Desarrollo de Software Numérico de Simulación de Flujos Geofísicos Basado en Volúmenes Finitos Usando Hardware Gráfico

TESIS DOCTORAL

Marc de la Asunción Hernández

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada

Octubre 2012

Editor: Editorial de la Universidad de Granada
Autor: Marc de la Asunción Hernández
D.L.: GR 1073-2013
ISBN: 978-84-9028-478-0

Desarrollo de Software Numérico de Simulación de Flujos Geofísicos Basado en Volúmenes Finitos Usando Hardware Gráfico

Memoria que presenta para optar al título de Doctor en Informática

Marc de la Asunción Hernández

Dirigida por los doctores

José Miguel Mantas Ruiz
Manuel Jesús Castro Díaz

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada

Octubre 2012

Agradecimientos

El trabajo descrito en esta tesis doctoral ha sido parcialmente subvencionado por los proyectos DGI-MEC MTM2008-06349-C03-03, DGI-MEC MTM2009-11923 y DGI-MICINN MTM2011-27739-C04-02.

Los cómputos en GPU se han llevado a cabo en el Laboratorio de Métodos Numéricos de la Universidad de Málaga.

Índice

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Capítulo 1

Introducción

La resolución numérica de las ecuaciones de aguas someras resulta útil en varias aplicaciones relacionadas con flujos geofísicos, como por ejemplo, la simulación de flujos en ríos, roturas de presas, inundaciones, transporte de sustancias contaminantes. simulación de corrientes costeras, etc. En particular, el sistema de aguas someras bicapa puede ser útil para simular el intercambio continuo de dos flujos de agua oceánicos (como ocurre en el Estrecho de Gibraltar), corrientes en las proximidades de los deltas de ríos (en la confluencia entre agua dulce y salada) o tsunamis generados mediante avalanchas submarinas donde el material que genera la avalancha se supone parcialmente fluidizado. No obstante, la simulación de estos fenómenos requiere una gran demanda computacional debido al gran tamaño de los dominios espacial y temporal, e incluso pueden exigir resultados en tiempo real. Por este motivo se necesitan resolvedores eficientes para plataformas de altas prestaciones, con objeto de poder manejar escenarios de estas características en tiempos razonables.

Debido al alto grado de paralelismo de datos que presenta la resolución de los sistemas de aguas someras mediante el método de volúmenes finitos, las modernas tarjetas gráficas o Unidades de Procesamiento Gráfico (GPUs) son una buena alternativa de bajo coste para acelerar la resolución numérica de estas simulaciones, ya que una GPU contiene cientos de procesadores capaces de operar en paralelo. Más aún, el uso de clusters de GPUs permite superar las limitaciones de memoria y capacidad de cómputo que tiene una sola GPU, posibilitando el uso de mallados con un gran número de volúmenes y explotando asimismo la capacidad de cómputo en paralelo de todas las tarjetas del cluster.

En los últimos años han aparecido distintas plataformas que posibilitan la programación de aplicaciones de propósito general en GPU, evitando que el programador tenga que tratar con la interfaz gráfica de la GPU. De todas ellas, CUDA (Compute Unified Device Architecture) [NVIb] es la que actualmente goza de mayor popularidad en trabajos relacionados con cien-

cia e ingeniería, permitiendo aprovechar al máximo la arquitectura de las modernas tarjetas gráficas NVIDIA.

A raíz de la aparición de estas plataformas, han surgido varias publicaciones en las que se explota la arquitectura masivamente paralela de las GPUs con el objetivo de mejorar la eficiencia de resolvedores de sistemas de aguas someras (ver el capítulo 2 para una revisión del estado del arte). Sin embargo, hasta donde alcanza nuestro conocimiento, todos los trabajos publicados hasta la fecha en este sentido hacen uso de mallas estructuradas, y en ninguno de ellos se utilizan mallas triangulares arbitrarias ni se aborda la resolución de sistemas bicapa. Una malla triangular, a diferencia de una malla estructurada, permite especificar cualquier dominio irregular e incluso establecer distintas resoluciones espaciales en diferentes zonas del dominio. Resulta importante, por tanto, solventar esta carencia realizando un amplio estudio sobre la mejora de la resolución de sistemas de aguas someras bicapa en mallas triangulares arbitrarias mediante el uso de tarjetas gráficas.

1.1. Objetivos

Exponemos a continuación los principales objetivos que se pretenden conseguir con la realización de esta tesis doctoral. En todos los casos trabajaremos con métodos de volúmenes finitos:

- Mejorar el rendimiento de resolvedores de sistemas de aguas someras bicapa de orden uno para mallas triangulares mediante el uso de tarjetas gráficas NVIDIA y el entorno de programación CUDA. Asimismo sería deseable identificar esquemas numéricos que permitan reducir la alta intensidad computacional que conlleva el uso de sistemas bicapa, conservando en lo posible la calidad de las soluciones obtenidas.

- Adaptar e implementar en GPU un algoritmo de alto orden para la resolución de sistemas de aguas someras bicapa en mallas triangulares, analizando su precisión y eficiencia computacional.

- Diseñar e implementar un algoritmo multi-GPU con el objetivo de mejorar el rendimiento de resolvedores de sistemas de aguas someras bicapa de orden uno para mallas triangulares usando clusters de GPUs de forma eficiente.

- Aplicar estas técnicas para acelerar la resolución numérica de un resolvedor que permita simular escenarios reales en mallas triangulares. Concretamente, consideraremos un esquema numérico de orden uno para la simulación de tsunamis generados mediante avalanchas. Igualmente, se plantea como objetivo la implementación de una herramienta de visualización que permita representar estas simulaciones de forma realista.

- También sería deseable analizar las diferencias obtenidas, en cuanto a precisión y eficiencia computacional, al usar simple o doble precisión numérica en GPU, tanto para esquemas de orden uno como de alto orden.

1.2. Estructura de la Memoria

El resto de esta memoria se estructura del siguiente modo:

- En el capítulo 2 se da una breve introducción a la simulación de flujos geofísicos y a la programación de GPUs y de sistemas multi-GPU. Asimismo se revisa el estado del arte sobre implementación de sistemas de aguas someras en plataformas de alto rendimiento, y se describe el modelo de programación de CUDA y su arquitectura Fermi, que han sido utilizados en esta tesis.

- En el capítulo 3 se adaptan e implementan en GPU varios esquemas numéricos de orden uno para la resolución numérica del sistema de aguas someras bicapa en mallas triangulares, y se realiza un análisis de la precisión y eficiencia computacional de todos ellos.

- En el capítulo 4 se presenta un algoritmo de alto orden para la resolución numérica del sistema de aguas someras bicapa en mallas triangulares, y se describe su adaptación e implementación en GPU para los órdenes 2 y 3. Al igual que en el capítulo 3, se realizan varios experimentos y se analiza la precisión y eficiencia computacional obtenidas.

- En el capítulo 5 se describen dos implementaciones multi-GPU de un esquema de orden 1 presentado en el capítulo 3 para la resolución numerica del sistema de aguas someras bicapa usando mallas triangulares, y se realiza un análisis de su escalabilidad efectuando varios experimentos.

- En el capítulo 6 se adapta e implementa en GPU un esquema numérico para la simulación de tsunamis generados por avalanchas submarinas y subaéreas en mallas estructuradas y triangulares, y se aplica a dos problemas utilizando datos reales. Asimismo se presenta la herramienta de visualización que se ha implementado usando OpenGL.

- En el capítulo 7 se exponen las conclusiones generales, prestando especial atención a los objetivos que se plantearon en la sección 1.1, y se proponen algunas líneas de actuación futuras.

Capítulo 2

Simulación de Flujos Geofísicos Usando GPUs

Resumen: En este capítulo se da una breve introducción a la simulación de flujos geofísicos y a la programación de GPUs y de sistemas multi-GPU, y se hace una revisión de los trabajos previos sobre implementaciones de sistemas de aguas someras en plataformas de alto rendimiento como clústers de ordenadores y GPUs. Asimismo se describe el modelo de programación de CUDA y la arquitectura Fermi, la tercera generación de tarjetas gráficas de NVIDIA con soporte para CUDA. Finalmente se exponen varias posibilidades para visualizar los resultados obtenidos de simulación de flujos geofísicos.

2.1. Introducción

Las ecuaciones de aguas someras, formuladas bajo la forma de sistemas de leyes de conservación con términos fuente, son ampliamente utilizadas para modelar el flujo de una o dos capas de fluido que se supone homogéneo y no viscoso. En el caso de flujos estratificados, se supone que estos son inmiscibles. La resolución numérica de estos sistemas es útil en varias aplicaciones relacionadas con flujos geofísicos. Concretamente, entre sus aplicaciones prácticas, podemos destacar:

- Flujos en ríos, canales, estrechos, lagos, etc.
- Flujos oceánicos.
- Predicción de transporte de sustancia contaminante.
- Predicción del comportamiento de estructuras hidráulicas.

- Flujos moviéndose sobre áreas secas (rotura de presas, inundaciones, etc).
- Gestión de riesgo de inundaciones.
- Estudios sobre el impacto en el medio ambiente de un vertido tóxico.
- Simulaciones de afloramientos, impacto en zonas pesqueras.

Sin embargo, estas simulaciones imponen una gran demanda computacional debido a las dimensiones del dominio, tanto en espacio como en tiempo. Como consecuencia, se requieren simuladores muy eficientes para resolver y analizar estos problemas en tiempos razonables. Dado que la resolución numérica de los sistemas de aguas someras presenta un alto grado de paralelismo de datos (es decir, una misma tarea debe realizarse sobre múltiples datos), existen trabajos en los que se ha tratado la aceleración de estas simulaciones mediante el uso de hardware paralelo. Así, por ejemplo, en [CGGP06] se presenta el esquema numérico de Roe para la simulación 2D de aguas someras y una implementación paralela de dicho esquema en un clúster de PCs. En [CGGP08] se mejora esta implementación paralela mediante el uso de instrucciones optimizadas SSE (Streaming SIMD Extensions) con el objetivo de acelerar los cálculos con matrices y vectores de tamaño pequeño en cada nodo del clúster. A pesar de que estas mejoras hacen posible obtener resultados en menos tiempo computacional, las simulaciones más exigentes todavía requieren demasiado tiempo de ejecución, incluso utilizando clusters con un número elevado de nodos.

2.2. Programación de GPUs

Durante los últimos años, las tarjetas gráficas o Unidades de Procesamiento Gráfico (GPUs) han evolucionado drásticamente. Las modernas GPUs proporcionan cientos de procesadores optimizados para llevar a cabo operaciones aritméticas en paralelo, y pueden ser una solución de bajo coste para acelerar la resolución numérica de modelos matemáticos en ciencia e ingeniería (ver [RS06, OHL+08] para una revisión del tema). Las GPUs resultan especialmente adecuadas en aplicaciones donde existe un alto grado de paralelismo de datos. En este tipo de aplicaciones, por tanto, las GPUs son capaces de alcanzar un rendimiento sustancialmente mayor que el que se podría obtener en una CPU.

Hasta aproximadamente mediados de la década de 2000, el desarrollo de un programa en GPU para tareas no gráficas era una labor difícil y presentaba además varios inconvenientes. En primer lugar, la GPU sólo se podía programar a través de una interfaz gráfica como por ejemplo OpenGL [Shr09]

o Cg [FK03], imponiendo una alta curva de aprendizaje y una interfaz de programación inadecuada para aplicaciones no gráficas. En segundo lugar, el código obtenido era difícil de entender y de mantener. Finalmente, la ausencia de algunas características de programación, como la lectura y escritura en posiciones arbitrarias de memoria o el uso de doble precición numérica (disponible desde hace poco), hacía que algunas aplicaciones científicas no pudieran implementarse en GPU.

Algunas propuestas de implementaciones de simuladores de aguas someras en GPU empleando interfaces gráficas son [HHL+05, LMn+09]. En [HHL+05] se implementa un esquema centrado explícito upwind en una GeForce 7800 GTX para simular un sistema de aguas someras de una capa, consiguiendo una ganancia de 15-30 respecto a una implementación en CPU. En [LMn+09] se describe una implementación en GPU del esquema numérico presentado en [CGGP06], consiguiendo una ganancia de dos órdenes de magnitud en una GeForce 8800 Ultra respecto a una implementación en CPU. Ambas propuestas utilizan OpenGL y el lenguaje de shading Cg.

Con objeto de solventar las anteriores dificultades, en los últimos años han aparecido distintas plataformas que permiten la programación de aplicaciones de propósito general en GPU (GPGPU, General Purpose computation on GPU) evitando que el programador tenga que tratar con los aspectos gráficos de la GPU. En este sentido, en el año 2006 NVIDIA introdujo el entorno de programación *CUDA* (Compute Unified Device Architecture) [NVIb], consistente en una extensión del lenguaje C con soporte para C++ que permite la programación de aplicaciones GPGPU en tarjetas NVIDIA. CUDA es compatible con tarjetas GeForce 8 o superior, Tesla y Quadro. En la sección 2.3 se resume el modelo de programación de CUDA y su arquitectura Fermi.

Otras alternativas surgidas en los últimos años para la programación de aplicaciones de propósito general en GPU son *Brook+* [AMD07] (entorno de programación creado por AMD), *OpenCL* [Khr] (especificación creada por el Grupo Khronos y soportada en tarjetas NVIDIA y AMD) y *DirectCompute* [Mica] (componente de la interfaz gráfica DirectX [Micb] creado por Microsoft y que funciona en sistemas operativos Windows).

Desde la aparición de estas plataformas, se han publicado en la literatura varios trabajos que explotan la arquitectura masivamente paralela de las GPUs con el objetivo de mejorar la eficiencia de resolvedores de sistemas de aguas someras. Ver, por ejemplo, [BP07, BHLN10, BSA12, LHS+09, WAK10]. En la sección 2.4 se citan otros trabajos que hacen uso de sistemas multi-GPU. Dos implementaciones de esquemas numéricos para la simulación 2D de aguas someras con transporte de contaminante inerte pueden verse en [LAA+09] (en GPU usando Brook+) y en [GFA+09] (en un procesador Cell).

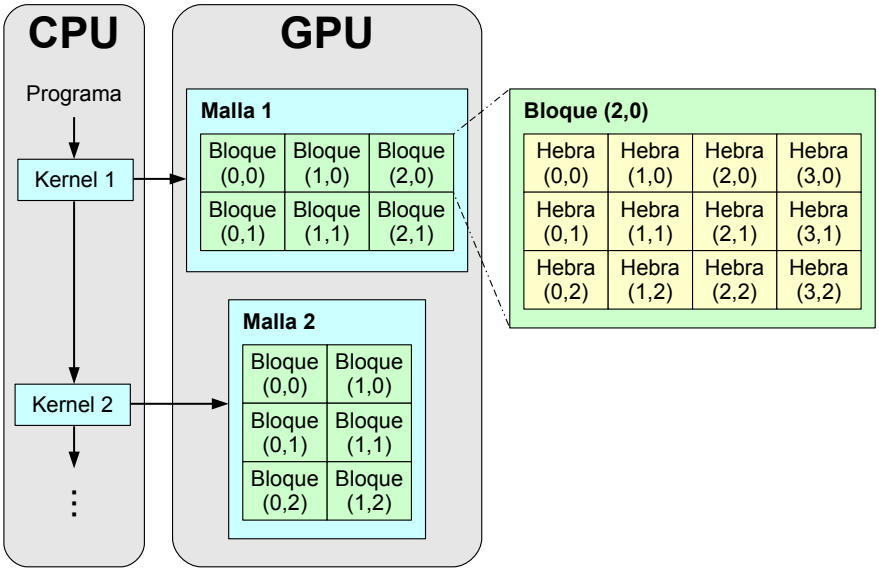


Figura 2.1: Modelo de programación de CUDA.

Se ha elegido CUDA porque es el entorno de programación de GPUs más utilizado en la actualidad, con una amplia comunidad de usuarios, su lenguaje de programación es una extensión de C, tiene versiones para Windows, Linux y Mac OS, y está especialmente diseñado para aprovechar al máximo la arquitectura de las tarjetas NVIDIA.

2.3. Modelo de Programación y Arquitectura de CUDA

2.3.1. Modelo de Programación

Según el marco de trabajo de CUDA, tanto la CPU como la GPU mantienen su propia memoria. Es posible copiar datos de memoria de CPU a memoria de GPU y viceversa.

En CUDA, las funciones que se invocan desde CPU y se ejecutan en GPU se llaman *kernels*. Un kernel se ejecuta en la GPU formando una malla de bloques de hebras, que se ejecutan de forma concurrente, y donde cada hebra ejecuta una instancia del kernel. Todos los bloques y hebras están indexados espacialmente, de forma que la posición espacial de cada hebra puede determinarse en tiempo de ejecución (ver Figura 2.1). La GPU posee varios multiprocesadores (MP) y cada bloque de hebras es asignado a un único multiprocesador para su ejecución. A su vez, cada multiprocesador de

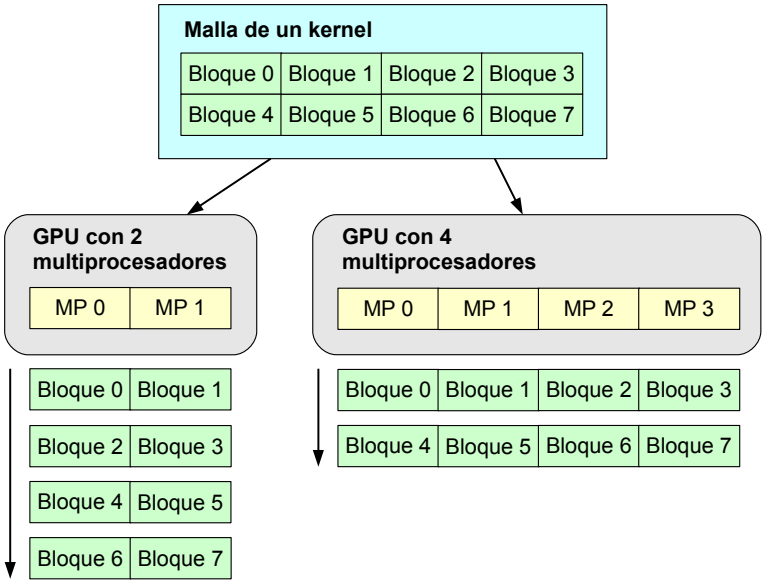


Figura 2.2: Asignación de bloques de hebras a multiprocesadores.

la GPU tiene un determinado número de procesadores, y cada procesador ejecuta una hebra. Cuando los bloques de hebras terminan su ejecución, se lanzan nuevos bloques en los multiprocesadores desocupados. La Figura 2.2 muestra un ejemplo de asignación de bloques de hebras a multiprocesadores.

Cada bloque de hebras se divide en grupos de hebras llamados *warps*, donde el tamaño de un warp es de 32 hebras. En tarjetas Fermi, se pueden ejecutar dos warps en paralelo en cada multiprocesador. Periódicamente un planificador cambia de un warp a otro, lo que permite ocultar la alta latencia cuando se accede a la memoria de la GPU, ya que algunas hebras pueden continuar su ejecución mientras otras hebras están esperando.

Cada hebra que se ejecuta en la GPU tiene acceso a los siguientes espacios de memoria:

- *Registros*: Cada hebra tiene sus propios registros de 32 bits accesibles para lectura y escritura. Cada multiprocesador de la GPU tiene un determinado número de registros, que son divididos y asignados a cada hebra que se ejecuta en ese multiprocesador.

- *Memoria compartida*: Está compartida por todas las hebras de un bloque. En Fermi, su tamaño es configurable a 16 o 48 KB. Es accesible para lectura y escritura sólo desde GPU y es mucho más rápida que la memoria global. Las hebras de un mismo bloque pueden cooperar entre sí mediante el uso de la memoria compartida.

- *Memoria global*: Está compartida por todos los bloques de una malla. Es accesible para lectura y escritura desde CPU y GPU. Es más lenta que la memoria compartida debido a su alta latencia. En tarjetas Fermi, es posible configurar el tamaño de la caché L1 asociada a la memoria global (ver la sección 2.3.2 para más detalles).

- *Memoria de constantes*: Está compartida por todos los bloques de una malla. Es accesible para lectura desde GPU y para escritura desde CPU. Su tamaño es de 64 KB y utiliza una memoria caché cuyo tamaño es de 8 KB por multiprocesador.

- *Memoria de texturas*: Está compartida por todos los bloques de una malla. Es accesible para lectura desde GPU y para escritura desde CPU, aunque si la textura está asociada a memoria lineal de GPU, también es posible escribir en ella desde GPU. Utiliza una caché y está optimizada para localidad espacial 2D, es decir, es especialmente adecuada si cada hebra sólo tiene que acceder a su entorno espacial más cercano en memoria de texturas. El tamaño de la caché varía entre 6 y 8 KB por multiprocesador. El uso de texturas puede resultar una mejor opción que la memoria global si los accesos a memoria de GPU no pueden ser alineados [NVIe].

CUDA también soporta la implementación de operaciones de reducción y escaneo para su ejecución paralela en GPU. Sea v_i, $i = 0, \ldots, N-1$, el elemento i-ésimo de un vector $\boldsymbol{v}$ de N elementos. Una operación típica de reducción es el cálculo de la suma de todos los elementos de $\boldsymbol{v}$, mientras que una operación típica de escaneo es el cálculo, para cada elemento v_i de $\boldsymbol{v}$, de la suma de todos los elementos v_j de $\boldsymbol{v}$, donde $0 \leq j < i$. En [Har07] y [HSO07] se describe cómo implementar en CUDA operaciones de reducción y escaneo, respectivamente.

2.3.2. Arquitectura Fermi

Fermi [NVIc], surgida en 2010, es el nombre de la arquitectura de la tercera generación de tarjetas gráficas NVIDIA con soporte para CUDA, tras la aparición de la arquitectura G80 en 2006 (ejemplo: GeForce 8800 GTX, Tesla C870), y de GT200 en 2008 (ejemplo: GeForce GTX 280, Tesla C1060). Las principales novedades de GT200 frente a G80 fueron un aumento del número de procesadores de GPU (de 128 a 240), y la posibilidad de usar doble precisión numérica.

La Figura 2.3 muestra el esquema general de una GPU con arquitectura Fermi. La GPU está formada por 16 multiprocesadores SIMD (Single Instruction, Multiple Data), y cada uno de ellos tiene 32 procesadores, lo que hace un total de 512 procesadores. En cada ciclo de reloj, cada procesador de un multiprocesador ejecuta la misma instrucción para un warp,

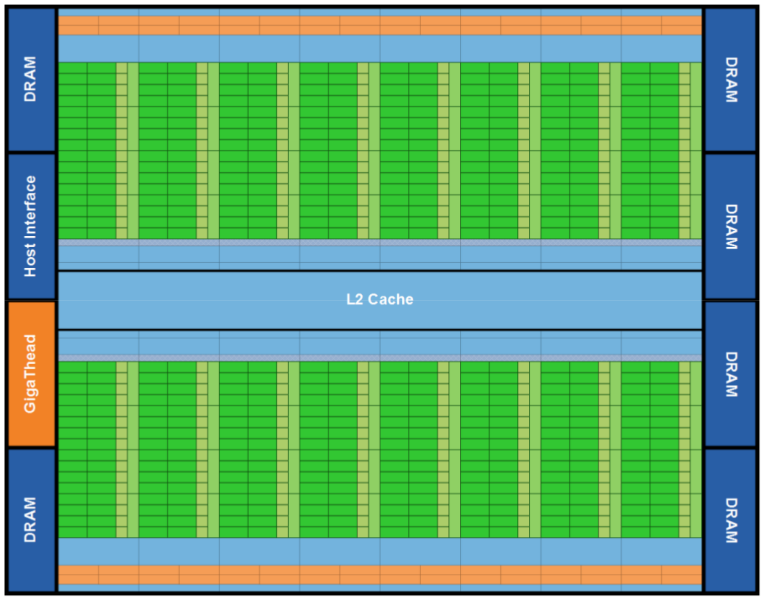


Figura 2.3: Arquitectura Fermi. Fuente: [NVIc]

pero opera sobre datos distintos. La GPU tiene 6 particiones de memoria, soportando hasta un total de 6 GB de memoria GDDR5 DRAM, con una interfaz de memoria de 384 bits. La interfaz de host conecta la GPU con la CPU mediante PCI Express. Por último, el planificador global GigaThread se encarga de distribuir los bloques de hebras entre los multiprocesadores.

