δt.

Paso 3. Calculo de la divergencia y obtención del campo de corrección para la presión

Este paso requiere la solución de la ecuación de corrección de la presión

$$
\frac{\mathbf{u}_C^{n+1} - \mathbf{u}_C^*}{\Delta t} = -\nabla p_C' \tag{2.73}
$$

con la restricción de que el campo de velocidad final $\mathbf{u}_C^{n+1}$ tenga divergencia nula $\nabla\cdot\mathbf{u}^{n+1} = 0$. Aplicando el operador divergencia miembro a miembro se obtiene la ecuación:

$$
\nabla\cdot\left(\nabla p_C'\right) = \frac{1}{\Delta t}\nabla\cdot\mathbf{u}_C^* \tag{2.74}
$$

integrando miembro a miembro, aplicando el teorema de la divergencia y utilizando las velocidades en centros de cara obtenidas en el paso 2:

$$
\begin{aligned}
\int_{V_C}\nabla\cdot\left(\nabla p_C'\right)dV &= \frac{1}{\Delta t}\int_{V_C}\nabla\cdot\mathbf{u}_C^* dV \\
\Leftrightarrow \int_{\partial V_C}\nabla p_f'\cdot\mathbf{dS} &= \frac{1}{\Delta t}\int_{\partial V_C}\mathbf{u}^*\cdot\mathbf{dS} \\
\Leftrightarrow \sum_{f\sim nb(C)}\nabla p_f'\cdot\mathbf{S}_f = \sum_{F\sim NB(C)}\frac{p_F' - p_C'}{h}h^2 &= \frac{1}{\Delta t}\sum_{f\sim nb(C)}\mathbf{U}_f^*\cdot\mathbf{S}_f
\end{aligned}
\tag{2.75}
$$

[3] Para computar $\mathbf{u}_f^{HR}$ utilice (2.5).

Esta ecuación de Poisson es análoga a (2.66) pero con coeficientes de la matriz uniformes para todas las celdas del dominio, es decir los coeficientes de la ecuación escrita en la forma (2.66) son simplemente:

$$\begin{aligned} a_C^{p'} &= -\sum_{F\sim NB(C)} a_F^{p'} = -6h; \quad a_F^{p'} = h; \\ b_C^{p'} &= \frac{h^2}{\Delta t}\left(U_e^* - U_w^* + U_n^* - U_s^* + U_t^* - U_b^*\right) \end{aligned} \tag{2.76}$$

Paso 4. Cómputo de $\mathbf{u}^{n+1}$, $\mathbf{U}^{n+1}$ y $\hat{p}^{n+1}$ (paso de proyección)

Una vez calculada la corrección para la presión, los campos se actualizan de forma independiente mediante:

$$\hat{p}^{n+1} = \frac{p^{n+1}}{\rho} = \hat{p}^n + p' \quad ; \quad \mathbf{u}_C^{n+1} = \mathbf{u}_C^* - \Delta t \nabla p'_C \quad ; \quad \mathbf{U}_f^{n+1} = \mathbf{U}_f^* - \Delta t \nabla p'_f \tag{2.77}$$

note que las velocidades en centros de celda se actualizan con el gradiente del campo p' computado en centro de celda mientras que las velocidades en centro de cara (componente normal a la misma) se calculan utilizando el gradiente de p' *compacto* computado en centro de cara, usando solamente valores adyacentes a la cara, por ejemplo para la componente x y cara *east* vale:

$$u_C^{n+1} = u_C^* - \Delta t\left(\frac{p'_E - p'_W}{h}\right); \qquad U_e^{n+1} = U_e^* - \Delta t\left(\frac{p'_E - p'_C}{h}\right) \tag{2.78}$$

Algoritmo 3: Algoritmo de pasos fraccionados para flujos incompresibles en grillas colocadas.

1 setear $\mathbf{U}^{(0)}, \mathbf{u}^{(0)}, \hat{p}^{(0)} \leftarrow \mathbf{U}^n, \mathbf{u}^n, \hat{p}^n$// (solución a tiempo $t = n\Delta t$ disponible);
2 **mientras** $t < T_{final}$ **hacer**
3 ensamblar el sistema lineal (2.69) y resolver para $\mathbf{u}^*$ // (ecuación de momento);
4 computar $\mathbf{U}_f^*$ con el proceso de promediado (2.72) // (interp. Rhie-Chow);
5 calcular la divergencia utilizando $\mathbf{U}^*$ y resolver la ecuación (2.75) para p'// (ecuación de corrección de la presión);
6 corregir los campos $\mathbf{u}^{n+1}, \mathbf{U}^{n+1}, \hat{p}^{n+1}$ usando (2.77);
7 avanzar al siguiente paso de tiempo;
8 **fin**

Finalmente el algoritmo 3, presenta las operaciones del método de pasos fraccionados implementado.

Capítulo 3

Algoritmos en GPU

En las décadas recientes, las unidades de procesamiento gráfico (GPU) se han vuelto uno de los más populares aceleradores debido a su arquitectura masivamente paralela. En comparación con las plataformas CPU, las GPU están equipadas con más unidades de cómputo designadas como *CUDA cores* dentro de cada chip. Esto resulta beneficioso para aplicaciones basadas en FVM que pueden ser intensivas en cálculos, obteniendo aceleraciones significativas al ejecutarse en GPU.

Uno de los principales cuellos de botella en el rendimiento de los códigos basados en GPU es que se requieren una gran cantidad de accesos a memoria global de la GPU en cada paso de tiempo. Por otro lado, en varios algoritmos las dependencias de instrucciones y las ramificaciones son muchas veces un paso inevitable.

Los desafíos que deben sortearse al implementar un código en GPU son entre otras cosas, que se dispone de un espacio limitado de memoria en el chip de la GPU, la latencia de acceso a memoria causada por los frecuentes accesos a memoria global y la latencia de instrucciones causadas al utilizar operaciones trascendentes, resultan problemáticas al combinarse con la dependencia de ejecución dentro de los algoritmos que surgen de aplicar los métodos de discretización.

3.1. Arquitectura GPU

Las GPUs poseen varios multiprocesadores de transmisión (SM por sus siglas en inglés), donde cada uno posee sus propios CUDA cores y memoria que se puede acceder de forma explícita como la memoria compartida (*shared memory*) o de forma implícita como la memoria tipo caché L1 (ver figura 3.1-derecha). Entre las principales diferencias de la arquitectura GPU con las unidades CPU es que los hilos están en gran medida particionados y la manipulación de diferentes hilos tiene un costo prácticamente nulo, por otro lado, las CPU están optimizadas para cálculo secuenciales donde la latencia es un factor importante, mientras que las GPUs logran un alto rendimiento en códigos paralelos siempre que se cumplan las demandas de rendimiento.

Históricamente programar en GPU era una tarea compleja hasta el surgimiento de CUDA, que permite una programación paralela en GPU sin necesidad aplicar técnicas que requieran conocimientos sobre gráficos. CUDA permite organizar los hilos en bloques que pueden sincronizarse entre sí, y la ejecución de los mismos se puede realizar en cualquier orden para maximizar el paralelismo disponible, a su vez los bloques se pueden organizar en grillas (ver figura 3.1-izquierda).

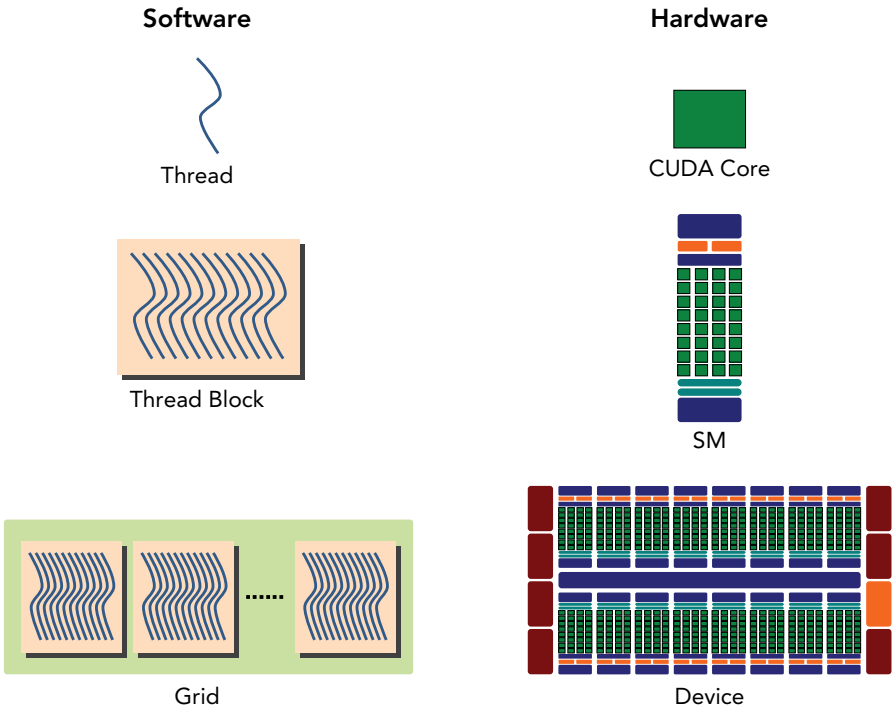

Figura 3.1: Modelo de ejecución CUDA - Software versus Hardware, imagen tomada de [33].

3.1.1. Factores que afectan la eficiencia del código en GPU

Los factores que impactan en la eficiencia incluyen la ramificación divergente, el tipo de almacenamiento de memoria y el acceso a memoria coalescente. La ramificación divergente ocurre con la lógica de control de la aplicación CUDA. En el modelo se permite la ejecución simultánea de hasta 32 hilos (CUDA threads), esta unidad se conoce como warp [128]. En la ejecución de más hilos, las GPUs compartirán el procesador entre los warps. Si dentro de un warp hay un condicional que lleva a dos o más caminos separados, entonces se produce una ramificación divergente. La divergencia serializa algunos hilos de warp, reduciendo el impacto de la paralelización. En dichos casos, la ejecución de cada camino se divide en warps separados y cada hilo que no esté en la rama en ejecución permanece inactivo. Esta serialización afecta negativamente el rendimiento del código en CUDA.

El acceso coalescente a la memoria reduce el número de accesos a memoria global o compartida que hace la GPU para realizar operaciones sobre los datos, en efecto, el procesador puede almacenar algunos datos en una memoria caché que es significativamente más rápida, luego, puede operar sobre estos datos más rápidamente ya que se encuentran almacenados en una memoria más rápida, esto es posible gracias al acceso coalescente, que reduce la cantidad de accesos a memoria necesarios y mejora la eficiencia de la GPU al procesar datos en paralelo.

A continuación se describen los tipos de memoria que posee el dispositivo (GPU).

3.1.2. Jerarquías de memorias en CUDA

CUDA establece distintos niveles de memoria, estos tipos de almacenamiento proporcionan diversos beneficios tanto a la complejidad de los algoritmos como a su eficiencia. La jerarquía de memoria de las GPUs permite una gestión eficiente de los datos y proporciona diferentes niveles de almacenamiento para adaptarse a las necesidades específicas de los algoritmos y las operaciones realizadas en la GPU. En la figura 3.2 se presenta un esquema de la ubicación de cada tipo de memoria, la jeraquía se compone de:

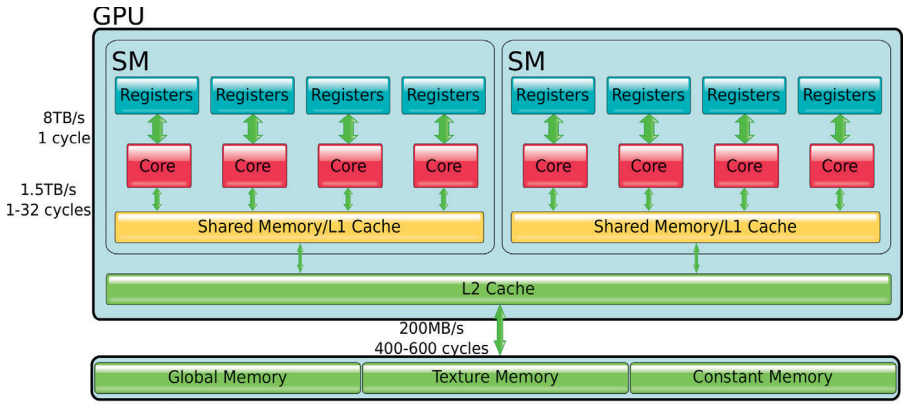

Figura 3.2: Jerarquía de memoria de la GPU. Cada bus esta etiquetado con valores típicos de ancho de banda y latencia, imagen tomada de [17].

- Registros: son la memoria más rápida y de menor capacidad disponible en la GPU. Se utiliza para almacenar datos y variables locales dentro de un hilo y tienen un tiempo de acceso extremadamente rápido.
- Memoria compartida: es una memoria de tamaño limitado que se comparte entre hilos dentro de un bloque. Permite una comunicación rápida y cooperativa entre hilos de un bloque y tiene un tiempo de acceso más rápido que la memoria global pero más lento que la memoria de registros.
- Caché L1: es una memoria caché de bajo nivel que almacena datos y fragmentos de datos que se acceden con frecuencia. Proporciona un acceso rápido a datos que se encuentren en la memoria global y ayuda a reducir la latencia de acceso a memoria. La principal diferencia entre la memoria compartida y la caché L1 es que el código administra explícitamente el contenido de la primera, mientras que la segunda se administra automáticamente, sin embargo CUDA permite administrar que fracción de la memoria disponible se utiliza para memoria compartida y memoria caché L1. Se espera que ambas tengan un rendimiento similar.
- Memoria global: es la memoria de mayor capacidad en la GPU y se utiliza para almacenar datos accesibles por todos los hilos de la GPU. Tiene un tiempo de acceso más lento en comparación con los registros, la memoria compartida y la caché L1. Es la única parte de la memoria que es accesible desde la CPU (*host*) mediante las funciones de la biblioteca CUDA.
- Caché L2: al igual que la memoria caché L1, este segundo nivel de memoria caché almacena datos y fragmentos de datos que se acceden con frecuencia. La gestión de memoria en L2 también es manejada automáticamente por el hardware. La diferencia entre ambos niveles de memoria caché radica en que L2 es compartida entre todos los SMs mientras que L1 sólo se comparte entre procesos dentro de un mismo SM, lo que implica una diferencia de rendimiento (L2 es comparable con la memoria global y L1 proporciona un acceso varias veces más rápido). El flujo de datos desde un proceso dentro de una función que solicita a memoria global pasa primero por L1 y ante la falta pasa a L2 y ante una falta nuevamente finaliza la solicitud en la memoria global de la GPU.
- Memoria constante: es una memoria de solo lectura que se utiliza para almacenar datos constantes que son accesibles por todos los hilos de la GPU. Proporciona un acceso rápido a los datos constantes y es útil para almacenar constantes matemáticas o tablas de consulta.
- Memoria de textura: es una memoria especializada que se utiliza para el mapeo de texturas en aplicaciones gráficas.

Las memorias administradas para las implementaciones son: la memoria global (donde residen todos los vectores y arreglos almacenados durante la ejecución del código), la memoria shared (donde se cargan los datos leídos desde la memoria global, el tiempo de vida es el tiempo que se ejecute el bloque), la memoria de registro (donde se alocan variables auxiliares y permite el rehúso de datos ya accedidos, el tiempo de vida es el tiempo de cada hilo).

Para finalizar se describe la terminología fundamental que se utiliza en el contexto de programación en CUDA:

- Thread (hilo): Es la unidad más pequeña de ejecución en una GPU. Cada hilo realiza una porción de trabajo independiente. Los hilos se agrupan en bloques y se ejecutan en paralelo.
- Warp: Un warp es un grupo de hilos que se ejecutan simultáneamente en una GPU. En las GPUs de NVIDIA, un warp consiste en 32 hilos. Los hilos dentro de un warp se ejecutan en paralelo y siguen la misma instrucción, pero operan con diferentes datos.
- Block (bloque): Un bloque es un grupo de hilos que se ejecutan en un multiprocesador de la GPU. Los bloques se dividen en warps, y cada multiprocesador ejecuta múltiples bloques de forma concurrente. Los hilos dentro de un bloque pueden colaborar y comunicarse entre sí a través de memoria compartida.
- Grid: Un grid es una colección de bloques. Define la estructura de la ejecución en paralelo de los hilos en una GPU. Los bloques dentro de un grid pueden ejecutarse en paralelo en diferentes multiprocesadores.
- Kernel: Es una función o rutina que se ejecuta en la GPU. Un kernel se lanza en la GPU y se encarga de coordinar la ejecución de los hilos, bloques y grids. Cada instancia de kernel se ejecuta en un solo multiprocesador y puede manejar múltiples bloques y miles de hilos.

Para resumir, los hilos son las unidades de ejecución más pequeñas, los warps son grupos de hilos que se ejecutan en paralelo, los bloques son grupos de hilos que se ejecutan en un multiprocesador, los grids son colecciones de bloques y el kernel es la función principal que coordina la ejecución de los hilos, bloques y grids en la GPU.

3.2. Solución de sistemas lineales en GPU

La discretización de las ecuaciones de gobierno para casos estacionarios o en el caso de métodos implícitos en casos transientes, resulta en una serie de sistemas lineales de ecuaciones algebraicas que deben resolverse. Las matrices son dispersas y con estructura de banda, según el esquema de discretización espacial que se utilice. Las técnicas para resolver los sistemas lineales se agrupan en métodos directos o iterativos. Los métodos directos rara vez son utilizados en arquitecturas masivamente paralelas y en problemas de gran escala. En general se prefieren los métodos iterativos por el menor costo computacional por iteración y el bajo requerimiento memoria.

Una estrategia de cómputo muy utilizada es almacenar las diagonales de la matriz, esto es viable en GPU principalmente para el caso de SPD en problemas donde la diagonal principal es función de las subdiagonales y se requieren almacenar 2 o 3 diagonales para el caso 2D y 3D respectivamente, para con ello realizar el producto matriz vector (operación SPMV). Por otra parte, en el contexto de las GPUs una estrategia eficiente en cuanto a requerimientos de memoria es utilizar el estilo conocido como *matrix free* donde se genera una función que realiza la operación lineal u operación tipo *stencil*, esto es $\mathbf{y} \leftarrow f(\mathbf{x})$ tal que $f = \mathbf{A}\mathbf{x}$ ensamblando dinámicamente los coeficientes del sistema para multiplicar por las componentes de $\mathbf{x}$. Esta función se puede pasar como objeto al resolvedor lineal para realizar la operación SPMV requerida en el algoritmo.

Los métodos iterativos pueden clasificarse en dos grandes grupos, los métodos estacionarios (por ejemplo métodos de Jacobi, Gauss-Seidel GS, Sobre-Relajación Sucesiva SOR) y los métodos no estacionarios (entre los métodos más utilizados se encuentran Gradientes Conjugados CG,

Gradientes Biconjugados BiCG, Gradientes Biconjugados Estabilizado BiCGStab, Residuo Mínimo Generalizado GMRES). Los métodos estacionarios tienen poco requerimiento de cómputo por iteración pero su convergencia en general es lenta. Para grandes sistemas de ecuaciones se prefieren los algoritmos conocidos como métodos del subespacio de Krylov que forman parte de los métodos no estacionarios. Los métodos de Krylov convergen en una cantidad K de pasos donde K es directamente proporcional al tamaño del sistema.

Por otro lado, el paso que más consumo de tiempo insume en los algoritmos para resolver las ecuaciones de NS es la resolución del sistema para obtener la corrección para la presión, ecuación de Poisson (PPE) [80, 189]. Una estrategia ampliamente utilizada para reducir los tiempos es utilizar la Transformada Rápida de Fourier (FFT) [40, 69, 161], que en el contexto GPU se puede realizar de manera muy eficiente con la biblioteca de última generación cuFFT [1] nativa de CUDA, este método clasificaría como un método directo y permite obtener el resultado con solo $O(N \log N)$ operaciones siendo N el número de celdas.

Al resolver el problema de la PPE mediante un método de Krylov, se ha observado que hay una caída de la tasa de convergencia incluso en sistemas de mediano tamaño, luego de eliminar los errores iniciales. A raíz de ello se han desarrollado la técnicas multigrilla (MG) [26, 70, 168, 179]. Si bien los métodos MG forman parte de los métodos iterativos estacionarios, para la PPE tienen un costo de $O(N)$ operaciones e incluso pueden obtenerse tasas de convergencia independiente del tamaño de grilla. Por otro lado, Feng et al. [58] han mostrado que en GPU el método multigrilla geométrico GMG tiene un desempeño superior a la estrategia FFT para el problema de Poisson, en concreto, GMG resulta entre 33 % y 23 % más rápido que FFT para el problema Poisson 2D y 3D en GPU.

Para esta tesis se implementan los métodos CG, BiCGStab, MG y diferentes precondicionadores para CG y BiCGStab, los primeros métodos se describen en el presente capítulo mientras que el método MG se trata en el siguiente capítulo.

3.2.1. Método de Gradientes Conjugados

El método CG fue originalmente desarrollado por Hestenes y Steifel [73] y se aplica para matrices SPD. El método se basa en actualizar el sistema en cada iteración mirando direcciones de búsqueda $\mathbf{p}_j$ conjugadas una de otras. La tasa de convergencia del método CG se deteriora si hay autovalores pequeños en el espectro de la matriz $\mathbf{A}$ del sistema, para acelerar la convergencia, se puede transformar el sistema lineal aplicando una matriz $\mathbf{M}$. $\mathbf{M}$ es llamada precondicionador y para CG se asume que también es SPD de manera que preservará las propiedades de la matriz $\mathbf{A}$ original del sistema. La idea es elegir $\mathbf{M}$ tal que el sistema transformado $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$ tiene un número de condición menor que el original, es decir $\kappa(\mathbf{M}^{-1}\mathbf{A}) < \kappa(\mathbf{A})$. Por otro lado, la matriz $\mathbf{M}$ no se computa nunca explícitamente, en lugar de ello el algoritmo de gradientes conjugados precondicionado (PCG) se formula usando la operación $\mathbf{M}^{-1}$. El algoritmo 4 presenta los pasos del método PCG en la forma que fue implementada en esta tesis. Nótese que si en la línea 6 (paso de precondicionamiento) se realiza una copia ($\mathbf{z}_i \leftarrow \mathbf{r}_i$) o se utiliza $\mathbf{M} = \mathbf{I}$, se obtiene directamente el método CG.

3.2.2. Método de Gradientes Biconjugados Estabilizado

El método BiCGStab es una modificación de CG, propuesta por Van der Vorst [172], mientras que CG puede mantener vectores de residuos ortogonales si la matriz es simétrica, BiCGStab puede aplicarse a matrices no simétricas, para ello, se utilizan dos pares de vectores residuos, direcciones de búsqueda y constantes, uno para $\mathbf{A}$ y otro para su transpuesta. Las características similares a CG lo hacen atractivo como solver para matrices no simétricas. El algoritmo 5 presenta la implementación utilizada en la tesis.

Algoritmo 4: Algoritmo de gradientes conjugados precondicionado PCG.

$\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0$ // A: matriz del sistema, b: lado derecho (RHS), x_0: solución inicial ;
$rr_0 \leftarrow \langle \mathbf{r}_0, \mathbf{r}_0 \rangle$;
$rr \leftarrow rr_0$;
$i \leftarrow 0$;
mientras $(\sqrt{rr} > tol_{abs} + \sqrt{rr_0} tol_{rel})$ & $(i < iter_{\text{máx}})$ **hacer**
  $\mathbf{z}_i \leftarrow \mathbf{M}^{-1}\mathbf{r}_i$ // paso de precondicionamiento;
  $\rho_i \leftarrow \langle \mathbf{r}_i, \mathbf{z}_i \rangle$;
  si $i == 0$ **entonces**
    $\mathbf{p}_i \leftarrow \mathbf{z}_i$
  fin
  en otro caso
    $\beta_i \leftarrow \rho_i / \rho_{i-1}$;
    $\mathbf{p}_{i+1} \leftarrow \mathbf{z}_i + \beta_i \mathbf{p}_i$;
    $\mathbf{y}_{i+1} \leftarrow \mathbf{A}\mathbf{p}_{i+1}$;
    $\alpha_i \leftarrow \rho_i / \langle \mathbf{y}_{i+1}, \mathbf{p}_{i+1} \rangle$;
    $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \alpha_i \mathbf{p}_{i+1}$;
    $\mathbf{r}_{i+1} \leftarrow \mathbf{r}_i - \alpha_i \mathbf{y}_{i+1}$;
    $rr \leftarrow \langle \mathbf{r}_{i+1}, \mathbf{r}_{i+1} \rangle$;
    $i \leftarrow i + 1$;
  fin
fin

3.2.3. Precondicionadores

Entre los precondicionadores más simples y adecuados para paralelizar se encuentran: el precondicionador de Jacobi (o precondicionador diagonal) y los precondicionadores basados en la Series de Neumann Truncadas [68]. Designados respectivamente como D y TN, en cada caso, la matriz de precondicionamiento luce como:

$$\mathbf{M}_D = \mathbf{D}, \qquad \mathbf{M}_{TN1} = \left(\mathbf{I} + \mathbf{L}\mathbf{D}^{-1}\right) \mathbf{D} \left(\mathbf{I} + (\mathbf{L}\mathbf{D}^{-1})^T\right) \tag{3.1}$$

donde $\mathbf{I}$ es la matriz identidad, $\mathbf{D}$ es la parte diagonal y $\mathbf{L}$ la parte triangular inferior estricta de la matriz del sistema $\mathbf{A}$, ambos precondicionadores son simétricos. La inversión del sistema en el primer caso es trivial, mientras que para el segundo precondicionador, se aplica la aproximación de las series de Neumann. En efecto, el factor $\left(\mathbf{I} + \mathbf{L}\mathbf{D}^{-1}\right)^{-1}$ puede definirse con la serie:

$$\mathbf{I} - \mathbf{L}\mathbf{D}^{-1} + \left(\mathbf{L}\mathbf{D}^{-1}\right)^2 - \left(\mathbf{L}\mathbf{D}^{-1}\right)^3 + \dots$$

eligiendo la serie truncada de primer orden como la inversa del precondicionador:

$$\mathbf{M}_{TN1}^{-1} = \left(\mathbf{I} - \mathbf{D}^{-1}\mathbf{L}^T\right) \mathbf{D}^{-1} \left(\mathbf{I} - \mathbf{L}\mathbf{D}^{-1}\right) \tag{3.2}$$

Definiendo $\mathbf{E} = \mathbf{I} - \mathbf{L}\mathbf{D}^{-1}$, se puede escribir $\mathbf{M}_{TN1}^{-1} = \mathbf{E}^T\mathbf{D}^{-1}\mathbf{E}$. Entonces para la implementación en GPU en este caso, la matriz $\mathbf{D}$ se almacena explícitamente y luego, la inversión del sistema $\mathbf{M}\mathbf{z} = \mathbf{r}$ (es decir $\mathbf{z} \leftarrow \mathbf{M}^{-1}\mathbf{r}$) se lleva a cabo en dos operaciones: primero el producto $\mathbf{z}_{\text{tmp}} = \mathbf{D}^{-1}\mathbf{E}\mathbf{r}$ y luego el producto $\mathbf{z} = \mathbf{E}^T\mathbf{z}_{\text{tmp}}$.

Aunque estos dos precondicionadores básicos son muy fáciles de implementar en GPU, no resultan tan efectivos para algunos problemas. Los precondicionadores más efectivos utilizados en la actualidad en los resolvedores rápidos para la PPE son los basados en la FFT [161], los basados en técnicas MG (precondicionadores multinivel) y los basados en técnicas de deflación [12, 125]. Sin embargo, la implementación en GPU de estos últimos no es tan eficiente ya que parte del código se ejecuta en CPU, véase por ejemplo [22].

Algoritmo 5: Algoritmo de gradientes bi conjugados estabilidado precondicionado PBiCGStab.

$\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0$ // A: matriz del sistema, b: lado derecho (RHS), x_0: solución inicial ;
$\mathbf{p}_0 \leftarrow \mathbf{r}_0$;
$\hat{\mathbf{r}}_0 \leftarrow \mathbf{r}_0$;
$rr_0 \leftarrow \langle \mathbf{r}_0, \mathbf{r}_0 \rangle$;
$rr \leftarrow rr_0$;
$i \leftarrow 0$;
mientras $(\sqrt{rr} > tol_{abs} + \sqrt{rr_0} tol_{rel})$ & $(i < iter_{\text{máx}})$ **hacer**
    $\mathbf{M}\mathbf{p}_i \leftarrow \mathbf{M}^{-1}\mathbf{p}_i$ // paso de precondicionamiento;
    $\mathbf{A}\mathbf{M}\mathbf{p}_i \leftarrow \mathbf{A}\mathbf{M}\mathbf{p}_i$// operación tipo stencil;
    $\alpha_i \leftarrow \langle \mathbf{A}\mathbf{M}\mathbf{p}_i, \hat{\mathbf{r}}_i \rangle$;
    $\mathbf{s}_i \leftarrow \mathbf{r}_i - \alpha_i \mathbf{A}\mathbf{M}\mathbf{p}_i$;
    $ss_i \leftarrow \langle \mathbf{s}_i, \mathbf{s}_i \rangle$;
    si $\sqrt{ss_i} < tol_{abs} + \sqrt{rr_0} tol_{rel}$ **entonces**
        $\mathbf{x}_i \leftarrow \mathbf{x}_i + \alpha \mathbf{M}\mathbf{p}_i$;
        interrumpir;
    fin
    en otro caso
        $\mathbf{M}\mathbf{s}_i \leftarrow \mathbf{M}^{-1}\mathbf{s}_i$ // paso de precondicionamiento;
        $\mathbf{A}\mathbf{M}\mathbf{s}_i \leftarrow \mathbf{A}\mathbf{M}\mathbf{s}_i$// operación tipo stencil;
        $\omega_i \leftarrow \langle \mathbf{A}\mathbf{M}\mathbf{s}_i, \mathbf{s}_i \rangle$ / $\langle \mathbf{A}\mathbf{M}\mathbf{s}_i, \mathbf{A}\mathbf{M}\mathbf{s}_i \rangle$;
        $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \alpha_i \mathbf{M}\mathbf{p}_i + \omega \mathbf{M}\mathbf{s}_i$;
        $\mathbf{r}_{i+1} \leftarrow \mathbf{s}_i - \omega_i \mathbf{M}\mathbf{s}_i$;
        $\beta_i \leftarrow \langle \mathbf{r}_{i+1}, \hat{\mathbf{r}}_i \rangle$ / $\langle \mathbf{r}_i, \hat{\mathbf{r}}_i \rangle$ (α_i / ω_i);
        $\mathbf{p}_{i+1} \leftarrow \mathbf{r}_{i+1} + \beta_i (\mathbf{p}_i - \omega_i \mathbf{A}\mathbf{M}\mathbf{p}_i)$;
        $rr \leftarrow \langle \mathbf{r}_{i+1}, \mathbf{r}_{i+1} \rangle$;
        $i \leftarrow i + 1$;
    fin
fin

3.2.4. Implementación GPU de los algoritmos PCG y PBiCGStab

Se implementaron los algoritmos 4 y 5 en GPU. La iteración PCG consta de las siguientes operaciones básicas de álgebra lineal: 3 op. AXPY, 3 op. de producto interno, 2 op. tipo SpMV (una correspondiente a la resolución del sistema del paso de precondicionamiento). Por su parte la iteración PBiCGSTab consta de 6 op. AXPY, 4 op. tipo SpMV (dos de ellas corresponden a la resolución del precondicionamiento) y 6 op. de producto interno.

La paralelización en GPU de la operación AXPY es trivial, en este caso por cada índice se requieren 2 lecturas (X e Y) y 1 escritura (resultado) en memoria global, con lo que no hay margen para optimizar la operación y se utiliza directamente una implementación propia. En cambio el producto interno requiere una operación de reducción y comunicación global entre hilos (CUDA threads), entre las librerías e implementaciones propias siguiendo diferentes estrategias, la más eficiente fue la implementación que proporciona la librería de NVIDIA CUBLAS [2], por otro lado, al ser una librería nativa, no se pierde el soporte al usar diferentes versiones de la herramienta CUDA (CUDA Toolkit) como sucede por ejemplo si se utiliza la librería CUSP [42] o Thrust [5] donde la mayoría de veces no se tiene migrado el total de funciones de una versión a otra impidiendo el funcionamiento de las implementaciones al cambiar la versión del compilador nvcc. Finalmente las operaciones SpMV consideran el paso de precondicionamiento y el producto $\mathbf{Ax}$ de los algoritmos. Para el paso $\mathbf{Ax}$ se implementan según el problema a resolver, en general se utiliza el estilo matrix-free donde cada hilo procesa el equivalente a una fila de $\mathbf{A}$ computando a partir de algún campo de variables los coeficientes de la plantilla (stencil) requeridos. La segunda estrategia que se ha probado consiste en almacenar $\mathbf{A}$ en formato diagonal (DIA), que por ejemplo para el problema de Poisson para la presión en 2D, al tratarse de una plantilla de 5 puntos simétrico donde además los coeficientes guardan la relación $a_C = -(a_E + a_W + a_N + a_S)$ sólo basta con almacenar 2 diagonales. Ambas estrategias tienen un desempeño similar, demandando menos tiempo la segunda opción, con la desventaja de que requiere más memoria para almacenamiento de las diagonales. Respecto al paso de precondicionamiento, dependen del precondicionador en particular.

3.3. Paralelización de los métodos explícitos e implícitos en GPU

Las implementaciones en GPU desarrolladas en esta tesis se realizan teniendo en cuenta que el almacenamiento de datos y la totalidad de cálculo se realicen completamente en la tarjeta gráfica. Para mejorar la eficiencia se tuvo en cuenta la arquitectura SIMT (*Single Instruction Multiple Threads*) de CUDA, se divide el procesamiento de la malla en diferentes CUDA *kernels*, de manera que se tiene un kernel para cada caso de borde y uno para las celdas interiores de acuerdo al siguiente esquema para problemas en 3D (ver figura 3.3): 8 kernels para procesar cada vértice del dominio, 12 kernels para procesar las aristas, 6 kernels para procesar las caras y 1 kernel para procesar el interior del dominio. De esta forma todos estos kernels pueden correrse de manera concurrente en operaciones que dependan de los datos del tiempo anterior (esquemas explícitos) o de la iteración anterior (esquemas implícitos).

La implementación del kernel para procesar el interior del dominio consiste en un lazo que permite recorrer el dominio en dirección z mediante planos xy uno para cada elevación z del dominio discretizado (ver figura 3.4). Respecto a los accesos a los datos del dispositivo dentro del kernel, en general para problemas 3D se requerirán por cada celda, 7 accesos, de los cuales 2 corresponden al plano superior e inferior del plano xy que se está procesando. La estrategia adoptada entonces es que cada hilo (CUDA threads) procese un punto del plano actual, que el plano actual xy sea cargado en memoria compartida y que el plano actual pase a ser el plano posterior en la siguiente iteración del lazo interno del kernel, evitando así los accesos redundantes a la memoria global del dispositivo. Para los planos del borde se procede igual que los planos xy interiores del dominio, usando memoria compartida. Los cálculo asociados a las aristas y a los

vértices se realizan sin usar memoria compartida debido a que no se observaron mejoras en el rendimiento. Con este esquema se busca explotar el paralelismo de manera de alcanzar un método que sea escalable al aumentar el tamaño del dominio computacional. De lo contrario, en caso de utilizar arreglos de hilos tridimensionales, se corre el riesgo de estar limitado en cuanto a los índices de los bloques [23, 114].

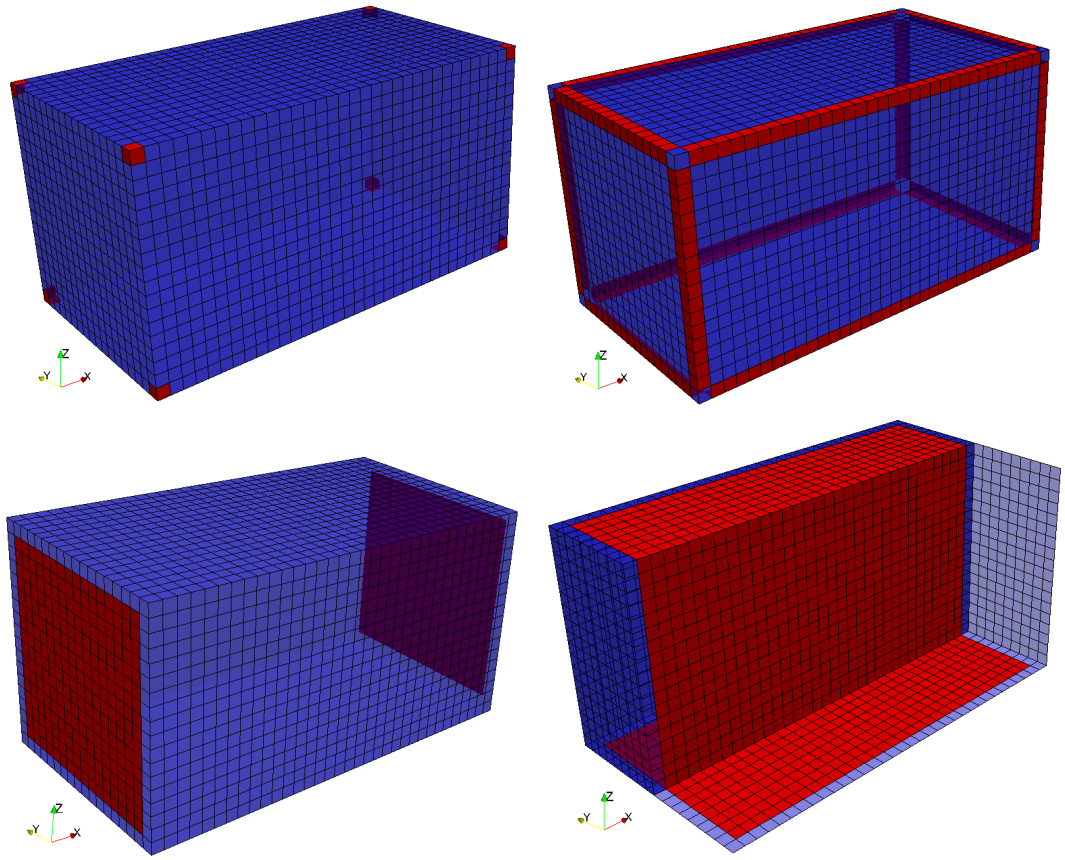

Figura 3.3: División del procesamiento del dominio computacional en el caso 3D. 4 funciones para vértices, 12 funciones para aristas, 6 funciones para caras, 1 función para interior del dominio.

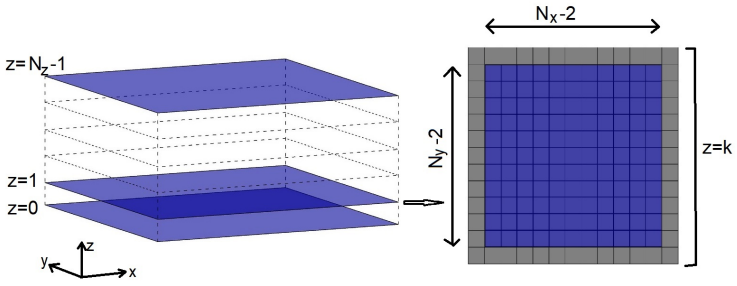

Figura 3.4: Procesamiento del dominio computacional por planos.

Para los algoritmos implícitos se requieren resolver sistemas lineales de ecuaciones, la estrategia mayormente utilizada es el estilo matrix free. Los resolvedores lineales se implementaron mediante el uso de clases. En concreto, se definió una clase para el operador lineal, que es llamado para hacer el producto matriz vector $\mathbf{Ax}$ requerido dentro del resolvedor. Así es que el operador lineal que llama el resolvedor consiste en varios kernels, dividiendo el procesamiento del dominio como se mencionó antes. Como la operación consiste en el producto de una fila de una matriz por el vector auxiliar del resolvedor, al usar esta estrategia se evita el almacenamiento de la matriz ya

que se ensambla directamente en el momento de usarla.

Para el caso de problemas 2D, se sigue una estrategia similar con el siguiente esquema (ver figura 3.5):

- 4 kernels para procesar cada vértice del dominio.
- 4 kernels para procesar las aristas.
- 1 kernel para procesar el interior del dominio.

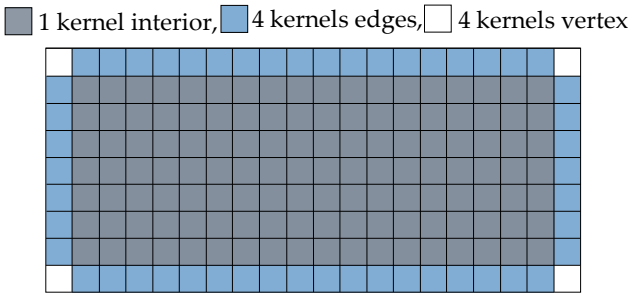

Figura 3.5: Procesamiento del dominio computacional en el caso 2D.

Cabe mencionar que para problemas 2D muchas veces hay muy poco rehúso de los datos en las operaciones involucradas, por ello, en algunos casos se ha omitido el uso de la memoria compartida y directamente se utiliza la memoria de registro y lecturas a memoria global.

3.4. Hardware y software

La totalidad de algoritmos implementados en esta tesis, inlcuyendo los de este capítulo y los siguientes, fueron ejecutados en alguno de los siguientes 2 equipos de cómputo, el primero posee una GPU de dos generaciones previas (Tesla K40) y el segundo con una GPU de una generación previa (Tesla V100).

- **Equipo 1.** Dell PowerEdge R720 (2013/14): 2 CPU Intel Xeon (R) E5-2620v2 (6 cores cada uno corriendo a 2.1GHz), 128GB DDR3 de memoria RAM. 2 GPUs NVIDIA Tesla K40 (2880 CUDA cores, corriendo a 745MHz, 12GB de memoria GDDR5), conectadas mediante el bus PCI-e.
- **Equipo 2.** Dell PowerEdge R740 (2018/19): 2 CPU Intel Xeon Gold 6138 (20 cores cada uno, corriendo a 2.0GHz) con 128GB DDR4 de memoria RAM. 2 GPUs NVIDIA Tesla V100 (5120 CUDA cores, corriendo a 1230MHz, 32GB de memoria HBM2), conectadas mediante el bus PCI-e.

En cuanto al software, el sistema operativo en ambos equipos es CentOS Linux v7, la versión del compilador GCC es 4.8.5. Para el compilador nvcc de CUDA, se utilizaron las versiones v9.1 en el equipo 1 y v10.1 en el equipo 2. Todas las implementaciones se ejecutan en DP (doble precisión). Para las operaciones de álgebra lineal, se hace uso de la librería CUBLAS [2], entre otras librerías provistas por NVIDIA.

A modo informativo, se menciona que en general la diferencia de rendimiento entre ambas generaciones de GPUs está en un factor de entre 5 a 7 en favor de la última generación según el algoritmo implementado. La mejora se debe en parte a la mayor cantidad de CUDA cores de una generación respecto de la otra y mejoras en el ancho de banda para lecturas/escrituras en memoria global en la última generación, lo que implica que muchas veces conviene hacer uso intensivo de la memoria de registros o leer en memoria global las variables en lugar de utilizar la memoria compartida.

Finalmente se comenta que en el caso del cómputo paralelo en CPU, para varias pruebas se encontró que el desempeño de los 12 cores del equipo 1, igualan o superan veces el desempeño de los 40 cores del equipo 2, ejecutando el mismo código en diferentes equipos. En problemas pequeños esto se explica en que el mayor grado de paralelismo podría penalizar, sin embargo esto se observó incluso para problemas relativamente grandes. Esto indica que entre 2013 y 2018 (5 generaciones) no se han producidos mejoras significativas en cuanto al rendimiento de las CPUs, mientras que en GPU, el salto de una generación a otra implica ganancias de medio orden de magnitud en tiempos de cómputo y de memoria.

3.5. Advección difusión 2D usando esquemas lineales

Este experimento pretende dar algunas pautas sobre la eficiencia a la hora de elegir una combinación de métodos explícitos o implícitos según la plataforma paralela utilizada.

En este caso de estudio se compara la performance de varios métodos para resolver la ecuación de transporte (2.1) en 2D usando diferentes implementaciones paralelas basadas en GPU y en CPU. Los códigos en CPU se implementaron usando C+OpenMP. Las ecuaciones se resuelven usando FVM en mallas estructuradas con la estrategia de paralelización mencionada en la sección anterior. Se implementan esquemas temporales implícito (CN) y explícitos (FE y RK2) con un esquema espacial de alto orden (HO) centrado (CD). El sistema lineal resultante del método implícito se resuelve usando el método de gradientes bi-conjugados estabilizado (BiCGStab) (implementaciones en C+CUDA y C+OpenMP). Para evaluar la eficiencia de las paralelizaciones en diferentes arquitecturas y paradigmas, se comparan la precisión, la velocidad de cálculo y las tasas de cómputo para diferentes tamaños de grilla. Las propiedades de convergencia de los diferentes esquemas se evalúan en relación con la discretización espacial y temporal.

3.5.1. Caso de estudio: transporte de un pulso, ecuación con solución analítica

Se considera la ecuación (2.1) con $f = 0$ en $\Omega = [-2, 2] \times [-2, 2]$:

$$\frac{\partial C}{\partial t} + \nabla\cdot(\mathbf{u}C - \nu\nabla C) = 0 \in \Omega \tag{3.3}$$

con el campo de velocidades $\mathbf{u} = (-y, x)$ en m/s y el coeficiente de difusión $\nu = 10^{-2}$ en m^2/s. La solución analítica para este problema viene dada por [92]:

$$C(\mathbf{x}, t) = \frac{1}{4\pi\nu t} e^{-\frac{r^2}{4\nu t}}, \quad \text{con } r^2 = (x - \hat{x})^2 + (y - \hat{y})^2 \tag{3.4}$$

donde

$$\hat{x} = x_0 \cos t - y_0 \sin t \ , \ \hat{y} = -x_0 \sin t + y_0 \cos t \tag{3.5}$$

observe que como $C(\mathbf{x}, t)$ presenta una singularidad en $t = 0$, para las pruebas se establece como tiempo inicial $t_0 = \pi/2$s $\approx 1{,}57$s y tiempo final $t_{\text{final}} = 5\pi/2$s $\approx 7{,}85$s que corresponde a un giro completo. La figura 3.6 presenta la evolución temporal de la solución numérica para una malla de 128×128 utilizando el esquema temporal implícito CN con Δt y esquema espacial centrado como se describe en las siguientes secciones. Todas las pruebas fueron ejecutadas en el Equipo 2.

3.5.2. Discretización espacial y temporal

Se discretizó el problema (3.3) mediante FVM centrado en celda. Para los parámetros elegidos, el problema puede resolverse utilizando un esquema espacial centrado (CD) para ambos términos, advectivo y difusivo. Recordando la ecuación (2.2), la ecuación semidiscreta en este caso luce:

$$\left(\frac{\partial C}{\partial t}\right)_C h^2 + \sum_{F\sim nb(C)} \left(\dot{m}_f \frac{C_F + C_C}{2} - \nu \frac{C_F - C_C}{h} h\right) = 0 \tag{3.6}$$

donde $\dot{m}_f = \mathbf{u}_f \cdot \mathbf{S}_f$ debe evaluarse para cada término entre los centros F y C. Note que este esquema es lineal por lo tanto no se requiere aplicar la estrategia de corrección diferida (DC).

Para la discretización temporal se utilizaron los esquemas temporales explícitos FE y RK2, de primer y segundo orden respectivamente, y el esquema temporal implícito CN (orden $O(\Delta t^2)$).

Esquemas explícitos

La expresión para el esquema FE surge de aplicar la fórmula (2.10) a la ecuación semidiscreta. La ecuación para las celdas internas es:

$$C_C^{n+1} = C_C^n + \frac{\Delta t}{h^2}\left[\nu \sum_{F\sim nb(C)} (C_F^n - C_C^n) - \sum_{F\sim nb(C)} \dot{m}_f \frac{C_F^n + C_C^n}{2}\right] \tag{3.7}$$

Para el esquema RK2, se aplica la ecuación anterior dos veces, en el primer paso se obtiene el campo $\mathbf{C}^{n+1}$, con ello se evalúa nuevamente la ecuación en $\mathbf{C}^{n+1}$ para obtener $\mathbf{C}^{n+2}$, finalmente se obtiene la solución para el nuevo tiempo mediante $\mathbf{C}^{n+1} = 0{,}5(\mathbf{C}^{n+2} + \mathbf{C}^n)$ tal como lo indica las operaciones (2.12).

Como el campo de velocidades es variable en el dominio, el criterio de estabilidad numérica que estimó para este caso fue:

$$\Delta t \leq \frac{h^2}{4\nu + 2h} \tag{3.8}$$

La fórmula anterior resulta de la condición (2.20) para el caso 2D, evaluando el campo de velocidades en algún punto de acuerdo a su definición $\mathbf{u} = (-y, x)$. Esta condición se utilizó para elegir el paso de tiempo tanto en el método de FE como RK2. Aunque es una aproximación, cabe mencionar que en los experimentos numéricos se encontró que excediendo ese valor la solución numérica diverge mientras que si se elige un paso temporal muy próximo a (3.8) la solución se mantiene estable durante toda la simulación.

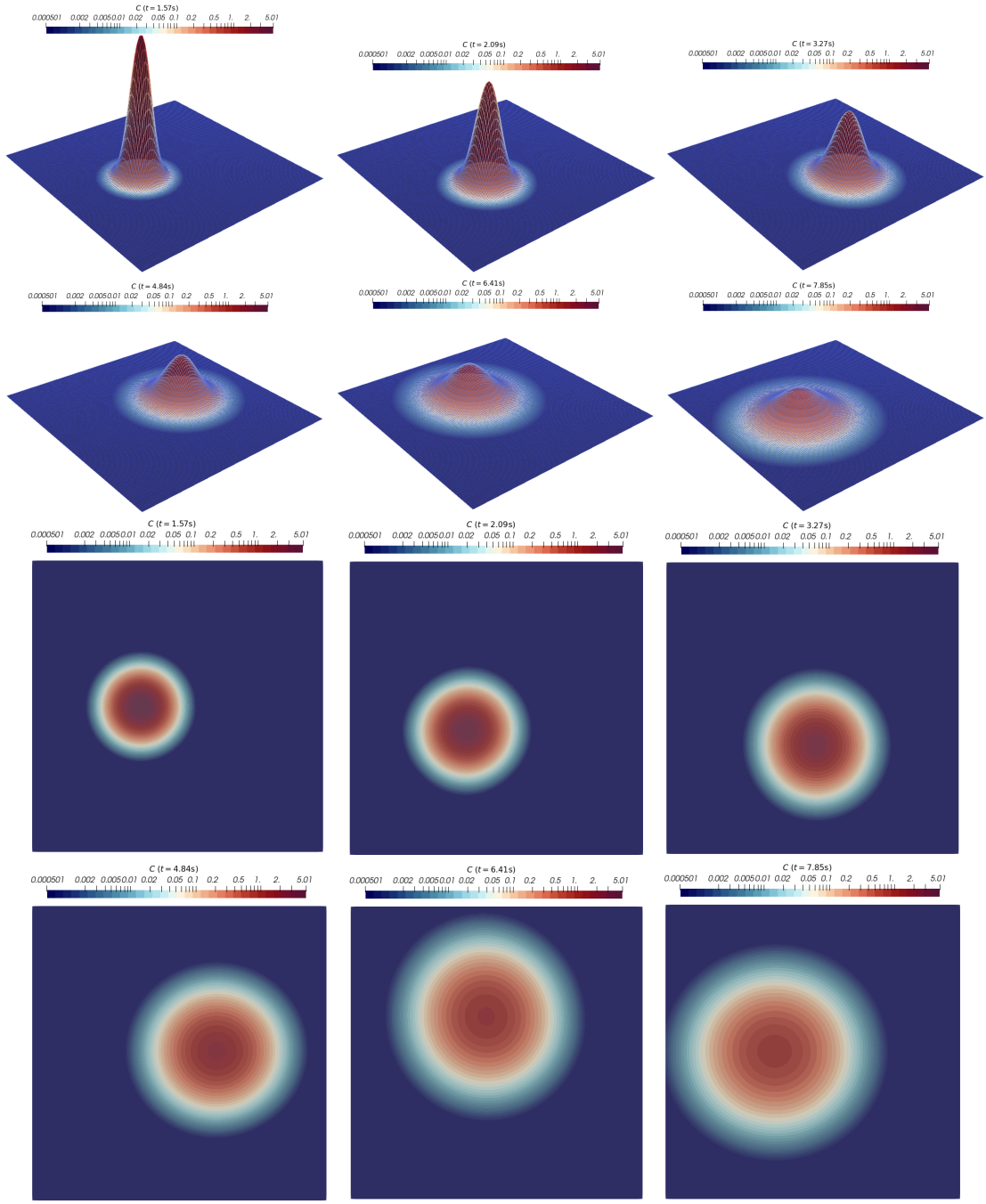

Figura 3.6: Solución numérica del problema para tiempos $t = \{\pi/2; 2{,}09; 3{,}27; 4{,}84; 6{,}41; 5\pi/2\}$ en segundos. Esquema temporal CN, esquema espacial CD (términos advectivo y difusivo), $\Delta x = \Delta y = 4/128 = 0{,}03125$, $\Delta t \approx 0{,}03808$s.

Esquema implícito

La ecuación del esquema CN para celdas internas surge de aplicar la fórmula (2.16):

$$\begin{aligned}\frac{h^2}{\Delta t}C_C^{n+1} - \frac{1}{2}\sum_{F\sim nb(C)} \nu\left(C_F^{n+1} - C_C^{n+1}\right) + \frac{1}{4}\sum_{F\sim nb(C)} \dot{m}_f(C_F^{n+1} + C_C^{n+1}) \\ = \frac{h^2}{\Delta t}C_C^{n} + \frac{1}{2}\sum_{F\sim nb(C)} \nu\left(C_F^{n} - C_C^{n}\right) - \frac{1}{4}\sum_{F\sim nb(C)} \dot{m}_f(C_F^{n} + C_C^{n})\end{aligned} \tag{3.9}$$

Este esquema es incondicionalmente estable, sin embargo en las pruebas se eligieron pasos de tiempo de manera que el orden del error del esquema temporal sea inferior al que proporciona el esquema espacial.

3.5.3. Detalles de las implementaciones paralelas

Se implementaron versiones paralelas de los tres métodos en GPU y CPU. Para las versiones CPU se utilizó la interfaz de programación de aplicaciones (API) para sistemas de memoria compartida OpenMP. Con ello las estrategias de paralelización en CPU y GPU resultan bastante similares entre sí y son las descritas en la sección anterior, es decir usando el esquema de 4 funciones para los vértices, 4 funciones para las aristas y 1 función para procesar el interior del dominio. Solamente se destacarán las principales diferencias que se tuvieron en cuenta.

Paralelización de los métodos explícitos

En los métodos explícitos en CPU utilizando OpenMP, los bucles se paralelizaron mediante el uso de macros `#pragma omp parallel` para abrir las regiones paralelas y `#pragma omp for` para paralelizar los lazos, entre otras cosas, teniendo en cuenta que el almacenamiento en memoria y el recorrido que hacen los procesos sobre ella aproveche la ubicación contigua en memoria (almacenamiento tipo *row-major* o *col-major*). Además de lo anterior no hay mayores diferencias en cuanto a la estrategia de paralelización en CPU y GPU para los métodos explícitos.

Para ilustrar el algoritmo del método explícito se utilizará la siguiente notación: C^*_{input}, C^*_{output} indican punteros a los únicos dos vectores alojados en la memoria global de la GPU. El algoritmo 6 describe el método explícito.

Algoritmo 6: Algoritmo explícito FE para la ecuación de advección difusión 2D.

1 inicialización: $C^*_{\text{input}} \leftarrow C^n$; $t \leftarrow T_{\text{inicial}}$;
2 **mientras** $t < T_{final}$ **hacer**
3 $t \leftarrow t + \Delta t$;
4 compute_kernel_interior<<<grid, block>>>(C^*_{output}, C^*_{input});
5 compute_kernel_edge_north<<<grid.x, block.x >>>(C^*_{output}, C^*_{input});
6 compute_kernel_edge_west<<<grid.y, block.y >>>(C^*_{output}, C^*_{input});
7 compute_kernel_vertex_SW<<<1, 1>>>(C^*_{output}, C^*_{input});
8 ...
9 sincronización de la GPU: `cudaDeviceSynchronize();`
10 intercambio de punteros: $C^*_{\text{input}} \longleftrightarrow C^*_{\text{output}}$;
11 **fin**

De hecho en la implementación el bloque central del código es una única función `next_step(`C^*_{output}, C^*_{input}`)` que llama a la ejecución de los (1+4+4) kernels que actualizan la solución en todo el dominio computacional. Note que al tratarse de un método explícito todos los kernels pueden ejecutarse de forma concurrente o independiente debido a la no dependencia de datos. El algoritmo en CPU es idéntico con la diferencia que no se requiere una configuración de grillas y bloques, por ello no se presentarán aquí.

Para el método explícito RK2 se requiere almacenar un tercer vector $C^*_{\text{output},2}$. El algoritmo 7 presenta la implementación utilizada.

Donde `next_step()` es la función utilizada para el método FE mientras que la función `AXPBY(x,y,output,a,b)` realiza la operación output $\leftarrow ax + by$.

Finalmente cabe mencionar que para las implementaciones en GPU, debido al poco rehúso de datos que puede hacerse en las operaciones involucradas, se ha omitido el uso de la memoria compartida y solo se utiliza la memoria de registro y lecturas a memoria global de la GPU.

Algoritmo 7: Algoritmo explícito RK2 para la ecuación de advección difusión 2D.

inicialización: $C^*_{\text{input}} \leftarrow C^n$; $t \leftarrow T_{\text{inicial}}$;
mientras $t < T_{final}$ **hacer**
  $t \leftarrow t + \Delta t$;
  `next_step` (C^*_{output}, C^*_{input}) ;
  `next_step` ($C^*_{\text{output,2}}$, C^*_{output}) ;
  `AXPBY` ($C^*_{\text{output,2}}$, C^*_{input},C^*_{output},0.5,0.5) ;
  sincronización de la GPU: `cudaDeviceSynchronize()` ;
  intercambio de punteros: $C^*_{\text{input}} \longleftrightarrow C^*_{\text{output}}$;
fin

Paralelización del método implícito

Para el algoritmo implícito se requiere resolver un sistema lineal de ecuaciones $Ax = rhs$ que si bien en esta prueba la matriz resulta simétrica debido al uso del esquema CD, en caso de utilizar otros esquemas de HO como por ejemplo FROMM, SOU o QUICK, la matriz resulta no simétrica, entonces se utilizó directamente el solver BiCGStab implementado en GPU y CPU de acuerdo a lo descrito secciones atrás. Cabe mencionar que si bien el costo del método BiCGStab es el doble que CG, la cantidad de iteraciones necesarias en el sistema simétrico se reduce prácticamente a la mitad, entonces los tiempos de cómputo se ven alterados usando uno u otro esquema. En cuanto a requerimiento de memoria, en BiCGStab se necesita almacenar el doble de vectores pero eso no fue una limitante en este estudio.

Para el caso implícito, la diferencia principal entre la implementación CPU y GPU es que mientras que en GPU se realiza todo utilizando estilo matrix free, en CPU se almacenan las 5 diagonales del sistema debido al mayor espacio disponible de memoria RAM, además de ello, se almacenan en vectores de igual tamaño completando con 0 las sub diagonales W,E,N y S, con ello se logra que la operación SMPV de producto de matriz dispersa por vector se realiza de forma paralela sin condicionales directamente en un bucle de 0 a $N_x N_y$ paralelizado mediante la API de OpenMP. En cuanto a las operaciones de reducción requeridas en el solver, se utilizaron directivas como `#pragma omp for reduction(+:result)` para el producto interno entre vectores.

De forma resumida, se implementó una función que actualiza el lado derecho `update_RHS()` en GPU y otra para la versión CPU usando OpenMP, una función que realiza la operación tipo stencil en GPU `stencil_op(x,y)` mientras que en CPU se tiene una función que llena las diagonales de la matriz y posteriormente dicha matriz es usada por la función `SMPV(A,x,y)`. El algoritmo 8 presenta la implementación utilizada en GPU, siendo similar, la implementación en CPU.

Algoritmo 8: Algoritmo implícito CN para la ecuación de advección difusión 2D.

inicialización: $C^*_{\text{input}} \leftarrow C^n$;
mientras $t < T_{final}$ **hacer**
  $t \leftarrow t + \Delta t$;
  computar RHS vía kernels concurrentes: `update_RHS` (rhs^*,C^*_{input}) ;
  sincronización: `cudaDeviceSynchronize()` ;
  llamada al solver: `BiCGStab` (C^*_{output}, rhs^*) ;
  intercambio de punteros: $C^*_{\text{input}} \longleftrightarrow C^*_{\text{output}}$;
fin

3.5.4. Validación: convergencia del esquema temporal

Se realizó el estudio de convergencia temporal solamente para el método implícito (CN), para ello se eligió el tamaño de grilla intermedio $N_x = N_y = 4096$, los pasos de tiempo elegidos para la prueba fueron de la forma $\Delta t = 2\pi/k$ s donde k representa la cantidad de pasos de tiempo en cada caso y adopta los valores de la siguiente lista:

$$k = 164, 328, 656, 1309, 2618, 5235, 7853, 10470, 20940 \tag{3.10}$$

La figura 3.7 presenta los resultados obtenidos, aquí $||\epsilon_C||_2$ indica el error en el campo C evaluado utilizando la norma L_2 que es equivalente al RMS (*Root Mean Square*) entre la solución numérica y la analítica:

$$RMS = \sqrt{\left(\sum_{i=1}^{N_x N_y} (C_{\mathrm{i,exac}} - C_{\mathrm{i,num}})^2\right) / N_x N_y} \tag{3.11}$$

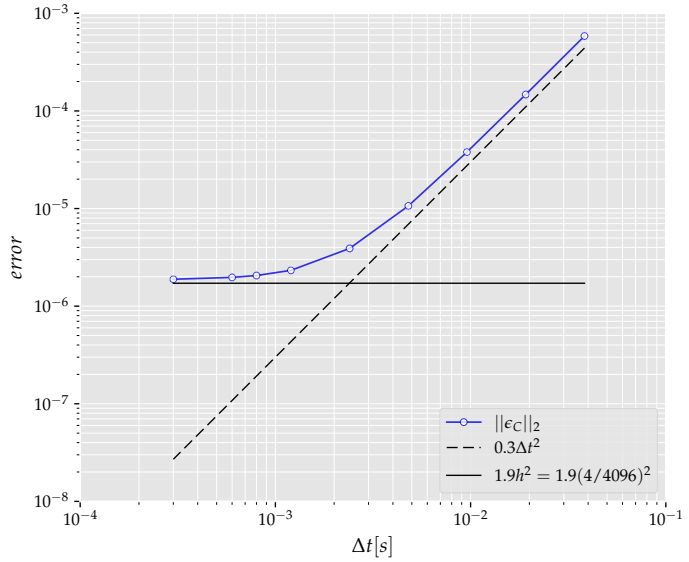

Figura 3.7: Estudio de la convergencia del esquema temporal CN $N_x = N_y = 4096$.

Para más claridad, se graficaron también las rectas $0{,}3\Delta t^2$ y la recta $1{,}9h^2$ que representan los órdenes de convergencia del esquema temporal y espacial respectivamente.

Puede observarse que $\epsilon_1/\epsilon_2 \sim (\Delta t_1/\Delta t_2)^2$ con una pendiente en escala logarítmica aproximada de $(\log \epsilon_1 - \log \epsilon_2)/(\log \Delta t_1 - \log \Delta t_2) \sim 2$ que es lo que corresponde a un esquema de segundo orden. A su vez, el error se reduce hasta alcanzar un valor del orden de precisión del esquema espacial ($O(h^2)$).

3.5.5. Validación: convergencia del esquema espacial

Se realizó el estudio de convergencia en malla utilizando los tamaños de grilla indicados en la tabla 3.1. Para diseñar la prueba se siguieron estos criterios:

- Para los esquemas explícitos la condición de estabilidad esta dada de forma aproximada por la expresión (3.8). Se comenzó con la prueba del esquema RK2 debido a que en este caso el error temporal ($O(\Delta t^2)$) es muy inferior al error espacial esperado ($O(h^2)$) debido a que Δt se elije de la condición de estabilidad. Con lo anterior puede estimarse cual es el error real del esquema espacial en cada tamaño de grilla.

- Para el esquema FE, si se utilizan estos mismos pasos de tiempo, se observó que los errores están casi en un orden de magnitud encima de los errores del esquema RK2. Como el esquema FE es $O(\Delta t)$ si se quiere alcanzar un error similar Δt debe reducirse en la misma proporción de los errores de ambos métodos, esto es, si se quiere que el error del esquema temporal sea inferior al del esquema espacial, para el método FE debe elegirse un paso de tiempo nuevo con el siguiente criterio. $\Delta t_{\text{new}}/\Delta t_{\text{stable}} \sim ||\epsilon_{\text{RK2}}||/||\epsilon_{\text{FE}}||$.

- Para el método implícito el error del esquema CN es $O(\Delta t^2)$, una primera aproximación es optar por elegir $\Delta t_1 \sim (||\epsilon_{\text{RK2}}||)^{1/2}$. Sin embargo al tratarse de esquemas diferentes es muy probable que los errores ϵ_{CN} y ϵ_{RK2} difieran, aunque ambos estén en el mismo orden, esto significa que eligiendo pasos de tiempo inferiores a Δt_1 el error podría reducirse y en ese caso, la posterior evaluación del desempeño no sería justa debido a que el error del esquema temporal supera al error del esquema espacial. En efecto, para el esquema CN se eligió el paso de tiempo más grande tal que el mismo sea inferior al error de aproximación espacial (observe figura 3.7).