2.3.2.1. Estructura del Multiprocesador

La Figura 2.4 muestra la estructura de un multiprocesador de la GPU. Podemos distinguir los siguientes elementos:

- 32 procesadores, donde cada uno de ellos tiene una unidad para operaciones con enteros y otra unidad para operaciones en coma flotante.
- 16 unidades de carga y almacenamiento (LD/ST), permitiendo el cálculo de direcciones de memoria para 16 hebras en un ciclo de reloj.
- 4 unidades de funciones especiales (SFU) que ejecutan funciones trascendentes como seno, coseno o raíz cuadrada. Cada SFU ejecuta una instrucción por hebra y por ciclo de reloj, con lo que un warp ejecuta una función trascendente en 8 ciclos de reloj.
- 2 planificadores de warps y 2 unidades de envío de instrucciones. Periódicamente se seleccionan dos warps y se envía una instrucción de

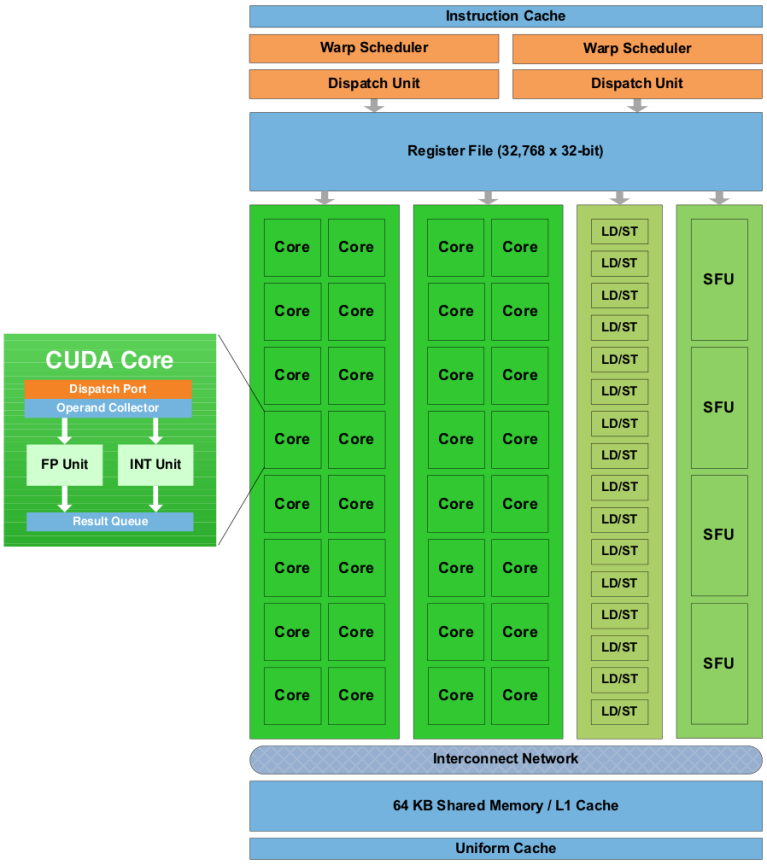


Figura 2.4: Multiprocesador de Fermi. Fuente: [NVIc]

cada warp a 16 procesadores, 16 unidades LD/ST o 4 SFUs. Los warps se ejecutan de forma independiente. Sin embargo, una instrucción en doble precisión de un warp no puede ejecutarse al mismo tiempo que otra instrución de otro warp.

- Una memoria de 64 KB configurable como 48 KB de memoria compartida y 16 KB de caché L1, o bien 16 KB de memoria compartida y 48 KB de caché L1. Las arquitecturas G80 y GT200 tenían un tamaño fijo de 16 KB para la memoria compartida.

En Fermi, las operaciones en doble precisión son 2 veces más lentas que en simple precisión. Esto supone una mejora significativa respecto a la arquitectura GT200, donde eran 8 veces más lentas.

La Tabla 2.1 muestra los valores de algunos parámetros de CUDA en la arquitectura Fermi.

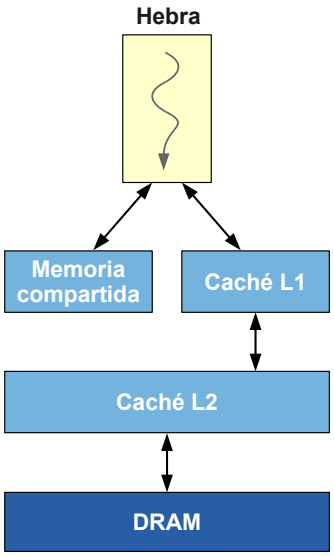


Figura 2.5: Jerarquía de memoria en Fermi.

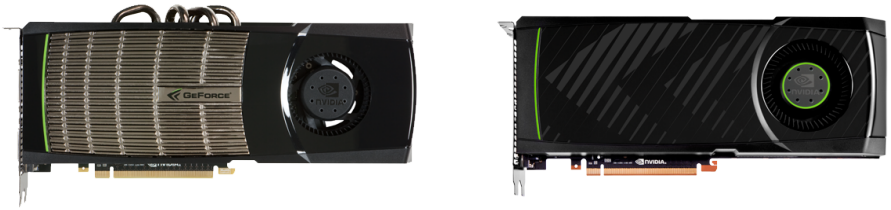
Parámetro	Valor
Número de registros de 32 bits por MP	32768
Máximo número de hebras por bloque	1024
Máximas dimensiones de malla	65535 × 65535 × 65535
Máximas dimensiones de bloque	1024 × 1024 × 64
Máximo número de bloques activos por MP	8
Máximo número de warps activos por MP	48
Máximo número de hebras activas por MP	1536
Máximo número de instrucciones por kernel	512 millones

Tabla 2.1: Valores de algunos parámetros de CUDA en la arquitectura Fermi.

2.3.2.2. Jerarquía de Memorias Caché

Existen aplicaciones para las que la memoria compartida resulta una mejor opción que una memoria caché, mientras que en otros casos la memoria caché es más beneficiosa. Por este motivo, en Fermi se permite al programador configurar la partición entre la memoria compartida y la caché L1. Como se dijo en la sección 2.3.2.1, se pueden asignar 48 KB de memoria compartida y 16 KB de caché L1, o bien 16 KB de memoria compartida y 48 KB de caché L1. Esta partición puede especificarse para cada kernel.

La arquitectura Fermi implementa una jerarquía de memoria unificada para lecturas y escrituras en memoria, que se muestra en la Figura 2.5, con



(a) (b)

Figura 2.6: GPUs Fermi. (a) GeForce GTX 480, (b) GeForce GTX 580.

una caché L1 por multiprocesador y una caché L2 que sirve todas las operaciones (lecturas, escrituras y texturas). El tamaño de la caché L2 es de 768 KB. En programas cuyos accesos a memoria no se conocen de antemano, la configuración de 48 KB para la caché L1 proporciona un mayor rendimiento que el acceso directo a memoria DRAM [NVIc]. La caché L1 también almacena desbordamientos de registros en programas complejos. En las arquitecturas G80 y GT200 este desbordamiento de registros se realizaba en memoria DRAM, aumentando la latencia.

Otras características que incorpora la arquitectura Fermi son la protección basada en ECC (Error Correcting Code) de datos en memoria, la ejecución de hasta 16 kernels de forma concurrente, y operaciones atómicas más eficientes que en GT200 (ver [NVIc] para más detalles).

En 2012 apareció Kepler [NVId], la cuarta arquitectura de tarjetas gráficas NVIDIA con soporte para CUDA (ejemplo: GeForce GTX 680, Tesla K20). Algunas de las mejoras que incorpora esta arquitectura respecto a Fermi son un aumento significativo del número de procesadores (de 512 a 1536), la posibilidad de que un kernel ejecute otros kernels (permitiendo la implementación de algoritmos paralelos más variados) o la posibilidad de que varios procesos de CPU utilicen una única GPU simultáneamente. No obstante, en esta tesis doctoral trabajaremos con tarjetas basadas en Fermi. Concretamente, usaremos la GeForce GTX 480 (480 procesadores a 1401 MHz, 177 GB/seg de ancho de banda y 1536 MB de memoria) y la GeForce GTX 580 (512 procesadores a 1544 MHz, 192 GB/seg de ancho de banda y 1536 MB de memoria). En la Figura 2.6 se muestran imágenes de ambas tarjetas, obtenidas de la web de NVIDIA.

2.4. Programación de Sistemas Multi-GPU

A pesar de que el uso de una GPU ya permite alcanzar el rendimiento requerido en muchas aplicaciones, existen otros casos en los que una sola GPU resulta insuficiente, como por ejemplo problemas donde se trabaje con

mallas de gran tamaño, que requieran un gran número de pasos de tiempo, o incluso que exijan predicciones precisas en tiempo real, como aproximar el efecto de un derrame inesperado de fuel. Estas características sugieren la utilización de varias GPUs en paralelo, superando de este modo las limitaciones de memoria y capacidad de cómputo que tiene una sola GPU.

Un enfoque para el uso de varias GPUs se basa en la programación de sistemas multi-GPU con memoria compartida. Esta aproximación combina una herramienta de programación multihebra en CPU con memoria compartida (ejemplo: OpenMP [CJvdP07]), con otra herramienta de programación de GPUs (ejemplo: CUDA), y ha sido utilizada en diversos trabajos relacionados con mecánica de fluidos [BS10, GRMG11, TS12]. Sin embargo, este enfoque tiene el inconveniente de que sólo permite utilizar una única estación de trabajo con un máximo de 8 GPUs.

Un enfoque menos restrictivo consiste en el uso de clusters de ordenadores, cada uno equipado con una o varias GPUs. A diferencia del enfoque multi-GPU con memoria compartida, en este caso no existen limitaciones en cuanto al número de nodos del cluster ni al número de GPUs totales. De este modo, las limitaciones de memoria propias de un nodo se pueden superar mediante el reparto de los datos entre las memorias disponibles en los diferentes nodos, permitiendo realizar simulaciones en mallas con más volúmenes y aprovechando asimismo la capacidad de procesamiento en paralelo del cluster. Para realizar el particionamiento de una malla triangular en varias submallas existen herramientas como Chaco [HL94], METIS [KK99] o JOSTLE [WC07].

El uso de clusters de GPUs para acelerar cálculos intensivos y, en particular, relacionados con mecánica de fluidos, está ganando en popularidad [MA09, FQKYS04, JTS10, KEGM10, MLF+12]. La mayor parte de las propuestas existentes que emplean clusters de GPUs usan MPI (Message Passing Interface) [MPI] para la comunicación entre los procesos que se ejecutan en diferentes nodos del cluster, y CUDA para programar cada GPU. Una estrategia habitual en sistemas multi-GPU para reducir la sobrecarga de las comunicaciones MPI consiste en usar funciones de comunicación MPI no bloqueantes con el objetivo de solapar las transferencias de datos entre nodos con la computación en GPU y con las tranferencias entre CPU y GPU.

2.5. Visualización de Resultados

Otro aspecto importante en la simulación de flujos geofísicos es la visualización de los resultados obtenidos. En este sentido, una posibilidad es programar nuestra propia herramienta de visualización. Para ello, existen librerías y entornos de programación que nos facilitan la labor permitiendo la creación de gráficos 2D y 3D. Algunos de los más populares son:

- *OpenGL* (Open Graphics Library) [Shr09] es una especificación que define una interfaz (API) para escribir aplicaciones que produzcan gráficos 2D y 3D. Existen implementaciones de OpenGL para Windows, Linux, Mac OS e incluso para consolas como PlayStation 3 y Wii. Actualmente la especificación de OpenGL está controlada por el Grupo Khronos.

- *DirectX* [Micb] es un conjunto de librerías para realizar tareas multimedia, especialmente programación de juegos y vídeo. Fue creado por Microsoft y sólo funciona en plataformas de dicha compañía, tales como sistemas operativos Windows o la consola Xbox 360.

- *VTK* (Visualization Toolkit) [Kitb] es una librería de clases de C++ creada por Kitware para realizar gráficos 3D, procesamiento de imágenes y visualización. Es de código abierto y multiplataforma.

- *Matlab* [Mat] es un entorno de programación creado por MathWorks para el desarrollo de algoritmos, el análisis de datos, la visualización y el cálculo numérico. Tiene versiones para Windows, Linux y Mac OS.

OpenGL y DirectX son herramientas para programación a bajo nivel, mientras que VTK y Matlab tienen un mayor nivel de abstracción. Como es habitual en estos casos, OpenGL y DirectX son las opciones más flexibles, pero tienen el inconveniente de una elevada curva de aprendizaje y una mayor complejidad a la hora de la implementación. En cambio, VTK y Matlab son menos flexibles pero su utilización es más sencilla.

Por último, otra posibilidad para visualizar los resultados obtenidos consiste en usar aplicaciones específicas para la visualización y el análisis de datos científicos. Podemos citar las siguientes:

- *Tecplot* [Tec] es una familia de aplicaciones de visualización y análisis de datos para mecánica de fluidos. Todas las aplicaciones tienen versiones para Windows y Linux.

- *ParaView* [Kita] es una aplicación de código abierto y multiplataforma creada por Kitware para la visualización y el análisis de datos.

Capítulo 3

Esquemas de Orden Uno para el Sistema de Aguas Someras Bicapa

Resumen: En este capítulo se adaptan e implementan en GPU varios esquemas numéricos de orden uno para la resolución numérica del sistema de aguas someras bicapa en mallas triangulares, se realizan varios experimentos y se analizan los resultados obtenidos en términos de precisión y de eficiencia computacional. El contenido de este capítulo ha sido parcialmente publicado en [dlAMC10, COdlA+11, dlAMCFN11, dlAMCFN12, dlAMC12].

3.1. Sistema de Aguas Someras Bicapa

El sistema de aguas someras de una capa es un sistema de leyes de conservación con términos fuente que modela el flujo de un fluido poco profundo homogéneo y no viscoso, bajo la influencia de la gravedad g. En el caso bicapa, el sistema idealmente modela el flujo de un fluido estratificado, que suponemos compuesto por dos capas superpuestas de fluidos inmiscibles, y tiene la siguiente forma:

$$\begin{cases} \dfrac{\partial h_1}{\partial t} + \dfrac{\partial q_{1,x}}{\partial x} + \dfrac{\partial q_{1,y}}{\partial y} = 0 \\ \dfrac{\partial q_{1,x}}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_{1,x}^2}{h_1} + \dfrac{g}{2}h_1^2\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_{1,x}q_{1,y}}{h_1}\right) = -gh_1\dfrac{\partial h_2}{\partial x} + gh_1\dfrac{\partial H}{\partial x} + S_{f_1}(W) \\ \dfrac{\partial q_{1,y}}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_{1,x}q_{1,y}}{h_1}\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_{1,y}^2}{h_1} + \dfrac{g}{2}h_1^2\right) = -gh_1\dfrac{\partial h_2}{\partial y} + gh_1\dfrac{\partial H}{\partial y} + S_{f_2}(W) \\ \dfrac{\partial h_2}{\partial t} + \dfrac{\partial q_{2,x}}{\partial x} + \dfrac{\partial q_{2,y}}{\partial y} = 0 \\ \dfrac{\partial q_{2,x}}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_{2,x}^2}{h_2} + \dfrac{g}{2}h_2^2\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_{2,x}q_{2,y}}{h_2}\right) = -grh_2\dfrac{\partial h_1}{\partial x} + gh_2\dfrac{\partial H}{\partial x} + S_{f_3}(W) \\ \dfrac{\partial q_{2,y}}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_{2,x}q_{2,y}}{h_2}\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_{2,y}^2}{h_2} + \dfrac{g}{2}h_2^2\right) = -grh_2\dfrac{\partial h_1}{\partial y} + gh_2\dfrac{\partial H}{\partial y} + S_{f_4}(W) \end{cases} \tag{3.1}$$

En estas ecuaciones, el subíndice 1 hace referencia a la capa superior, y el subíndice 2 a la capa inferior. $h_i(x, y, t)$ denota el espesor de la capa de agua i en el punto $(x, y) \in D \subset \mathbb{R}^2$ en el instante t, siendo D la proyección horizontal del dominio bidimensional que ocupa el fluido. $H(x, y)$ indica la profundidad a la que se encuentra el fondo en el punto (x, y) medida desde un nivel de referencia fijo, y $\boldsymbol{q}_i(x, y, t) = (q_{i,x}(x, y, t),\, q_{i,y}(x, y, t))$ es el caudal de la capa de agua i en el punto (x, y) en el instante t. El caudal está relacionado con la velocidad promediada en altura $\boldsymbol{u}_i(x, y, t)$ mediante la expresión: $\boldsymbol{q}_i(x, y, t) = h_i(x, y, t)\,\boldsymbol{u}_i(x, y, t)$, $i = 1, 2$. $r = \rho_1/\rho_2$ es el ratio de las densidades constantes de las capas ($\rho_1 < \rho_2$), que en aplicaciones oceanográficas realistas está cercano a 1. Notar que $H(x, y)$ no depende de t, es decir, el fondo no varía a lo largo de la simulación. En la Figura 3.1 se muestra gráficamente la relación entre h_1, h_2 y H.

Los términos $S_{f_i}(W)$, $i = 1, \ldots, 4$, modelan los términos de fricción con el fondo fijo y entre capas, la acción del viento y el efecto Coriolis. Por simplicidad, en este capítulo los supondremos igual a cero.

El sistema (3.1) puede escribirse como un sistema de leyes de conservación con términos fuente y productos no conservativos del siguiente modo:

$$\frac{\partial W}{\partial t} + \frac{\partial F_1}{\partial x}(W) + \frac{\partial F_2}{\partial y}(W) = B_1(W)\frac{\partial W}{\partial x} + B_2(W)\frac{\partial W}{\partial y} + S_1(W)\frac{\partial H}{\partial x} + S_2(W)\frac{\partial H}{\partial y} \tag{3.2}$$

donde

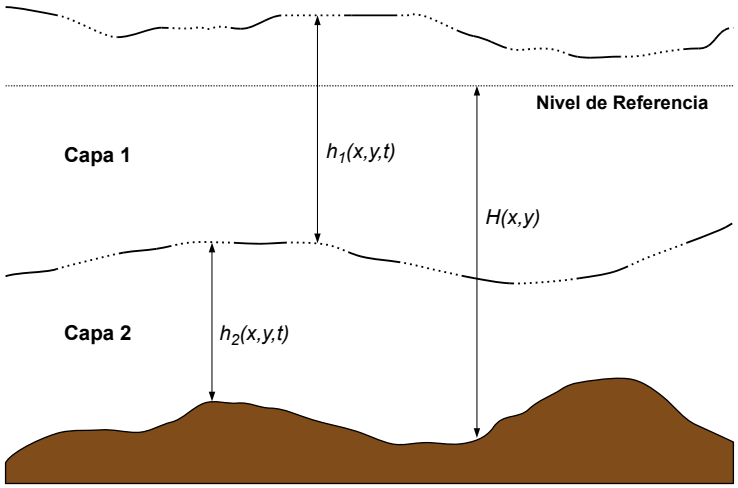


Figura 3.1: Sistema de aguas someras bicapa. Relación entre h_1, h_2 y H.

$$W = \begin{pmatrix} h_1 \\ q_{1,x} \\ q_{1,y} \\ h_2 \\ q_{2,x} \\ q_{2,y} \end{pmatrix}, \quad F_1(W) = \begin{pmatrix} q_{1,x} \\ \frac{q_{1,x}^2}{h_1} + \frac{1}{2}gh_1^2 \\ \frac{q_{1,x}q_{1,y}}{h_1} \\ q_{2,x} \\ \frac{q_{2,x}^2}{h_2} + \frac{1}{2}gh_2^2 \\ \frac{q_{2,x}q_{2,y}}{h_2} \end{pmatrix}, \quad F_2(W) = \begin{pmatrix} q_{1,y} \\ \frac{q_{1,x}q_{1,y}}{h_1} \\ \frac{q_{1,y}^2}{h_1} + \frac{1}{2}gh_1^2 \\ q_{2,y} \\ \frac{q_{2,x}q_{2,y}}{h_2} \\ \frac{q_{2,y}^2}{h_2} + \frac{1}{2}gh_2^2 \end{pmatrix},$$

$$B_1(W) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -gh_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -rgh_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \qquad S_1(W) = \begin{pmatrix} 0 \\ gh_1 \\ 0 \\ 0 \\ gh_2 \\ 0 \end{pmatrix},$$

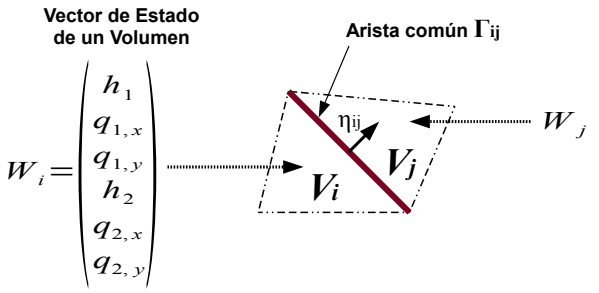


Figura 3.2: Volúmenes finitos para mallas triangulares.

$$B_2(W) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -gh_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -rgh_2 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \qquad S_2(W) = \begin{pmatrix} 0 \\ 0 \\ gh_1 \\ 0 \\ 0 \\ gh_2 \end{pmatrix}. \quad (3.3)$$

3.2. Esquema Roe Clásico

Para discretizar el sistema (3.2), el dominio D se divide en L celdas o volúmenes finitos $V_i \subset \mathbb{R}^2$, $i = 1, \ldots, L$, que se asume que son polígonos cerrados. En esta tesis, dado que trabajamos con mallas triangulares, los volúmenes son triángulos. Dado un volumen finito V_i, $N_i \subset \mathbb{R}^2$ es el baricentro de V_i, $\aleph_i$ es el conjunto de índices j tales que V_j es un vecino de V_i; Γ_{ij} es la arista común de dos volúmenes vecinos V_i y V_j, y $|\Gamma_{ij}|$ es su longitud; $\boldsymbol{\eta}_{ij} = (\eta_{ij,x}, \eta_{ij,y})$ es el vector normal unitario a la arista Γ_{ij} y apunta hacia el volumen V_j (ver Figura 3.2).

Denotamos por W_i^n una aproximación del promedio de la solución en el volumen V_i en el instante t^n:

$$W_i^n \cong \frac{1}{|V_i|} \int_{V_i} W\big(x, y, t^n\big)\, dx\, dy$$

donde $|V_i|$ es el área del volumen y $t^n = t^{n-1} + \Delta t$, siendo Δt el incremento de tiempo.

Conocida la aproximación en el tiempo t^n, W_i^n, para avanzar en el tiempo se considera una familia de problemas de Riemann unidimensionales proyectados en la dirección normal a cada arista $|\Gamma_{ij}|$ de la malla. Estos problemas de Riemann se aproximan mediante un esquema de Roe conservativo por caminos ("path-conservative") [PC04]. Finalmente, las soluciones de estos

problemas de Riemann lineales se promedian en los volúmenes para obtener las aproximaciones del promedio de la solución en cada volumen para el tiempo t^{n+1}. El esquema numérico queda como sigue (ver [CGGP06]):

$$W_i^{n+1} = W_i^n - \frac{\Delta t}{|V_i|} \sum_{j \in \aleph_i} |\Gamma_{ij}| \, \mathcal{F}_{ij}^{ROE^-}(W_i, W_j, H_i, H_j) \tag{3.4}$$

donde

$$\mathcal{F}_{ij}^{ROE^-}(W_i, W_j, H_i, H_j) = P_{ij}^- \left(A_{ij} \left(W_j^n - W_i^n \right) - S_{ij} \left(H_j - H_i \right) \right) + F_{\boldsymbol{\eta}_{ij}}(W_i) \,,$$
$$\mathcal{F}_{ij}^{ROE^+}(W_i, W_j, H_i, H_j) = P_{ij}^+ \left(A_{ij} \left(W_j^n - W_i^n \right) - S_{ij} \left(H_j - H_i \right) \right) - F_{\boldsymbol{\eta}_{ij}}(W_j) \,.$$

$\mathcal{F}_{ij}^{ROE^-}$ y $\mathcal{F}_{ij}^{ROE^+}$ representan las contribuciones de la arista Γ_{ij} a los volúmenes V_i y V_j, respectivamente. $F_{\boldsymbol{\eta}}(W)$ viene dado por:

$$F_{\boldsymbol{\eta}}(W) = F_1(W)\, \eta_x + F_2(W)\, \eta_y \,.$$

Por su parte, $H_\alpha = H(N_\alpha)$, $\alpha = i, j$, y A_{ij} y S_{ij} son las evaluaciones de

$$A\,(W, \boldsymbol{\eta}) = \frac{\partial F_1}{\partial W}(W)\, \eta_x + \frac{\partial F_2}{\partial W}(W)\, \eta_y - \left(B_1(W)\, \eta_x + B_2(W)\, \eta_y \right)$$

y

$$S\,(W, \boldsymbol{\eta}) = S_1(W)\, \eta_x + S_2(W)\, \eta_y$$

en $(W, \boldsymbol{\eta}) = \left(W_{ij}^n, \boldsymbol{\eta}_{ij} \right)$, respectivamente, siendo W_{ij}^n el "estado intermedio" de Roe definido a partir de W_i^n y W_j^n. La matriz $P_{ij}^\pm$ se calcula como:

$$P_{ij}^\pm = \frac{1}{2} K_{ij} \cdot (I \pm \operatorname{sgn}(D_{ij})) \cdot K_{ij}^{-1}$$

donde I es la matriz identidad, D_{ij} es la matriz diagonal cuyos coeficientes son los autovalores de A_{ij}, y K_{ij} es una matriz cuyas columnas son los autovectores asociados. Finalmente, $\operatorname{sgn}(D_{ij})$ es la matriz diagonal cuyos coeficientes son el signo de los autovalores de la matriz A_{ij}.

En el caso del sistema (3.1), se tienen las siguientes expresiones:

$$W_i^n = \begin{pmatrix} h_{1,i}^n \\ q_{1,i,x}^n \\ q_{1,i,y}^n \\ h_{2,i}^n \\ q_{2,i,x}^n \\ q_{2,i,y}^n \end{pmatrix}, \qquad u_{k,i,\alpha}^n = \frac{q_{k,i,\alpha}^n}{h_{k,i}^n} \,, \quad k = 1, 2, \quad \alpha = x, y.$$

A_{ij} se define como sigue:

$$A_{ij} = \begin{pmatrix} J_{ij}^1 & B_{ij}^{1,2} \\ B_{ij}^{2,1} & J_{ij}^2 \end{pmatrix}$$

donde

$$J_{ij}^k = \begin{pmatrix} 0 & \eta_{ij,x} & \eta_{ij,y} \\ J_{ij,21}^k & 2u_{k,ij,x}\eta_{ij,x} + u_{k,ij,y}\eta_{ij,y} & u_{k,ij,x}\eta_{ij,y} \\ J_{ij,31}^k & u_{k,ij,y}\eta_{ij,x} & u_{k,ij,x}\eta_{ij,x} + 2u_{k,ij,y}\eta_{ij,y} \end{pmatrix},$$

$$J_{ij,21}^k = (-u_{k,ij,x}^2 + c_{k,ij}^2)\eta_{ij,x} - u_{k,ij,x}u_{k,ij,y}\eta_{ij,y}\,,$$

$$J_{ij,31}^k = -u_{k,ij,x}u_{k,ij,y}\eta_{ij,x} + (-u_{k,ij,y}^2 + c_{k,ij}^2)\eta_{ij,y}\,,$$

$$B_{ij}^{1,2} = \begin{pmatrix} 0 & 0 & 0 \\ c_{1,ij}^2\eta_{ij,x} & 0 & 0 \\ c_{1,ij}^2\eta_{ij,y} & 0 & 0 \end{pmatrix}, \qquad B_{ij}^{2,1} = \begin{pmatrix} 0 & 0 & 0 \\ rc_{2,ij}^2\eta_{ij,x} & 0 & 0 \\ rc_{2,ij}^2\eta_{ij,y} & 0 & 0 \end{pmatrix}$$

y

$$c_{k,ij} = \sqrt{gh_{k,ij}}\,, \quad u_{k,ij,\alpha} = \frac{\sqrt{h_{k,i}^n}\,u_{k,i,\alpha}^n + \sqrt{h_{k,j}^n}\,u_{k,j,\alpha}^n}{\sqrt{h_{k,i}^n} + \sqrt{h_{k,j}^n}}\,,$$
$$h_{k,ij} = \frac{h_{k,i}^n + h_{k,j}^n}{2}\,, \quad k = 1,2, \quad \alpha = x,y.$$

Finalmente, S_{ij} viene dado por:

$$S_{ij} = \begin{pmatrix} 0 \\ c_{1,ij}^2\eta_{ij,x} \\ c_{1,ij}^2\eta_{ij,y} \\ 0 \\ c_{2,ij}^2\eta_{ij,x} \\ c_{2,ij}^2\eta_{ij,y} \end{pmatrix}.$$

Debido al carácter explícito del esquema numérico, es necesario imponer una condición CFL (Courant–Friedrichs–Lewy) para garantizar su estabilidad lineal. En la práctica, esta condición supone una restricción en el incremento de tiempo, dado por:

$$\Delta t^n = \min_{i=1,\dots,L} \left(\frac{\sum_{j\in\aleph_i} |\Gamma_{ij}| \, \|\mathcal{D}_{ij}\|_\infty}{2\gamma|V_i|} \right)^{-1} \tag{3.5}$$

donde $0 < \gamma \leq 1$ y $\|\mathcal{D}_{ij}\|_\infty$ es la norma infinito de la matriz $\mathcal{D}_{ij}$, es decir, el máximo autovalor de la matriz A_{ij}. El incremento de tiempo resultante puede ser pequeño, lo que significa que puede ser necesario un gran número de iteraciones en problemas donde el tiempo a simular sea elevado.

3.3. Esquema IR-Roe

Una propiedad interesante del sistema (3.2) es que es invariante por rotaciones. Efectivamente, si definimos:

$$T_{\boldsymbol{\eta}} = \begin{pmatrix} R_{\boldsymbol{\eta}} & \mathbf{0} \\ \mathbf{0} & R_{\boldsymbol{\eta}} \end{pmatrix}, \quad R_{\boldsymbol{\eta}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \eta_x & \eta_y \\ 0 & -\eta_y & \eta_x \end{pmatrix}, \quad \forall \boldsymbol{\eta} = (\eta_x, \eta_y) \in \mathbb{R}^2, \quad \|\boldsymbol{\eta}\| = 1$$

y denotamos $F_{\boldsymbol{\eta}}(W) = F_1(W)\eta_x + F_2(W)\eta_y$, $\boldsymbol{B}(W) = (B_1(W), B_2(W))$, y $\boldsymbol{S}(W) = (S_1(W), S_2(W))$, entonces:

$$F_{\boldsymbol{\eta}}(W) = T_{\boldsymbol{\eta}}^{-1} F_1(T_{\boldsymbol{\eta}} W), \qquad T_{\boldsymbol{\eta}} \boldsymbol{B}(W) \cdot \boldsymbol{\eta} = B_1(T_{\boldsymbol{\eta}} W),$$
$$T_{\boldsymbol{\eta}} \boldsymbol{S}(W) \cdot \boldsymbol{\eta} = S_1(T_{\boldsymbol{\eta}} W).$$

Además, se puede comprobar que $T_{\boldsymbol{\eta}} W$ verifica el sistema:

$$\partial_t (T_{\boldsymbol{\eta}} W) + \partial_{\boldsymbol{\eta}} F_1(T_{\boldsymbol{\eta}} W) = B_1(T_{\boldsymbol{\eta}} W)\partial_{\boldsymbol{\eta}} W + S_1(T_{\boldsymbol{\eta}} W)\partial_{\boldsymbol{\eta}} H + Q_{\boldsymbol{\eta}^\perp} \tag{3.6}$$

donde $Q_{\boldsymbol{\eta}^\perp} = T_{\boldsymbol{\eta}} \Big(- \partial_{\boldsymbol{\eta}^\perp} F_{\boldsymbol{\eta}^\perp}(W) + \boldsymbol{B}(W) \cdot \boldsymbol{\eta}^\perp \partial_{\boldsymbol{\eta}^\perp} W + \boldsymbol{S}(W) \cdot \boldsymbol{\eta}^\perp \partial_{\boldsymbol{\eta}^\perp} H \Big)$, siendo $\boldsymbol{\eta}^\perp = (-\eta_y, \eta_x)$.

Nótese que la propiedad de invariancia por rotación nos permite definir un esquema 1D para el sistema definido en (3.6) donde cancelamos el vector $Q_{\boldsymbol{\eta}^\perp}$, que contiene las componentes tangenciales del sistema (ver [FN09] para más detalles). Además, el sistema 1D resultante se puede descomponer en dos partes, una correspondiente al sistema de aguas poco profundas 1D usual, que corresponde a la primera, segunda, cuarta y quinta ecuaciones de (3.6), y dos ecuaciones "de transporte" relacionadas con la componente tangencial de los campos de velocidad, correspondientes a la tercera y sexta ecuaciones de (3.6).

El sistema de aguas poco profundas bicapa 1D es un sistema de cuatro ecuaciones y cuatro incógnitas. Por tanto, el número de autovalores del

mismo es cuatro ($\lambda_1 < \lambda_2 < \lambda_3 < \lambda_4$). Desafortunadamente, no existe una expresión explícita sencilla para los mismos, por lo que habitualmente se recurre a aproximarlos mediante algún algoritmo numérico. En flujos geofísicos bicapa se verifica que $\lambda_1 < 0$ y $\lambda_4 > 0$.

Teniendo en cuenta la descomposición anterior, podemos escribir el siguiente esquema numérico:

$$W_i^{n+1} = W_i^n - \frac{\Delta t}{|V_i|} \sum_{j \in \aleph_i} |\Gamma_{ij}| \, \mathcal{F}_{ij}^{IR-ROE^-}(W_i, W_j, H_i, H_j) \tag{3.7}$$

donde $\mathcal{F}_{ij}^{IR-ROE^\pm}$ se obtiene aplicando los siguientes pasos:

1. Definimos $W_{\boldsymbol{\eta}} = [h_1 \quad q_{1,\boldsymbol{\eta}} \quad h_2 \quad q_{2,\boldsymbol{\eta}}]^{\mathrm{T}} = (T_{\boldsymbol{\eta}} W)_{[1,2,4,5]}$, y $W_{\boldsymbol{\eta}^\perp} = \left[q_{1,\boldsymbol{\eta}^\perp} \quad q_{2,\boldsymbol{\eta}^\perp}\right]^{\mathrm{T}} = (T_{\boldsymbol{\eta}} W)_{[3,6]}$, donde $W_{[i_1,\ldots,i_s]}$ es el vector definido a partir del vector W tomando sus componentes $i_1, \ldots, i_s$.

2. Sea $\Phi^\pm_{\boldsymbol{\eta}_{ij}}$ el flujo de Roe 1D asociado al sistema de aguas poco profundas 1D de dos capas definido por la primera, segunda, cuarta y quinta ecuaciones del sistema (3.6) donde el término $Q_{\boldsymbol{\eta}^\perp}$ ha sido cancelado:

$$\begin{aligned}
\Phi^-_{\boldsymbol{\eta}_{ij}} &= \mathcal{P}^-_{ij}\left(\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}) - \mathcal{B}_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i}\right) - \mathcal{S}_{ij}\left(H_j - H_i\right)\right) \\
&\quad + \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}), \\
\Phi^+_{\boldsymbol{\eta}_{ij}} &= \mathcal{P}^+_{ij}\left(\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}) - \mathcal{B}_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i}\right) - \mathcal{S}_{ij}\left(H_j - H_i\right)\right) \\
&\quad - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}),
\end{aligned}$$

donde

$$\begin{aligned}
\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij}}) = F_1(T_{\boldsymbol{\eta}_{ij}} W)_{[1,2,4,5]}, \quad \mathcal{S}_{ij}(H_j - H_i) &= \left(S_{1,ij}(H_j - H_i)\right)_{[1,2,4,5]}, \\
\mathcal{B}_{ij}(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i}) &= \left(B_{1,ij}(T_{\boldsymbol{\eta}_{ij}}(W_j - W_i))\right)_{[1,2,4,5]}
\end{aligned} \tag{3.8}$$

siendo

$$B_{1,ij} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -gh_{1,ij} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -rgh_{2,ij} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad S_{1,ij} = \begin{pmatrix} 0 \\ gh_{1,ij} \\ 0 \\ 0 \\ gh_{2,ij} \\ 0 \end{pmatrix}.$$

Finalmente, $\mathcal{P}^\pm_{ij} = \frac{1}{2}\mathcal{K}_{ij} \cdot (I \pm \mathrm{sgn}(\mathcal{D}_{ij})) \cdot \mathcal{K}_{ij}^{-1}$, donde I es la matriz identidad, $\mathcal{D}_{ij}$ es la matriz diagonal cuyos coeficientes son los autovalores de $\mathcal{A}_{ij}$, $\mathcal{K}_{ij}$ es una matriz cuyas columnas son los autovectores

asociados, y $\mathrm{sgn}(\mathcal{D}_{ij})$ es la matriz diagonal cuyos coeficientes son el signo de los autovalores de la matriz $\mathcal{A}_{ij}$, siendo

$$\mathcal{A}_{ij} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ gh_{1,ij} - \left(u_{1,\boldsymbol{\eta}_{ij}}\right)^2 & 2u_{1,\boldsymbol{\eta}_{ij}} & gh_{1,ij} & 0 \\ 0 & 0 & 0 & 1 \\ rgh_{2,ij} & 0 & gh_{2,ij} - \left(u_{2,\boldsymbol{\eta}_{ij}}\right)^2 & 2u_{2,\boldsymbol{\eta}_{ij}} \end{pmatrix}$$

con

$$u_{k,\boldsymbol{\eta}_{ij}} = \begin{pmatrix} u_{k,ij,x} & u_{k,ij,y} \end{pmatrix} \cdot \boldsymbol{\eta}_{ij}, \quad u_{k,ij,\alpha} = \frac{\sqrt{h_{k,i}}\, u_{k,i,\alpha} + \sqrt{h_{k,j}}\, u_{k,j,\alpha}}{\sqrt{h_{k,i}} + \sqrt{h_{k,j}}} \ ,$$

$$h_{k,ij} = \frac{h_{k,i} + h_{k,j}}{2}\,, \quad k = 1, 2, \quad \alpha = x, y.$$

3. Sea $\Phi^{\pm}_{\boldsymbol{\eta}^{\perp}_{ij}} = \mp \begin{bmatrix} \left(\Phi^{-}_{\boldsymbol{\eta}_{ij}}\right)_{[1]} u^{*}_{1,\boldsymbol{\eta}^{\perp}_{ij}} & \left(\Phi^{-}_{\boldsymbol{\eta}_{ij}}\right)_{[3]} u^{*}_{2,\boldsymbol{\eta}^{\perp}_{ij}} \end{bmatrix}^{\mathrm{T}}$, donde $u^{*}_{k,\boldsymbol{\eta}^{\perp}_{ij}}$ se define como sigue:

$$u^{*}_{k,\boldsymbol{\eta}^{\perp}_{ij}} = \begin{cases} \frac{q_{k,i,\boldsymbol{\eta}^{\perp}_{ij}}}{h_{k,i}} & \text{si } \left(\Phi^{-}_{\boldsymbol{\eta}_{ij}}\right)_{[2k-1]} > 0 \\ \frac{q_{k,j,\boldsymbol{\eta}^{\perp}_{ij}}}{h_{k,j}} & \text{en otro caso} \end{cases}, \quad k = 1, 2.$$

4. Finalmente, el flujo global se define como $\mathcal{F}^{IR-ROE^{\pm}}_{ij} = T^{-1}_{\boldsymbol{\eta}_{ij}} F^{\pm}_{ij}$, donde

$$F^{\pm}_{ij} = \begin{bmatrix} \left(\Phi^{\pm}_{\boldsymbol{\eta}_{ij}}\right)_{[1]} & \left(\Phi^{\pm}_{\boldsymbol{\eta}_{ij}}\right)_{[2]} & \left(\Phi^{\pm}_{\boldsymbol{\eta}^{\perp}_{ij}}\right)_{[1]} & \left(\Phi^{\pm}_{\boldsymbol{\eta}_{ij}}\right)_{[3]} & \left(\Phi^{\pm}_{\boldsymbol{\eta}_{ij}}\right)_{[4]} & \left(\Phi^{\pm}_{\boldsymbol{\eta}^{\perp}_{ij}}\right)_{[2]} \end{bmatrix}.$$

Nótese que el esquema IR-Roe incluye el cálculo de los autovalores y autovectores de la matriz $\mathcal{A}_{ij}$, de tamaño 4×4, mientras que en el esquema Roe clásico dicho cálculo se realiza sobre una matriz de tamaño 6×6 (ver sección 3.2).

Al igual que en el esquema Roe clásico, imponemos la condición CFL (3.5) para garantizar la estabilidad del esquema.

3.4. Esquemas PVM

Los esquemas PVM (Polynomial Viscosity Matrix) son una familia de esquemas numéricos para sistemas hiperbólicos no conservativos [CFN12]

que se definen en base a matrices de viscosidad obtenidas a partir de la evaluación polinómica de una matriz de Roe. Estos métodos tienen la ventaja de que sólo necesitan cierta información sobre los autovalores del sistema, y no se necesita realizar la descomposición espectral de la matriz de Roe, a diferencia de los dos esquemas descritos en las secciones 3.2 y 3.3. Como consecuencia, son más rápidos que el método de Roe. Los esquemas PVM pueden considerarse una generalización de los esquemas propuestos en [DPRV99] para sistemas no conservativos. Como veremos más adelante, algunos resolvedores conocidos como Rusanov, HLL [HLvL83] o FORCE [TB00] pueden escribirse como un esquema PVM. Los esquemas PVM, cuando se aplican sobre el sistema (3.1), pueden escribirse del siguiente modo:

$$W_i^{n+1} = W_i^n - \frac{\Delta t}{|V_i|} \sum_{j \in \aleph_i} |\Gamma_{ij}| \, \mathcal{F}_{ij}^{PVM^-}(W_i, W_j, H_i, H_j) \tag{3.9}$$

donde $\mathcal{F}_{ij}^{PVM^\pm}$ se obtiene aplicando un proceso similar al que se describió en la sección 3.3 para obtener $\mathcal{F}_{ij}^{IR-ROE^\pm}$. Concretamente, los pasos 1, 3 y 4 son idénticos. Sólo difiere el paso 2, que se redefine como sigue:

2. El flujo 1D $\Phi^\pm_{\boldsymbol{\eta}_{ij}}$ se define del siguiente modo:

$$\begin{aligned}
\Phi^-_{\boldsymbol{\eta}_{ij}} &= \frac{1}{2}\Big(\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}) - \mathcal{B}_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i}\right) - \mathcal{S}_{ij}\left(H_j - H_i\right) \\
&\quad - Q_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i} - \mathcal{A}_{ij}^{-1}\mathcal{S}_{ij}\left(H_j - H_i\right)\right)\Big) + \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}), \\
\Phi^+_{\boldsymbol{\eta}_{ij}} &= \frac{1}{2}\Big(\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}) - \mathcal{B}_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i}\right) - \mathcal{S}_{ij}\left(H_j - H_i\right) \\
&\quad + Q_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i} - \mathcal{A}_{ij}^{-1}\mathcal{S}_{ij}\left(H_j - H_i\right)\right)\Big) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}),
\end{aligned} \tag{3.10}$$

donde $\mathcal{F}_1(W_{\boldsymbol{\eta}_{ij}})$, $\mathcal{S}_{ij}(H_j - H_i)$ y $\mathcal{B}_{ij}(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i})$ están definidos en (3.8).

Q_{ij} es la matriz de viscosidad, definida como $Q_{ij} = P_l^{ij}(\mathcal{A}_{ij})$, siendo $P_l^{ij}(x)$ un polinomio de grado l, $P_l^{ij}(x) = \sum_{k=0}^{l} \alpha_k^{ij} \, x^k$, y $\mathcal{A}_{ij}$ una matriz de Roe, es decir, Q_{ij} puede expresarse del siguiente modo:

$$Q_{ij} = \alpha_0^{ij} I + \alpha_1^{ij} \mathcal{A}_{ij} + \alpha_2^{ij} \mathcal{A}_{ij}^2 + \ldots + \alpha_l^{ij} \mathcal{A}_{ij}^l$$

donde I es la matriz identidad y α_k^{ij}, $k = 0, \ldots, l$, son los coeficientes del polinomio. Así, eligiendo distintos polinomios se obtienen diferentes matrices de viscosidad y, por tanto, distintos métodos. La elección de los polinomios influirá, por un lado, en la estabilidad del esquema numérico y, por otro lado, en la precisión del mismo. En particular,

los esquemas PVM que se consideran en este trabajo son linealmente L^∞ estables con la condición CFL usual.

En adelante, eliminaremos el superíndice ij en la notación de los polinomios, sus coeficientes y los autovalores de la matriz $\mathcal{A}_{ij}$, a fin de simplificar la notación.

3.4.1. Esquemas PVM 1D

Denotamos por PVM-$l(S_0, \ldots, S_k)$ el esquema numérico cuya matriz de viscosidad Q_{ij} está definida en base al polinomio $P_l(x)$, cuyos coeficientes dependen de la elección de los parámetros $S_0, \ldots, S_k$. Estos parámetros están relacionados con la velocidad de propagación de las ondas. A continuación presentamos varios ejemplos de resolvedores PVM:

- **PVM-$(n-1)$U$(\lambda_1, \ldots, \lambda_n)$ o método de Roe**. El esquema de Roe se corresponde con $Q_{ij} = |\mathcal{A}_{ij}|$, donde $|\mathcal{A}_{ij}| = \mathcal{K}_{ij} \, |\mathcal{D}_{ij}| \, \mathcal{K}_{ij}^{-1}$, siendo $\mathcal{D}_{ij}$ la matriz diagonal cuyos coeficientes son los autovalores de $\mathcal{A}_{ij}$, y $\mathcal{K}_{ij}$ la matriz cuyas columnas son los autovectores asociados. Para reescribir el método de Roe como un esquema PVM, Q_{ij} se define del siguiente modo:

$$Q_{ij} = \sum_{k=0}^{n-1} \alpha_k \, \mathcal{A}_{ij}^k$$

donde α_k, $k = 0, \ldots, n-1$ son las soluciones del siguiente sistema de ecuaciones lineal:

$$\begin{pmatrix} 1 & \lambda_1 & \cdots & (\lambda_1)^{n-1} \\ 1 & \lambda_2 & \cdots & (\lambda_2)^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_n & \cdots & (\lambda_n)^{n-1} \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{n-1} \end{pmatrix} = \begin{pmatrix} |\lambda_1| \\ |\lambda_2| \\ \vdots \\ |\lambda_n| \end{pmatrix} \tag{3.11}$$

siendo $\lambda_1 < \ldots < \lambda_n$ los autovalores de la matriz $\mathcal{A}_{ij}$ (en el sistema bicapa 1D, $n = 4$). Nótese que la ecuación k-ésima del sistema (3.11) representa la igualdad $P_{n-1}(\lambda_k) = |\lambda_k|$. Dicho sistema tiene una única solución si todos los autovalores son distintos.

- **PVM-0(S_0) o método de Rusanov**. La elección más simple para un esquema PVM se corresponde con $P_0(x) = S_0$, es decir, $y = P_0(x)$ es una línea horizontal (ver Figura 3.3a). En particular, la elección $S_0 = S_{Rus} = \text{máx}_i \, |\lambda_i|$ se corresponde con el esquema de Rusanov.

- **PVM-1U(S_L, S_R) o método HLL**. El esquema HLL [HLvL83] puede escribirse como un método PVM-1U(S_L, S_R) considerando $S_L = \text{mín}_i\ |\lambda_i|$, $S_R = \text{máx}_i\ |\lambda_i|$, $i = 1, \ldots, n$, y el polinomio

$$P_1(x) = \alpha_0 + \alpha_1 x, \quad \text{tal que } P_1(S_L) = |S_L|,\ P_1(S_R) = |S_R|.$$

  Esto es, $y = P_1(x)$ es la línea recta que pasa por los puntos $(S_L, |S_L|)$ y $(S_R, |S_R|)$ (ver Figura 3.3b). Realizando algunos cálculos elementales, se tiene:

$$\alpha_0 = \frac{S_R|S_L| - S_L|S_R|}{S_R - S_L}, \qquad \alpha_1 = \frac{|S_R| - |S_L|}{S_R - S_L}.$$

  Nótese que si $\lambda_1 = -\lambda_n$, entonces PVM-1U(S_L, S_R) coindice con PVM-0(S_{Rus}).

- **PVM-2(S_0) o métodos de tipo FORCE**. Consideramos en este punto esquemas PVM cuyo polinomio asociado es (ver Figura 3.3a):

$$P_2(x) = \alpha_0 + \alpha_2 x^2, \quad \text{tal que } P_2(S_0) = |S_0|,\ P_2'(S_0) = 1$$

  donde $S_0 = S_{Rus}$. Realizando algunos cálculos, tenemos:

$$\alpha_0 = \frac{S_0}{2}, \qquad \alpha_2 = \frac{1}{2S_0}.$$

  Puede demostrarse que estos esquemas PVM se corresponden con los métodos de tipo FORCE definidos en [TB00].

- **PVM-2U(S_L, S_R) o método de Degond**. En [DPRV99] se propone un esquema numérico para sistemas conservativos, que puede extenderse fácilmente a sistemas no conservativos considerando un esquema PVM asociado al polinomio (ver Figura 3.3b)

$$P_2(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2, \quad \text{tal que } P_2(S_L) = |S_L|,$$
$$P_2(S_R) = |S_R|,\ P_2'(S_L) = \text{sgn}(S_L)$$

  donde

$$S_L = \begin{cases} \lambda_1 & \text{si } |\lambda_1| \geq |\lambda_n| \\ \lambda_n & \text{en otro caso} \end{cases}, \qquad S_R = \begin{cases} \lambda_n & \text{si } |\lambda_1| \geq |\lambda_n| \\ \lambda_1 & \text{en otro caso} \end{cases}$$

  Los coeficientes del polinomio vienen dados por:

$$\alpha_0 = \frac{(S_L)^2 S_R\big(\text{sgn}(S_R) - \text{sgn}(S_L)\big)}{(S_R - S_L)^2}, \quad \alpha_2 = \frac{S_R\big(\text{sgn}(S_R) - \text{sgn}(S_L)\big)}{(S_R - S_L)^2},$$

$$\alpha_1 = \frac{S_L(|S_L| - |S_R|) + S_R\big(\text{sgn}(S_L)S_R - S_L\text{sgn}(S_R)\big)}{(S_R - S_L)^2}.$$

Nótese que si $\lambda_1 = -\lambda_n$, entonces PVM-2U(S_L, S_R) coindice con PVM-2(S_{Rus}).