En la tabla 3.1 se indican la cantidad k de pasos de tiempo elegidos para cada uno de los métodos y mallas, similar a como se indicó antes el tamaño del paso de tiempo viene dado por $\Delta t = 2\pi/k$.

$N_x = N_y$	128	256	512	1024	2048	4096	8192	16384
# Mcell	0.0164	0.0655	0.2621	1.0485	4.1943	16.777	67.108	268.435
k_{CN}	165	315	645	1299	2553	5235	10469	29233
k_{RK2}	740	1847	5730	19690	72320	276405	1079880	4268046
k_{FE}	740	1847	5730	19690	72320	276405	1079880	4268046
$k_{\text{FE,fine}}$	2960	11082	45840	187055	723200	3040455	11878680	46948506

Tabla 3.1: Dimensiones de los casos de prueba utilizados para el estudio de convergencia espacial y número de pasos de tiempo adoptados para simular 2πs de simulación en el problema de advección difusión 2D.

Siguiendo los criterios mencionados anteriormente, se elaboró la figura 3.8 donde puede observarse que en los 3 métodos la convergencia es a orden $O(h^2)$. En la figura se incorporaron las rectas h vs h y h vs h^2 para facilitar la comparación. Puede observarse que eligiendo el máximo paso de tiempo estable para el método FE los errores del esquema temporal están muy por encima del error de la malla, aquí se observa que los errores tienden a una pendiente de segundo orden. Sin embargo, esto no representa directamente la convergencia del método ya que esto ocurre porque para $h \to 0$, en el denominador de (3.8) escrito en la forma $4\nu/h^2 + 2/h$, el primer término pesa más que el segundo y luego al refinar la malla siempre se está reduciendo el paso de tiempo en proporción a h^2.

Puede observarse que el error del esquema FE al refinar el paso de tiempo (Δt_{fine} en la gráfica) se puede reducir hasta alcanzar un error que está en el orden de los demás esquemas, asegurando de esta manera que la convergencia se debe a la precisión espacial sin influencia del error del esquema temporal.

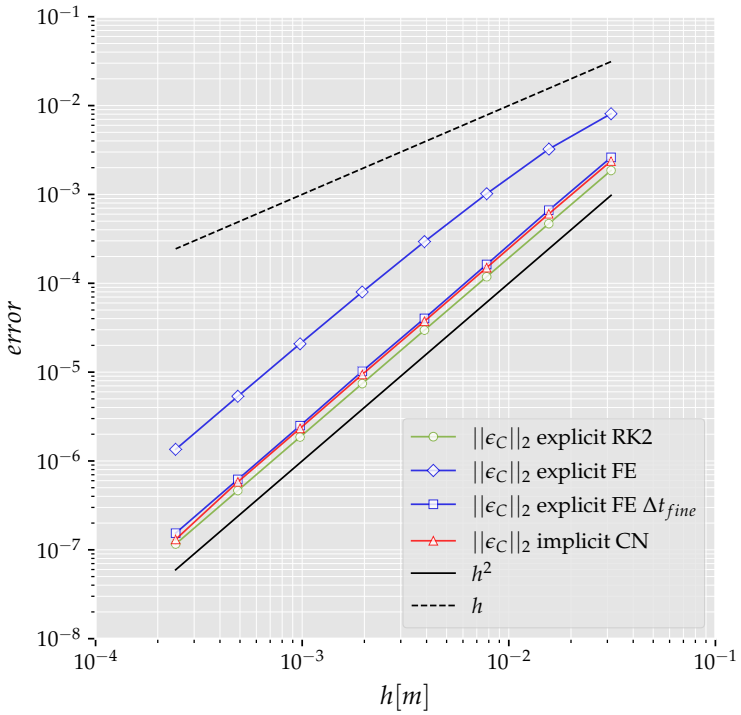

Figura 3.8: Estudio de la convergencia del esquema espacial de los métodos explicitos (FE y RK2) e implícito (CN).

3.5.6. Evaluación del desempeño

Para medir la eficiencia de los algoritmos se valúa la velocidad a la que se procesan las celdas de la siguiente manera. Para el método FE se calcula:

$$\text{rate}_{\text{FE}} = \frac{N_{\text{steps}} \times N_x \times N_y}{t_{\text{total}} \times 10^6} \left[\frac{Mcells}{sec}\right] \tag{3.12}$$

donde N_{steps} es el número de pasos de tiempo realizados, t_{total} el tiempo de cómputo total de la simulación en segundos. Para el método RK2, se requiere el equivalente de 2 pasos explícitos tipo FE y una operación AXPBY que omitimos su costo frente al resto.

$$\text{rate}_{\text{RK2}} = \frac{2N_{\text{steps}} \times N_x \times N_y}{t_{\text{total}} \times 10^6} \left[\frac{Mcells}{sec}\right] \tag{3.13}$$

Para el caso implícito se estima una *rate* equivalente considerando que para cada paso de tiempo se requieren realizar varias iteraciones en el resolvedor BiCGStab (incluye 2 operaciones tipo stencil, cada una similar a un paso FE) y calcular el lado derecho (operación equivalente a un paso FE), de esta manera la tasa viene dada por:

$$\text{rate}_{\text{CN}} = \frac{(2N_{\text{it,BiCGStab}} + N_{\text{steps}}) \times N_x \times N_y}{t_{\text{total}} \times 10^6} \left[\frac{Mcells}{sec}\right] \tag{3.14}$$

Evaluación de la tasa de procesamiento

En la tabla 3.2 se presentan los resultados obtenidos para las tasas de procesamiento, estos valores se graficaron en la figura 3.9-izquierda. Como se mencionó al comienzo de la sección las pruebas se han realizado en el equipamiento 2, utilizando 40 núcleos en la ejecución de las versiones CPU paralelas.

Puede verse que en la GPU se produce una saturación recién a partir de la grilla de 4096 celdas por dirección ($\sim$ 16,7 Mcell) mientras que en CPU la saturación ocurre a partir de la grilla de 512 celdas por dirección ($\sim$ 0,262 Mcell), cabe destacar que para el método implícito las tasas

de cómputo se reducen para los tamaños grandes indicando que las operaciones de reducción requeridas en el solver BiCGStab deterioran el rendimiento en la medida que el problema crece en número de celdas.

$N_x = N_y$	128	256	512	1024	2048	4096	8192	16384
# Mcell	0.0164	0.0655	0.2621	1.0485	4.1943	16.777	67.108	268.435
$\text{rate}_{\text{RK2,GPU}}$	203.45	829.33	2886.15	7746.68	16398	23190	25805	25688
$\text{rate}_{\text{RK2,CPU}}$	96.29	248.6	551.2	689.5	725.9	685.9	685.8	586.7
$\text{rate}_{\text{GPU}}/\text{rate}_{\text{CPU}}$	2.1	3.3	5.2	11.2	22.6	33.8	37.6	43.7
$\text{rate}_{\text{FE,GPU}}$	242.88	953.3	3831	9823	23469	36966	42684	42210
$\text{rate}_{\text{FE,CPU}}$	117.67	289.08	602.5	739.5	794.6	808.4	819.7	811.4
$\text{rate}_{\text{GPU}}/\text{rate}_{\text{CPU}}$	2.1	3.3	6.3	13.2	29.5	45.7	52.1	52.0
$\text{rate}_{\text{CN,GPU}}$	103.9	449.1	1420	3169	5037	5991	6304	6251
$\text{rate}_{\text{CN,CPU}}$	60.28	168.8	293.4	572.7	377.9	267.4	247.1	190.2
$\text{rate}_{\text{GPU}}/\text{rate}_{\text{CPU}}$	1.7	2.6	4.8	5.5	13.3	22.4	25.5	32.8

Tabla 3.2: Tasas de procesamiento en Mcell/s obtenidas para los diferentes métodos en GPU y CPU paralelo (40 cores) en el problema de advección difusión 2D.

De acuerdo a la tabla 3.2 la relación entre las tasas de cómputo en GPU y CPU que en este caso ya es la medida de la aceleración o speedup obtenido en cada algoritmo, puede verse que en FE se alcanza la mayor ganancia logrando factores de 50×, en segundo lugar el método RK2 donde se logra una aceleración de hasta 40× y finalmente el método implícito CN se logra aceleraciones de casi 33× en problemas del orden de 268 millones de celdas.

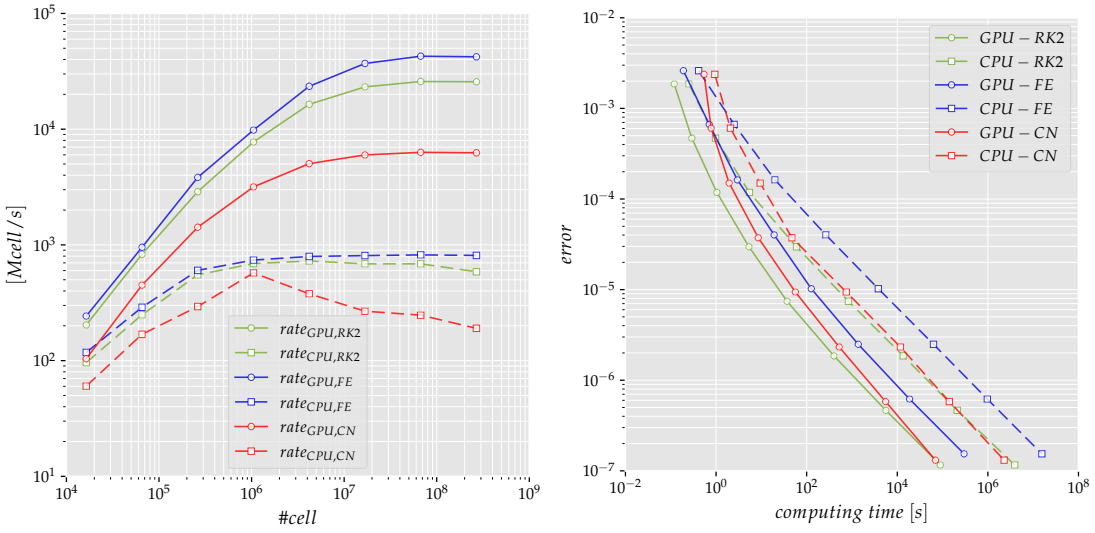

Figura 3.9: Eficiencia de los algoritmos en base a la tasa de procesamiento (izquierda) y comparación del error obtenido para diferentes tiempos de cómputo (derecha) para los métodos explícitos (FE y RK2) e implícito (CN).

Si se comparan las tasas de cómputo en GPU entre los métodos explícitos RK2 y FE, el método FE tiene un mejor desempeño (entre 1.2 y 1.6 veces mejor), sin embargo esto no compensa la cantidad de pasos de tiempos requeridos para obtener una solución con error del mismo orden que el método RK2 (compare tabla 3.3 y figura 3.9-derecha) siendo más rápido el último (entre 1.6 y 3.6 veces más rápido). Esta relación entre las tasas de cómputo de FE y RK2 puede explicarse en que para todos los algoritmos implementados estan limitados por el ancho de banda de la GPU en lugar de la capacidad de procesamiento debido al bajo número de operaciones requeridas en los métodos. En el caso de FE para cada paso de tiempo se requieren 2 lecturas/escrituras en memoria, para el método RK2 se requieren 7 lecturas/escrituras (4 corresponden a los 2 pasos tipo FE y 3 corresponden a la operación AXPBY), como en el cálculo del rate ya se tuvo en cuenta un factor de 2, si no se considera dicho factor la relación estaría entre 1.4 y 3.2, siendo que la relación entre

lecturas/escrituras de cada método es de 7 a 2. Es decir en RK2 requiere 3.5 mas operaciones a memoria global que en FE, la relación es algo menor debido a que en los pasos tipo FE se realizan algunos cómputos mientras y en un caso cada hilo debe leer 5 valores de un vector (lectura de la solución a tiempo anterior) y un solo valor del otro vector (escritura de la nueva solución) mientras que en la operación AXPBY cada hilo lee o escribe un valor por cada uno de los 3 vectores.

En lo que respecta al método implícito CN, las tasas de cómputo son entre 2 y 4 veces menores que el método RK2. Esto puede explicarse en la cantidad de lecturas que se requieren en los pasos del solver BiCGStab que no fueron consideradas en el cálculo (6 op. tipo AXPBY y 6 op. de producto interno que implican además una operación de reducción a un solo valor). Sin embargo esto se compensa en una ganancia en términos de pasos temporales requeridos para obtener una solución con el mismo error respecto del método RK2, con una relación que varía entre 4.5 y 146 (compare en tabla 3.1 k_{RK2} vs k_{CN}).

Errores y tiempos de cálculo

En la tabla 3.3 se presentan los errores de cómputo obtenidos (ver figura 3.8) y los respectivos tiempos de cómputo en GPU y CPU, además se presentan los tiempos en GPU y respectivos errores para la solución con FE con el máximo paso de tiempo estable que puede adoptarse.

En la figura 3.9-derecha se graficaron los valores de la tabla 3.3. Puede verse que para proble-

$N_x = N_y$	128	256	512	1024	2048	4096	8192	16384
# Mcell	0.0164	0.0655	0.2621	1.0485	4.1943	16.777	67.108	268.435
$\text{error}_{\text{RK2}}$	1.861e-03	4.689e-04	1.182e-04	2.968e-05	7.431e-06	1.858e-06	4.647e-07	1.161e-07
time_{GPU}	0.119	0.292	1.04	5.33	36.99	399.94	5617	89201
time_{CPU}	0.252	0.973	5.45	59.88	835.67	13520	2.113e+05	3.905e+06
$\text{error}_{\text{FE},\Delta t_1}$	2.609e-03	6.655e-04	1.627e-04	4.020e-05	1.023e-05	2.491e-06	6.198e-07	1.542e-07
time_{GPU}	0.190	0.707	2.96	19.17	126.58	1379	18676	2.985e+05
time_{CPU}	0.412	2.51	19.94	265.24	3817	63096	9.725e+05	1.553e+07
error_{CN}	2.376e-03	6.048e-04	1.499e-04	3.736e-05	9.410e-06	2.327e-06	5.814e-07	1.310e-07
time_{GPU}	0.544	0.781	1.95	8.56	56.28	524	5528	69685
time_{CPU}	0.938	2.08	9.42	47.37	750.23	11728	1.410e+05	2.291e+06
$\text{error}_{\text{FE},\Delta t_{\text{stab}}}$	8.104e-03	3.236e-03	1.017e-03	2.937e-04	7.979e-05	2.086e-05	5.340e-06	1.351e-06
time_{GPU}	0.050	0.127	0.392	2.10	12.92	125.44	1698	27142

Tabla 3.3: Errores y tiempos de cómputo en segundos obtenidos para los diferentes métodos en GPU y CPU paralelo (40 cores) en el problema de advección difusión 2D.

mas de hasta 4096×4096, el algoritmo más rápido en GPU es RK2, mientras que para problemas más grandes, resulta más rápido el método CN en GPU. La relación entre los tiempos de cómputo varía respectivamente entre 4.5 para 128×128 y se reduce en la medida que el número de celdas aumenta llegando a un factor de 0.78 para la grilla más fina 16384×16384. El comportamiento en CPU es un tanto diferente, y el método CN resulta más rápido que RK2 para número de celdas mayores a 512×512 llegando a ser hasta 1.7 veces más rápido CN que RK2.

El speedup entre GPU y CPU ya se discutió en la subsección anterior, se remarca que en todos los casos las implementaciones GPU resultan mas eficientes que su contraparte utilizando cómputo paralelo en CPU.

Para finalizar cabe mencionar que aunque los tiempos de cómputo del método FE en GPU, utilizando el paso de tiempo más grande permitido por la condición de estabilidad, son inferiores que los métodos RK2 y CN los errores resultan mayores en casi un orden de magnitud que éstos.

3.5.7. Conclusiones

Entre las principales conclusiones se puede decir que los tiempos de cómputo en GPU del método explícito RK2 e implícito CN no difieren en gran medida siendo más eficiente el método RK2 para problemas de hasta 16.7 millones de celdas, situación que se revierte para problemas con

mayor número de celdas donde el método implícito es más eficiente. Esta situación se debe a que el método explícito comienza a tener una fuerte restricción de estabilidad que se debe principalmente al término difusivo. Luego, en un problema de advección pura el método explícito resultará más rápido que el implícito. A pesar de esta diferencia en tiempos de cómputo, el método RK2 resulta en gran medida más sencillo de implementar que el implícito, más aún, en caso de utilizarse esquemas no lineales (TVD o HR) el método CN requiere la implementación de una estrategia de correción diferida o similar para tratar la no linealidad del esquema espacial.

En cuanto al método FE es el más sencillo de implementar sin embargo, se mostró que para lograr una buena eficacia en la resolución con métodos explícitos, se requiere un método de mayor orden como RK2 o AB para poder competir con un algoritmo implícito como CN.

Sobre las aceleraciones en GPU respecto a CPU, puede observarse que en general, se requiere un clúster de al menos 30 o 40 computadoras de similares características ($30 \times 40 = 1200$ o $40 \times 40 = 1600$ núcleos) para poder igualar una sola tarjeta de vídeo similar a la Nvidia Tesla V100.

Los resultados de este estudio muestran que aunque en la bibliografía se eligen los métodos explícitos en GPU, los métodos implícitos pueden tener una eficiencia similar en términos de tiempos de cómputo aunque su implementación es más compleja que los métodos explícitos. Finalmente la GPU resulta en una alternativa realmente económica para resolver problemas de gran tamaño en comparación con las variantes CPU que requieren de un clúster de tamaño medio.

3.6. Advección difusión 2D usando esquemas no lineales - TVD

El objetivo de este experimento es comparar la implementación GPU de los métodos TVD con las soluciones provistas por métodos lagrangianos, en un problema complejo de resolver a altos números de Péclet. Para el estudio, se realizan implementaciones en GPU utilizando esquemas de alta resolución (HR), donde se comparan el esquema explícito FE versus el esquema implícito CN para resolver un problema a $Pe = 100$ y advección pura ($Pe = \infty$).

3.6.1. Caso de estudio: evaluación de la macro dispersión en un medio altamente heterogéneo

El flujo y transporte en medios porosos resulta complejo debido a la heterogeneidad espacial de multiescala que presenta las propiedades hidráulicas del medio. Por lo tanto, los métodos numéricos precisos y eficientes son fundamentales para reproducir, comprender y predecir estos procesos.

Para esta prueba, se resuelve una ecuación de advección-dispersión que es similar a la ecuación (2.1) pero considerando $f = 0$ y con $\mathbf{D}$ representando el tensor de dispersión hidrodinámica (m^2/s) definido por:

$$\mathbf{D} = \frac{1}{\sqrt{u_x^2 + u_y^2}} \begin{bmatrix} \alpha_L u_x^2 + \alpha_T u_y^2 & (\alpha_L - \alpha_T)\, u_x u_y \\ (\alpha_L - \alpha_T)\, u_x u_y & \alpha_L u_y^2 + \alpha_T u_x^2 \end{bmatrix} + \begin{bmatrix} D_m & 0 \\ 0 & D_m \end{bmatrix} \tag{3.15}$$

donde α_L y α_T son la dispersividad longitudinal y transversal respectivamente en m, D_m es el coeficiente de difusión molecular en m^2/s y $\mathbf{u} = (u_x, u_y)$ es el campo de velocidades en m/s. Este problema es complejo de resolver y en la mayoría de trabajos se utilizan métodos lagrangianos, entonces aquí se busca mostrar que los métodos eulerianos implementados en GPU pueden resolver con buena precisión este tipo de problemas en tiempos reducidos.

La idea es aplicar los métodos descritos en el capítulo 2 para la caracterización de la macro dispersión en un conjunto de casos con diferentes grados de heterogeneidad. El experimento consiste realizar simulaciones de transporte en dominios 2D con campos de velocidad heterogéneos, calculados a partir de campos de conductividad hidráulica $K(x)$ lognormalmente distribuidos con

covarianza exponencial. A partir de las soluciones, se evalúan las estadísticas para los coeficientes de macro dispersión longitudinal y transversal utilizando el método de los momentos. En el marco de esta tesis, los resultados que se presentarán en las próximas subsecciones fueron publicados en [21], allí se pueden encontrar más detalles sobre las pruebas llevadas a cabo y antecedentes bibliográficos de esta temática. En lo que sigue solamente se resumirán las principales secciones.

Configuración del dominio computacional

Se utilizaron las mismas grillas computacionales para la generación del campo de conductividad, el cálculo del flujo y la simulación del transporte. Las dimensiones del dominio en cada caso se definieron en función $\sigma_{\ln K}$ donde $\sigma^2_{\ln K}$ representa la varianza del logaritmo de la conductividad. En efecto, para obtener los valores de macro dispersión asintóticos de forma correcta, las dimensiones consideradas fueron: $L_x \times L_y \in \{205\lambda \times 204{,}8\lambda; 410\lambda \times 410\lambda; 820\lambda \times 820\lambda; 820\lambda \times 1640\lambda\}$, donde λ es la longitud de correlación, de esta forma, con una resolución de $\lambda/h = 10$ (h: paso de malla) se obtienen grillas de 4.2 a 134.4 millones de celdas.

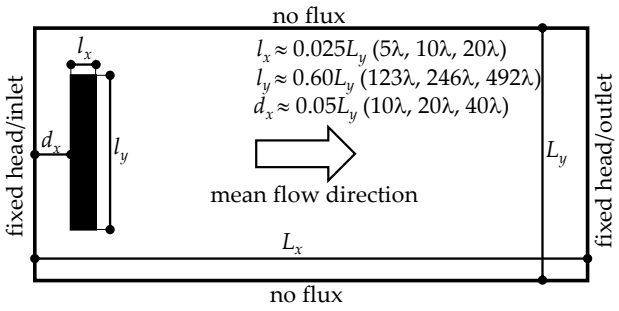

Figura 3.10: Definición del dominio del problema y condiciones de borde para la ecuación de flujo y ecuación de transporte.

Las estadísticas están basadas en experimentos tipo Monte Carlo (MC) considerando 100 campos de conductividad aleatorios para cada caso o tamaño de problema estudiado. La condición inicial para la ecuación de transporte es una ventana de $l_x \times l_y \approx 0{,}025L_y \times 0{,}6L_x$ de alta concentración centrada verticalmente en el dominio y ubicada a una distancia $d_x \approx 0{,}05L_y$ desde la entrada de flujo (ver tabla 3.4). En cuanto a la ecuación de flujo y su solución numérica y la generación de los campos de conductividad $K(\mathbf{x})$, no la describiremos aquí ya que se aborda en el siguiente capítulo (ver segundo caso de estudio) o también, puede encontrarse en [21]. Para la ecuación de transporte se consideran las condiciones de borde de no flujo (bordes *north* y *south*) y sólo flujo advectivo (bordes *west* y *east*). En la figura 3.10 se presenta la configuración de dominios utilizada.

Discretización espacial y temporal

La discretización de la ecuación de transporte requiere el uso de esquemas HR debido a que se trata de problemas a altos valores de Péclet, además de ello para el método implícito se debe aplicar la estrategia DC para tratar la no linealidad del esquema espacial. La implementación se

basó en el esquema MINMOD [145]. Para la discretización temporal se aplicó el método (θ):

$$\left(\frac{C^{t+\Delta t}-C^{t}}{\Delta t}\right)_C h^2 = (1-\theta)\left[h\sum_{F\sim NB(C)}\mathbf{D}_f\left(\frac{C_F-C_C}{h}\right)-\sum_{f\sim nb(C)}\dot{m}_f C_f^{HR}\right]^{t} +\theta\left[h\sum_{F\sim NB(C)}\mathbf{D}_f\left(\frac{C_F-C_C}{h}\right)-\sum_{f\sim nb(C)}\dot{m}_f C_f^{HR}\right]^{t+\Delta t} \tag{3.16}$$

La expresión para el método explícito surge al reemplazar $\theta = 0$:

$$\begin{aligned} C_C^{n+1} &= C_C^n + \frac{\Delta t}{h^2}\left[\sum_{F\sim NB(C)}\mathbf{D}_f(C_F^n-C_C^n)-\sum_{f\sim nb(C)}\dot{m}_f C_f^{n,HR}\right]; \\ \text{con } \dot{m}_f C_f^{n,HR} &= \left[C_C^n+\frac{1}{2}\psi(r_f^{n,+})\left(C_F^n-C_C^n\right)\right]||\dot{m}_f,0|| \\ &\quad -\left[C_F^n+\frac{1}{2}\psi(r_f^{n,-})\left(C_C^n-C_F^n\right)\right]||-\dot{m}_f,0|| \end{aligned} \tag{3.17}$$

Nótese que $\psi(r_f^{n,+})$ y $\psi(r_f^{n,-})$ implica una no linealidad que en este caso se resuelve de forma explícita.

Para el método implícito CN se reemplaza $\theta = 1/2$, resultando en una ecuación no lineal que se resuelve iterativamente aplicando la técnica DC. De esta forma, en cada paso de tiempo puede resolverse el sistema no lineal de ecuaciones mediante el siguiente esquema iterativo:

$$\begin{aligned} &\frac{h^2}{\Delta t}C_C^{(k+1)}-\frac{1}{2}\left[\sum_{F\sim NB(C)}\mathbf{D}_f(C_F^{(k+1)}-C_C^{(k+1)})-\sum_{f\sim nb(C)}\dot{m}_f C_f^{(k+1),U}\right] \\ &= \frac{1}{2}\left[\frac{h^2}{\Delta t}C_C^n+\sum_{F\sim NB(C)}\mathbf{D}_f(C_F^n-C_C^n)-\sum_{f\sim nb(C)}\dot{m}_f C_f^{n,HR}\right] \\ &\quad -\frac{1}{2}\sum_{f\sim nb(C)}\dot{m}_f\left(C_f^{(k),HR}-C_f^{(k),U}\right) \end{aligned} \tag{3.18}$$

aquí se considera el superíndice (k) para la solución de la iteración interna, eligiendo $C^{(0)} = C^n$ en cada paso de tiempo y una vez alcanzada la convergencia se considera $C^{n+1} = C^{(k+1)}$. Debe notarse que el sistema lineal de ecuaciones debe resolverse para cada iteración y la componente no lineal se tiene en cuenta en el último sumando del RHS de forma explícita para cada iteración (ver estrategia DC descrita en capítulo 2).

Detalles de la implementación GPU

La estrategia adoptada para la implementación en CUDA es similar a la del ejemplo mostrado en la sección anterior. En cuanto a los accesos a los datos en la memoria global de la GPU, para el método explicito cada kernel requiere 14 accesos a memoria por cada celda (5+5 corresponden al campo C^{n+1} y C^n, 2+2 corresponden a u_x y u_y) en el caso de advección pura y advección difusión. Mientras que para el caso de advección dispersión, se requieren 26 accesos (5+9 correspondientes a los campos C^{n+1} y C^n, 6+6 corresponden a u_x y u_y).

El mayor número de accesos para el último caso se debe a la necesidad de interpolar las diferentes componentes de los gradientes en las caras (recordar que el tensor $\mathbf{D} \in \mathbb{R}^{2\times 2}$ no es diagonal):

$$\left(\frac{\partial C}{\partial y}\right)_e = \frac{1}{2}\left[\frac{(C_{NE} - C_{SE})}{2h} + \frac{(C_N - C_S)}{2h}\right] \tag{3.19}$$

Además, para obtener todas las componentes del campo de velocidad en caras, se necesita computar algunos promedios, por ejemplo para obtener la componente y de la velocidad la cara east de la celda se requieren los valores en las caras n, s de la celda C y los valores vecinos de la celda E denotados como nE y sE:

$$(u_y)_e = \frac{1}{4}\left[(u_y)_n + (u_y)_s + (u_y)_{nE} + (u_y)_{sE}\right] \tag{3.20}$$

El algoritmo explícito no se presentará debido a que es idéntico al implementado para el caso de estudio anterior (mediante una función del tipo `next_step()` que ordena la ejecución de los kernels concurrentes).

A continuación se discute la paralelización del método implícito. En este caso se requiere resolver un sistema lineal de ecuaciones de la forma $Ax = rhs$ con una matriz no simétrica, nuevamente se utiliza el método BiCGStab implementado en GPU usando el estilo matrix free. La estrategia de paralelización es utilizando funciones del tipo:

- `kernel_linear_operator` $(C^{(k+1)*}_{\text{output}}, C^{(k+1)*}_{\text{input}}, u^*_x, u^*_y)$;
- `kernel_compute_RHS` $(rhs^*, C^{n*}, C^{(k)*}, u^*_x, u^*_y)$;

para el operador lineal utilizado en el solver BiCGStab se requieren 14 y 26 accesos a memoria global de la GPU para cada celda, como en el caso explícito, un caso u otro dependerá si se modela o no la dispersión. Para el kernel que computa el RHS, se requieren 19 accesos (5+5+5 corresponden a C^n, $C^{(k)}$ y rhs, 2+2 para leer u_x y u_y) y 31 accesos (9+5+5 correspondientes a C^n, $C^{(k)}$ y rhs, 6+6 para leer u_x, u_y) de acuerdo al caso de transporte simulado (con o sin dispersión). El algoritmo (16) describe la implementación del método implícito.

Algoritmo 9: Método implícito para la ecuación de transporte 2D con esquema TVD utilizando el enfoque de corrección diferida (DC).

1 inicialización: $C^*_{\text{input}} \leftarrow C^n$;
2 **mientras** $t < T_{final}$ **hacer**
3 $\quad t \leftarrow t + \Delta t$;
4 $\quad$ copiar sol. anterior como semilla: `cudaMemcpy` $(C^{(k)*}$, C^*_{input},`cudaMemcpyDeviceToDevice`);
5 $\quad dif \leftarrow 1{,}0$;
6 $\quad iter \leftarrow 0$;
7 $\quad$ **mientras** *(* $(dif > 1{,}0e-6)$ **and** $(iter < iter_{\text{máx}})$ *)* **hacer**
8 $\quad\quad$ computar RHS: vía kernels concurrentes;
9 $\quad\quad$ sincronización: `cudaDeviceSynchronize()`;
10 $\quad\quad$ llamada al solver: `BiCGStab` $(C^{(k+1)*}, rhs^*)$;
11 $\quad\quad dif \leftarrow ||C^{(k+1)} - C^{(k)}||_{L_\infty}$;
12 $\quad\quad iter \leftarrow iter + 1$;
13 $\quad$ **fin**
14 $\quad$ setear la nueva solución: $C^*_{\text{output}} \leftarrow C^{(k+1)*}$;
15 $\quad$ intercambio de punteros: $C^*_{\text{input}} \longleftrightarrow C^*_{\text{output}}$;
16 **fin**

Campo de conductividad y discretización de la malla

Se generaron campos de conductividad $\mathbf{K}(\mathbf{x})$ considerando mallas uniformes $\Delta x = \Delta y = h$ y con longitudes de correlación $\lambda = 10h$ para valores de varianzas $\sigma^2_{\ln K} \in \{0{,}25; 1{,}00; 2{,}50; 4{,}00; 6{,}25\}$.

Las figuras 3.11, 3.12 y 3.13 muestran un campo de conductividad generado para varianza $\sigma^2_{\ln K} = 6{,}25$ y el campo de velocidad determinado aplicando la ley de Darcy, calculado para diferentes varianzas ($\sigma^2_{\ln K} = 1{,}00$ y $\sigma^2_{\ln K} = 6{,}25$). Puede observarse que el campo de conductividad varía en rangos de hasta 10 órdenes de magnitud y la alta heterogeneidad que presenta produce grandes valores de velocidad localizada indicando caminos preferenciales para el transporte.

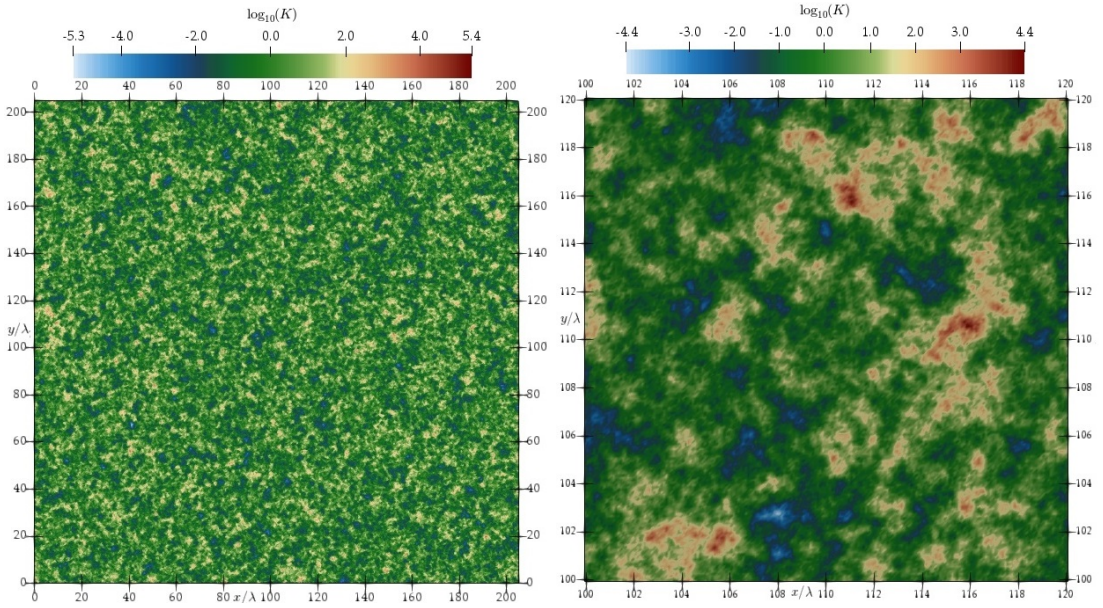

Figura 3.11: Campo de conductividad lognormal con varianza $\sigma^2_{\ln K} = 6{,}25$; longitud de correlación $\lambda = 10$; covarianza exponencial; para el dominio completo (izquierda) y una ventana interior de $[20x/\lambda \times 20x/\lambda]$ (derecha).

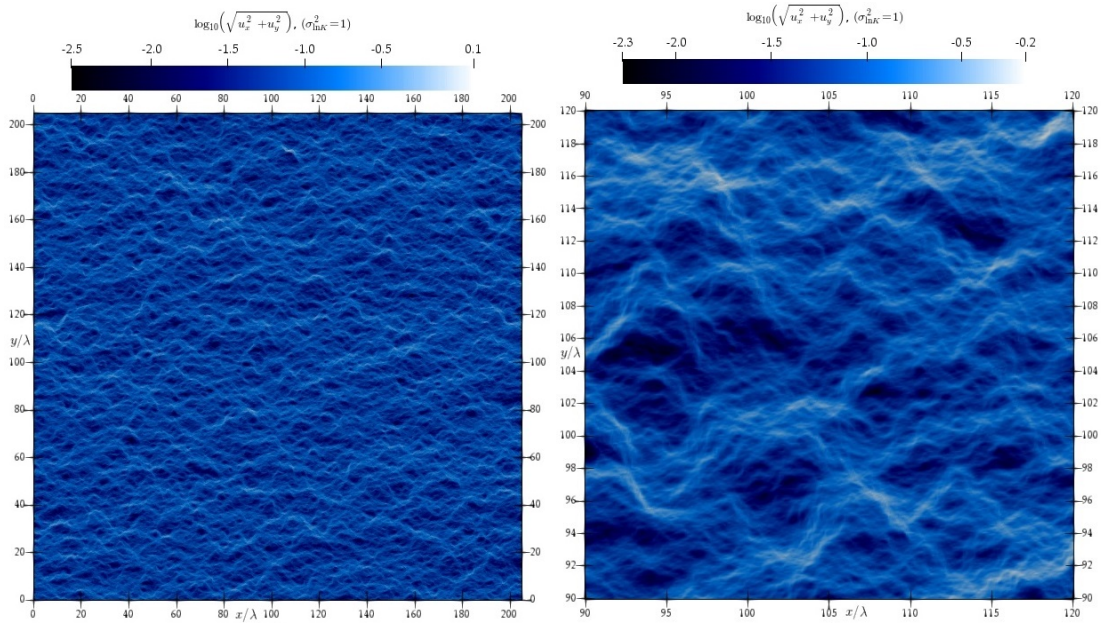

Figura 3.12: Campo de velocidad simulado para campo de conductividad lognormal con varianza de $\sigma^2_{\ln K} = 1$; longitud de correlación $\lambda = 10$; covarianza exponencial; para el dominio completo (izquierda) y una ventana interior de $[30x/\lambda \times 30x/\lambda]$ (derecha).

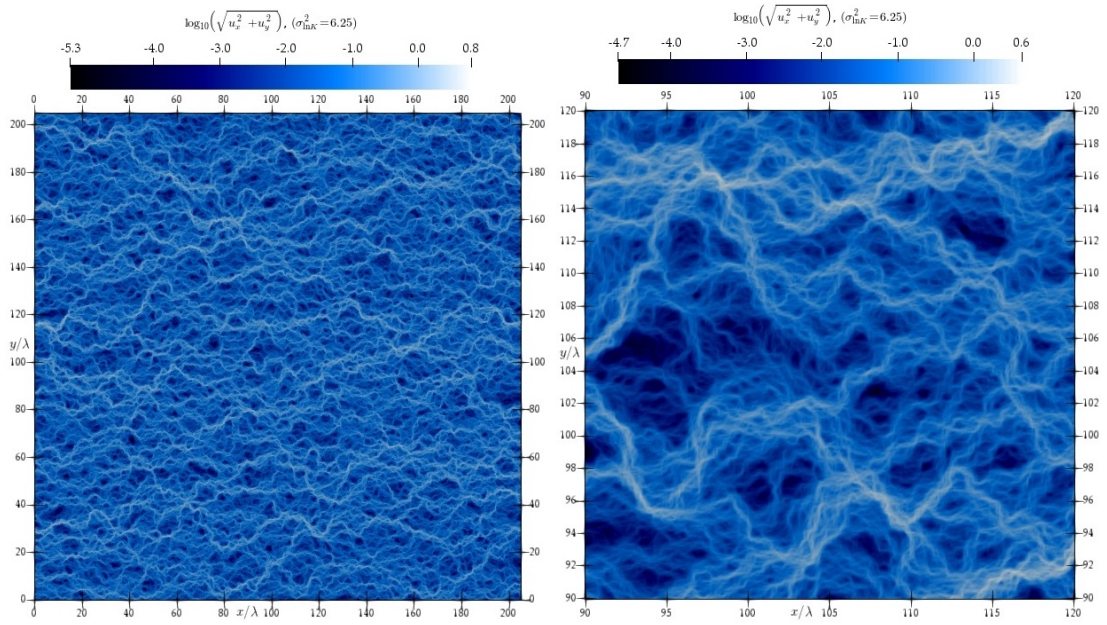

Figura 3.13: Campo de velocidad simulado para campo de conductividad lognormal con varianza de $\sigma^2_{\ln K} = 6{,}25$; longitud de correlación $\lambda = 10$; covarianza exponencial; para el dominio completo (izquierda) y una ventana interior de $[30x/\lambda \times 30x/\lambda]$ (derecha).

3.6.2. Simulación del transporte

Los casos de transporte considerados fueron advección pura (PA), advección difusión (AD) y advección dispersión isotrópica (Aα) con $\alpha_L = \alpha_T$, para los últimos dos casos se definen los números de Péclet como $Pe_d = \lambda u/D_m$ y $Pe_L = \lambda/\alpha_L$. Para las simulaciones MC se consideró un sólo valor igual a $Pe_d = Pe_L = Pe_T = 100$. En la tabla 3.4 se presentan las combinaciones de casos simuladas.

Tabla 3.4: Parámetros y tipos de casos considerados para las simulaciones numéricas.

Parámetros/caso	**valores**
$\sigma^2_{\ln K}$	0.25, 1, 2.25, 4, 6.25
Tamaño de grilla $\Delta x = \Delta y = h$	5m
Resolución: $\lambda/\Delta x = \lambda/\Delta y$	10
Modelo de covarianza	Exponencial isotrópica
Tipo de transporte	Advección pura $Pe = \infty$
	Advección difusión $Pe_d = 100$
	Advección dispersión $Pe_L = Pe_T = 100$
$[L_x/\lambda, L_y/\lambda]$	[205,204.8] para $\sigma^2_{\ln K} \in \{0.25, 1\}$
	[410,410] para $\sigma^2_{\ln K} = 2{,}25$
	[820,820] para $\sigma^2_{\ln K} = 4$
	[1640,820] para $\sigma^2_{\ln K} = 6{,}25$
Ventana de inyección	$l_x \times l_y = 0{,}025L_y \times 0{,}60L_y$
Número de simulaciones	$N = 100$

Las figuras 3.14 y 3.15 muestran cualitativamente los resultados numéricos para el mismo campo de conductividad considerando los 3 casos de transporte (Aα, AD y PA).

Para ver la influencia de la varianza de la conductividad en el transporte, en la figura 3.16 se presenta el transporte considerando advección difusión (AD) en un mismo instante de tiempo de simulación considerando diferentes grados de heterogeneidad en el campo de conductividades utilizado para computar el campo de velocidades $\mathbf{u}$.

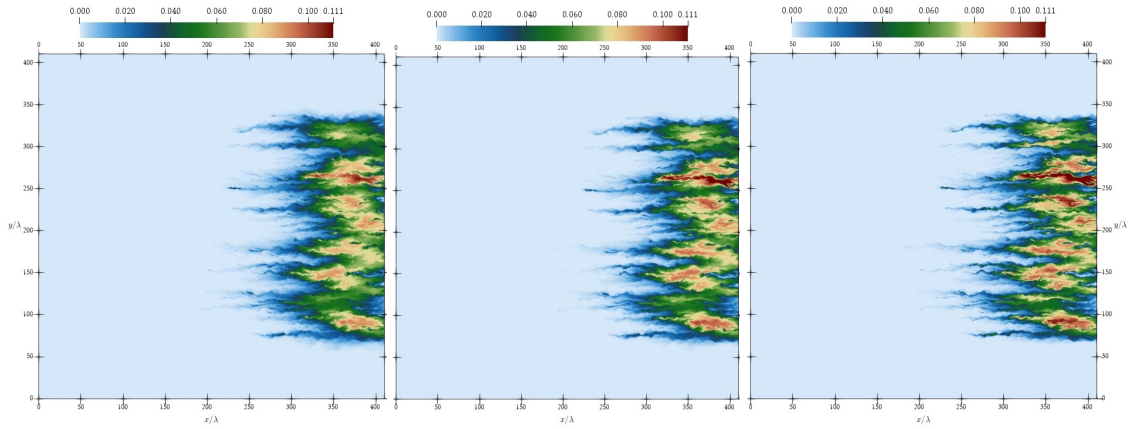

Figura 3.14: Comparación para diferentes tipos de transporte $\sigma^2_{\ln K} = 2{,}25$ al final de la simulación ($t_{\text{final}} = 350\lambda/\overline{u}$), de izquierda a derecha: Aα, AD and PA.

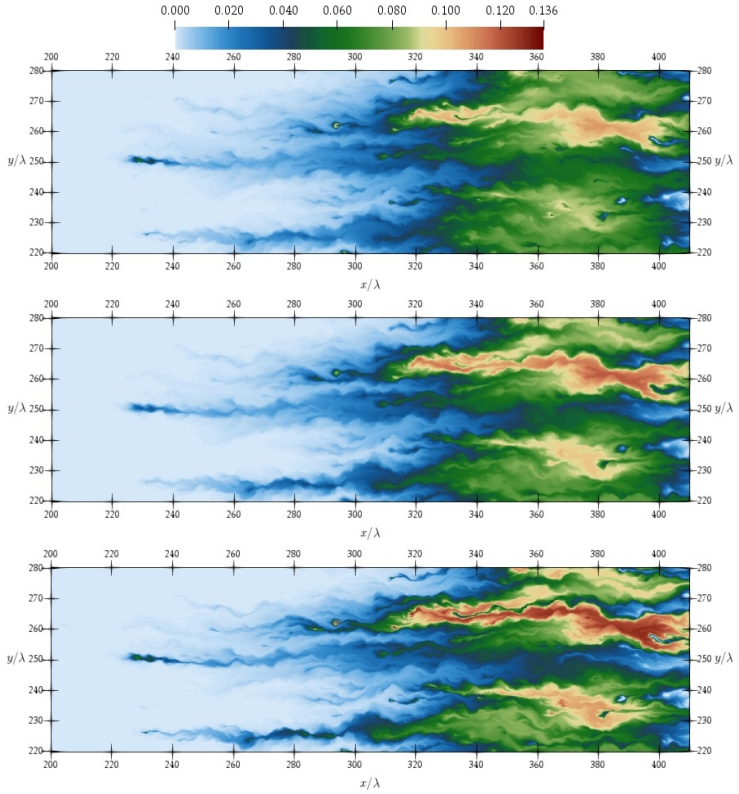

Figura 3.15: Comparación para diferentes tipos de transporte $\sigma^2_{\ln K} = 2{,}25$ al final de la simulación ($t_{\text{final}} = 350\lambda/\overline{u}$), en una ventana interior de $[60x/\lambda \times 21x/\lambda]$, de arriba hacia abajo: Aα, AD and PA.

En todos los casos puede verse que el efecto gobernante es la advección ya que la masa de soluto se mueve más o menos según los valores del campo $\mathbf{u}$. A su vez, en los casos de mayor heterogeneidad pueden observarse zonas de muy baja velocidad, donde la masa queda capturada hasta que eventualmente se puede mover por el efecto de la difusión.

Cómputo de la macrodispersión

Para la evaluación del coeficiente de macrodispersión, los momentos se calcularon de acuerdo a la siguiente expresión

$$M_{nm}(t) = \iint_{\Omega} x^n y^m C(x,t) dxdy \tag{3.21}$$

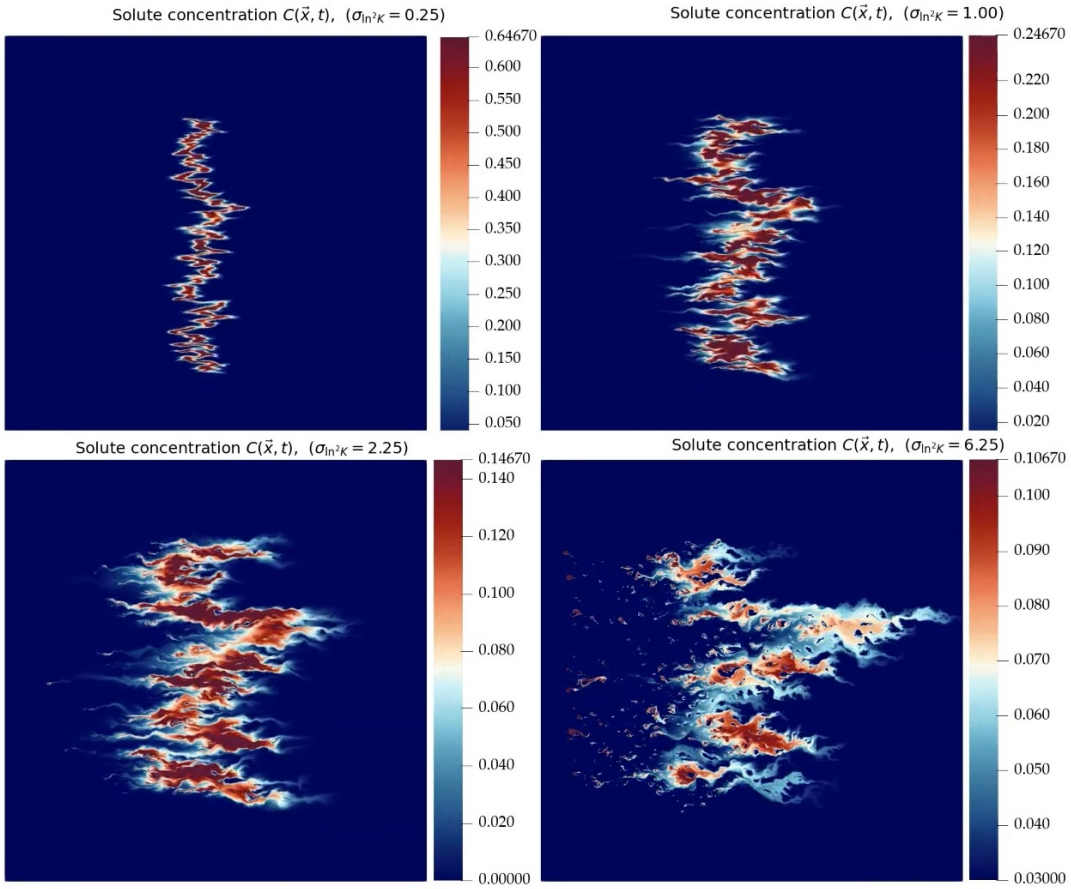

Figura 3.16: Comparación del transporte en un mismo instante de tiempo dado por $t = 100\lambda/\overline{u}$, considerando AD para diferentes grados de heterogeneidad dados por las varianzas $\sigma^2_{\ln K} = 0{,}25$, $\sigma^2_{\ln K} = 1{,}00$, $\sigma^2_{\ln K} = 2{,}25$, $\sigma^2_{\ln K} = 6{,}25$.

donde $m+n$ es el orden del momento (con $m, n \in \{0;1;2\}$) y Ω el dominio de simulación. Con esta expresión, los coeficientes de macrodispersion longitudinal y transversal vienen dados por:

$$D_L(t) = \frac{1}{2\lambda\overline{u}N}\sum_{i=1}^{N}\frac{d}{dt}\left[\frac{M_{20}(t)}{M_{00}(t)} - \left(\frac{M_{10}(t)}{M_{00}(t)}\right)^2\right] \tag{3.22}$$

$$D_T(t) = \frac{1}{2\lambda\overline{u}N}\sum_{i=1}^{N}\frac{d}{dt}\left[\frac{M_{02}(t)}{M_{00}(t)} - \left(\frac{M_{01}(t)}{M_{00}(t)}\right)^2\right] \tag{3.23}$$

aquí $\overline{u}$ indica la velocidad media de la pluma en la dirección x y $N = 100$ es el número de simulaciones para cada conjunto de parámetros.

Validación: comparación con métodos Lagrangianos

Como validación de los algoritmos eulerianos implementados, se reprodujeron los resultados reportados en la bibliografía obtenidos con métodos lagrangianos, como por ejemplo el método conocido como *random walk particle tracking*.

La figura 3.17 compara los coeficientes de macrodispersión longitudinal medio (promediado sobre 100 realizaciones) para los casos de AD y Aα y diferentes varianzas $\sigma^2_{\ln K}$ con los resultados obtenidos por [20, 44].

La figura 3.18 muestra los valores asintóticos para el caso PA. Puede verse que los resultados son similares a los obtenidos en [44].

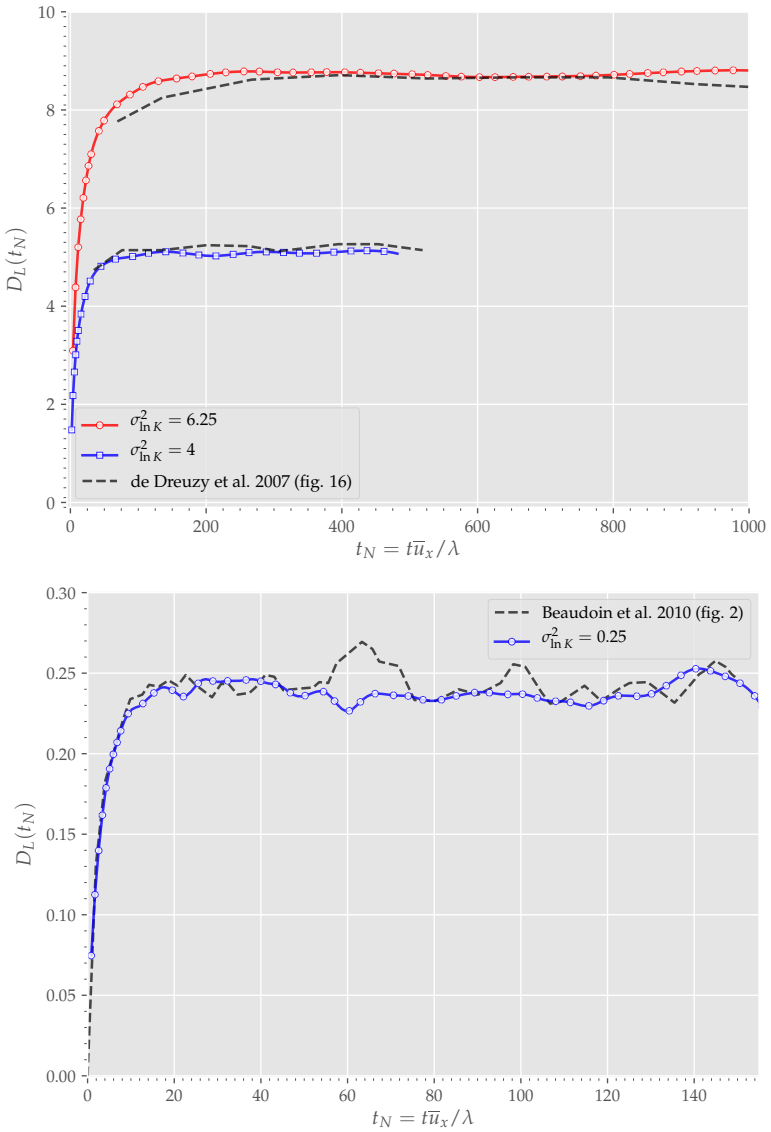

Figura 3.17: Comparación del coeficiente de macodispersión obtenidos con el presente método para diferentes tipos de transporte, AD (arriba) y Aα (abajo), y varianzas $\sigma^2_{\ln K}$ con los resultados de [20, 44] utilizando métodos lagrangianos.

Evaluación del desempeño de los algoritmos implícito y explícito

Se evaluó la precisión de los métodos implícito y explícito usando un esquema TVD mediante un análisis del error utilizando una solución de referencia obtenida con el algoritmo implícito ($O(\Delta t^2)$) considerando un paso de tiempo suficientemente pequeño. Para hacer una comparación justa, se estableció una relación entre los pasos de tiempo de cada método, que fue determinada experimentalmente para obtener un error del mismo orden.

Específicamente, para el método explícito, se utilizó el paso de tiempo más grande de forma que la solución permanece estable durante toda la simulación (tener en cuenta que la restricción ocurre debido a que en algunos pocos lugares del dominio hay velocidades muy grandes).

Para el caso implícito, el paso de tiempo se eligió para que el error cometido sea del mismo orden que el caso explícito. A modo de referencia, los pasos de tiempo utilizados en el método implícito son entre 8 y 34 veces más grandes que el considerado en el método explícito. La figura 3.19 muestra la evolución de la macrodispersividad longitudinal y transversal obtenida por ambos métodos para un caso en particular. Como puede verse ambos resultados están en el mismo orden.

La tabla 3.5 muestra el desempeño en términos del tiempo de cómputo requerido para cada

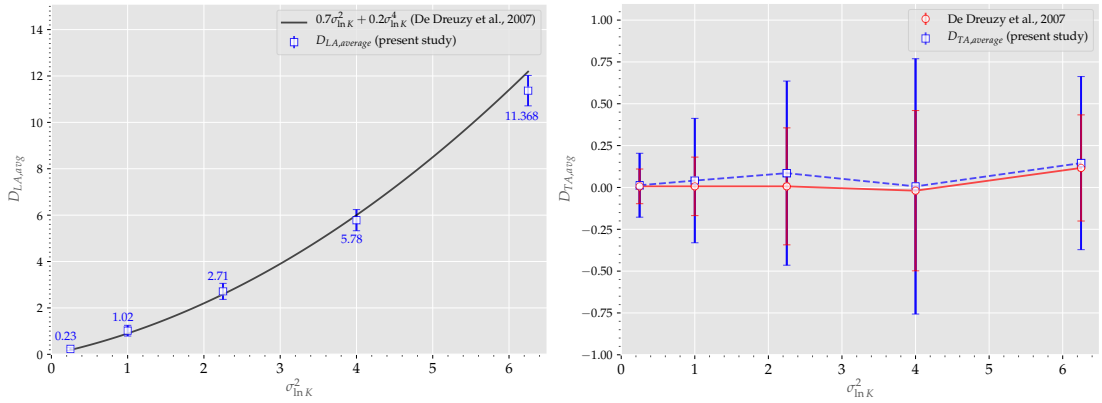

Figura 3.18: Coeficientes de macrodispersión longitudinal y transversal asintóticos como función de la log conductividad para el caso de advección pura. Las barras verticales indican la desviación estándar.

caso de transporte (promedio de 100 realizaciones), de acuerdo al tipo de transporte simulado, el método utilizado y el hardware donde se ejecutó la simulación.

Debe notarse que los tiempos de cómputo para los casos de transporte PA y AD son muy similares. Esto se debe al hecho de que las operaciones que se agregan en el caso AD corresponden a una multiplicación usando un valor fijo (D_m) que no implica una lectura a la memoria global de la GPU.

Para el caso Aα se agregan más operaciones y lecturas a la memoria global para poder interpolar el tensor de dispersividad en los centros de caras, lo cual afecta significativamente el desempeño.

Tabla 3.5: Comparación del tiempo de cómputo para los métodos implícito y explícito para los casos de transporte: Advección Pura (PA), Advección-Difusión (AD), Advección- Dispersión (Aα) ejecutados en diferentes equipos de cómputo.

		$\sigma^2_{\ln K}$	0.25	1	2.25	4	6.25
		$N_X \times N_Y$ [Mcell]	4.198	4.198	16.81	67.24	134.48
		$t_{\text{final}}\overline{u}/\lambda$	190	190	350	600	1000
Equipo - GPU	**Método**	**Caso de Transporte**	**Tiempo de cómputo promedio** [min]				
	Explícito	PA / AD	0.68	1.52	18.5	368	2333
	Explícito	Aα	1.23	3.88	60,1	1046	6331
Eq. 1 - Tesla K40	Implícito	PA	5.98	7.68	66,6	571	*
	Implícito	AD	3.88	5.92	55,3	525	*
	Implícito	Aα	6.35	9.05	76,3	697	*
	Explícito	PA / AD	0.10	0.22	3,69	46,6	311
	Explícito	Aα	0.17	0.52	8,16	122	833
Eq. 2 - Tesla V100	Implícito	PA	1.22	1.57	12,3	101	409
	Implícito	AD	0.78	1.22	10,2	92,6	391
	Implícito	Aα	1.15	1.65	12,9	115	443

* Excede la memoria RAM de la GPU del equipo 1.

Los tiempos reportados en la tabla 3.6 muestran que el algoritmo explícito es mucho más eficiente que el implícito en la mayoría de los casos. Por otro lado, como en el método implícito se requiere resolver un sistema no lineal de ecuaciones en cada paso de tiempo, el tiempo de cómputo total es sensible a los parámetros seteados en el solver BiCGStab y la tolerancia del bucle externo utilizado para la técnica de corrección diferida. El tipo de transporte simulado también afecta el desempeño, puede verse que las soluciones en los casos AD y Aα, al ser más suaves la convergencia es más rápida.

En la tabla 3.6 se comparan la proporción entre los tiempos de cómputo de los métodos implícito y explícito para diferentes casos. Esta tabla muestra que el efecto que tiene el tamaño del problema en el desempeño de los métodos para los tres tipos de transporte es que la proporción se

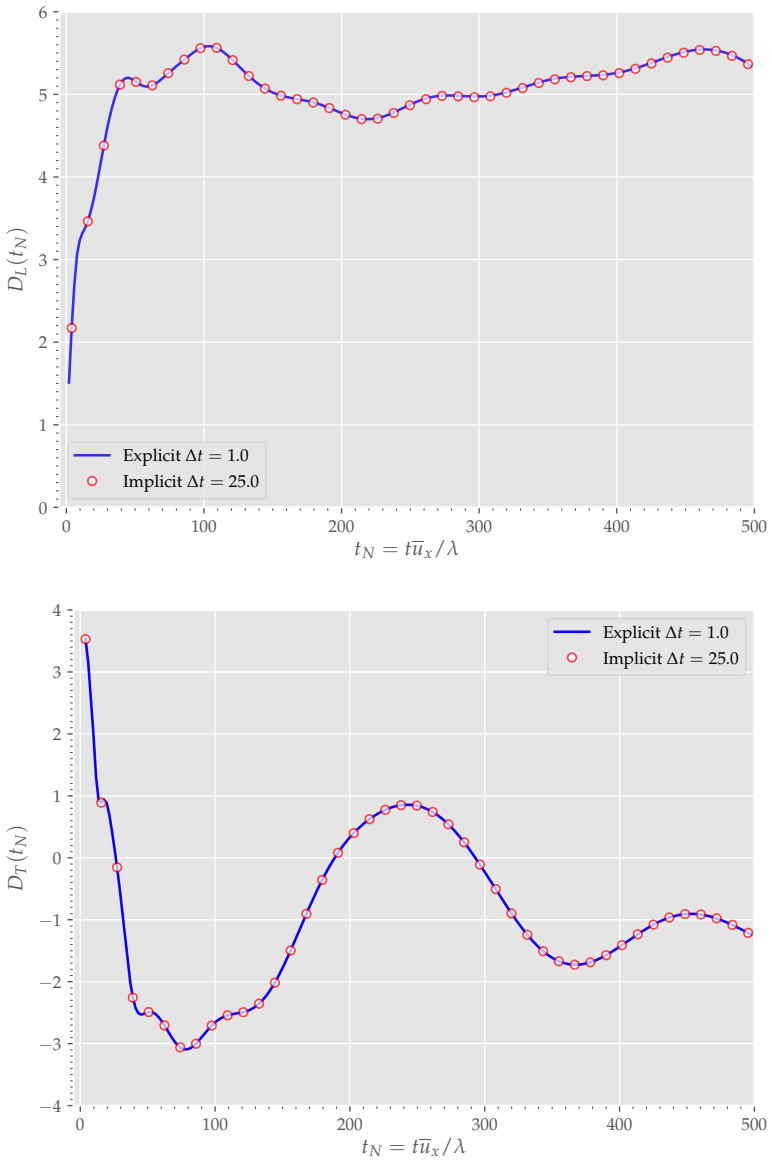

Figura 3.19: Coeficiente de dispersión longitudinal y transversal normalizado, para una realización con $\sigma^2_{\ln K} = 4$, obtenida usando el algoritmo explícito e implícito respectivamente.

reduce a medida que el problema crece.

Esto se explica principalmente porque cuando el sistema lineal crece, el esquema numérico considerado para el método implícito se vuelve más eficiente, debido a la estrategia utilizada que vincula el solver BiCGStab y la técnica DC. Dado que el resultado del solver lineal se usa dentro de un ciclo iterativo, se estableció un número máximo de iteraciones en lugar de forzar al solver a alcanzar una cierta precisión. Esta estrategia se puede adaptar dinámicamente monitoreando el residuo y aumentando o disminuyendo el número de iteraciones máximas del solver en cada iteración de lazo DC.

También puede observarse que si la solución es más suave como en el caso Aα, para el tamaño de dominio más grande el método implícito puede ser hasta 2 veces más rápido que el explícito. Por otro lado, el método explícito, a pesar de ser más lento que el implícito, requiere menos memoria, lo que permite resolver problemas de mayor tamaño. Este es una aspecto particularmente importante ya que la memoria de las GPU es limitada.

Como puede verse en las tablas 3.5 y 3.6 el caso de 134.48 millones de celdas no se puede resolver en la GPU K40 (el tamaño máximo para el método implícito es 107.08 Mcell y 299.63

Tabla 3.6: Relación entre los tiempos de cómputo de los métodos implícito y explícito para los diferentes casos de transporte y ejecución en diferentes equipos de cómputo.

	$\sigma^2_{\ln K}$	0.25	1	2.25	4	6.25
	$N_X \times N_Y$ [Mcell]	4.198	4.198	16.81	67.24	134.48
	$t_{\text{final}}\overline{u}/\lambda$	190	190	350	600	1000
Equipo - GPU	**Caso de transporte**	**Relación:** $t_{\text{implicit}}/t_{\text{explicit}}$				
Eq. 1 - Tesla K40	PA	8.8	5.1	3.6	1.6	*
	AD	5.7	3.9	3.0	1.4	*
	Aα	5.1	2.3	1.3	0.7	*
Eq. 2 - Tesla V100	PA	11.8	7.0	3.3	2.2	1.3
	AD	7.6	5.4	2.8	2.0	1.3
	Aα	6.8	3.2	1.6	0.9	0.5

* Método implícito excede la memoria RAM de la GPU del equipo 1.

Mcell para el caso explícito). Para la GPU V100 el tamaño máximo que se puede resolver es de 302.76 Mcell para el método implícito y 841 Mcell para el explícito.

3.6.3. Conclusiones

En este ejemplo se presentó la implementación de dos métodos en GPU para resolver el problema de transporte en el caso de advección dominante. Para ello se implementaron esquemas HR o TVD dando como resultados algoritmos con una precisión espacial de $O(h^2)$. Para el algoritmo implícito el esquema TVD se implementó usando la técnica iterativa de corrección diferida. Según la revisión bibliográfica esta es la primera vez que se utiliza este enfoque completamente euleriano para determinar la macrodispersión en medios altamente heterogéneos en un contexto de estudios MC.

Se mostró que los algoritmos resuelven correctamente problemas con gradientes pronunciados para advección pura, advección difusión y advección dispersión para valores de $Pe = 100$ considerando dominios de hasta 134.48 millones de celdas.

Para probar los métodos se consideró el problema desafiante de la macrodispersión longitudinal y transversal para varianzas de la log conductividad hasta $6{,}25$. La comparación con datos numéricos publicados obtenidos a partir de métodos como *random walk particle tracking* de alto rendimiento validan los esquemas numéricos propuestos.