- **PVM-4(S_L, S_R) y PVM-4(S_0)**. En este punto consideramos esquemas PVM definidos por polinomios bicuadráticos de la forma (ver Figuras 3.3a y 3.3b):

$$P_4(x) = \alpha_0 + \alpha_2 x^2 + \alpha_4 x^4, \quad \text{tal que } P_4(S_L) = |S_L|,$$
$$P_4(S_R) = |S_R|,\ P_4'(S_R) = 1$$

donde

$$S_L = \begin{cases} \lambda_1 & \text{si } |\lambda_1| \geq |\lambda_n| \\ \lambda_n & \text{en otro caso} \end{cases}, \quad S_R = \begin{cases} \max\limits_{2\leq i\leq n} |\lambda_i| & \text{si } |\lambda_1| \geq |\lambda_n| \\ \max\limits_{1\leq i\leq n-1} |\lambda_i| & \text{en otro caso} \end{cases}$$

Los coeficientes del polinomio son:

$$\alpha_0 = \frac{|S_L||S_R|(|S_R| + 2|S_L|)}{2(|S_R| + |S_L|)^2}, \qquad \alpha_2 = \frac{1}{2|S_L|} + \frac{|S_L|}{(|S_R| + |S_L|)^2},$$

$$\alpha_4 = \frac{-1}{2|S_L|(|S_R| + |S_L|)^2}.$$

Nótese que α_0, α_2 y α_4 también están bien definidos si $S_L = S_R = S_0$. En este caso, los coeficientes se reducen a:

$$\alpha_0 = \frac{3S_0}{8}, \qquad \alpha_2 = \frac{3}{4S_0}, \qquad \alpha_4 = \frac{-1}{8(S_0)^3}$$

y el método se denota como PVM-4(S_0), con $S_0 = S_{Rus}$.

- **PVM-IFCP**. El esquema PVM-IFCP (Intermediate Field Capturing Parabola) para el sistema bicapa [FNCP11] está definido por el polinomio $P_2(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2$ (ver Figura 3.3c), donde los coeficientes α_k, $k = 0, 1, 2$, son las soluciones del siguiente sistema:

$$\begin{pmatrix} 1 & \lambda_1 & (\lambda_1)^2 \\ 1 & \lambda_n & (\lambda_n)^2 \\ 1 & \chi_{int} & (\chi_{int})^2 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} |\lambda_1| \\ |\lambda_n| \\ |\chi_{int}| \end{pmatrix} \tag{3.12}$$

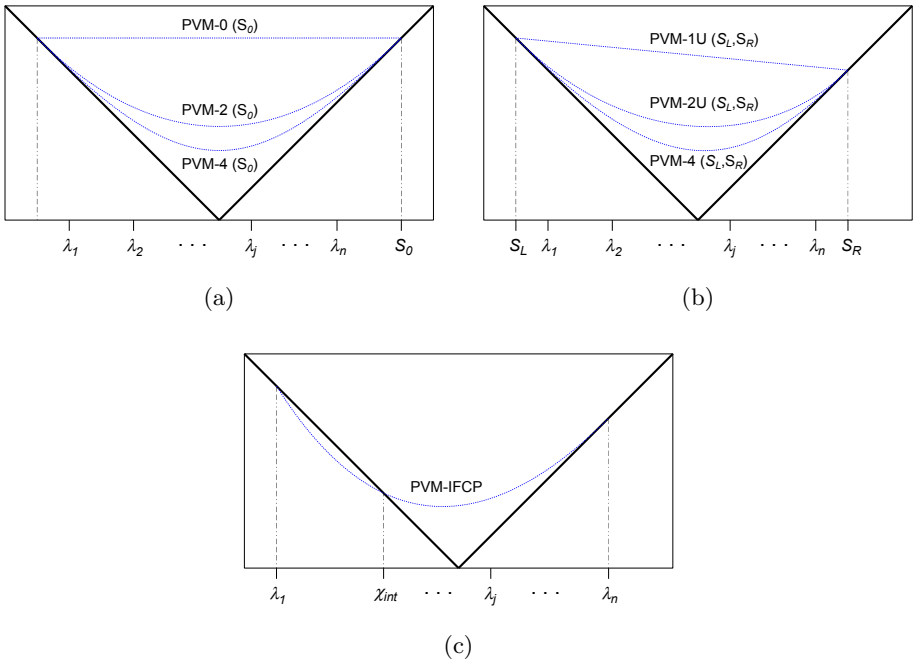


Figura 3.3: Gráficas de los polinomios asociados a cada esquema PVM. En negrita se representa la gráfica $y = |x|$ asociada al método de Roe: (a) PVM-$l(S_0)$, $l = 0, 2, 4$; (b) PVM-lU(S_L, S_R), $l = 1, 2$, y PVM-4(S_L, S_R); (c) PVM-IFCP.

con $\chi_{int} = \mathcal{S}_{ext} \cdot \max\limits_{2\le i\le n-1}(|\lambda_i|)$, y

$$\mathcal{S}_{ext} = \begin{cases} \operatorname{sgn}(\lambda_1 + \lambda_n) & \text{si } (\lambda_1 + \lambda_n) \neq 0 \\ 1 & \text{en otro caso} \end{cases}$$

Las expresiones de los coeficientes α_k, $k = 0, 1, 2$, son:

$$\alpha_0 = \delta_1\lambda_n\chi_{int} + \delta_n\lambda_1\chi_{int} + \delta_{int}\lambda_1\lambda_n, \qquad \alpha_2 = \delta_1 + \delta_n + \delta_{int},$$

$$\alpha_1 = -\lambda_1(\delta_n + \delta_{int}) - \lambda_n(\delta_1 + \chi_{int}) - \chi_{int}(\delta_1 + \delta_n)$$

donde

$$\delta_1 = \frac{|\lambda_1|}{(\lambda_1 - \lambda_n)(\lambda_1 - \chi_{int})}, \qquad \delta_n = \frac{|\lambda_n|}{(\lambda_n - \lambda_1)(\lambda_n - \chi_{int})},$$

$$\delta_{int} = \frac{|\chi_{int}|}{(\chi_{int} - \lambda_1)(\chi_{int} - \lambda_n)}.$$

3.5. Implementación en CUDA

En esta sección se identifican las fuentes de paralelismo de datos existentes en los esquemas numéricos descritos en las secciones 3.2, 3.3 y 3.4. Asimismo, se aborda la adaptación a GPU de estos esquemas y las implementaciones que se han realizado usando CUDA.

3.5.1. Fuentes de Paralelismo

A continuación proponemos un algoritmo paralelo basado en los esquemas numéricos descritos en las secciones 3.2, 3.3 y 3.4. Este algoritmo describe los cálculos que permiten una paralelización de datos de grano fino. La Figura 3.4 muestra esquemáticamente las etapas de dicho algoritmo paralelo, donde los pasos principales aparecen etiquetados con un número dentro de un círculo, y las principales fuentes de paralelismo de datos se representan con rectángulos superpuestos.

En primer lugar, la malla de volúmenes finitos debe construirse e inicializarse convenientemente. La inicialización de la malla consiste en asignar valores iniciales para los vectores de estado W_i y fijar el valor de H_i para cada volumen V_i de la malla, $i = 1, \ldots, L$. Seguidamente se calcula el incremento de tiempo Δt inicial y se entra en el bucle principal, donde se itera el proceso hasta que se alcanza el tiempo final de simulación t_{fin}. A continuación se describen los pasos principales del algoritmo paralelo:

- **Inicialización de los acumuladores**: Al principio de cada iteración del bucle principal, para cada volumen V_i, se inicializan a cero sus acumuladores M_i (un vector 6×1) y Z_i (un escalar). Para cada volumen V_i, los acumuladores M_i y Z_i almacenarán las contribuciones de sus aristas para el cálculo del nuevo vector de estado W_i y para el cálculo del incremento local de tiempo Δt_i, respectivamente.

① **Procesamiento de las aristas**: Para cada arista Γ_{ij} común a dos volúmenes adyacentes V_i y V_j se realizan los siguientes cálculos:

  - Se calculan los vectores $M_{ij}^{\pm} = |\Gamma_{ij}|\, F_{ij}^{\pm}$. M_{ij}^{-} y M_{ij}^{+} representan las contribuciones de la arista al cálculo de los nuevos estados de V_i y V_j, respectivamente. Estas contribuciones deben sumarse a los acumuladores M_i y M_j asociados a V_i y V_j, respectivamente. Este cálculo es el más costoso de todo el proceso, ya que contiene numerosas operaciones con vectores y matrices. En el esquema Roe clásico, $F_{ij}^{\pm}$ se corresponde con $\mathcal{F}_{ij}^{ROE^{\pm}}$ (ver sección 3.2); en el esquema IR-Roe, $F_{ij}^{\pm}$ se corresponde con $\mathcal{F}_{ij}^{IR-ROE^{\pm}}$ (ver sección 3.3), y en los esquemas PVM, $F_{ij}^{\pm}$ se corresponde con $\mathcal{F}_{ij}^{PVM^{\pm}}$ (ver sección 3.4).

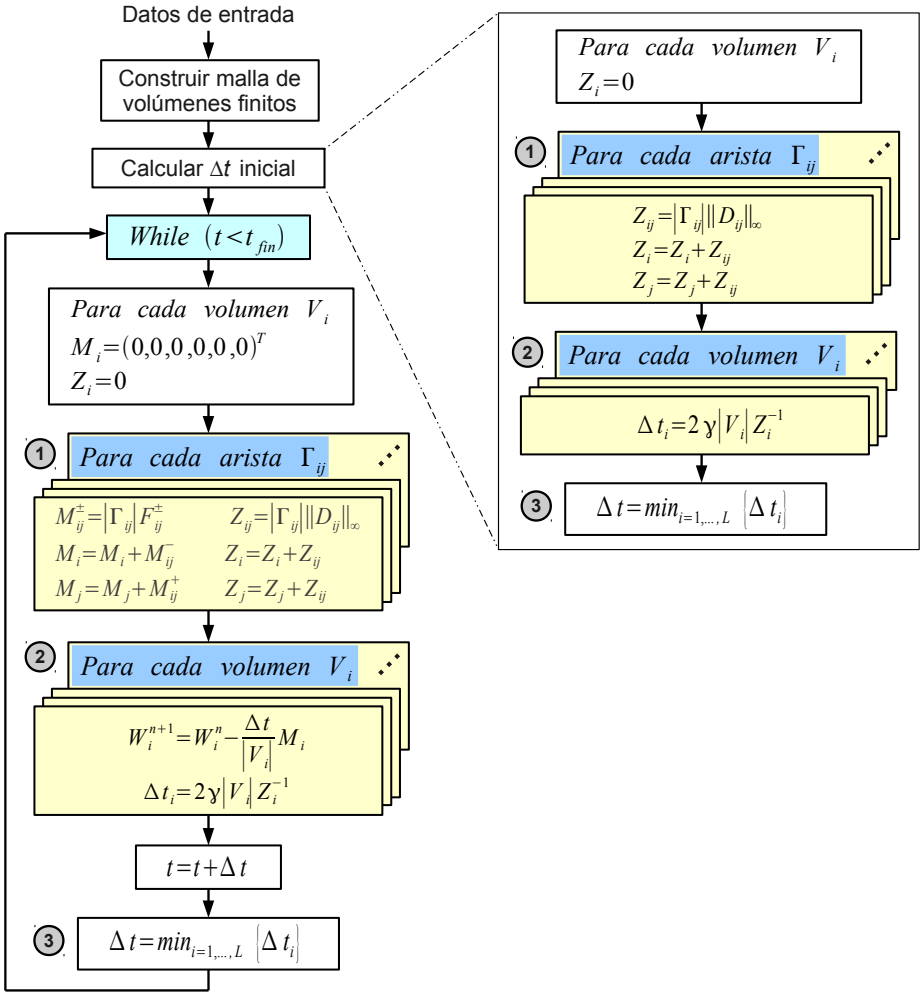


Figura 3.4: Fuentes de paralelismo de los esquemas de orden 1.

- Se calcula el valor $Z_{ij} = |\Gamma_{ij}| \, \|\mathcal{D}_{ij}\|_\infty$. Z_{ij} representa la contribución de la arista al cálculo de los valores locales Δt de V_i y V_j (ver Ecuación (3.5)). Esta contribución debe sumarse a los acumuladores Z_i y Z_j asociados a V_i y V_j, respectivamente.

El procesamiento de las aristas puede hacerse en paralelo, ya que los cálculos asociados a cada arista son independientes de los efectuados para las restantes aristas. Además, nótese que cada arista Γ_{ij} sólo necesita los datos de sus volúmenes vecinos V_i y V_j para llevar a cabo sus cómputos. Por tanto, existe una alta localidad espacial.

② **Cálculo del nuevo estado W_i^{n+1} y del valor local Δt_i para cada volumen**: Para cada volumen V_i se realizan los siguientes cálculos:

- El nuevo estado W_i^{n+1} se obtiene a partir del valor del acumulador M_i del siguiente modo: $W_i^{n+1} = W_i^n - \frac{\Delta t}{|V_i|} M_i$. Esta expresión se corresponde con la Ecuación (3.4) para el esquema Roe clásico, (3.7) para el esquema IR-Roe, y (3.9) para los esquemas PVM.
- El valor local Δt_i se obtiene a partir del valor del acumulador Z_i del siguiente modo (ver Ecuación (3.5)): $\Delta t_i = 2\gamma \, |V_i| \, Z_i^{-1}$.

El procesamiento de los volúmenes también puede hacerse en paralelo, ya que los cálculos asociados a cada volumen son independientes de los realizados para los restantes volúmenes.

③ **Obtención del mínimo Δt**: En este paso se obtiene el mínimo de los valores locales Δt_i de cada volumen V_i. Este mínimo Δt representa el siguiente incremento de tiempo que se aplicará en la simulación. Nótese que este paso también puede paralelizarse mediante la aplicación de un algoritmo de reducción [GKKG03].

Nótese que en los pasos ① y ② del cálculo del incremento de tiempo inicial sólo se efectúan las operaciones necesarias para el cálculo de dicho paso de tiempo a partir de los datos iniciales (ver Figura 3.4).

Debido a que los esquemas numéricos presentan un alto grado de paralelismo de datos, son especialmente adecuados para ser implementados en arquitecturas CUDA.

3.5.2. Detalles de la Implementación

En esta sección describiremos los aspectos más destacados de las implementaciones del algoritmo paralelo expuesto en la sección 3.5.1 que se han realizado usando el entorno de programación CUDA. Se trata de una variante de la implementación descrita en [dlAMC10] para mallas triangulares. Los pasos genéricos del algoritmo implementado se muestran en la Figura 3.5 (la estructura del algoritmo es la misma para todos los esquemas numéricos: Roe clásico, IR-Roe y esquemas PVM). Cada paso que se ejecuta en la GPU se asigna a un kernel de CUDA. A continuación describimos cada uno de los pasos del algoritmo:

- **Construir malla de volúmenes finitos**: En este paso se construye la estructura de datos que se usará en GPU. Para cada volumen V_i, almacenamos su estado (h_1, $q_{1,x}$, $q_{1,y}$, h_2, $q_{2,x}$ y $q_{2,y}$), su profundidad H y su área. Definimos dos arrays de elementos `float4`. El tamaño de ambos arrays es el número de volúmenes. El primer array contiene los

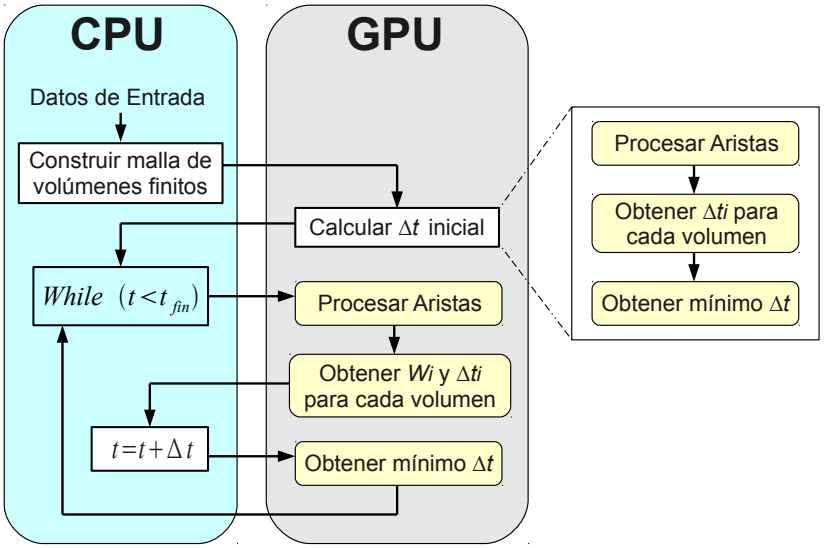


Figura 3.5: Algoritmo paralelo implementado en CUDA para el orden 1.

parámetros h_1, $q_{1,x}$, $q_{1,y}$ y H, mientras que el segundo array contiene h_2, $q_{2,x}$, $q_{2,y}$ y el área. Ambos arrays se almacenan en la memoria de texturas como texturas 1D. Como se dijo en la sección 3.5.1, cada arista sólo necesita los datos de sus dos volúmenes adyacentes para llevar a cabo sus cómputos. Debido a ello, en este caso la memoria de texturas es una mejor opción que la memoria compartida. Esta última es más adecuada cuando todas las hebras de un bloque necesitan acceder a muchos elementos comunes de memoria global, y cada hebra carga una pequeña parte de todos estos elementos en memoria compartida. El uso de texturas 1D, a diferencia de nuestros trabajos descritos en [dlAMC10] y [dlAMC11] en los que usábamos texturas 2D para mallas estructuradas, se debe a que la estructura de una malla triangular no sigue ningún patrón regular. Debido a esto último, a la hora de leer y escribir los estados de los volúmenes, los accesos a memoria global no están alineados, por lo que el uso de la memoria de texturas también es una mejor opción frente al uso de la memoria global [NVIe].

Para cada arista Γ_{ij}, almacenamos su normal ($\eta_{ij,x}$, $\eta_{ij,y}$), las posiciones de los volúmenes V_i y V_j en los anteriores arrays, y dos identificadores correspondientes a dos acumuladores (denotados por un valor `int`: 1, 2 o 3) donde la arista escribe sus contribuciones a los volúmenes V_i y V_j, respectivamente. En la siguiente etapa del algoritmo se describen con más detalle estos acumuladores. Para almacenar toda esta información, usamos dos arrays en memoria global, donde el tamaño de cada array es el número de aristas: un array de elementos `float2` para

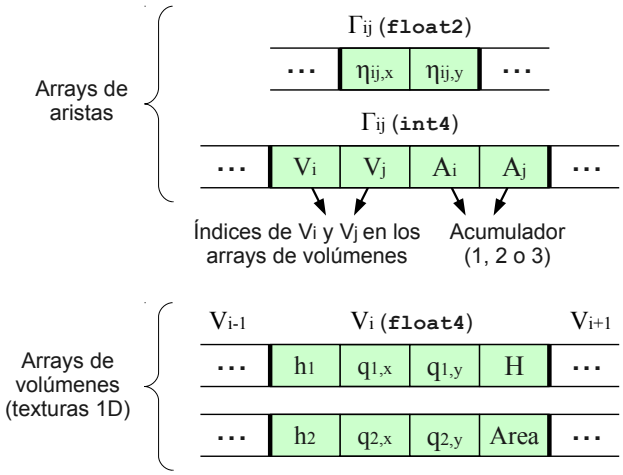


Figura 3.6: Arrays de aristas y volúmenes en GPU.

almacenar la normal, y otro array de elementos `int4` para almacenar los últimos cuatro valores enteros. En la Figura 3.6 se esquematiza el formato de los arrays de aristas y volúmenes.

Podemos saber en tiempo de ejecución si una arista es frontera comprobando si el valor de la posición del volumen V_j es -1.

- **Procesar aristas**: En [dlAMC10] y [dlAMC11], dado que trabajábamos con mallas estructuradas, el procesamiento de las aristas se dividió en procesamiento de aristas horizontales y verticales, permitiendo que algunos términos del cálculo numérico fueran eliminados y mejorando de este modo la eficiencia. Para mallas triangulares esto no es posible. Por tanto, debemos realizar todos los cálculos y usamos un solo kernel para procesar todas las aristas.

  En el procesamiento de aristas, cada hebra representa una arista y calcula las contribuciones de la arista a sus volúmenes adyacentes tal y como se describió en la sección 3.5.1.

  Un aspecto importante es el modo como las aristas (hebras) se sincronizan entre ellas cuando contribuyen a un volumen concreto, es decir, cómo se suman las contribuciones de las aristas para cada volumen. Una arista interna contribuye a sus dos volúmenes adyacentes, mientras que una arista que está en la frontera sólo contribuye a su único volumen adyacente. Esta cuestión se ha resuelto mediante el uso de seis acumuladores en memoria global, donde cada uno de ellos es un array de elementos `float4`. El tamaño de cada acumulador es el número total de volúmenes. Llamemos a los acumuladores 1-1, 1-2, 2-1, 2-2, 3-1 y 3-2. Cada elemento de los acumuladores 1-1, 2-1 y 3-1 almacena las contribuciones de las aristas a la capa 1 de W_i (un vector

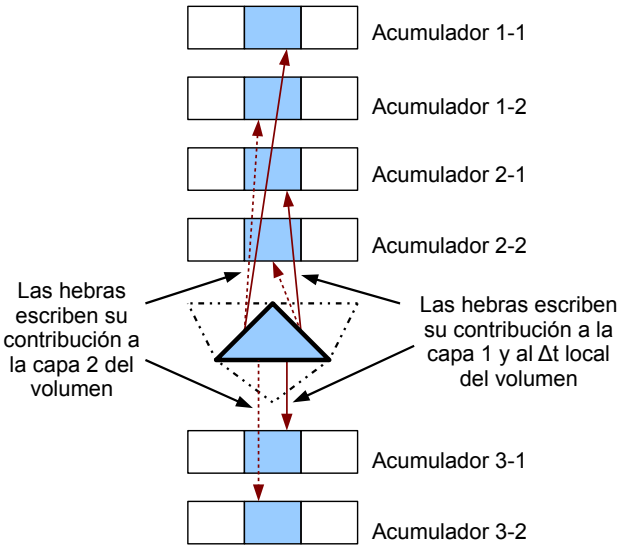


Figura 3.7: Cálculo de la suma de las contribuciones de cada arista para cada volumen.

3×1) y al Δt local del volumen (un valor `float`), mientras que cada elemento de los acumuladores 1-2, 2-2 y 3-2 almacena las contribuciones de las aristas a la capa 2 de W_i (un vector 3×1). La ordenación de los volúmenes en los acumuladores es la misma que en los arrays que almacenan el estado de los volúmenes. La Figura 3.7 muestra el proceso gráficamente.

Nótese que otra opción sería dividir las aristas de la malla en varios conjuntos disjuntos, de forma que cada uno de ellos sólo tenga una arista de cada volumen. De este modo, cada conjunto de aristas se procesaría por separado usando distintos kernels y sólo serían necesarios dos acumuladores en lugar de seis. Este enfoque ahorraría memoria de GPU pero sería más ineficiente debido al mayor número de kernels para procesar las aristas y al mayor número de accesos a memoria de GPU necesarios. Esta aproximación se utilizará en el capítulo 6, cuando trabajemos con un esquema numérico pensado para simular problemas más complejos en los que es necesario garantizar que h_1 y h_2 siempre son mayores o iguales que cero.

- **Obtener W_i^{n+1} y Δt_i para cada volumen**: En este paso, cada hebra representa un volumen y obtiene el siguiente estado W_i^{n+1} (a partir de M_i) y el Δt_i local del volumen V_i (a partir de Z_i) tal y como se describe en el paso ② del algoritmo paralelo expuesto en la sección 3.5.1.

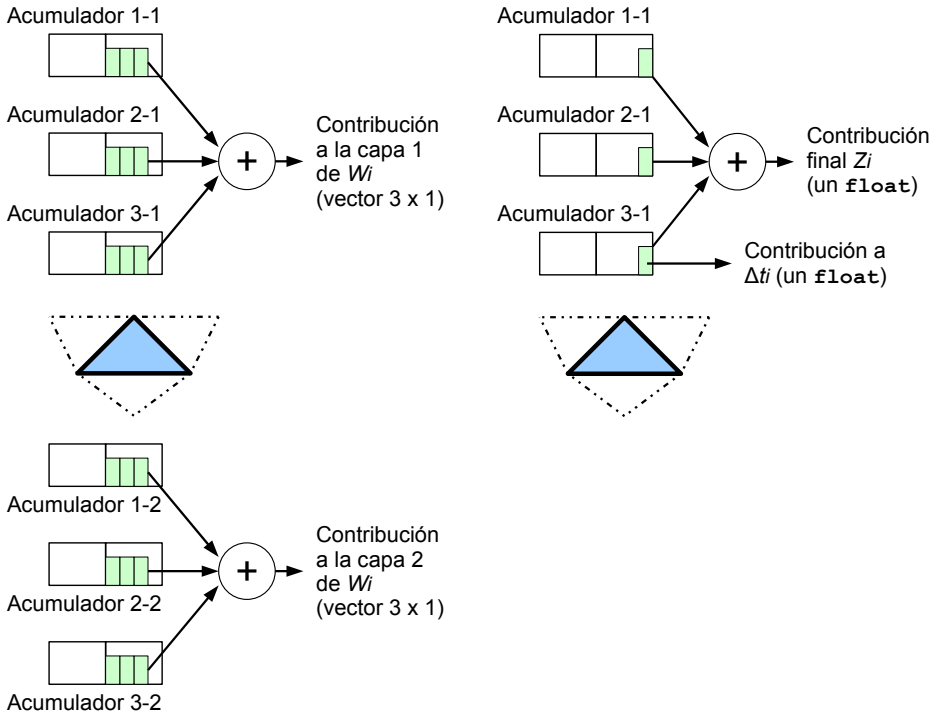


Figura 3.8: Cálculo de la contribución final de las aristas para cada volumen: (a) Contribución al cálculo de W_i; (b) Contribución al cálculo de Δt_i.

La contribución final M_i se obtiene del siguiente modo: los tres primeros elementos de M_i (contribución a la capa 1) se obtienen sumando los tres vectores 3×1 almacenados en las posiciones correspondientes al volumen V_i de los acumuladores 1-1, 2-1 y 3-1, mientras que los tres últimos elementos de M_i (contribución a la capa 2) se obtienen sumando los tres vectores 3×1 almacenados en las posiciones correspondientes al volumen V_i de los acumuladores 1-2, 2-2 y 3-2 (ver Figura 3.8a).

La contribución final Z_i se obtiene sumando los tres valores `float` almacenados en las posiciones correspondientes al volumen V_i de los acumuladores 1-1, 2-1 y 3-1 (ver Figura 3.8b).

- **Obtener el mínimo** Δt: En este paso se obtiene el mínimo de los Δt_i locales de los volúmenes. Dependiendo del número de volúmenes, el cálculo del mínimo se realiza en CPU o en GPU. Si el número de volúmenes es mayor que un determinado valor umbral (elegido experimentalmente), aplicamos un algoritmo de reducción en GPU (el algoritmo de reducción utilizado es el kernel 7 –el más optimizado– del

ejemplo de reducción incluido en el GPU Computing SDK de NVIDIA [NVIa]). En otro caso, el array de los valores Δt_i se copia de GPU a CPU, y el proceso de la obtención del mínimo se lleva a cabo en CPU, ya que resulta más eficiente.

También se han hecho implementaciones de este algoritmo en CUDA usando doble precisión para todos los esquemas numéricos. Las diferencias respecto a las implementaciones en simple precisión son las siguientes:

- Los datos de los volúmenes, en lugar de almacenarse en dos arrays de elementos `float4`, se almacenan en cuatro arrays de elementos `double2`, donde los tres primeros arrays contienen el estado de los volúmenes y el último array contiene la profundidad H y el área de los volúmenes.

- En lugar de seis acumuladores de elementos `float4`, se utilizan nueve acumuladores de elementos `double2` (donde se almacenan las contribuciones a W_i) y tres acumuladores de elementos `double` (donde se almacenan las contribuciones al Δt local de cada volumen).

Finalmente, comentar que en la implementación CUDA del esquema Roe clásico en simple precisión, la parte correspondiente al cálculo de los autovalores y autovectores de la matriz A_{ij} (ver sección 3.2) se efectúa en doble precisión debido a la aparición de inestabilidad numérica si todos los cálculos se realizan en simple precisión.

3.6. Resultados Experimentales

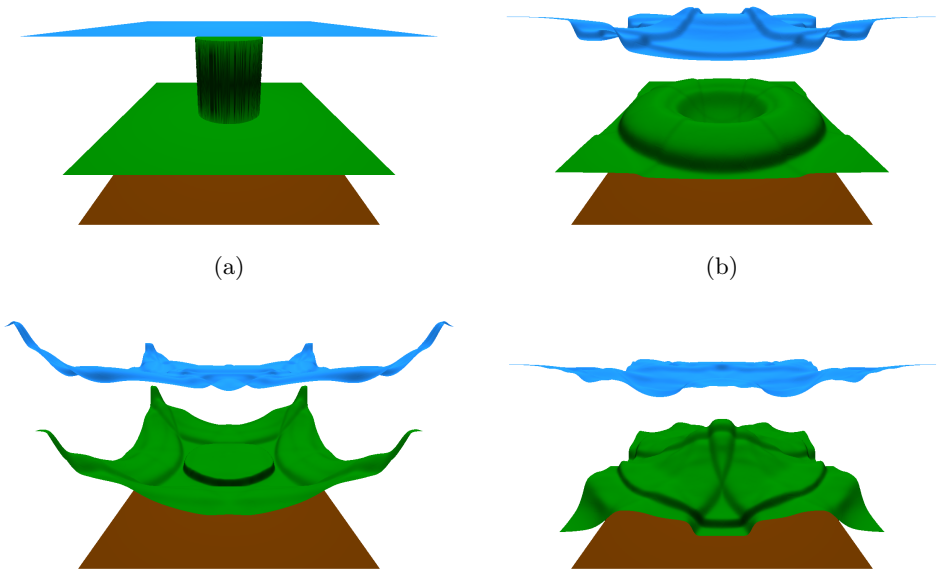
En esta sección ejecutaremos las implementaciones CUDA descritas en la sección 3.5.2 utilizando distintos esquemas numéricos y problemas de ejemplo, y analizaremos los resultados obtenidos, tanto en términos de precisión como de eficiciencia computacional.

3.6.1. Problemas de Ejemplo

Para la realización de los distintos experimentos, consideraremos dos problemas de ejemplo:

- **Ejemplo 1**. Este ejemplo consiste en la rotura de una presa circular interna en el dominio $[-5,5]\times[-5,5]$. La profundidad del fondo viene dada por $H(x,y)=6$ y las condiciones iniciales son:

$$W_i^0(x,y) = \big(h_1(x,y),\quad 0,\quad 0,\quad h_2(x,y),\quad 0,\quad 0\big)^{\mathrm{T}}$$



(a) (b) (c) (d)

Figura 3.9: Evolución del fluido en el ejemplo 1 con una malla triangular de 1024000 volúmenes y el esquema IR-Roe: (a) 0 seg; (b) 1 seg; (c) 2 seg; (d) 3 seg.

donde

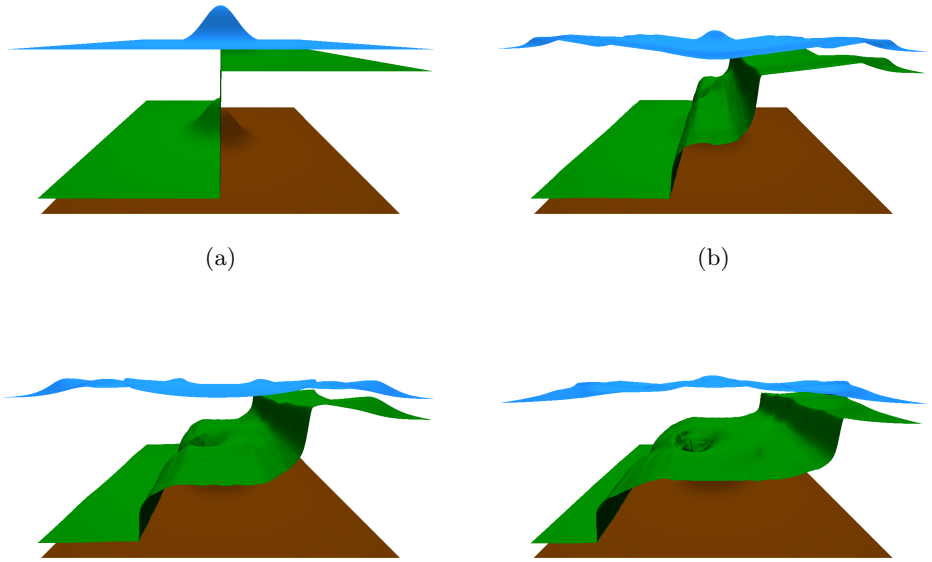
$$h_1(x,y) = \begin{cases} 4.0 & \text{si } \sqrt{x^2+y^2} > 1.5 \\ 0.5 & \text{en otro caso} \end{cases}, \qquad h_2(x,y) = H - h_1(x,y)$$

El ratio de densidades es $r = 0.5$ y el parámetro CFL es $\gamma = 0.9$. Se consideran condiciones de contorno de tipo pared ($q_1 \cdot \boldsymbol{\eta} = 0,\ q_2 \cdot \boldsymbol{\eta} = 0$). La Figura 3.9 muestra la evolución del fluido en distintos instantes de tiempo para una malla triangular de 1024000 volúmenes usando el esquema IR-Roe.

- **Ejemplo 2**. Este ejemplo consiste en dos capas de agua superpuestas en el dominio $[-5,5] \times [-5,5]$. La función de profundidad viene dada por $H(x,y) = 1 - 1.5 \cdot e^{-x^2-y^2}$ y el estado inicial es:

$$W_i^0(x,y) = \big(h_1(x,y),\quad 0,\quad 0,\quad h_2(x,y),\quad 0,\quad 0\big)^{\mathrm{T}}$$

donde



(a) (b)

(c) (d)

Figura 3.10: Evolución del fluido en el ejemplo 2 con una malla triangular de 1024000 volúmenes y el esquema IR-Roe: (a) 0 seg; (b) 2 seg; (c) 4 seg; (d) 6 seg.

$$h_1(x,y) = \begin{cases} 4.0 & \text{si } x \geq 0 \\ 0.5 & \text{en otro caso} \end{cases}, \quad h_2(x,y) = \begin{cases} 0.5 & \text{si } x \geq 0 \\ 4.0 & \text{en otro caso} \end{cases}$$

El ratio de densidades es $r = 0.98$ y el parámetro CFL es $\gamma = 0.9$. Se consideran condiciones de contorno de tipo pared ($q_1 \cdot \boldsymbol{\eta} = 0,\ q_2 \cdot \boldsymbol{\eta} = 0$). La Figura 3.10 muestra la evolución del fluido en distintos instantes de tiempo para una malla triangular de 1024000 volúmenes usando el esquema IR-Roe.

Para la generación de las distintas mallas triangulares de ejemplo, se ha utilizado el toolbox Partial Differential Equation de MATLAB [Mat]. Inicialmente se ha creado una malla de 4000 volúmenes. A continuación se han hecho sucesivos refinamientos obteniendo cada malla a partir de la anterior dividiendo cada triángulo en cuatro hasta llegar a una malla de 1024000 volúmenes. Además se ha creado otra malla de 2080560 volúmenes usando el mismo software.

Volúmenes	Roe clásico		IR-Roe		PVM-IFCP	
	MATLAB	RCM	MATLAB	RCM	MATLAB	RCM
4000	0.069	0.072	0.020	0.022	0.0048	0.0050
16000	0.44	0.38	0.092	0.081	0.020	0.024
64000	3.13	2.36	0.65	0.51	0.14	0.15
256000	22.91	15.12	4.68	3.26	1.01	1.00
1024000	167.2	99.98	33.97	21.34	7.81	7.47
2080560	625.6	384.6	127.1	78.58	29.37	27.98

Tabla 3.1: Tiempos de ejecución en segundos para el ejemplo 1 antes y después de aplicar al algoritmo RCM usando una GeForce GTX 580.

Volúmenes	Roe clásico		IR-Roe		PVM-IFCP	
	MATLAB	RCM	MATLAB	RCM	MATLAB	RCM
4000	0.059	0.061	0.022	0.020	0.0042	0.0045
16000	0.37	0.34	0.11	0.089	0.017	0.020
64000	2.71	2.19	0.79	0.53	0.11	0.12
256000	20.97	14.80	6.06	3.31	0.85	0.86
1024000	166.9	103.5	46.07	21.35	6.73	6.37
2080560	623.8	395.7	169.2	74.66	25.15	23.40

Tabla 3.2: Tiempos de ejecución en segundos para el ejemplo 2 antes y después de aplicar al algoritmo RCM usando una GeForce GTX 580.

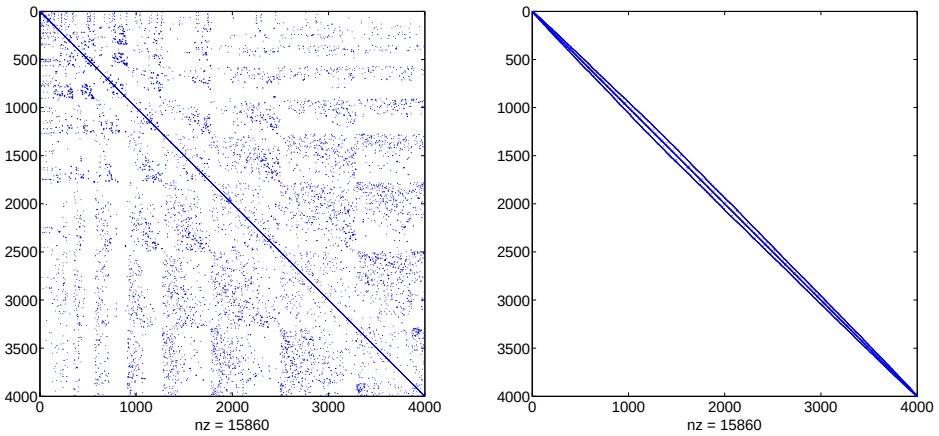
3.6.2. Influencia de la Ordenación de los Volúmenes

Un modo de tratar de mejorar la eficiencia de los códigos CUDA desarrollados es reordenar los volúmenes en los arrays que almacenan los datos de los volúmenes con el objetivo de aumentar la localidad espacial, de forma que volúmenes que son adyacentes en la malla estén en posiciones cercanas en memoria global y de texturas, permitiendo así un mejor uso de las cachés L1 y de texturas. En esta sección estudiaremos la influencia que tiene en los tiempos obtenidos en GPU la ordenación de los volúmenes en los arrays.

Para ello, ejecutaremos los programas CUDA en simple precisión de los esquemas Roe clásico, IR-Roe y PVM-IFCP sobre los dos ejemplos descritos en la sección 3.6.1 considerando dos ordenaciones de volúmenes: la original proporcionada por MATLAB y la resultante de aplicar el algoritmo invertido de Cuthill-McKee a las mallas originales.

El algoritmo de Cuthill-McKee [CM69] es un algoritmo para reducir el ancho de banda[1] de una matriz dispersa. El algoritmo invertido de Cuthill-McKee (RCM, Reverse Cuthill-McKee) [Geo71] es una modificación del algoritmo original donde se invierte el orden de los índices resultantes. El

[1]Definimos el ancho de banda de una matriz A de tamaño $n \times n$ como: máx$\{|i-j|$, $1 \leq i, j \leq n$ tal que $A(i,j) \neq 0\}$.



(a) (b)

Figura 3.11: Matrices de adyacencia de la malla de 4000 volúmenes antes y después de aplicar el algoritmo RCM. `nz` indica el número de elementos distintos de cero: (a) Malla original obtenida con MATLAB; (b) Malla obtenida tras aplicar el algoritmo RCM.

algoritmo RCM normalmente proporciona un perfil[2] de la matriz más reducido que el algoritmo original, por lo que suele ser el más utilizado.

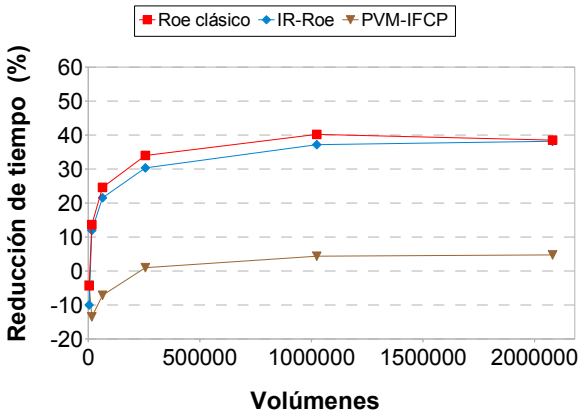
Por su parte, la ordenación de las aristas en los vectores que almacenan los datos de las aristas depende de la ordenación de los volúmenes. Concretamente, para ordenar las aristas se van recorriendo consecutivamente los volúmenes según su ordenación y, para cada uno de ellos, se recorren sus tres aristas y se añaden sus datos a los vectores en el caso de que aún no se hayan añadido.

Para la aplicación del algoritmo RCM se ha utilizado la función `symrcm` de MATLAB. El tiempo de simulación es de 0.1 segundos y los programas CUDA se ejecutarán en una tarjeta GeForce GTX 580.

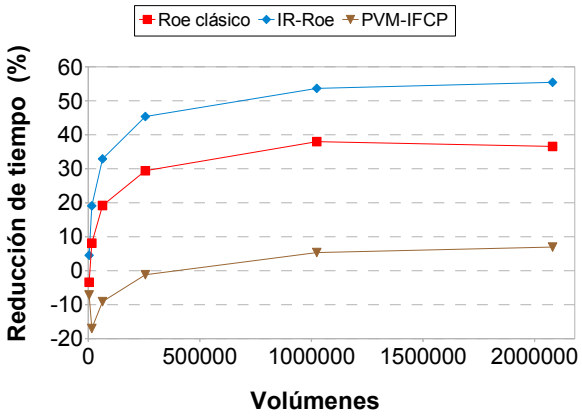
Para antes y después de aplicar el algoritmo RCM, las Tablas 3.1 y 3.2 muestran los tiempos de ejecución en segundos para los ejemplos 1 y 2, respectivamente, la Figura 3.11 representa gráficamente las matrices de adyacencia de la malla de 4000 volúmenes, y la Tabla 3.3 muestra el ancho de banda de todas las mallas. La Figura 3.12 representa la reducción de tiempo obtenida con RCM respecto a las mallas generadas con MATLAB para ambos ejemplos.

Podemos ver que, para las mallas más grandes, los tiempos obtenidos con RCM han sido mejores en todos los casos. Concretamente, con los esquemas Roe clásico e IR-Roe, los tiempos se han reducido aproximadamente entre

[2]Definimos el perfil de una matriz A de tamaño $n \times n$ como: $\sum_{i=1}^{n}(i - f_i)$, donde $f_i = \text{mín}\{j \text{ tal que } A(i,j) \neq 0\}$.



(a)



(b)

Figura 3.12: Reducción del tiempo de ejecución obtenida con RCM respecto a las mallas originales generadas con MATLAB en una GeForce GTX 580: (a) Ejemplo 1; (b) Ejemplo 2.

Volúmenes	MATLAB	RCM
4000	3786	73
16000	12000	147
64000	48000	294
256000	192000	598
1024000	768000	1206
2080560	1996159	1728

Tabla 3.3: Ancho de banda de las mallas antes y después de aplicar el algoritmo RCM.

un 37 y un 55 % para las mallas de más de un millón de volúmenes en ambos ejemplos, mientras que en el esquema PVM-IFCP esta reducción ha sido del 5 %. La reducción del tiempo de ejecución parece ser mayor en esquemas con alta intensidad computacional. Por tanto, la ordenación de volúmenes resultante de aplicar el algoritmo RCM será la que se usará a lo largo de esta tesis.

3.6.3. Análisis de Precisión

En esta sección analizaremos la precisión de las soluciones obtenidas con todos los esquemas numéricos presentados en las secciones 3.2, 3.3 y 3.4. Para ello, para cada uno de los dos ejemplos descritos en la sección 3.6.1, obtendremos en primer lugar una solución de referencia con la malla de 1024000 volúmenes usando el esquema IR-Roe para un tiempo de simulación dado. En segundo lugar obtendremos, para el mismo tiempo de simulación, distintas soluciones con la malla de 64000 volúmenes empleando todos los esquemas numéricos. Finalmente, calcularemos la norma L^1 relativa al dominio D de la diferencia entre la solución de referencia y cada una de las restantes soluciones. Dado que la malla de 1024000 volúmenes es un refinamiento de la malla de 64000 volúmenes, dicha norma L^1 viene dada por:

$$\frac{1}{|D|}\sum_{i=1}^{L}\left|\,|V_i|\,s_i-\sum_{j=1}^{P}|V_{ij}|\,s_{ij}\right|$$

donde $|D|$ es el área del dominio D, s_i es el valor de la variable de estado (h_1, $q_{1,x}$, $q_{1,y}$, h_2, $q_{2,x}$ o $q_{2,y}$) del volumen V_i de la malla gruesa, P es el número de volúmenes en el que se subdivide cada volumen de la malla gruesa (en nuestro caso, 16), V_{ij} es el volumen j-ésimo interior a V_i, y s_{ij} es el valor que tiene la variable de estado del volumen V_{ij} de la malla fina.

Para el ejemplo 1 consideraremos un tiempo de simulación de 1 segundo, mientras que para el ejemplo 2 será de 4 segundos. Todas las ejecuciones se realizarán con las implementaciones CUDA de doble precisión. La tabla 3.4 muestra las normas L^1 obtenidas en el ejemplo 1 para todos los esquemas numéricos, ordenados del que más se aproxima a la solución de referencia al que menos. La tabla 3.5 muestra la misma información para el ejemplo 2. En las Figuras 3.13 y 3.14 se muestran cortes del plano $y = 0$ para los ejemplos 1 y 2, respectivamente, con las soluciones obtenidas con varios esquemas.

Podemos ver que en ambos ejemplos se han obtenido resultados similares. El esquema Roe clásico es el que más se ha aproximado a la solución de referencia. Los métodos IR-Roe y PVM Roe han dado resultados muy cercanos a los obtenidos con Roe clásico y sólo ligeramente mejores que el PVM-IFCP. Los esquemas PVM-2 y PVM-0 han proporcionado soluciones

Esquema	Norma L^1 $(\times 10^{-2})$ (h_1	$q_{1,x}$	$q_{1,y}$	h_2	$q_{2,x}$	$q_{2,y}$)
Roe clásico	(4.623	12.951	13.142	4.042	7.750	7.786)
IR-Roe y PVM Roe	(4.646	12.980	13.170	4.056	7.770	7.805)
PVM-IFCP	(4.704	12.968	13.152	4.101	7.817	7.846)
PVM-4	(6.004	14.698	14.877	5.111	9.236	9.270)
PVM-2U	(7.087	15.927	16.089	5.886	10.310	10.340)
PVM-2	(7.093	15.940	16.101	5.897	10.328	10.359)
PVM-1U	(11.115	20.836	20.913	8.705	14.289	14.306)
PVM-0	(11.333	21.171	21.238	8.956	14.709	14.727)

Tabla 3.4: Norma L^1 $(\times 10^{-2})$ de la diferencia entre la solución de referencia con la malla de 1024000 volúmenes y la solución obtenida con cada esquema numérico y la malla de 64000 volúmenes para el ejemplo 1.

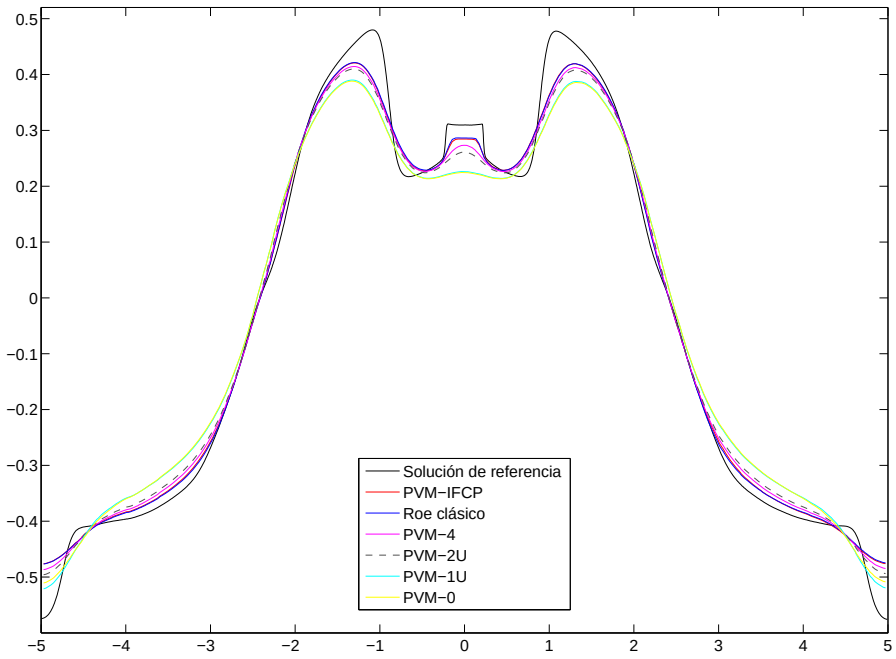
Esquema	Norma L^1 $(\times 10^{-2})$ (h_1	$q_{1,x}$	$q_{1,y}$	h_2	$q_{2,x}$	$q_{2,y}$)
Roe clásico	(3.831	3.367	2.577	3.798	3.171	2.635)
IR-Roe y PVM Roe	(3.850	3.372	2.576	3.817	3.176	2.636)
PVM-IFCP	(3.968	3.394	2.576	3.941	3.205	2.639)
PVM-4	(12.249	6.226	2.691	12.131	5.920	2.800)
PVM-2U	(14.446	7.086	2.732	14.303	6.771	2.856)
PVM-2	(14.447	7.086	2.733	14.304	6.772	2.856)
PVM-1U	(20.384	9.747	2.863	20.154	9.440	3.037)
PVM-0	(20.453	9.785	2.882	20.222	9.482	3.056)

Tabla 3.5: Norma L^1 $(\times 10^{-2})$ de la diferencia entre la solución de referencia con la malla de 1024000 volúmenes y la solución obtenida con cada esquema numérico y la malla de 64000 volúmenes para el ejemplo 2.