Los tiempos de cómputo muestran que el método explícito, a pesar de ser un método $O(\Delta t)$ y por lo tanto requiere un paso de tiempo más pequeño, tiene un mejor desempeño en GPU en la mayoría de los casos. Este método, además de tener un algoritmo relativamente simple de implementar, requiere menos memoria que el implícito. El método implícito presentó mejor desempeño para problemas grandes. Este comportamiento se debe a la no linealidad del esquema advectivo utilizado. El motivo principal es que a medida que el sistema crece, la estrategia de cómputo adoptada en el método implícito se vuelve más eficiente debido al vínculo entre los parámetros del solver BiCGStab y la técnica DC. La desventaja de este método es que requiere más memoria GPU, lo que puede ser en varias ocasiones una limitación importante.

Los tiempos de cómputo reportados en este caso de estudio muestran que es factible llevar a cabo estudios de macrodispersión para medios altamente heterogéneos utilizando métodos totalmente eulerianos en GPU. Esto representa una alternativa potencialmente atractiva para resolver problemas que involucran la interacción de fases móviles y estacionarias, o el acoplamiento entre flujo y transporte, ya que los cálculos de flujo y transporte podrían realizarse sobre la misma malla (a diferencia de los métodos lagrangianos utilizados solamente para el transporte escalar).

3.7. Ecuación de difusión no lineal

Esta prueba consiste en la resolución de una ecuación de difusión no lineal en 3D usando el FVM, comparando diferentes esquemas temporales, explícito e implícito, considerando para el último, el método de NR y el solver CG (Gradientes Conjugados) para el sistema lineal de ecuaciones. El objetivo es evaluar el desempeño de cada esquema en GPU comparando precisión, velocidad de cálculo, tamaño de malla y también considerando el criterio de convergencia usado. Para evaluar las propiedades de convergencia de los diferentes esquemas en relación a la discretización espacial y temporal, se propone una solución analítica arbitraria, la cual satisface la ecuación diferencial mediante la elección de un término fuente elegido en función de la misma.

Se considera el problema parabólico no lineal en 3D para $t \in [0, T], T > 0$,

$$\begin{aligned} \frac{\partial \phi}{\partial t} = \nabla \cdot (\mathbf{A}(\phi) \nabla \phi) + f, \text{ en } \Omega \in \mathbb{R}^3, \\ \phi = 0, \text{ sobre } \partial\Omega, \\ \text{con } \phi(0) = \phi^0, \text{ en } \Omega \end{aligned} \tag{3.24}$$

donde $\mathbf{A}(\phi) = \text{diag}\,(a_1(\phi), a_2(\phi), a_3(\phi))$, es una función matricial a valores reales, acotada y estrictamente definida positiva. En adelante asumimos que existe una función $\phi(\mathbf{x}, t)$, con $\mathbf{x} = (x, y, z) \in \Omega$, suficientemente suave solución del problema (3.24). Para este ejemplo se considerará directamente el caso de difusión isotrópica con $a_i(\phi) = a(\phi)$, $i = 1, 2, 3$.

3.7.1. Discretización de la ecuación de difusión no lineal

Se transforma el problema (3.24) en un sistema de ecuaciones algebraicas usando el método de los volúmenes finitos (FVM), para luego resolver el problema discretizado en GPU.

Para la discretización espacial se consideró un dominio cúbico $\Omega = [0, 1] \times [0, 1] \times [0, 1]$, con mallas estructuradas de paso constante en cada dirección $\Delta x = \Delta y = \Delta z = h = 1/m$, siendo m la cantidad de celdas por dirección, con condiciones de borde $\phi(\mathbf{x}, t) = 0$ sobre $\partial\Omega$ y $t > 0$. Se consideró un coeficiente de difusión $a(\phi) = \kappa(\phi + 1)$, donde κ es análogo al coeficiente de difusión lineal, para las simulaciones se usó $\kappa = 1$. Se consideró la siguiente solución analítica:

$$\phi(\mathbf{x}, t) = 160 e^{-t}(x - x^2)(y - y^2)(z - z^2) \tag{3.25}$$

y se calculó f de forma $\phi(\mathbf{x}, t)$ safisfaga (3.24):

$$\begin{aligned} f = 160 e^{-t} xyz(x-1)(y-1)(z-1) \\ -320\kappa xy e^{-t}(x-1)(y-1)\left(160 xyz e^{-t}(x-1)(y-1)(z-1) - 1\right) \\ -320\kappa xz e^{-t}(x-1)(z-1)\left(160 xyz e^{-t}(x-1)(y-1)(z-1) - 1\right) \\ -320\kappa yz e^{-t}(y-1)(z-1)\left(160 xyz e^{-t}(x-1)(y-1)(z-1) - 1\right) \\ -25600 x^2 y^2 e^{-2t}(2z-1)^2(x-1)^2(y-1)^2 \\ -25600 x^2 z^2 e^{-2t}(2y-1)^2(x-1)^2(z-1)^2 \\ -25600 y^2 z^2 e^{-2t}(2x-1)^2(y-1)^2(z-1)^2 \end{aligned} \tag{3.26}$$

Con esta solución propuesta, la condición inicial para el problema es $\phi(\mathbf{x}, 0) = \phi^0 = 160(x - x^2)(y - y^2)(z - z^2)$. Mediante esta solución es posible evaluar los errores para las diferentes implementaciones que se realizaron.

Para aplicar el FVM al problema (3.24), primero se integra sobre la celda C:

$$\int_{V_C} \frac{\partial \phi}{\partial t} dV = \int_{V_C} \nabla \cdot (\mathbf{A}(\phi) \nabla \phi)\, dV + \int_{V_C} f dV \tag{3.27}$$

usando el teorema de la divergencia para el segundo término y el teorema del valor medio para el término temporal y fuente:

$$\begin{aligned}\left(\frac{\partial\phi}{\partial t}\right)_C V_C &= \int_{\partial V_C} (\mathbf{A}(\phi)\nabla\phi)\, dS + f_C V_C \Leftrightarrow \\ \left(\frac{\partial\phi}{\partial t}\right)_C h^3 = \sum_{f\sim nb(C)} (\mathbf{A}(\phi_f)\nabla\phi_f)\, h^2 + f_C h^3 &= \sum_{f\sim nb(C)} (a(\phi_f)\nabla\phi_f)\, h^2 + f_C h^3\end{aligned} \tag{3.28}$$

donde la sumatoria abarca todos los centros de cara f de la celda C. Haciendo la suma de las contribuciones a través de todas las caras de la celda C, usando diferencias centradas para el gradiente sobre las caras e interpolación lineal para el coeficiente de difusión $a(\phi)$ se obtiene la forma semi-discretizada de (3.28):

$$\left(\frac{\partial\phi}{\partial t}\right)_C h^3 = h^2 \left(\sum_{F\sim NB(C)} \left(\frac{a_C + a_F}{2}\right) \left(\frac{\phi_F - \phi_C}{h}\right) + f_C h \right) \tag{3.29}$$

Para la discretización temporal se aplicó el método theta (θ) (2.17) a la ecuación (3.29):

$$\begin{aligned}\left(\frac{\phi^{n+1} - \phi^n}{\Delta t}\right)_C h^3 = h^2(1-\theta) &\left(\sum_{F\sim NB(C)} \left(\frac{a_C + a_F}{2}\right) \left(\frac{\phi_F - \phi_C}{h}\right) + f_C h \right)^n \\ &+ h^2\theta \left(\sum_{F\sim NB(C)} \left(\frac{a_C + a_F}{2}\right) \left(\frac{\phi_F - \phi_C}{h}\right) + f_C h \right)^{n+1}\end{aligned} \tag{3.30}$$

donde $\theta \in [0, 1]$, de forma que para $\theta = 0$ se obtiene el esquema FE y para $\theta = 1/2$ se obtiene el esquema implícito CN.

Esquema explícito - Diferencias hacia adelante

Reemplazando $\theta = 0$ en (3.30), re-ordenando y agrupando se obtiene el método explícito o FE:

$$\phi_C^{n+1} = \phi_C^n + \frac{\Delta t}{2h^2} \left(\sum_{F\sim NB(C)} (a_C^n + a_F^n)\,(\phi_F^n - \phi_C^n) + 2 f_C^n h^2 \right) \tag{3.31}$$

Aquí se destaca que si bien aparecen términos en la sumatoria no lineales en la variable ϕ en la ecuación (3.31) debido a que se consideran los casos en que $a_C^n = a(\phi_C^n)$, todos son evaluados en tiempo actual, resultando en una expresión explícita que se puede evaluar en tiempo futuro $n+1$ conociendo la solución actual, esto es lo que constituye la principal ventaja del método explícito.

Esquema implícito - Diferencias centradas

Para el esquema CN se reemplaza $\theta = 1/2$ en (3.30):

$$\begin{aligned}\frac{\phi_C^{n+1} - \phi_C^n}{\Delta t} h^3 = \frac{h^2}{2} &\left(\sum_{F\sim NB(C)} \left(\frac{a_C^n + a_F^n}{2}\right) \left(\frac{\phi_F^n - \phi_C^n}{h}\right) + f_C^n h \right) \\ &+ \frac{h^2}{2} \left(\sum_{F\sim NB(C)} \left(\frac{a_C^{n+1} + a_F^{n+1}}{2}\right) \left(\frac{\phi_F^{n+1} - \phi_C^{n+1}}{h}\right) + f_C^{n+1} h \right)\end{aligned} \tag{3.32}$$

El segundo término del lado derecho de la ecuación (3.32) indica que para calcular la solución en instante $n+1$ es necesario resolver un sistema lineal de ecuaciones algebraicas. Para el caso en

que a es independiente de la variable (ecuación de difusión lineal) el sistema que se debe resolver es lineal. Cuando a depende de la variable se debe resolver un sistema no lineal, así por ejemplo si se considera un coeficiente de difusión $a(\phi) = \kappa(\phi + 1)$ el sistema no lineal luce:

$$\begin{aligned}\frac{\phi_C^{n+1} - \phi_C^n}{\Delta t} h^3 = \frac{h^2\kappa}{2} \left(\sum_{F \sim NB(C)} \left(\frac{(\phi_C^n + 1) + (\phi_F^n + 1)}{2} \right) \left(\frac{\phi_F^n - \phi_C^n}{h} \right) + f_C^n h \right) \\ + \frac{h^2\kappa}{2} \left(\sum_{F \sim NB(C)} \left(\frac{(\phi_C^{n+1} + 1) + (\phi_F^{n+1} + 1)}{2} \right) \left(\frac{\phi_F^{n+1} - \phi_C^{n+1}}{h} \right) + f_C^{n+1} h \right)\end{aligned} \tag{3.33}$$

Para resolver el sistema no lineal que surge de considerar a la ecuación (3.32) para todo el dominio se utiliza el método de NR. Reescribiendo (3.32) en forma de residuo se obtiene:

$$\begin{aligned}R(\phi^{n+1}) = \phi_C^n - \phi_C^{n+1} + \frac{\Delta t}{2} \left(f_C^{n+1} + f_C^n \right) + \\ \frac{\Delta t}{4h^2} \sum_{F \sim NB(C)} \left[\left(A_C^n + A_F^n \right) \left(\phi_F^n - \phi_C^n \right) + \left(A_C^{n+1} + A_F^{n+1} \right) \left(\phi_F^{n+1} - \phi_C^{n+1} \right) \right] = 0\end{aligned} \tag{3.34}$$

El método de NR consiste en resolver iterativamente el sistema linealizado

$$R(\phi) \approx R(\phi^{(k)}) + \mathbf{J}(\phi^{(k)}) \left(\phi^{(k+1)} - \phi^{(k)} \right) = 0 \tag{3.35}$$

Definiendo $\Delta\phi^{(k+1)} = \phi^{(k+1)} - \phi^{(k)}$ la ecuación (3.35) puede expresarse como

$$R(\phi^{(k)}) + \mathbf{J}(\phi^{(k)}) \Delta\phi^{(k+1)} = 0 \tag{3.36}$$

o equivalentemente

$$\phi^{(k+1)} = \phi^{(k)} - \left(\mathbf{J}(\phi^{(k)}) \right)^{-1} R(\phi^{(k)}) \tag{3.37}$$

donde $\mathbf{J}(\phi^{(k)})$ es el Jacobiano del residuo R evaluado en $\phi^{(k)}$ que corresponde a la variable en tiempo futuro ϕ^{n+1} en la k-ésima iteración.

3.7.2. Algoritmos

Se implementaron en GPU dos algoritmos: uno explícito y otro considerando el esquema CN. La implementación del esquema explícito (ecuación (3.31)) es trivial ya que por cada paso de tiempo hay una única instrucción.

A falta de un análisis con el método de Von Neumann para la estabilidad global del método explícito, en esta ecuación no lineal, se optó por hacer una comprobación de la estabilidad restringiendo el paso de tiempo de acuerdo al número de Fo local, que para el problema 3D resuelto es $\Delta t < \frac{h^2}{a(\phi)\, 6}$. Como el coeficiente de difusión va cambiando al cambiar el tiempo, el paso de tiempo es adaptativo de forma que $Fo \approx 0{,}998$ durante toda la simulación, en particular, se actualiza Δt cada 2000 pasos de tiempo determinando $\phi_{\text{máx}}$ en el dominio lo que permite calcular el $\Delta t_{\text{máx}}$ de acuerdo al estado de la solución.

Para ilustrar el algoritmo para el método explícito se usa la siguiente notación: ϕ^*_{input} y ϕ^*_{output} indican los punteros a los únicos dos vectores almacenados en la memoria global del dispositivo. El algoritmo implementado para el caso explícito se presenta en 10.

Para el algoritmo implícito se requiere resolver el sistema lineal de ecuaciones, la matriz es SPD (simétrica definida positiva) así que se utilizó el resolvedor CG, con estilo matrix free. En este caso, los cálculos de una celda consisten en el producto de una fila de la matriz $\mathbf{J}(\phi^{(k)})$ por el vector auxiliar del resolvedor. Con ello se evita el almacenamiento de la matriz $\mathbf{J}$ ya que se ensambla en el momento de usarla. Respecto al cómputo del residuo $R(\phi^{(k)})$, se realiza con

Algoritmo 10: Algoritmo explícito para el problema de difusión no lineal 3D en GPU.

inicialización $\phi^*_{\text{input}} \leftarrow \phi^n$ apunta al vector que contiene el estado inicial ϕ^n;
$\phi^*_{\text{output}} \leftarrow \phi^{n+1}$ apunta al vector donde se guardará la nueva solución ;
mientras $t < T_{final}$ **hacer**
    $t \leftarrow t + \Delta t$;
    `compute_kernel_Interior(`$\phi^*_{\text{output}}, \phi^*_{\text{input}}$`);`
    `compute_kernel_Vertex1(`$\phi^*_{\text{output}}, \phi^*_{\text{input}}$`);`
    `...;`
    `compute_kernel_Edge1(`$\phi^*_{\text{output}}, \phi^*_{\text{input}}$`);`
    `compute_kernel_Side1(`$\phi^*_{\text{output}}, \phi^*_{\text{input}}$`);`
    sincronización: `cudaDeviceSynchronize();`
    intercambio de punteros: $\phi^*_{\text{output}} \longleftrightarrow \phi^*_{\text{input}}$;
fin

Algoritmo 11: Algoritmo implícito para el problema de difusión no lineal 3D en GPU.

inicialización $\phi^*_{\text{input}} \leftarrow \phi^n$ apunta al vector que contiene el estado inicial ϕ^n;
$\phi^*_{\text{output}} \leftarrow \phi^{n+1}$ apunta al vector donde se guardará la nueva solución ;
$k \leftarrow 0, t \leftarrow 0$;
mientras $t < T_{final}$ **hacer**
    $t \leftarrow t + \Delta t$;
    cálculo del vector residuo inicial $R(\phi^{(0)})$ con kernels concurrentes;
    sincronización: `cudaDeviceSynchronize();`
    cálculo del la norma del residuo inicial $r^{(0)}$ (usando rutinas basadas en CUBLAS);
    $k \leftarrow 0$;
    mientras $r^{(k)} > \text{tol}_{rel}\ r^{(0)} + \text{tol}_{abs}$ **hacer**
        seteo del operador lineal para el producto $\mathbf{Ax}$ (función de $\phi^{(k)}$);
        solución del sistema usando el resolvedor CG;
        $\phi^{(k+1)} \leftarrow \phi^{(k)} + \Delta\phi^{(k)}$ operación axpy con kernels concurrentes;
        sincronización: `cudaDeviceSynchronize();`
        cálculo del vector residuo $R(\phi^{(k)})$ con kernels concurrentes;
        sincronización: `cudaDeviceSynchronize();`
        cálculo del la norma del residuo $r^{(k)}$ (usando rutinas basadas en CUBLAS);
    fin
    $\phi^{n+1} \leftarrow \phi^{(k+1)}$;
    intercambio de punteros: $\phi^*_{\text{output}} \longleftrightarrow \phi^*_{\text{input}}$;
fin

la misma estrategia de paralelización que el caso explícito, mediante procedimientos del tipo: kernel$(R^*, \phi^{n*}, \phi^{(k)*})$. El algoritmo 11 presenta la estrategia implementada en GPU.

En el método implícito se evaluaron variantes de los algoritmos encontrándose que en los casos en que el sistema es muy no lineal, resolver una sola iteración NR implica la solución del sistema lineal con CG hasta una precisión aceptable y esto requiere de una cantidad considerable de iteraciones CG, mientras que si el sistema se resuelve de manera aproximada en CG, esto es, por ejemplo fijando la cantidad máxima de iteraciones CG y actualizando el sistema lineal a resolver (iteración NR) se obtienen ganancias de tiempo de cómputo en un factor de 10 para un mismo error final en la solución obtenida al salir del lazo de NR. Por el contrario, en la medida que el problema se vuelve mas lineal, es más conveniente hacer una sola iteración de NR y resolver el sistema con CG hasta una precisión aceptable.

Los mejores desempeños para el algoritmo implícito se encontraron variando dinámicamente el número de iteraciones CG, el criterio que se utilizó para esto fue el de incrementar en un porcentaje dado el número $IT_{\text{máx},CG}$ (iteraciones máximas del CG) si la tasa de convergencia del residuo del método de NR baja en menos de un orden de magnitud ($\log(r^{(k)}/r^{(k-1)}) \sim 1$) y reducir $IT_{\text{máx},CG}$ si la tasa de convergencia de NR es muy alta ($\log(r^{(k)}/r^{(k-1)}) > 3/2$). De esta manera se evita realizar iteraciones CG sin antes actualizar el sistema a resolver, es decir pasar a la siguiente iteración de NR en los casos que la convergencia NR avanza rápido, y resolver mejor el sistema (aumentando $IT_{\text{máx},CG}$) si la tasa de convergencia de NR es baja.

3.7.3. Validación: estudio de convergencia numérica

Los algoritmos implementados se ejecutaron en el equipo 1. Para comparar los esquemas de discretización temporal y espacial, se eligió un tiempo final de simulación igual a 1.84s, considerando que la condición inicial alcanza un máximo igual a $\phi^0_{\text{máx}} = \phi(1/2, 1/2, 1/2, 0) = 2{,}5$ resultando el coeficiente de difusión igual a $a(\phi^0_{\text{máx}}) = 3{,}5$, mientras que para $t = 1{,}84$s, $\phi^{1,84}_{\text{máx}} = 0{,}397$ y el coeficiente de difusión vale $a(\phi) = 1{,}397$. Observando que $a(\phi) = \kappa(\phi + 1) \to 1$ al avanzar en el tiempo, por lo tanto el problema va perdiendo la no linealidad conforme avanza la simulación. En el intervalo de simulación seleccionado el problema mantiene un componente no lineal importante.

Se realizó un estudio de convergencia en malla para la discretización espacial, tanto para el esquema explícito como para el implícito y un estudio para la convergencia respecto a la discretización temporal, en este caso se realizó el análisis solamente para el esquema implícito, debido a que el paso de tiempo en el esquema explícito viene condicionado por el número de Fourier local $Fo = 6a(\phi)\Delta t/h^2$ y para valores de Δt menores, el error debido a la discretización temporal es menor al que brinda el esquema espacial usado, luego la solución numérica obtenida resulta prácticamente igual para cualquier Δt menor al impuesto por el número Fo. Para las simulaciones se consideraron diversas discretizaciones espaciales teniendo en cuenta la arquitectura SIMT, para esto se eligen grillas de manera que el número de celdas internas N_{int} por dirección (x, y, z) sea múltiplo de 32 (a excepción de las dos grillas mas pequeñas), como los casos de borde se tratan en kernel separados el número de celdas total por dirección resulta en $(1 + N_{int} + 1)$. Los tamaños de grillas analizadas en este trabajo variaron desde 10x10x10 a 898x898x898 (724.150.792 celdas). Cabe aclarar que para el caso implícito el tamaño máximo que se pudo correr fue de 514x514x514 (135.796.744 celdas) debido a la limitación de tamaño en la memoria del dispositivo. En la tabla 3.7 se muestran las dimensiones de los casos simulados. Por último mencionamos que los resultados en las grillas que exceden las 514 celdas por dirección del caso explícito se utilizaron para evaluar el desempeño (sección 8) y que para el estudio de convergencia en espacial y temporal solo se muestran las comparaciones hasta ese tamaño de grilla.

En la figura 3.20 se muestra la solución numérica obtenida para diferentes instantes de tiempo.

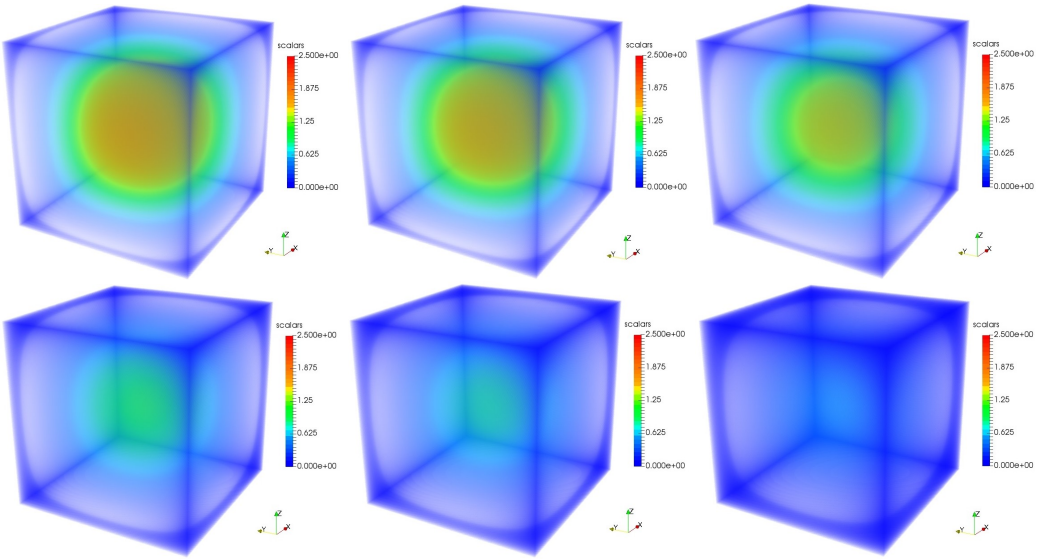

Figura 3.20: Solución numérica para diferentes instantes de tiempo distribuidos uniformemente desde $t = 0s$ a $t =$ 1,84s.

$N_xN_yN_z$	# cell	$N_xN_yN_z$	# cell
10x10x10	1.000	322x322x322	33.386.248
18x18x18	5.832	418x418x418	73.034.632
34x34x34	39.304	450x450x450	91.125.000
66x66x66	287.496	514x514x514	135.796.744
98x98x98	941.192	610x610x610*	226.981.000*
130x130x130	2.197.000	706x706x706*	351.895.816*
194x194x194	7.301.384	770x770x770*	456.533.000*
258x258x258	17.173.512	898x898x898*	724.150.792*

Tabla 3.7: Dimensiones de los casos de prueba utilizados ((*) Solamente en el esquema explícito).

Convergencia del esquema temporal

Se evaluó la tasa de convergencia para el esquema temporal de CN en las grillas de 66^3, 130^3, 258^3 y 514^3, el tiempo de simulación para realizar la comparación con la solución analítica fue de 1,84s, el error (ϵ) se evaluó usando el RMS (Root Mean Square) entre la solución numérica y la analítica

$$RMS = \sqrt{\left(\sum_{i=0}^{N_xN_yN_z} \left(\phi_{\text{i,exac}} - \phi_{\text{i,num}}\right)^2\right) / N_xN_yN_z} \tag{3.38}$$

En la figura 3.21 se muestran los resultados obtenidos para las 4 grillas, en todos los casos el error se reduce hasta estabilizarse en un valor que viene dado por la resolución espacial de la grilla, puede apreciarse también que el esquema temporal es efectivamente de $O(\Delta t^2)$, por lo tanto se tiene una tasa de convergencia que es cuadrática al variar el paso de tiempo. En concreto puede apreciarse $\epsilon_1/\epsilon_2 \sim (\Delta t_1/\Delta t_2)^2$ y que la pendiente en escala logarítmica es aproximadamente de $\frac{(\log(\epsilon_1)-\log(\epsilon_2))}{(\log(\Delta t_1)-\log(\Delta t_2))} \sim 2$, y que por lo tanto, al reducir el paso de tiempo a la mitad el error se reduce en un factor de 4.

Convergencia del esquema espacial

Para evaluar la convergencia en malla hay que tener en cuenta algunos factores. El primero a considerar es que como la estabilidad condicional del esquema explícito está sujeta al número de

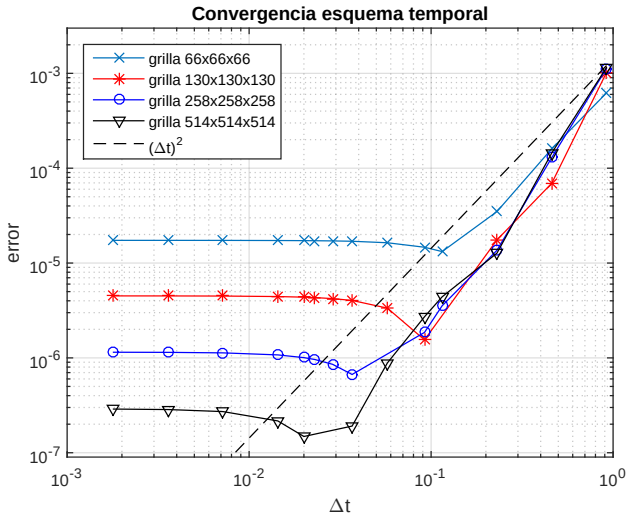

Figura 3.21: Estudio de la convergencia del esquema temporal para 4 dimensiones de grilla diferentes.

Fourier, el paso de tiempo debe acompañar el tamaño del paso de malla bajo la relación $\Delta t \sim h^2$, de esta manera en general se tiene que el error del esquema temporal no afecta el orden de la solución. El segundo factor a considerar es que el esquema implícito resultó ser incondicionalmente estable y por lo tanto un paso de malla dado h_1 se pudieron correr casos para diferentes pasos de tiempo. Entonces para realizar una comparación entre ambos esquemas se eligió el paso de tiempo de manera que el error temporal esté por debajo del error espacial. Esto último permite evaluar diferentes grillas que pueden dar errores mayores al de la discretización temporal. El segundo factor que se menciona puede verse en la figura 3.22 (izquierda) la cual muestra dos corridas del caso implícito usando $\Delta t = 0{,}115$s y $\Delta t = 0{,}0575$s, de manera que para mallas cada vez mas finas, el error se estabiliza en un valor dado por el orden de precisión del método CN como es de esperar. En la figura 3.22 (derecha) se muestra la tasa de convergencia para el método explícito e implícito juntos, puede verse que ambos tienen una tasa de convergencia cuadrática al refinar la malla.

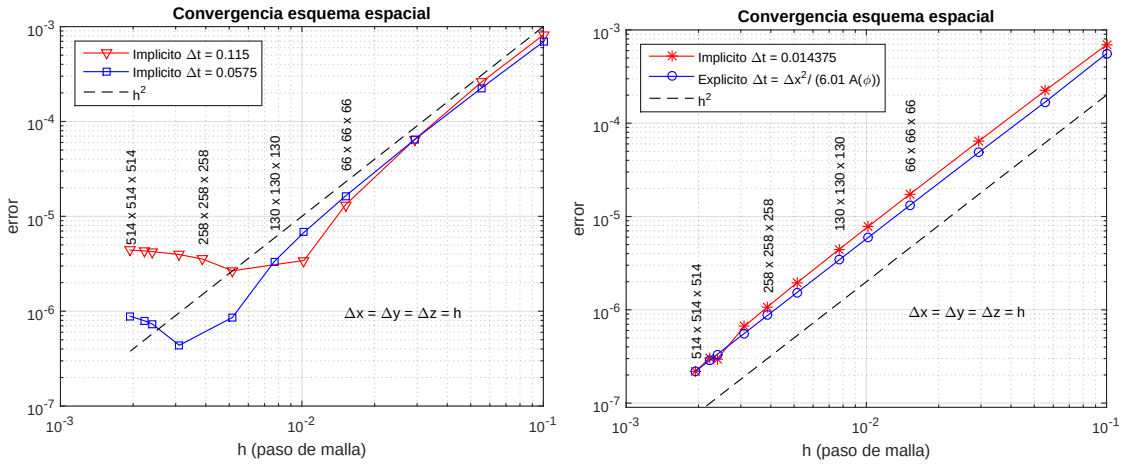

Figura 3.22: Convergencia del esquema espacial del método implícito para dos pasos de tiempo diferentes (izquierda) y comparación con el método explícito (derecha).

3.7.4. Evaluación del desempeño

Para medir la eficiencia del algoritmo explícito se evalúa la velocidad a la que se procesan las celdas de la siguiente forma:

$$\text{rate} = \frac{N_{steps} \times N_x \times N_y \times N_z}{t_{total} \times 10^6} \left[\frac{Mcells}{sec}\right] \tag{3.39}$$

siendo N_{steps} el número de pasos de tiempo realizados, t_{total} el tiempo total de la simulación en segundos. Para el caso implícito se puede estimar una *rate* equivalente considerando que para cada paso de tiempo se tienen que realizar varias iteraciones en el resolvedor CG y calcular el residuo o lado derecho para el método de NR al menos una vez por cada iteración, de manera que la eficiencia viene dada por:

$$\text{rate} = \frac{(N_{it,CG} + N_{it,NR}) \times N_x \times N_y \times N_z}{t_{total} \times 10^6} \left[\frac{Mcells}{sec}\right] \tag{3.40}$$

siendo $N_{it,CG}$ el número total de iteraciones CG realizadas, $N_{it,NR}$ el acumulado de iteraciones en el método de NR. Esto se eligió así debido a que la cantidad de operaciones calculadas en los kernels, tanto para el producto Ax de CG como para el cálculo del residuo $R^{(k)}$ de NR son similares a las del método explícito en cada paso de tiempo, las únicas dos diferencias entre los esquemas es que en el método explícito por cada paso de tiempo y por celda se requieren dos lecturas/escrituras: (ϕ_{input}, ϕ_{output}), mientras que en el implícito se requieren tres lecturas/escrituras en la memoria global: ($\Delta\phi^{(k)}_{\text{input}}$, $\Delta\phi^{(k)}_{\text{output}}$, $\phi^{(k)}$) y ($R^{(k)}$, ϕ^n, $\phi^{(k)}$) para CG y residuo de NR respectivamente, y por otro lado en el resolvedor CG se realizan algunas operaciones de reducción (normas, producto interno, etcétera) y del tipo axpy. Sin embargo se considera que las operaciones más costosas son las dos usadas para el cálculo del rate.

Además se evaluaron los tiempos de cálculo de cada método para diferentes resoluciones de grilla, y los tiempos de cálculo para obtener diferentes errores al comparar con la analítica.

Evaluación de la tasa de procesamiento

En la tabla 3.8 se muestra un resumen de los resultados obtenidos para las tasas de procesamiento, mas valores se muestran en la figura 3.23. Puede apreciarse que el algoritmo explícito explota la potencia de cálculo de la GPU hasta que se produce una saturación en la tasa para dominios computacionales mayores a los 33 millones de celdas hasta los 135 millones donde se ve una caída del desempeño para dominios mayores manteniéndose un valor residual de 2880 Mcell/sec. Puede verse que el algoritmo implícito no aprovecha al máximo el hardware y la relación entre las tasas de procesamiento en cada método es de 1 a 3 en todas las grillas analizadas.

$N_x = N_y = N_z$	66	130	258	322	418	514	706*	898*
[Mcell]	0,33	2,2	17,2	33,4	73,0	135,8	351,9*	724,1*
rate Explícito [Mcell/sec]	521	1444	2608	2931	2940	2983	2885	2834
rate Implícito [Mcell/sec]	138	516	837	888	899	856	-	-
r_{exp}/r_{imp}	3,77	2,79	3,11	3,30	3,27	3,48	-	-

Tabla 3.8: Tasas de procesamiento obtenidas para los dos métodos explícito e implícito ((*) Solamente en el esquema explícito).

Errores y tiempos de calculo

Se evaluó el tiempo computacional o de cálculo requerido por celda por cada segundo de simulación, un tiempo de cómputo mayor por segundo de simulación significa que el algoritmo es más rápido. El costo computacional de la simulación es proporcional al número de celdas procesadas

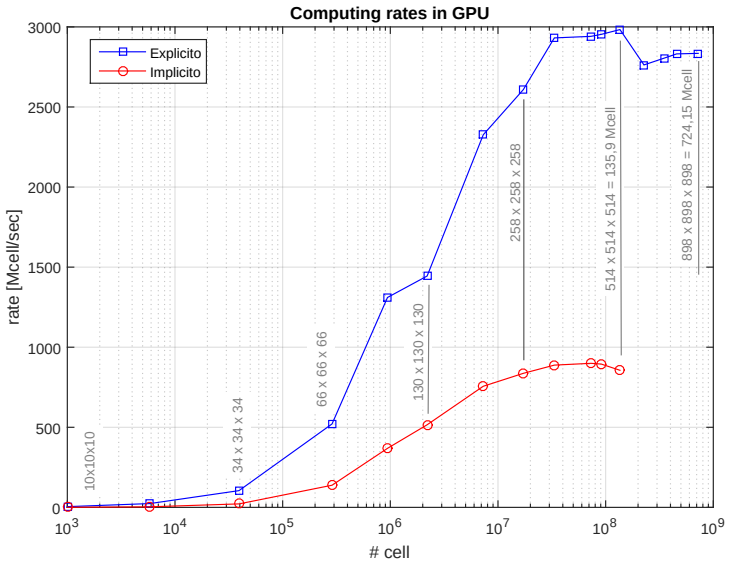

Figura 3.23: Eficiencia de los dos algoritmos en base a la tasa de procesamiento.

$N = N_x N_y N_z$ y a la cantidad de pasos de tiempo requeridos en cada segundo de simulación, es decir $\text{costo}_{comp} = O(steps/sec \times N) = O(N/\Delta t)$, considerando que en el método explícito el paso de tiempo está restringido por el número de Fourier, deducimos que:

$$\Delta t = O(h^2) = O\left((1/N_x)^2\right) = O\left(\left(1/N^{1/3}\right)^2\right) = O(N^{-2/3}) \tag{3.41}$$

esto es:

$$\text{costo}_{comp} = O(N/N^{-2/3}) = O(N^{5/3}) \tag{3.42}$$

entonces se espera que el tiempo computacional se incremente a la potencia 5/3 del número total de celdas procesadas para un mismo tiempo de simulación. En la figura 3.24 (izquierda) se muestra que la tendencia final en el método explícito es la esperada mientras que para el implícito la pendiente es un poco menor y próxima a 4/3, esto se explica debido a que el paso de tiempo se mantuvo constante en dichas pruebas, en particular, se fijó $\Delta t = 0{,}014375$, o en otras palabras se fijó la cantidad de pasos de tiempo en cada simulación. Por otro lado en el método implícito el costo computacional por cada paso de tiempo es dado por la cantidad de iteraciones del resolvedor CG siendo esto último proporcional a la cantidad de celdas por dirección ($\sqrt[3]{N}$) como puede verse en la figura 3.24 (derecha). Entonces se espera un costo computacional para el método implícito igual a:

$$\text{costo}_{comp} = \underbrace{1/\Delta t}_{const} \times N_{it,CG,total} \times N = O(N^{1/3} \times N) = O(N^{4/3}) \tag{3.43}$$

Respecto a los errores, en la figura 3.25 (izquierda) se muestra que decrecen a una tasa del orden $O(N^{-2/3})$ en ambos casos (explícito e implícito) esto es equivalente a lo mostrado en el análisis de convergencia de la discretización espacial debido a que $h = N^{-1/3}$.

Finalmente en la figura 3.25 (derecha) se comparan los errores obtenidos en los dos esquemas para sus respectivos tiempos de cálculo, se observa que para problemas pequeños (menores a 0.3 Mcell $\approx 66^3$) es mas eficiente el método explícito y para problemas mas grandes la relación se invierte, siendo mas eficiente el método implícito, esta diferencia crece al aumentar el número de celdas llegando una relación de 1/10 en los tiempos de cálculo requeridos por el método implícito y explícito respectivamente. Esto se debe a que el $\Delta t_{\text{máx}}$ en el esquema explícito está fuertemente condicionado con el número de Fourier y el implícito permite pasos de tiempo mayores (en este ejemplo $\Delta t_{\text{implicito}} = 0{,}014375$), además de esto, en el método implícito se pueden variar los

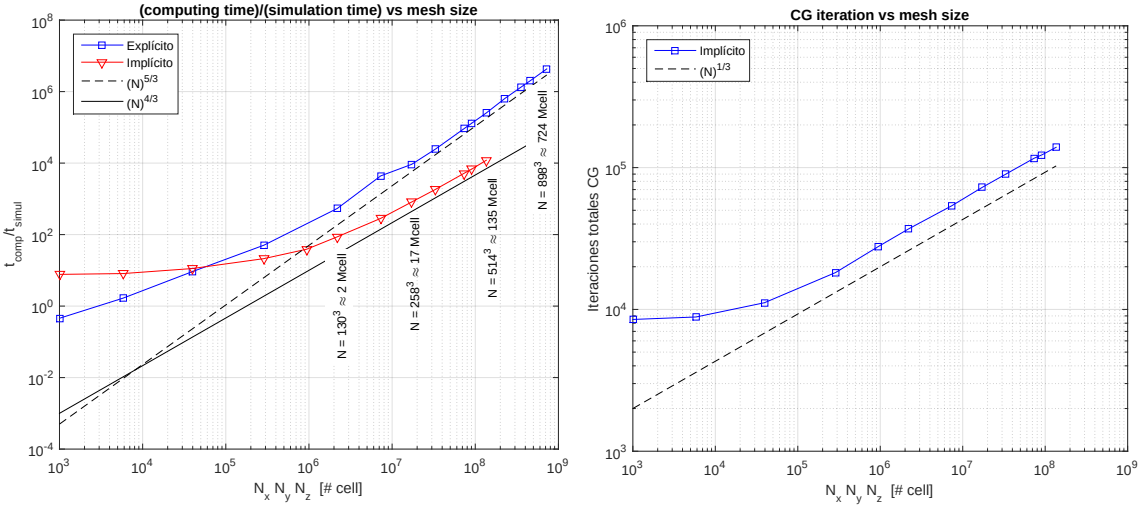

Figura 3.24: Tiempos de cálculo por segundo de simulación versus el tamaño de la grilla (izquierda) y número de iteraciones de CG acumuladas en la simulación (derecha)

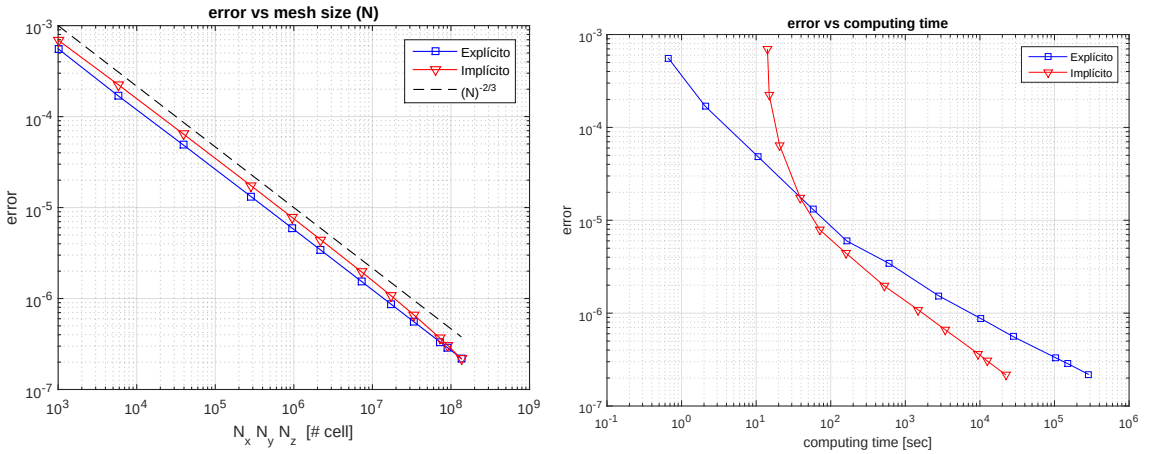

Figura 3.25: Error obtenido para $t = 1{,}84$s versus el tamaño de la grilla (izquierda) y comparación del error obtenido para diferentes tiempos de cálculo en el método explícito e implícito (derecha).

criterios de convergencia del resolvedor CG y el lazo de NR permitiendo un fuerte incremento en la performance del algoritmo.

3.7.5. Conclusiones

Los resultados muestran que ambos algoritmos (explícito e implícito) explotan al máximo las capacidades de la GPU, logrando simular problemas con muy alta resolución en el caso explícito (hasta 724 millones de celdas) sin embargo en el caso analizado se tiene una fuerte restricción en la estabilidad numérica del método requiriendo pasos de tiempo relativamente pequeños.

Respecto al método implícito se comprobó que a pesar de tener valores de rates mas bajos respecto al algoritmo explícito, variando los criterios de convergencia en el resolvedor CG según la evolución del residuo en el método de NR, es posible obtener un buen desempeño global. En particular, se gana mucha eficiencia fijando las iteraciones máximas del resolvedor CG en cada iteración de NR en lugar de forzar la solución de CG hasta cierta precisión. En relación a esto, se probó que el orden de precisión temporal en el método de NR no cambia al realizar más iteraciones, pero como contrapartida se vió que para problemas cada vez mas no lineales, resolver una única vez el sistema de ecuaciones con el resolvedor CG (una sola iteración de NR) requiere un número considerablemente mayor de iteraciones CG. Por lo tanto se obtuvieron mucho mejores

resultados resolviendo de forma inexacta el sistema en el resolvedor (fijando el número de iteraciones CG dinámicamente) y actualizando mas veces el sistema lineal a resolver (iteraciones NR) hasta obtener una precisión deseada al salir del lazo NR. Esta misma característica se encuentra en los algoritmos segregados por ejemplo el método SIMPLE (Semi Implicit Method for Pressure Linked Equation) usados para resolver las ecuaciones de Navier Stokes en los cuales no se resuelve el sistema de forma precisa en cada iteración interna del algoritmo sino que solo se fuerza la conservación de masa al final de cada paso de tiempo [61].

Cabe mencionar que los algoritmos implementados alcanzan el límite del ancho de banda de la tarjeta antes de alcanzar el límite en la capacidad de procesamiento de la GPU. Esto se debe a que las operaciones realizadas por cada CUDA Threads tiene una intensidad aritmética muy baja y se consume más tiempo en la lectura-escritura de datos requeridos para los cálculos. Lo anterior explica el porqué de los resultados obtenidos en términos del rate explícito versus implícito, ya que en el último se requiren tres veces mas operaciones de lectura-escritura en memoria global. Finalmente se destaca que la combinación de criterios y métodos en el algoritmo implícito aceleran la convergencia global del problema, requiriendo menos operaciones totales para llegar al mismo resultado que el algoritmo explícito.

3.8. Comparación de algoritmos SIMPLE y FS

En esta prueba se resuelven las ecuaciones para flujos incompresibles en estado estacionario y transiente, en dominios 2D y 3D, implementando dos métodos en GPU, FSM y SIMPLE. Se realiza además una evaluación utilizando diferentes métricas para medir el desempeño, discutiendo tasas de procesamiento para diferentes tamaños de malla. El caso de estudio es el test del flujo en una cavidad cuadrada (cúbica en 3D) con tapa deslizante.

3.8.1. Discretización espacial y temporal

Para esta prueba, se resolvieron numéricamente el conjunto de ecuaciones de NS aplicando los algoritmos 2 y 3 desarrollados en el capítulo 2 que implementan el método SIMPLE y FSM respectivamente, como esquema de discretización espacial, se utilizo CN para SIMPLE y la combinación CN y AB (difusión y advección respectivamente) en el FSM. Para la discretización espacial se utilizaron el método CD en las pruebas 3D mientras que para las pruebas 2D se utilizaron los esquemas CD y QUICK de acuerdo al número de Reynolds a simular.

A continuación se brindan algunos detalles sobre la implementación de cada algoritmo en GPU.

Implementación del algoritmo SIMPLE en GPU

Para implementar el algoritmo SIMPLE se elaboraron las siguientes funciones:

- `aC_RHS_residuo();`
- `stencil_op_momentum();`
- `rhie_chow_interp();`
- `RHS_pressure();`
- `stencil_op_PPE();`
- `update_mf();`

Todas las funciones se componen de 9 kernels (caso 2D) o 27 kernels (caso 3D) con ejecución concurrente donde el kernel que procesa el interior del dominio en el caso 3D se paraleliza recorriendo por planos horizontales como se explicó secciones atrás. A continuación se describen las variables para el caso 3D.

Todas las variables se almacenan en la memoria de la GPU, las variables almacenadas propias del método son 22 y corresponden a los siguientes vectores: aC_{uvw} (1 vector con los coeficientes centrales iguales para las 3 ecuaciones de momento, los aportes de las condiciones de borde se incorporan posteriormente en cada caso por separado); u_n, v_n, w_n, pn (4 vectores para las variables a tiempo actual); u_*, v_*, w_* (3 vectores para las variables de iteración dentro del lazo SIMPLE); u_0, v_0, w_0, pp (4 vectores para los campos de velocidad a tiempo pasado y el campo de corrección de la presión); m_e, m_n, m_t (3 vectores con los flujos en caras); $rhs_x, rhs_y, rhs_z, rhs_p$ (4 vectores con los lados derechos de cada ecuación de momento y ecuación para la corrección de la presión); res_u, res_v, res_w (3 vectores para monitorear el residuo de las ecuaciones de momento). Además de los 22 vectores, se requieren 8 vectores auxiliares utilizados en los solvers BiCGStab y CG.

La función `aC_RHS_residuo();` calcula el vector de coeficientes centrales aC_{uvw} que se utiliza para ensamblar las ecuaciones de momento, para la interpolación de Rhie Chow y para la ecuacion de corrección de la presión, calcula el RHS de las ecuaciones de momento y el residuo de las mismas para chequear el proceso de convergencia.

La función `stencil_op_momentum();` realiza la operación tipo stencil para llamada por el solver lineal de las ecuaciones de momento. Utilizando un esquema centrado o upwind según si se utilizan esquemas TVD vía técnica DC o no.

La función `rhie_chow_interp();` realiza la interpolación de Rhie-Chow para los flujos m_e, m_n, m_t en caras que se utilizaran como RHS en la ecuación de la presión (PPE).

La función `RHS_pressure();` calcula el desbalance en los flujos másicos y determina el vector RHS de la PPE.

La función `stencil_op_PPE();` realiza la operación tipo stencil que requiere el solver para la presión utilizando el vector aC_{uvw}.

La función `update_mf();` corrige los campos de velocidad, de flujos en caras y el campo de presiones.

Cabe destacar que en estas funciones se hace un rehúso intensivo de los datos leídos en memotia global de la GPU es por ello, que por ejemplo la función `aC_RHS_residuo();` realiza las operaciones requeridas para las 3 ecuaciones de momento ya que coinciden muchas cosas al momento de ensamblar los sistemas (principalmente lado derecho y cómputo del residuo para chequear la convergencia). Para los vectores que más se van a utilizar se los carga en memoria compartida *shared memory* para liberar la memoria de registros de la cual también se hace uso intensivo para permitir el rehúso de variables auxiliares durante el cálculo del kernel. Finalmente, el algoritmo 12 presenta la implementación basada en GPU que se realizó.

Implementación del algoritmo FSM en GPU

La implementación del FSM es relativamente más sencilla que en SIMPLE. En este caso se implementaron las siguientes funciones:

- `RHS_momentum();`
- `stencil_op_momentum();`
- `update_Ufaces();`
- `RHS_pressure();`
- `stencil_op_PPE();`
- `update_p_u_U();`

Algoritmo 12: Implementación GPU del algoritmo SIMPLE en 3D.

```
1  inicialización de los campos;
2  t ← 0;
3  mientras t < T_final hacer
4      conv ← 0, iter ← 0, tol ← 1,0;
5      mientras conv == 0 hacer
6          iter ← iter + 1;
7          coeficientes para ecs. momento y RHS: aC_RHS_residuo();
8          calculo de la norma de residuos para chequeo de convergencia:
             residuo_x ← ||res_x||, residuo_y ← ||res_y||, residuo_z ← ||res_z||;
9          llamada al solver: BiCGStab(u_*, rhs_u);
10         llamada al solver: BiCGStab(v_*, rhs_u);
11         llamada al solver: BiCGStab(w_*, rhs_w);
12         actualización de flujos másicos vía interpolación de Rhie-Chow:
             rhie_chow_interp(aC_uvw,m_e,m_n,m_t,pn);
13         computar RHS para PPE: RHS_pressure(m_e,m_n,m_t,rhs_p);
14         calculo de la norma del res. p. chequeo de conv.: residuo_p ← ||rhs_p||;
15         llamada al solver: CG(pp, rhs_p);
16         corrección de los campos de velocidad, flujos en caras y campo de presión:
             update_mf();
17         chequeo de convergencia: tol ← máx(residuo_x, residuo_y, residuo_z, residuo_p);
18         si tol < 1e − 8 entonces
19             conv ← 1;
20         fin
21         si iter < iter_máx entonces
22             conv ← 1;
23         fin
24     fin
25     intercambio de punteros para los campos de velocidad;
26     t ← t + Δt;
27 fin
```

Las variables almacenadas en GPU propias del método son 18: y corresponden a los siguientes vectores: pn, pp (2 vectores, uno para el campo de presiones y el campo de correción de la presión); u_n, v_n, w_n (3 vectores para las velocidades a tiempo actual); u_o, v_o, w_o (3 vectores para los campos de velocidad a tiempo pasado); U_o, V_o, W_o (3 vectores con los flujos en caras a tiempo actual); U_{oo}, V_{oo}, W_{oo} (3 vectores con los flujos en caras a tiempo pasado); $rhs_x, rhs_y, rhs_z, rhs_p$ (4 vectores con los lados derechos de cada ecuación de momento y ecuación para la corrección de la presión). Además de los 18 vectores, se requieren 4 vectores auxiliares utilizados en el solver CG. En este caso las operaciones que realiza cada función no necesitan describirse. El algoritmo 13 presenta la implementación diseñada para GPU de este método.

Algoritmo 13: Implementación GPU del algoritmo FSM en 3D.

1 inicialización de los campos;
2 $t \leftarrow 0$;
3 **mientras** $t < T_{final}$ **hacer**
4 calculo de los RHS de las ecuaciones de momento: `RHS_momentum()`;
5 intercambio de punteros de velocidades y flujos a tiempo pasado para las tres direcciones: $u_o \leftrightarrow u_{oo}, U_o \leftrightarrow U_{oo}, \ldots$;
6 llamada al solver: `CG(`u_o, rhs$_u$`)`;
7 llamada al solver: `CG(`v_o, rhs$_u$`)`;
8 llamada al solver: `CG(`w_o, rhs$_w$`)`;
9 actualización de flujos másicos vía interpolación de Rhie-Chow: `update_Ufaces(`u_o,v_o,w_o,U_o,V_o,W_o,pn`)`;
10 computar RHS para PPE: `RHS_pressure(`U_o,V_o,W_o,rhs_p`)`;
11 llamada al solver: `CG(`pp, rhs$_p$`)`;
12 corrección de los campos de velocidad, flujos en caras y campo de presiones: `update_p_u_U(`$u_o, v_o, w_o, U_o, V_o, W_o, pn$`)`;
13 $t \leftarrow t + \Delta t$;
14 **fin**

3.8.2. Validación: Caso 2D - cavidad cuadrada con tapa deslizante

En la figura 3.26 se presenta el esquema para el problema 2D y las respectivas condiciones de borde aplicadas.

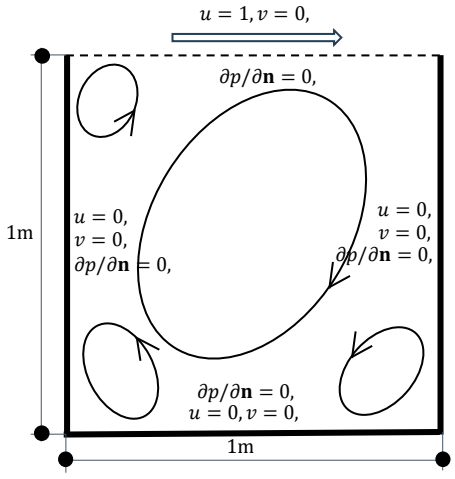

Figura 3.26: Dominio y condiciones de borde para el problema 2D de la cavidad cuadrada con tapa deslizante.

Caso estacionario

El problema se resolvió para diferentes valores del número de Reynolds a saber: $Re = 100, 1000, 3200, 5000, 7$ Los 3 primeros casos se resolvieron utilizando el FSM con el esquema CD hasta alcanzar el estado estacionario, mientras que para los últimos 3 casos se utilizó SIMPLE estacionario porque al tratarse de problemas a altos Re el estado estacionario demora más tiempo en alcanzarse con un método transiente, además de ello, se utilizó el esquema QUICK que es mas preciso para flujos a mayor Re. En todos los casos se utilizó una grilla de 128×128.

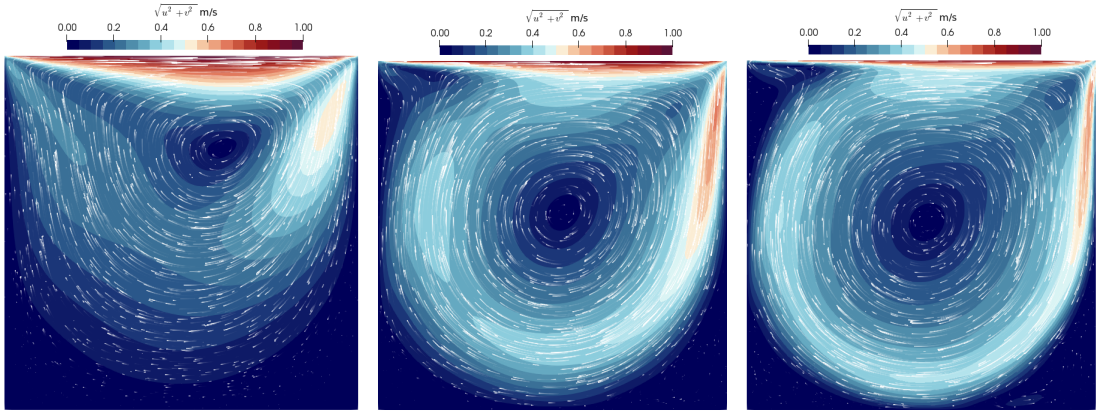

Figura 3.27: Solución numérica para $Re = 100, 1000, 3200$ utilizando el algoritmo FSM y un esquema espacial CD para el término advectivo y difusivo.

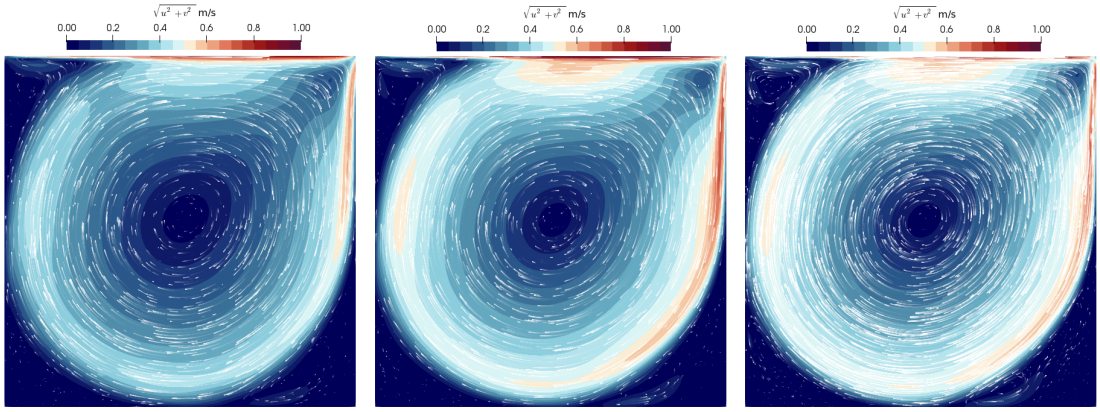

Figura 3.28: Solución numérica para $Re = 5000, 7500, 10000$ utilizando el algoritmo SIMPLE estacionario y un esquema espacial QUICK para el término advectivo y CD para el término difusivo.

Para validar los resultados, se compararon los perfiles de velocidad en las líneas centrales del dominio con los resultados obtenidos por Guia et. al. [63]. Las figuras 3.29 y 3.30 presenta la comparación para diferentes valores de Re, puede verse que la solución provista por ambos algoritmos es realmente aceptable considerando que se utilizó una grilla relativamente gruesa para la prueba.

A modo de referencia en cuanto a tiempos de cómputo, la solución del problema estacionario utilizando el algoritmo SIMPLE en grillas de 128×128 y 256×256 requiere 34s y 84s respectivamente en el equipo 1 (GPU Nvidia Tesla K40), mientras que en el equipo 2 (GPU Nvidia Tesla V100) el tiempo de cómputo es de 16s y 28s respectivamente, en ambos casos, la tolerancia en los residuos del método fue 1e-6. Cabe mencionar que estos tamaños están lejos de aprovechar las capacidades de cómputo de la GPU debido al tamaño del problema.

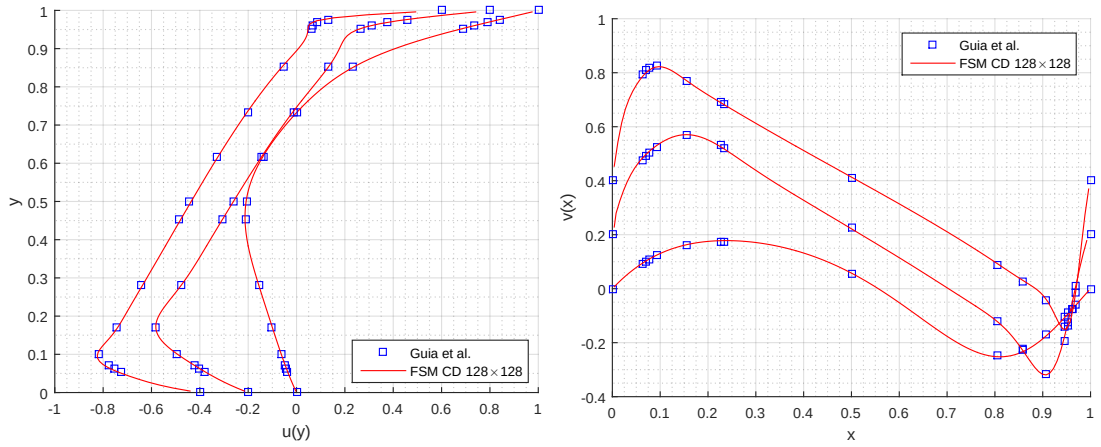

Figura 3.29: Perfiles de velocidad para las líneas centrales para la cavidad en 2D, $Re = 100, 1000, 3200$ utilizando el algoritmo FSM y el esquema espacial CD para el término advectivo y difusivo, comparación con los resultados de Guia et. al. [63].

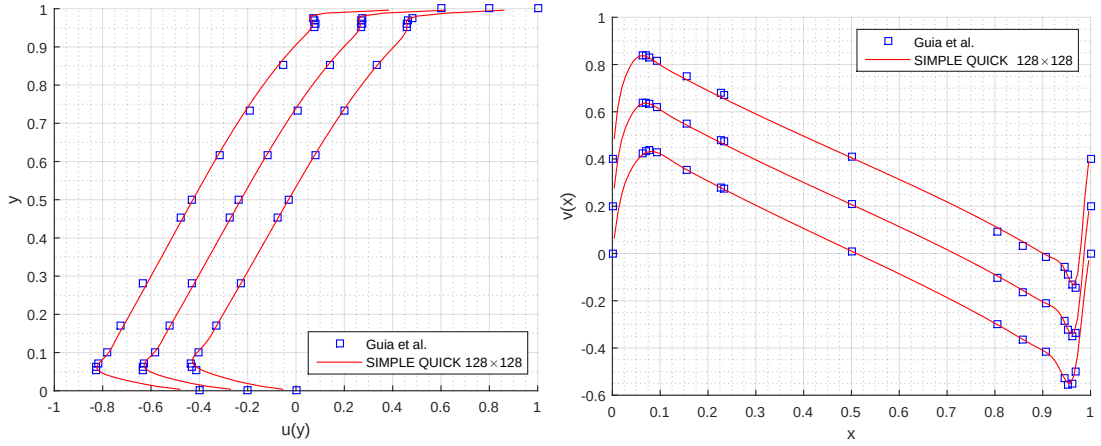

Figura 3.30: Perfiles de velocidad para las líneas centrales para la cavidad en 2D, $Re = 5000, 7500, 10000$ utilizando el algoritmo SIMPLE y un esquema espacial QUICK para el término advectivo y CD para el término difusivo, comparación con los resultados de Guia et. al. [63].

3.8.3. Validación: Caso 3D - cavidad cúbica con tapa deslizante

En esta prueba se resuelve el problema de la cavidad en 3D a $Re = 1000$ utilizando los 2 algoritmos. En el primer caso se resuelve la ecuación hasta el tiempo $T_{\text{final}} = 40$s donde se asume que ha alcanzado el estado estacionario.

En la figura 3.31 se presenta la solución numérica obtenida sobre una grilla de $130 \times 130 \times 130$ para diferentes instantes de tiempo.

La figura 3.32 presenta las isosuperficies de vorticidad y las lineas de corriente una vez alcanzado el estado estacionario. En la figura 3.33 se comparan los perfiles de velocidad en las líneas centrales x y z con los resultados obtenidos por Ku et al. [89] utilizando métodos pseudoespectrales. Puede observarse que los resultados coinciden con los valores de referencia para el caso 3D. Si bien para los perfiles de la figura 3.33 se utilizó la solución provista por FSM, el resultado que se obtiene utilizando el algoritmo SIMPLE en estado estacionario es idéntica, por ello no se incorporó en la figura.

Desempeño en GPU de los algoritmos SIMPLE y FSM

En este segundo caso, se ejecutaron ambos algoritmos en el equipo 1 para diferentes tamaños de grilla, en la tabla 3.9 se muestran los números de celdas por dirección de cada grilla. La simu-

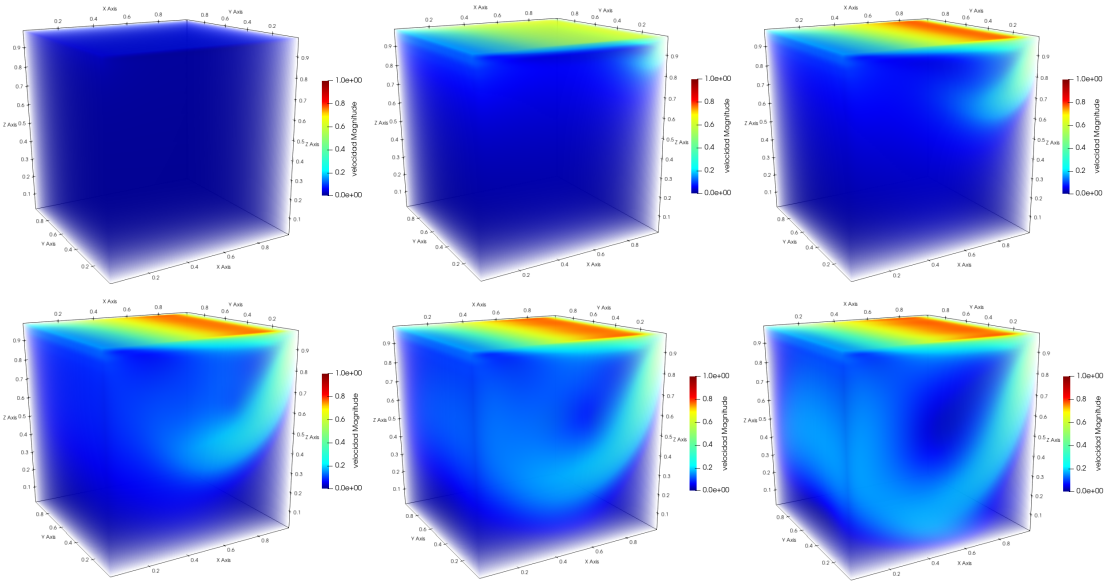

Figura 3.31: Solución numérica de la cavidad cúbica a $Re = 1000$ para diferentes instantes de tiempo.

lación se llevó a cabo hasta un tiempo físico de simulación $T_{\text{final}} = 16\text{s}$, con un paso de tiempo $\Delta t = 0{,}0025\text{s}$ con lo que la cantidad de pasos de tiempo realizados es $k = 6400$. Se calcularon las tasas de procesamiento en Mcell/s para cada grilla y algoritmo de acuerdo a la expresión:

$$\text{rate} = \frac{N_{\text{steps}} \times N_x \times N_y \times N_z}{t_{\text{total}} \times 10^6} \left[\frac{\text{Mcells}}{\text{sec}}\right] \tag{3.44}$$

$N_x = N_y = N_z$	34	66	130	258	354
[Mcell]	0.039	0.29	2.20	17.17	44.36
$\text{rate}_{\text{SIMPLE}}$ [Mcell/sec]	18.5	109.6	337.5	472.3	550,8
rate_{FS} [Mcell/sec]	48.2	252.0	673.1	786.3	716.9
$\text{rate}_{\text{FS}}/\text{rate}_{\text{SIMPLE}}$	2.60	2.30	1.99	1.66	1.30

Tabla 3.9: Tasas de procesamiento en Mcell/s obtenidas para el algorimto FSM y SIMPLE en GPU utilizando el Equipo 1.

La figura 3.34 presenta las tasas de procesamiento versus el número de celdas del dominio computacional para las pruebas ejecutadas en el Equipo 1. Puede observarse que el algoritmo FSM tiene un mejor desempeño que el método SIMPLE, aunque para problemas grandes puede observarse una caída en el rendimiento, esto podría explicarse en que el algoritmo FSM es más sencillo que SIMPLE y tiene una intensidad de cómputo pequeña en relación a los accesos a memoria global, luego, a medida que el problema crece el ancho de banda de la GPU comienza a degradar el rendimiento.

El algoritmo SIMPLE por el contrario, como se explicó al comienzo de la sección, posee gran intensidad de cómputo con mucha re-utilización de los datos una vez leídos desde la memoria global, en consecuencia el rendimiento tiende a mejorar al incrementar el número de celdas totales de la grilla porque se aprovecha mejor la capacidad de cómputo de la GPU en relación al ancho de banda.

En cuanto a los tiempos de cómputo, el FSM es más rápido que SIMPLE y la relación de tiempos se reduce en la medida que el problema crece. Para problemas suficientemente grandes, ambos algoritmos consumen un tiempo de cómputo similar.

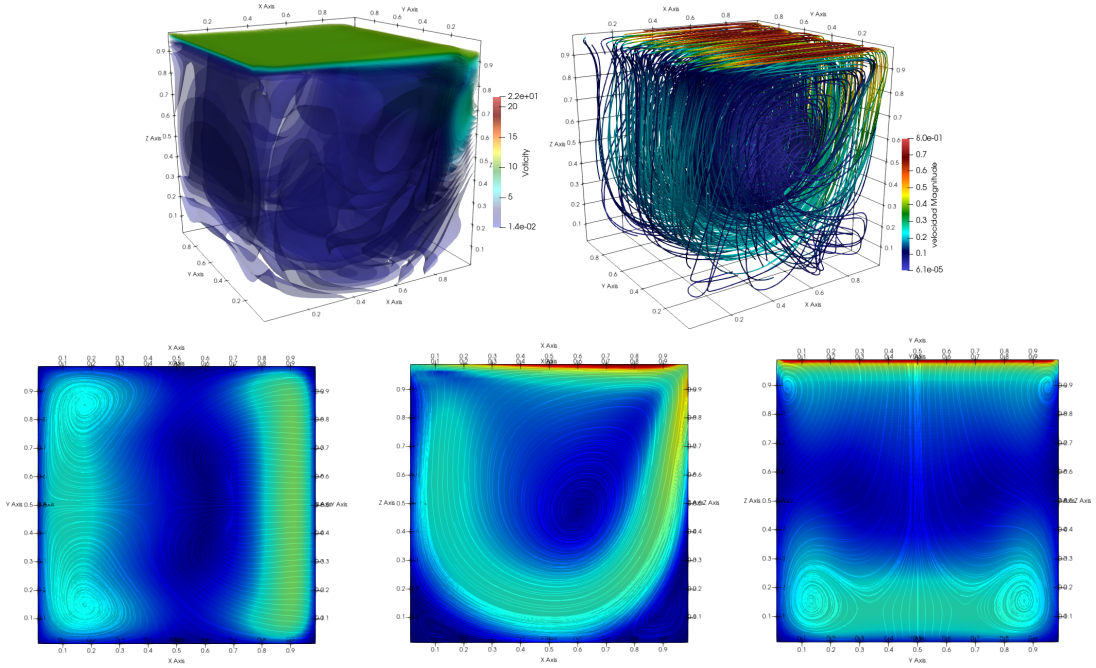

Figura 3.32: Isosuperficies de vorticidad y lineas de corriente (arriba), lineas de corriente en los planos centrales (abajo) para el problema de la cavidad cúbica a $Re = 1000$.

Conclusiones para problemas 3D

Ambos algoritmos poseen buen desempeño en GPU, explotando las capacidades de cómputo de la misma, logrando simular problemas con resolución de hasta 44 millones de celdas en el equipo 1 (GPU Tesla K40). Sin embargo debe destacarse que el FSM es en gran medida más fácil de implementar en comparación al método SIMPLE. Además de ello, el uso del esquema temporal AB para el término advectivo, permite que en caso de incorporar esquemas TVD sea mucho más sencillo debido a que se trata explícitamente, mientras que en SIMPLE, al tener un lazo interno, implica que para usar esquemas TVD, debe utilizarse una técnica DC o alguna linealización.

Por otro lado, en caso que se requieran resolver problemas estacionarios, el algoritmo SIMPLE es más adecuado ya que permite obtener la solución en tiempos reducidos en comparación al FSM, esto es particularmente útil en problemas a altos Reynolds.

Finalmente, aunque el algoritmo SIMPLE posee tasas de cómputo más bajas, para problemas grandes, los tiempos de cálculo se comienzan a parecerse a los del FSM. Lo anterior se explica en que el SIMPLE es un método segregado y no se requiere resolver a alta precisión en cada iteración del lazo, esto permite acelerar velocidad de convergencia global del método en cada paso de tiempo.

Cabe mencionar que ambos métodos se podrían mejorar considerablemente si se utiliza la FFT o métodos multigrilla para la resolución de la ecuación de la presión. Sin embargo, estas técnicas son más fáciles de implementar en el FSM por ser constantes los coeficientes de la matriz, cosa que no ocurre en SIMPLE debido a que los coeficientes en la PPE dependen del campo de velocidades.

En base a los resultados obtenidos, para los capítulos siguientes se utilizará el FSM, donde se incorporarán las técnicas de fronteras embebidas para poder resolver las ecuaciones en dominios con fronteras irregulares.

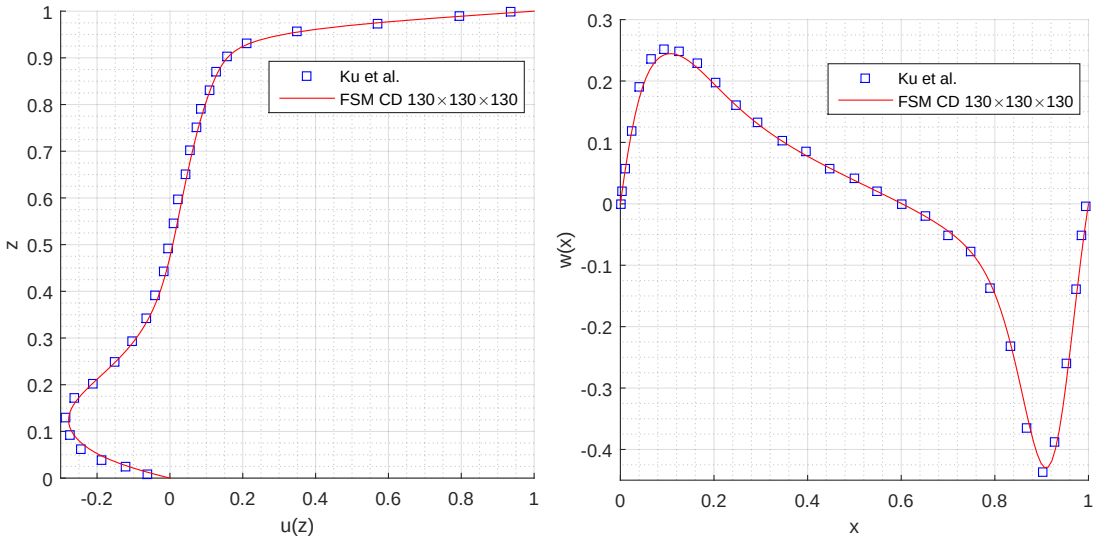

Figura 3.33: Perfiles de velocidad para las líneas centrales en el problema de la cavidad cúbica a $Re = 1000$ utilizando el algoritmo FSM solucion en estado estacionario ($t_{\text{final}} = 40$s), utilizando un esquema espacial CD para el término advectivo y difusivo, comparación con los resultados de Ku et al. [89].

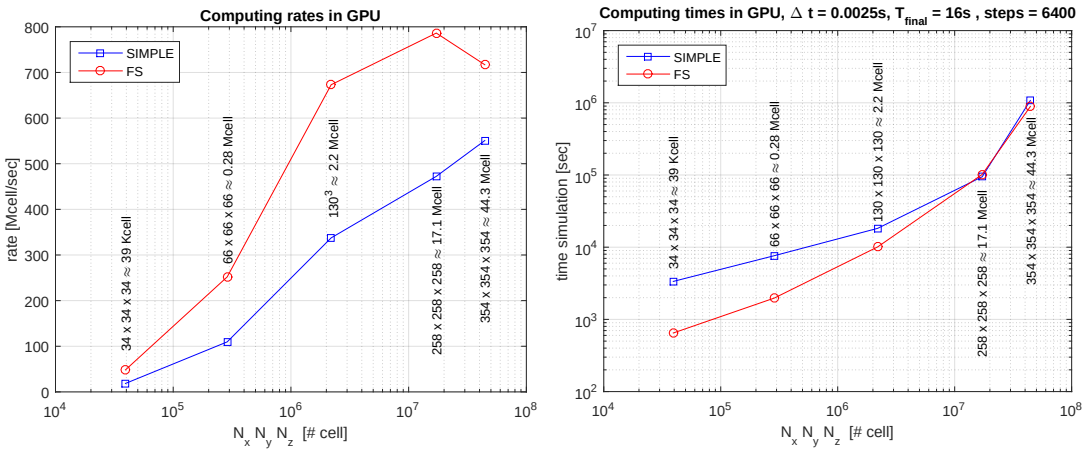

Figura 3.34: Solución numérica de la cavidad cúbica a $Re = 1000$ para diferentes instantes de tiempo.

Capítulo 4

Métodos multigrilla

A partir de una revisión bibliográfica, se identificó que existen pocas implementaciones diseñadas para ejecutarse enteramente en GPU, la mayoría son aceleraciones de resolvedores basados en CPU y por otro lado, las implementaciones en general resultan muy complejas. Esto motiva a implementar un algoritmo multigrilla (MG) en GPU. Para esta tesis se implementa un método MG centrado en celda (CCMG) basado completamente en GPU que puede utilizarse como precondicionador multinivel o directamente como resolvedor independiente.

Las estrategias MG consisten en la combinación de resolvedores iterativos clásicos con una jerarquía de discretizaciones que proporcionan un método con convergencia acelerada, constituyendo así uno de los solucionadores más eficientes disponibles en la actualidad. En estos métodos, el error se corrige como un subproblema en una grilla gruesa, donde se resuelve un problema con menos incógnitas y luego se interpola esa corrección a la grilla fina (esto se conoce como *corrección de grilla gruesa*).

4.1. Método Multigrilla en GPU

En el contexto GMG, hay dos formas de aplicar el engrosamiento (*coarsening*), MG centrado en vértices (VCMG) o centrado en celda (CCMG) [179]. En la primera, cada grilla gruesa se obtiene quitando vértices sucesivamente, partiendo de la cuadrícula fina, la segunda consiste en unir sucesivamente las celdas finas para obtener una celda en la grilla gruesa, por ejemplo se forma una celda en la malla gruesa uniendo cuatro celdas de la malla fina. Por lo tanto, en VCMG, cada grilla más gruesa es un subconjunto de nodos de la grilla fina anterior, mientras que en CCMG dan como resultado grillas no anidadas, pero resulta en un camino natural si se aplica el FVM. Cuando de trata de problemas con coeficientes de difusión muy variables entre celdas del dominio, VCMG requiere un operador de transferencia dependiente del problema [119, 168, 179], tales operadores dependientes apuntan a aproximar la condición de salto correcta usando información del operador discreto. Se han introducido algunos operadores dependientes en [9, 46] demostrando ser efectivos.

En [150] presentan un algoritmo SMG (Semicoarsing Multigrid) adecuado para ese tipo de problemas, este método se paraleliza más tarde en CPU usando memoria distribuida (versión MPI) en HYPRE [27, 55, 56]. Este método ha sido aplicado como solver directo o como precondicionador en PCG [11, 52].

Por otro lado, dentro de las implementaciones basadas en GPU del método MG, destacamos la librería de Nvidia AmgX que contiene métodos multigrilla algebraicos AMG basados en agregación, en [122] reportan hasta $5\times$ de aceleración usando una GPU Nvidia Tesla K40 versus la versión paralela en CPU de HYPRE ejecutada en una estación de trabajo con un CPU Intel Xeon E5-2690v2 10-cores a 3.0GHz. En [110, 111] diseñan e implementan una versión basada en GPU del SMG de [150], en el estudio de escalabilidad para casos 2D, reportan tiempos totales (etapas de configuración y solución) del mismo orden comparado con AmgX de Nvidia para tamaños

hasta 16M de celdas en una Tesla K40, sin embargo este método involucra la solución de algunos sistemas con métodos como el algoritmo de Thomas [150] o en una opción más compatible con las GPU como las estrategias PCR (Parallel Cyclic Reduction) [3, 47, 110].

El algoritmo MG más básico se conoce como *ciclo de dos grillas* y puede resumirse como:

1. Realizar algunas iteraciones en el sistema original, $\mathbf{A}_h\mathbf{\Phi}_h = \mathbf{b}_h$ en la grilla fina (con paso de malla h). Llamando $\hat{\mathbf{\Phi}}$ a la solución obtenida en esas iteraciones, el residuo se calcula como $\mathbf{r}_h = \mathbf{b}_h - \mathbf{A}_h\hat{\mathbf{\Phi}}$.

2. EL residuo se traslada desde la grilla fina a la gruesa con paso de malla $2h$ (op. de restricción), se realizan pocas iteraciones en esta grilla pero en una ecuación para el error: $\mathbf{A}_{2h}\mathbf{e}_{2h} = \mathbf{r}_{2h}$ comenzando desde una solución inicial homogénea $\mathbf{e}_{2h} = \mathbf{0}$.

3. El error $\mathbf{e}_{2h}$ se transfiere a la grilla fina y se realiza la corrección: $\hat{\mathbf{\Phi}}_{\text{new}} = \hat{\mathbf{\Phi}} + e_h$.

4. Repetir el proceso.

Entonces, los pasos principales que deben definirse para una estrategia GMG consisten en: operación de *suavizado* o *relajación*, *restricción*, *interpolación* o *prolongación*, *tipo de ciclo* y *operador en grilla gruesa*. La relajación reduce los errores de alta frecuencia (suavizado) usando unos pocos pasos con un método iterativo muy simple (por ejemplo Jacobi o Gauss-Seidel). La restricción es la etapa en la que el residuo en la grilla fina se transfiere a la grilla gruesa para resolver el error. La prolongación consiste en interpolar esa corrección calculada en la malla gruesa sobre la malla fina. El tipo de ciclo es el tipo de recursividad que se realiza usando el *ciclo de dos grillas*. El operador de grilla gruesa es el operador diferencial discreto que nos permitirá obtener la matriz del sistema para la malla gruesa $\mathbf{A}_{2h}$.

En el CCMG para esta tesis, la grilla gruesa se construye vía celdas. Es decir, cada celda gruesa es la unión de cuatro celdas más finas resultando grillas con tamaños de celda $2h; 4h, 8h, \ldots$. De esta forma, los centros de las celdas de una grilla gruesa no pertenecen a la grilla más fina anterior.

4.1.1. Operador en grilla gruesa y enfoque de discretización

Para obtener el operador en grilla gruesa $\mathbf{A}_{2h}$, representación de $\mathbf{A}_h$ (grilla fina), hay dos técnicas: aproximación de grilla gruesa de Galerkin (GCA) o aproximación de grilla gruesa por discretización (DCA). GCA se basa en operaciones algebraicas desde los operadores de restricción y prolongación, esto es $\mathbf{A}_{2h} = \mathbf{R}_h^{2h}\mathbf{A}_h\mathbf{P}_{2h}^h$, donde $\mathbf{R}_h^{2h}$ y $\mathbf{P}_{2h}^h$ son los operadores de restricción y prolongación respectivamente. DCA en cambio, consiste en obtener $\mathbf{A}_{2h}$ aplicando la misma técnica de discretización utilizada en $\mathbf{A}_h$. Aquí surge el principal problema en CCMG, y es que si se usa GCA, el stencil que da como resultado $\mathbf{A}_{2h}$ crece en la mayoría de los casos si se utilizan operadores de transferencia de segundo orden [119]. Es por ello que generalmente se elige DCA para CCMG, obteniendo resultados de convergencia similares a VCMG para casos homogéneos, sin embargo, en el caso de problemas muy heterogéneos, se pueden obtener tasas de convergencia muy pobres, incluso puede ocurrir divergencia si los coeficientes de difusión para construir el stencil $\mathbf{A}_{2h}$ no se eligen correctamente [168]. Según los requisitos de la arquitectura de cómputo de las GPUs, se prefiere operadores que mantengan un stencil compacto en la grilla gruesa. Aunque dichos operadores podrían causar una tasa de convergencia baja, en esos casos, eso se resuelve mediante una combinación con el método CG con un precondicionador CCMG.

4.1.2. Operadores de transferencia

El operador de prolongación mas simple que se puede usar es una interpolación constante a trozos (CP) y su adjunto escalado como operador de restricción (CR):

$$\mathbf{P}_{2h}^{h} = \begin{bmatrix} 1 & & 1 \\ & * & \\ 1 & & 1 \end{bmatrix}_{2h}^{h}, \quad \mathbf{R}_{h}^{2h} = \frac{1}{4} \begin{bmatrix} 1 & & 1 \\ & * & \\ 1 & & 1 \end{bmatrix}_{h}^{2h} \tag{4.1}$$

la expresión (4.1) indica en notación de stencil los pesos con los que el valor correspondiente de la grilla gruesa construye el valor de la celda fina y los pesos que multiplican los valores de la malla fina para realizar la restricción en la malla gruesa (ver figura 4.2 - izquierda), $*$ indica un valor de celda en la grilla gruesa. Entre las ventajas de estos operadores, se destaca que para realizar la restricción en una celda gruesa, sólo se requieren 4 lecturas de memoria en la grilla fina y para la prolongación se lee un solo valor en la grilla gruesa para cada celda de la grilla fina. Debe tenerse en cuenta que, dado que son operadores compactos, no se requieren modificaciones para las celdas de frontera, lo que facilita la implementación. Por otro lado, se puede demostrar que la construcción del operador discreto vía GCA en forma escalada, es decir $\mathbf{A}_{2h} = \frac{1}{2}\mathbf{R}_{h}^{2h}\mathbf{A}_{h}\mathbf{P}_{2h}^{h}$ es equivalente a DCA en la grilla gruesa usando coeficientes de difusión $\nu(\mathbf{x})$ que surgen de promediar aritméticamente el coeficiente de cada grilla [90, 120]. Este proceso se representa en la figura 4.1: en un primer paso se obtienen los valores en las caras ν_f a partir de los coeficientes en los centros de celda ν_f usando la media armónica, luego ν_f en las mallas gruesas se calculan como la media aritmética de los dos valores más cercanos.

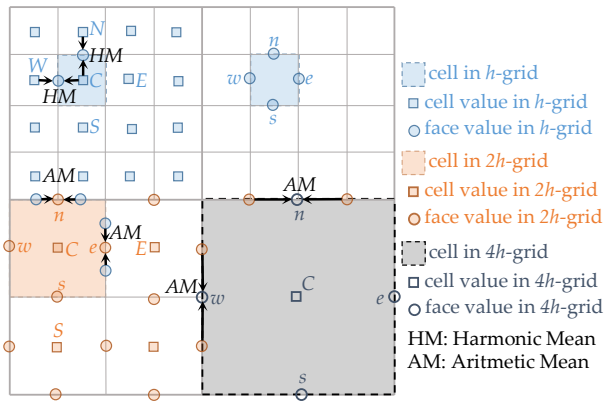

Figura 4.1: Estrategia de promediado para obtener los coeficientes de difusión en el proceso DCA escalado en diferentes grillas.

Los operadores compactos tienen la desventaja que son hasta primer orden, donde el orden se refiere al orden del polinomio que se puede interpolar exactamente más 1. Para obtener tasas de convergencia independientes del número de celdas de la grilla (tamaño de la grilla) en CCMG, la condición que debe cumplirse entre los órdenes polinómicos de los operadores de prolongación (m_P) y restricción (m_R) comparando con el orden de diferenciación de la ecuación diferencial en derivadas parciales (EDP) (m) [70] es:

$$m_P + m_R > m \tag{4.2}$$

en [119], basados en el Análisis Local de Fourier (LFA) en el contexto del CCMG, considerando los órdenes de baja y alta frecuencia para el operador, resaltan la *condición débil* de [72] que es menos restrictiva que (4.2) y enumera los órdenes del error de alta frecuencia que pueden diferir del orden polinomial (ver [119] para más detalles y el orden basado en las frecuencias del error):

$$m_P^{\text{high}} + m_R^{\text{high}} \geq m \tag{4.3}$$

la condición (4.3) solo es válida para GCA, por lo que la combinación CP-CR es una buena candidata en este tipo de problema porque tiene la equivalencia entre DCA y GCA y satisface (4.3) aunque violen la condición (4.2), además de ser fácil de implementar [90, 91, 119, 120]. En el caso de DCA, el orden del polinomio debe incrementarse en al menos uno de los operadores de transferencia para cumplir la condición (4.2).

Entre los operadores de interpolación de orden superior se pueden mencionar los siguientes: (BP) interpolación bilineal, (WP) Wesseling-Kalhil [81, 178], (KP) Kwak [93] y (MP) Mathioudakis [112]. Las expresiones para los diferentes operadores de prolongación se muestran en (4.4) y (4.5), el operador de restricción se determina en cada caso como el adjunto escalado de P.

$$\mathbf{P}_{\text{BP}} = \frac{1}{16}\begin{bmatrix} 1 & 3 & & 3 & 1 \\ 3 & 9 & & 9 & 3 \\ & & * & & \\ 3 & 9 & & 9 & 3 \\ 1 & 3 & & 3 & 1 \end{bmatrix}^{h}_{2h}, \quad \mathbf{P}_{\text{WP}} = \frac{1}{4}\begin{bmatrix} 1 & 1 & & 0 & 0 \\ 1 & 3 & & 2 & 0 \\ & & * & & \\ 0 & 2 & & 3 & 1 \\ 0 & 0 & & 1 & 1 \end{bmatrix}^{h}_{2h}, \tag{4.4}$$

$$\mathbf{P}_{\text{KP}} = \frac{1}{4}\begin{bmatrix} 0 & 1 & & 1 & 0 \\ 1 & 2 & & 2 & 1 \\ & & * & & \\ 1 & 2 & & 2 & 1 \\ 0 & 1 & & 1 & 0 \end{bmatrix}^{h}_{2h}, \quad \mathbf{P}_{\text{MP}} = \frac{1}{4}\begin{bmatrix} 1 & 0 & & 0 & 1 \\ 0 & 3 & & 3 & 0 \\ & & * & & \\ 0 & 3 & & 3 & 0 \\ 1 & 0 & & 0 & 1 \end{bmatrix}^{h}_{2h}, \tag{4.5}$$

En la figura 4.2 (derecha) están representados ambos procesos de transferencia entre grillas (BP y BR), hay que tener en cuenta que (4.4) y (4.5) solo son válidas para celdas interiores y deben modificarse en las celdas de borde, si bien hay fórmulas generales, también puede hacerse mediante el uso de nodos fantasma (*ghost nodes*), en la figura 4.2 se muestran dichos nodos en caso de restricción (cuadrados oscuros) o para prolongación (círculos obscuros). Si la condición de borde es tipo Dirichlet, los valores de los nodos G y g se extrapolan linealmente asumiendo un valor cero a lo largo del borde [119]. Para la condición de borde tipo Neumann, la extrapolación se realiza considerando que una función constante en el borde se transfiera correctamente (Neumann homogénea). Entonces los valores extrapolados hacia los nodos fantasma para restringir el residuo y prolongar en corrección de grilla gruesa son:

$$r_{\text{g}}^{Dirichlet} = -r_{\text{interior}}^{h}, \quad r_{\text{g}}^{Neumann} = r_{\text{interior}}^{h} \tag{4.6}$$

$$e_{\text{G}}^{Dirichlet} = -e_{\text{interior}}^{2h}, \quad e_{\text{G}}^{Neumann} = e_{\text{interior}}^{2h} \tag{4.7}$$

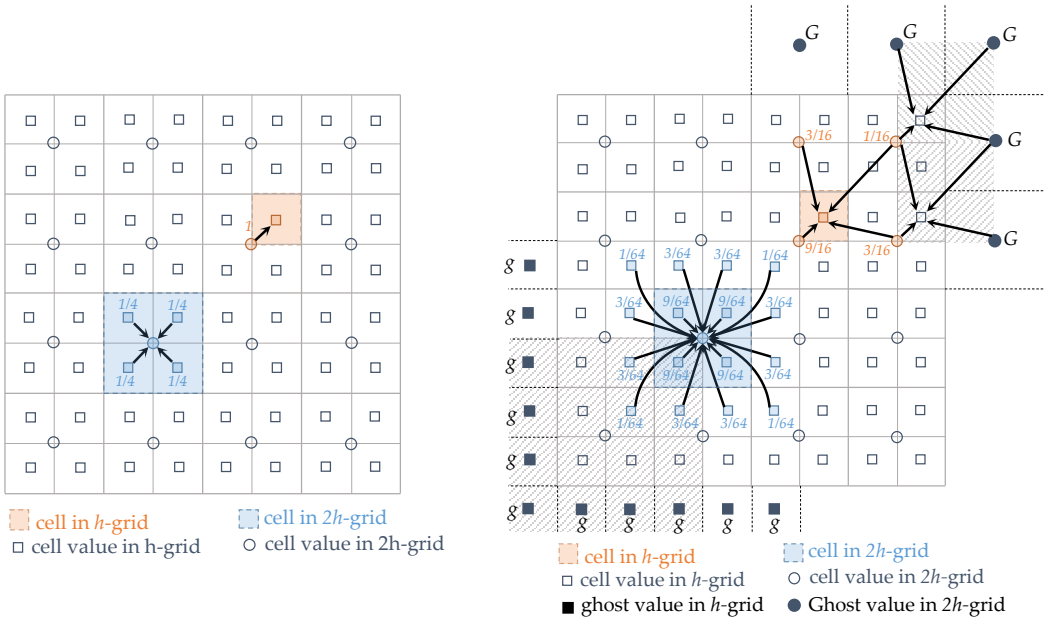

Figura 4.2: Representación de los operadores de transferencia para el caso de función constante a trozos CP/CR (izquierda) y función bilineal a trozos BP/BR (derecha). Los cuadrados y círculos indican los centros de celda de la grilla fina y gruesa respectivamente. Las operaciones de restricción y prolongación se indican en azul y naranja respectivamente.

4.1.3. Estrategia para obtención de los coeficientes de difusión del operador $\mathbf{A}_{2h}$ en la grilla gruesa

Como se mencionó antes, al usar GCA la plantilla crece demasiado en casi todas las combinaciones anteriores, esto deteriora la eficiencia de los algoritmos ejecutados en GPU debido a la mayor cantidad de lecturas en las etapas de restricción, prolongación y suavizado, siendo crucial la última ya que se requiere realizar muchas veces la operación tipo stencil ($A\Phi$). En [119] se muestran los crecimientos de las plantillas de los operadores $\mathbf{A}_{2h}$ para diversas combinaciones de operadores de transferencia partiendo del caso de plantillas de 5, 7 y 9 puntos en $\mathbf{A}_h$.

A continuación se presenta a modo de ejemplo como luce la plantilla del operador $\mathbf{A}_{2h}$ si se escoge la combinación BP-CR en el caso 2D. Considere la figura 4.3, el operador en la celda C de la grilla gruesa se obtiene como:

$$\begin{aligned}\mathbf{A}_{2h}\phi &= \left(\mathbf{R}_h^{2h}\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_C \\ &= \frac{1}{4}\left[\left(\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_1 + \left(\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_2 + \left(\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_3 + \left(\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_4\right]\end{aligned} \tag{4.8}$$

en donde el primer término se obtiene

$$\begin{aligned}\left(\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_1 = &\frac{1}{16}e_1^h\left(9\phi_E + 3\phi_C + 3\phi_{NE} + \phi_N\right) + \frac{1}{16}w_1^h\left(9\phi_C + 3\phi_N + 3\phi_W + \phi_{NW}\right) \\ &+\frac{1}{16}n_1^h\left(9\phi_N + 3\phi_C + 3\phi_{NE} + \phi_E\right) + \frac{1}{16}s_1^h\left(9\phi_C + 3\phi_S + 3\phi_E + \phi_{SE}\right) \\ &+\frac{1}{16}c_1^h\left(9\phi_C + 3\phi_E + 3\phi_N + \phi_{NE}\right)\end{aligned} \tag{4.9}$$

procediendo igual para los términos restantes, se obtiene el operador en la grilla gruesa:

$$\begin{aligned}\mathbf{A}_{2h}\phi = &e^{2h}\phi_E + w^{2h}\phi_W + n^{2h}\phi_N + s^{2h}\phi_S + c^{2h}\phi_C \\ &+ne^{2h}\phi_{NE} + nw^{2h}\phi_{NW} + se^{2h}\phi_{SE} + sw^{2h}\phi_{SW}\end{aligned} \tag{4.10}$$

cuyos coeficientes se computan a partir de los coeficientes en la grilla fina como:

$$
\begin{aligned}
e^{2h} &= \frac{1}{64}\left(3*(c_1^h+c_4^h+e_2^h+e_3^h+n_4^h+s_1^h)+9*(e_1^h+e_4^h)+(n_1^h+s_4^h)\right)\\
w^{2h} &= \frac{1}{64}\left(3*(c_2^h+c_3^h+n_3^h+s_2^h+w_1^h+w_4^h)+9*(w_2^h+w_3^h)+(n_2^h+s_3^h)\right)\\
n^{2h} &= \frac{1}{64}\left(3*(c_1^h+c_2^h+e_2^h+n_3^h+n_4^h+w_1^h)+9*(n_1^h+n_2^h)+(e_1^h+w_2^h)\right)\\
s^{2h} &= \frac{1}{64}\left(3*(c_3^h+c_4^h+e_3^h+s_1^h+s_2^h+w_4^h)+9*(s_3^h+s_4^h)+(w_3^h+e_4^h)\right)\\
ne^{2h} &= \frac{1}{64}\left(3*(e_1^h+n_1^h)+(c_1^h+e_2^h+n_4^h)\right)\\
nw^{2h} &= \frac{1}{64}\left(3*(n_2^h+w_2^h)+(c_2^h+w_1^h+n_3^h)\right)\\
se^{2h} &= \frac{1}{4}\left(3*(s_3^h+w_3^h)+(c_3^h+s_2^h+w_4^h)\right)\\
sw^{2h} &= \frac{1}{64}\left(3*(s_4^h+e_4^h)+(c_4^h+e_3^h+s_1^h)\right)\\
c^{2h} &= -\frac{3}{32}\left(e_1^h+e_4^h+w_2^h+w_3^h+n_1^h+n_2^h+s_3^h+s_4^h\right)\\
&= -\left(e^{2h}+w^{2h}+n^{2h}+s^{2h}+ne^{2h}+nw^{2h}+se^{2h}+sw^{2h}\right)
\end{aligned}
\tag{4.11}
$$

Luego, la plantilla de 5 puntos en la grilla fina se transforma en una plantilla de 9 puntos en la grilla gruesa, desde donde se mantiene de 9 puntos para las sucesivas grillas mas gruesas $(4h, 8h, \dots)$. Si se hubiese elegido la combinación BP-BR, la plantilla de 5 puntos del caso 2D se transforma en una plantilla de 21 puntos, y la situación es más crítica para los problemas en 3D.

Con ello puede verse que resulta muy poco práctica la implementación en GPU usando GCA con operadores de alto orden. Las únicas combinaciones que satisfacen (4.2) manteniendo una plantilla compacta en todas las grillas son CP-CR, WP-CR o CP-WR, mas aún, en el último caso DCA equivale a GCA si se eligen los mismos coeficientes de difusión que surgen del proceso que mostrado para CP-CR (figura 4.1) [81]. En efecto dichos coeficientes para la celda C de la grilla gruesa usando CP-CR (figura 4.3-izquierda) se obtienen usando la operación escalada:

$$
\begin{aligned}
\mathbf{A}_{2h}\phi &= \frac{1}{2}\left(\mathbf{R}_h^{2h}\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_C\\
&= \frac{1}{8}\left[\left(\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_1+\left(\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_2+\left(\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_3+\left(\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_4\right]
\end{aligned}
\tag{4.12}
$$

en donde el primer término resulta como:

$$
\left(\mathbf{A}_h\mathbf{P}_{2h}^h\phi\right)_1 = e_1^h\phi_E + w_1^h\phi_C + n_1^h\phi_N + s_1^h\phi_C + c_1^h\phi_C \tag{4.13}
$$

procediendo igual para los términos restantes, se obtiene el operador en la grilla gruesa:

$$
\mathbf{A}_{2h}\phi = e^{2h}\phi_E + w^{2h}\phi_W + n^{2h}\phi_N + s^{2h}\phi_S + c^{2h}\phi_C \tag{4.14}
$$

cuyos coeficientes se computan a partir de los coeficientes en la grilla fina como (compare con figura 4.1):

$$
\begin{aligned}
e^{2h} &= \frac{1}{8}\left(e_1^h+e_4^h\right)\ ;\ w^{2h} = \frac{1}{8}\left(w_2^h+w_3^h\right)\ ;\ n^{2h} = \frac{1}{8}\left(n_1^h+n_2^h\right)\\
s^{2h} &= \frac{1}{8}\left(s_3^h+s_4^h\right)\ ;\ c^{2h} = -\left(e^{2h}+w^{2h}+n^{2h}+s^{2h}\right)
\end{aligned}
\tag{4.15}
$$

El uso de operadores de mayor orden vía DCA motiva a investigar una manera de obtener los coeficientes de difusión para las grillas más gruesas buscando no degradar la velocidad de convergencia debido a heterogeneidades en los coeficientes de la ecuación de difusión. Desde un punto

de vista heurístico, pensando en que llevar el problema original en la grilla fina a las grillas cada vez más gruesas sucesivamente, corresponde a un proceso de homogeneización, en esta tesis se propone restringir (*upscaling*) los coeficientes de difusión utilizando el promedio geométrico de 4 celdas cada vez, en la región análoga al operador CR (ver figura 4.2 - izquierda). Si a, b, c y d denotan los centros de celda en la grilla h, usados para formar la celda A de la grilla $2h$, el coeficiente ν se computa así:

$$\nu_A = (\nu_a\ \nu_b\ \nu_c\ \nu_d)^{1/4} = exp\left[\frac{1}{4}\left(\ln(\nu_a) + \ln(\nu_b) + \ln(\nu_c) + \ln(\nu_d)\right)\right] \tag{4.16}$$

una vez obtenidos los coeficientes de difusión ν en centros de celda, la discretización en la grilla $2h$ se realiza aplicando directamente el promedio geométrico (esquema CD) o bien utilizando la media armónica en caso que los coeficientes sean fuertemente variables.

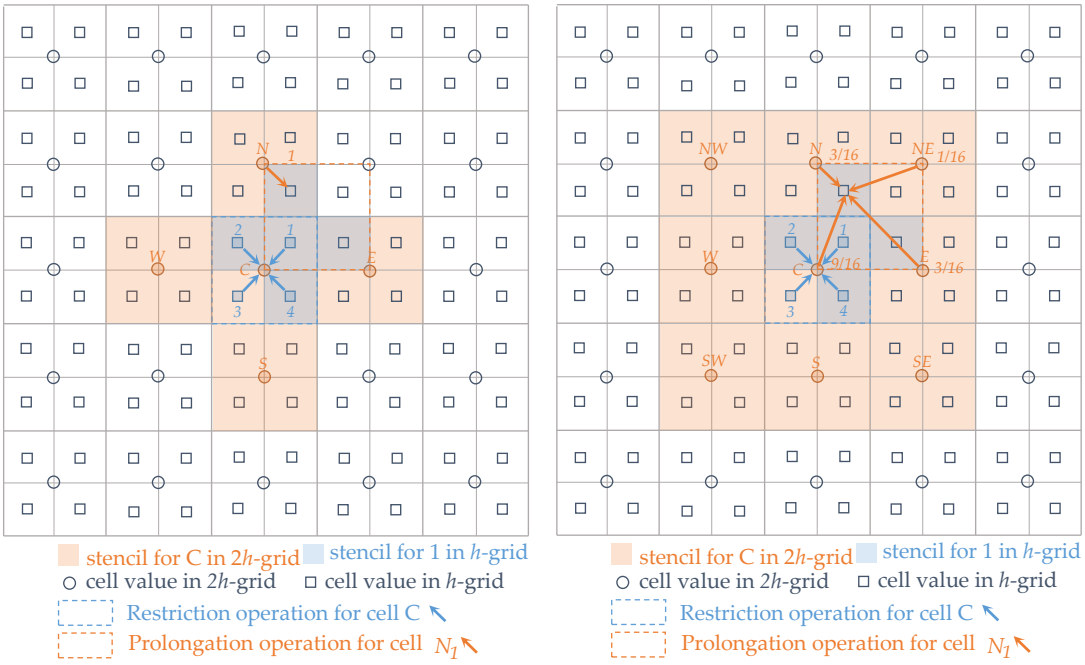

Figura 4.3: Plantilla del operador en grilla gruesa $\mathbf{A}_{2h}$ para un operador $\mathbf{A}_h$ de 5 puntos en la grilla fina utilizando las combinaciones CR-CP (izquierda) y CR-BP (derecha). Se muestra la operación de restricción para la celda $C \in 2h$-grid y la operación de prolongación para la celda $N_1 \in h$-grid vecina superior de la celda $C_1 \in h$-grid.

Al realizar los experimentos que se presentan en las secciones posteriores se ha verificado que utilizar (4.16) permite obtener una ganancia de tiempo en un factor superior a $3\times$ respecto al enfoque usado en [90, 120] (figura 4.1). Esto se debe a que la media geométrica es más representativa para los casos heterogéneos, a la vez que se utiliza la información de más información de las celdas finas al ensamblar el sistema para la grilla gruesa. Se destaca que el principal beneficio de este enfoque, es que permite utilizar todas las combinaciones de operadores. Por otro lado, en las pruebas realizadas también se encontró que las combinaciones CP-CR y BP-CR son las más efectivas, obteniendo los mejores desempeños aunque no muy distintos entre sí. Los métodos VCMG por su parte pueden armarse usando operadores-dependientes que son más efectivos, pero involucran más operaciones para su configuración y requieren mas almacenamiento, mientras que CCMG resulta más fácil de implementar en GPU por el grado de paralelismo que puede obtenerse, incluso puede implementarse usando estilo *matrix-free*, debido a que en cada grilla solamente se requiere de $\nu(\mathbf{x})$ en centro de celda para ensamblar el sistema cada vez, y por otra parte los operadores $\mathbf{P}_{2h}^{h}$ y $\mathbf{R}_{h}^{2h}$ no requieren almacenamiento ya que se toman directamente como una función que opera entre dos niveles de grilla.

4.1.4. Operadores de relajación o suavizado

El diseño de operadores de suavizado efectivos es problema dependiente. Como en esta tesis interesan operadores compatibles con implementaciones que usan paralelismo de grano fino para aprovechar la potencia de cómputo de las GPU modernas, se decidió a utilizar dos técnicas de suavizado clásicas:

- Weighted Jacobi (WJ)
- Gauss-Seidel Red-Black ordering (GSRB)

WJ es completamente paralelo, es decir en cada iteración se puede actualizar cada punto de forma independiente. GSRB es más efectivo sin perder tanto grado de paralelismo ya que se actualiza la mitad de las celdas por vez (primero nodos *Red*, luego nodos *black*). GSRB tiene la desventaja de que solamente puede usarse en el caso de stencil de 5 puntos (o 7 puntos en 3D), pero en los casos que se requieren la combinación CP-CR conserva el tamaño del stencil, en otros casos, por ejemplo, stencil de 7 o 9 puntos (en problemas 2D) o ya sea que se elija operadores de transferencia de mayor orden, deben usarse WJ o elegirse otra variante por ejemplo *four color Gauss-Seidel*. El costo de cada iteración MG es directamente proporcional al número de iteraciones de relajación o suavizado además de su influencia en la tasa de convergencia. En general basta con 1 a 3 pasos de suavizado, pero si el problema es fuertemente variable en los casos estudiados se han usado hasta 4 pasos como máximo. Los algoritmos 14 y 15 presentan ambas estrategias de suavizado.

Algoritmo 14: Método Weighted Jacobi (WJ)

entrada: $\mathbf{x}_0$ (solución inicial), $\mathbf{b}$ (RHS), $\mathbf{A}$ (matriz del sistema), $\mathbf{D}$ (diagonal principal matriz del sistema), ω (factor de amortiguamiento), $N_{\text{máx}}$ (número de iteraciones);
salida: $\mathbf{x}_N$ (solución relajada/suavizada);
para $i = 1, \ldots, N_{\text{máx}}$ **hacer**
    $\mathbf{r}_{i-1} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_{i-1}$// actualización del residuo;
    $\mathbf{x}_i \leftarrow \mathbf{x}_{i-1} + \omega \mathbf{D}^{-1}\mathbf{r}_{i-1}$// actualización de la solución;
fin

Algoritmo 15: Método Gauss-Seidel Red-Black ordering paralelo (GSRB)

entrada: $\mathbf{x}_0$ (solución inicial), $\mathbf{b}$ (RHS), $\mathbf{A}$ (matriz del sistema), $\mathbf{D}$ (diagonal principal matriz del sistema), $N_{\text{máx}}$ (número de iteraciones);
salida: $\mathbf{x}_N$ (solución relajada/suavizada);
dividir los nodos en 2 conjuntos disjuntos (damero Red-Black)
$\mathbf{x}^{\text{R}}$, $\mathbf{x}^{\text{B}}$, $\mathbf{A}^{(\text{R,B})}$, $\mathbf{A}^{(\text{B,R})}$;
para $i = 1, \ldots, N_{\text{máx}}$ **hacer**
    $\mathbf{x}_i^{\text{R}} \leftarrow \mathbf{D}_{\text{R,R}}^{-1}\left(\mathbf{b}^{\text{R}} - \mathbf{A}^{(\text{R,B})}\mathbf{x}_{i-1}^{\text{B}}\right)$// actualización solución nodos Red;
    $\mathbf{x}_i^{\text{B}} \leftarrow {\mathbf{D}^{-1}}_{\text{B,B}}\left(\mathbf{b}^{\text{B}} - \mathbf{A}^{(\text{B,R})}\mathbf{x}_i^{\text{R}}\right)$// actualización solución nodos Black;
fin

4.1.5. Tipos de ciclos multigrilla y precondicionador multinivel

Partiendo del ciclo de dos grillas y el uso de la jerarquía de grillas, hay tres combinaciones principalmente usadas basadas en la recursión de ese algoritmo, conocidas como ciclo V, W y F. El primero constituye el ciclo más simple, el ciclo W esta pensado para pasar más tiempo en las grillas más gruesas y recordando que los barridos son más rápidos que en las grillas finas, generalmente resulta superior a un ciclo V cuando los algoritmos están basados en CPU. El ciclo F (*full*

multigrid cycle) es una serie de ciclos V invertidos, comenzando desde la malla más gruesa. En la bibliografía se puede encontrar mucha información sobre esto. Aquí por simplicidad algorítmica para la implementación solamente se discuten los ciclos V y W.

El ciclo W puede ser más efectivo en el sentido de que requieren menos iteraciones MG totales para llegar a una tolerancia de error dada, siendo más adecuado para trabajar en arquitecturas paralelas donde el número de procesos en paralelo es $O(10) \sim O(100)$, pero en las GPU actuales el orden de procesadores en general es de $O(1000)$, entonces el uso de esa estrategia afecta la eficiencia del método ya que el nivel de paralelismo no es suficiente para la cantidad de procesadores disponibles.

Finalmente en el algoritmo 16 se describen los pasos del la estrategia MG que puede usarse ya sea como solver iterativo independiente (CCMG) o como operador $\mathbf{M}$ de precondicionamiento en PCG ($\mathbf{M}^{-1}_{CCMG}$). Por otro lado, en el método PCG, se requiere que la matriz de precondicionamiento $\mathbf{M}$, sea un operador lineal definido positivo y preferentemente simétrico. Cuando se utiliza MG como un precondicionador, esto se puede garantizar si se respetan las siguientes condiciones: usar operadores de relajación simétricos y aplicar la misma cantidad de pasos de pre y post relajación [168, 179].

Para simplificar aquí $\mathbf{A}$ denota una estructura que contiene los elementos necesarios para computar el producto $\mathbf{A}_l\mathbf{x}_l$ en el nivel l de grilla correspondiente. Considerando que $\mathbf{A}_l$ es simétrica, una opción es almacenando las 2 diagonales inferiores por matriz (diagonales de los vecinos W y S), luego los elementos de la diagonal principal C se determinan como la suma de los elementos restantes de la fila. ν_{pre} y ν_{post} corresponden al número de iteraciones (WJ or GSRB) en la etapa de pre y post relajación, que en MG-puro pueden diferir, sin embargo, en PCG se debe elegir $\nu_{\text{pre}} = \nu_{\text{post}}$ para mantener la simetría del precondicionador. γ permite seleccionar entre ciclo V y ciclo W, debe tenerse en cuenta que el algoritmo se simplifica un poco al considerar solo el ciclo V a la vez que requiere el almacenamiento de 2 vectores menos ya que no se debe acumular las correcciones sobre los ciclos. En la línea 8 se resuelve de forma aproximada el sistema en una grilla gruesa de (4×4) usando GSRB or CG, en las pruebas no se han observado diferencias en la convergencia global al resolver a precisión máquina en una grilla gruesa de (1×1). Respecto a las operaciones de transferencia entre grillas, denotadas anteriormente en forma matricial como $\mathbf{R}^{2h}_h$ y $\mathbf{P}^h_{2h}$, una de las ventajas de CCMG es que las operaciones se pueden computar de forma muy simple sin que se requiera almacenar explícitamente alguna matriz.

4.1.6. Detalles de la implementación del método CCMG en GPU

En el solver implementado se requieren definir los vectores de diferentes tamaños según el nivel de grilla, para esto se deben alocatar en GPU y en algunos casos pre calcular algunas cantidades, como W_l y S_l, subdiagonales de la matriz del sistema $\mathbf{A}_l$, como también la configuración de grillas y bloques para ejecutar los kernels en el nivel que corresponda. Esta etapa corresponde a un *setup* y se realiza mediante punteros a vectores de punteros en GPU. Esto permite realizar todo de forma dinámica (alocatación y prellenado si corresponde). En particular para el ciclo V se definen los vectores de punteros al device $\_W*$,$\_S*$, $\_r*$, $\_r^*_{\text{new}}$, $\_e*$ que contienen los punteros a arreglos en el device de diferentes tamaños uno para cada nivel de grilla.

Paralelización de los operadores de transferencia

La estrategia adoptada para la implementación de los kernels es que cada hilo (CUDA threads) procese un punto de la grilla. Como en una transferencia hay involucradas dos grillas (fina y gruesa), se distinguen estos dos casos, para restricción se consideran la cantidad de nodos de la grilla gruesa y en la prolongación se considera el número de nodos de la grilla fina. Debido al poco rehúso de datos en las operaciones involucradas, se ha omitido el uso de la memoria compartida y solamente se utiliza la memoria de registro y lecturas a memoria global de la GPU. Respecto a esto, para el operador CR se requieren 5 operaciones a memoria global (1 escritura de $\mathbf{r}_H$ en $grilla - H$

Algoritmo 16: Ciclo multigrilla para usar como resolvedor o precondicionador

entrada: l (current level), $\mathbf{b}_h$ (RHS), $\mathbf{A}$ (system matrix for all levels), ν_{pre} (number of pre-smoothing steps), ν_{post} (number of post-smoothing steps), γ (number of cycles, $\gamma = 1$:V-cycle, $\gamma = 2$:W-cycle);
salida: $\mathbf{e}_h$ (correction at level l);
$\mathbf{e}_h \leftarrow \mathbf{0}$;
$\mathbf{e}_h \leftarrow \text{smooth}(\mathbf{A}_h, \mathbf{b}_h, \mathbf{e}_h, \nu_{\text{pre}})$ // pre-smoothing;
$\mathbf{r}_h \leftarrow \mathbf{b}_h - \mathbf{A}_h\mathbf{e}_h$ // update the residual ;
$\mathbf{r}_H \leftarrow \text{restriction}(\mathbf{r}_h)$ // transfer residual to coarse level;
si $l == 0$ **entonces**
    solve $\mathbf{A}_H\mathbf{e}_H = \mathbf{r}_H$ // obtain error at coarse level;
fin
en otro caso
    $\mathbf{e}_H \leftarrow \mathbf{0}$;
    para $i = 1, \gamma$ **hacer**
        $\mathbf{r}_{H,\text{new}} \leftarrow \mathbf{r}_H - \mathbf{A}_H\mathbf{e}_H$ // update the residual;
        $\mathbf{e}_{H,\text{new}} \leftarrow MG_{\text{CYCLE}}(l-1, \mathbf{r}_{H,\text{new}}, \mathbf{A}, \nu_{\text{pre}}, \nu_{\text{post}}, \gamma)$// recursion;
        $\mathbf{e}_H \leftarrow \mathbf{e}_H + \mathbf{e}_{H,\text{new}}$ // accumulate corrections over cycles;
    fin
fin
$\mathbf{e}_{h,\text{CGC}} \leftarrow \text{prolongation}(e_H)$ // transfer coarse-grid correction to fine grid;
$\mathbf{e}_h \leftarrow \mathbf{e}_h + \mathbf{e}_{h,\text{CGC}}$ // add up correction;
$\mathbf{e}_h \leftarrow \text{smooth}(\mathbf{A}_h, \mathbf{b}_h, \mathbf{e}_h, \nu_{\text{post}})$ // post-smoothing;

y 4 lecturas de $\mathbf{r}_h$ en $grilla - h$) y para CP solo 2 (1 escritura $\mathbf{e}_h$ y una lecturas desde $\mathbf{e}_H$). Como la paralelización resulta trivial, no hay más detalles. La paralelización del operador de restricción CR se puede realizar directamente en un solo kernel, pues no hay modificaciones en las celdas que limitan el dominio. Para CP en cambio, se distinguen los casos (interior, ejes, vértices). Si se quiere implementar operadores de mayor orden, tanto restricción como prolongación convienen realizarse en diferentes rutinas ya que deben modificarse en los contornos. Respecto a la cantidad de operaciones a memoria involucradas, es mucho mayor que al usar operadores compactos (CP-CR) y esto se acentúa en las GPUs más antiguas.

Configuración del operador lineal en grillas gruesas

El operador lineal en cada nivel $\mathbf{A}_l$ se implementa a través de lo arreglos W_l y S_l que contienen las diagonales inferiores (west y south) para el problema de Poisson en caso 2D con coeficientes variables. Estas diagonales se pueden computar mediante el proceso descrito en la figura 4.1 paralelizado mediante una función $((W_{2h}, S_{2h}) \leftarrow f(W_h, S_h), (W_{4h}, S_{4h}) \leftarrow f(W_{2h}, S_{2h}), \dots)$ que llena las diagonales de cada nivel a partir de la del nivel anterior. Otra opción para el caso heterogéneo es almacenar directamente un solo vector en cada nivel con los coeficientes de difusión ν_l calculados mediante (4.16) y operando en cada nivel con estilo matrix-free con la misma función del nivel más fino pero cambiando la configuración de grillas y bloques de hilos. Ambas estrategias tienen una performance similar siendo la segunda más simple de implementar. Cabe mencionar que para el caso homogéneo, el stencil en cada nivel es con los mismos coeficientes y no es necesario crear ninguna función adicional para ello.

4.2. Desempeño del método MG en GPU en la solución de problemas tipo Poisson

Para mostrar las ventajas en términos del rendimiento para las implementaciones basadas en GPU presentadas en esta tesis, se realizan comparaciones con bibliotecas de álgebra lineal bien establecidas como HYPRE y AmgX, en problemas tipo Poisson (equivalentes a la PPE) en el caso 2D con coeficientes constantes (ver ecuación (2.75)) y variables en el dominio (ver ecuación (2.66)).

La biblioteca HYPRE de resolvedores lineales hace posibles simulaciones detalladas más grandes, al resolver más rápido que los métodos tradicionales a gran escala. Ofrece un conjunto integral de resolvedores escalables para simulación científica a gran escala, que presenta métodos MG paralelos para problemas en grillas estructuradas y no estructuradas. La biblioteca HYPRE es altamente portátil y admite varios lenguajes.

AmgX es una biblioteca acelerada por GPU flexible que agiliza la solución de grandes sistemas lineales, permitiendo al usuario construir mediante una API-C varios resolvedores y confeccionadores optimizados para el paralelismo masivo en GPU y extendiendo a multiGPU mediante la biblioteca MPI.

Para realizar los experimentos numéricos, se utilizó el equipo 2 (GPU Tesla V100) considerando 32 cores para las ejecuciones en CPU paralelo. Para los campos de conductividad variables se utilizó la librería CURAND [4] para la generación de secuencias aleatorias.

4.2.1. Prueba 1: Problema de Poisson 2D con coeficientes uniformes

Esta prueba consiste en la solución numérica vía FVM del problema de Poisson en un cuadrado unitario $\Omega = [0,1] \times [0,1]$ con coeficientes de difusión constantes y condiciones de contorno tipo Dirichlet homogéneas ($\phi = 0$):

$$-\nabla^2 \phi(\mathbf{x}) = 1 \tag{4.17}$$

Para las pruebas se consideran las siguientes variantes basadas en CPU y GPU:

1. **Variantes basadas en GPU:**
   - CG sin precondicionador: CG_{gpu}.
   - PCG, precondicionador basado en las series de Neumann truncadas a primer orden y un solo nivel de precondicionamiento: PCG_{TNS1}.
   - PCG precondicionado con multigrilla centrado en celda (CCMG) que consiste en un ciclo V o W usando 2 pre/post pasos de relajación ($\text{V}(2,2)$ o $\text{W}(2,2)$) de GSRB o WJ: $PCG_{\text{mgV-GS}}$, $PCG_{\text{mgV-WJ}}$ y $PCG_{\text{mgW-GS}}$.
   - CCMG como solver independiente con las mismas variantes de ciclos y operadores de suavizado que los precondicionadores: $MG_{\text{V-GS}}$, $MG_{\text{V-WJ}}$ y $MG_{\text{W-GS}}$.
   - PCG precondicionado con un ciclo V Multigrilla Algebraico (AMG) de Nvidia AmgX, con la configuración $\text{V}(2,2)$ y L1-Jacobi Smoother: PCG_{AmgX}.
2. **Variantes en CPU:**
   - CG sin precondicionador versión paralela OpenMP: $CG_{\text{cpu-OMP}}$.
   - CG sin precondicionador versión paralela MPI de HYPRE: $CG_{\text{cpu-hypre}}$.
   - Semicoarsing Algebraic Multigrid de HYPRE con la configuración $\text{V}(1,1)$ y relajador GS: SMG_{hypre}.
   - PCG con Semicoarsing Algebraic Multigrid de HYPRE, con la misma configuración que SMG_{hypre}: $PCG_{\text{SMG-hyp}}$.

A excepción de AmgX, todas las variantes basadas en GPU corresponden a implementaciones desarrolladas en la tesis. Las versiones paralelas en CPU se ejecutaron con 32 procesadores (implementaciones propias basadas en OpenMP y otras basadas en MPI utilizando HYPRE). La solución en todos los casos se lleva a cabo hasta una reducción relativa de 6 órdenes de magnitud en la norma L_2 del residuo, es decir $||\mathbf{r}^k||/||\mathbf{r}^0|| < 10^{-6}$, donde el residuo en la iteración k se computa como $\mathbf{r}^k = \mathbf{b} - \mathbf{A}\mathbf{\Phi}^k$. Los dominios computacionales para las pruebas varían desde 1024 a 16384 celdas por dirección dependiendo de la memoria RAM disponible en la GPU o en la CPU. En los algoritmos que involucran CCMG (como solver o precondicionador) en GPU la grilla más gruesa es de 4×4 celdas.

Resultados y conclusiones de la Prueba 1

La figura 4.4 muestra la solución numérica para el problema (4.17). La tabla 4.1 presenta los tiempos de cálculo y el número de iteraciones CG/PCG o ciclos MG realizados para llegar al nivel de error definido.

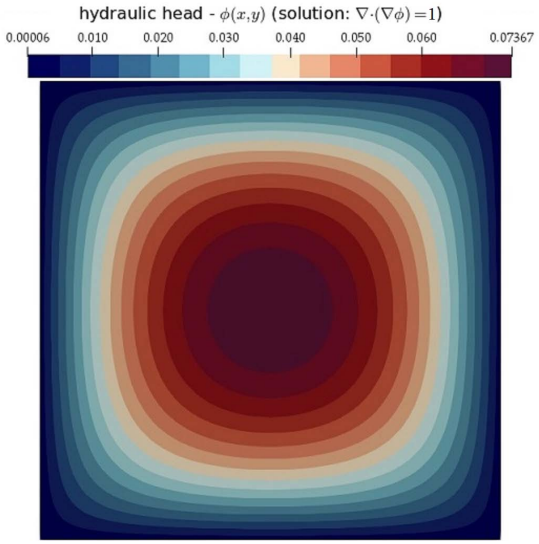

Figura 4.4: Solución numérica de la ecuación (4.17).

Tabla 4.1: Tiempos de calculo en segundos y número de iteraciones para diferentes variantes del solver y diferentes tamaños de dominio. Las 4 primeras variantes se ejecutan en CPU y el resto corresponden a variantes en GPU.

$N_x = N_y$	1024		2048		4096		8192		16384	
$N_x \times N_y$	1.049 M celdas		4.194 M celdas		16.77 M celdas		67.11 M celdas		268.4 M celdas	
Solver	t [s]	# iter	t [s]	# iter	t [s]	# iter	t [s]	# iter	t [s]	# iter
$CG_{\text{cpu-OMP}}$	3.19	1672	42.90	3377	340.0	6819	3256.6	13756	33542.4	27745
$CG_{\text{cpu-hypre}}$	3.32	1672	37.8	3377	390.9	6819	3418.8	13756	39264.6	27745
SMG_{hypre}	0.31	9	1.63	9	8.34	10	23.6	10	122.9	11
$PCG_{\text{SMG-hyp}}$	0.32	5	1.66	6	7.37	6	22.0	6	83.57	6
CG_{gpu}	0.74	1672	3.64	3377	24.4	6819	186.0	13756	1494.4	27745
PCG_{TNS1}	0.49	885	2.58	1985	17.2	3604	129.5	7273	1110.2	14666
$PCG_{\text{mgV-GS}}$	0.035	6	0.078	7	0.23	7	0.85	7	3.68	7
$PCG_{\text{mgV-WJ}}$	0.056	12	0.113	13	0.34	13	1.41	14	5.78	14
$PCG_{\text{mgW-GS}}$	1.22	5	2.47	5	4.97	5	9.77	5	20.75	5
$MG_{\text{V-GS}}$	0.042	9	0.078	9	0.24	10	0.96	10	3.92	10
$MG_{\text{V-WJ}}$	0.18	47	0.39	58	1.26	72	5.31	89	25.52	111
$MG_{\text{W-GS}}$	1.31	6	2.65	6	5.29	6	11.1	6	23.97	6
PCG_{AmgX}	0.13	9	0.43	9	1.67	10	6.50	10	-	-

En base a estos resultados se pueden destacar las siguientes conclusiones:

- Los algoritmos en CPU paralelo (utilizando los paradigmas OpenMP y MPI) usando 32 procesadores tienen un desempeño similar entre ellos. Mientras que con la implementación GPU (CG_{gpu}) se tiene una ganancia en los tiempos de entre $4\times$ a $22\times$ mejorando en la medida que el problema crece en número de celdas.

- Puede verse que para problemas grandes, es mandativo el uso de precondicionadores. Si se comparan las versiones en GPU CG_{gpu} y PCG_{TNS1} puede observarse que a pesar que las iteraciones se reducen en un 50 %, los tiempos de cómputo se reducen solo en 70 % (ganancia en un factor de $1{,}4\times$)debido al costo del precondicionador basado en las series de Neumann truncadas a 1er orden, requiriendo a su vez el almacenamiento de otro vector adicional para realizar la operación.

- Comparando la efectividad de los operadores de relajación *Weighted-Jacobi* y *Gauss-Seidel Red-Black* puede verse que al usar CCMG como solver independiente con un ciclo V (compare $MG_{\text{V-GS}}$ vs. $MG_{\text{V-WJ}}$), GSRB a pesar de tener un grado de paralelización de la mitad respecto de WJ, la operación de suavizado resulta más efectiva pues en la medida que el problema crece, la cantidad de ciclos V usando WJ es hasta 10 veces más para llegar al mismo nivel de error, efecto que se traduce a los tiempos de cómputo, ganando más de $5\times$ de tiempo al usar $MG_{\text{V-GS}}$ para la mayoría de los tamaños. En caso que se utilice CCMG como precondicionador dentro de PCG el aumento en la iteraciones no es tan marcado llegando hasta el doble, y los tiempos de cómputo guardan una relación de $\sim 1{,}5\times$ (compare $PCG_{\text{mgV-GS}}$ vs. $PCG_{\text{mgV-WJ}}$).

- En cuanto al tipo de ciclo MG, comparando $MG_{\text{V-GS}}$ vs. $MG_{\text{W-GS}}$, puede observarse que a pesar de que el segundo requiere sólo 6 ciclos mientras el primero entre 9 y 10, los ciclos V son mucho más baratos respecto a los ciclos W, esto se debe a que este ciclo está pensado para pasar más tiempo en los niveles más gruesos y no se aprovecha el paralelismo de grano fino donde se tiene un número grande de procesadores disponibles ($\sim O(1000)$ hilos). La diferencia de tiempo entre un ciclo y va desde $30\times$ a $6\times$ en favor del ciclo V disminuyendo en la medida que crece el número de celdas. Si se utiliza el ciclo MG W como precondicionador se observan prácticamente las mismas diferencias (compare $PCG_{\text{mgV-GS}}$ vs. $PCG_{\text{mgW-GS}}$).

- En lo que respecta al uso de CCMG como solver independiente o como precondicionador multinivel en PCG, si se compara $MG_{\text{V-GS}}$ vs. $PCG_{\text{mgV-GS}}$ se deduce que iteraciones requeridas en PCG son entre 66 % y 78 % que compensado con el mayor costo por iteración PCG respecto al ciclo MG independiente resultan en tiempos de cómputo muy similares siendo hasta un 5 % más rápido PCG. Cabe mencionar que el uso de CCMG como precondicionador se hace necesario en problemas de Poisson con coeficientes muy heterogéneos donde CCMG falla como solver independiente (ver siguiente subsección).

- En la implementación paralela CPU de HYPRE, comparando el método SMG como solver independiente o como precondicionador (SMG_{hypre} vs. $PCG_{\text{SMG-hypre}}$), la diferencia es algo más notable, en este caso PCG llega a ser hasta $1{,}47\times$ más rápido que SMG en el dominio de $16384^2 = 268{,}4$ Millones de celdas.

- Comparando con la librería en GPU de AmgX (PCG_{AmgX}), el solver $PCG_{\text{mgV-GS}}$ es de $3{,}7\times$ a $7{,}6\times$ más rápido, con la ventaja que el requerimiento de memoria de la implementación propia es de 4.83GB (18.14GB resp. en tamaño 16384^2) mientras que la versión AmgX requiere 29.03GB para el caso $8192^2 = 67{,}11$ Millones de celdas siendo 32GB el límite en la tarjeta utilizada en las pruebas (Nvidia Tesla V100).

- Comparando la versión GPU $PCG_{\text{mgV-GS}}$ con la versión paralela en CPU $PCG_{\text{SMG-hyp}}$ la ganancia en tiempo llega a ser hasta $32\times$ en favor de la GPU, esto significa que en el mejor

de los casos, esto es, sin considerar el tiempo de comunicación entre nodos de un clúster, para lograr un desempeño similar, se requieren 32 equipos de cómputo similares, es decir $32 * 32 = 1024$ procesadores en CPU.

- Para finalizar, la figura 4.5 muestra la comparación de los tiempos para los diferentes tamaños de grilla, de las versiones $PCG_{\text{mgV-GS}}$, PCG_{AmgX} y $PCG_{\text{SMG-hypre}}$. Puede apreciarse que la tendencia en los órdenes de tiempos de cómputo se mantiene en la medida que se refina la malla.