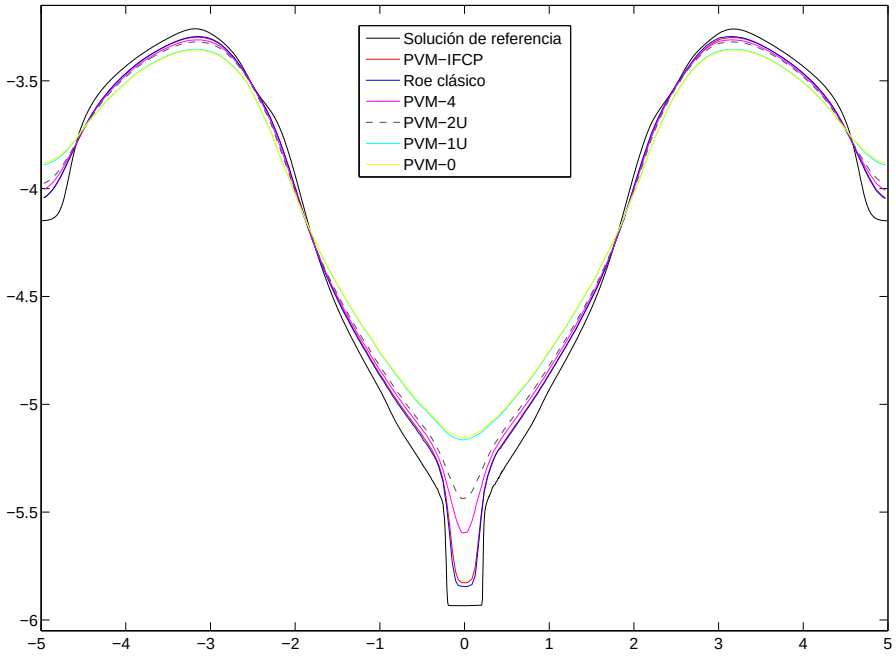
un poco peores que las obtenidas con PVM-2U y PVM-1U, respectivamente, para ambos ejemplos. De todos los esquemas analizados, el PVM-0 es el que ha dado peores resultados. Como ejemplo, en la Figura 3.13b se puede ver que los métodos PVM-0 y PVM-1U no han conseguido representar el hundimiento del fluido que se forma en la capa 2 alrededor del punto central, algo que sí han conseguido en mayor o menor medida el resto de esquemas.

3.6.4. Análisis de Eficiencia

En esta sección estudiaremos la eficiencia computacional de las distintas implementaciones CUDA presentadas en la sección 3.5.2. Para ello, ejecutaremos dichos programas CUDA para cada uno de los dos ejemplos descritos en la sección 3.6.1. Asimismo, para cada esquema numérico se han realizado dos implementaciones en CPU, una para una sola hebra y otra para cuatro hebras usando OpenMP [CJvdP07] con el objetivo de aprovechar los procesadores multinúcleo. Ambas están implementadas en C++, utilizan doble precisión numérica y hacen uso de la librería matemática Eigen [Eig].

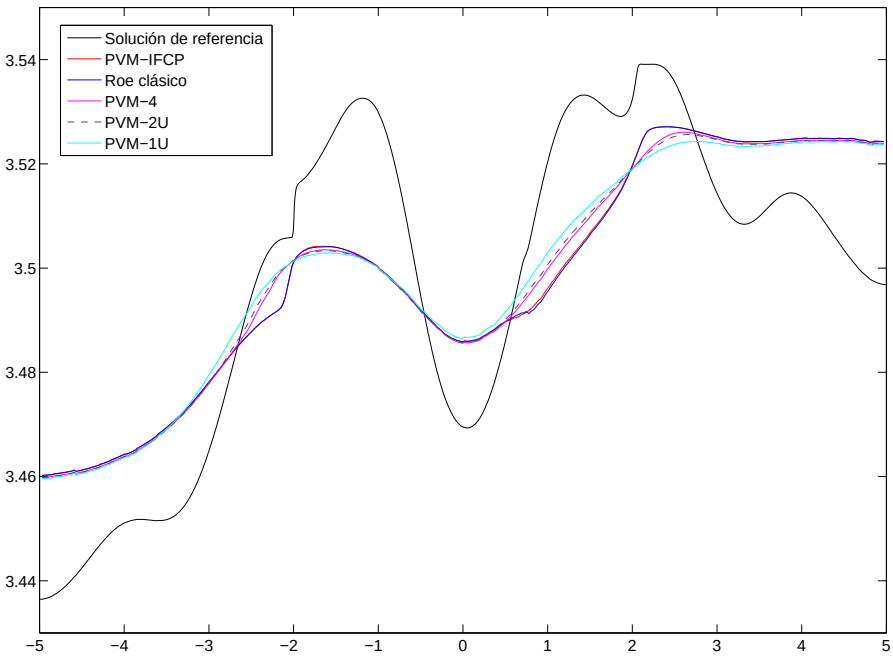


(a)

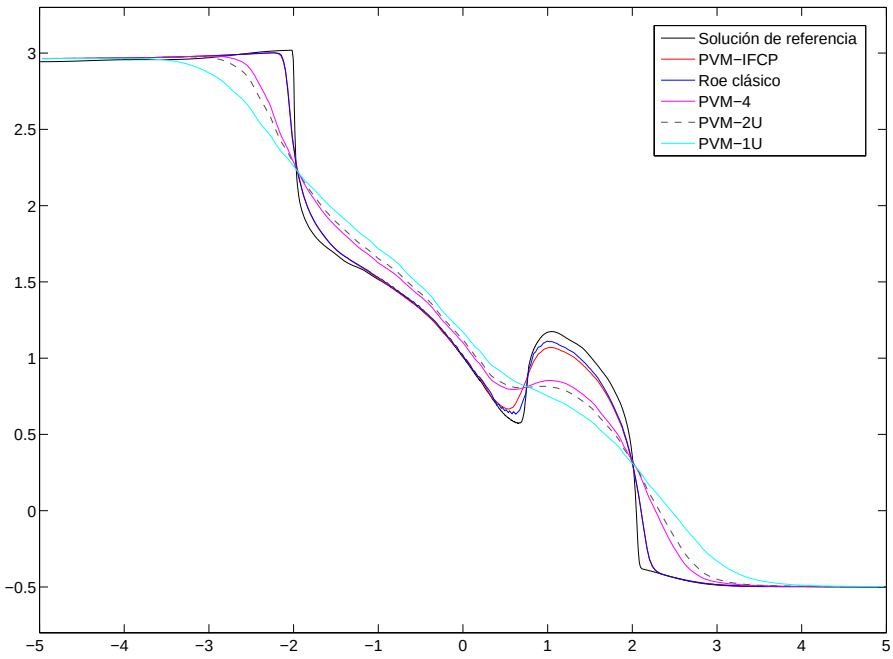


(b)

Figura 3.13: Corte del plano $y = 0$ con las soluciones obtenidas con varios esquemas numéricos para el ejemplo 1 en el tiempo 1 seg: (a) Capa 1; (b) Capa 2.



(a)



(b)

Figura 3.14: Corte del plano $y = 0$ con las soluciones obtenidas con varios esquemas numéricos para el ejemplo 2 en el tiempo 4 seg: (a) Capa 1; (b) Capa 2.

Como se comentó en la sección 3.5.2, para evitar problemas de inestabilidad numérica, en la implementación CUDA en simple precisión del esquema Roe clásico el cálculo de los autovectores y autovalores de la matriz A_{ij} se realiza en doble precisión. Para ello se ha hecho uso de la función `rg` de la librería EISPACK [Eis], traducida a C mediante la utilidad `f2c`.

Todos los programas se ejecutarán en un Core i7 920 con 4 GB de memoria RAM, y se usará una tarjeta GeForce GTX 580. El tiempo de simulación es de 0.1 segundos para ambos ejemplos. Se ha elegido el valor umbral 4096 para decidir entre llevar a cabo la obtención del mínimo Δt en CPU o en GPU. Se han asignado tamaños de 48 KB y 16 KB a la caché L1 y a la memoria compartida de GPU, respectivamente, para el kernel de procesamiento de aristas. El tamaño de un bloque de hebras para los kernels de procesamiento de aristas y de obtención del siguiente estado W_i^{n+1} (y del Δt local) es de 64 hebras en ambos kernels.

Las Tablas 3.6 a 3.14 muestran los tiempos obtenidos en segundos para todas las implementaciones, esquemas numéricos y tamaños de malla en el ejemplo 1. Las Tablas 3.15 a 3.23 muestran la misma información para el ejemplo 2. En dichas tablas, SP significa simple precisión, DP doble precisión, y SP+DP significa que se ha usado doble precisión únicamente en una parte del cómputo. Las Figuras 3.15 y 3.16 muestran gráficamente todos los tiempos de ejecución obtenidos con la malla de 2080560 volúmenes en los ejemplos 1 y 2, respectivamente. Las Figuras 3.17 y 3.18 representan gráficamente la ganancia obtenida con todos los programas CUDA de simple precisión respecto a las dos versiones CPU para los ejemplos 1 y 2, respectivamente. Las Tablas 3.24 y 3.25 indican el número de iteraciones que se han ejecutado con todos los esquemas numéricos y tamaños de malla para los ejemplos 1 y 2, respectivamente.

Como se puede comprobar, los resultados obtenidos en ambos ejemplos son bastante similares. El método de Roe clásico ha sido el que peores tiempos de GPU ha proporcionado, seguido del IR-Roe, el PVM-Roe y, finalmente, todos los demás esquemas PVM (PVM-0, PVM-1U, PVM-2, PVM-2U, PVM-4 y PVM-IFCP). Estos últimos son los que han obtenido los mejores tiempos de ejecución en GPU debido a su menor complejidad computacional. Nótese que los tiempos de GPU conseguidos por estos últimos esquemas han sido todos ellos mejores que los alcanzados por los métodos de Roe también en el ejemplo 1, a pesar de haber realizado aproximadamente un 7 % más de iteraciones en dicho ejemplo (ver Tabla 3.24).

Podemos ver asimismo que, para todos los esquemas numéricos excepto el Roe clásico, las implementaciones CUDA de doble precisión han sido entre 2 y 4 veces más lentas que las versiones CUDA equivalentes en simple precisión en ambos ejemplos. Esta diferencia es algo menor en el caso del método Roe clásico debido a que la etapa más costosa, que es el cálculo de los autovalores y autovectores de la matriz A_{ij}, se realiza en doble precisión.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP+DP	CUDA DP
4000	1.12	0.30	0.072	0.080
16000	8.77	2.34	0.38	0.45
64000	70.22	18.71	2.36	3.01
256000	571.4	152.7	15.12	21.54
1024000	4982.6	1333.4	100.0	160.8
2080560	19206.1	5366.8	384.6	682.9

Tabla 3.6: Tiempos de ejecución en segundos con el esquema Roe clásico para el ejemplo 1.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.50	0.13	0.022	0.044
16000	4.00	1.08	0.081	0.19
64000	32.22	8.77	0.52	1.18
256000	249.7	68.33	3.27	8.19
1024000	1982.5	539.6	21.39	56.90
2080560	7781.5	2115.4	78.80	218.1

Tabla 3.7: Tiempos de ejecución en segundos con el esquema IR-Roe para el ejemplo 1.

Como se comentó en la sección 2.3.2, en arquitecturas Fermi, el rendimiento de las operaciones de coma flotante en doble precisión es aproximadamente el 50 % de las de simple precisión.

Respecto a la ganancia obtenida, la implementación del método de Roe clásico es la que menos ganancia ha conseguido (aproximadamente 50 y 14 respecto a las versiones CPU de 1 y 4 hebras, respectivamente), seguida de las implementaciones del IR-Roe y el PVM-Roe, el PVM-IFCP y, finalmente, el resto de esquemas PVM (PVM-0, PVM-1U, PVM-2, PVM-2U y PVM-4), obteniendo estos últimos ganancias de aproximadamente 270-300 y 75-80 respecto a las versiones CPU secuencial y multihebra, respectivamente. Como era de esperar, las implementaciones CPU de 4 hebras han obtenido ganancias ligeramente inferiores a 4 respecto a las versiones secuenciales en todos los casos.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.49	0.13	0.022	0.042
16000	3.97	1.07	0.079	0.18
64000	31.69	8.66	0.51	1.14
256000	246.7	67.48	3.21	7.97
1024000	1956.8	530.7	21.02	55.62
2080560	7673.9	2095.6	77.40	213.3

Tabla 3.8: Tiempos de ejecución en segundos con el esquema PVM-Roe para el ejemplo 1.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.47	0.13	0.0048	0.013
16000	3.72	0.99	0.022	0.074
64000	29.73	8.16	0.13	0.49
256000	237.1	66.57	0.93	3.57
1024000	1847.3	513.6	6.93	27.29
2080560	7119.2	1980.2	25.94	102.8

Tabla 3.9: Tiempos de ejecución en segundos con el esquema PVM-0 para el ejemplo 1.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.48	0.13	0.0048	0.013
16000	3.74	0.99	0.022	0.075
64000	29.82	8.21	0.14	0.50
256000	238.0	66.57	0.95	3.59
1024000	1854.7	515.4	7.10	27.48
2080560	7131.2	1988.5	26.59	103.5

Tabla 3.10: Tiempos de ejecución en segundos con el esquema PVM-1U para el ejemplo 1.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.48	0.13	0.0048	0.013
16000	3.71	0.99	0.022	0.075
64000	29.81	8.22	0.14	0.49
256000	238.9	67.20	0.95	3.59
1024000	1868.1	519.7	7.08	27.44
2080560	7204.6	2008.0	26.51	103.4

Tabla 3.11: Tiempos de ejecución en segundos con el esquema PVM-2 para el ejemplo 1.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.49	0.13	0.0048	0.013
16000	3.78	1.01	0.022	0.075
64000	30.29	8.33	0.14	0.50
256000	241.7	67.83	0.96	3.62
1024000	1891.3	524.8	7.14	27.65
2080560	7280.3	2026.7	26.73	104.1

Tabla 3.12: Tiempos de ejecución en segundos con el esquema PVM-2U para el ejemplo 1.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.48	0.13	0.0048	0.013
16000	3.76	0.99	0.023	0.075
64000	30.17	8.31	0.14	0.50
256000	242.0	67.75	0.96	3.61
1024000	1894.6	525.4	7.18	27.60
2080560	7309.8	2032.4	26.90	103.9

Tabla 3.13: Tiempos de ejecución en segundos con el esquema PVM-4 para el ejemplo 1.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.48	0.13	0.0050	0.014
16000	3.76	1.01	0.024	0.077
64000	30.19	8.31	0.15	0.51
256000	240.4	67.64	1.00	3.70
1024000	1884.5	526.2	7.47	28.29
2080560	7261.3	2020.5	27.98	106.5

Tabla 3.14: Tiempos de ejecución en segundos con el esquema PVM-IFCP para el ejemplo 1.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP+DP	CUDA DP
4000	1.06	0.28	0.061	0.069
16000	8.27	2.13	0.34	0.44
64000	66.35	17.08	2.19	2.97
256000	530.2	135.6	14.80	20.72
1024000	4269.9	1109.2	103.5	151.8
2080560	16064.0	4229.0	395.7	606.2

Tabla 3.15: Tiempos de ejecución en segundos con el esquema Roe clásico para el ejemplo 2.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.50	0.14	0.021	0.032
16000	3.91	1.04	0.090	0.15
64000	31.38	8.19	0.53	0.90
256000	249.2	64.71	3.34	6.08
1024000	1988.3	516.4	21.53	43.96
2080560	7440.6	1929.5	75.41	161.9

Tabla 3.16: Tiempos de ejecución en segundos con el esquema IR-Roe para el ejemplo 2.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.50	0.13	0.020	0.030
16000	3.88	1.02	0.088	0.14
64000	30.97	8.07	0.52	0.88
256000	246.3	63.79	3.28	5.92
1024000	1965.0	510.5	21.16	42.94
2080560	7332.0	1912.2	74.13	158.3

Tabla 3.17: Tiempos de ejecución en segundos con el esquema PVM-Roe para el ejemplo 2.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.45	0.12	0.0042	0.011
16000	3.55	0.94	0.019	0.064
64000	28.26	7.39	0.11	0.43
256000	225.5	58.48	0.80	3.16
1024000	1805.4	469.3	5.91	24.42
2080560	6743.2	1756.6	21.66	90.66

Tabla 3.18: Tiempos de ejecución en segundos con el esquema PVM-0 para el ejemplo 2.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.46	0.12	0.0043	0.011
16000	3.56	0.94	0.019	0.064
64000	28.48	7.43	0.12	0.43
256000	226.6	58.56	0.82	3.18
1024000	1816.0	468.3	6.05	24.56
2080560	6762.1	1790.5	22.19	91.15

Tabla 3.19: Tiempos de ejecución en segundos con el esquema PVM-1U para el ejemplo 2.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.46	0.12	0.0043	0.011
16000	3.56	0.94	0.019	0.064
64000	28.52	7.45	0.12	0.43
256000	227.2	58.91	0.81	3.18
1024000	1822.4	475.7	6.01	24.55
2080560	6796.0	1772.4	22.08	91.11

Tabla 3.20: Tiempos de ejecución en segundos con el esquema PVM-2 para el ejemplo 2.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.46	0.12	0.0043	0.011
16000	3.57	0.94	0.019	0.065
64000	28.71	7.47	0.12	0.44
256000	228.0	59.90	0.82	3.20
1024000	1828.0	478.0	6.06	24.70
2080560	6819.8	1771.7	22.28	91.68

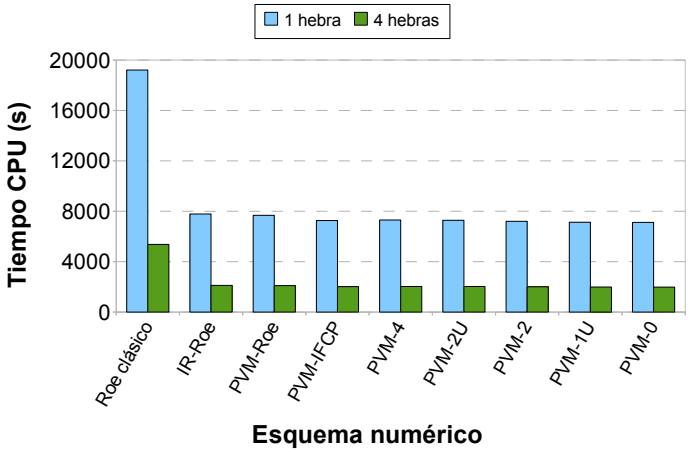
Tabla 3.21: Tiempos de ejecución en segundos con el esquema PVM-2U para el ejemplo 2.

Volúmenes	CPU	CPU	GTX 580	
	1 hebra	4 hebras	CUDA SP	CUDA DP
4000	0.46	0.12	0.0043	0.011
16000	3.56	0.93	0.019	0.064
64000	28.64	7.44	0.12	0.44
256000	228.5	58.98	0.82	3.20
1024000	1821.3	473.6	6.10	24.68
2080560	6877.1	1776.7	22.41	91.60

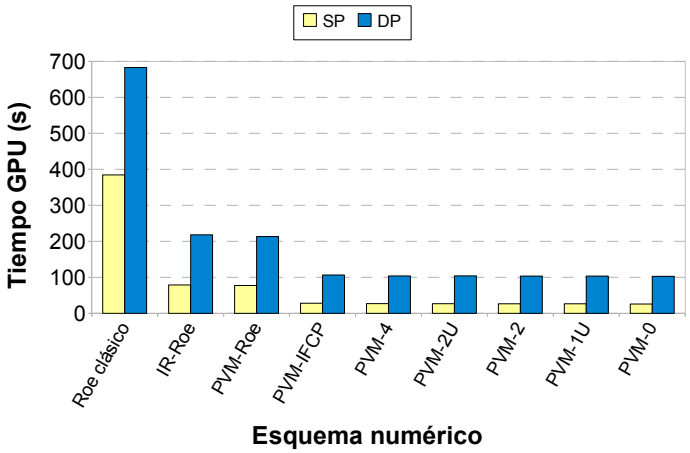
Tabla 3.22: Tiempos de ejecución en segundos con el esquema PVM-4 para el ejemplo 2.

Volúmenes	CPU 1 hebra	CPU 4 hebras	GTX 580 CUDA SP	GTX 580 CUDA DP
4000	0.46	0.12	0.0044	0.012
16000	3.59	0.94	0.020	0.067
64000	28.76	7.50	0.12	0.45
256000	227.6	59.22	0.86	3.31
1024000	1826.7	475.1	6.37	25.56
2080560	6823.1	1773.3	23.39	94.85

Tabla 3.23: Tiempos de ejecución en segundos con el esquema PVM-IFCP para el ejemplo 2.

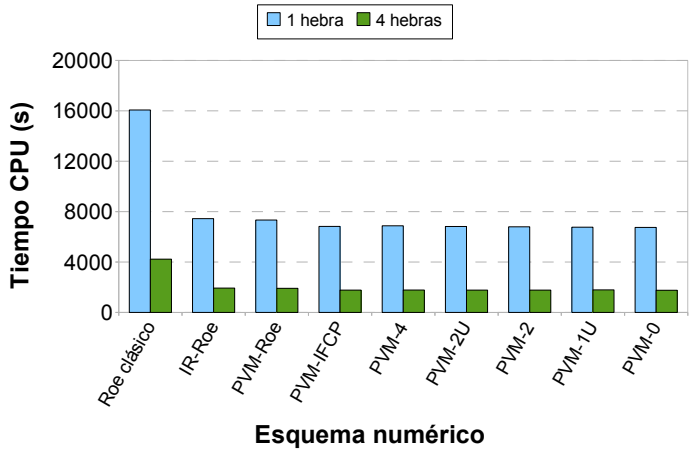


(a)

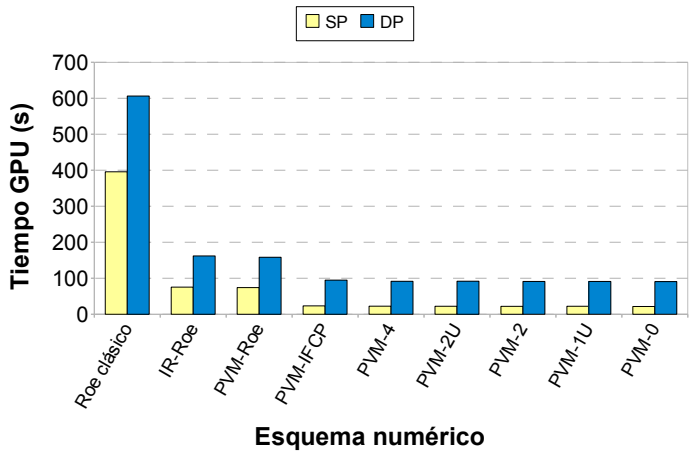


(b)

Figura 3.15: Tiempos de ejecución con todos los esquemas numéricos y la malla de 2080560 volúmenes para el ejemplo 1: (a) CPU; (b) GPU.



(a)



(b)

Figura 3.16: Tiempos de ejecución con todos los esquemas numéricos y la malla de 2080560 volúmenes para el ejemplo 2: (a) CPU; (b) GPU.