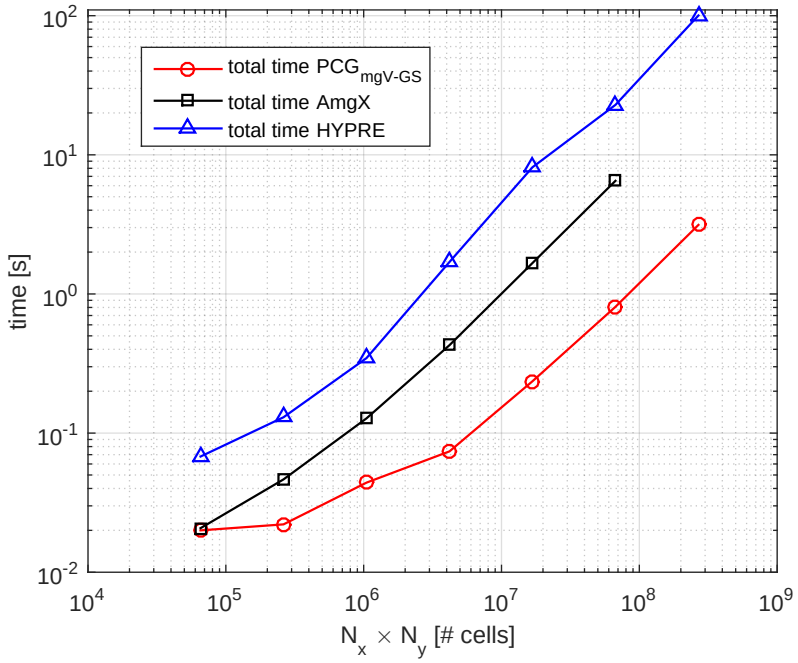

Figura 4.5: Tiempos de cálculo en segundos versus tamaño de la grilla, comparación de los resolvedores en GPU $PCG_{\text{mgV-GS}}$, PCG_{AmgX} y la versión paralela en CPU $PCG_{\text{SMG-hypre}}$.

4.2.2. Prueba 2: Problema de Poisson 2D con coeficientes altamente variables

A los efectos de mostrar el desempeño de los métodos implementados en problemas con coeficientes variables, se resolverá un problema que resulta un verdadero desafío. El problema consiste en resolver el flujo en un medio poroso con alto grado de heterogeneidad en las propiedades hidráulicas del medio. Esto requiere resolver un problema de Poisson con coeficientes de difusión aleatorios con fuertes discontinuidades. La variación de la permeabilidad absoluta es directamente proporcional a la variación en la discontinuidad de los coeficientes de la matriz del sistema, alcanzando muchos órdenes de magnitud, por ejemplo, por encima de 12 órdenes para un campo de conductividad K lognormal con una varianza $\sigma^2_{\ln K} = 9$.

Considere las ecuaciones de Darcy sin divergencia [19]

$$\mathbf{u} = -\mathbf{K}\nabla\phi \tag{4.18}$$

donde $\mathbf{u}$ es la velocidad de Darcy, $\mathbf{K}$ el tensor de conductividad hidráulica y ϕ el potencial hidráulico. Combinando (4.18) con la condición de incompresibilidad (2.37) se obtiene la ecuación de flujo en un medio poroso:

$$\nabla\cdot\left[\mathbf{K}(\mathbf{x})\nabla\phi(\mathbf{x})\right] = 0 \tag{4.19}$$

aquí se considera $\mathbf{K}(\mathbf{x}) = \text{diag}(k_1(\mathbf{x}), k_2(\mathbf{x}))$ y $k_1 = k_2 = k(\mathbf{x})$, tratando el caso donde $k(\mathbf{x})$ es discontinuo entre celdas. Se consideran campos de conductividad con distribución lognormal con función de correlación exponencial para varianzas de $\ln(\mathbf{K})$ hasta un valor de 9.

El grado de heterogeneidad es gobernado por las propiedades estadísticas λ y $\sigma_{\ln K}$ que son la longitud de correlación y la desviación estándar del logaritmo de K.

Para las pruebas se consideran dominios cuadrados $L_x = L_y = L$ con una resolución de malla $\lambda/\Delta x = \lambda/\Delta y = 10$ con un paso de malla $\Delta x = \Delta y = h = 5$m. El tamaño del dominio físico (L) se escaló de acuerdo al número de celdas por dirección ($N_x = N_y$) y la longitud de correlación λ, del siguiente modo: $L/\lambda \in \{25{,}6;\ 51{,}2;\ 102{,}4;\ 204{,}8;\ 409{,}6;\ 819{,}2;\ 1638{,}4\}$, resultando en grillas desde $\sim 0{,}07$ a $\sim 268{,}4$ Millones de celdas.

Las condiciones de borde para la ecuación de flujo (4.19) son:

- Condición tipo Neumann (no flujo) para bordes *north* y *south*.
- Condición tipo Dirichlet (valor fijo) para bordes *east* y *west*.

En la figura 4.6 pueden verse las condiciones de borde y dominios definidos para las pruebas.

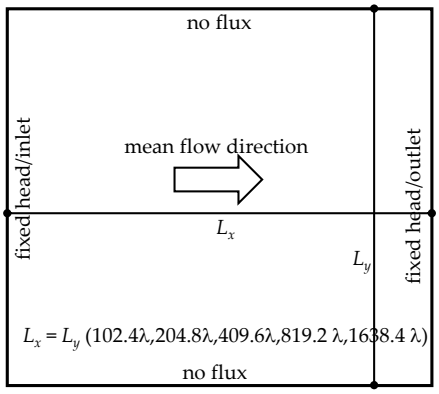

Figura 4.6: Configuración del dominio y condiciones de borde para la ecuación de flujo (4.19).

Campo de conductividad

El campo de conductividad se generó en GPU utilizando el método propuesto por [154, 155]. El campo se determina a partir de $\mathbf{K}(\mathbf{x}) = \mathbf{I}\exp(f(\mathbf{x}))$, donde:

$$f(\mathbf{x}) = \sqrt{\frac{2}{N}}\sigma^2_{\ln K}\sum_{i=1}^{N}\cos\left(m_1^{(i)}x + m_2^{(i)}y + \theta^{(i)}\right) \tag{4.20}$$

aquí $\sigma^2_{\ln K}$ es la varianza de $\ln(K)$; $\mathbf{x} = (x, y)$ son las coordenadas de un punto en el dominio computacional; $m_{1,2}^{(i)}$ se muestrean de acuerdo con una función de densidad de probabilidad conjunta elegida de acuerdo con el tipo de correlación requerida. $\theta^{(i)} \sim U(0, 2\pi)$ se muestrea usando una distribución uniforme. Para el caso de correlación exponencial, $k_{1,2}^{(i)}$ debe muestrearse con la distribución bidimensional de Cauchy-Lorentz:

$$p(m_1, m_2) = \frac{1}{2\pi}\frac{\lambda^2}{\left[(m_1\lambda)^2 + (m_2\lambda)^2 + 1\right]^{3/2}} \tag{4.21}$$

Para obtener m_1 y m_2 se busca una función de distribución acumulada (CDF) de la ecuación (4.21) integrando p en $\mathbb{R}^2$. Utilizando el cambio de variables $m_1 = \rho\cos\varphi/\lambda$ y $m_2 = \rho\sin\varphi/\lambda$, con el siguiente jacobiano para la transformación ρ/λ^2 resulta en:

$$\begin{gathered}\iint_{\mathbb{R}^2}\frac{1}{2\pi}\frac{\lambda^2\ dxdy}{\left[(m_1\lambda)^2 + (m_2\lambda)^2 + 1\right]^{3/2}} = \\ \lim_{R\to\infty}\int_0^{2\pi}\int_0^{R}\frac{1}{2\pi}\frac{\rho\ d\varphi d\rho}{(\rho^2+1)^{3/2}} = \lim_{R\to\infty}\left[1 - \frac{1}{\sqrt{R^2+1}}\right] = 1\end{gathered} \tag{4.22}$$

entonces la CDF es $\Theta(r) = 1 - \frac{1}{\sqrt{r^2+1}} = u$ y su inversa:

$$r = \left[\frac{1-(1-u)^2}{(1-u)^2}\right]^{1/2} \tag{4.23}$$

Eligiendo $u_i \sim U(0,1)$ y $\varphi_i \sim U(0,2\pi)$ uniformemente distribuidos, $m^{(i)}$ se determina como:

$$m_1^{(i)} = r_i \cos\varphi_i/\lambda, \quad m_2^{(i)} = r_i \sin\varphi_i/\lambda \tag{4.24}$$

Los campos generados tienen la siguiente función de correlación:

$$C_{\text{exp}}(r) = \sigma^2_{\ln K}\ exp\left(-\frac{|r|}{\lambda}\right) \tag{4.25}$$

donde $|r|$ es la distancia entre dos puntos.

En las pruebas se resuelve el problema (4.19) para diferentes varianzas de la log-conductividad en los siguientes rangos $\sigma^2_{\ln K} \in \{1; 2{,}25; 4; 6{,}25; 9\}$, eligiendo los solvers que fueron más eficientes en la prueba 1. A modo de ejemplo, en la figura 4.7 se muestra un campo de conductividad generado aleatoriamente con el método (4.20) para una grilla de 512×512 y los campos de conductividad para las grillas más gruesas utilizadas en el precondicionador basado en CCMG, calculados mediante la estrategia (4.16) propuesta en esta tesis (grillas de 256×256, 128×128, 64×64, 32×32 y 16×16).

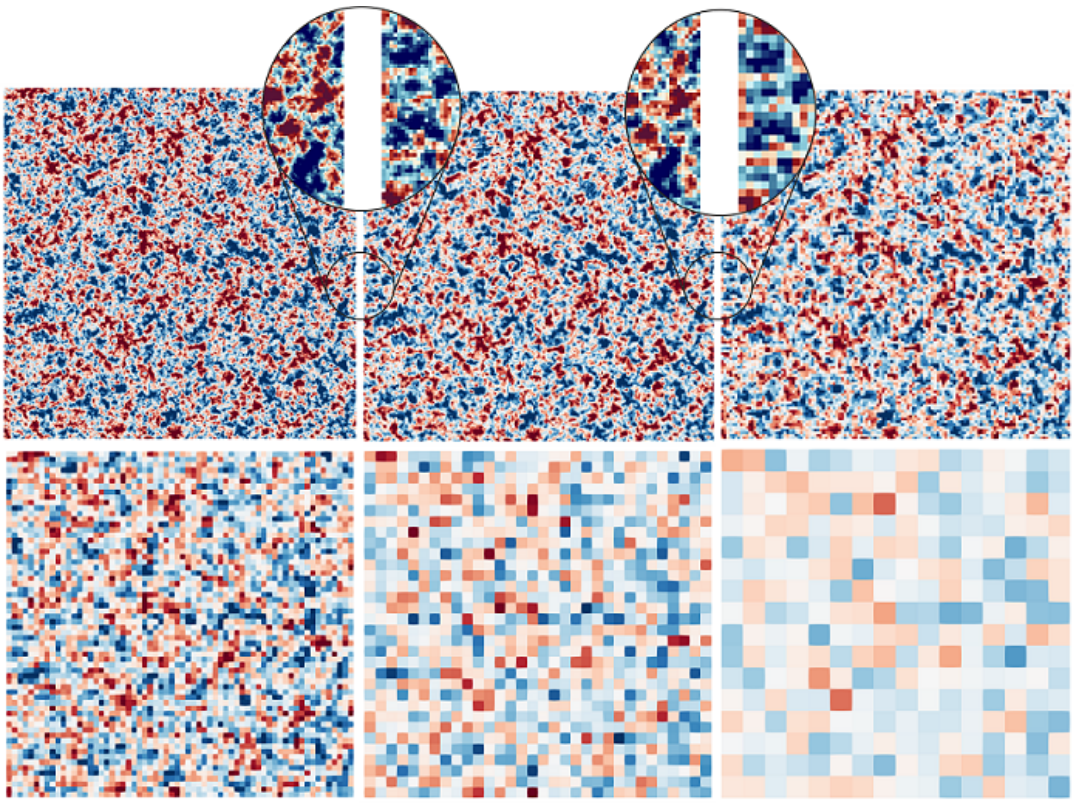

Figura 4.7: Estrategia de restricción (4.16) para el campo de conductividad en las grillas más gruesas $(h, 2h, 4h, 8h, \dots)$. Grilla desde 512×512 a 16×16.

Discretización espacial

Integrando (4.19) en la celda C y aplicando los teoremas de la divergencia y del valor medio, se obtiene:

$$\begin{aligned}\int_{V_C} \nabla\cdot\left[k(\mathbf{x})\nabla\phi(\mathbf{x})\right] dV &= \int_{\partial V_C} k(\mathbf{x})\nabla\phi(\mathbf{x})\cdot \mathbf{dS} \\ &= \sum_{f\sim nb(C)} \left(k(\mathbf{x})\nabla\phi(\mathbf{x})\right)_f \cdot \mathbf{S}_f = 0\end{aligned} \tag{4.26}$$

Como la conductividad hidráulica es variable en el dominio, una forma adecuada de evaluar $k\nabla\phi$ en el centro de una cara es usando el promedio armónico entre los valores de la celda que comparten la cara (C y F):

$$k_f = \left(\frac{1}{2}\left(\frac{1}{k_C}+\frac{1}{k_F}\right)\right)^{-1} = \left(\frac{2k_Ck_F}{k_C+k_F}\right) \tag{4.27}$$

finalmente, utilizando un esquema espacial CD para aproximar el gradiente, (4.26) se puede reescribir de manera discreta para la celda C como:

$$\sum_{F\sim NB(C)} \left(\frac{2k_Ck_F}{k_C+k_F}\right)\left(\frac{\phi_F-\phi_C}{h}\right) h = 0 \tag{4.28}$$

Al considerar todas las celdas, con los cambios correspondientes en (4.28) si alguna de las celdas vecinas se ubican fuera del dominio, se obtiene el sistema lineal de ecuaciones de la forma $\mathbf{A\Phi} = \mathbf{b}$, donde $\mathbf{A}$, $\mathbf{\Phi}$ y $\mathbf{b}$ representan respectivamente la matriz del sistema, el vector del lado derecho (RHS). En este caso la matriz del sistema es como antes, simétrica definida positiva (SPD), por lo que se aplican los mismos métodos que en la prueba 1, sin embargo el sistema está muy mal condicionado y al usar PCG se requiere un buen precondicionador para lograr la convergencia en tiempos de cómputo aceptables.

Elección de los resolvedores

En las pruebas realizadas, se observó que para varianzas altas en dominios grandes, el método CCMG como solver independiente tiene tasas de convergencia muy bajas e incluso muchas veces el método termina en divergencia. Sin embargo si se lo utiliza como precondicionador dentro de PCG se obtiene un algoritmo bastante robusto. Este hecho se debe a que unos pocos modos propios tardan en converger. Es decir, el espectro de la matriz de iteración M del algoritmo CCMG, estará muy agrupada en torno a unos pocos valores, situación que resulta favorable al utilizar el método CG debido a que pocas iteraciones serán suficientes para reducir los errores en las frecuencias que no convergen en CCMG (para más detalles sobre este problema se pueden consultar: [179] p. 202 a 204, [168] p. 278 a 280, [129]).

En base a lo anterior, para resolver el sistema lineal finalmente se utilizaron los siguientes resolvedores basados en PCG:

- PCG precondicionado con multigrilla centrado en celda (CCMG) que consiste en un ciclo V, usando 2 pre/post pasos de relajación (V$(2,2)$) de GSRB: $PCG_{\text{mgV-GS}}$.
- PCG precondicionado con un ciclo V Multigrilla Algebraico (AMG) de Nvidia AmgX, con la configuración V$(2,2)$ y L1-Jacobi Smoother, con interpolación D2: PCG_{AmgX}.
- PCG con Semicoarsing Algebraic Multigrid de HYPRE, con la misma configuración que SMG_{hypre}: PCG_{hypre}.

Donde la primera corresponde a la implementación propia, la segunda, a la implementación de Nvidia AmgX, ambas variantes basadas completamente en GPU y la última variante es basada en CPU paralelo usando MPI, implementada en la interfaz HYPRE.

Como los campos de conductividad son aleatorios, para medir los tiempos y el número de iteraciones, siempre se realiza el promedio sobre 100 pruebas en el dominio indicado con el método elegido. La figura 4.8 muestra dos campos aleatorios de conductividad y las respectivas soluciones numéricas de la ecuación (4.19) en un dominio físico de $25{,}6\lambda \times 25{,}6\lambda$ sobre una grilla computacional de 256×256.

Resultados al aumentar el tamaño del dominio

Esta prueba consiste en mantener fijo el paso de malla ($\Delta x = \Delta y = h$) y la resolución $\lambda/h = 10$, incrementando el tamaño del dominio variando desde 256×256 ($\sim$ 65 mil celdas) hasta 16384×16384 ($\sim$ 268,4 Millones de celdas), considerando una varianza $\sigma^2_{\ln K} = 9$. En todos los casos la solución se lleva a cabo hasta una reduccion relativa de 6 órdenes de magnitud en la norma L_2 del residuo (ver prueba 1).

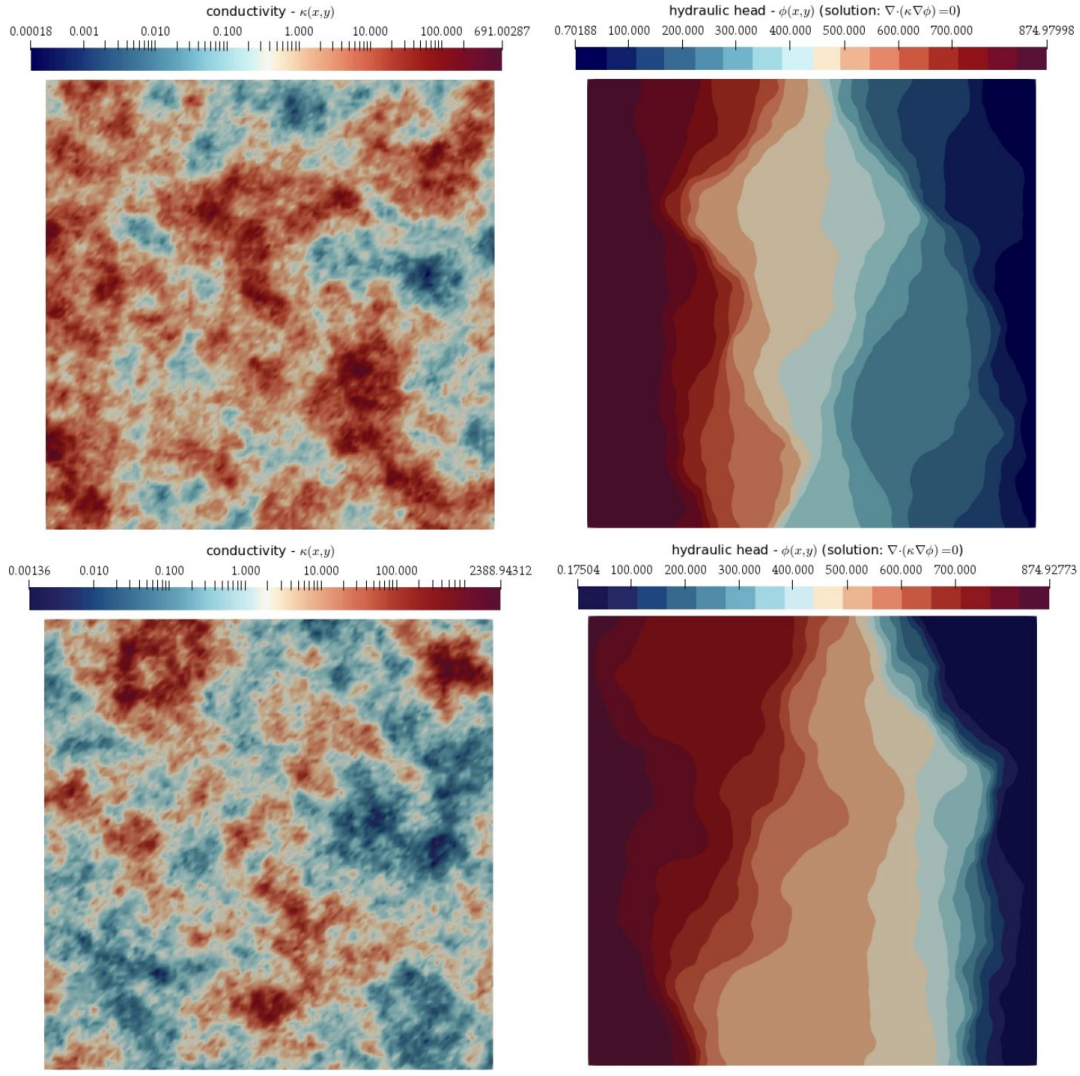

Figura 4.8: Campos de conductividad generados aleatoriamente con una varianza de $\sigma^2_{\ln K} = 6{,}25$ (izquierda) y la solución de la ecuación de flujo (derecha) para dichos campos. Grilla computacional de 256×256, dominio físico a $25{,}6\lambda \times 25{,}6\lambda$, λ: longitud de correlación.

La tabla 4.2 presenta los resultados obtenidos para los 3 resolvedores elegidos para la prueba, puede observarse que el algoritmo más robusto es el de AmgX debido al bajo número de iteraciones requeridas en cada tamaño de dominio. Mientras que los otros dos métodos mantienen la cantidad de iteraciones en el mismo orden (variaciones entre 8 % y 36 %) siendo menos robusto

la implementación de esta tesis. Sin embargo puede observarse que a partir de 1 millón de celdas, el algoritmo más eficiente en cuanto a tiempos de cálculo es la implementación de este trabajo ($PCG_{\text{mgV-GS}}$), llegando a ser hasta 2 y 20 veces más rápida que el de la librería AmgX e HYPRE respectivamente.

En la figura 4.9 se presentan los resultados de la tabla, puede observarse que al incrementar el tamaño del dominio, la tendencia de los tiempos mantiene la tendencia de crecimiento como en la prueba 1.

Tabla 4.2: Tiempos de cálculo en segundos y número de iteraciones para el problema de Poisson con coeficientes variables, fijando la varianza $\sigma^2_{\ln K} = 9$, la resolución $h/\lambda = 10$, con $h = 5$m e incrementando el tamaño del dominio desde $(25{,}6\lambda)^2$ hasta $(819{,}2\lambda)^2$. Comparación entre los diferentes resolvedores. El *speedup* se computa para el tiempo de $PCG_{\text{mgV-GS}}$.

	$\sigma_{\ln K} = 3$ ($\sigma^2_{\ln K} = 9$)							
$N_x \times N_y$	$PCG_{\text{mgV-GS}}$		PCG_{AmgX}			PCG_{hypre} (32 cpu cores)		
[Mcell]	t [s]	# iter	t [s]	# iter	speedup mgV-GS	t [s]	# iter	speedup mgV-GS
0.07	0.056	10	0.021	6	0,37×	0.091	8	1,61×
0.26	0.079	13	0.040	7	0,51×	0.189	11	2,40×
1.05	0.100	13	0.111	7	1,11×	0.655	12	6,54×
4.19	0.241	19	0.402	7	1,67×	3.308	15	13,74×
16.78	0.699	22	1.394	8	1,99×	15.31	17	21,89×
67.11	2.771	26	5.507	8	1,99×	56.27	19	20,31×

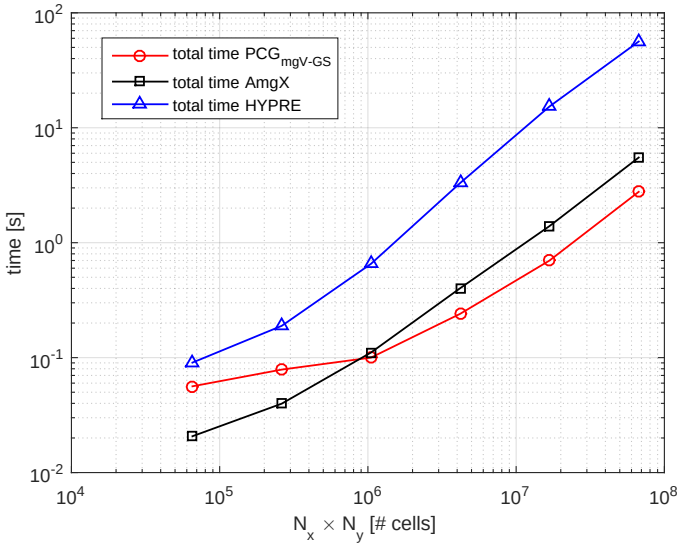

Figura 4.9: Tiempos de cálculo en segundos versus tamaño de la grilla, para la prueba de Poisson con coeficientes variables, fijando la varianza $\sigma^2_{\ln K} = 9$, la resolución $h/\lambda = 10$, con $h = 5$m e incrementando el tamaño del dominio desde $(25{,}6\lambda)^2$ hasta $(819{,}2\lambda)^2$. Comparación entre los diferentes resolvedores.

Resultados al aumentar el grado de heterogeneidad del medio

Esta prueba consiste en fijar el tamaño del problema en $819{,}2\lambda \times 819{,}2\lambda$, la resolución y el paso de malla e ir incrementando la varianza del campo de conductividades en los rangos $\sigma^2_{\ln K} \in \{1; 2{,}25; 4; 6{,}25; 9\}$.

En la tabla 4.3 se presentan los tiempos de cómputo y número de iteraciones promedio para cada grado de heterogeneidad. La figura 4.10 presenta los tiempos de cómputo versus el grado de heterogeneidad dado por la varianza de la log conductividad.

Puede observarse que para problemas con baja heterogeneidad se tiene mejor desempeño del método $PCG_{\text{mgV-GS}}$ reduciendo los tiempos de cómputo hasta 1 orden de magnitud respecto a la implementación de AmgX y hasta 45 veces más rápido que el resolvedor de HYPRE. En la medida que la heterogeneidad crece, el desempeño empeora debido al número de iteraciones requeridas, mientras que el resolvedor de AmgX muestra ser robusto manteniendo el número de iteraciones y en consecuencia los tiempos de cómputo, en un valor casi constante acorde al tamaño del problema e independiente del grado de heterogeneidad. En cuanto a la versión CPU de HYPRE, el comportamiento es similar al $PCG_{\text{mgV-GS}}$ pero es más robusto en problemas heterogéneos. No obstante, los tiempos de cómputo más bajos en todos los casos se obtienen al utilizar $PCG_{\text{mgV-GS}}$ logrando ganancias entre 2 y 10 veces respecto a la versión AmgX y factores entre 20 y 45 en comparación con la versión CPU paralela de HYPRE.

Tabla 4.3: Tiempos de cálculo en segundos y número de iteraciones para el problema de Poisson con coeficientes variables, fijando la resolución $h/\lambda = 10$, con $h = 5$m y el tamaño de malla en $(819{,}2\lambda)^2$ e incrementando la varianza $\sigma^2_{\ln K}$ desde 1 hasta 3. Comparación entre los diferentes resolvedores. El *speedup* se computa para el tiempo de $PCG_{\text{mgV-GS}}$.

$N_x = N_y = 8192$ ($\sim$ 67,1 Mcell)								
	$PCG_{\text{mgV-GS}}$		PCG_{AmgX}			PCG_{hypre} (32 cpu cores)		
$\sigma_{\ln K}$	t [s]	# iter	t [s]	# iter	speedup mgV-GS	t [s]	# iter	speedup mgV-GS
1.0	0.542	5	5.983	8	11,05×	24.362	7	44,98×
1.5	0.755	7	5.895	8	7,81×	29.112	9	38,55×
2.0	1.177	11	5.740	8	4,88×	35.890	11	30,50×
2.5	1.711	16	5.601	8	3,27×	42.474	15	24,82×
3.0	2.771	26	5.507	8	1,99×	56.273	19	20,31×

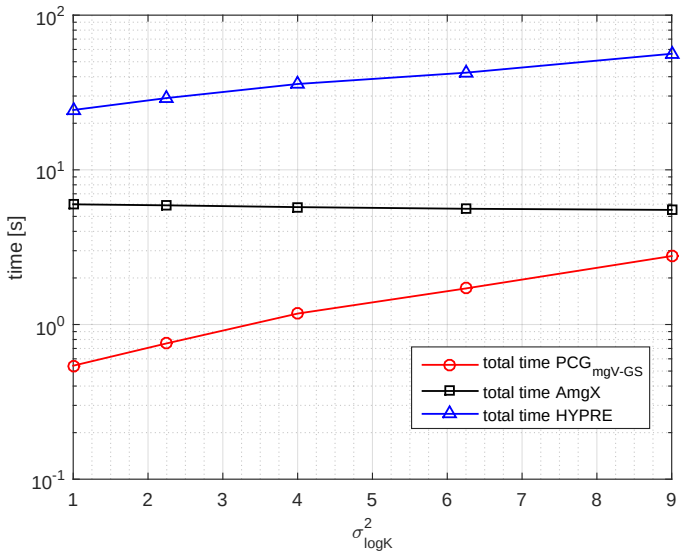

Figura 4.10: Tiempos de cálculo en segundos versus varianza de la log conductividad, para la prueba de Poisson con coeficientes variables, fijando la resolución $h/\lambda = 10$, con $h = 5$m y el tamaño del dominio en $(819{,}2\lambda)^2$. Comparación entre los diferentes resolvedores.

Principales conclusiones de la prueba 2

Los resultados muestran que aunque la implementación $PCG_{\text{mgV-GS}}$ es menos robusta para problemas heterogéneos que las otras dos versiones (AmgX e HYPRE), tiene un mejor desempeño en cuanto a tiempos de cálculo. La aceleración respecto a la versión CPU paralela indica que se

requeriría de un clúster de al menos 20 nodos de las mismas características que el equipo 2 (en cuanto a la configuración de la CPU), para igualar el resultado obtenido en una sola tarjeta.

La aceleración respecto a la versión GPU de AmgX no fue tan significativa, pero en base a la prueba 1, el requerimiento de memoria de este resolvedor es del orden de 29GB para un problema de $8192 \times 8192 \sim 67{,}1 Mcell$, mientras que con el algoritmo CCMG implementado en la tarjeta V100 (32GB de memoria) podrían resolverse problemas hasta un tamaño de $24576 \times 24576 \sim 603{,}9 Mcell$ debido al bajo consumo en memoria.

4.3. Efecto del CCMG para la PPE en FSM

Se extendió el solver CCMG para utilizarlo como precondicionador en el paso para el cálculo de corrección de la presión dentro del algoritmo FSM en 3D. Para esta prueba se ejecutó el problema de la cavidad cúbica con tapa deslizante hasta un tiempo físico de 16s en el equipo 2. El esquema espacial es CD para todos los términos de las ecuaciones, los esquemas temporales son AB y CN para los términos advectivo y difusivo respectivamente. El solver lineal utilizado tanto para las ecuaciones de momento en las tres direcciones fue CG mientras que para la PPE se utilizó CG y PCG precondicionado con CCMG.

Para evaluar la mejora en el desempeño al incorporar el método CCMG como precondicionador en el paso de la PPE, se ejecutó el problema para grillas de $N_x \times N_y \times N_z$, con $N_x = N_y = N_z$ para valores de 32, 64, 128 y 256 celdas por dirección, esto es, en grillas de 32.768 a 16.777.216 de celdas. En la tabla 4.4 se presentan el número de iteraciones acumuladas en los solver CG para las ecuaciones de momento en las tres direcciones y las acumuladas en el paso PPE a lo largo de toda la simulación así como el tiempo total de cálculo. La figura 4.11 presenta los resultados de la tabla.

Puede observarse que la ganancia obtenida se debe a una fuerte reducción en el número de iteraciones totales del solver PCG, la diferencia se acentúa para problemas más grandes, en cuanto a tiempos de cómputo se observa que para problemas pequeños no hay mejoras debido a que el costo computacional del precondicionador no es compensado por la reducción de las iteraciones, sin embargo para problemas de 256^3 se obtiene una ganancia de $13\times$. Si bien el efecto no es tan notable, debe recordarse que la cantidad de iteraciones es una función del número de celdas por dirección, esto implica que en los casos 2D, un dominio de 4096^2 ($\sim$ 16,77 Mcell) el número de iteraciones del solver es mucho mayor, si bien no se presenta dicho estudio, para las pruebas realizadas la ganancia en esos casos es de $50\times$.

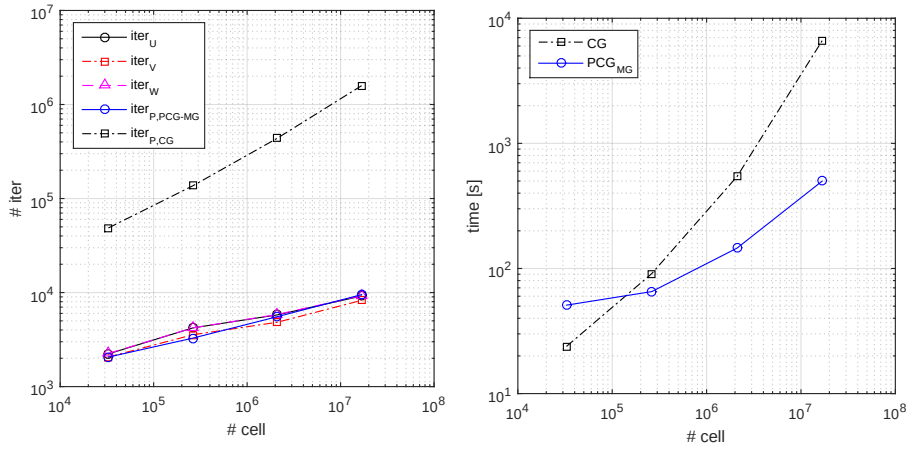

Figura 4.11: Número de iteraciones acumuladas versus numero total de celdas del dominio (izquierda) y tiempo de cómputo total versus número de celdas (derecha), al resolver el problema de la cavidad cúbica hasta un tiempo físico de simulación $T_{\text{final}} = 16$s.

Tabla 4.4: Número de iteraciones acumuladas para cada sistema lineal en el algoritmo FSM, para diferentes números de celdas por dirección, al resolver el problema de la cavidad cúbica hasta un tiempo físico de simulación de 16s utilizando el equipo 2.

$N_x = N_y = N_z$	$\#\text{iter}_{\text{U}}$	$\#\text{iter}_{\text{V}}$	$\#\text{iter}_{\text{W}}$	$\#\text{iter}_{\text{P}}$ (CG)	tiempo	$\#\text{iter}_{\text{P}}$ (PCG_{MG})	tiempo
32	2227	2063	2252	48636	23.82 s	2059	50.92 s
64	4208	3557	4212	136450	90.90 s	3275	65.33 s
128	5790	4832	5845	437044	541.75 s	5537	145.32 s
256	9187	8263	9268	1586394	6621.39 s	9508	499.26 s

Capítulo 5

Representación de geometrías complejas

En esta tesis se busca resolver problemas en geometrías que pueden ser irregulares, como por ejemplo pilas de puentes u otro tipo de objetos inmersos en una corriente líquida. Sin embargo los algoritmos implementados en GPU utilizando grillas cartesianas uniformes, no resultan flexibles ni se pueden adaptar bien a geometrías complejas.

A su vez, en los procesos de erosión y sedimentación, la frontera inferior del dominio tiene cambios a partir de las formas de fondo que se van originando y en muchos casos son estrictamente tridimensionales. Esto implicaría realizar un remallado cada vez que cambia esa frontera del dominio.

Para evolucionar la posición de la interfaz de la superficie del lecho se utiliza el método de conjuntos de nivel o LSM por sus siglas en inglés (*Level Set Method*). Esta estrategia se combina con el método de fronteras embebidas o IBM (*Immersed Boundary Method*) que consiste en utilizar una grilla que cubre el dominio completo (fluido y sólido inmerso) y la interacción del cuerpo sólido sobre la corriente líquida circundante se representa mediante la imposición de fuerzas ficticias. Estas fuerzas se agregan a la ecuación de momento como un término fuente. Esto permite obtener un algoritmo compatible con los requerimientos de la arquitectura CUDA principalmente por el uso de grillas cartesianas y el bajo requerimiento de almacenamiento en memoria a la vez que el uso de grillas cartesianas permiten el uso de solvers rápidos para la PPE.

5.1. Métodos de fronteras embebidas - IBM

El IBM fue desarrollado originalmente por Peskin [139, 140], allí se representaba una membrana elástica mediante una serie de puntos lagrangianos que pueden variar en el tiempo ubicados por encima de una grilla euleriana fija (ver figura 5.1-izquierda), la clave del método consiste en distribuir de manera conservativa sobre la grilla euleriana, las fuerzas aplicadas en los puntos lagrangianos, de esta manera, se incorpora una fuerza dentro de las ecuaciones de NS discretizadas en la grilla cartesiana uniforme. La idea es que en el proceso de discretización de las ecuaciones, no se requiere que la grilla euleriana coincida con los nodos que describen la frontera inmersa y además, con las fuerzas aplicadas en los puntos lagrangianos se logra satisfacer la condición de no slip sobre la superficie sumergida. Debe tenerse en cuenta que se plantea una fuerza aplicada a través de la frontera inmersa (fuerza singular), dicha fuerza no puede representarse fácilmente durante la discretización y debe distribuirse en la grilla euleriana con un ancho de varias celdas. En la figura 5.1-izquierda se presentan los puntos lagrangianos con las componentes de fuerza en x e y del punto i, dichas fuerzas se distribuyen en la grilla euleriana sobre un ancho de 3 celdas en cada sentido (obteniendo $f_{C,x}$, $f_{F,x}$, $F \sim \{NW, W, SW, N, S, NE, E, SE\}$ y el análogo para las componentes y). Se piensa que al realizar la distribución de la fuerza en la grilla euleriana la representación precisa de la frontera se suaviza, lo que es indeseable en flujos a altos Reynolds.

Posteriormente se ha extendido el método para cuerpos rígidos, sin embargo esto planetó algunos desafíos ya que las fuerzas cerca del límite sólido agregan problemas de inestabilidad por ser singulares sobre la frontera inmersa. Usualmente este método se conoce como enfoque de forzado continuo [117].

Otro método, conocido como enfoque de forzado discreto, fue desarrollado posteriormente, por ejemplo [54, 64, 118, 177], en este caso la idea es modificar el operador discreto en las celdas fluidas más cerca del sólido para tener en cuenta la frontera inmersa (ver figura 5.1-centro). Generalmente este tipo de métodos producen una mejor precisión espacial si se incorporan esquemas de discretización adecuados al desarrollar los operadores. De acuerdo a como se imponga la condición de contorno sobre la frontera embebida, este enfoque a su vez se clasifica en forzamiento directo o indirecto [117]. En el primero se establecen directamente las condiciones de borde (BCs) sobre las variables primitivas, $\mathbf{u}$ y p (ver por ejemplo [54, 118]), mientras que en el segundo se impone una fuerza generada a partir de la ecuación de momento, la cual contiene indirectamente las variables primitivas (ver por ejemplo [16, 66]).

Dentro del segundo enfoque se han implementados diferentes variantes para prescribir directamente los valores de las variables (forzamiento directo), una primer estrategia es calcular las velocidades en los nodos fluidos, adyacentes al sólido inmerso mediante interpolación entre los valores fluidos hacia un lado y el valor impuesto en la frontera del sólido hacia el otro (por ejemplo velocidad nula en el caso no-slip, ver figura 5.1-centro), este enfoque se puede formular con una aproximación de segundo orden, pero la interpolación se desarrolla por direcciones como el método FDM por lo que hay cierta ambigüedad en la elección de la dirección en la cual se realiza la interpolación. Una de las primeras referencias de este método es [54] aunque este método se ha implementado en GPU por Krishnan [87].

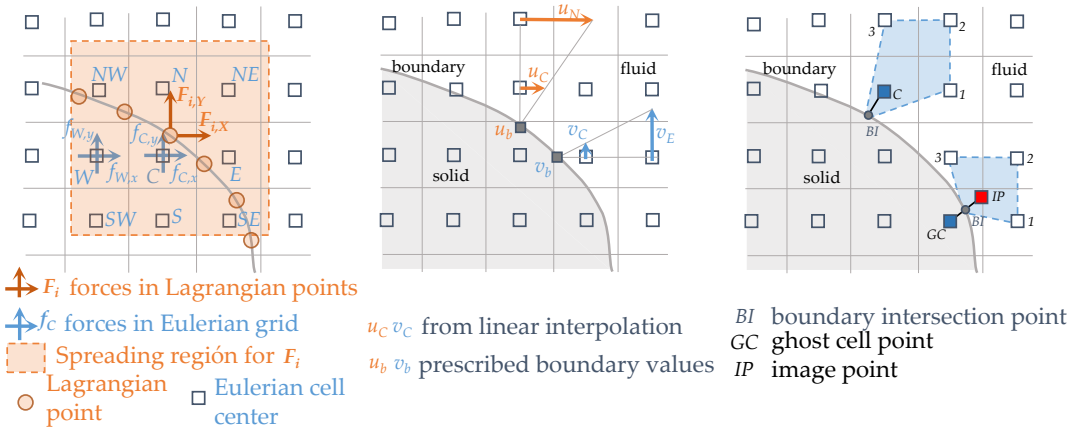

Figura 5.1: (izquierda) Representación esquemática del método de forzado continuo, aquí la fuerza aplicada en el nodo lagrangiano i, $F_{i,X}$ y $F_{i,Y}$ se distribuyen utilizando una función tipo δ de Dirac discreta de ancho $3h$ (área sombreada), en primer lugar se evalúa la fuerza discreta en los nodos lagrangianos, luego estos valores se multilpican por la función δ y se integran en cada celda euleriana para obtener las fuerzas $f_{C,x}$, $f_{C,y}$ del nodo C y sus vecinos donde se resuelven las ecuaciones de NS. (centro) esquema de interpolación del método de forzado directo propuesto por [54] pero en grilla colocada, la elección de la componente u y v fue arbitraria para ejemplificar. (derecha) Esquema para el método de forzamiento directo utilizando la direccion normal la frontera inmersa mediante interpolación bilineal de los nodos más cercanos y el valor prescripto en el borde, en el primer caso, el valor u_C del nodo C se obtiene por interpolación, en el segundo caso, el valor u_{IP} se obtiene por interpolación y posteriormente se extrapola el valor u_{GC} usando extrapolación lineal $u_{GC} += 2u_{BI} - u_{IP}$.

Para evitar la ambiguedad en la elección de la dirección de interpolación, hay métodos que utilizan directamente la dirección normal al la frontera inmersa (hacia el interior del dominio fluido), por ejemplo [16, 76, 84].

Madjumar et. al. [107] proponen combinar esta última estrategia con la técnica de nodos fantasma, que se conoce métodos GCM por sus siglas en inglés (*ghost cell method*). Este método da buenos resultados y ha sido ampliamente utilizado desde entonces hasta la actualidad [13, 34, 43,

141, 169], incluidos problemas de transporte de sedimentos [86].

La figura 5.1-derecha presenta la estrategia de forzamiento directo utilizando la dirección normal a la frontera, que puede realizarse por interpolación involucrando únicamente valores del campo fluido y el valor impuesto en la frontera, o bien mediante el uso de nodos ficticios (nodos fantasma) ubicados dentro el sólido donde se impone el valor utilizando los puntos IP y BI (punto imagen:interpolado e intersección con la frontera: prescripto, respectivamente).

En esta tesis se realiza una implementación IBM utilizando el FSM con nodos fantasma, inspirada en el trabajo de Mittal et. al. [116], con una serie de modificaciones para facilitar la implementación del código y optimizar el desempeño en GPU. Las principales diferencias respecto al método original [116], son:

- Aquí se utilizan esquemas TVD en lugar de un esquema CD para la advección.
- El esquema de interpolación para obtener el valor de la variable en el Punto Imagen (IP) se realiza utilizando una celda *master* con dominio en $[0, h] \times [0, h] \times [0, h]$ mediante un mapeo trilineal como se explica en las subsecciones posteriores. Esto mejora el rendimiento y requerimientos de memoria en GPU.
- El demarcado de las celdas fantasma (GC), celdas sólidas (SC), celdas fluidas (FC), punto de intersección con la frontera (BI) y punto imagen (IP), así como el cómputo de las normales y distancias respectivas se realiza utilizando la función distancia con signo (SDF), es decir métodos Level Set (LS).
- Las ecuaciones de momento se resuelven utilizando BICGStab y la de la presión se resuelve usando PCG con un preacondicionador CCMG.

5.1.1. Formulación del método de fronteras embebidas

A continuación se presenta la formulación del método de fronteras embebidas utilizando la técnica de nodos fantasma (GC-IBM). Se parte de una representación LS mediante la SDF del objeto inmerso, este campo $\phi(\mathbf{x}, t)$ viene dado por:

$$\begin{cases} \phi(\mathbf{x}, t) > 0 & \text{si} \quad \mathbf{x} \in (\Omega_f \setminus \partial\Omega_s) \\ \phi(\mathbf{x}, t) = 0 & \text{si} \quad \mathbf{x} \in \partial\Omega_s \\ \phi(\mathbf{x}, t) < 0 & \text{si} \quad \mathbf{x} \in (\Omega_s \setminus \partial\Omega_s) \end{cases} \tag{5.1}$$

donde Ω_f, Ω_s y $\partial\Omega_s$ son las regiones ocupadas por el fluido, el sólido y la frontera o interfaz que las separa respectivamente.

De esta manera, evaluando el signo de la función SDF ($\text{sgn}(\phi)$) se pueden clasificar los centros de celda C según su ubicación en todo el dominio, en *celdas fluidas* ($\phi_C > 0$) y *celdas sólidas* ($\phi_C \leq 0$). Con ello es posible definir las *celdas fantasma* como aquellas en las que $\phi_C \leq 0$ y $\phi_F > 0$ para algún vecino F de los 4 posibles ($F \sim \{E, W, N, S\}$ en grilla cartesiana).

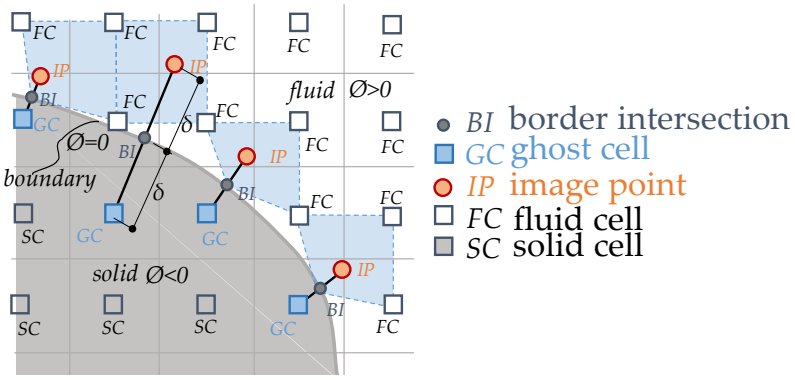

Figura 5.2: Esquema de trabajo para el metodo IBM implementado. La frontera separa los centros de celda fluidas (FC, $\phi > 0$), los centros de celda sólidas (SC, $\phi \leq 0$), los centros de celda fantasma (GC), los puntos de intersección del segmento normal al borde con el mismo (BI) y los puntos imagen (IP) utilizados en la interpolación.

En la figura 5.2 se muestran todos los tipos de nodos de la clasificación que se usan en el método para el caso 2D por simplicidad. Luego de identificar los nodos GC se considera el segmento normal al borde que pasa por GC y su intersección con el borde (punto BI), con ello se define el punto imagen (IP) que se ubica a una distancia igual a $\delta = ||\mathbf{x}_{GC} - \mathbf{x}_{BI}||$ utilizando la misma normal, con ello $||\mathbf{x}_{IP} - \mathbf{x}_{BI}|| = \delta$.

Utilizando los métodos LS, la normal en el punto BI se puede computar mediante la función marcadora:

$$\mathbf{n}_{BI} = \left(\frac{\nabla\phi}{||\nabla\phi||}\right)_{BI} \approx \left(\frac{\nabla\phi}{||\nabla\phi||}\right)_{GC} \tag{5.2}$$

por otro lado $\delta = ||\mathbf{x}_{GC} - \mathbf{x}_{BI}|| = |\phi(\mathbf{x}_{GC})| = |\phi_{GC}|$ y las coordenadas del punto imagen se obtienen como:

$$\mathbf{x}_{IP} = \mathbf{x}_{GC} + 2\delta\mathbf{n}_{BI} = \mathbf{x}_{GC} - 2\phi_{GC}\mathbf{n}_{BI} \tag{5.3}$$

donde se tiene en cuenta que $\phi_{GC} \leq 0$.

Una vez identificados las coordenadas de los puntos imagen, se interpola el valor funcional a partir de los valores vecinos más cercanos utilizando interpolación bilineal (trilineal en 3D), por ejemplo si la variable a resolver es la componente x de la velocidad se obtiene como:

$$u(x, y) = C_1xy + C_2x + C_3y + C_4 \tag{5.4}$$

donde los coeficientes C_i, $i = 1 \ldots 4$ ($i = 1 \ldots 8$ en 3D) se calculan en términos de los 4 (8 en 3D) valores vecinos, es decir utilizando:

$$u(x_i, y_i) = u_i = C_1x_iy_i + C_2x_i + C_3y_i + C_4, \text{con } i = 1, 2, 3, 4 \tag{5.5}$$

la idea es determinar previamente los coeficientes de la interpolación para luego computar el valor interpolado como:

$$\begin{aligned} u(x_{IP}, y_{IP}) = u_{IP} &= C_1x_{IP}y_{IP} + C_2x_{IP} + C_3y_{IP} + C_4 \\ \Leftrightarrow u_{IP} &= \sum_{i=1}^{4} \xi_i u_i + O(\delta^2) \end{aligned} \tag{5.6}$$

aquí el término $O(\delta^2)$ representa el error de truncamiento de la aproximación bilineal (trilineal) (ver [116]). Con esto, la ecuación para la celda GC se calcula teniendo en cuenta ese valor, la condición de borde a aplicar y el valor en la frontera. Para el caso de BC tipo Dirichlet, la extrapolación lineal a utilizar es:

$$u_{BI} = \left(\frac{u_{IP} + u_{GC}}{2}\right) \Leftrightarrow u_{GC} = 2u_{BI} - u_{IP} \tag{5.7}$$

Para el caso de BC tipo Neumann, la extrapolación es:

$$\left(\frac{\partial u}{\partial \mathbf{n}}\right)_{BI} = \frac{u_{IP} - u_{GC}}{2\delta} \Leftrightarrow u_{GC} = u_{IP} - 2\delta \left(\frac{\partial u}{\partial \mathbf{n}}\right)_{BI} \tag{5.8}$$

Finalmente las expresiones (5.7) o (5.8) se utilizan como ecuaciones para imponer implícitamente los valores en las celdas GC.

Dichas ecuaciones para GC en conjunto con las ecuaciones triviales $u_{SC} = 0$ para el resto de celdas sólidas (ubicadas dentro del sólido inmerso) y las ecuaciones que surgen de aplicar el FVM para las celdas fluidas FC, conforman un sistema de ecuaciones en el dominio completo (sólido incluido), los valores GC permitirán de este modo que se satisfaga la condición de borde en la frontera embebida.

La figura 5.3 muestra el proceso de identificación de las celdas fantasma GC los correspondientes puntos imagen en el fluido, puede verse que al refinar la grilla, el soporte de celdas GC aumenta.

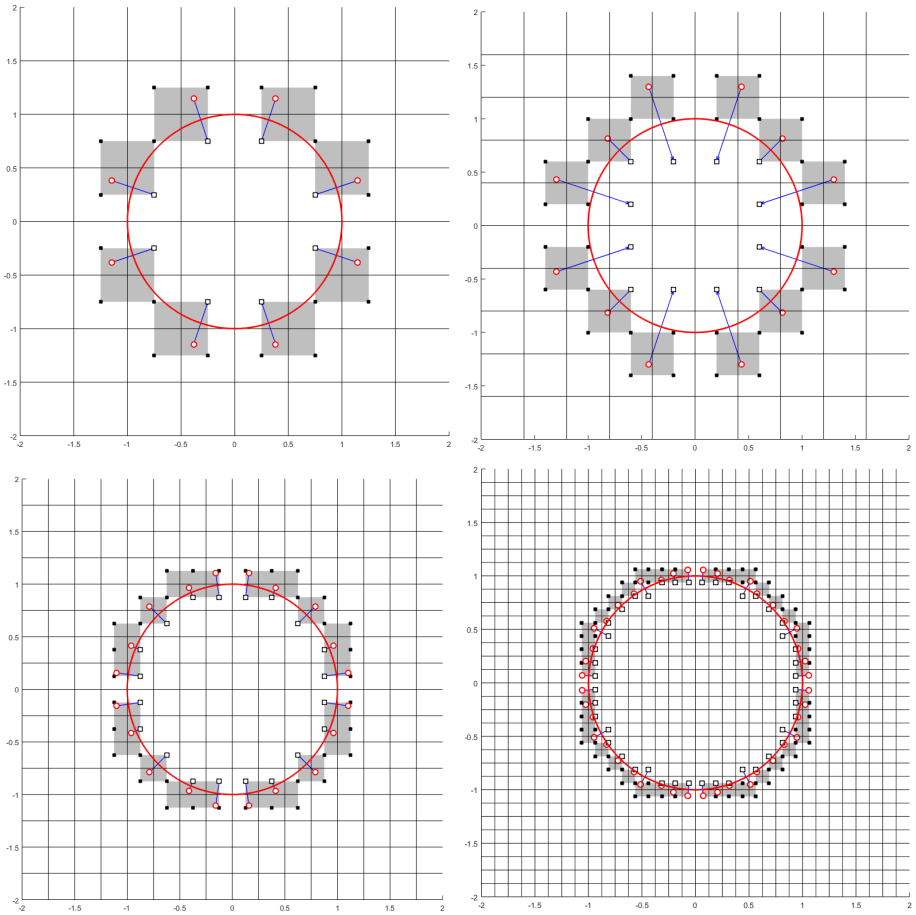

Figura 5.3: Identificación de los GC en el sólido y su punto imagen IP en el fluido ubicado a la misma distancia desde borde, para diferentes niveles de resolución de malla ($N_x = 8, 10, 16, 32$).

En la figura la zona sombreada indica los centros de celda que rodean al punto IP. Puede verse entonces que al calcular los coeficientes de la interpolación (ecuación (5.5)), en muchos casos la propia celda GC formaría parte de la información utilizada para calcular u_{IP} quedando un sistema mal puesto, esto se remedia utilizando la ecuación 5.5 con el dato prescrito en el punto BI, para BC tipo Dirichlet:

$$u_{BI} = C_1 x_{BI} y_{BI} + C_2 x_{BI} + C_3 y_{BI} + C_4 \tag{5.9}$$

o su correspondiente derivada normal para BC tipo Neumann:

$$\left(\frac{\partial u}{\partial \mathbf{n}}\right)_{BI} = \nabla u \cdot \mathbf{n}_{BI} \Leftrightarrow \left(\frac{\partial u}{\partial \mathbf{n}}\right)_{BI} = C_1\left(y_{BI}n_x + x_{BI}n_y\right) + C_2 n_x + C_3 n_y \tag{5.10}$$

donde $\mathbf{n}_{BI} = (n_x, n_y)$.

La expresión para el punto imagen, (5.6) en el caso que IP esté completamente rodeado de nodos fluidos se puede simplificar bastante al usar una grilla uniforme (ver figura 5.4):

$$\begin{gathered} C_1 = (u_1 - u_2 + u_3 - u_4)/h^2 \;\; ; \;\; C_2 = -C_1 y_1 - (u_1 - u_2)/h; \\ C_3 = -C_1 x_1 - (u_1 - u_4)/h \;\; ; \;\; C_4 = u_1 - C_1 x_1 y_1 - C_2 x_1 - C_3 y_1; \\ u_{IP} = x_{IP} y_{IP} C_1 + x_{IP} C_2 + y_{IP} C_3 + C_4; \end{gathered} \tag{5.11}$$

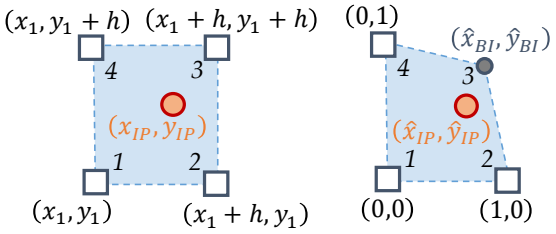

Figura 5.4: Valores utilizados para realizar la interpolación bilineal (caso 2D) en el punto imagen (IP) para el caso en que 1, 2, 3 y 4 sean celdas fluidas (FC) y cuando se utiliza como dato el punto intersección con el borde (BI) trabajando en coordenadas reducidas.

Para el caso en que se deba utilizar el punto de intersección BI en la interpolación conviene utilizar las variables en forma reducida (ver figura 5.4-derecha):

$$\hat{x} = \frac{x - x_1}{h} \;\; ; \;\; \hat{y} = \frac{y - y_1}{h} \tag{5.12}$$

con ello, en el caso de BC tipo Dirichlet, la expresión se simplifica a:

$$\begin{gathered} u_{IP} = \hat{x}_{IP}\hat{y}_{IP}C_1 + \hat{x}_{IP}(u_2 - u_1) + \hat{y}_{IP}(u_4 - u_1) + u_1; \\ \text{con } C_1 = \frac{u_3 + u_1(\hat{x}_{BI} + \hat{y}_{BI} - 1)}{\hat{x}_{BI}\hat{y}_{BI}} - \frac{u_2}{\hat{y}_{BI}} - \frac{u_4}{\hat{x}_{BI}}; \end{gathered} \tag{5.13}$$

mientras que para el caso 3D la expresión resulta en:

$$\begin{gathered} u_{IP} = C_1\hat{x}_{IP}\hat{y}_{IP}\hat{z}_{IP} + (u_1 - u_2 + u_3 - u_4)\hat{x}_{IP}\hat{y}_{IP} + \\ (u_1 - u_2 - u_5 + u_6)\hat{x}_{IP}\hat{z}_{IP} + (u_1 - u_4 - u_5 + u_8)\hat{y}_{IP}\hat{z}_{IP} \\ +(u_2 - u_1)\hat{x}_{IP} + (u_4 - u_1)\hat{y}_{IP} + (u_5 - u_1)\hat{z}_{IP} + u_1; \\ \text{con } C_1 = \frac{u_7}{\hat{x}_{BI}\hat{y}_{BI}\hat{z}_{BI}} - \frac{u_6}{\hat{y}_{BI}} - \frac{u_3}{\hat{z}_{BI}} - \frac{u_8}{\hat{x}_{BI}} \\ + \frac{u_5(\hat{x}_{BI} + \hat{y}_{BI} - 1)}{\hat{x}_{BI}\hat{y}_{BI}} + \frac{u_4(\hat{x}_{BI} + \hat{z}_{BI} - 1)}{\hat{x}_{BI}\hat{z}_{BI}} + \frac{u_2(\hat{y}_{BI} + \hat{z}_{BI} - 1)}{\hat{y}_{BI}\hat{z}_{BI}} \\ - \frac{u_1((\hat{x}_{BI} - 1)(\hat{y}_{BI} + \hat{z}_{BI} - 1) + \hat{y}_{BI}\hat{z}_{BI})}{\hat{x}_{BI}\hat{y}_{BI}\hat{z}_{BI}}; \end{gathered} \tag{5.14}$$

En el caso de BC tipo Neumann, las expresiones deben modificarse teniendo en cuenta la ecuación (5.10).

5.1.2. Detalles de la implementación GPU

Para la implementación del IBM en CUDA se utilizó como base el código del FSM desarrollado previamente, donde se incorporaron las siguientes modificaciones. Se evalúa el signo de la función marcadora LS (ϕ) para clasificar las celdas durante el computo del stencil. En particular en la operación tipo stencil de la ecuación de momento, cada hilo clasifica si se trata de una celda FC, SC o GC y en base a ello utiliza las siguientes operaciones de la fila C de la matriz del sistema:

1. la ecuación (2.69) si $\phi_C > 0$
2. la ecuación $u_C = 0$ si $\phi_C \leq 0$ y $\phi_F \leq 0$ ($F \sim \{N, S, E, W\}$ o $F \sim \{N, S, E, W, T, B\}$)
3. si $\phi_C \leq 0$ y $\phi_F > 0$ para algún F, entonces según la BC a aplicar sobre la frontera inmersa se utilizan:
   - la ecuación $u_C + u_{IP} = 2u_{BI}$, utilizando la ecuación (5.13) ((5.14) en 3D) para evaluar u_{IP} (BC tipo Dirichlet).
   - la ecuación $u_C - u_{IP} = -2\delta \left(\frac{\partial u}{\partial \mathbf{n}}\right)_{BI}$, utilizando una versión modificada de (5.13) o (5.14) para evaluar u_{IP} en el caso de BC tipo Neumann.

en cuanto al RHS de la ecuación de momento, se modifica solamente si el hilo esta procesando celdas SC, GC. El sistema se resuelve utilizando el método BiCGStab, debido a la leve pérdida de simetría de la ecuación.

En cuánto a la solución de la ecuación para la presión, se evaluaron diferentes variantes del IBM:

1. Resolver utilizando BiCGStab, aplicando la BC tipo Neumann en las celdas fantasma GC, para ello se modificó solamente el stencil de la presión como en la ecuación de momento.
2. Resolver utilizando BiCGStab con un preacondicionador basado en CCMG con las modificaciones propuestas en el trabajo de Udaykumar [170] y la variante más actual presentada por Botto [24].
3. Resolver utilizando BiCGStab con una versión modificada del preacondicionador CCMG en la cual se utiliza un ciclo V sin tener en cuenta el sólido en las grillas gruesas y previamente antes de salir del preacondicionador se realiza un paso donde se rellenan los valores en las celdas GC de forma explícita teniendo en cuenta que $u_{GC} = u_{IP}$ (por ej. para el caso de BC tipo Neumann homogénea para la presión).
4. Resolver la ecuacion de Poisson utilizando PCG con el preacondicionador CCMG, sin aplicar explícitamente las condiciones de borde en la presión y dejando que el campo de presiones se ajuste de acuerdo al campo de velocidades del paso predictor.

Las alternativas 1, 2 y 3 proveen prácticamente la misma solución del sistema lineal, aunque el mejor desempeño se tiene en la alternativa 3. Sin embargo, teniendo en cuenta el modelo de ejecución de la GPU, los mejores rendimientos se obtuvieron con la última alternativa (4).

El hecho de no aplicar explícitamente condiciones de contorno sobre el sólido al resolver la ecuación de la presión es algo frecuentemente discutido en las referencias. Es de esperar que al no aplicar las BC en la PPE, en el paso de proyección, se modifique la BC Dirichlet en el campo de velocidad, sin embargo, en las pruebas se comprobó que los cambios en el campo de velocidad para las celdas próximas al borde son muy pequeños (en el orden $O(10^{-4})$) al compararlos con los valores de velocidad entonces resulta válido despreciar tales cambios.

Entre los trabajos que resuelven la presión en todo el dominio sin considerar el sólido se destacan los siguientes:

- En [54] Fadlun et. al. argumentan que al aplicar la condición no-slip, las velocidades normal y tangencial deben anularse $\mathbf{u}_n = \mathbf{u}_\tau = \mathbf{0}$, luego considerando la ecuación de momentum expresada en dichas componentes, se deduce que se está considerando implícitamente $\partial p/\partial \mathbf{n} = 0$ ([54], págs. 58-59).

- En [131] Pan resuelven la PPE sin distinguir tipos de celda, ya que considera que como dentro del cuerpo inmerso existe una distribución de velocidades incompresible prescrita, entonces no hay cambios en las propiedades a través de la superficie del cuerpo y entonces la presión dentro del cuerpo se rige por la misma ecuación que está afuera. A su vez, la naturaleza elíptica de la PPE asegura que la presión dentro del cuerpo se ajustará según el campo de presión fuera de él. ([131], p. 283).

- Deen et. al. [45] y posteriormente Maitri et. al. [106] mencionan explícitamente que no realizan un tratamiento especial al resolver la PPE en las celdas próximas al la región sólida.

- En un trabajo más actual, Chi et. al. [34] realizan una discusión sobre el flujo de masa no físico que se genera a través de la superficie del cuerpo en el caso de imponer una BC tipo Neumann en la PPE cerca de la frontera sólida como se realiza en el trabajo de Mittal et. al. [116] ([34], págs. 15-17).

En las simulaciones se utilizará el método IBM con la variante 4 para la PPE, no obstante en caso de que se desee resolver la PPE aplicando BC tipo Neumann en la superficie del cuerpo inmerso, se pueden aplicar algunas de las demás variantes. Aquí cabe mencionar que, si bien para BC tipo Dirichlet el método tiene una aproximación de segundo orden, al aplicar la BC tipo Neumann con el método IBM propuesto, se tendrá una aproximación de primer orden en la frontera, sin embargo, como el gradiente de presión en las ecuaciones está multiplicado por Δt y la restriccion CFL en el esquema temporal AB para el término advectivo implica que $O(\Delta t) \sim O(h)$, entonces la precisión de segundo orden del FSM al combinar con esta estrategia GC-IBM no se ve afectada [116].

Por lo antedicho, no se requiere una implementación del IBM a segundo orden para las BC tipo Neumann para el caso de la PPE. No obstante, en el caso de una ecuación de transporte escalar para otro tipo de problemas, se puede realizar una extrapolación de segundo orden aplicando la estrategia propuesta por Pan en [132], la cual mantiene un stencil relativamente compacto, haciéndolo más compatible con la arquitectura CUDA.

En las siguientes subsecciones se realizan algunas pruebas de validación del método propuesto.

5.1.3. Convergencia en malla - Ecuación de difusión transiente

En esta prueba se resuelve la siguiente ecuación de difusión transiente:

$$\frac{\partial \phi}{\partial t} = \nabla \cdot (\nabla \phi) \tag{5.15}$$

en un dominio cuadrado $\Omega = [0,1] \times [0,1]$ donde se incluye una frontera inmersa dada por la siguiente expresión $\Gamma = \left\{(x-1/2)^2 + (y-1/2)^2 = 1/4\right\}$, para dos condiciones de borde tipo Dirichlet en las fronteras exterior e interior del dominio:

(A) $\phi = 0$ sobre la frontera exterior del cuadrado unitario ($\partial\Omega$) y $\phi = 1$ sobre la frontera inmersa (circunferencia interior) Γ y dentro de ella, considerando $t \geq 0$.

(B) $\phi(x) = 16x^2(x-1)^2$ sobre los bordes horizontales (borde norte y sur del cuadrado), $\phi = 0$ sobre los bordes verticales (borde oeste y este del cuadrado) y $\phi = 0$ sobre la frontera inmersa Γ y dentro de ella.

La ecuación se resuelve utilizando el FVM con un esquema espacial centrado, con el stencil modificado en las celdas del borde del cuadrado para conservar el segundo orden. El esquema temporal utilizado es FE con un paso de tiempo Δt calculado de acuerdo al número de Fourier, de esta forma el error temporal se mantiene por debajo del error que proporciona el esquema espacial, además se tuvo en cuenta que sea un múltiplo de tiempo final T de la simulación para poder calcular el error para los diferentes tamaños de malla en el mismo instante. La condición inicial es $\phi(\mathbf{x}, 0) = 0$. El problema se hace evolucionar hasta un tiempo físico de $T = 0{,}2$s. La figura 5.5 muestra las soluciones numéricas de ambas pruebas utilizando una malla de $N_x = N_y = 128$.

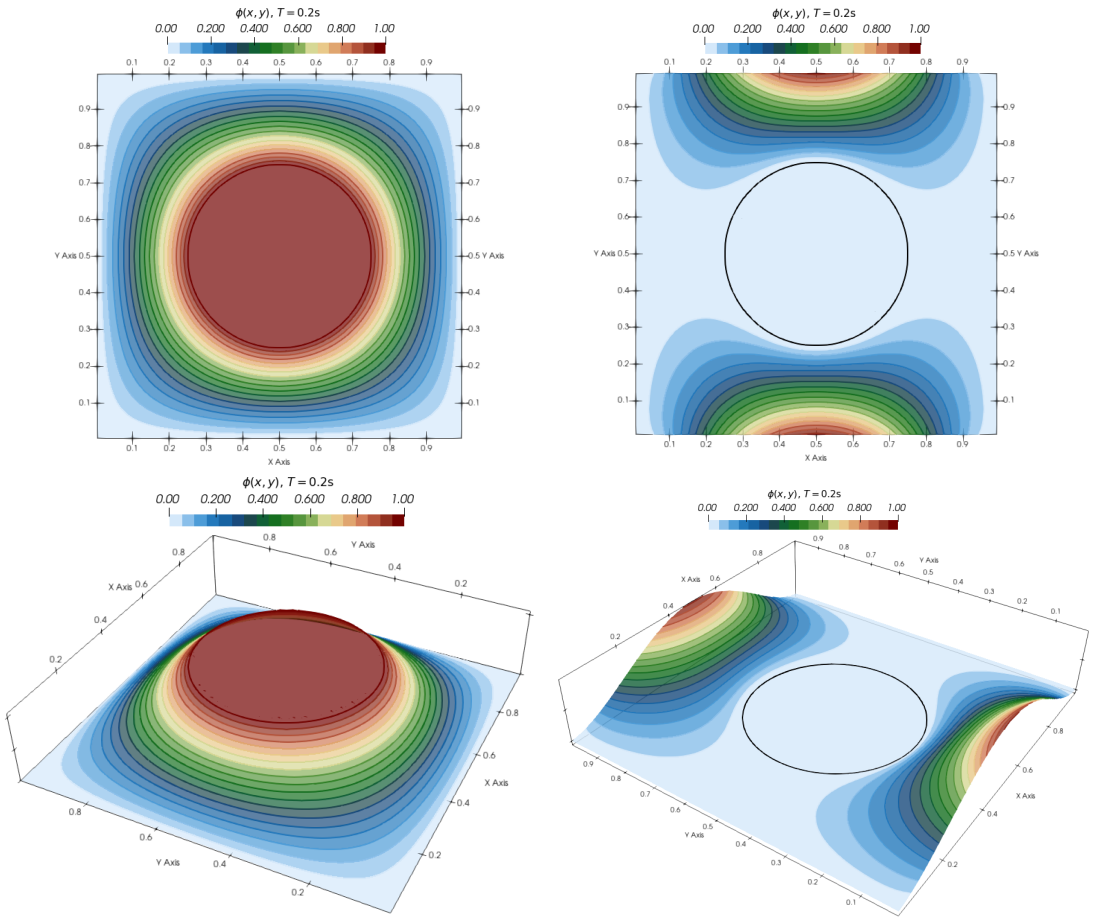

Figura 5.5: Solución numérica del problema A (izquierda) y problema B (derecha), utilizando la grilla de 128×128. Por claridad, en la solución B se incorporó la curva de nivel 0 indicada en color negro.

Para calcular el error se utiliza la solución numérica sobre una malla fina de 1024×1024 celdas. Como en el esquema colocado, los centros de celda de las grillas gruesas no coinciden con los de la grilla fina, se realiza una interpolación usando spline cúbico de la solución fina en cada grilla y en el cómputo del error se excluyen los nodos interiores al cuerpo. Para la prueba se utilizaron grillas de 32, 64, 128, 256 y 512 celdas por dirección. En la figura 5.6 se presenta el error en las tres normas usuales (L_1, L_2 y L_∞):

$$||\epsilon||_1 = \sum_{i=1}^{M} |\phi_{i,\text{num}} - \phi_{i,\text{fina}}| \tag{5.16}$$

$$||\epsilon||_2 = \left(\sum_{i=1}^{M} \frac{(\phi_{i,\text{num}} - \phi_{i,\text{fina}})^2}{M} \right)^{1/2} \tag{5.17}$$

$$||\epsilon||_\infty = \max_{i=1}^{M} |\phi_{i,\text{num}} - \phi_{i,\text{fina}}| \tag{5.18}$$

donde M es el número de celdas luego de excluir las celdas ubicadas dentro del cuerpo inmerso. Se incluyen las rectas con pendiente de primer y segundo orden para comparación ($O(h)$ y $O(h^2)$). Puede observarse que las normas del error tienen una convergencia próxima a segundo orden.

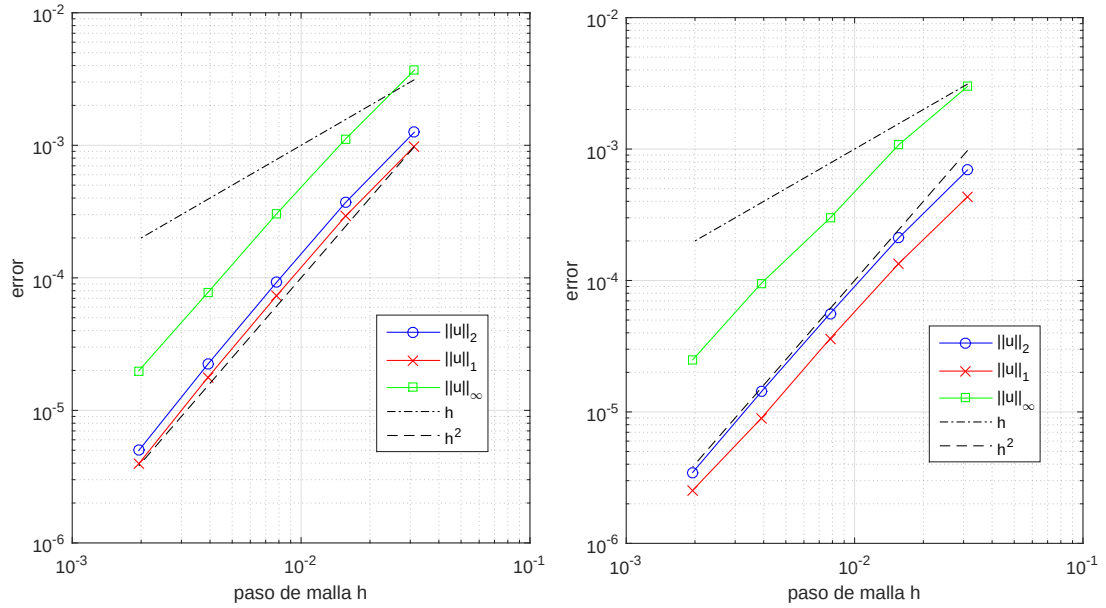

Figura 5.6: Normas L_1, L_2 y L_∞ del error versus paso de malla h.

5.1.4. Convergencia en malla - Flujo en una cavidad cuadrada con cilindro embebido

En esta prueba se resuelve el problema del flujo en una cavidad con la tapa en movimiento *lid-driven cavity flow* con un cilindro fijo inmerso en 2D. Como en el problema clásico, la componente x de la velocidad en la tapa superior tiene valor constante, en general $u = 1$m/s, mientras que en las paredes verticales $u = 0$m/s, es decir hay una discontinuidad de salto en las BC, por ello, aquí se realiza el estudio de convergencia en una variante regularizada, para ello, se utiliza la siguiente condición de borde para la tapa:

$$\mathbf{u}_{\text{top}} = (u_{\text{top}}, v_{\text{top}}) = (16x^2(x-1)^2, 0)\text{m/s}; \quad x \in [0,1] \tag{5.19}$$

Las BCs para el resto de lados es $\mathbf{u} = (0,0)$ incluyendo la frontera inmersa que es un cilindro de diámetro $0{,}5$m. La cavidad tiene $L = 1$m de lado, los parámetros físicos utilizados para la prueba son: $\rho = 100\text{kg}/m^3$ (densidad), $\mu = 1$kg/m/s (viscosidad dinámica), $\nu = \mu/\rho = 0{,}01\text{m}^2/\text{s}$ (viscosidad cinemática), el número de Reynolds basado en la velocidad máxima es $Re = 100$. La solución de referencia para calcular los errores es la correspondiente a una grilla de 1024×1024. El esquema temporal de discretización es la combinación de AB para el término advectivo y CN para el término difusivo, el esquema espacial es CD, el paso de tiempo elegido es $\Delta t = 5 \times 10^{-4}$s. El tiempo físico simulado es $T = 500\Delta t = 0{,}25$s.

La figura 5.7 presenta la solución numérica obtenida para la grilla de 128×128, en la gráfica se presentan el campo de velocidad, (u, v) (arriba) con la representación de las trayectorias (abajo), se presenta además el campo de vorticidad (ω_z) en una escala de colores arbitraria.

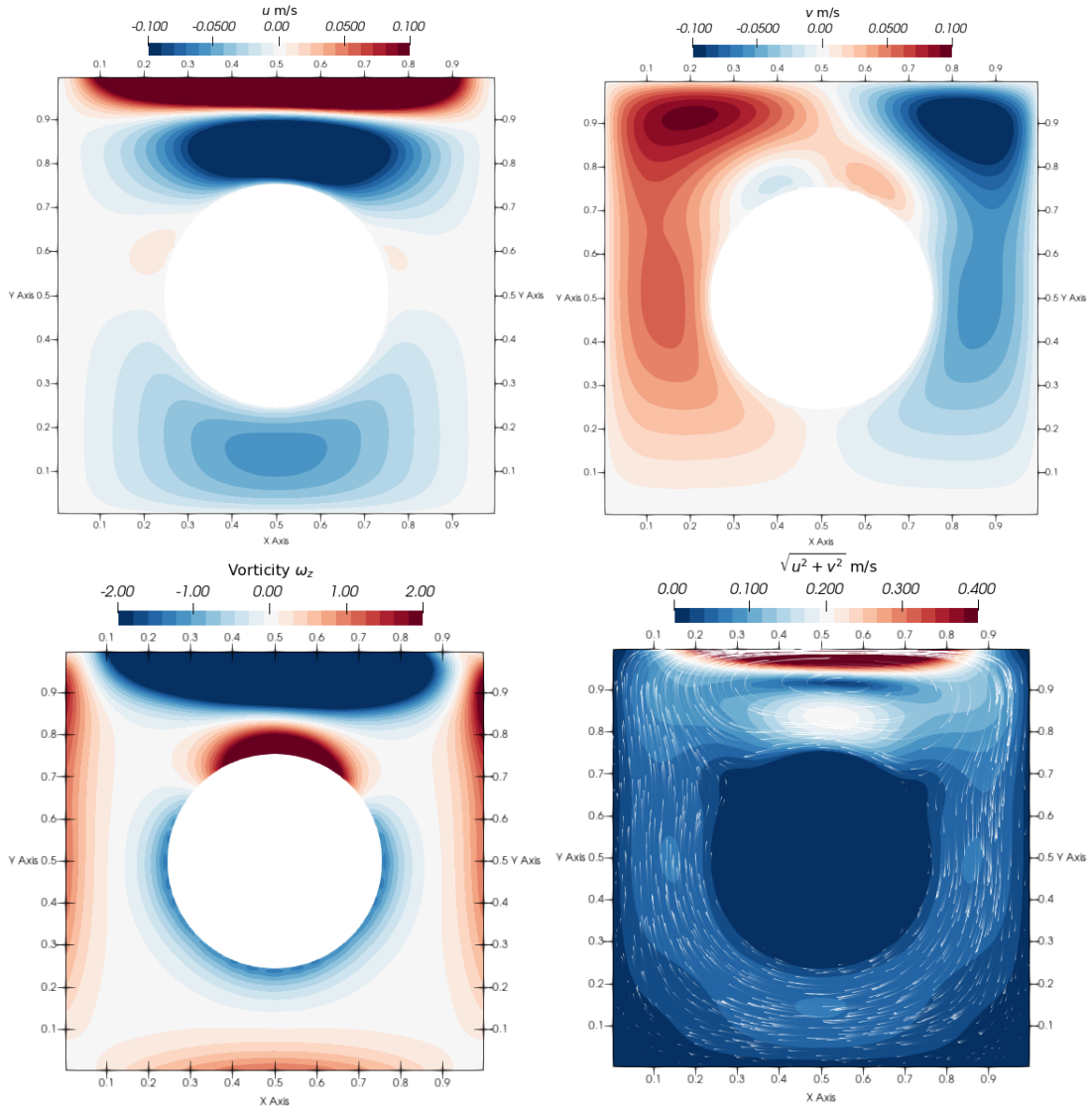

Figura 5.7: Solución numérica del problema de la cavidad cuadrada regularizada, con un cilindro fijo embebido en el centro, para la grilla de 128×128. Componentes u y v de la velocidad (arriba), campo de vorticidad y módulo del campo de velocidades junto a la representación del campo de velocidad con puntos aleatorios (abajo).

Para el estudio de convergencia se utilizaron grillas de 32, 64, 128, 256 y 512 celdas por dirección, En la figura 5.8 se presentan los errores en las tres normas usuales, calculados de la misma forma que en las pruebas anteriores con la ecuación de difusión. En la figura 5.8 se presentan los resultados de convergencia en malla para ambas componentes de la velocidad y del campo de presiones, puede observarse que el método converge a segundo orden al refinar.

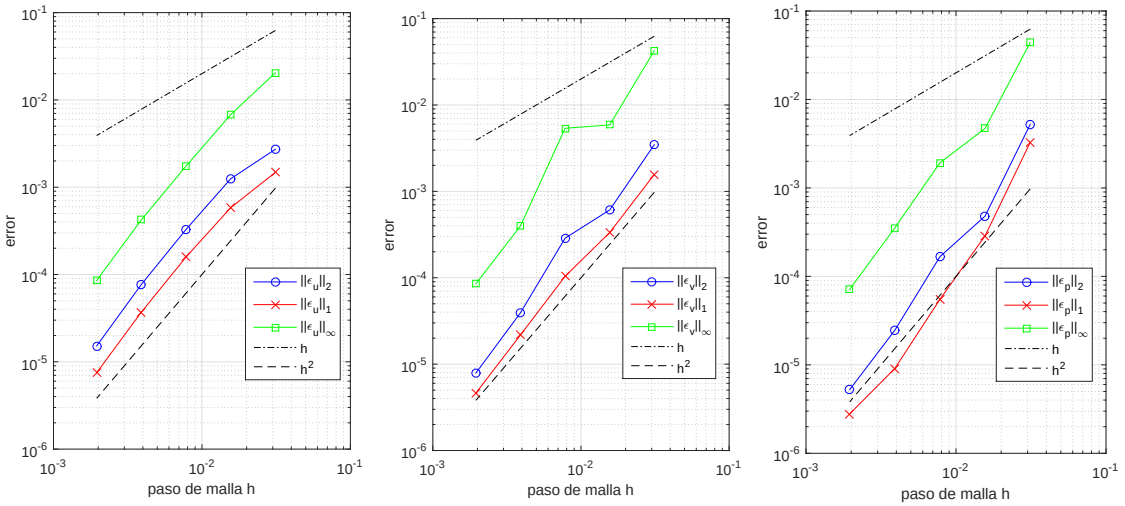

Figura 5.8: Normas L_1, L_2 y L_∞ del error versus paso de malla h para las componentes u, v de la velocidad y la presión p.

5.1.5. Flujo alrededor de un cilindro circular a $Re = 40$

En esta prueba se resuelve numéricamente el flujo alrededor de un cilindro circular a número de Reynolds $Re = 40$. La prueba consiste en un cilindro de diámetro $D = 1$m, colocado en el centro de un dominio cuadrado $30D \times 30D$. Las BC utilizadas para la velocidad son: en la entrada (eje vertical a la izquierda) y paredes horizontales superior e inferior son un flujo uniforme constante $\mathbf{u}_\infty = (u_\infty, 0)$ con $u_\infty = 1$m/s, en la salida se utilizó una BC del tipo convectiva para ambas componentes de $\mathbf{u}$ calculada mediante la siguiente expresión

$$\frac{\partial \mathbf{u}}{\partial t} + u_\infty \frac{\partial \mathbf{u}}{\partial x} = \mathbf{0} \tag{5.20}$$

para la presión se utilizaron BCs tipo Neumann homogéneas en todos los lados y se fijó la presión en la primer celda para proveer un sistema resoluble.

El problema se resolvió en un dominio computacional de 4096×4096 ($h \approx 0{,}0073$m), se utilizaron los mismos esquemas espaciales y temporales que en la prueba anterior. El paso de tiempo elegido es de $\Delta t = 1 \times 10^{-3}$s y la simulación se realizó hasta llegar a un tiempo final de $T = 20$s, alcanzando de este modo el estado estacionario. Los parámetros físicos se fijaron de manera que resulte un número de Reynolds basado en el diámetro del cilindro de $Re_D = 40$. Se compararon los resultados de la vorticidad y los resultados del coeficiente de presión sobre la superficie del cilindro con otros autores.

La figura 5.9 muestra las curvas de isovalores para la vorticidad comparables a las soluciones de otros autores, para facilitar la comparación, en la figura se muestran los resultados obtenidos por Taira y Colonius [164] y los resultados obtenidos por Krishnan [87], puede observarse que los resultados son satisfactorios considerando que la vorticidad en general es un parámetro sensible a los cambios.

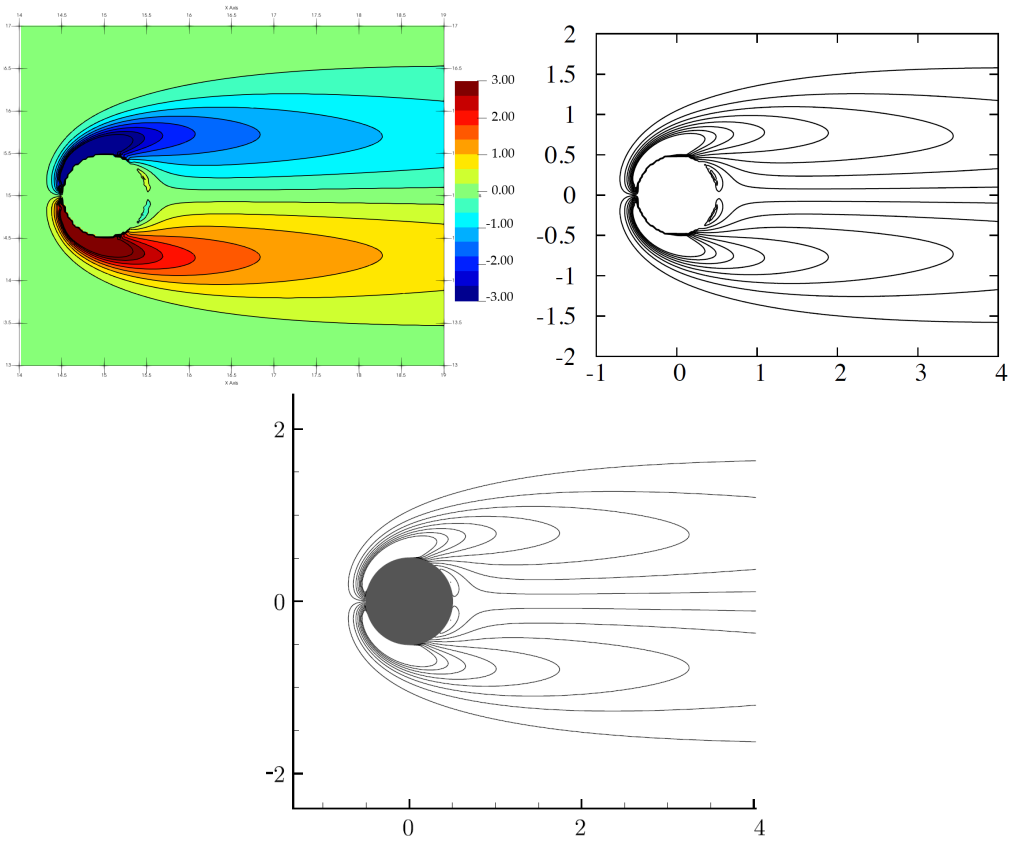

Figura 5.9: Curvas de isovalores de vorticidad para la prueba de flujo alrededor de un cilindro a $Re = 40$, contornos entre valores -3 y 3 graficados cada 0.4. Resultados de la presente implementación (izquierda), resultados presentados por Krishnan [87] (derecha) y resultados presentados por Taira y Colonius [164] (centro).

En la figura 5.10 se muestran los resultados para el coeficiente de presión normalizado definido como $C_p = (p - p_\infty)/\frac{1}{2}\rho U_\infty^2$, considerando $p_\infty = 0$ como presión ambiente. Puede observarse que la distribución de presiones sobre la superficie se aproxima bien con los resultados numéricos presentados por Tseng y Ferziger [169].

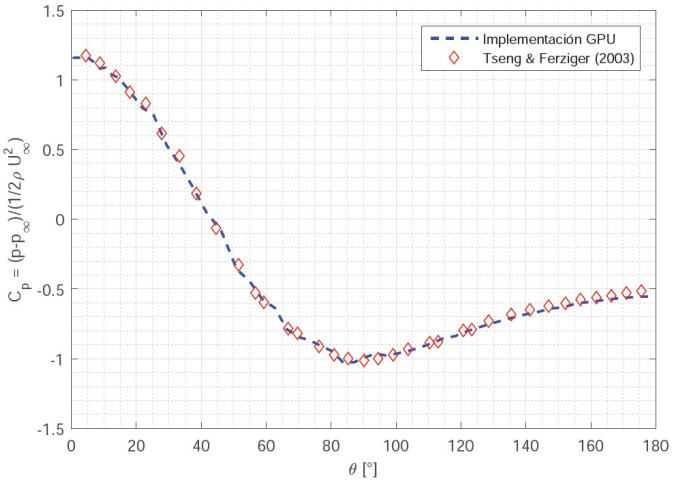

Figura 5.10: Coeficiente de presión sobre el cilindro a $Re = 40$.

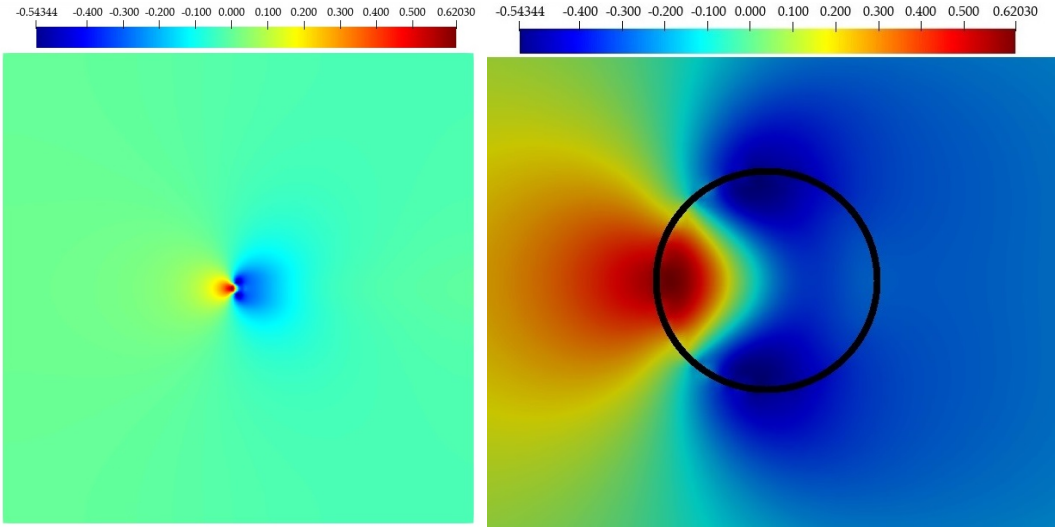

Figura 5.11: Contornos del campo de presiones para el flujo alrededor del cilindro a $Re = 40$.

5.1.6. Flujo alrededor de una esfera

Otro experimento que permite probar la fidelidad del método en flujos tridimensionales es el flujo alrededor de una esfera fija. Para números de Reynolds basados en el diámetro de la esfera inferiores a $Re_D = 210$ el flujo es estacionario axisimétrico, para $210 \leq Re_D \leq 270$ el flujo es estacionario no axisimétrico mientras que para $Re_D \geq 280$ el flujo se vuelve no estacionario y no axisimétrico.

En esta prueba se utiliza un dominio computacional de $[-8D, 8D] \times [-4D, 4D] \times [-4D, 4D]$, donde D es el diámetro de la esfera con centro en el punto $(-4D, 0, 0)$ (ver figura 5.12). Las BCs aplicadas para la velocidad son: flujo uniforme $\mathbf{u}_\infty\infty = (u_\infty, 0, 0)$ en la entrada, BC convectiva en la salida (como en el problema anterior), en el resto de paredes se aplicó BC tipo *slip* (resbalamiento) mientras que sobre la superficie de la esfera se aplicó una BC tipo *no-slip* (condición de adherencia) impuesta mediante el IBM. Para la presión las BCs son del tipo Neumann en las paredes y entrada, mientras que en la salida se impuso $p_\infty = 0$. En la figura 5.12 se indican la configuración del dominio y las BCs.

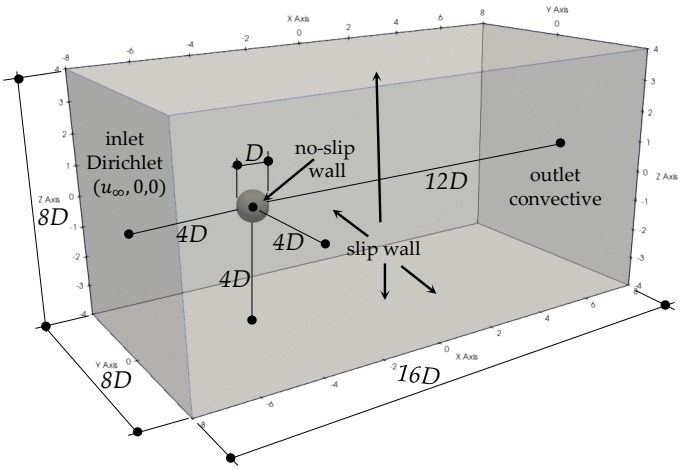

Figura 5.12: Esquema de dominio computacional para el problema de flujo alrededor de una esfera estacionaria.

La grilla utilizada es de $512 \times 256 \times 256$, el esquema espacial utilizado es CD para el término advectivo y difusivo, como esquemas temporales se utilizaron CN y AB para el término difusivo y advectivo respectivamente, para asegurar la estabilidad y una buena resolución el paso de tiempo

se eligió como $\Delta t = \frac{0{,}5h}{U_\infty 1{,}5}$. Se calculó el coeficiente de arrastre definido como:

$$C_D = \frac{F_D}{\frac{1}{2}\rho u_\infty A_{CS}} \tag{5.21}$$

siendo ρ la densidad del fluido, $A_{CS} = \pi D^2/4$ el área transversal de la esfera y F_D la fuerza de arrastre evaluada como se indica en [95], esto es, calcular numéricamente la forma integral de la componente x de la ecuación de momento:

$$\begin{aligned} F_D &= -\int_{\Omega_0} f_1 dV \\ F_D &= -\frac{\partial}{\partial t}\int_{\Omega_0} \rho u dV - \int_{\partial\Omega_0} \rho u \mathbf{u}\cdot\mathbf{dS} - \int_{\partial\Omega_0} p n_x ds + \int_{\partial\Omega_0} \mu\left(\frac{\partial u}{\partial x_j} + \frac{\partial u_j}{\partial x}\right) n_j ds \end{aligned} \tag{5.22}$$

donde Ω_0 es un dominio cúbico que encierra la esfera y $\partial\Omega_0$ su frontera.

Se realizaron simulaciones para diferentes valores de $Re_D = \rho u_\infty D/\mu$: 50, 75, 100, 150, 200, 250, 300, 350 y 400. La tabla 5.1 muestra que los resultados del coeficiente de arrastre promedio tienen un ajuste razonable al compararlo con los reportados en estudios previos. En la figura 5.13 puede observarse que estos valores se ajustan bien a la correlación de Clift et al. [39].

Tabla 5.1: Coeficiente de arrastre promedio $\overline{C_D}$ para la esfera. (*) Resultados experimentales.

Re	50	75	100	150	200	250	300	350	400
Kim et al. [84]	-	-	1.087	-	-	0.701	0.657	-	-
Choi et al. [36]	-	-	1.090	-	-	0.700	0.658	-	-
Johnson y Patel [79]	-	-	1.080	-	-	0.700	0.656	-	-
Ross y Willmarth [146] (*)	1.603	1.291	1.073	0.894	0.728	0.712	0.629	0.633	0.584
Tabata e Itakura [163]	1.579	-	1.090	0.889	0.772	-	-	-	-
Este trabajo	1.603	1.278	1.095	0.887	0.765	0.693	0.645	0.615	0.579

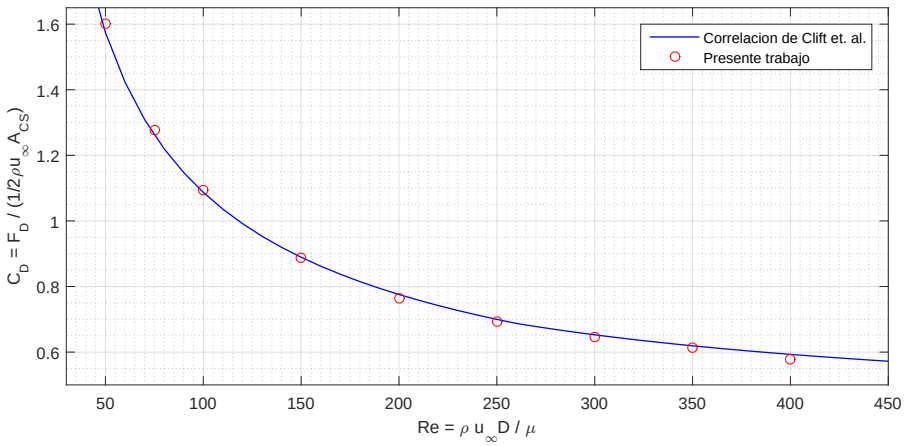

Figura 5.13: Evaluación del coeficiente de arrastre C_D oara una esfera a diferentes números de Reynolds, comparación con formula de correlación de Clift et. al. [39].

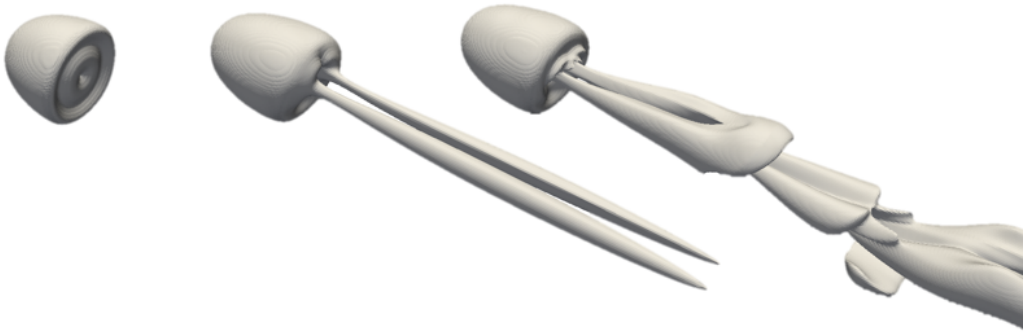

Figura 5.14: Estructuras de desprendimiento de vórtices. Isosuperficies de criterio Q de vorticidad, de izquierda a derecha $Re = 100, 250, 300$.

5.2. Métodos Level Set

El LSM es una técnica numérica que frecuentemente se utiliza para problemas que involucran la interfaz entre fluidos en movimiento. El método se basa en una representación implícita de la interfaz, cuya ecuación de movimiento se aproxima numéricamente usando ecuaciones hiperbólicas. El LSM permite manejar problemas en los que la velocidad de la interfaz depende de manera sensible de las propiedades geométricas locales como curvatura y dirección normal, así como también saltos en las propiedades físicas y las condiciones de borde determinadas por la ubicación de la interfaz.

El LSM fue propuesto por Osher y Sethian [130]. En esta tesis se utiliza para modelar el lecho que oficia de interfaz entre agua y sedimentos.

Como se explica en el capítulo 6, en cada paso de tiempo morfológico, la interfaz debe actualizarse ya sea advectando la función ϕ (SDF en 3D) a partir de un campo de velocidades vertical determinado o bien calculando explícitamente una nueva aproximación de la función SDF a partir del campo 2D que provee los niveles del lecho de acuerdo a la ecuación de Exner. En ambos casos la función distancia nueva puede que no se mantenga como tal y eventualmente debe reemplazarse por una función SDF sin cambiar la ubicación de la interfaz, es decir, sin mover el conjunto de nivel cero ($\phi(x, y, z) = 0$).

Este proceso se denomina reinicialización. La función ϕ debe ser reinicializada luego de una cierta cantidad de tiempo, aquí se realiza en cada paso de tiempo morfológico.

5.2.1. Reinicialización

El método de reinicialización influye en la precisión general del LSM y en la conservación de la masa. Constituyendo así un paso importante para mantener la función ϕ como una función SDF preservando la propiedad dada por $\phi(\mathbf{x}) = 0$.

Por lo tanto se requiere realizar esto de forma adecuada y precisa para mantener la calidad del LSM y garantizar alta precisión y conservación de masa en el cálculo de la interfaz en movimiento.

Existen diferentes métodos para realizar esto, en [18] puede encontrarse una descripción, entre estos, los dos métodos más difundidos son: los basados en la solución de una ecuación diferencial hiperbólica y el método de marcha rápida (FMM por sus siglas en inglés). A pesar que el FMM es un algoritmo eficiente y preciso, su paralelización resulta muy dificultosa debido a la dependencia en el flujo de datos que se crea [148]. Entre los avances recientes, Lee et al. (2016) [98] han presentado un algoritmo eficiente usando la fórmula de Hopf-Lax para resolver la ecuación Eikonal ($||\nabla\phi|| = 1$), conformando una estrategia con paralelismo trivial. Posteriormente Royston et al. (2018) [148] han presentado una paralelización del método en GPU.

Como aquí la reinicialización se requiere para corregir principalmente las propiedades cerca del conjunto $\phi(\mathbf{x}) = 0$, cosa que insume pocas iteraciones si se utiliza los métodos clásicos y

además, se realizan en cada paso de tiempo morfológico, a los efectos de esta tesis, se optó por seguir directamente un método basado en la ecuación diferencial:

$$\begin{cases} \dfrac{\partial \phi}{\partial t} + \text{sgn}(\phi^0)\left(||\nabla \phi|| - 1\right) = 0 \\ \phi(\mathbf{x}, 0) = \phi^0(\mathbf{x}) \end{cases} \tag{5.23}$$

donde $\text{sgn}(*)$ representa el signo de $*$. No obstante en futuras optimizaciones o en caso de incorporar problemas a superficie libre se recomienda realizar una implementación basada en GPU similar a la presentada en [148].

La ecuación (5.23) se discretizó utilizando FDM tal como se describe en el trabajo de Min (2010) [115], con la respectiva extensión a 3D.

En particular, la discretización espacial de $||\nabla \phi||$ consiste en aplicar un esquema espacial tipo ENO (por sus siglas en inglés *Essentially Non-Oscillatory*) de segundo orden en los argumentos del Hamiltoniano numérico que aquí se presenta en 2D por simplicidad:

$$||\nabla \phi|| \approx H_G(D_x^+ \phi_C, D_x^- \phi_C, D_y^+ \phi_C, D_y^- \phi_C) \tag{5.24}$$

donde $D_x^{\pm}$ y $D_y^{\pm}$ son las diferencias finitas ENO hacia un lado u otro en dirección x e y dadas por:

$$\begin{aligned} D_x^+ \phi_C &= \frac{\phi_E - \phi_C}{\Delta x} - \frac{\Delta x}{2}\text{minmod}(D_{xx}\phi_C, D_{xx}\phi_E), \\ D_x^- \phi_C &= \frac{\phi_C - \phi_W}{\Delta x} - \frac{\Delta x}{2}\text{minmod}(D_{xx}\phi_C, D_{xx}\phi_W) \end{aligned} \tag{5.25}$$

las expresiones para $D_y^{\pm}$ son similares considerando los valores ϕ_N y ϕ_S. El factor $D_{xx}\phi_C$ denota al laplaciano numérico en diferencias centradas dado por:

$$D_{xx}\phi_C = \frac{\phi_E - 2\phi_C + \phi_W}{\Delta x^2} \tag{5.26}$$

y el operador $\text{minmod}(a, b)$ o módulo mínimo dado por

$$\text{minmod}(a, b) = \begin{cases} 0, & \text{si } ab < 0 \\ a, & \text{si } |a| < |b| \\ b, & \text{en otro caso} \end{cases} \tag{5.27}$$

Por su parte, el Hamiltoniano numérico H_G está dado por:

$$H_G(a, b, c, d) = \begin{cases} \sqrt{\text{máx}\left((a^-)^2, (b^+)^2\right) + \text{máx}\left((c^-)^2, (d^+)^2\right)}, & \text{si sgn}(\phi^0) \geq 0, \\ \sqrt{\text{máx}\left((a^+)^2, (b^-)^2\right) + \text{máx}\left((c^+)^2, (d^-)^2\right)}, & \text{si sgn}(\phi^0) < 0, \end{cases} \tag{5.28}$$

aquí el signo en el superíndice indica la siguiente evaluación de máximo o mínimo $a^+ = \text{máx}(a, 0)$, $a^- = \text{mín}(a, 0)$.

Hasta aquí el esquema espacial es adecuado para puntos ubicados lejos de $\phi = 0$, mientras que los puntos cerca del contorno pueden modificar la ubicación de la interface, por ello, en dichos puntos deben corregirse las fórmulas para imponer $\phi = 0$ siempre que $\phi^0 = 0$. En concreto la modificación requerida consiste en reemplazar $D_x^{\pm}\phi_C$ (y respectivamente $D_y^{\pm}\phi_C$) por:

$$\begin{aligned} D_x^+ \phi_C &= \frac{0 - \phi_C}{\Delta x^+} - \frac{\Delta x^+}{2}\text{minmod}(D_{xx}\phi_C, D_{xx}\phi_E) \\ D_x^- \phi_C &= \frac{\phi_C - 0}{\Delta x^-} + \frac{\Delta x^-}{2}\text{minmod}(D_{xx}\phi_C, D_{xx}\phi_W) \end{aligned} \tag{5.29}$$

donde $\Delta x^{\pm}$ es calculado como:

$$\Delta x^+ = \begin{cases} \Delta x\left(\dfrac{1}{2} + \dfrac{\phi_C^0 - \phi_E^0 - \text{sgn}\left(\phi_C^0 - \phi_E^0\right)\sqrt{D}}{\phi_{xx}^0}\right), & \text{si } |\phi_{xx}|^0 > \epsilon \\ \Delta x \dfrac{\phi_C^0}{\phi_C^0 - \phi_E^0}, & \text{sino} \end{cases} \tag{5.30}$$

donde $\phi^0_{xx} = \text{minmod}(\phi^0_W - 2\phi^0_C + \phi^0_E, \phi^0_C - 2\phi^0_E + \phi^0_{EE})$, $D = (\phi^0_{xx}/2 - \phi^0_C - \phi^0_E)^2 - 4\phi^0_C\phi^0_E$ y teniendo en cuenta que esta ecuación es valida cuando $\phi^0_C\phi^0_E < 0$.

Mientras que para el caso $\phi^0_C\phi^0_W < 0$, debe usarse:

$$\Delta x^- = \begin{cases} \Delta x \left(\dfrac{1}{2} + \dfrac{\phi^0_C - \phi^0_W - \text{sgn}\left(\phi^0_C - \phi^0_W\right)\sqrt{D}}{\phi^0_{xx}} \right), & \text{si } |\phi_{xx}|^0 > \epsilon \\ \Delta x \dfrac{\phi^0_C}{\phi^0_C - \phi^0_W}, & \text{sino} \end{cases} \tag{5.31}$$

con $\phi^0_{xx} = \text{minmod}(\phi^0_W - 2\phi^0_C + \phi^0_E, \phi^0_C - 2\phi^0_W + \phi^0_{WW})$, $D = (\phi^0_{xx}/2 - \phi^0_C - \phi^0_W)^2 - 4\phi^0_C\phi^0_W$.

Para finalizar, la discretización temporal utilizada es FE por simplicidad. El paso de tiempo adoptado en el caso 2D (respectivamente en 3D incoporando z) es:

$$\Delta t = 0{,}45\,\text{mín}(\Delta x^+, \Delta x^-, \Delta y^+, \Delta y^-) \tag{5.32}$$

La deducción de las últimas fórmulas puede encontrarse en [115].

5.2.2. Ejemplos numéricos 2D

El siguiente ejemplo ilustrativo consiste en aplicar el método de reinicialización considerando la siguiente función LS inicial:

$$\phi^0(x, y) = \left((x-1)^2 + (y-1)^2 + 0{,}1\right)\left(\sqrt{x^2 + y^2} - 1\right) \tag{5.33}$$

Aquí el conjunto $\phi^0(x, y) = 0$ es una circunferencia de radio 1 centrada en el origen de coordenadas, sin embargo ϕ^0 no constituye una SDF, es decir no tiene la propiedad $||\nabla\phi|| = 1$. La figura 5.15 presenta la solución final, donde puede verse que se logran obtener las propiedades deseadas para la SDF. Para finalizar, a modo de ejemplo en la figura 5.16 se muestra la solución final de la función marcadora utilizada para aplicar la estrategia GC-IBM en un problema lecho erosionable, el proceso para construir la condición inicial ϕ^0 en este caso se describe en el capítulo 6.

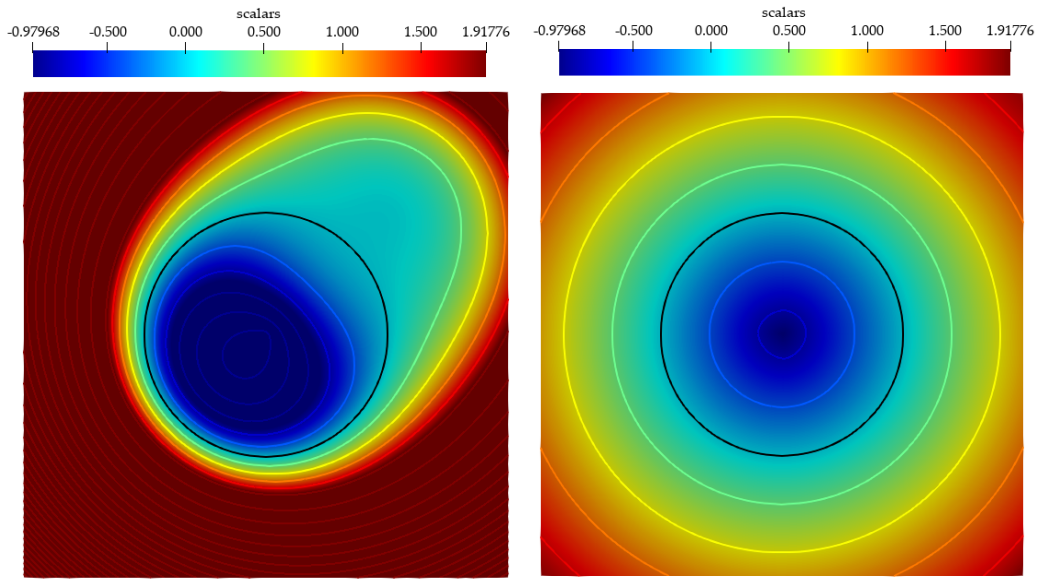

Figura 5.15: Contornos de nivel para la condición inicial ϕ^0 del problema (5.33) (izquierda) y contornos de nivel de la solución final ϕ luego de aplicar el proceso de reinicialización (derecha). Para claridad se ha demarcado en negro el contorno $\phi(\mathbf{x}) = 0$ para ϕ y ϕ^0.

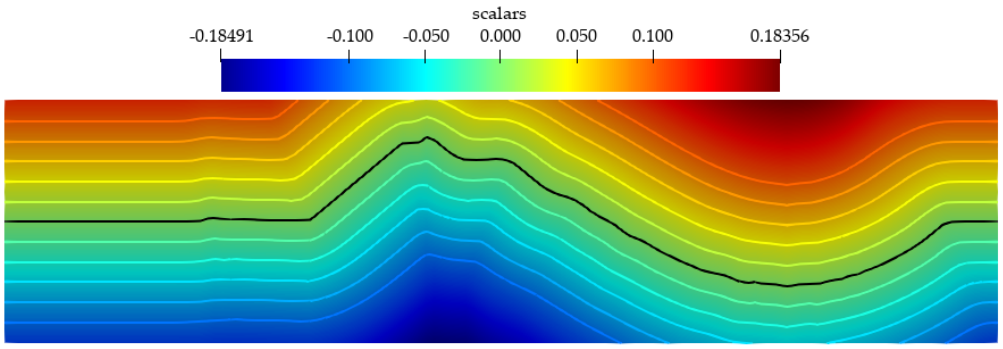

Figura 5.16: SDF utilizada para demarcar el lecho erosionable en un problema 2D luego de aplicar el proceso de reinicialización.

Capítulo 6

Ecuaciones y modelos para el transporte de sedimentos

6.1. Introducción

El transporte de sedimentos se divide en: carga de lavado (material muy fino transportado en suspensión, es aquel que no influye en la hidrodinámica local), y el transporte total de lecho (sedimento transportado en el fondo más el transportado en suspensión, dependiendo del tamaño del sedimento y de la velocidad del flujo). Las propiedades principales del sedimento son el tamaño de partícula, la forma, densidad, velocidad de caída, porosidad y concentración, mientras que las principales propiedades del transporte de sedimento son las variables hidráulicas interrelacionadas tales como la descarga, la pendiente de fondo, la sección de flujo, velocidad, turbulencia, rugosidad hidráulica. La erosión y sedimentación constituyen procesos que pueden requerir simulaciones en largos períodos de tiempo.

El movimiento de los sedimentos involucra procesos complejos. El campo de velocidades del flujo de agua es tridimensional, con zonas de recirculación y flujos secundarios. Las formas del lecho que surgen muchas veces aumentan la rugosidad, pudiendo modificar incluso el régimen del curso. Como consecuencia del transporte total (carga de fondo y suspendida), la superficie del lecho cambia y su evolución temporal se modela usando la ecuación de Exner.

Hay varias fórmulas de transporte $\mathbf{q}_s$ empíricas para diferentes aplicaciones, las más difundidas son las correspondientes a Shield [153], Meyer-Peter & Muller [113], Einstein [49], Engelund & Hansen [51], Engelund y Fredsøe [50], Yalin [185], Fernandes-Luque y van Beek [60], Van Rijn [176], entre otros. Los parámetros que se requieren son: el tamaño medio d_s de los granos (d_{50} en general), la densidad del fluido ρ, la densidad relativa $s = \rho_s/\rho$, siendo ρ_s la densidad del sedimento, la aceleración de la gravedad $\mathbf{g}$, el esfuerzo de corte crítico τ_c, y el esfuerzo de corte en el fondo $\boldsymbol{\tau}$.

En el presente capítulo se abordan los aspectos teóricos de la formulación para ambos tipos de transporte, sin embargo, en el caso de aplicación propuesto solamente se considerará el transporte en modalidad de fondo.

6.2. Transporte de fondo

Los granos comienzan a moverse cuando la tensión en el lecho $\boldsymbol{\tau}_b$ exceda un valor crítico (τ_c). En las primeras etapas de transporte, las partículas se mueven deslizándose y rodando sobre la superficie del lecho, pero un pequeño aumento en el esfuerzo $\boldsymbol{\tau}_b$ hará que los granos salten desde el fondo siguiendo trayectorias de tipo balísticas. Este último modo de transporte sólido por carga de fondo, se conoce como saltación [62]. Dentro de una delgada capa cercana al fondo, se establece una zona de intercambio entre el lecho y el sedimento transportado en suspensión.

6.2.1. Fórmulas para transporte de fondo $\mathbf{q}_b$

La mayoría de estudios numéricos dividen la carga de transporte en carga de fondo y carga en suspensión. Las fórmulas de transporte por carga de fondo correspondientes al movimiento adyacente al lecho se basan principalmente en el concepto del esfuerzo de corte crítico τ_c, que establece un valor para el cual, una vez alcanzado por el esfuerzo de corte que genera el flujo sobre la superficie, el sedimento será transportado.

Las siguientes fórmulas frecuentemente han sido aplicadas para predecir el transporte de fondo en simulaciones numéricas 2D y 3D, por ejemplo en [8, 30, 83, 86, 103, 123, 147]:

- Meyer-Peter y Müller (1948) [113]:

$$\mathbf{q}_b^* = \frac{\mathbf{q}_b}{\sqrt{(s-1)gd_{50}^3}} = 8\left(\frac{\boldsymbol{\tau}_b}{\rho(s-1)gd_{50}} - \tau_c^*\right)^{3/2} \tag{6.1}$$

  siendo $\mathbf{q}_b^*$ la tasa de transporte adimensional, $\mathbf{q}_b$ la tasa volumétrica de transporte de fondo [m^3/s/m], $\boldsymbol{\tau}_b$ el esfuerzo de corte en el lecho [N/m^2], $g = ||\mathbf{g}||$ la aceleración de la gravedad [m/s^2], d_{50} en [m] el diámetro efectivo 50 del sedimento oficiando en cierta manera como el diámetro promedio, $s = \rho_S/\rho$ la densidad relativa del sedimento respecto al agua, usualmente se utiliza $s = 2{,}65$, $\tau_c^* = 0{,}047$ el valor de esfuerzo de corte crítico adoptado con frecuencia en arenas. La fórmula es empírica y se verificó con datos para arena gruesa y grava uniformes. Notar que para problemas 2D $\mathbf{q}_b = q_{b,x}$ mientras que en problemas 3D se utiliza $\mathbf{q}_b = (q_{b,x}, q_{b,y})$ y de manera similar debe considerarse $\boldsymbol{\tau}_b = (\tau_{b,x}, \tau_{b,y})$ en problemas 3D.

  Wong y Parker (2006) [180] trabajaron en un re-análisis de los datos usados por Meyer-Peter y Müller (MPM), logrando un mejor ajuste mediante las siguientes dos formas alternativas:

$$\mathbf{q}_b^* = 4{,}93\left(\boldsymbol{\tau}^* - 0{,}047\right)^{1{,}6} \tag{6.2}$$

$$\mathbf{q}_b^* = 3{,}97\left(\boldsymbol{\tau}^* - 0{,}0495\right)^{3/2} \tag{6.3}$$

  donde $\boldsymbol{\tau}_b^*$ es el esfuerzo de corte adimensionalizado.

- Engelund y Fredsøe (1976) [50]:

$$\mathbf{q}_b^* = 18{,}74\left(\boldsymbol{\tau}_b^* - \boldsymbol{\tau}_c^*\right)\left[\left(\boldsymbol{\tau}_b^*\right)^{1/2} - 0{,}7\left(\boldsymbol{\tau}_c^*\right)^{1/2}\right] \tag{6.4}$$

  donde se adopta $\tau_c^* = 0{,}05$. Esta ecuación da resultados muy próximos a los de la fórmula de MPM, aunque en las pruebas numéricas para esta tesis se observó que esta formula en general da tasas de transporte mayores. No obstante ambas fórmulas sobreestiman el transporte de fondo para grandes valores de esfuerzo de corte [62].

- Van Rijn (1984) [176]:

$$\mathbf{q}_b^* = 0{,}053\frac{T^{2,1}}{D_*^{0,3}} \tag{6.5}$$

  la ecuación esta basada en el tamaño de partícula adimensional y el parámetro de etapa de transporte (*transport stage parameter*) T, definidos respectivamente como:

$$D_* = d_{50}\left(\frac{g(s-1)}{\nu^2}\right)^{1/3} = R_{ep}^{2/3} \tag{6.6}$$

$$T = \frac{\boldsymbol{\tau}_s^* - \boldsymbol{\tau}_c^*}{\boldsymbol{\tau}_c^*} \tag{6.7}$$

donde R_{ep} se conoce usualmente como el número de Reynolds de las partículas del lecho pues viene definido por

$$R_{ep} = \frac{\sqrt{g(s-1)d_{50}}\, d_{50}}{\nu} \tag{6.8}$$

con ν la viscosidad cinemática del fluido. $\boldsymbol{\tau}_s^*$ es el esfuerzo de corte debido a la resistencia de grano, frecuentemente estimado (ver por ejemplo [8]) como $\boldsymbol{\tau}_s = \rho \mathbf{u}_*$, siendo $\mathbf{u}_*$ la velocidad de corte considerando un perfil logarítmico cerca del lecho $\mathbf{u}_* = \mathbf{u}\kappa / \ln(30z/k_s)$, con $\mathbf{u}$ la velocidad a una altura z medida desde el fondo, $\kappa = 0{,}41$ la constante de Von Karman y la rugosidad equivalente Nikuradse estimada como $k_s = 3d_{50}$. $\boldsymbol{\tau}_c^*$, como antes, es el esfuerzo de corte crítico para iniciación del movimiento del diagrama de Shields. Esta fórmula puede usarse para estimar las tasas de transporte de fondo en casos de partículas con tamaños en un rango de 0.2 y 2.0 mm [62].

- Nielsen (1992) [126]:

$$\mathbf{q}_b^* = 12\boldsymbol{\tau}_b^{*1/2}(\boldsymbol{\tau}_b^* - \boldsymbol{\tau}_c^*) \tag{6.9}$$

fórmula de transporte de fondo adecuada para arenas y gravas con tamaño uniformes.

- Fernandez-Luque y van Beek (1976) [60]:

$$\mathbf{q}_b^* = 5{,}7\,(\boldsymbol{\tau}_b^* - \boldsymbol{\tau}_c^*)^{3/2} \tag{6.10}$$

donde τ_c^* se elige variable según el diámetro d_{50} entre 0.05 para diámetros de 0.9 mm y 0.058 para material de 3.3 mm.

La lista anterior, lejos de ser exhaustiva, menciona los modelos de carga de fondo encontrados generalmente al revisar trabajos de simulación numérica de transporte de sedimentos.

6.3. Transporte en suspensión

Como sólo los granos muy finos se transportan como material en suspensión, para describir el transporte de este tipo de material se utiliza la ecuación de advección-difusión. La idea es que esta fracción de sedimento tenga un tamaño de grano lo suficientemente pequeño para poder omitir el efecto inercial de las partículas [103].

Se considera todo el material arrastrado en suspensión como una concentración de masa C en el fluido, la ecuación de conservación es la ecuación (2.1) donde se agrega un término adevctivo que contempla la sedimentación por peso propio:

$$\begin{aligned} &\frac{\partial C}{\partial t} + \nabla\cdot(\mathbf{u}C - \nu_t \nabla C) + w_s \frac{\partial c}{\partial z} = 0 \\ \Leftrightarrow\ &\frac{\partial C}{\partial t} + \nabla\cdot\left(\mathbf{u}C - w_s \frac{\mathbf{g}}{||\mathbf{g}||} - \nu_t \nabla C\right) = 0 \end{aligned} \tag{6.11}$$

donde $\mathbf{u}$ es la velocidad del fluido, ν_t es la difusividad con valores muy pequeños, en el orden de $10^{-7}\mathrm{m}^2/\mathrm{s}$ [28] y frecuentemente se adopta igual a la viscosidad de remolino turbulenta [103], el término adicional que simula el transporte por gravedad, tiene en cuenta la velocidad vertical de sedimentación w_s, que puede estimarse mediante extensiones de la ley de Stokes a partir de datos experimentales, por ejemplo:

- Zanke (1977) [188]:

$$w_s = \frac{10\nu}{d_{50}}\left[\left(1 + \frac{0{,}01(s-1)g d_{50}^3}{\nu^2}\right)^{1/2} - 1\right] \tag{6.12}$$

- Ferguson y Church (2004) [59]:

$$w_s = \frac{(s-1)gd_{50}^2}{C_1\nu + \left(0{,}75C_2(s-1)gd_{50}^2\right)^{1/2}} \tag{6.13}$$

en la última fórmula C_1 es la constante de la ley de Stokes, valor teórico para esferas ($C_1 = 18$), C_2 forma parte del ajuste a datos experimentales realizado por [59], quienes recomiendan adoptar $C_2 = 1{,}0$ y hasta un máximo de $C_2 = 1{,}2$ para granos con forma angular.

Las condiciones de borde adoptadas para la ecuación son del tipo Dirichlet $C = c_{\text{ref}}$ sobre el lecho y BC tipo Neumann $\partial C/\partial \mathbf{n} = 0$ en el resto de fronteras del fluido, siendo c_{ref} es una concentración de referencia en una capa cerca de la superficie del lecho. Sin embargo en caso que se modelen problemas a superficie libre, puede considerarse una BC tipo mixta (ver por ejemplo [86]) o bien BC tipo Dirichlet homogénea (ver por ejemplo [103]).

En cuanto a la solución numérica de la ecuación (6.11), se realiza utilizando los algoritmos desarrollados en los capítulos 2 y 3 teniendo en cuenta la estrategia CG-IBM descrita en el capítulo 5 para tratar las geometrías complejas.

6.4. Modelo de evolución del lecho - Ecuación de Exner

La ecuación que gobierna los cambios de nivel del lecho, es la ecuación de Exner [53] que surge a partir de la conservación de masa sobre el fondo. La ecuación en su forma más simplificada luce como:

$$\frac{\partial \xi}{\partial t} = -\nabla\cdot \mathbf{q}_b\left(\boldsymbol{\tau}_b\right) \tag{6.14}$$

en donde se explicita la dependencia de la tasa de transporte con el esfuerzo de corte en el lecho, que a su vez, es una función del campo de velocidades del flujo $\mathbf{u}$, por otro lado ξ representa el nivel del lecho medido desde una referencia (ver figura 6.1).

Hay ecuaciones más complejas que contemplan el intercambio entre carga de fondo y carga en suspensión, como también correcciones cuando se está fuera de las condiciones de equilibrio. Para esta tesis se utiliza la siguiente versión simplificada de la fórmula propuesta por [181]:

$$\frac{\partial \xi}{\partial t} = \frac{1}{(1-n)}\left[-\nabla\cdot \mathbf{q}_b\left(\boldsymbol{\tau}_b\right) + D_b - E_b\right] \tag{6.15}$$

aquí n es la porosidad, mientras que D_b y E_b corresponden a la tasa de deposición y de arrastre hacia suspensión respectivamente, de manera que el término adicional $(D_b - E_b)$ representa el flujo neto que atraviesa la interfaz entre la capa adyacente al lecho y la capa de sedimento que viaja en suspensión (ver figura 6.1), esto permite lograr el acoplamiento del modelo de evolución del lecho junto con el modelo de carga en suspensión Las BC aplicadas sobre la ecuación (6.15) en general son del tipo Neumann homogéneas.

En cuanto a las tasas de deposición y erosión, hay diversas fórmulas para su estimación. Burkow y Griebel (2018) [30] adoptan lo siguiente:

$$D_b = \text{máx}\left(C - c_{\text{ref}}, 0\right) \tag{6.16}$$

$$E_b = M\,\text{máx}\left(||\boldsymbol{\tau}_b|| - \boldsymbol{\tau}_c, 0\right)\frac{||\boldsymbol{\tau}_b||}{\rho(s-1)gd_{50}} \tag{6.17}$$

donde $M = 2{,}2\times 10^{-3}$s/m es un parámetro empírico. En caso que D_b sea positivo, cuando las concentraciones en el fondo superan el valor de referencia, se incrementa el nivel del lecho en un valor igual a $\Delta\xi = D_b h/\rho_s$ donde h es el paso de malla de la grilla.

La idea es, que al utilizar la condición de borde $C = c_{\text{ref}}$ cerca del lecho en el modelo de advección-difusión (6.11), permite que si hay masa disponible para deposición (ecuación (6.16))

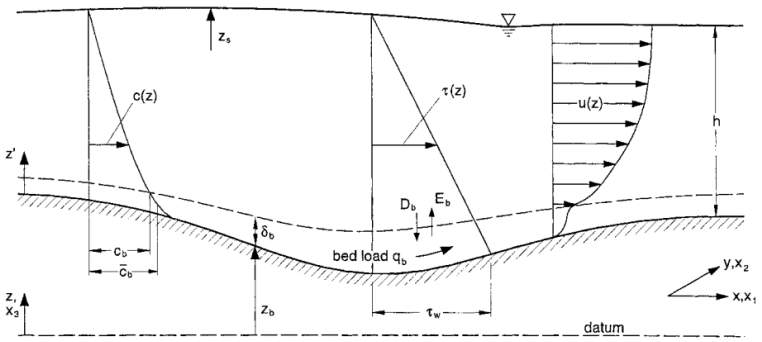

FIG. 1. Flow Configuration

Figura 6.1: Esquema de configuración del flujo, imagen extraída de Wu et al. [181].

entonces eso significará un aumento en el nivel del lecho, en caso contrario, la c_{ref} prevalece y el fondo permanece erosionable. Incluso ellos proponen modificar la BC de la ecuación (6.11) de acuerdo lo siguiente:

$$C = c_{\text{ref}} + \frac{E_b}{||\mathbf{u}_n||} - D_b \tag{6.18}$$

donde $\mathbf{u}_n$ es la componente de la velocidad normal al del lecho, cerca de él.

Otros autores como Ahmad et al. (2018) [8], Afzal et al. [149] (2015) utilizan el modelo propuesto por Wu et al. [181] que adopta lo siguiente:

$$D_b = w_s c_b \tag{6.19}$$

donde c_b es la concentración de sedimento en el primer centro de celda sobre el lecho, que provee la solución numérica de (6.11). Mientras que la tasa de arrastre hacia suspensión la calculan mediante la fórmula de van Rijn [144]:

$$E_b = w_s c_{\text{bed,susp.load}} \tag{6.20}$$

$$c_{\text{bed,susp.load}} = 0{,}015 \frac{d_{50}}{a} \frac{\left(\frac{\tau - \tau_c}{\tau_c}\right)^{1,5}}{\left(d_{50}\left(\frac{(s-1)g}{\nu^2}\right)^{1/3}\right)^{0,3}} \tag{6.21}$$

siendo $c_{\text{bed,susp.load}}$ la concentración en un nivel de referencia a, cerca del lecho, usualmente se elige una distancia $a = 0{,}05H$ desde el lecho, donde H es la profundidad del agua.

6.4.1. Discretización de la ecuación de Exner

Para la discretización de la ecuación de Exner, (6.14) o (6.15), se requieren calcular previamente las tasas de transporte de fondo mediante alguna de las fórmulas empíricas presentadas en la sección 6.2 para lo cual se requiere estimar los esfuerzos de corte como se indicará más adelante. Aquí se asumirá que se cuenta con los valores de $\mathbf{q}_b$ calculados en centros de celdas en un dominio computacional para el lecho compuesto por una grilla 2D. La misma consideración se tiene para el término $D_b - E_b$.