La Figura 3.19 representa gráficamente el número de GFLOPS (miles de millones de operaciones de coma flotante por segundo) obtenido con todas las implementaciones CUDA de simple precisión para ambos ejemplos. Vemos que, en los dos ejemplos, los esquemas numéricos que han obtenido un mejor aprovechamiento de la GPU en términos de GFLOPS han sido el IR-Roe y el PVM-Roe, ambos con aproximadamente 60-70 GFLOPS. A continuación todos los demás esquemas PVM con unos 35-40 GFLOPS, y finalmente el esquema Roe Clásico con 30 GFLOPS. A pesar de que este último esquema es el que tiene mayor intensidad computacional, el hecho de que el cálculo de los autovectores y autovalores de la matriz A_{ij} se realice en doble precisión ha influido negativamente en el número de GFLOPS obte-

Volúmenes	Roe clásico, IR-Roe y PVM-Roe	PVM-0, PVM-1U, PVM-2, PVM-2U, PVM-4 y PVM-IFCP
4000	22	23
16000	42	45
64000	84	90
256000	168	179
1024000	334	358
2080560	626	668

Tabla 3.24: Iteraciones realizadas con todos los esquemas numéricos y tamaños de malla para el ejemplo 1.

Volúmenes	Roe clásico, IR-Roe y PVM-Roe	PVM-0, PVM-1U, PVM-2, PVM-2U, PVM-4 y PVM-IFCP
4000	20	20
16000	39	39
64000	78	78
256000	155	155
1024000	310	310
2080560	569	570

Tabla 3.25: Iteraciones realizadas con todos los esquemas numéricos y tamaños de malla para el ejemplo 2.

nidos. El máximo rendimiento teórico para la GTX 580 en simple precisión es de 1581 GFLOPS.

Las Tablas 3.26 y 3.27 muestran el porcentaje del tiempo de ejecución empleado en cada kernel para la malla más grande con las implementaciones CUDA de simple precisión de los esquemas Roe clásico, IR-Roe y PVM-IFCP para ambos ejemplos. En el esquema Roe clásico, más del 98 % del tiempo de ejecución se ha empleado en el kernel que procesa las aristas. Este porcentaje ha disminuido al 92 y 75 % aproximadamente en los esquemas IR-Roe y PVM-IFCP, respectivamente. El hecho de que el porcentaje de tiempo empleado en el kernel que obtiene el siguiente estado de un volumen sea algo mayor de lo que cabría esperar se debe a que, en las implementaciones realizadas, se lleva a cabo un filtrado adicional del estado obtenido (ver el Apéndice A para más detalles).

También se han comparado las soluciones obtenidas en CPU y en GPU. Para ello, se ha calculado la norma L^1 de la diferencia entre las soluciones obtenidas (los vectores de estado obtenidos en el tiempo t_{fin}) con la versión CPU de 1 hebra y los programas CUDA para todos los tamaños de malla. Los órdenes de magnitud de dicha norma L^1 obtenidos con las implementaciones CUDA de simple precisión oscilan entre 10^{-4} y 10^{-6}, mientras que los

Kernel	% Tiempo de ejecución		
	Roe clásico	IR-Roe	PVM-IFCP
Procesar aristas	98.39	92.12	76.79
Obtener W_i^{n+1} y Δt_i	1.60	7.82	23.02
Obtener mínimo Δt	0.01	0.06	0.19

Tabla 3.26: Porcentaje del tiempo de ejecución empleado en cada kernel para la malla de 2080560 volúmenes con las implementaciones CUDA de simple precisión de los esquemas Roe clásico, IR-Roe y PVM-IFCP para el ejemplo 1 usando una GeForce GTX 580.

Kernel	% Tiempo de ejecución		
	Roe clásico	IR-Roe	PVM-IFCP
Procesar aristas	98.46	91.88	74.51
Obtener W_i^{n+1} y Δt_i	1.53	8.06	25.30
Obtener mínimo Δt	0.01	0.06	0.19

Tabla 3.27: Porcentaje del tiempo de ejecución empleado en cada kernel para la malla de 2080560 volúmenes con las implementaciones CUDA de simple precisión de los esquemas Roe clásico, IR-Roe y PVM-IFCP para el ejemplo 2 usando una GeForce GTX 580.

obtenidos con las implementaciones CUDA de doble precisión oscilan entre 10^{-13} y 10^{-15}, lo que pone de manifiesto la distinta precisión alcanzada en GPU usando simple y doble precisión numérica.

Como último comentario, a partir de los resultados obtenidos en los experimentos realizados en las secciones 3.6.3 y 3.6.4 podemos concluir que el esquema PVM-IFCP es el que ha proporcionado un mejor equilibrio entre precisión y eficiencia computacional en GPU. Con dicho esquema, hemos obtenido soluciones casi igual de precisas que los métodos de Roe, alcanzando una eficiencia computacional comparable a la del resto de esquemas PVM.

3.7. Conclusiones

En este capítulo se han adaptado e implementado en GPU varios esquemas numéricos de orden uno para el sistema de aguas someras bicapa en mallas triangulares. Las implementaciones GPU de los métodos de Roe clásico, IR-Roe y PVM Roe no han proporcionado prestaciones tan buenas como ocurre con los demás esquemas PVM (PVM-0, PVM-1U, PVM-2, PVM-2U, PVM-4 y PVM-IFCP) en términos de tiempo de ejecución y ganancia obtenida con respecto a implementaciones equivalentes en CPU. Esto se ha debido principalmente a que los métodos Roe clásico, IR-Roe y PVM-Roe tienen una mayor intensidad computacional, requiriendo incluso el primero

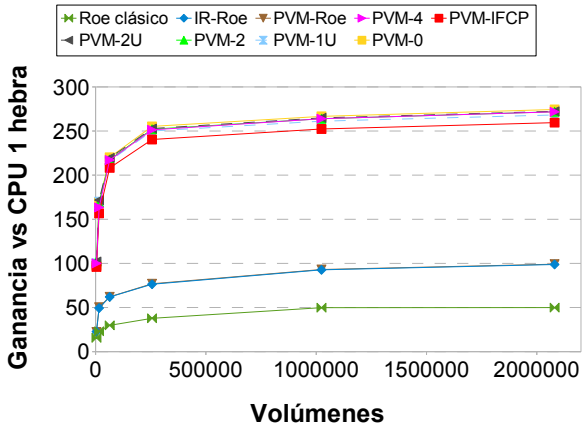
de ellos el uso de doble precisión numérica en el proceso de descomposición espectral de la matriz A_{ij} para mantener su estabilidad. Por el contrario, los esquemas PVM-0, PVM-1U, PVM-2, PVM-2U, PVM-4 y PVM-IFCP presentan una demanda computacional menor al no requerir la descomposición espectral de A_{ij} y sus implementaciones son todas ellas estables en simple precisión, lo que ha causado que obtengan mejores tiempos de ejecución en GPU, acelerando las simulaciones numéricas en dos órdenes de magnitud respecto a implementaciones CPU secuenciales, llegando a obtener ganancias de 300 y 80 en una GeForce GTX 580 respecto a versiones CPU de 1 y 4 hebras, respectivamente, ejecutadas en un Core i7 920. De todos los esquemas analizados, el PVM-IFCP es el que ha proporcionado un mejor equilibrio entre precisión de los resultados obtenidos y eficiencia computacional en GPU.

Las implementaciones CUDA en simple precisión de los métodos de IR-Roe y PVM-Roe son las que mejor han aprovechado la tarjeta gráfica en términos de rendimiento absoluto, obteniendo ambas entre 60 y 70 GFLOPS.

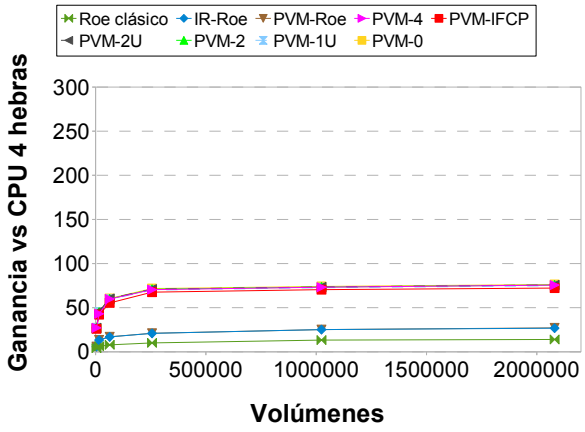
Las implementaciones CUDA de doble precisión han resultado ser entre 2 y 4 veces más lentas que las versiones CUDA equivalentes en simple precisión, si bien la precisión numérica alcanzada ha sido lógicamente mayor con doble precisión.

Asimismo, hemos medido el porcentaje de tiempo de ejecución empleado por cada etapa del proceso en las implementaciones GPU en simple precisión de los métodos de Roe clásico, IR-Roe y PVM-IFCP usando una malla de aproximadamente dos millones de volúmenes. En todos los casos, la etapa más costosa ha resultado ser el procesamiento de aristas, consumiendo un porcentaje del tiempo de ejecución que varía entre el 75 % en el caso del esquema PVM-IFCP y el 98 % para el método de Roe clásico.

Finalmente, la ordenación de los volúmenes en los arrays que almacenan los datos de los volúmenes en GPU ha demostrado ser un factor que influye en los tiempos de ejecución que se obtienen. Concretamente, aplicando la ordenación de Cuthill-McKee inversa a dichos arrays con el objetivo de aumentar la localidad espacial hemos conseguido mejorar los tiempos de GPU entre un 5 y un 55 % aproximadamente (dependiendo del esquema numérico) respecto a los obtenidos con la ordenación proporcionada al crear las mallas con el toolkit Partial Differential Equation de MATLAB.

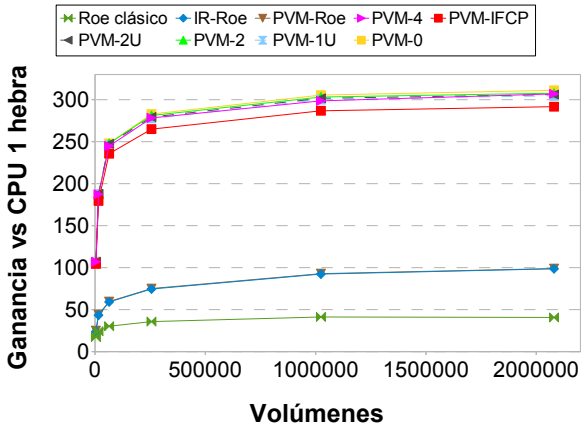


(a)

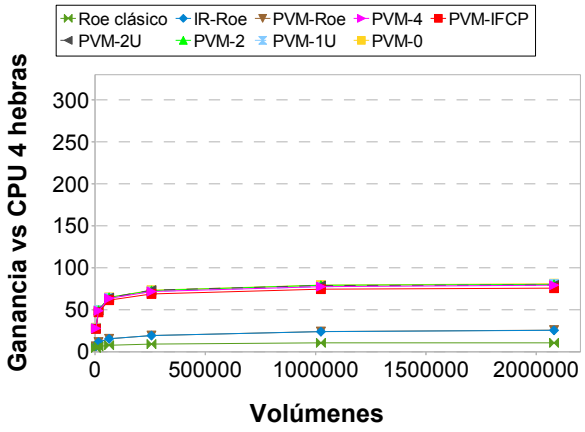


(b)

Figura 3.17: Ganancia obtenida con todas las implementaciones CUDA de simple precisión en una GeForce GTX 580 para el ejemplo 1: (a) Respecto a la versión CPU de 1 hebra; (b) Respecto a la versión CPU de 4 hebras.

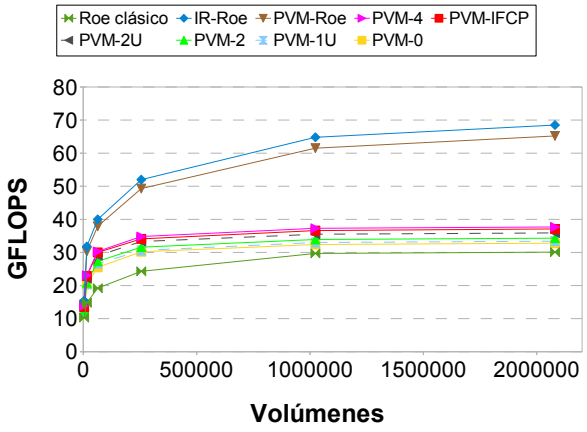


(a)

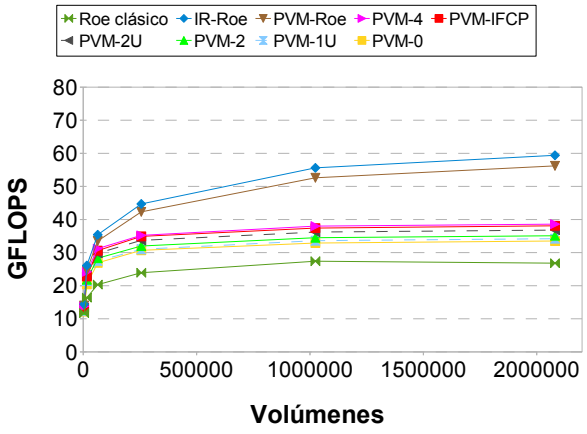


(b)

Figura 3.18: Ganancia obtenida con todas las implementaciones CUDA de simple precisión en una GeForce GTX 580 para el ejemplo 2: (a) Respecto a la versión CPU de 1 hebra; (b) Respecto a la versión CPU de 4 hebras.



(a)



(b)

Figura 3.19: GFLOPS obtenidos con todas las implementaciones CUDA de simple precisión en una GeForce GTX 580: (a) Ejemplo 1; (b) Ejemplo 2.

Capítulo 4

Esquemas de Alto Orden para el Sistema de Aguas Someras Bicapa

RESUMEN: En este capítulo se presenta un algoritmo de alto orden para la resolución numérica del sistema de aguas someras bicapa en mallas triangulares, se describe su adaptación e implementación en GPU y se efectúan varios experimentos usando las implementaciones realizadas del algoritmo en orden 2 y 3. Finalmente, se analizan los resultados obtenidos en términos de precisión y de eficiencia computacional.

4.1. Introducción

Un modo de aumentar la precisión de los resultados obtenidos con un esquema de orden 1 es aplicando un operador de reconstrucción de estados. Los métodos de alto orden basados en la reconstrucción de estados pueden construirse del siguiente modo: dado un esquema de primer orden con función de flujo numérico $\mathcal{F}_{ij}^{-}(W_i, W_j, H_i, H_j)$, se considera un operador de reconstrucción de orden p, esto es, un operador que asocia a un conjunto de valores $\{W_i\}_{i=1}^{L}$ en los volúmenes dos familias de funciones definidas en las aristas:

$$\gamma \in \Gamma_{ij} \to W_{ij}^{\pm}(\gamma)$$

tales que, siempre que

$$W_i = \frac{1}{|V_i|} \int_{V_i} W(\boldsymbol{x})\, d\boldsymbol{x} \tag{4.1}$$

para alguna función suave W, entonces:

$$W_{ij}^{\pm}(\gamma) = W(\gamma) + O(\Delta x^p), \quad \forall \gamma \in \Gamma_{ij}.$$

Una vez elegidos el esquema de orden 1 y el operador de reconstrucción, se puede utilizar el método de líneas para desarrollar esquemas de alto orden para el sistema (3.2). La idea es discretizar sólo en espacio, dejando el problema continuo en tiempo. Este procedimiento conduce a un sistema de ecuaciones diferenciales ordinarias que debe resolverse numéricamente. La elección del método numérico es importante: una discretización en tiempo linealmente estable que no sea TVD (Total Variation Diminishing) puede generar oscilaciones espurias incluso para discretizaciones espaciales TVD. Consideraremos aquí los métodos TVD Runge-Kutta introducidos en [Shu88]. Finalmente, se utilizan fórmulas de cuadratura para aproximar todas las integrales que aparecen.

En este capítulo adaptaremos e implementaremos en GPU el operador de reconstrucción introducido en [DK07]. Aplicaremos esta reconstrucción junto con el esquema PVM-IFCP descrito en el capítulo anterior, por ser el esquema que mejor relación precisión-eficiencia ha proporcionado. A continuación se describe con detalle el esquema numérico de alto orden.

4.2. Esquema Numérico

En [CFNF$^+$09] se presentó la siguiente extensión semi-discreta de alto orden de un esquema de primer orden como los descritos en el capítulo 3 para el sistema (3.2):

$$\begin{aligned} W_i(t) \;=\; & \frac{-1}{|V_i|} \sum_{j \in \aleph_i} \int_{\Gamma_{ij}} \mathcal{F}_{ij}^{-}\big(W_{ij}^{-}(\gamma,t),\, W_{ij}^{+}(\gamma,t),\, H_{ij}^{-}(\gamma),\, H_{ij}^{+}(\gamma)\big)\, d\gamma \\ & + \frac{1}{|V_i|} \int_{V_i} \left(B_1\big(P_{W_i}^t(\boldsymbol{x})\big) \frac{\partial}{\partial_x} P_{W_i}^t(\boldsymbol{x}) + B_2\big(P_{W_i}^t(\boldsymbol{x})\big) \frac{\partial}{\partial_y} P_{W_i}^t(\boldsymbol{x}) \right) d\boldsymbol{x} \\ & + \frac{1}{|V_i|} \int_{V_i} \left(S_1\big(P_{W_i}^t(\boldsymbol{x})\big) \frac{\partial}{\partial_x} P_{H_i}(\boldsymbol{x}) + S_2\big(P_{W_i}^t(\boldsymbol{x})\big) \frac{\partial}{\partial_y} P_{H_i}(\boldsymbol{x}) \right) d\boldsymbol{x} \quad (4.2) \end{aligned}$$

donde B_1, B_2, S_1 y S_2 se definieron en (3.3). $P_{W_i}^t(\boldsymbol{x})$ es una función (generalmente un polinomio) que aproxima el valor de $W(\boldsymbol{x}, t)$ en el volumen V_i y que se calcula a partir de los promedios de $W(t)$ en un entorno cercano a V_i (un *stencil*). Podemos pensar en un stencil como en un conjunto de volúmenes.

Análogamente, la función $P_{H_i}(\boldsymbol{x})$ aproxima el valor de $H(\boldsymbol{x})$ en el volumen V_i y se calcula a partir de los promedios de H en el mismo stencil.

Nótese que, dado que H no depende del tiempo, la función que lo aproxima tampoco.

Estas funciones de aproximación normalmente se definen mediante técnicas de interpolación o aproximación. Una vez estas funciones han sido construidas, las reconstrucciones $W_{ij}^{-}(\gamma,t)$ y $W_{ij}^{+}(\gamma,t)$, con $\gamma \in \Gamma_{ij}$, se definen como:

$$W_{ij}^{-}(\gamma,t) = \lim_{x\to\gamma} P_{W_i}^{t}(\boldsymbol{x}), \qquad W_{ij}^{+}(\gamma,t) = \lim_{x\to\gamma} P_{W_j}^{t}(\boldsymbol{x}).$$

Claramente, para todo $\gamma \in \Gamma_{ij}$ se satisface:

$$W_{ij}^{-}(\gamma,t) = W_{ji}^{+}(\gamma,t), \qquad W_{ij}^{+}(\gamma,t) = W_{ji}^{-}(\gamma,t).$$

Las reconstrucciones $H_{ij}^{\pm}(\gamma)$ se definen del mismo modo.

Además, el operador de reconstrución $P_{W_i}^{t}(\boldsymbol{x})$ (y análogamente, $P_{H_i}(\boldsymbol{x})$) satisface las siguientes propiedades:

1. Es conservativo, es decir, la siguiente igualdad se cumple para todo volumen V_i:

$$W_i(t) = \frac{1}{|V_i|}\int_{V_i} P_{W_i}^{t}(\boldsymbol{x})\, d\boldsymbol{x}.$$

2. Si el operador se aplica a una secuencia $\{W_i\}$ que satisface (4.1) para alguna función suave W, entonces:

$$W_{ij}^{\pm}(\gamma,t) = W(\gamma,t) + O(\Delta x^{p}), \quad \forall\gamma \in \Gamma_{ij}\,,$$

$$W_{ij}^{+}(\gamma,t) - W_{ij}^{-}(\gamma,t) = O(\Delta x^{p+1}), \quad \forall\gamma \in \Gamma_{ij}\,.$$

3. Es de orden q en el interior de los volúmenes, es decir, si el operador se aplica a una secuencia $\{W_i\}$ que satisface (4.1) para alguna función suave W, entonces:

$$P_{W_i}^{t}(\boldsymbol{x}) = W(\boldsymbol{x},t) + O(\Delta x^{q}), \quad \forall\boldsymbol{x} \in interior(V_i).$$

4. El gradiente de $P_{W_i}^{t}$ proporciona una aproximación de orden m del gradiente de W:

$$\nabla P_{W_i}^{t}(\boldsymbol{x}) = \nabla W(\boldsymbol{x},t) + O(\Delta x^{m}), \quad \forall\boldsymbol{x} \in interior(V_i).$$

En general, $m \leq q \leq p$. Si, por ejemplo, las funciones de aproximación son polinomios de grado p obtenidos interpolando los valores en los volúmenes de un stencil, entonces $m = p - 1$ y $q = p$.

Obsérvese que en esquemas de orden 1, dado que los valores de W y H se suponen constantes en todo el volumen, las derivadas que aparecen en las integrales del volumen de (4.2) son cero y, por tanto, en orden 1 estas integrales son cero.

La discretización temporal se realiza mediante el uso de esquemas numéricos de tipo TVD Runge-Kutta [Shu88]. Concretamente, dado el valor de W^n en el tiempo t^n, en orden 2 se aplica el siguiente esquema:

$$\begin{aligned} W_i^{n+1/2} &= W_i^n + \frac{\Delta t^n}{|\,V_i\,|}\, L\big(\{W_j^n,\, H_j;\, j \in \mathcal{B}_i\}\big) \\ W_i^{n+1} &= \frac{1}{2}\left(W_i^n + W_i^{n+1/2} + \frac{\Delta t^n}{|\,V_i\,|}\, L\Big(\big\{W_j^{n+1/2},\, H_j;\, j \in \mathcal{B}_i\big\}\Big)\right) \end{aligned} \tag{4.3}$$

mientras que en orden 3 se utiliza el siguiente esquema:

$$\begin{aligned} W_i^{n+1/3} &= W_i^n + \frac{\Delta t^n}{|\,V_i\,|}\, L\big(\{W_j^n,\, H_j;\, j \in \mathcal{B}_i\}\big) \\ W_i^{n+2/3} &= \frac{1}{4}\left(3W_i^n + W_i^{n+1/3} + \frac{\Delta t^n}{|\,V_i\,|}\, L\Big(\big\{W_j^{n+1/3},\, H_j;\, j \in \mathcal{B}_i\big\}\Big)\right) \\ W_i^{n+1} &= \frac{1}{3}\left(W_i^n + 2W_i^{n+2/3} + \frac{2\,\Delta t^n}{|\,V_i\,|}\, L\Big(\big\{W_j^{n+2/3},\, H_j;\, j \in \mathcal{B}_i\big\}\Big)\right) \end{aligned} \tag{4.4}$$

donde $\mathcal{B}_i$ es el conjunto de índices j tales que V_j pertenece al stencil asociado al volumen V_i, y

$$\begin{aligned} L\big(\{W_j^n,\, H_j;\, j \in \mathcal{B}_i\}\big) \;=\; & -\sum_{j \in \aleph_i} \int_{\Gamma_{ij}} \mathcal{F}_{ij}^{-}\big(W_{ij}^{-}(\gamma,t),\, W_{ij}^{+}(\gamma,t),\, H_{ij}^{-}(\gamma),\, H_{ij}^{+}(\gamma)\big)\, d\gamma \\ & + \int_{V_i} \left(B_1\big(P_{W_i}^t(\boldsymbol{x})\big)\frac{\partial}{\partial_x} P_{W_i}^t(\boldsymbol{x}) + B_2\big(P_{W_i}^t(\boldsymbol{x})\big)\frac{\partial}{\partial_y} P_{W_i}^t(\boldsymbol{x})\right) d\boldsymbol{x} \\ & + \int_{V_i} \left(S_1\big(P_{W_i}^t(\boldsymbol{x})\big)\frac{\partial}{\partial_x} P_{H_i}(\boldsymbol{x}) + S_2\big(P_{W_i}^t(\boldsymbol{x})\big)\frac{\partial}{\partial_y} P_{H_i}(\boldsymbol{x})\right) d\boldsymbol{x} \end{aligned} \tag{4.5}$$

Δt^n es el paso de tiempo y está calculado usando la condición CFL usual (3.5).

Finalmente, es necesario usar fórmulas de cuadratura para aproximar las integrales que aparecen en (4.5). Las integrales del volumen se aproximarán mediante un punto de cuadratura en orden 2 (el baricentro del volumen), y 4 en orden 3. En la Tabla 9.2 de [ZT00] se muestran varias fórmulas de cuadratura para triángulos. En la sección 4.5.2 se explicará con más detalle

(a) (b)

Figura 4.1: Puntos de cuadratura de una arista. (a) Orden 2. El peso del punto es 1, (b) Orden 3, $a = 0.5 - \frac{1}{2\sqrt{3}}$, $b = 0.5 + \frac{1}{2\sqrt{3}}$. El peso de ambos puntos es 0.5.

el cálculo de estas integrales del volumen. La integral de la arista, por su parte, se aproximará mediante un punto de cuadratura en orden 2 (el punto medio de la arista), y 2 en orden 3. La Figura 4.1 muestra los puntos y pesos utilizados para aproximar la integral de la arista en ambos casos.