Integrando miembro a miembro la ecuación (6.15) sobre la celda C, aplicando el teorema del valor medio y de la divergencia se obtiene la ecuación semi discretizada:

$$\left(\frac{\partial \xi}{\partial t}\right)_C V_C = \frac{1}{(1-n)} \left[-\int_{\partial V_C} \mathbf{q} \cdot \mathbf{dS} + (D_b - E_b)_C V_C \right] \tag{6.22}$$

Aplicando el teorema de valor medio al término de integral de superficie se tiene:

$$\int_{\partial V_C} \mathbf{q}_b \cdot \mathbf{dS} = \sum_{f \sim nb(C)} \mathbf{q}_{b,f} \cdot \mathbf{S}_f \tag{6.23}$$

donde las componentes normales en las caras verticales y horizontales son respectivamente $q_{b,x}$ y $q_{b,y}$, para simplificar la notación en esta parte, el vector de las tasas de transporte por fondo se denotará como $(q_{b,x}, q_{b,y}) = (Q^x, Q^y) = \mathbf{Q}$. El esquema más simple de usar es el esquema CD, que implica:

$$\begin{aligned} \sum_{f \sim nb(C)} \mathbf{q}_{b,f} \cdot \mathbf{S}_f &= \left(Q^x_e - Q^x_w + Q^y_n - Q^y_s\right) h \\ \sum_{f \sim nb(C)} \mathbf{q}_{b,f} \cdot \mathbf{S}_f &= \left(\frac{Q^x_E - Q^x_W}{2} + \frac{Q^y_N - Q^y_S}{2}\right) h \end{aligned} \tag{6.24}$$

este esquema, a pesar de ser segundo orden, tiene problemas de estabilidad si no se utilizan pasos de tiempo relativamente pequeños en el esquema temporal explícito (debe elegirse $\Delta t \sim O(h^2)$). Por ello es deseable aplicar un esquema HR, sin embargo, en este caso el esquema de interpolación no se está realizando sobre la variable a resolver (ξ) sino que se está aplicando lo que podría llamarse "término fuente", para ello la estrategia que se propone es computar el término como:

$$\sum_{f \sim nb(C)} \mathbf{q}_{b,f} \cdot \mathbf{S}_f = \sum_{f \sim nb(C)} \dot{m}_f Q^{HR}_f \tag{6.25}$$

donde se utiliza la ecuación (2.5) para calcular Q^{HR}_f y en este caso se define un flujo unitario $\dot{m}_f = \text{sgn}\left(\mathbf{Q}_f \cdot \mathbf{S}_f\right) ||\mathbf{S}_f||$, donde la función $\text{sgn}(*)$ da el signo del escalar $*$. Esto permite aplicar los métodos TVD desarrollados en el capítulo 2. Entonces considerando la dirección del vector $\mathbf{Q}_f$ (de manera análodo a la velocidad $\mathbf{u}_f$) que atraviesa el centro de cara f de la celda C, por ejemplo para la cara e se debe calcular de la siguiente forma:

$$\begin{aligned} \dot{m}_e Q^{HR}_e = &\left[Q^x_C + \frac{1}{2}\psi(r^+_e)\left(Q^x_E - Q^x_C\right)\right] ||\dot{m}_e, 0|| - \left[Q^x_E + \frac{1}{2}\psi(r^-_e)\left(Q^x_C - Q^x_E\right)\right] || - \dot{m}_e, 0||; \\ &\text{con } r^+_e = \frac{Q^x_C - Q^x_W}{Q^x_E - Q^x_C},\ r^-_e = \frac{Q^x_E - Q^x_{EE}}{Q^x_C - Q^x_E},\ \dot{m}_e = \text{sgn}(Q^x_C + Q^x_E)\ h \end{aligned} \tag{6.26}$$

para la cara s luce:

$$\begin{aligned} \dot{m}_s Q^{HR}_s = &\left[Q^y_C + \frac{1}{2}\psi(r^+_s)\left(Q^y_S - Q^y_C\right)\right] ||\dot{m}_s, 0|| - \left[Q^y_S + \frac{1}{2}\psi(r^-_s)\left(Q^y_C - Q^y_S\right)\right] || - \dot{m}_s, 0||; \\ &\text{con } r^+_s = \frac{Q^y_C - Q^y_N}{Q^y_S - Q^y_C},\ r^-_s = \frac{Q^y_S - Q^y_{SS}}{Q^y_C - Q^y_S},\ \dot{m}_s = -\text{sgn}(Q^y_C + Q^y_S)\ h \end{aligned} \tag{6.27}$$

Aplicando esquema temporal explícito de segundo orden AB, el nivel del lecho actualizado en cada celda se computa como:

$$\begin{aligned} \xi^{n+1}_C = \xi^n_C - \frac{\Delta t}{h^2(n-1)} &\left[\frac{3}{2}\left(\sum_{f \sim nb(C)} \dot{m}^n_f Q^{HR,n}_f - (D^n_{b,C} - E^n_{b,C})h^2\right)\right. \\ &\left. -\frac{1}{2}\left(\sum_{f \sim nb(C)} \dot{m}^{n-1}_f Q^{HR,n-1}_f - (D^{n-1}_{b,C} - E^{n-1}_{b,C})h^2\right)\right] \end{aligned} \tag{6.28}$$

6.5. Modelo de turbulencia LES

El modelo de turbulencia LES por sus siglas en inglés (*Large Eddy Simulation*) es una técnica común usada para simular flujos turbulentos. La idea es que el solver para flujo incompresible calcule los remolinos más grandes mientras que los remolinos más pequeños implícitamente se tienen en cuenta mediante un modelo de escala de subgrilla (SGS, por sus siglas en inglés).

Partiendo de las ecuaciones de NS filtradas:

$$\begin{gathered}\nabla\cdot\overline{\mathbf{u}}=0\\ \frac{\partial(\overline{\mathbf{u}})}{\partial t}+\nabla\cdot(\overline{\mathbf{u}}\,\overline{\mathbf{u}})=-\nabla\overline{p}+\nabla\cdot(\nu\nabla\overline{\mathbf{u}})+\nabla\cdot\mathbf{T}\end{gathered}\tag{6.29}$$

donde el término $\mathbf{T}$ surge del filtrado del término no lineal y representa los esfuerzos del modelo SGS y está dado por el tensor $T_{ij}=\overline{u_iu_j}-\overline{u}_i\overline{u}_j$ en notación indicial. El modelo SGS más simple para aproximar dicho término es el de Smagorinsky (1963) [156] que consiste en asumir que las componentes del tensor son proporcionales a los gradientes de deformación:

$$\mathbf{T}=2\nu_t\mathbf{S}_t\tag{6.30}$$

siendo $\mathbf{S}_t$, el tensor de tasa de deformación (*rate of strain*) para la escala resuelta, definido como:

$$\mathbf{S}_t=\frac{1}{2}\left(\nabla\mathbf{u}+\nabla\mathbf{u}^T\right)\tag{6.31}$$

por otro lado, ν_t es la viscosidad turbulenta del modelo SGS que se calcula como $\nu_t=(C_s\Delta)^2\,S_t$, en donde S_t está definido como $S_t=(2\mathbf{S}_t\cdot\mathbf{S}_t)^{1/2}=(2[\mathbf{S}_t]_{ij}[\mathbf{S}_t]_{ij})^{1/2}$, C_s es la constante de Smagorinsky que bajo la hipótesis de turbulencia homogénea isotrópica se utiliza el valor propuesto por Lilly (1967) [101] $C_s=0{,}173$, Δ es la escala de longitud del modelo, que aquí se elige directamente como $\Delta=h$.

La ventaja de este modelo es que a pesar que requiere grillas moderadamente finas, aunque menos que en simulación numérica directa (DNS), es un modelo relativamente sencillo de incorporar al código, la estrategia explícita adoptada es que se computa la viscosidad de remolino ν_t en cada paso de tiempo utilizando los campos de velocidades del tiempo anterior y se incorporan en las ecuaciones de momento redefiniendo $\nu\leftarrow(\nu+\nu_t)$ y calculando los valores en caras mediante la media aritmética o media armónica para un flujo muy heterogéneo.

6.5.1. Ley de pared

En la medida que el flujo se haga más turbulento, la capa límite adyacente a las paredes tiene características diferentes en comparación con el flujo en el seno del fluido. Para describir el flujo cerca de las paredes frecuentemente se utilizan modelos de *ley de pared*.

En este trabajo se eligió una de las leyes de pared más simples, como lo es el modelo propuesto por Spalding (1961) [160], que en coordenadas adimensionales luce como:

$$\begin{gathered}y^+=u^++e^{-\kappa B}\left[e^{\kappa u^+}-1-\kappa u^+-\frac{(\kappa u^+)^2}{2}-\frac{(\kappa u^+)^3}{6}\right]\\ \text{con}\quad u^+=\frac{u}{u_\tau},\quad y^+=\frac{u_\tau y}{\nu},\quad u_\tau=\left(\frac{\tau_w}{\rho}\right)\end{gathered}\tag{6.32}$$

donde τ_w es el esfuerzo de corte en la pared, u_τ es la velocidad de corte o de fricción, ν la viscosidad cinemática, $\kappa=0{,}41$ la constante de von Karman y $B=5{,}5$. Con este modelo, dada una distancia y medida desde la pared en dirección normal y la velocidad tangencial $u=u_t$ en ese

punto, se puede resolver la ecuación no lineal (6.32) vía NR, para obtener u_τ (y en consecuencia también τ_w), en efecto:

$$\begin{aligned} R(u_\tau) &\approx R\left(u_\tau^0\right) + R'\left(u_\tau^0\right)\left(u_\tau - u_\tau^0\right) = 0 \\ \Rightarrow u_\tau^{(k+1)} &= u_\tau^{(k)} - \frac{R(u_\tau^{(k)})}{R'(u_\tau^{(k)})} \end{aligned} \tag{6.33}$$

donde R es la ecuación (6.32) escrita en forma de residuo y R' su derivada:

$$\begin{aligned} R(u_\tau) &= u^+ - y^+ - e^{-B\kappa}\left[\kappa u^+ - e^{\kappa u^+} + \frac{(\kappa u^+)^2}{2} + \frac{(\kappa u^+)^3}{6} + 1\right] \\ R'(u_\tau) &= \frac{dR}{du_\tau} = e^{-\kappa B}\left(\frac{\kappa u^+}{u_\tau} + \frac{(\kappa u^+)^2}{u_\tau} + \frac{(\kappa u^+)^3}{2u_\tau} - \frac{\kappa u^+ e^{\kappa u^+}}{u_\tau}\right) - \frac{u^+}{u_\tau} - \frac{y}{\nu} \end{aligned} \tag{6.34}$$

Cabe mencionar que se utilizan u^+ y u_τ en la misma fórmula por cuestiones de implementación, es decir, en la iteración NR se mantienen fijos y, u y se computa u_τ, con ello se actualiza u^+ para la nueva iteración NR.

La ley propuesta por Spalding puede incorporarse al método de fronteras embebidas descripto en el capítulo 5 con las siguientes modificaciones. Considere el esquema 2D por simplicidad presentado en la figura 6.2. La idea es imponer una condición de contorno sobre el sólido similar a una condición de resbalamiento puro (*slip condition*), esto significa que se aplica una BC tipo Dirichlet homogénea para la componente normal de la velocidad $\mathbf{u_n}$ (condición de no penetrabilidad) y BC tipo Neumann para la componente tangencial de la velocidad $\mathbf{u_t}$ que involucre el esfuerzo de corte en la pared τ_w.

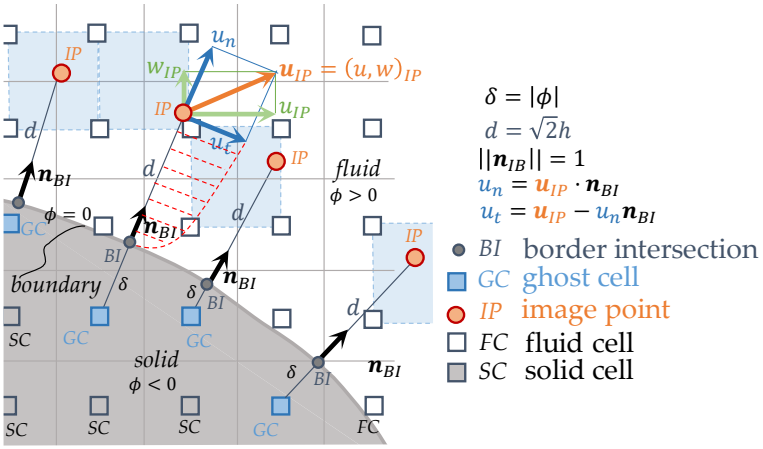

Figura 6.2: Esquema propuesto para imponer una ley de pared utilizando el método GC-IBM.

Si se considera el punto imagen a una distancia δ como se desarrolló antes, al momento de resolver (6.33) puede ocurrir que los valores y^+ en el punto imagen (IP) den muy distintos de una celda a otra debido a que dichos puntos se distribuyen arbitrariamente según la ubicación del nodo GC respecto de la superficie, luego el punto a distancia $y = \delta$ se ubicará en diferentes lugares del perfil teórico. Si bien una de las ventajas de la ley de Spalding es que se tiene un único perfil de velocidades, de aplicar otra ley se estaría utilizando la parte lineal del perfil, la región de transisión o la región logarítmica del perfil. Puede ocurrir además que el punto IP quede muy cerca de la superficie y generar inestabilidades. Para ser consistentes al momento de estimar u_τ resolviendo (6.33) se elige un punto que se ubique en la región fluida de manera que en la interpolación, no se utilice el punto BI, esto se garantiza eligiendo $d = \sqrt{2}h$ ($d = \sqrt{3}h$ en 3D).

Una vez determinado IP de forma similar a como se desarrolló en el método GC-IBM, ubicado a una distancia $\delta + d$ del punto GC y a una distancia d desde la intersección con el borde BI. Para

aplicar las BCs utilizando una ley de pared para la componente tangencial, en el nodo fantasma debe imponerse:

$$\begin{aligned}\mathbf{u}_{\mathbf{n}}^{\mathrm{GC}} &= -\frac{\delta}{d}\mathbf{u}_{\mathbf{n}}^{\mathrm{IP}} = -\frac{|\phi_{\mathrm{GC}}|}{d}\mathbf{u}_{\mathbf{n}}^{\mathrm{IP}} \\ \mathbf{u}_{\mathbf{t}}^{\mathrm{GC}} &= \mathbf{u}_{\mathbf{t}}^{\mathrm{IP}} - \frac{(\delta+d)}{(\mu+\mu_t)^{\mathrm{IP}}}\tau_w \mathbf{t}^{\mathrm{BI}}\end{aligned} \tag{6.35}$$

donde τ_w se obtiene iterativamente con la ecuación 6.33, $(\mu+\mu_t)$ es la viscosidad dinámica efectiva que incluye la viscosidad molecular y la turbulenta (el valor en el nodo IP se obtiene por interpolación). Por otro lado, geométricamente puede descomponerse $\mathbf{u}^{\mathrm{IP}} = \mathbf{u}_{\mathbf{n}}^{\mathrm{IP}} + \mathbf{u}_{\mathbf{t}}^{\mathrm{IP}}$ como se indica en la figura 6.2:

$$\begin{aligned}\mathbf{u}_{\mathbf{n}}^{\mathrm{IP}} &= \left(\mathbf{u}^{\mathrm{IP}} \cdot \mathbf{n}^{\mathrm{BI}}\right)\mathbf{n}^{\mathrm{BI}} \\ \mathbf{u}_{\mathbf{t}}^{\mathrm{IP}} &= \mathbf{u}^{\mathrm{IP}} - \left(\mathbf{u}^{\mathrm{IP}} \cdot \mathbf{n}^{\mathrm{BI}}\right)\mathbf{n}^{\mathrm{BI}}\end{aligned} \tag{6.36}$$

reemplazando (6.36) en (6.35) y teniendo en cuenta que $\mathbf{t}^{\mathrm{BI}} = \mathbf{u}_{\mathbf{t}}^{\mathrm{IP}}/||\mathbf{u}_{\mathbf{t}}^{\mathrm{IP}}||$ se obtiene:

$$\mathbf{u}^{\mathrm{GC}} = \left[\underbrace{1 - \frac{(\delta+d)\,\tau_w}{(\mu+\mu_t)^{\mathrm{IP}}\,||\mathbf{u}_{\mathbf{t}}^{\mathrm{IP}}||}}_{\mathrm{I}}\right]\mathbf{u}^{\mathrm{IP}} - \left(\mathbf{u}^{\mathrm{IP}} \cdot \mathbf{n}^{\mathrm{BI}}\right)\mathbf{n}^{\mathrm{BI}}\left\{\underbrace{\frac{\delta}{d} + 1 + \frac{(\delta+d)\,\tau_w}{(\mu+\mu_t)^{\mathrm{IP}}\,||\mathbf{u}_{\mathbf{t}}^{\mathrm{IP}}||}}_{\mathrm{II}}\right\} \tag{6.37}$$

entonces, por ejemplo para el caso de la componente x de la velocidad en el caso 3D luce como:

$$u^{\mathrm{GC}} = [I]\,u^{\mathrm{IP}} - \left(u^{\mathrm{IP}}n_x + v^{\mathrm{IP}}n_y + w^{\mathrm{IP}}n_z\right)n_x\,\{II\} \tag{6.38}$$

en donde se utilizaron I y II para indicar las expresiones entre paréntesis rectos y llaves de la ecuación (6.37). Re ordenando y agrupando, las expresiones para las 3 ecuaciones de momento quedan:

$$\begin{aligned}u^{\mathrm{GC}} &= \underbrace{u^{\mathrm{IP}}\left([I] - n_x^2\,\{II\}\right)}_{\text{implicit}} - \underbrace{n_x\left(v^{\mathrm{IP}}n_y + w^{\mathrm{IP}}n_z\right)\{II\}}_{\text{explicit}} \\ v^{\mathrm{GC}} &= \underbrace{v^{\mathrm{IP}}\left([I] - n_y^2\,\{II\}\right)}_{\text{implicit}} - \underbrace{n_y\left(u^{\mathrm{IP}}n_x + w^{\mathrm{IP}}n_z\right)\{II\}}_{\text{explicit}} \\ w^{\mathrm{GC}} &= \underbrace{w^{\mathrm{IP}}\left([I] - n_z^2\,\{II\}\right)}_{\text{implicit}} - \underbrace{n_z\left(u^{\mathrm{IP}}n_x + v^{\mathrm{IP}}n_y\right)\{II\}}_{\text{explicit}}\end{aligned} \tag{6.39}$$

Considerando pasos de tiempo pequeños en el FSM al resolver las ecuaciones de NS y el esquema temporal AB para el término advectivo, se logran resolver las ecuaciones de momento separadas por cada dirección. El tratamiento que se le da a la condición de borde desarrollada, es implícito para el término que involucra la variable que se esta resolviendo (en este ejemplo, la componente u). El término restante, que involucra las demás variables, se lo incorpora explícitamente en el lado derecho de la ecuación, utilizando los valores a tiempo anterior disponibles. No obstante se le podría dar a todos los términos un tratamiento implícito, mediante un proceso iterativo con las 3 ecuaciones de momento.

Para resumir, la estrategia en GPU para aplicar la ley de pared es que si el hilo está procesando un nodo GC, evalúa $\delta = |\phi(\mathbf{x}_{\mathrm{GC}})|$, con ello calcula las coordenadas $\mathbf{x}_{\mathrm{IP}}$ e interpola en el nodo IP el valor de u, v, w y $\mu_t = \rho\nu_t$, luego con los valores fijos de d y $u_t = ||\mathbf{u}_t^{\mathrm{IP}}||$ compute mediante NR (ecuación (6.33)) la velocidad de fricción u_τ que permite hallar $\tau_w = \rho u_\tau^2$. Con los valores calculados, se computa el lado de derecho de las ecuaciones de momento en las tres direcciones (parte explícita de la ecuación (6.39)), mediante el uso de variables auxiliares se guardan algunos valores para ser usados durante el cómputo de la operación tipo stencil (parte implícita de la ecuación (6.39)) al resolver el sistema lineal para cada componente de velocidad.

La técnica para aplicar la ley de pared descrita en esta sección está inspirada a partir de métodos desarrollados también para flujos compresibles, como se presenta en los siguientes trabajos [75, 137, 157, 158].

6.6. Estrategia para estimación de esfuerzos de corte en el lecho

Hay diferentes técnicas para calcular los esfuerzos de corte $\boldsymbol{\tau}_b$ requeridos para estimar la tasa de transporte de fondo $\mathbf{q}_b$. En [7] se describen una lista algo extensa de algunas técnicas implementadas en el software REEF3D [6], entre las que pueden encontrarse formulaciones basadas en el esfuerzo cortante promedio en el lecho [14], en la ley del esfuerzo cuadrático [151], en el factor de fricción f, en leyes de pared (*wall function*), basadas en las tensiones de Reynolds [142], en la energía cinética turbulenta y en la viscosidad turbulenta ν_t y la velocidad tangencial [190].

Para esta tesis se implementó una técnica basada en la ley de pared similar a como se describió antes, con las siguientes diferencias. Se computa el esfuerzo de corte $\boldsymbol{\tau}_b$ para las celdas fluidas adyacentes al sólido, utilizando la función SDF se obtiene el punto BI (ubicado a una distancia $\delta = \phi_C$) a partir de donde se computa un punto IP a una distancia d como antes, la idea es computar u_τ mediante la ley de pared, utilizando d y $||\mathbf{u}_\mathbf{t}^{\text{IP}}||$ y con ello estimar τ_w, luego, con las componentes u_C y v_C del vector $\mathbf{u}_C$ se reparte τ_w de la siguiente forma:

$$\boldsymbol{\tau}_b = (\tau_{b,x}, \tau_{b,y}) = \frac{\tau_w}{U}(u_C, v_C), \quad \text{con } U = \sqrt{u_C^2 + v_C^2} \tag{6.40}$$

6.7. Modelos de deslizamiento de granos

En el proceso de transporte de sedimentos a veces se presentan formas de fondo poco realistas cuando las pendientes superan el ángulo de reposo del medio granular. Los taludes inestables naturalmente son propensos a deslizamientos hasta alcanzar una pendiente estable [192]. Para tratar ello, en la literatura pueden encontrarse diferentes técnicas, por ejemplo [29, 67, 83, 100, 147, 159]. Uno de los requisitos importantes a cumplir es la conservación de masa, sin embargo esto se logra al aplicar FVM, otro requisito deseable es que el algoritmo se base en un proceso físico y provea una única solución [159].

El método implementado para esta tesis se basa en la ecuación de difusión para el nivel de la superficie granular, que utiliza los gradientes dados por las diferencias de alturas de la superficie del lecho. Una variante del método puede encontrarse en [28] y otra variante extendida a grillas no estructuradas puede encontrarse en [159].

Los cambios en las pendientes físicamente se deben a la gravitación, de acuerdo a ello el flujo $\mathbf{q}$ que desliza es proporcional al gradiente:

$$\boldsymbol{\Gamma} = -\kappa \nabla \xi \tag{6.41}$$

donde $\boldsymbol{\Gamma}$ está expresado en [m^3/s], ξ [m] es la altura del nivel del lecho medida desde alguna referencia y κ [m^2/s] representa la difusividad de la pendiente de deslizamiento. La ley de conservación para el nivel sin considerar transporte por advección, es decir considerando sólo la difusión por el efecto de la gravedad viene dada por:

$$\begin{aligned} \frac{\partial \xi}{\partial t} + \nabla \cdot \boldsymbol{\Gamma} &= 0 \\ \frac{\partial \xi}{\partial t} - \nabla \cdot (\kappa \nabla \xi) &= 0 \end{aligned} \tag{6.42}$$

donde $\xi(\mathbf{x}, t)$ es el nivel de la superficie para un tiempo artificial dado. En la implementación se considera un coeficiente de difusión constante dado por:

$$\kappa(\nabla \xi) = \begin{cases} \kappa_0, & \text{si } \alpha \geq \alpha_C \\ 0, & \text{si } \alpha < \alpha_C \end{cases} \tag{6.43}$$

aquí α_C representa el ángulo de reposo crítico del medio granular y α el ángulo de inclinación del lecho dado por $\alpha = \tan^{-1}(||\nabla\xi||)$, debido a que la norma (euclídea) del gradiente de ξ es una medida de la pendiente de lecho, esto es $\tan(\alpha) = ||\nabla\xi||$. Notar que se ha explicitado la dependencia de κ con el gradiente de ξ, lo que constituye una no linealidad en la ecuación diferencial (6.42).

En esta tesis se considera el coeficiente de difusión constante κ_0 entre 0.5 y 1.0, este valor debe ser elegido adecuadamente para que en el proceso de solución numérica de la ecuación 6.42, la corrección no sea excesiva (sobre-corrección) y a su vez evitar que se requieran muchas iteraciones a lo largo del tiempo artificial para llegar a una solución que cumpla la condición $\alpha_i \leq \alpha_C$ $(i = 1, ..., N_x N_y)$. No obstante, es razonable considerar κ_0 proporcional a la tasa de transporte de fondo, ya que es un indicador del grado de actividad del cambio morfológico que se está dando [159].

6.7.1. Discretización de la ecuación para el modelo de deslizamiento

Se resuelve numéricamente la ecuación 6.42 en 2D mediante el método FVM. Al tratarse de una ecuación no lineal y no contener una fórmula analítica de $\kappa(\nabla\xi)$ para aplicar el método NR, se elige resolverla utilizando el método explícito FE para la discretización temporal, además como este esquema se aplica en un tiempo artificial se utilizará la notación con superíndice (k) para indicar la iteración en el k-ésimo paso, es decir para un tiempo artificial $t_k = k\Delta t$. Integrando 6.42 sobre la celda C, aplicando el teorema de la divergencia y del valor medio se puede escribir:

$$\left(\frac{\xi_C^{(k+1)} - \xi_C^{(k)}}{\Delta t}\right) V_C = \sum_{f\sim nb(C)} \kappa_f^{(k)} \nabla\xi_f^{(k)} \cdot \mathbf{S}_f \tag{6.44}$$

aplicando un esquema centrado y considerando la grilla cartesiana uniforme

$$\xi_C^{(k+1)} = \xi_C^{(k)} + \frac{\Delta t}{h^2} \sum_{F\sim NB(C)} \kappa_f^{(k)} \left(\frac{\xi_F^{(k)} - \xi_C^{(k)}}{h}\right) h \tag{6.45}$$

Se requiere determinar $\kappa_f^{(k)}$ para los centros de cara, considerando (6.43) estos valores dependen del ángulo de inclinación que está dado por:

$$\phi_f = \tan^{-1}(||\nabla\xi_f||) = \tan^{-1}\sqrt{\left(\frac{\partial\xi}{\partial x}\right)_f^2 + \left(\frac{\partial\xi}{\partial y}\right)_f^2} \tag{6.46}$$

Entonces, por ejemplo para la cara *east* la derivada parcial x se computa de forma local como es usual:

$$\left(\frac{\partial\xi}{\partial x}\right)_e = \frac{\xi_E - \xi_C}{h} \tag{6.47}$$

mientras que la derivada parcial y en la cara e debe calcularse como el promedio entre $(\partial\xi/\partial y)_C$ y $(\partial\xi/\partial y)_E$:

$$\left(\frac{\partial\xi}{\partial y}\right)_e = \frac{1}{2}\left(\frac{\xi_{NE} - \phi_{SE}}{2h} + \frac{\xi_N - \phi_S}{2h}\right) = \frac{\xi_{NE} + \xi_N - \xi_{SE} - \xi_S}{4h} \tag{6.48}$$

usando una fórmula análoga para obtener alguna de las componentes del gradiente en las demás caras $(\partial\xi/\partial y)_w$, $(\partial\xi/\partial x)_n$ y $(\partial\xi/\partial x)_s$, en las cuales no se puede computar la derivada en forma local (ecuación (6.47)).

La estrategia explícita entonces, es que la solución para cada paso de tiempo se realiza mediante el algoritmo 17.

Algoritmo 17: Estrategia de cómputo explícita para cada paso de tiempo en el método limitador de pendientes.

1 computar $\nabla\xi_e^{(k)}$, $\nabla\xi_w^{(k)}$, $\nabla\xi_n^{(k)}$ y $\nabla\xi_s^{(k)}$ utilizando (6.47) y (6.48);
2 calcular $\alpha_f = \tan^{-1}\left(||\nabla\xi_f^{(k)}||\right)$, $f \sim \{e, w, n, s\}$;
3 definir $\kappa_f = \kappa_0$ si $\alpha_f \geq \alpha_C$ o $\kappa_f = 0$ si $\alpha_f < \phi_C$, para $f \sim \{e, w, n, s\}$;
4 computar $\xi_C^{(k+1)}$ usando (6.45);

Las condiciones de contorno utilizadas en la ecuación son del tipo Neumann homogéneas sobre todos los contornos (sean embebidos o no), es decir $\partial\xi/\partial\mathbf{n} = 0$. Finalmente, el criterio de parada elegido para el limitador de pendientes basado en la ecuación de difusión es que $||\boldsymbol{\xi}^{(k+1)} - \boldsymbol{\xi}^{(k)}||_\infty >$ tol, con una tolerancia tol $= 10^{-10}$. En general pocas iteraciones bastan para que en todo el dominio ($i = 1, ..., N_x N_y$) se satisfaga que $\alpha_i \leq \alpha_C$.

La condición de estabilidad para el paso de tiempo artificial es $\Delta t/h^2 \leq 1/(2\kappa_0)$ [159], sin embargo, para evitar la sobre corrección de las pendientes del dominio, se optó por elegir un paso de tiempo (en general menor) dado por $\Delta t = h^3$ como recomienda Burkow [28].

En cuanto a la precisión temporal del método, en ambos casos el error temporal es menor que el error del esquema espacial, sin embargo al alcanzarse el estado estacionario en la ecuación la solución debería ser independiente del valor Δt elegido.

6.7.2. Validación del método de deslizamiento

Al aplicar el algoritmo no se requiere probar la conservación de masa debido a que, por construcción, el método es conservativo.

Para validar el método se resuelve el problema de colapso de una columna de arena bajo condiciones de no flujo, es decir solamente bajo efectos de la gravedad. Diversos experimentos confirman que para este problema, la solución tiene la forma de un pila de cono [105, 191, 192]. La figura 6.3 muestra las condiciones iniciales para el problema, una columna de altura $h = 6$cm y diámetro $d = 10$cm, el material tiene un ángulo de reposo de $\alpha_C = 35°$.

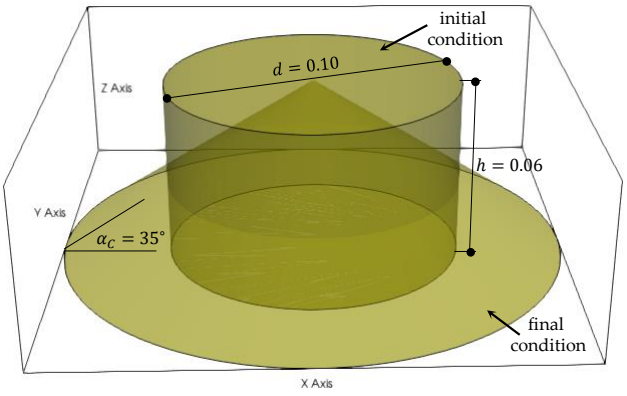

Figura 6.3: Esquema para el test del colapso de una pila de arena, condición inicial y condición final esperada, dimensiones en [m].

El dominio computacional es un cuadrado de $[0, L] \times [0, L]$, con $L = 0{,}8$m donde se ubica la pila en el centro $(0{,}4, 0{,}4)$. El problema se discretizó con una grilla de 160×160 celdas de manera que $\Delta x = \Delta y = 0{,}005$m para poder comparar la solución con el trabajo de otros autores. En la figura 6.4 (izquierda) se muestra la solución numérica obtenida y a modo ilustrativo se presenta la imagen extraída de los experimentos de Lube et al. [105] para problemas de colapso de columnas de material granular. Puede observarse que la solución numérica luce similar la condición final esperada.

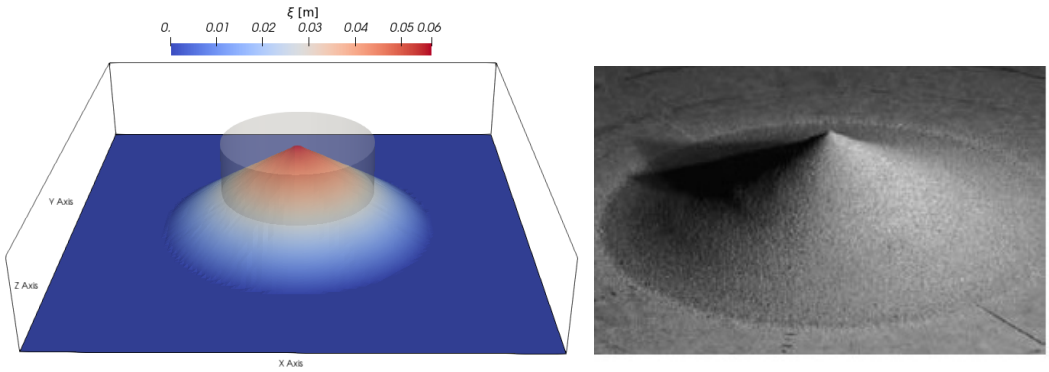

Figura 6.4: Solución numérica obtenida con la ecuación (6.45) para el problema del colapso de una columna de arena (izquierda). Ilustración de una prueba experimental extraída del trabajo de Lube et al. [105].

En la figura 6.5 se presenta el perfil a lo largo del centro de la pila, los valores de la norma del gradiente y las curvas de nivel de la solución final, se comparan además con la solución obtenida por [159] utilizando un método similar en una grilla rectangular de las mismas características pero con otro esquema de discretización espacial. En cuando a la pendiente de reposo, el valor teórico es de $\tan(35°) \approx 0{,}70$ mientras que la pendiente de la solución numérica dada por $S_0 = ||\nabla\xi||$, brinda valores inferiores pero muy próximos al teórico (figura 6.5 arriba-derecha). Los valores son acordes a los obtenidos en [159].

6.8. Re-mapeo del fondo móvil. Actualización de la función marcadora

A partir del los cambios en el nivel del lecho $\xi(\mathbf{x}, t)$ dados por la ecuación de Exner ((6.14) o (6.15)) y la aplicación del modelo de deslizamiento de granos, algoritmo 17, se debe actualizar la función SDF o $\phi(\mathbf{x}, t)$ para aplicar las técnicas de fronteras embebidas en el paso de tiempo siguiente. Para actualizar esta función marcadora, muchos trabajos emplean técnicas LS, por ejemplo [8, 86], mediante una ecuación de movimiento para la interfaz:

$$\frac{\partial\phi}{\partial t} + F||\nabla\phi|| = 0 \tag{6.49}$$

donde definen la velocidad de movimiento como $F = \partial\xi/\partial t$, correspondiente a la dirección vertical ya que $\Delta\xi = \Delta z$. El lado derecho de la ecuación (6.15) se utiliza como velocidad vertical F. Con ello advectan la función SDF ϕ y realizan un proceso de reinicialización para quitar la difusión numérica y mantener las propiedades deseadas.

Para esta tesis se siguió el procedimiento propuesto por Burkow (2016) [28] que consiste en realizar una estimación inicial de la función SDF ϕ_0 para luego pasar al proceso de reinicialización de la función ϕ. La estimación se basa en calcular para cada punto $\mathbf{x}_C = (x_C, y_C, z)$ la mínima distancia entre él y la superficie $\phi(\mathbf{x}, t) = 0$ que corresponde al conjunto $\Gamma_{\text{sed}} = \{\mathbf{x} \in \mathbb{R}^3 / z(x, y) = \xi\}$, dando el signo según el punto se ubique por encima o por debajo del lecho y donde el mínimo se realiza en una banda bidireccional de ancho 3 como muestra la figura 6.6:

$$\begin{gathered}\phi_0(\mathbf{x}_C) = \text{sgn}(z - \xi_C) \min_i \{d_i\} \\ \text{con } d_i = ||\mathbf{x}_C - \xi(\mathbf{x}_i)||, \ \{i = C, E, W, N, S, NE, SE, NW, SW\}\end{gathered} \tag{6.50}$$

Si bien, para puntos lejos del lecho la estimación podría alejarse de la distancia real a la superficie, en la medida que el punto esté más cerca de la superficie, la estimación de la ubicación de la superficie dada por el conjunto $\phi_0 = 0$ es correcta (en su versión discreta).

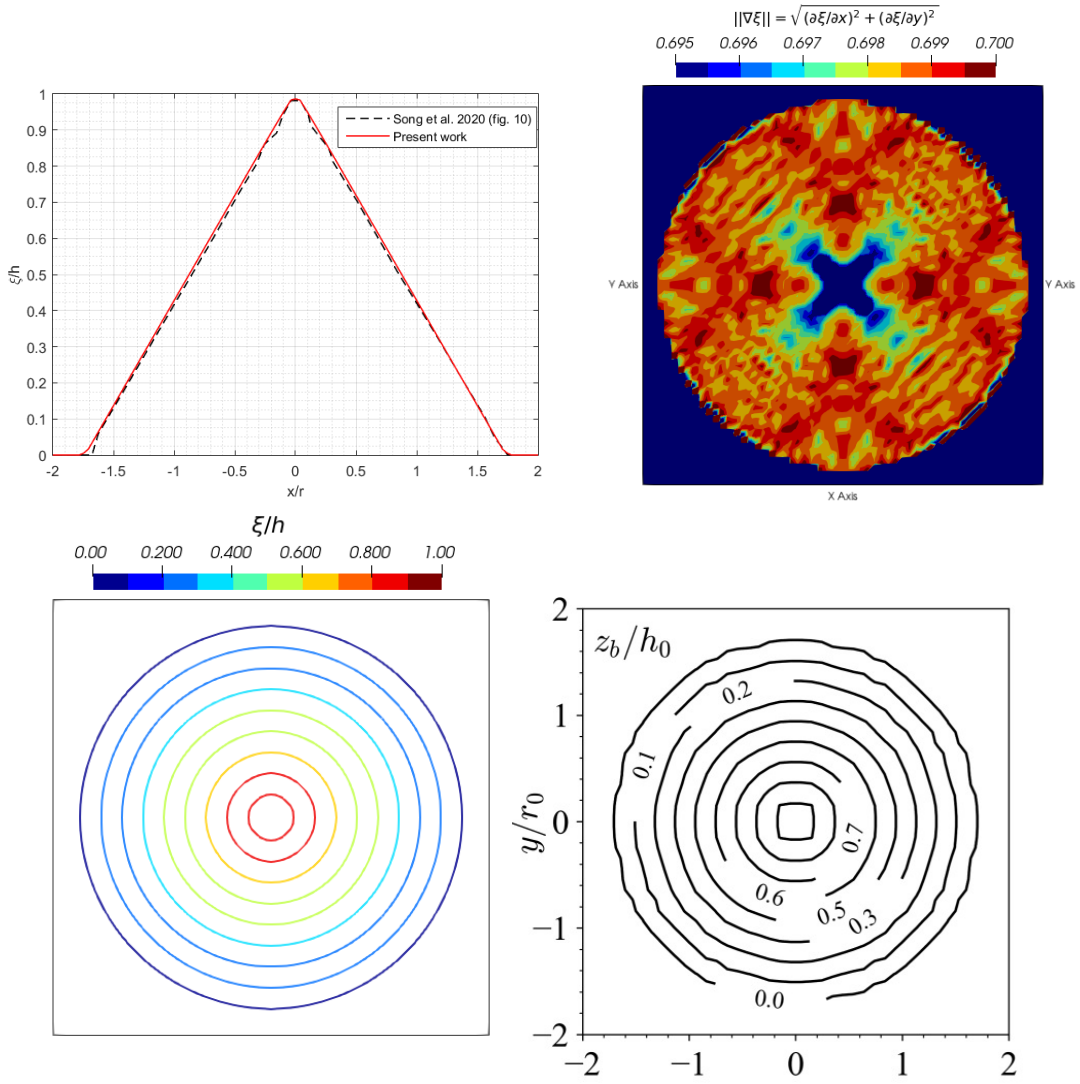

Figura 6.5: Perfil final de la pila (arriba-izquierda), valores de la norma del gradiente (arriba-derecha) y curvas de nivel del la solución numérica (abajo-izquierda). Comparación con los resultados obtenidos por Song et al. [159].

El paso siguiente es realizar el proceso de reinicialización de la SDF explicado en el capítulo 5, partiendo de la estimación inicial, esto corregirá la función ϕ para todo el dominio sin modificar la estimación inicial $\phi_0 = 0$.

Este enfoque tiene prácticamente el mismo costo que la operación de advectar el campo ϕ (ecuación (6.49)) y ambos métodos proveen casi la misma solución.

6.9. Acoplamiento morfodinámico e hidrodinámico

El acoplamiento entre el modelo hidrodinámico (ecuaciones de NS) y el modelo morfodinámico (ecuaciones para el transporte de sedimentos y evolución del lecho) se trata como un problema de interacción fluido-estructura (fluido-sedimento), para lo cual existen diferentes enfoques, los dos grandes tipos son: acoplamiento débil o explícito y acoplamiento fuerte o implícito [57]. En la mayoría de los trabajos analizados se prefiere el algoritmo de particionamiento explícito por el menor costo computacional y porque permite incorporar las rutinas del sedimento en un código preexistente. Además, aunque este enfoque es más propenso a presentar inestabilidades, las escalas temporales de cambio en el dominio son mucho más grandes que la escala temporal de cambios en el flujo incompresible.

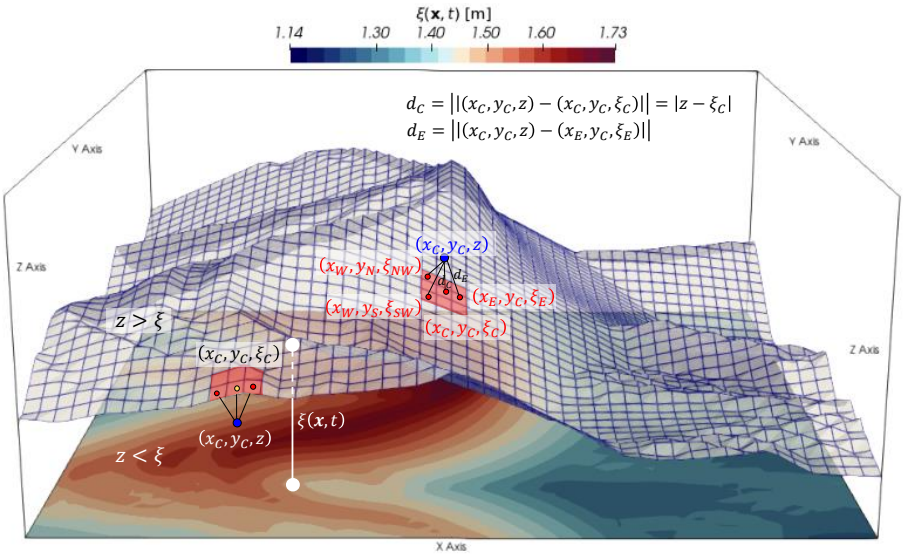

Figura 6.6: Esquema de cálculo la SDF ϕ_0 a partir de la superficie del lecho $z = \xi(x, y)$.

El algoritmo de particionamiento explícito consiste en resolver de manera independiente las ecuaciones del flujo (NS) en un dominio fijo, luego con la solución del campo de velocidades, computar los esfuerzos de corte en el lecho, las tasas de transporte, resolver la ecuación de advección difusión para el transporte en suspensión, resolver la ecuación de Exner y finalmente actualizar el dominio computacional para avanzar al siguiente paso de tiempo.

La evolución del lecho (erosión y deposición de sedimentos) es sensible a las variaciones del flujo, es decir, ambos procesos están interconectados y debería utilizarse el mismo paso de tiempo para los modelos hidrodinámico y morfodinámico. Sin embargo, esta estrategia de acoplamiento puede resultar costosa. Por otro lado, mientras que el sedimento en suspensión y el flujo tienen la misma escala temporal de cambios, la escala temporal de cambios en el nivel del lecho es mucho mayor, por ello se propone un enfoque desacoplado en el cual la evolución del lecho se calcula utilizando un paso de tiempo Δt_{sed} mucho más grande que el paso de tiempo elegido para las ecuaciones del flujo Δt. Estrategia que también puede utilizarse en el caso de que la concentración de sedimentos en suspensión sea muy baja como para alterar las condiciones del flujo formando así un proceso de retrolimentación o acoplamiento.

El algoritmo 18 presenta el enfoque de interacción entre el flujo y lecho propuestos para esta tesis, en el mismo se elige $\Delta t_{\text{sed}} = n_{\text{sed}} \Delta t$, con $n_{\text{sed}} = 10$, a modo de ejemplo, el manual del software REEF3D [6] indica que el valor por defecto que utiliza es $n_{\text{sed}} = 20$.

No obstante, pruebas realizadas utilizando $\Delta t_{\text{sed}} = \Delta t$ muestran soluciones muy similares a las obtenidas usando $\Delta t_{\text{sed}} = n_{\text{sed}} \Delta t$, con $n_{\text{sed}} = 5, 10, 20$, según la intensidad de cambios en el lecho y sin violar el paso de tiempo máximo según criterios de estabilidad en la solución de la ecuación de Exner. Por otro lado, utilizar valores promedio del esfuerzo de corte $\overline{\tau}_b$ proporciona una simulación más estable y con transiciones más suaves, a modo de ejemplo, en la figura 6.7 se presenta en vista de planta los esfuerzos de corte promedio para el caso de erosión alrededor de un objeto prismático rectangular recto.

6.9.1. Problema de cavidad rectangular con sedimento

Esta prueba consiste en evaluar el algoritmo completo en un caso teórico simple bidimensional, el problema se trata de una cavidad rectangular de dimensiones $2L \times L$ con tapa móvil de manera que se genera un flujo a $R_e = 100$. En el dominio se incorpora una frontera inferior erosionable a un nivel $L/4$ desde el fondo. El objetivo es verificar cualitativamente el solver considerando solamente transporte de fondo y a su vez evaluar el efecto del modelo de deslizamiento de granos

Algoritmo 18: Acoplamiento temporal débil para el solver de NS con el modelo de sedimentos.

$\mathbf{u}^n \leftarrow \mathbf{u}^0$,$p^n \leftarrow p^0$, $\xi^n \leftarrow \xi^0$, $\phi^n \leftarrow \phi^0$, move_bed $\leftarrow false$, $n_{\text{sed}} \leftarrow 10$, $j \leftarrow 0$;
mientras *($t < T_{final}$)* **hacer**
  si *($move\_bed == true$)* **entonces**
    calcular nuevo dominio ϕ^{n+1} (estimación inicial ϕ_0 y reinicialización);
  fin
  calcular viscosidad turbulenta ν_t^{n+1} ;
  resolver $\mathbf{u}^{n+1}$, p^{n+1} (FSM con LES y GC-IBM);
  calcular E_b, D_b;
  resolver C^{n+1} (sedimento en suspensión);
  si *($t \leq T_{sed,init} - n_{sed}\Delta t$)* **entonces**
    computar $\boldsymbol{\tau}_b$;
    acumular $\hat{\boldsymbol{\tau}} = \hat{\boldsymbol{\tau}} + \boldsymbol{\tau}_b$;
  fin
  si *(($t \geq T_{sed,init}$) & ($j \,\%n_{sed} == 0$))* **entonces**
    calcular esfuerzo promedio $\overline{\boldsymbol{\tau}}_b \leftarrow \hat{\boldsymbol{\tau}}/n_{\text{sed}}$;
    calcular $\mathbf{q}_b$, ;
    resolver ξ^{n+1};
    corregir ξ^{n+1} con algoritmo de deslizamiento de granos;
    move_bed $\leftarrow true$;
    $\hat{\boldsymbol{\tau}} \leftarrow 0$
  fin
  en otro caso
    move_bed $\leftarrow false$;
  fin
  $j \leftarrow j + 1$;
  $t \leftarrow t + \Delta t$;
fin

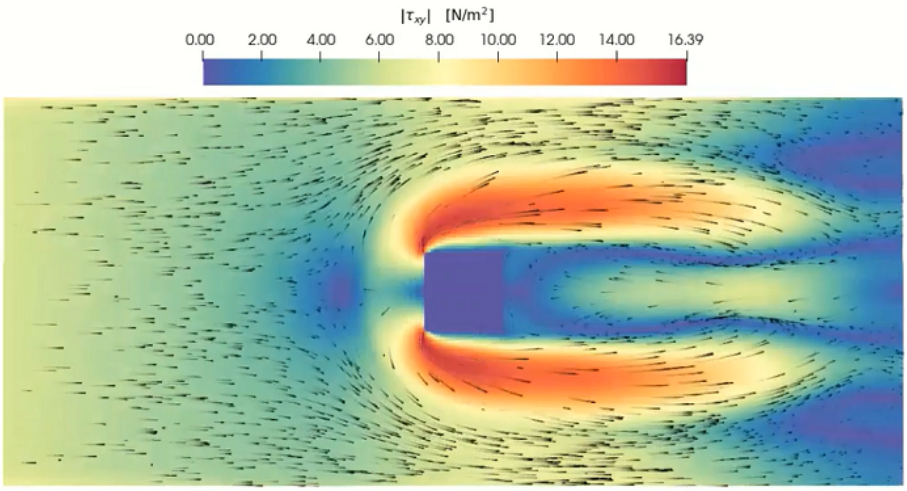

Figura 6.7: Esfuerzos de corte $\overline{\tau}_b$ promediados durante un paso de tiempo Δt_{sed} del modelo morfodinámico.

dentro del resolvedor.

Los parámetros del modelo hidrodinámico para la simulación son $L = 0{,}5$m, $\rho = 1000$kg/m^3, $\mu = 3{,}75$kg/m/s, $\Delta t = 2 \times 10^{-3}$s, $T_{\text{final}} = 248$s, grilla computacional $N_x \times N_y = 256 \times 128$. Parámetros del modelo morfodinámico $\rho_s = 2650$kg/m^3, $d_{50} = 1$mm, $n_{\text{sed}} = 20$, $\Delta t_{\text{sed}} = 4 \times 10^{-2}$s, fórmula para el transporte de fondo MPM. Modelo de deslizamiento de granos: ángulo de reposo α_C 30° y 40°, $\tau_c = 0{,}047$, $\kappa_0 = 0{,}5$.

El esquema de discretización espacial para el resolvedor basado en FSM es CD (término advectivo y difusivo) y los esquemas temporales CN (difusión) y AB (advección). Esquema de discretización temporal AB y esquema espacial de alta resolución MINMOD para la ecuación de Exner.

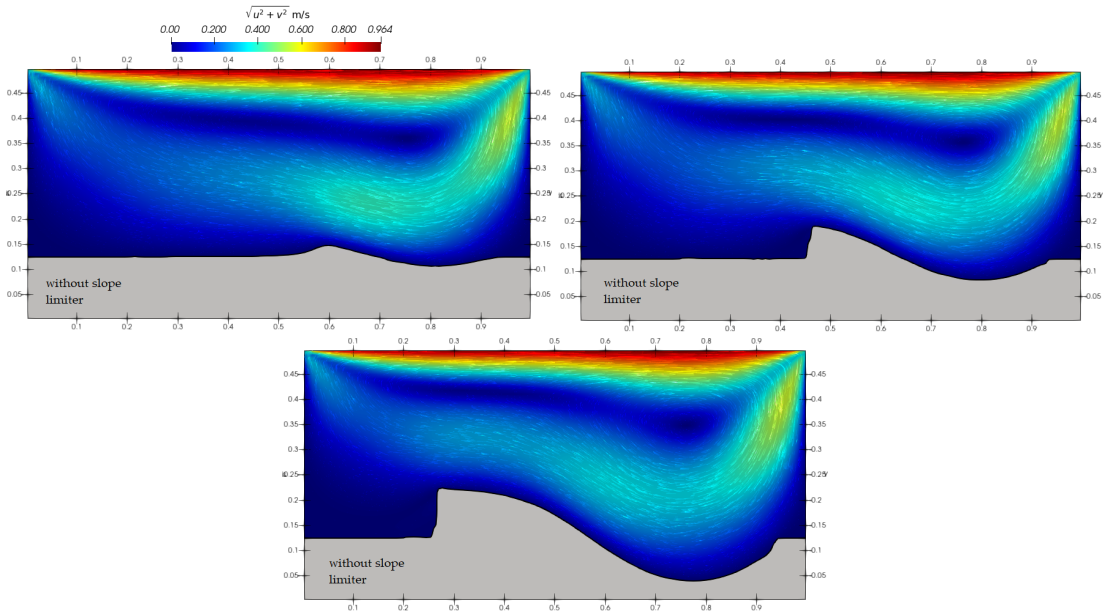

Figura 6.8: Resultados de la prueba de la cavidad cuadrada con sedimento para $Re = 100$ en instantes $t = 10$s, $t = 40$s y $t = 204$s, sin utilizar el modelo de deslizamiento de granos.

Las figuras 6.8 y 6.9 muestran la evolución de la simulación para diferentes instantes de tiempo sin utilizar el modelo de deslizamiento de granos y utilizándolo con ángulos de reposo de 30° y 40°. Puede apreciarse que el comportamiento del solver es el mismo siempre que no se superen las pendientes que indica el ángulo de reposo α_C, obteniéndose profundidades de erosión similares.

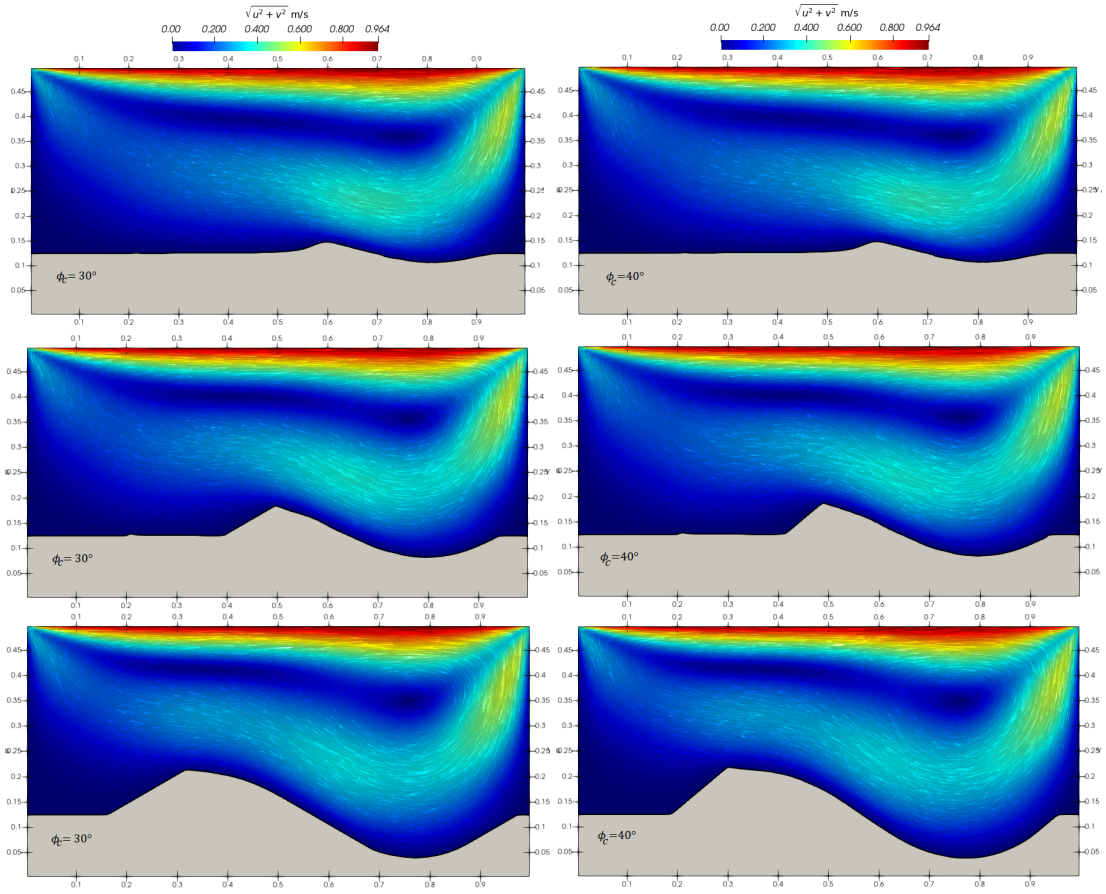

Figura 6.9: Prueba de la cavidad cuadrada con sedimento para $Re = 100$ en instantes $t = 10$s, $t = 40$s y $t = 204$s, utilizando el modelo de deslizamiento de granos con ángulos de reposo de 30° (izquierda) y 40° (derecha).

Los cambios comienzan al superarse dicha pendiente máxima, por otro lado, como el método de deslizamiento se basa en la ecuación de difusión, puede verse como la masa que excede el ángulo se reparte hacia ambos lados desde el interior del talud. Cabe mencionar que a pesar que en la solución de la ecuación de Exner analíticamente se podría producir el fenómeno de rompimiento o pliege hacia adelante de la duna (intersección de las características). La estrategia propuesta para el re-mapeo del fondo móvil adoptada no permite el solapamiento de niveles, por ello el talud de transporta de forma vertical, no obstante el algoritmo de deslizamiento no permite que se originen esos resultados no físicos en la superficie del sedimento.

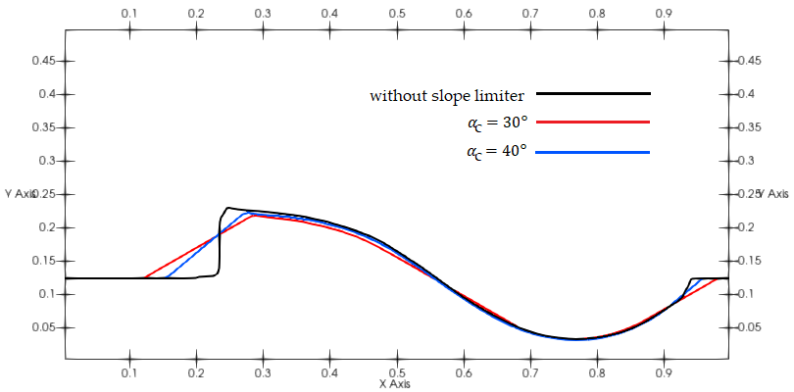

Figura 6.10: Comparación de los perfiles finales del lecho para la prueba de la cavidad cuadrada con sedimento a $Re = 100$, instante $t = 248$s, utilizando el modelo de deslizamiento de granos con ángulos de reposo de 30° (rojo), 40° (azul) y sin usar el modelo de deslizamiento (negro).

6.10. Caso de estudio: erosión alrededor de un obstáculo rectangular

El objetivo de esta prueba es reproducir la huella de erosión que se produce alrededor de un prisma rectangular recto embebido en una corriente liquida y sobre un lecho erosionable, las complejas estructuras de vórtices alrededor del objeto y validar los resultados a partir de la comparación con los de otros trabajos.

6.10.1. Escenario experimental de la simulación

En la figura 6.11 se presenta la configuración del experimento numérico, el escenario consiste en un canal rectangular de $40L \times 5L \times 5L$ (largo $\times$ ancho $\times$ alto), con $L = 1$m. La grilla computacional es de $1024 \times 128 \times 128 \approx 16{,}7$ Millones de celdas. El canal cuenta con un lecho colocado a un nivel de $1{,}25L$ de forma que la profundidad del agua es de $3{,}75L$. Se coloca un obstáculo de $L \times L \times 2{,}5L$ embebido $1{,}25L$ bajo el lecho, y se coloca centrado en x e y como muestra el esquema. El canal posee una parte en lecho fijo (no erosionable) hasta una longitud de $5L$, el resto es lecho erosionable. El número de Reynolds de la simulación basado en la velocidad media y la longitud característica dada por las dimensiones del obstáculo (L) es de $Re = 1000$ ($\mu = 1$kg/m/s, $\rho = 1000$kg/m^3).

6.10.2. Condiciones de borde

Las condiciones de borde empleadas son:

- Cara *west*: condición de borde tipo entrada, BC tipo Neumann para la presión $\partial p/\partial \mathbf{n} = 0$, BC tipo dirichlet para la velocidad $v = w = 0$, $u \neq 0$ utilizando un perfil parabólico de forma que la $u_{\text{media}} = 1$m/s.

- Caras *north, south top*: condición de borde tipo slip-wall, BC tipo Dirichlet para la componente normal $\mathbf{u} = 0$, BC tipo Neumann para la componente tangencial $\partial \mathbf{u}/\partial \mathbf{t} = 0$ y para la presión $\partial p/\partial \mathbf{n} = 0$.

- Cara *east*: condición de borde tipo salida, BC tipo Neumann para la velocidad $\partial \mathbf{u}/\partial \mathbf{n} = 0$, BC tipo Dirichlet para la presión $p_{\text{outlet}} = 0$.

- Frontera inferior, lecho móvil y obstáculo: condición de borde no-slip-wall, BC tipo Dirichlet aplicando una ley de pared.

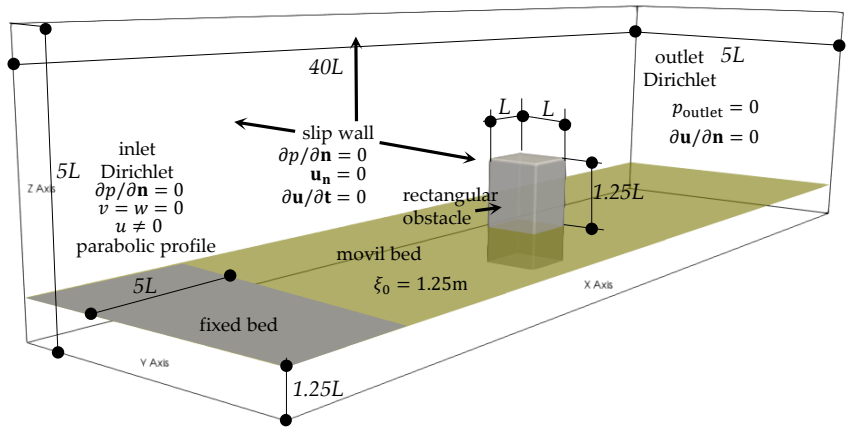

Figura 6.11: Esquema de dominio computacional para el problema de erosión alrededor de un obstáculo rectangular.

6.10.3. Solver y esquemas de discretización

El flujo incompresible se resolvió utilizando el FSM debido a que mostró ser un método semiexplícito con buen desempeño y menor requerimiento de memoria que el algoritmo SIMPLE, incluyendo a su vez el modelo de turbulencia LES. El arreglo de las variables es colocado, el acoplamiento velocidad-presión se logra utilizando la interpolación de Rhie-Chow. Los esquemas temporales elegidos fueron de segundo orden CN y AB para los términos de difusión y advección respectivamente, permitiendo aplicar un esquema espacial no lineal para el último de forma explícita mientras que la difusión se trata implícitamente evitando así la restricción en el paso de tiempo debida al número de Fourier. El esquema espacial para la difusión es CD mientras que el esquema para el término convectivo es MINMOD (esquema HR), esto se requiere para transportar gradientes pronunciados en un problema advectivo dominante sin que se originen inestabilidades numéricas. La geometría compleja se trata mediante el GC-IBM que resulta flexible para resolver las geometrías complejas que se forman en el lecho. El solver lineal para la ecuación de momento es BiCGStab (tolerancia absoluta 1e-7) debido a que al aplicar la técnica IBM de forma implícita la matriz resultante en las ecuaciones de momento es no simétrica. Para la ecuación de la presión, en cambio se utilizó PCG preacondicionado con multigrilla mediante un ciclo V (tolerancia absoluta 1e-8) ya que resultó ser la estrategia con mejor desempeño y bajo requerimiento de memoria. Se eligió un paso de tiempo fijo para toda la simulación igual a $\Delta t \approx 0{,}5h/||u_{\text{máx}}|| \approx 0{,}008$s.

Para el transporte de sedimentos se utilizó un esquema temporal AB para evitar el uso de la técnica DC, mientras que el esquema espacial elegido fue MINMOD, con esta combinación se obtiene un esquema total de segundo orden que resulta estable en el espacio y tiempo sin degradar la eficiencia del solver. Se eligió $\Delta t_{\text{sed}} = 10\Delta t$. Solamente se considera el transporte por carga de fondo. Los parámetros elegidos son: diámetro medio del sedimento $d_{50} = 0{,}1$mm, densidad del sedimento $\rho_s = 2650\text{kg/m}^3$, ángulo de reposo $\alpha_C = 30°$, porosidad $n = 0{,}4$, esfuerzo de corte crítico $\tau_c = 0{,}0001\text{N/m}^2$. Se utilizó la fórmula de transporte de MPM. En la simulación se computó hasta un tiempo físico de aproximadamente 900s.

6.10.4. Análisis de los resultados numéricos

A continuación se discuten los resultados obtenidos en la simulación numérica para lecho fijo y lecho móvil.

Estructuras de vórtices en simulación a lecho plano

Previamente a la simulación en lecho erosionable, se realiza una simulación considerando que el lecho es plano, de forma que se obtenga un flujo desarrollado. En la figura 6.12, se presentan la lineas de corriente y las estructuras de vórtices para un instante de simulación.

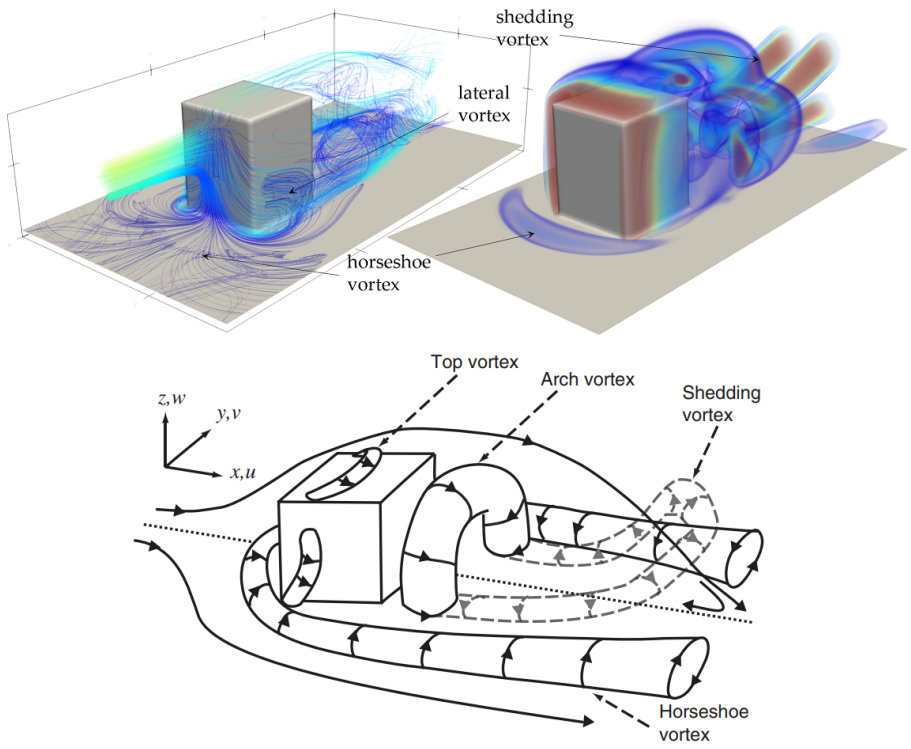

Figura 6.12: Lineas de corriente e isosuperficies de criterio Q de vorticidad para visualizar las estructuras de vórtices alrededor del prisma rectangular recto embebido en la corriente líquida para lecho fijo (arriba). Representación esquemática del flujo alrededor del cubo montado en una superficie plana, adaptación de Lacey y Rennie (2012) [94] (abajo).

A modo de comparación la figura 6.12 - abajo, muestra la representación esquemática presentada por Lacey y Rennie (2012) [94] basada en sus experimentos físicos. Puede observarse que el solver de flujo incompresible captura las complejas estructuras del flujo tridimensional esperables para el experimento.

Estructuras de vórtices con lecho erosionado

En la figura 6.13 se presentan algunos resultados de la simulación con un mapeo de textura de arena, se incorporan los contornos de nivel del lecho para demarcar la olla de erosión que se forma delante del obstáculo y la deposición en forma de pila posterior.

Mediante el uso de lineas de corriente se analiza la trayectoria de las partículas para representar los vórtices en forma de herradura primario y secundario que se forman luego de avanzada la socavación frontal, a modo de comparación, se muestra la representación esquemática de dichos vórtices observada en el estudio experimental de Schlömer et al. (2020) [152]. Dichos vórtices juegan un papel principal en el desarrollo de la olla de erosión delante de los obstáculos sumergidos.

La figura 6.14 presenta las iso superficies del segundo invariante del tensor gradiente de velocidad Q, conocido como criterio Q de vorticidad, para visualizar el desprendimiento de las estructuras de vórtice en forma de arco en la parte posterior (compare con figura 6.12-abajo) además puede observarse el vórtice de herradura primario.

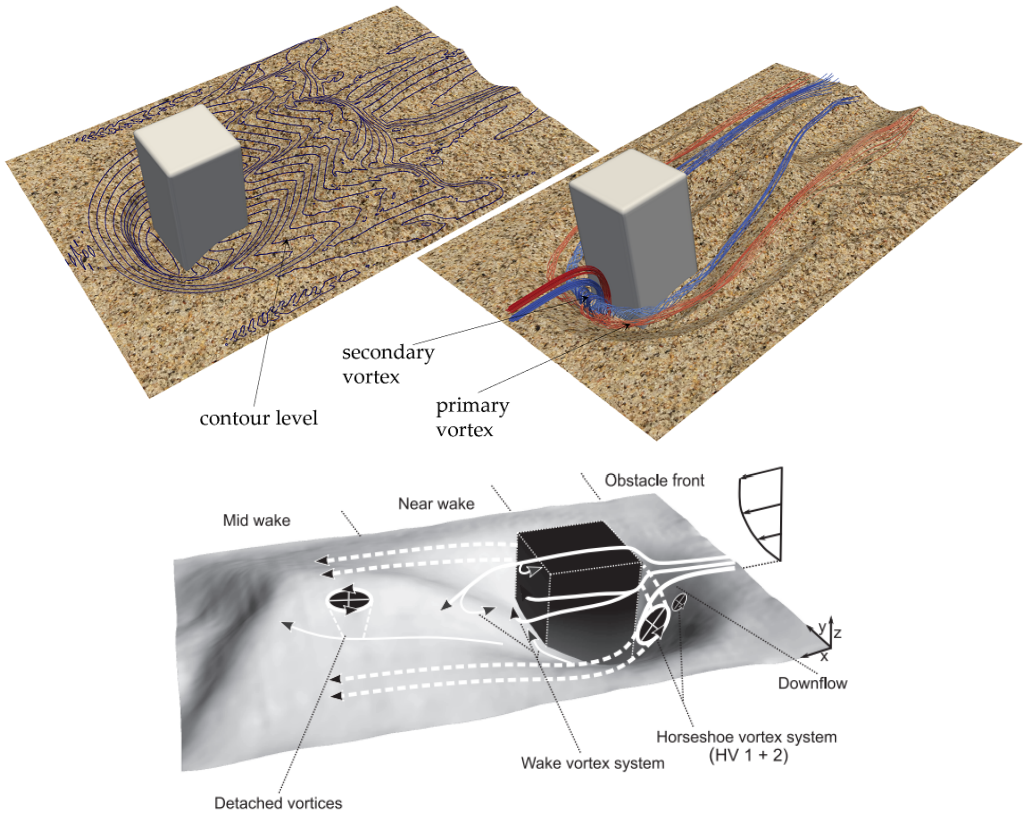

Figura 6.13: Resultados numéricos de la simulación: contornos de nivel en la olla de erosión y deposición tras el obstáculo (arriba-izquierda), vórtices de herradura primario y secundario (arriba-derecha), representación del lecho con un mapeo de textura de granos de arena. Representación esquemática del flujo alrededor del cubo montado en un lecho erosionado, adaptación de Schlömer et al. (2020) [152] (abajo).

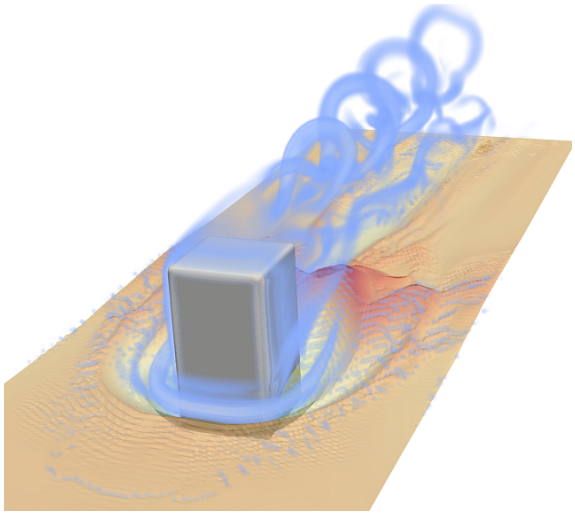

Figura 6.14: Visualización del desprendimiento de estructuras de vórtice en forma de arco utilizando las isosuperficies del criterio Q de vorticidad.

Evolución temporal de la socavación y deposición

La figura 6.15 muestra la evolución de la erosión cerca del obstáculo para distintos instantes de tiempo. Puede observarse el algoritmo para el transporte de sedimentos acoplado con el resolvedor logra capturar las diferentes formas de fondo que se originan por las corrientes complejas así como la migración de las mismas por el efecto de la tensión rasante en el lecho.

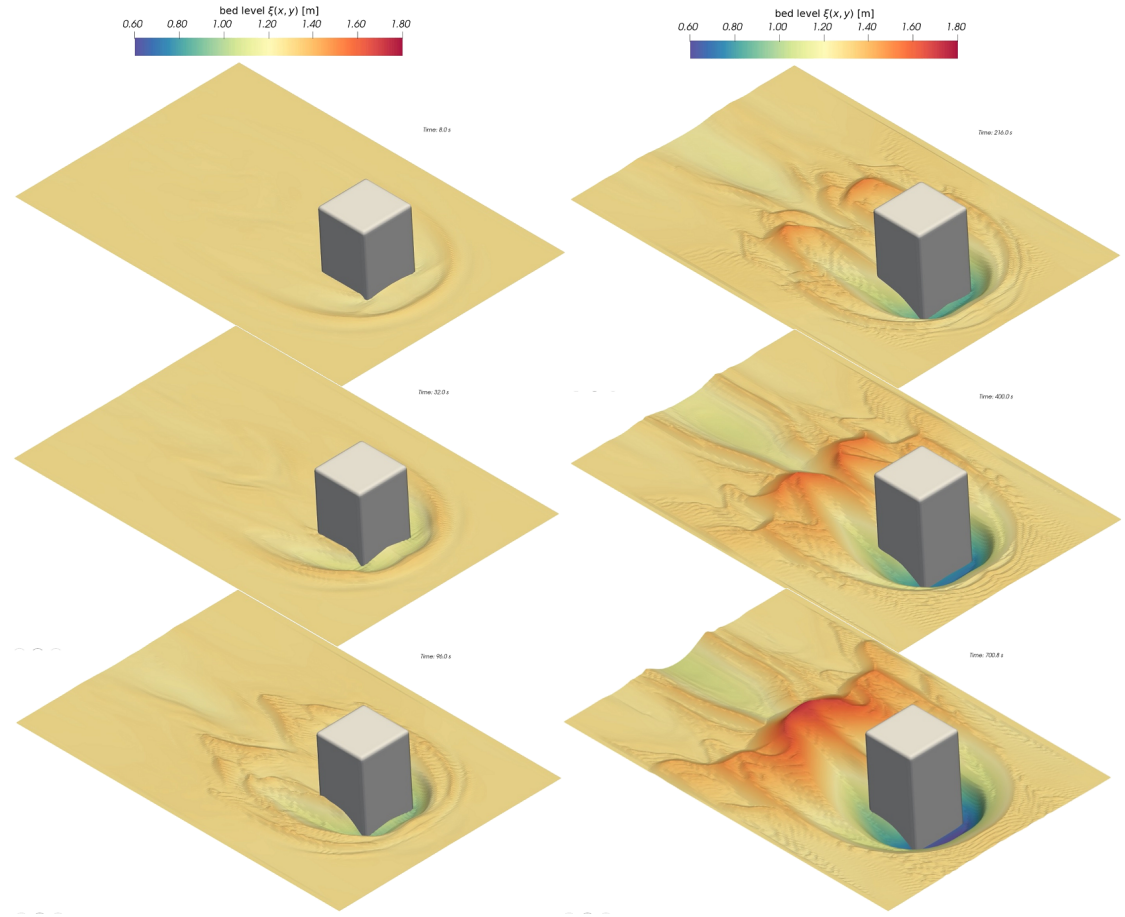

Figura 6.15: Evolución de la simulación de socavación alrededor del obstáculo rectangular para diferentes instantes de tiempo.

Comparación con otros autores

Se compararon los resultados obtenidos con los presentados por otros autores. La figura 6.16 presenta la socavación final para un tiempo físico de simulación de 900s aproximadamente (figura 6.16-arriba).

Se presenta la solución obtenida [1] por Burkow y Griebel (2016) [29] (figura 6.16-centro) utilizando el software NaSt3D con el agregado de un modelo morfodinámico, utilizando las mismas dimensiones y parámetros físicos (flujo y sedimento) que las utilizadas en esta tesis, la diferencia radica en el solver de flujo, el método de deslizamiento, los esquemas espaciales y temporales y algunas diferencias en las condiciones de borde en la entrada.

A su vez se muestra la captura del experimento físico basado en semejanza de Reynolds presentado por ellos (figura 6.16-abajo), en [29] pueden consultarse más detalles del mismo y los parámetros de la simulación numérica.

[1] Puede consultarse el video de la simulación en el siguiente enlace https://youtu.be/uPAWRhrW-l0.

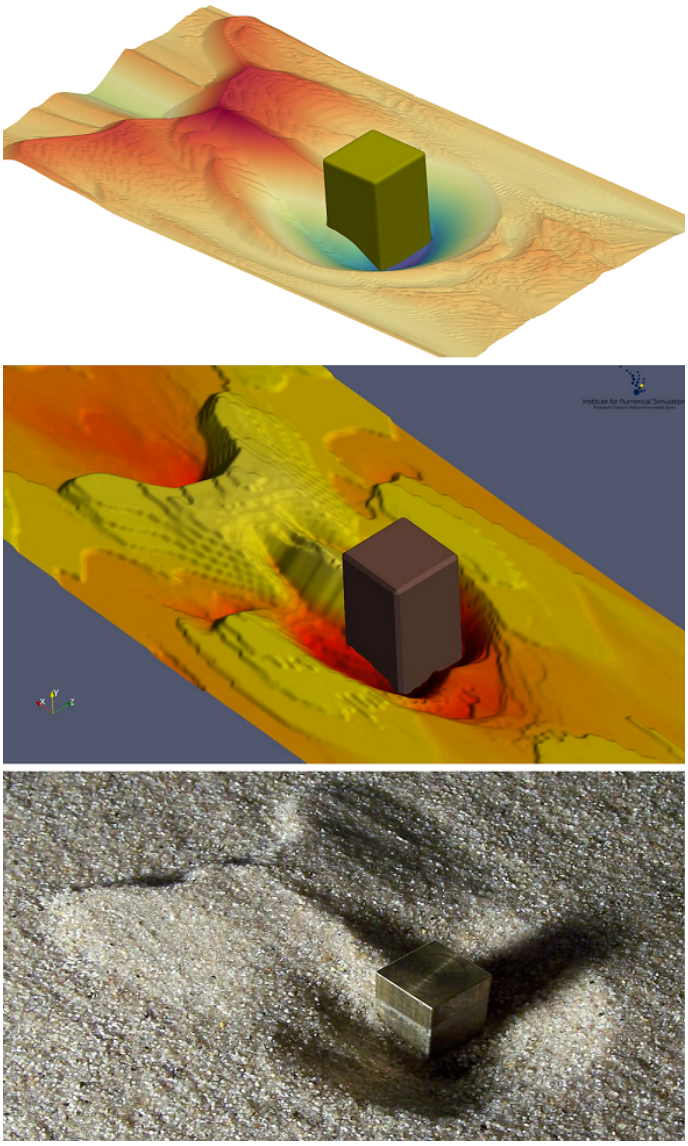

Figura 6.16: Comparación de resultados con otros trabajos. Resultados de la presente implementación (arriba), resultados numéricos obtenidos por Burkow y Griebel (2016) [29] mediante el software NaSt3D y el respectivo experimento físico a $Re = 1038$ (centro y abajo).

Puede observarse que los resultados obtenidos en la presente simulación están en concordancia con el experimento físico y la solución numérica obtenida con otro programa que usa otros algoritmos. Logrando una buena aproximación al patrón de erosión - sedimentación alrededor del obstáculo.

Finalmente en la figura 6.17 se compara la profundidad de erosión delante del obstáculo (curva azul) cuyos valores resultan más importantes para las aplicaciones de ingeniería fluvial. Puede observarse que los resultados concuerdan con las simulaciones de Burkow y Griebel (2016) [29].

Puede apreciarse que la altura de sedimentación (curva roja) difiere levemente, respecto a esto, hay que tener en cuenta que ambas simulaciones comparadas, se realizaron con diferentes condiciones de contorno, diferentes metodologías y algoritmos con lo que el resultado obtenido se considera suficientemente aceptable.

Tiempo de cómputo

En cuanto a los tiempos de cómputo de la simulación, en [29] reportan que el calculo hasta un tiempo físico de 450s consume alrededor de 180h (7.5 días $\approx$ 1 semana) mediante cálculo paralelo

en CPU utilizando 2 nodos de cómputo Dell PowerdEdge R910 equipados con 32 cores cada uno (64 cores en total) del tipo Intel Xeon X7560 corriendo a 2.226GHz. La simulación presentada se ejecutó utilizando el equipo 2 (GPU Tesla V100), cabe mencionar que todos los cálculos se ejecutan prácticamente de forma completa en GPU utilizando una sola tarjeta, la CPU se utiliza para lanzar las ejecuciones de los diferentes kernels en la GPU.

Para simular hasta el tiempo físico de 450s, el código elaborado en esta tesis consume alrededor de 10h, es decir que resulta unas 18 veces más rápido, o en otros términos, en el mejor de los casos se requieren 36 equipos de cómputo similares ($32 \times 36 = 1152$ cores) para igualar una tarjeta. Esto muestra que la GPU es una herramienta atractiva para realizar simulaciones en tiempos realmente reducidos.

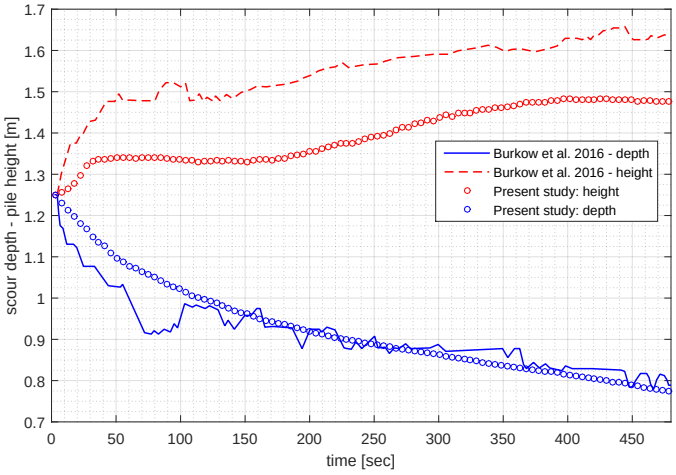

Figura 6.17: Validación de resultados: evolución temporal de la profundidad de socavación y la altura máxima de la duna depositada tras el obstáculo. Comparación con Burkow y Griebel (2016) [29].

Capítulo 7

Conclusiones

7.1. Conclusiones del trabajo

En este capítulo, se presentan las conclusiones obtenidas a partir de la investigación realizada en el marco de esta tesis de doctorado. Las mismas se basan en una análisis de la revisión de la literatura existente y la aplicación de las metodologías desarrolladas.

Los objetivos generales y específicos presentados, se centraron en el estudio e implementación de algoritmos del tipo FVM para CFD y el transporte de sedimentos basados en GPU. Para ello, se expuso de forma resumida los principales esquemas espaciales y temporales para la discretización de las ecuaciones de conservación utilizadas en el contexto de la dinámica de fluidos computacional. Dicha descripción pretende contribuir en una documentación introductoria para la comunidad científica.

A partir de esto, se desarrollaron diferentes estrategias de paralelización en GPU para los esquemas numéricos en el contexto del método de los volúmenes finitos con un ordenamiento de variables centrado en celda (arreglo colocado). Los algoritmos y métodos implementados en GPU incluyen: el desarrollo de esquemas TVD con enfoques implícito y explícito, implementación de un solver que resuelve el flujo incompresible utilizando dos métodos diferentes, SIMPLE y FSM. A su vez, se implementaron diferentes métodos para la solución de sistemas lineales dispersos en GPU que incluyen: método de Gradientes Conjugados, Gradientes Biconjugados Estabilizado y un método multigrilla simple y con un bajo requerimiento de memoria que puede utilizarse como solver independiente o como precondicionador dentro de los métodos de Krylov. Todos los métodos se encuentran empaquetados para formar una librería que puede adaptarse para resolver problemas con diferentes aplicaciones.

Para validar los métodos y demostrar el rendimiento que pueden alcanzar las implementaciones basadas en GPU, se resolvieron diferentes problemas, entre ellos: problemas de advección difusión usando esquemas lineales y no lineales en 2D, una ecuación de difusión no lineal en 3D, las ecuaciones de Navier Stokes en 2D y 3D y la solución de ecuaciones elípticas en 2D aplicando técnicas multigrilla. Se mostró que los métodos implícitos en GPU, a pesar de tener tasas de cómputo menores pueden tener un desempeño computacional realmente bueno en comparación con los métodos explícitos eligiendo correctamente los resolvedores y combinando diferentes estrategias. Esto refuta la idea general que se tiene sobre la preferencia de métodos que sean explícitos en GPU. Con la implementación de los esquemas TVD, se resolvió en GPU un problema advectivo dominante complejo de resolver de forma masiva en el contexto de simulaciones tipo Monte Carlo resuelto en general mediante *Particle Traking Method*, mostrando resultados similares obtenidos en tiempos de cómputo razonables, brindando una herramienta que al ser basada en métodos eulerianos podría extenderse sin dificultad a problemas reactivos incorporando la componente química a las ecuaciones de balance. Se mostró que en el caso de advección difusión, para que el método explícito sea más eficiente que la versión implícita puede ser necesario el uso de esquemas temporales de mayor orden, debido a que el error temporal del esquema FE puede ser

superior al error del esquema espacial. Las ganancias en tiempo de cómputo respecto a implementaciones CPU paralelas mostraron que en la mayoría de casos se requiere ejecutar los códigos en al menos 30 equipos funcionando en paralelo de similares características a los utilizados en esta tesis para equiparar el desempeño obtenido con una sola tarjeta de vídeo de la ante última generación. En cuanto al método multigrilla se lograron desempeños superiores a los de dos librerías muy utilizadas en la comunidad como son HYPRE y AmgX de Nvidia, donde se destaca una reducción significativa en el consumo de memoria en GPU respecto de la última. Para los métodos utilizados en las ecuaciones de NS en GPU el algoritmo SIMPLE a pesar de ser un método más complejo de implementar, para problemas suficientemente grandes, el desempeño puede ser equiparable al de un método del tipo semi-explícito como lo es el FSM.

Se implementó una estrategia IBM combinando técnicas de interpolación que hacen uso de nodos fantasma en GPU, incorporarndo además una estrategia para aplicar una condición de borde tipo *slip* o aplicar una ley de pared. Los resultados fueron validados considerando casos bien documentados en la bibliografía, comprobando que el método es efectivo para utilizar en GPU. Cabe mencionar que mientras otros códigos utilizan esquemas espaciales CD o QUICK para el término advectivo, aquí se incorporaron los esquemas TVD previamente desarrollados, haciendo el algoritmo para el flujo incompresible más versátil para aplicar en diferentes problemas.

Se logró acoplar al solver de flujo incompresible, un modelo de turbulencia básico tipo LES y un modelo de sedimentos que calcula las cargas de fondo en función de los esfuerzos de corte en el lecho, lo que permite actualizar las alturas del nivel del lecho y con ello la evolución del dominio computacional. Para obtener soluciones físicamente más realistas, se implementó un modelo de deslizamiento de granos que permite contemplar el efecto gravitatorio, evitando que se originen formas de fondo no físicas. Para la discretización de los términos de la ecuación de Exner se utilizaron los esquemas de alta resolución desarrollados previamente. El cálculo de los esfuerzo de corte se basó en la aplicación de una ley de pared. En el cálculo de la evolución del lecho y la evolución del flujo incompresible, se utilizó una estrategia de particionamiento explícito con un enfoque desacoplado donde los pasos de tiempo para el lecho Δt_{sed} son más grandes que para el flujo Δt en una relación n_{sed}, es decir $\Delta t_{\text{sed}} = n_{\text{sed}} \Delta t$. Para contrastar los resultados obtenidos por la librería desarrollada en esta Tesis, se resolvió un problema tridimensional de erosión localizada alrededor de un objeto rectangular documentado por otros autores, mostrando resultados cualitativamente y cuantitativamente satisfactorios pero en tiempos realmente reducidos. En concreto, la aceleración que se obtiene respecto a una implementación paralela CPU como el código NaSt3D ejecutada en un nodo con 2 equipos de cómputo de 32 cores cada uno, es de hasta 18 veces, lo que significa que se requerirían a lo menos, un clúster con 36 equipos de esas características ($32 \times 36 = 1152$ cores) para equiparar el tiempo de cómputo de una sola tarjeta. Estas mejoras de rendimiento constituyen el principal objetivo buscado con esta Tesis y se ha logrado en todos los casos estudiados.

En resumen, los resultados obtenidos en esta Tesis doctoral han demostrado la viabilidad y eficiencia de los esquemas TVD implícitos y explícitos en GPU, así como la efectividad de los métodos SIMPLE y FSM para resolver problemas de flujo incompresible. La implementación y acople de estos métodos con un solver para calcular el transporte de sedimentos en una librería completa ha brindado una herramienta valiosa para la comunidad científica. Las conclusiones obtenidas a través de esta investigación abren nuevas perspectivas y proporcionan una base sólida para futuras investigaciones en este campo utilizando GPU.

7.2. Sobre la evolución de las GPUs en los últimos años

Debe mencionarse que la limitación en memoria de la GPU al momento de comienzo de esta Tesis era de 12GB (Nvidia Tesla K40), posteriormente en 2018 surge la siguiente generación (Nvidia Tesla V100) de GPUs que provee una ampliación de memoria de hasta 32GB, la ganancia en tiempos de cómputo entre una y otra generación es del orden de 5 a 6 según diferentes pruebas

realizadas en la Tesis. Finalmente la última generación de tarjetas gráficas para cómputo científico Nvidia Tesla A100 (lanzamiento en 2020) posee una versión de hasta 80GB lo que permite abordar problemas más grandes como los que se realizan actualmente en CPU. En cuanto a rendimiento comparado con la Tesla V100, posee 7936 núcleos y permite una ganancia en tiempos de cómputo superior a $2\times$ de acuerdo a los diferentes benchmark reportados por el fabricante.

7.3. Lineas de investigación abiertas

Una de las limitaciones encontradas en el método IBM propuesto es que para altos Reynolds el grado de refinamiento que se requiere cerca de la frontera embebida es inviable de realizar en GPU para dominios grandes. Frente a esto, hay dos estrategias que se podrían aplicar, la primera consiste en utilizar métodos de mayor orden como el esquema WENO (5to orden) y un esquema de 4to orden para difusión, esto permite el uso de grillas relativamente gruesas, las dificultades en este caso es identificar la estructura de datos para aplicar la condición de contorno con una precisión de 2do o 3er orden al incorporar una frontera embebida. Esta estrategia ha mostrado buenos resultados en la versión CPU paralela, sin embargo requiere una gran estructura de datos que habría que repensar en GPU. La segunda estrategia es implementar grillas adaptativas que permitan refinar cerca de la frontera embebida capturando de una mejor forma las estructuras de flujo que se dan cerca de la capa límite responsables de la erosión. Esto abre unas líneas de investigación no exploradas en GPU en este contexto.

Otras mejoras consisten en incorporar modelos de turbulencia tipo RANS combinados con leyes de pared. Esto implica una investigación sobre las diferentes formulaciones a considerar para el transporte de sedimentos, al igual que la incorporación de modelos de turbulencia a la ecuación de transporte en suspensión, para lo cual casi no se ha encontrado información en la bibliografía.

Otra linea de investigación consiste en expandir el código utilizando la implementación de LSM para resolver problemas de erosión con superficie libre.

Por otra parte, la librería desarrollada permite el estudio e implementación de diferentes modelos de transporte de sedimentos, aunque en esta Tesis no se realizó la aplicación combinando transporte en modalidad de suspensión, actualmente casi no hay estudios que permitan validar los modelos teóricos, esto provee una herramienta económica para investigar las diferentes formulaciones.

Bibliografía

[1] Nvidia developer documentation, cuda toolkit documentation. 2022.

[2] Nvidia developer documentation, cuda toolkit documentation. 2022.

[3] Nvidia developer documentation, cuda toolkit documentation. 2022.

[4] Nvidia developer documentation, cuda toolkit documentation. 2022.

[5] Thrust: The c++ parallel algorithms library. 2022.

[6] Reef3d : Open-source hydrodynamics. 2023.

[7] Afzal M.S. *3d numerical modelling of sediment transport under current and waves*. Tesis de Maestría, Institutt for bygg, anlegg og transport, 2013.

[8] Ahmad N., Bihs H., Myrhaug D., Kamath A., y Arntsen Ø.A. Three-dimensional numerical modelling of wave-induced scour around piles in a side-by-side arrangement. *Coastal Engineering*, 138:132–151, 2018. https://doi.org/10.1016/j.coastaleng.2018.04.016.

[9] Alcouffe R.E., Brandt A., Dendy Jr J.E., y Painter J.W. The multi-grid method for the diffusion equation with strongly discontinuous coefficients. *SIAM Journal on Scientific and Statistical Computing*, 2(4):430–454, 1981. https://doi.org/10.1137/0902035.

[10] Armfield S. y Street R. An analysis and comparison of the time accuracy of fractional-step methods for the navier–stokes equations on staggered grids. *International Journal for Numerical Methods in Fluids*, 38(3):255–282, 2002. https://doi.org/10.1002/fld.217.

[11] Ashby S.F. y Falgout R.D. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. *Nuclear Science and Engineering*, 124(1):145–159, 1996. https://doi.org/10.13182/NSE96-A24230.

[12] Aubry R., Mut F., Löhner R., y Cebral J.R. Deflated preconditioned conjugate gradient solvers for the pressure-poisson equation. *Journal of Computational Physics*, 227(24):10196–10208, 2008. https://doi.org/10.1016/j.jcp.2008.08.025.

[13] Auguste F., Réa G., Paoli R., Lac C., Masson V., y Cariolle D. Implementation of an immersed boundary method in the meso-nh v5. 2 model: applications to an idealized urban environment. *Geoscientific Model Development*, 12(6):2607–2633, 2019.

[14] Babaeyan-Koopaei K., Ervine D., Carling P., y Cao Z. Velocity and turbulence measurements for two overbank flow events in river severn. *Journal of Hydraulic Engineering*, 128(10):891–900, 2002.

[15] Bagnold R.A. The flow of cohesionless grains in fluids. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 249(964):235–297, 1956. https://doi.org/https://doi.org/10.1098/rsta.1956.0020.

[16] Balaras E. Modeling complex boundaries using an external force field on fixed cartesian grids in large-eddy simulations. *Computers & Fluids*, 33(3):375–404, 2004. https://doi.org/10.1016/S0045-7930(03)00058-6.

[17] Barlas G. *Multicore and GPU Programming: An integrated approach*. Elsevier, 2014.

[18] Battaglia L. *Elementos finitos estabilizados para flujos con superficie libre: seguimiento y captura de interfase*. Tesis de Doctorado, 2009.

[19] Bear J. *Dynamics of Fuids in Porous Media*. American Elsevier, New York, 1972.

[20] Beaudoin A., de Dreuzy J.R., y Erhel J. Numerical monte carlo analysis of the influence of pore-scale dispersion on macrodispersion in 2-d heterogeneous porous media. *Water Resources Research*, 46(12), 2010. https://doi.org/10.1029/2010WR009576.

[21] Bessone L., Gamazo P., Dentz M., Storti M., y Ramos J. Gpu implementation of explicit and implicit eulerian methods with tvd schemes for solving 2d solute transport in heterogeneous flows. *Computational Geosciences*, 26(3):517–543, 2022. https://doi.org/10.1007/s10596-022-10136-8.

[22] Bohacek J., Kharicha A., Ludwig A., Wu M., Holzmann T., y Karimi-Sibaki E. A gpu solver for symmetric positive-definite matrices vs. traditional codes. *Computers & Mathematics with Applications*, 78(9):2933–2943, 2019. https://doi.org/10.1016/j.camwa.2019.02.034.

[23] Bondarenco M. Paralelización de la ecuación del transporte en arquitecturas de hardware masivamente paralelas. 2018.

[24] Botto L. A geometric multigrid poisson solver for domains containing solid inclusions. *Computer Physics Communications*, 184(3):1033–1044, 2013. https://doi.org/10.1016/j.cpc.2012.11.008.

[25] Brandao G.G. *Solution of the transport equation using graphical processing units*. Tesis de Doctorado, Ph. D. thesis, MSc thesis IST, 2009.

[26] Briggs W.L., Henson V.E., y McCormick S.F. *A multigrid tutorial*. SIAM, 2000.

[27] Brown P.N., Falgout R.D., y Jones J.E. Semicoarsening multigrid on distributed memory machines. *SIAM Journal on Scientific Computing*, 21(5):1823–1834, 2000. https://doi.org/10.1137/S1064827598339141.

[28] Burkow M. *The numerical simulation of a three dimensional fluid sediment system on arbitrarily shaped domains*. Tesis de Doctorado, Universitäts-und Landesbibliothek Bonn, 2016.

[29] Burkow M. y Griebel M. A full three dimensional numerical simulation of the sediment transport and the scouring at a rectangular obstacle. *Computers & Fluids*, 125:1–10, 2016. https://doi.org/10.1016/j.compfluid.2015.10.014.

[30] Burkow M. y Griebel M. Numerical simulation of the temporal evolution of a three dimensional barchanoid dune and the corresponding sediment dynamics. *Computers & Fluids*, 166:275–285, 2018. https://doi.org/10.1016/j.compfluid.2018.02.018.

[31] Chakravarthy S. y Osher S. High resolution applications of the osher upwind scheme for the euler equations. En *6th Computational Fluid Dynamics Conference Danvers*, página 1943. 1983. https://doi.org/10.2514/6.1983-1943.

[32] Che S., Boyer M., Meng J., Tarjan D., Sheaffer J.W., y Skadron K. A performance study of general-purpose applications on graphics processors using cuda. *Journal of parallel and distributed computing*, 68(10):1370–1380, 2008. https://doi.org/10.1016/j.jpdc.2008.05.014.

[33] Cheng J., Grossman M., y McKercher T. *Professional CUDA c programming*. John Wiley & Sons, 2014.

[34] Chi C., Abdelsamie A., y Thévenin D. A directional ghost-cell immersed boundary method for incompressible flows. *Journal of Computational Physics*, 404:109122, 2020.

[35] Choi H. y Moin P. Effects of the computational time step on numerical solutions of turbulent flow. *Journal of Computational Physics*, 113(1):1–4, 1994. https://doi.org/10.1006/jcph.1994.1112.

[36] Choi J.I., Oberoi R.C., Edwards J.R., y Rosati J.A. An immersed boundary method for complex incompressible flows. *Journal of Computational Physics*, 224(2):757–784, 2007. https://doi.org/10.1016/j.jcp.2006.10.032.

[37] Chorin A.J. Numerical solution of the navier-stokes equations. *Mathematics of computation*, 22(104):745–762, 1968. https://doi.org/10.2307/2004575.

[38] Chorin A.J. A numerical method for solving incompressible viscous flow problems. *Journal of computational physics*, 135(2):118–125, 1997. https://doi.org/10.1006/jcph.1997.5716.

[39] Clift R., Grace J.R., y Weber M.E. Bubbles, drops, and particles. 2005.

[40] Costarelli S.D., Garelli L., Cruchaga M.A., Storti M.A., Ausensi R., y Idelsohn S.R. An embedded strategy for the analysis of fluid structure interaction problems. *Computer Methods in Applied Mechanics and Engineering*, 300:106–128, 2016. https://doi.org/10.1016/j.cma.2015.11.001.

[41] Cotronis Y., Konstantinidis E., y Missirlis N.M. A gpu implementation for solving the convection diffusion equation using the local modified sor method. En *Numerical Computations with GPUs*, páginas 207–221. Springer, 2014. https://doi.org/10.1007/978-3-319-06548-9_10.

[42] Dalton S., Bell N., Olson L., y Garland M. Cusp: Generic parallel algorithms for sparse matrix and graph computations. 2014. Version 0.5.0.

[43] Das S., Panda A., Deen N., y Kuipers J. A sharp-interface immersed boundary method to simulate convective and conjugate heat transfer through highly complex periodic porous structures. *Chemical Engineering Science*, 191:1–18, 2018.

[44] de Dreuzy J.R., Beaudoin A., y Erhel J. Asymptotic dispersion in 2d heterogeneous porous media determined by parallel numerical simulations. *Water Resources Research*, 43(10), 2007. https://doi.org/10.1029/2006WR005394.

[45] Deen N.G., Kriebitzsch S.H., van der Hoef M.A., y Kuipers J. Direct numerical simulation of flow and heat transfer in dense fluid–particle systems. *Chemical engineering science*, 81:329–344, 2012. https://doi.org/10.1016/j.ces.2012.06.055.

[46] Dendy J. Black box multigrid. *Journal of Computational Physics*, 48(3):366–386, 1982. https://doi.org/10.1016/0021-9991(82)90057-2.

[47] Egloff D. Part i: High-performance tridiagonal solvers on gpus. *Wilmott Magazine*, páginas 32–40,

2010.

[48] Einstein H.A. Formulas for the transportation of bed load. *Transactions of ASCE*, páginas 561–597, 1941. https://doi.org/10.1061/TACEAT.0005468.

[49] Einstein H.A. The bed-load function for sediment transportation in open channel flows. Informe Técnico, 1950. https://doi.org/10.22004/ag.econ.156389.

[50] Engelund F. y Fredsøe J. A sediment transport model for straight alluvial channels. *Hydrology Research*, 7(5):293–306, 1976. https://doi.org/10.2166/nh.1976.0019.

[51] Engelund F. y Hansen E. A monograph on sediment transport. *Technisk Forlag, Copenhagen, Denmark*, 1967.

[52] Erhel J., de Dreuzy J.R., Beaudoin A., Bresciani E., y Tromeur-Dervout D. A parallel scientific software for heterogeneous hydrogeoloy. En *Parallel Computational Fluid Dynamics 2007*, páginas 39–48. Springer, 2009. https://doi.org/10.1007/978-3-540-92744-0_5.

[53] Exner F.M. Uber die wechselwirkung zwischen wasser und geschiebe in flussen. *Akad. Wiss. Wien Math. Naturwiss. Klasse*, 134(2a):165–204, 1925.

[54] Fadlun E., Verzicco R., Orlandi P., y Mohd-Yusof J. Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations. *Journal of computational physics*, 161(1):35–60, 2000. https://doi.org/10.1006/jcph.2000.6484.

[55] Falgout R.D. y Jones J.E. Multigrid on massively parallel architectures. En *Multigrid Methods VI*, páginas 101–107. Springer, 2000. https://doi.org/10.1007/978-3-642-58312-4_13.

[56] Falgout R.D., Jones J.E., y Yang U.M. The design and implementation of hypre, a library of parallel high performance preconditioners. En *Numerical solution of partial differential equations on parallel computers*, páginas 267–294. Springer, 2006. https://doi.org/10.1007/3-540-31619-1_8.

[57] Felippa C.A., Park K., y Farhat C. Partitioned analysis of coupled mechanical systems. *Computer methods in applied mechanics and engineering*, 190(24-25):3247–3270, 2001. https://doi.org/10.1016/S0045-7825(00)00391-1.

[58] Feng C., Shu S., Xu J., y Zhang C.S. Numerical study of geometric multigrid methods on cpu-gpu heterogeneous computers. *Advances in Applied Mathematics and Mechanics*, 6(1):1–23, 2014. https://doi.org/10.1017/S2070073300002411.

[59] Ferguson R. y Church M. A simple universal equation for grain settling velocity. *Journal of sedimentary Research*, 74(6):933–937, 2004. https://doi.org/10.1306/051204740933.

[60] Fernandez Luque R. y Van Beek R. Erosion and transport of bed-load sediment. *Journal of hydraulic research*, 14(2):127–144, 1976. https://doi.org/10.1061/JYCEAJ.0000874.

[61] Ferziger J.H. y Peric M. *Computational methods for fluid dynamics*. Springer Science & Business Media, 2012.

[62] García M.H. Sediment transport and morphodynamics. En *Sedimentation engineering: Processes, measurements, modeling, and practice*, páginas 21–163. 2008.

[63] Ghia U., Ghia K.N., y Shin C. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411, 1982. https://doi.org/10.1016/0021-9991(82)90058-4.

[64] Gibou F., Fedkiw R.P., Cheng L.T., y Kang M. A second-order-accurate symmetric discretization of the poisson equation on irregular domains. *Journal of Computational Physics*, 176(1):205–227, 2002. https://doi.org/10.1006/jcph.2001.6977.

[65] Godunov S.K. A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics. *Matematicheskii Sbornik*, 89(3):271–306, 1959. https://hal.science/hal-01620642.

[66] Goldstein D., Handler R., y Sirovich L. Modeling a no-slip flow boundary with an external force field. *Journal of computational physics*, 105(2):354–366, 1993. https://doi.org/10.1006/jcph.1993.1081.

[67] Grave M., Camata J.J., y Coutinho A.L. Residual-based variational multiscale 2d simulation of sediment transport with morphological changes. *Computers & Fluids*, 196:104312, 2020. https://doi.org/10.1016/j.compfluid.2019.104312.

[68] Gupta R., Van Gijzen M.B., y Vuik C.K. Efficient two-level preconditioned conjugate gradient method on the gpu. En *International Conference on High Performance Computing for Computational Science*, páginas 36–49. Springer, 2012. https://doi.org/10.1007/978-3-642-38718-0_7.

[69] Ha S., Park J., y You D. A gpu-accelerated semi-implicit fractional-step method for numerical solutions of incompressible navier–stokes equations. *Journal of Computational Physics*, 352:246–264, 2018. https://doi.org/10.1016/j.jcp.2017.09.055.

[70] Hackbusch W. *Multi-grid methods and applications*, volumen 4. Springer Science & Business Media, 2013.

[71] Harten A. High resolution schemes for hyperbolic conservation laws. *Journal of Computational Physics*, 49(3):357 – 393, 1983. ISSN 0021-9991. https://doi.org/10.1016/0021-9991(83)90136-5.

[72] Hemker P. On the order of prolongations and restrictions in multigrid procedures. *Journal of Computational and Applied Mathematics*, 32(3):423–429, 1990. https://doi.org/10.1016/0377-0427(90)90047-4.

[73] Hestenes M.R., Stiefel E., et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

[74] Hirsch C. *Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics*. Elsevier, 2007.

[75] Hu O., Zhao N., y Liu J. A ghost cell method for turbulent compressible viscous flows on adaptive cartesian grids. *Procedia Engineering*, 67:241–249, 2013. https://doi.org/10.1016/j.proeng.2013.12.023.

[76] Iaccarino G. y Verzicco R. Immersed boundary technique for turbulent flow simulations. *Appl. Mech. Rev.*, 56(3):331–347, 2003.

[77] Igounet P., Alfaro P., Pedemonte M., y Ezzatti P. A gpu implementation of the sip method. En *2011 30th International Conference of the Chilean Computer Science Society*, páginas 195–201. IEEE, 2011. https://doi.org/10.1109/SCCC.2011.26.

[78] Issa R.I. Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of computational physics*, 62(1):40–65, 1986. https://doi.org/10.1016/0021-9991(86)90099-9.

[79] Johnson T. y Patel V. Flow past a sphere up to a reynolds number of 300. *Journal of Fluid Mechanics*, 378:19–70, 1999. https://doi.org/10.1017/S0022112098003206.

[80] Karlsson N. *An incompressible Navier-Stokes equations solver on the GPU using CUDA*. Tesis de Doctorado, Master's Thesis. Chalmers University of Technology, 2013.

[81] Khalil M. y Wesseling P. Vertex-centered and cell-centered multigrid for interface problems. *Journal of Computational Physics*, 98(1):1–10, 1992. https://doi.org/10.1016/0021-9991(92)90168-X.

[82] Khosla P. y Rubin S. A diagonally dominant second-order accurate implicit scheme. *Computers & Fluids*, 2(2):207–209, 1974. https://doi.org/10.1016/0045-7930(74)90014-0.

[83] Khosronejad A., Kang S., Borazjani I., y Sotiropoulos F. Curvilinear immersed boundary method for simulating coupled flow and bed morphodynamic interactions due to sediment transport phenomena. *Advances in water resources*, 34(7):829–843, 2011. https://doi.org/10.1016/j.advwatres.2011.02.017.

[84] Kim J., Kim D., y Choi H. An immersed-boundary finite-volume method for simulations of flow in complex geometries. *Journal of computational physics*, 171(1):132–150, 2001. https://doi.org/10.1006/jcph.2001.6778.

[85] Kim J. y Moin P. Application of a fractional-step method to incompressible navier-stokes equations. *Journal of computational physics*, 59(2):308–323, 1985. https://doi.org/10.1016/0021-9991(85)90148-2.

[86] Kraft S., Wang Y., y Oberlack M. Large eddy simulation of sediment deformation in a turbulent flow by means of level-set method. *Journal of Hydraulic Engineering*, 137(11):1394–1405, 2011.

[87] Krishnan A. *Towards the study of flying snake aerodynamics, and an analysis of the direct forcing method*. Tesis de Doctorado, Boston University, 2015.

[88] Krishnan A., Mesnard O., y Barba L.A. cuibm: a gpu-based immersed boundary method code. *J. Open Source Softw.*, 2(15):301, 2017. https://doi.org/10.21105/joss.00301.

[89] Ku H.C., Hirsh R.S., y Taylor T.D. A pseudospectral method for solution of the three-dimensional incompressible navier-stokes equations. *Journal of Computational Physics*, 70(2):439–462, 1987. https://doi.org/10.1016/0021-9991(87)90190-2.

[90] Kumar P., Rodrigo C., Gaspar F., y Oosterlee K. On local fourier analysis of multigrid methods for pdes with jumping and random coefficients. *SIAM Journal on Scientific Computing*, 41(3):A1385–A1413, 2019. https://doi.org/10.1137/18M1173769.

[91] Kumar P., Rodrigo C., Gaspar F.J., y Oosterlee C.W. A parametric acceleration of multilevel monte carlo convergence for nonlinear variably saturated flow. *Computational Geosciences*, 24(1):311–331, 2020. https://doi.org/10.1007/s10596-019-09922-8.

[92] Kuzmin D. *A guide to numerical methods for transport equations*. University Erlangen-Nuremberg, 2010. http://www.mathematik.uni-dortmund.de/~kuzmin/Transport.pdf.

[93] Kwak D.Y. V-cycle multigrid for cell-centered finite differences. *SIAM Journal on Scientific Computing*, 21(2):552–564, 1999. https://doi.org/10.1137/S1064827597327310.

[94] Lacey R.J. y Rennie C.D. Laboratory investigation of turbulent flow structure around a bed-mounted cube at multiple flow stages. *Journal of Hydraulic Engineering*, 138(1):71–84, 2012. https://doi.org/

10.1061/(ASCE)HY.1943-7900.0000476.

[95] Lai M.C. y Peskin C.S. An immersed boundary method with formal second-order accuracy and reduced numerical viscosity. *Journal of computational Physics*, 160(2):705–719, 2000. https://doi.org/10.1006/jcph.2000.6483.

[96] Lax P.D. y Richtmyer R.D. Survey of the stability of linear finite difference equations. *Communications on pure and applied mathematics*, 9(2):267–293, 1956. https://doi.org/10.1002/cpa.3160090206.

[97] Layton S.K., Krishnan A., y Barba L.A. cuibm–a gpu-accelerated immersed boundary method. *arXiv preprint arXiv:1109.3524*, 2011. https://doi.org/10.48550/arXiv.1109.3524.

[98] Lee B., Darbon J., Osher S., y Kang M. Revisiting the redistancing problem using the hopf–lax formula. *Journal of Computational Physics*, 330:268–281, 2017. https://doi.org/10.1016/j.jcp.2016.11.005.

[99] Leonard B.P. A stable and accurate convective modelling procedure based on quadratic upstream interpolation. *Computer methods in applied mechanics and engineering*, 19(1):59–98, 1979. https://doi.org/10.1016/0045-7825(79)90034-3.

[100] Liang D., Cheng L., y Li F. Numerical modeling of flow and scour below a pipeline in currents: Part ii. scour simulation. *Coastal engineering*, 52(1):43–62, 2005. https://doi.org/10.1016/j.coastaleng.2004.09.001.

[101] Lilly D.K. The representation of small-scale turbulence in numerical simulation experiments. *IBM Form*, páginas 195–210, 1967.

[102] Lindholm E., Nickolls J., Oberman S., y Montrym J. Nvidia tesla: A unified graphics and computing architecture. *IEEE micro*, 28(2):39–55, 2008. https://doi.org/10.1109/MM.2008.31.

[103] Liu X. y García M.H. Three-dimensional numerical model with free water surface and mesh deformation for local sediment scour. *Journal of waterway, port, coastal, and ocean engineering*, 134(4):203–217, 2008. https://doi.org/10.1061/(ASCE)0733-950X(2008)134:4(203).

[104] Loppi N., Witherden F.D., Jameson A., y Vincent P.E. A high-order cross-platform incompressible navier–stokes solver via artificial compressibility with application to a turbulent jet. *Computer Physics Communications*, 233:193–205, 2018. https://doi.org/10.1016/j.cpc.2018.06.016.

[105] Lube G., Huppert H.E., Sparks R.S.J., y Hallworth M.A. Axisymmetric collapses of granular columns. *Journal of Fluid Mechanics*, 508:175–199, 2004. 10.1017/S0022112004009036.

[106] Maitri R., Das S., Kuipers J., Padding J., y Peters E. An improved ghost-cell sharp interface immersed boundary method with direct forcing for particle laden flows. *Computers & Fluids*, 175:111–128, 2018. https://doi.org/10.1016/j.compfluid.2018.08.018.

[107] Majumdar S., Iaccarino G., Durbin P., et al. Rans solvers with adaptive structured boundary non-conforming grids. *Annual Research Briefs*, 1, 2001.

[108] Maliska C.R. *A Solution Method for Three Dimensional Parabolic Fluid Flow Problems in Non-Orthogonal Coordinates.* Tesis de Doctorado, Ph.D. thesis, The University of Waterloo, Canada, 1981. https://elibrary.ru/gewmjb.

[109] Mandikas V.G. y Mathioudakis E.N. A parallel multigrid solver for incompressible flows on computing architectures with accelerators. *The Journal of Supercomputing*, 73(11):4931–4956, 2017. https://doi.org/10.1007/s11227-017-2066-y.

[110] Manea A., Tchelepi H., et al. A massively parallel semicoarsening multigrid linear solver on multicore and multi-gpu architectures. En *SPE Reservoir Simulation Conference*. Society of Petroleum Engineers, 2017. https://doi.org/10.2118/182718-MS.

[111] Manea A.M. *Parallel multigrid and multiscale flow solvers for high-performance-computing architectures*. Tesis de Doctorado, Stanford University, 2015.

[112] Mathioudakis E.N., Mandikas V.G., Kozyrakis G.V., Kampanis N.A., y Ekaterinaris J.A. Multigrid cell-centered techniques for high-order incompressible flow numerical solutions. *Aerospace Science and Technology*, 64:85–101, 2017. https://doi.org/10.1016/j.ast.2017.01.015.

[113] Meyer-Peter E. y Müller R. Formulas for bed-load transport. En *IAHSR 2nd meeting, Stockholm, appendix 2*. IAHR, 1948.

[114] Micikevicius P. 3d finite difference computation on gpus using cuda. En *Proceedings of 2nd workshop on general purpose processing on graphics processing units*, páginas 79–84. 2009. https://doi.org/10.1145/1513895.1513905.

[115] Min C. On reinitializing level set functions. *Journal of computational physics*, 229(8):2764–2772, 2010. https://doi.org/10.1016/j.jcp.2009.12.032.

[116] Mittal R., Dong H., Bozkurttas M., Najjar F., Vargas A., y Von Loebbecke A. A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. *Journal of*

computational physics, 227(10):4825–4852, 2008.

[117] Mittal R. y Iaccarino G. Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37:239–261, 2005. https://doi.org/10.1146/annurev.fluid.37.061903.175743.

[118] Mohd-Yusof J. Combined immersed boundary/b-spline method for simulations of flows in complex geometries in complex geometries ctr annual research briefs, nasa ames. *NASA Ames/Stanford University*, 1997.

[119] Mohr M. y Wienands R. Cell-centred multigrid revisited. *Computing and Visualization in Science*, 7(3-4):129–140, 2004. https://doi.org/10.1007/s00791-004-0137-0.

[120] Molenaar J. A simple cell-centered multigrid method for 3d interface problems. *Computers & Mathematics with Applications*, 31(9):25–33, 1996. https://doi.org/10.1016/0898-1221(96)00039-9.

[121] Moukalled F., Mangani L., Darwish M., et al. *The finite volume method in computational fluid dynamics*, volumen 113. Springer, 2016.

[122] Naumov M., Arsaev M., Castonguay P., Cohen J., Demouth J., Eaton J., Layton S., Markovskiy N., Reguly I., Sakharnykh N., et al. Amgx: A library for gpu accelerated algebraic multigrid and preconditioned iterative methods. *SIAM Journal on Scientific Computing*, 37(5):S602–S626, 2015. https://doi.org/10.1137/140980260.

[123] Nguyen Y. y Wells J. Modeling bedform development under turbulent flows using large-eddy-simulation and immersed-boundary-method. *Computers & Fluids*, 111:105–113, 2015. http://dx.doi.org/10.1016/j.compfluid.2015.01.014.

[124] Ni M.J. y Abdou M.A. A bridge between projection methods and simple type methods for incompressible navier–stokes equations. *International Journal for Numerical Methods in Engineering*, 72(12):1490–1512, 2007. https://doi.org/10.1002/nme.2054.

[125] Nicolaides R.A. Deflation of conjugate gradients with applications to boundary value problems. *SIAM Journal on Numerical Analysis*, 24(2):355–365, 1987. https://doi.org/10.1137/072402.

[126] Nielsen P. *Coastal bottom boundary layers and sediment transport*, volumen 4. World scientific, 1992.

[127] Niksiar P., Ashrafizadeh A., Shams M., y Madani A.H. Implementation of a gpu-based cfd code. En *2014 International Conference on Computational Science and Computational Intelligence*, volumen 1, páginas 84–89. IEEE, 2014. https://doi.org/10.1109/CSCI.2014.21.

[128] NVIDIA C. Cuda c best practices guide v. 4.0. 2011.

[129] Oosterlee C.W. y Washio T. An evaluation of parallel multigrid as a solver and a preconditioner for singularly perturbed problems. *SIAM Journal on Scientific Computing*, 19(1):87–110, 1998. https://doi.org/10.1137/S1064827596302825.

[130] Osher S. y Sethian J.A. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988. https://doi.org/10.1016/0021-9991(88)90002-2.

[131] Pan D. An immersed boundary method on unstructured cartesian meshes for incompressible flows with heat transfer. *Numerical Heat Transfer, Part B: Fundamentals*, 49(3):277–297, 2006. https://doi.org/10.1080/10407790500290709.

[132] Pan D. A simple and accurate ghost cell method for the computation of incompressible flows over immersed bodies with heat transfer. *Numerical Heat Transfer, Part B: Fundamentals*, 58(1):17–39, 2010. https://doi.org/10.1080/10407790.2010.504697.

[133] Pascau A. Cell face velocity alternatives in a structured colocated grid for the unsteady navier–stokes equations. *International Journal for Numerical Methods in Fluids*, 65(7):812–833, 2011. https://doi.org/10.1002/fld.2215.

[134] Patankar S. *Numerical Heat Transfer and Fluid Flow*. CRC Press, 1980.

[135] Patankar S. y Spalding D. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal of Heat and Mass Transfer*, 15(10):1787–1806, 1972. ISSN 0017-9310. https://doi.org/10.1016/0017-9310(72)90054-3.

[136] Patankar S.V. y Spalding D.B. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. En *Numerical Prediction of Flow, Heat Transfer, Turbulence and Combustion*, páginas 54–73. Elsevier, 1983. https://doi.org/10.1016/B978-0-08-030937-8.50013-1.

[137] Péron S., Benoit C., Renaud T., y Mary I. An immersed boundary method on cartesian adaptive grids for the simulation of compressible flows around arbitrary geometries. *Engineering with Computers*, 37(3):2419–2437, 2021. https://doi.org/10.1007/s00366-020-00950-y.

[138] Perot J.B. An analysis of the fractional step method. *Journal of Computational Physics*, 108(1):51–58, 1993. https://doi.org/10.1006/jcph.1993.1162.

[139] Peskin C.S. *Flow patterns around heart valves: a digital computer method for solving the equations*

of motion. Tesis de Doctorado, ProQuest Dissertations & Theses, 1972. https://search.proquest.com/openview/75b786647f0dc1b3262964f53a6866fe/1.pdf?pq-origsite=gscholar&cbl=18750&diss=y.

[140] Peskin C.S. Flow patterns around heart valves: a numerical method. *Journal of computational physics*, 10(2):252–271, 1972. https://doi.org/10.1016/0021-9991(72)90065-4.

[141] Picot J. y Glockner S. Reduction of the discretization stencil of direct forcing immersed boundary methods on rectangular cells: The ghost node shifting method. *Journal of Computational Physics*, 364:18–48, 2018.

[142] Rajaratnam N. y Nwachukwu B.A. Flow near groin-like structures. *Journal of Hydraulic Engineering*, 109(3):463–480, 1983.

[143] Rhie C.M. y Chow W.L. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA journal*, 21(11):1525–1532, 1983. https://doi.org/10.2514/3.8284.

[144] Rijn L.C.v. Sediment transport, part ii: suspended load transport. *Journal of hydraulic engineering*, 110(11):1613–1641, 1984. https://doi.org/10.1061/(ASCE)0733-9429(1984)110:11(1613).

[145] Roe P.L. Characteristic-based schemes for the euler equations. *Annual review of fluid mechanics*, 18(1):337–365, 1986. https://doi.org/10.1146/annurev.fl.18.010186.002005.

[146] Roos F.W. y Willmarth W.W. Some experimental results on sphere and disk drag. *AIAA journal*, 9(2):285–291, 1971. https://doi.org/10.2514/3.6164.

[147] Roulund A., Sumer B.M., Fredsøe J., y Michelsen J. Numerical and experimental investigation of flow and scour around a circular pile. *Journal of Fluid mechanics*, 534:351–401, 2005. 10.1017/S0022112005004507.

[148] Royston M., Pradhana A., Lee B., Chow Y.T., Yin W., Teran J., y Osher S. Parallel redistancing using the hopf–lax formula. *Journal of Computational Physics*, 365:7–17, 2018. https://doi.org/10.1016/j.jcp.2018.01.035.

[149] Saud Afzal M., Bihs H., Kamath A., y Arntsen Ø.A. Three-dimensional numerical modeling of pier scour under current and waves using level-set method. *Journal of Offshore Mechanics and Arctic Engineering*, 137(3), 2015. https://doi.org/10.1115/1.4029999.

[150] Schaffer S. A semicoarsening multigrid method for elliptic partial differential equations with highly discontinuous and anisotropic coefficients. *SIAM Journal on Scientific Computing*, 20(1):228–242, 1998. https://doi.org/10.1137/S1064827595281587.

[151] Schlichting H. y Gersten K. *Boundary-layer theory*. springer, 2016.

[152] Schlömer O., Grams P.E., Buscombe D., y Herget J. Geometry of obstacle marks at instream boulders—integration of laboratory investigations and field observations. *Earth Surface Processes and Landforms*, 46(3):659–679, 2021. https://doi.org/10.1002/esp.5055.

[153] Shields A. Application of similarity principles and turbulence research to bed-load movement. 1936.

[154] Shinozuka M. Simulation of multivariate and multidimensional random processes. *The Journal of the Acoustical Society of America*, 49(1B):357–368, 1971. https://doi.org/10.1121/1.1912338.

[155] Shinozuka M. y Jan C.M. Digital simulation of random processes and its applications. *Journal of sound and vibration*, 25(1):111–128, 1972. https://doi.org/10.1016/0022-460X(72)90600-1.

[156] Smagorinsky J. General circulation experiments with the primitive equations: I. the basic experiment. *Monthly weather review*, 91(3):99–164, 1963. https://doi.org/10.1175/1520-0493(1963)091%3C0099:GCEWTP%3E2.3.CO;2.

[157] Song Y., Lai Y.G., y Liu X. Improved adaptive immersed boundary method for smooth wall shear. En *World Environmental and Water Resources Congress 2020: Hydraulics, Waterways, and Water Distribution Systems Analysis*, páginas 119–128. American Society of Civil Engineers Reston, VA, 2020. https://doi.org/10.1061/9780784482971.012.

[158] Song Y., Lai Y.G., y Liu X. An improved immersed boundary method for simulating flow hydrodynamics in streams with complex terrains. *Water*, 12(8):2226, 2020. https://doi.org/10.3390/w12082226.

[159] Song Y., Xu Y., y Liu X. Physically based sand slide method in scour models based on slope-limited diffusion. *Journal of Hydraulic Engineering*, 146(11):04020074, 2020. https://doi.org/10.1061/(ASCE)HY.1943-7900.0001814.

[160] Spalding D.B. et al. A single formula for the law of the wall. *Journal of Applied mechanics*, 28(3):455–458, 1961.

[161] Storti M.A., Paz R.R., Dalcin L.D., Costarelli S.D., y Idelsohn S.R. A fft preconditioning technique for the solution of incompressible flow on gpus. *Computers & Fluids*, 74:44–57, 2013. https://doi.org/10.1016/j.compfluid.2012.12.019.

[162] Sweby P.K. High resolution schemes using flux limiters for hyperbolic conservation laws. *SIAM journal on numerical analysis*, 21(5):995–1011, 1984. https://doi.org/10.1137/0721062.

[163] Tabata M. y Itakura K. A precise computation of drag coefficients of a sphere. *Internatio-*

nal Journal of Computational Fluid Dynamics, 9(3-4):303–311, 1998. https://doi.org/10.1080/10618569808940861.

[164] Taira K. y Colonius T. The immersed boundary method: a projection approach. *Journal of Computational Physics*, 225(2):2118–2137, 2007. https://doi.org/10.1016/j.jcp.2007.03.005.

[165] Temam R. Sur l'approximation de la solution des équations de navier-stokes par la méthode des pas fractionnaires (i). *Archive for Rational Mechanics and Analysis*, 32:135–153, 1969. https://doi.org/10.1007/BF00247678.

[166] Temam R. Sur l'approximation de la solution des équations de navier-stokes par la méthode des pas fractionnaires (ii). *Archive for rational mechanics and analysis*, 33:377–385, 1969. https://doi.org/10.1007/BF00247696.

[167] Thibault J. y Senocak I. Cuda implementation of a navier-stokes solver on multi-gpu desktop platforms for incompressible flows. En *47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*, página 758. 2009. https://doi.org/10.2514/6.2009-758.

[168] Trottenberg U., Oosterlee C.W., y Schuller A. *Multigrid*. Elsevier, 2000.

[169] Tseng Y.H. y Ferziger J.H. A ghost-cell immersed boundary method for flow in complex geometry. *Journal of computational physics*, 192(2):593–623, 2003.

[170] Udaykumar H., Mittal R., Rampunggoon P., y Khanna A. A sharp interface cartesian grid method for simulating flows with complex moving boundaries. *Journal of computational physics*, 174(1):345–380, 2001. https://doi.org/10.1006/jcph.2001.6916.

[171] Ujaldón M. Cuda achievements and gpu challenges ahead. En *International Conference on Articulated Motion and Deformable Objects*, páginas 207–217. Springer, 2016. https://doi.org/10.1007/978-3-319-41778-3_20.

[172] Van der Vorst H.A. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing*, 13(2):631–644, 1992. https://doi.org/10.1137/0913035.

[173] Van Doormaal J.P. y Raithby G.D. Enhancements of the simple method for predicting incompressible fluid flows. *Numerical heat transfer*, 7(2):147–163, 1984. https://doi.org/10.1080/01495728408961817.

[174] Van Kan J. A second-order accurate pressure-correction scheme for viscous incompressible flow. *SIAM journal on scientific and statistical computing*, 7(3):870–891, 1986. https://doi.org/10.1137/0907059.

[175] Van Leer B. Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov's method. *Journal of computational Physics*, 32(1):101–136, 1979. https://doi.org/10.1016/0021-9991(79)90145-1.

[176] van Rijn L.C. Sediment transport, part i: Bed load transport. *Journal of Hydraulic Engineering*, 110(10):1431–1456, 1984. https://doi.org/10.1061/(ASCE)0733-9429(1984)110:10(1431).

[177] Verzicco R. y Orlandi P. A finite-difference scheme for three-dimensional incompressible flows in cylindrical coordinates. *Journal of Computational Physics*, 123(2):402–414, 1996. https://doi.org/10.1006/jcph.1996.0033.

[178] Wesseling P. Cell-centered multigrid for interface problems. *Journal of Computational Physics*, 79(1):85–91, 1988. https://doi.org/10.1016/0021-9991(88)90005-8.

[179] Wesseling P. Introduction to multigrid methods. 1995.

[180] Wong M. y Parker G. Reanalysis and correction of bed-load relation of meyer-peter and müller using their own database. *Journal of Hydraulic Engineering*, 132(11):1159–1168, 2006. https://doi.org/10.1061/(ASCE)0733-9429(2006)132:11(1159).

[181] Wu W., Rodi W., y Wenka T. 3d numerical modeling of flow and sediment transport in open channels. *Journal of hydraulic engineering*, 126(1):4–15, 2000. https://doi.org/10.1061/(ASCE)0733-9429(2000)126:1(4).

[182] Xiang Y., Yu B., Yuan Q., y Sun D. Gpu acceleration of cfd algorithm: Hsmac and simple. *Procedia Computer Science*, 108:1982–1989, 2017. https://doi.org/10.1016/j.procs.2017.05.124.

[183] Xu J., Fu H., Luk W., Gan L., Shi W., Xue W., Yang C., Jiang Y., He C., y Yang G. Optimizing finite volume method solvers on nvidia gpus. *IEEE Transactions on Parallel and Distributed Systems*, 30(12):2790–2805, 2019. https://doi.org/10.1109/TPDS.2019.2926084.

[184] Xu Q., Li R., y Xu M. High-performance implementation of parallel semi-implicit method for pressure linked equations solver on cpu+ gpu platform. *International Journal of Heat and Mass Transfer*, 182:121976, 2022. https://doi.org/10.1016/j.ijheatmasstransfer.2021.121976.

[185] Yalin M.S. An expression for bed-load transportation. *Journal of the Hydraulics Division*, 89(3):221–250, 1963. https://doi.org/10.1061/JYCEAJ.0000874.

[186] Ye T., Mittal R., Udaykumar H., y Shyy W. An accurate cartesian grid method for viscous incompressible flows with complex immersed boundaries. *Journal of computational physics*, 156(2):209–240, 1999. https://doi.org/10.1006/jcph.1999.6356.

[187] Zang Y., Street R.L., y Koseff J.R. A non-staggered grid, fractional step method for time-dependent incompressible navier-stokes equations in curvilinear coordinates. *Journal of Computational physics*, 114(1):18–33, 1994. https://doi.org/10.1006/jcph.1994.1146.

[188] Zanke U. Berechnung der sinkgeschwindigkeiten von sedimenten. 1977.

[189] Zaspel P. y Griebel M. Solving incompressible two-phase flows on multi-gpu clusters. *Computers & Fluids*, 80:356–364, 2013. https://doi.org/10.1016/j.compfluid.2012.01.021.

[190] Zeng J., Constantinescu G., y Weber L. A fully 3d non-hydrostatic model for prediction of flow, sediment transport and bed morphology in open channels. En *XXXIst International Association Hydraulic Research Congress*, páginas 554–560. 2005.

[191] Zhou L., Gao J., Hu C., y Li Q. Numerical simulation and testing verification of the interaction between track and sandy ground based on discrete element method. *Journal of Terramechanics*, 95:73–88, 2021. ISSN 0022-4898. https://doi.org/10.1016/j.jterra.2021.03.002.

[192] Zhou Y., Xu B.H., Yu A.B., y Zulli P. An experimental and numerical study of the angle of repose of coarse spheres. *Powder technology*, 125(1):45–54, 2002. https://doi.org/10.1016/S0032-5910(01)00520-4.

Doctorado en Ingeniería
mención Mecánica Computacional

Título de la obra:
Modelos numéricos en GPGPU para el tratamiento de fondos móviles erosionables

Autor: Lucas Carmelo Bessone Martínez

Lugar: Santa Fe, Argentina

Palabras Claves:

GPU, TVD, métodos eulerianos, computación de alto desempeño, volúmenes finitos, C-CUDA, erosión local, transporte de sedimentos