A continuación describimos el operador de reconstrucción que se ha implementado.

4.3. Operador de Reconstrucción

El principal ingrediente del esquema de alto orden descrito en la sección anterior es el operador de reconstrucción. En este trabajo utilizaremos el operador de reconstrucción propuesto por M. Dumbser y M. Käser en [DK07] para mallas triangulares.

Con el fin de simplificar la notación, eliminaremos la dependencia temporal del operador de reconstrucción. Además, lo definiremos sólo para una función escalar $u(\boldsymbol{x})$. Deberemos aplicar el proceso de reconstrucción a cada una de las variables del sistema (3.2) (W y H). Emplearemos la siguiente notación:

$$\bar{u}_i = \frac{1}{|V_i|} \int_{V_i} u(\boldsymbol{x})\, d\boldsymbol{x}.$$

donde $\bar{u}_i$ denota el promedio de $u(\boldsymbol{x})$ en el volumen V_i.

Disponemos de un nuevo sistema de coordenadas, al que de ahora en adelante llamaremos sistema de referencia. Este sistema de referencia se muestra en la Figura 4.2, donde la división del plano en seis regiones se explicará con posterioridad, en la sección 4.4. Sea V_i un triángulo del dominio original con vértices (x_1, y_1), (x_2, y_2) y (x_3, y_3), y sea $\hat{V}_0$ su triángulo asociado en el sistema de referencia, con vértices $(0, 0)$, $(1, 0)$ y $(0, 1)$ (ver Figura 4.2). La transformación que lleva un punto $(\hat{x}, \hat{y})$ del triángulo $\hat{V}_0$ al punto correspondiente (x, y) del triángulo V_i es la siguiente:

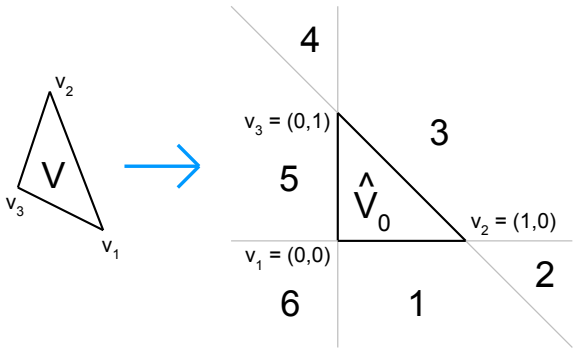


Figura 4.2: Stencils y transformación del sistema original al sistema de referencia.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix} \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

mientras que la transformación inversa viene dada por:

$$\begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} = \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix}^{-1} \begin{pmatrix} x - x_1 \\ y - y_1 \end{pmatrix}. \tag{4.6}$$

Todas las reconstrucciones se realizan en el sistema de referencia. Se considera que el valor reconstruido en un punto (x, y) del triángulo V_i del dominio original es una combinación lineal de varias funciones base (polinomios) evaluadas en el punto $(\hat{x}, \hat{y})$ correspondiente del triángulo $\hat{V}_0$ del sistema de referencia. Formalmente, definimos el operador de reconstrucción polinómico de $u(\boldsymbol{x})$ en el volumen V_i como:

$$P_i(x, y) = \sum_{k=0}^{l-1} \alpha_{i,k} \, \Psi_k(\hat{x}, \hat{y}) \tag{4.7}$$

donde $\Psi_k(\hat{x}, \hat{y})$ es una base de polinomios ortogonales. En este trabajo utilizaremos polinomios de grado 1 y 2. Esta base puede generarse mediante una fórmula recurrente, y para $l = 6$ tenemos:

- $\Psi_0(\hat{x}, \hat{y}) = 1$
- $\Psi_1(\hat{x}, \hat{y}) = 2\hat{x} - 1 + \hat{y}$
- $\Psi_2(\hat{x}, \hat{y}) = 3\hat{y} - 1$
- $\Psi_3(\hat{x}, \hat{y}) = 1 - 2\hat{y} + \hat{y}^2 - 6\hat{x} + 6\hat{x}\hat{y} + 6\hat{x}^2$

- $\Psi_4(\hat{x},\hat{y}) = 5\hat{y}^2 + 10\hat{x}\hat{y} - 6\hat{y} - 2\hat{x} + 1$
- $\Psi_5(\hat{x},\hat{y}) = 1 - 8\hat{y} + 10\hat{y}^2$

En orden 2 se consideran las tres primeras funciones, mientras que en orden 3 se consideran las seis, es decir, $l = 3$ en orden 2, y 6 en orden 3.

El cálculo de la reconstrucción $P_i(x,y)$ requiere el uso de un stencil $\mathcal{S}_i$ asociado al volumen V_i, es decir, un conjunto de triángulos en un entorno cercano a V_i $\left(\mathcal{S}_i = \bigcup_{j\in\mathcal{B}_i} V_j\right)$. El primer elemento de $\mathcal{S}_i$ siempre es el propio triángulo V_i. En la sección 4.4 detallaremos el proceso de obtención de dicho stencil. Llamaremos $\hat{\mathcal{S}}_i$ al stencil $\mathcal{S}_i$ llevado al sistema de referencia aplicando (4.6), es decir, $\hat{\mathcal{S}}_i = \bigcup_{j\in\mathcal{B}_i} \hat{V}_j$.

Para el cálculo de los coeficientes $\alpha_{i,k}$, $0 \le k < l$, que aparecen en (4.7), imponemos que el operador de reconstrucción debe ser conservativo, esto es:

$$\frac{1}{|V_j|}\int_{V_j} P_i(x,y)\,dx\,dy = \bar{u}_j\,, \quad \forall j \in \mathcal{B}_i. \tag{4.8}$$

Insertando (4.7) en (4.8), obtenemos:

$$\frac{1}{|\hat{V}_j|}\sum_{k=0}^{l-1} \alpha_{i,k}\int_{\hat{V}_j} \Psi_k(\hat{x},\hat{y})\,d\hat{x}\,d\hat{y} = \bar{u}_j\,, \quad \forall j \in \mathcal{B}_i.$$

o, de forma equivalente en modo matricial:

$$\begin{pmatrix} \frac{1}{|\hat{V}_0|}\int_{\hat{V}_0}\Psi_0 & \cdots & \frac{1}{|\hat{V}_0|}\int_{\hat{V}_0}\Psi_{l-1} \\ \vdots & \ddots & \vdots \\ \frac{1}{|\hat{V}_{s-1}|}\int_{\hat{V}_{s-1}}\Psi_0 & \cdots & \frac{1}{|\hat{V}_{s-1}|}\int_{\hat{V}_{s-1}}\Psi_{l-1} \end{pmatrix} \begin{pmatrix} \alpha_{i,0} \\ \alpha_{i,1} \\ \vdots \\ \alpha_{i,l-1} \end{pmatrix} = \begin{pmatrix} \bar{u}_0 \\ \bar{u}_1 \\ \vdots \\ \bar{u}_{s-1} \end{pmatrix} \tag{4.9}$$

donde s es el número de elementos del stencil $\hat{\mathcal{S}}_i$ y $\hat{V}_j$ es el triángulo j-ésimo del stencil en el sistema de referencia. Nótese que, para la resolución de (4.9), $\hat{\mathcal{S}}_i$ debe tener al menos l elementos. Sin embargo, en mallas triangulares, por cuestiones de estabilidad deben usarse más elementos que el mínimo necesario. Por este motivo, siguiendo las recomendaciones de [DK07], lo que haremos será usar el doble del mínimo requerido, es decir, $\hat{\mathcal{S}}_i$ tendrá 6 elementos en orden 2, y 12 elementos en orden 3.

Dado que $\Psi_0 = 1$, la primera columna de la matriz izquierda de (4.9) siempre es igual a uno. Por otra parte, debido a la propia definición de las funciones base, la primera fila de esa misma matriz es igual a cero en todas las posiciones menos la primera. Por ello, el valor del primer coeficiente queda determinado, siendo siempre igual al valor promedio en el triángulo

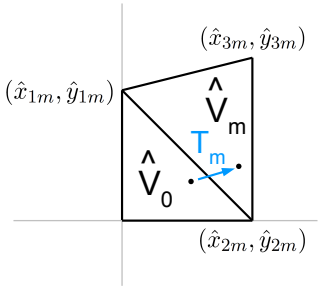


Figura 4.3: Transformación T_m que lleva un punto del triángulo $\hat{V}_0$ al triángulo $\hat{V}_m$.

$\hat{V}_0$ ($\alpha_{i,0} = \bar{u}_0$). Podemos reducir el sistema eliminando la primera fila e incorporando esta igualdad en el resto de ecuaciones, resultando finalmente el siguiente sistema:

$$\begin{pmatrix} \frac{1}{|\hat{V}_1|}\int_{\hat{V}_1}\Psi_1 & \cdots & \frac{1}{|\hat{V}_1|}\int_{\hat{V}_1}\Psi_{l-1} \\ \vdots & \ddots & \vdots \\ \frac{1}{|\hat{V}_{s-1}|}\int_{\hat{V}_{s-1}}\Psi_1 & \cdots & \frac{1}{|\hat{V}_{s-1}|}\int_{\hat{V}_{s-1}}\Psi_{l-1} \end{pmatrix} \begin{pmatrix} \alpha_{i,1} \\ \alpha_{i,2} \\ \vdots \\ \alpha_{i,l-1} \end{pmatrix} = \begin{pmatrix} \bar{u}_1 - \bar{u}_0 \\ \bar{u}_2 - \bar{u}_0 \\ \vdots \\ \bar{u}_{s-1} - \bar{u}_0 \end{pmatrix} \tag{4.10}$$

que se resuelve aplicando mínimos cuadrados, es decir, si A_i es la matriz izquierda de (4.10), α_i es el vector de coeficientes, y u es el vector de valores promedio de la derecha, $\alpha_i = (A_i^T A_i)^{-1} A_i^T u$.

Nótese que (4.8) sólo se cumple para el primer triángulo del stencil. Para el resto de triángulos, la igualdad de (4.8) se verifica de forma aproximada.

Para el cálculo de los coeficientes a_{mn} de la matriz A_i, $1 \leq m < s$, $1 \leq n < l$, se emplean puntos de cuadratura, del siguiente modo:

$$a_{mn} = \sum_{k=1}^{c} \beta_k \, \Psi_n(T_m(\hat{x}_k, \hat{y}_k))$$

donde c es el número de puntos de cuadratura, β_k es el peso asociado al punto de cuadratura k-ésimo, $(\hat{x}_k, \hat{y}_k)$ es el punto de cuadratura k-ésimo en el triángulo $\hat{V}_0$, y T_m es la transformación que lleva un punto del triángulo $\hat{V}_0$ al triángulo $\hat{V}_m$ (ver Figura 4.3). Esta transformación viene dada por:

$$\begin{pmatrix} \hat{x}_m \\ \hat{y}_m \end{pmatrix} = \begin{pmatrix} \hat{x}_{2m} - \hat{x}_{1m} & \hat{x}_{3m} - \hat{x}_{1m} \\ \hat{y}_{2m} - \hat{y}_{1m} & \hat{y}_{3m} - \hat{y}_{1m} \end{pmatrix} \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} + \begin{pmatrix} \hat{x}_{1m} \\ \hat{y}_{1m} \end{pmatrix}$$

donde $(\hat{x}_{1m}, \hat{y}_{1m})$, $(\hat{x}_{2m}, \hat{y}_{2m})$ y $(\hat{x}_{3m}, \hat{y}_{3m})$ son los vértices del triángulo $\hat{V}_m$.

Los valores promedio $\bar{u}_0, \ldots, \bar{u}_{s-1}$ necesarios para construir el vector de la derecha de (4.10) se obtienen para el estado inicial aplicando puntos de cuadratura en el triángulo correspondiente del dominio original. En sucesivas iteraciones de la simulación, los valores promedio de W vendrán dados simplemente por el valor de W en cada volumen.

4.4. Reconstruccion WENO

La opción más sencilla para la construcción del stencil $\mathcal{S}_i$ asociado al volumen V_i es ir recorriendo todos los triángulos vecinos de V_i por aristas hasta llegar al número de elementos deseado (como se dijo en la sección anterior, 6 en orden 2, y 12 orden 3). Sin embargo, en este caso surge un problema si en el stencil aparecen discontinuidades, ya que en este caso el operador de reconstrucción no es TVD. A fin de construir un operador de reconstrucción que sea TVD, es necesario detectar las regiones que tienen discontinuidades y las que son suaves. Para ello, consideraremos varios stencils asociados a distintas regiones del plano. Concretamente, consideraremos siete stencils: uno denominado el stencil central, y los otros seis se corresponden con las seis regiones del plano en las que dividimos el sistema de referencia (ver Figura 4.2). Notaremos como $\mathcal{S}_{i,0}$ al stencil central, y $\mathcal{S}_{i,j}$, $1 \leq j \leq 6$ al stencil asociado a la región j del plano.

El triángulo V_i es el primer elemento de todos los stencils $\mathcal{S}_{i,j}$, $0 \leq j \leq 6$. El stencil central se completa del modo indicado en el párrafo anterior: se van recorriendo vecinos de V_i por aristas y se van añadiendo al stencil central hasta alcanzar el número de elementos adecuado. Para completar el resto de stencils, de nuevo vamos recorriendo todos los vecinos de V_i por aristas, los llevamos al sistema de referencia aplicando la transformación (4.6) y los añadimos al stencil asociado a la región del sistema de referencia donde caiga el triángulo (concretamente, miramos en cuál de las seis regiones cae el baricentro del triángulo).

Resumidamente, los stencils tienen la siguiente forma:

- $\mathcal{S}_{i,0}$ (stencil central): V_i, <vecinos de V_i por aristas>.
- $\mathcal{S}_{i,j}$, $1 \leq j \leq 6$: V_i, <vecinos de V_i que, al llevarlos al sistema de referencia, sus baricentros caigan en la región j>.

Se van recorriendo vecinos de V_i hasta que todos los stencils tengan el número de elementos necesario o hasta llegar a un máximo número de vecinos procesados. Si un stencil ya tiene el número de elementos requerido, no se añaden más triángulos a dicho stencil. Si tras llegar al máximo número de vecinos procesados, hay stencils que no han alcanzado el número de elementos necesario, dichos stencils se descartan y no se tienen en cuenta en el proceso de reconstrucción, como se verá a continuación.

La creación de los stencils para cada volumen se realiza una única vez al principio y sólo depende de la estructura de la malla, es independiente del problema que se simule.

Llamamos *stencil completo* a un stencil que tiene el número de elementos necesario. Una vez construidos los siete stencils, el valor reconstruido en un punto (x, y) del volumen V_i en el dominio original se obtiene ponderando las reconstrucciones obtenidas con cada stencil, del siguiente modo:

$$P_i^{\text{WENO}}(x, y) = \sum_{j=0}^{6} m_{i,j}\, P_{i,j}(x, y)$$

donde $m_{i,j}$ es el estimador de regularidad asociado al stencil $\mathcal{S}_{i,j}$, y $P_{i,j}(x, y)$ es el valor reconstruido en el punto (x, y) aplicando (4.7) y considerando el stencil $\mathcal{S}_{i,j}$.

La suma de todos los estimadores de regularidad de un volumen es igual a 1, y se obtienen del siguiente modo:

$$m_{i,j} = \begin{cases} \dfrac{\tilde{\omega}_{i,j}}{\sum_{k=0}^{6} \tilde{\omega}_{i,k}} & \text{si } \mathcal{S}_{i,j} \text{ es un stencil completo} \\ 0 & \text{en otro caso} \end{cases} \tag{4.11}$$

siendo

$$\tilde{\omega}_{i,j} = \begin{cases} \dfrac{\varphi_j}{(\sigma_{i,j} + \varepsilon)^r} & \text{si } \mathcal{S}_{i,j} \text{ es un stencil completo} \\ 0 & \text{en otro caso} \end{cases} \tag{4.12}$$

En este trabajo se usan los valores $\varepsilon = 10^{-10}$ y $r = 8$. φ_j se define como:

$$\varphi_j = \begin{cases} 10^5 & \text{si } j = 0 \\ 1 & \text{en otro caso} \end{cases}$$

Es decir, para el stencil central, φ_j es mucho mayor que para los demás stencils. Valores menores de φ_0 dan mejores resultados en discontinuidades, mientras que valores mayores normalmente dan soluciones más suaves.

Por su parte, $\sigma_{i,j}$ se obtiene como la suma de las integrales en $\hat{V}_0$ del cuadrado de cada una de las derivadas no nulas de $P_{i,j}$, es decir, en orden 2 tenemos:

$$\sigma_{i,j} = \int_{\hat{V}_0} \left(\frac{\partial P_{i,j}}{\partial x} \right)^2 + \int_{\hat{V}_0} \left(\frac{\partial P_{i,j}}{\partial y} \right)^2$$

mientras que en orden 3, $\sigma_{i,j}$ viene dado por:

$$\sigma_{i,j} = \int_{\hat{V}_0} \left(\frac{\partial P_{i,j}}{\partial x}\right)^2 + \int_{\hat{V}_0} \left(\frac{\partial P_{i,j}}{\partial y}\right)^2 + \int_{\hat{V}_0} \left(\frac{\partial^2 P_{i,j}}{\partial x^2}\right)^2 + \int_{\hat{V}_0} \left(\frac{\partial^2 P_{i,j}}{\partial y^2}\right)^2 + 2\int_{\hat{V}_0} \left(\frac{\partial^2 P_{i,j}}{\partial x \partial y}\right)^2.$$

Sea $\alpha_{i,j,k}$ el coeficiente $\alpha_{i,k}$ asociado al stencil completo $\mathcal{S}_{i,j}$. La expresión asociada a $\sigma_{i,j}$ sólo depende de los coeficientes $\alpha_{i,j,0}, \ldots, \alpha_{i,j,l-1}$, y puede obtenerse mediante algún software matemático de cálculo simbólico, como Maple [Map] o Matlab [Mat]. Concretamente, el valor de $\sigma_{i,j}$ en orden 2 es:

$$\sigma_{i,j} = \frac{5}{2}\,{\alpha_{i,j,1}}^2 + 3\,\alpha_{i,j,1}\alpha_{i,2} + \frac{9}{2}\,{\alpha_{i,j,2}}^2.$$

En orden 3, $\sigma_{i,j}$ viene dado por:

$$\begin{aligned}\sigma_{i,j} \;=\; & 114\,{\alpha_{i,j,3}}^2 + \frac{5}{2}\,{\alpha_{i,j,1}}^2 + \frac{10}{3}\,\alpha_{i,j,1}\alpha_{i,j,4} + \frac{470}{3}\,{\alpha_{i,j,4}}^2 + \frac{2}{3}\,\alpha_{i,j,1}\alpha_{i,j,3} \\ & + \frac{428}{3}\,\alpha_{i,j,3}\alpha_{i,j,4} + 2\,\alpha_{i,j,2}\alpha_{i,j,3} + 38\,\alpha_{i,j,3}\alpha_{i,j,5} + \frac{614}{3}\,\alpha_{i,j,4}\alpha_{i,j,5} + 2\,\alpha_{i,j,2}\alpha_{i,j,4} \\ & - \frac{4}{3}\,\alpha_{i,j,1}\alpha_{i,j,5} + 212\,{\alpha_{i,j,5}}^2 + 3\,\alpha_{i,j,1}\alpha_{i,j,2} + \frac{9}{2}\,{\alpha_{i,j,2}}^2 - 4\,\alpha_{i,j,2}\alpha_{i,j,5}.\end{aligned}$$

El significado conceptual de los estimadores de regularidad es el siguiente: la idea es dar más peso a los stencils que no contienen discontinuidades. En stencils donde aparezcan discontinuidades, el valor de $\sigma_{i,j}$ crecerá (ya que las derivadas crecen en las discontinuidades), por lo que el estimador $m_{i,j}$ será prácticamente cero. Si tuviéramos la seguridad de que no existen discontinuidades en el dominio, bastaría con calcular únicamente el stencil central y efectuar todas las reconstrucciones a partir de este stencil. En este caso no sería necesario el cálculo de todos los demás stencils ni de los estimadores de regularidad.

4.5. Observaciones Finales

4.5.1. Variables Reconstruidas

El proceso de reconstrucción WENO descrito en la sección 4.4 se aplicará sobre la profundidad H y las siguientes variables: $\mu_1 = h_1 + h_2 - H$, $q_{1,x}$, $q_{1,y}$, $\mu_2 = h_2 - H$, $q_{2,x}$ y $q_{2,y}$. Esta elección de las variables a reconstruir garantiza que la reconstrucción obtenida en una situación de agua en reposo (esto es, $q_{1,x} = q_{1,y} = 0$, μ_1 constante y μ_2 constante) permanece invariante, con lo que el esquema de alto orden resultante no altera este tipo de soluciones (es exactamente bien equilibrado para el agua en reposo).

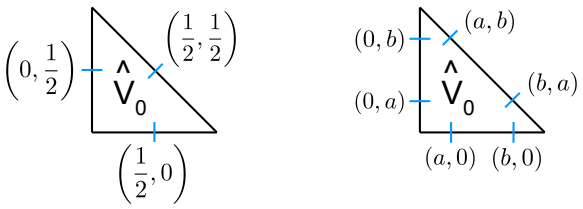


Figura 4.4: Puntos de las aristas de $\hat{V}_0$ donde se aplican las reconstrucciones para todos los volúmenes en el cálculo de la integral de la arista. (a) Orden 2, (b) Orden 3, $a = 0.5 - \frac{1}{2\sqrt{3}}$, $b = 0.5 + \frac{1}{2\sqrt{3}}$.

Dada una arista Γ_{ij}, las reconstrucciones $h^{\pm}_{1,ij}$ se obtienen como $\mu^{\pm}_{1,ij} - \mu^{\pm}_{2,ij}$, y las reconstrucciones $h^{\pm}_{2,ij}$ se obtienen como $\mu^{\pm}_{2,ij} + H^{\pm}_{ij}$. Asimismo, dado que el fondo no varía durante la simulación, las reconstrucciones $H^{\pm}_{ij}$, al igual que el proceso de creación de los stencils, se realizarán una sola vez al principio y se almacenarán los valores resultantes para cada volumen.

El proceso de reconstrucción también debe hacerse entre cada una de las iteraciones de (4.3) y (4.4) para todos los puntos de cuadratura y variables.

Dado que las reconstrucciones se realizan en el sistema de referencia, los puntos de las aristas sobre los que se calculan las reconstrucciones siempre son los mismos para todos los volúmenes (ver Figura 4.4).

La contribución al cálculo del Δt local de un volumen obtenida en cada punto de cuadratura de una arista también debe ponderarse con el peso correspondiente.

Respecto al sistema (4.10), como la matriz $(A_i^T A_i)^{-1} A_i^T$ necesaria para la resolución de dicho sistema sólo depende de la estructura de la malla, se computará una vez al principio. Nótese que, para cada volumen, se deben almacenar todos sus stencils completos y, para cada uno de ellos, esta matriz.

4.5.2. Cálculo de las Integrales del Volumen

Finalmente, comentaremos brevemente el cálculo de las integrales del volumen que aparecen en (4.5). Como se comentó en la sección 4.2, dichas integrales se calculan mediante puntos de cuadratura (1 en orden 2, y 4 en orden 3). Dado que las reconstrucciones se realizan en el sistema de referencia, los puntos que se consideran siempre son los mismos para todos los volúmenes: los puntos de cuadratura del triángulo $\hat{V}_0$. Nótese que es necesario obtener las reconstrucciones de H en cada punto de cuadratura de $\hat{V}_0$. De nuevo, dado que el fondo no varía durante la simulación, estas reconstrucciones se realizarán sólo una vez al principio y se almacenarán los valores obtenidos.

Para la reconstrucción de la derivada en x de $P_i(x,y)$, tenemos que:

$$\frac{\partial}{\partial x} P_i(x,y) = \frac{\partial}{\partial x} \left(\sum_{j=0}^{6} m_{i,j} \sum_{k=0}^{l-1} \alpha_{i,j,k} \, \Psi_k(R(x,y)) \right)$$

donde $R(x,y)$ es la transformación que lleva un punto del dominio original al sistema de referencia. Dicha transformación se representó en (4.6). Nótese que el resultado de $R(x,y)$ es el punto de cuadratura de $\hat{V}_0$ que estemos tratando en ese momento.

Aplicando la regla de la cadena, obtenemos finalmente:

$$\frac{\partial}{\partial x} P_i(x,y) = \sum_{j=0}^{6} m_{i,j} \sum_{k=0}^{l-1} \alpha_{i,j,k} \left(\frac{\partial}{\partial \hat{x}} \Psi_k(\hat{x},\hat{y}) \frac{\partial \hat{x}}{\partial x} + \frac{\partial}{\partial \hat{y}} \Psi_k(\hat{x},\hat{y}) \frac{\partial \hat{y}}{\partial x} \right).$$

donde $\frac{\partial \hat{x}}{\partial x}$ y $\frac{\partial \hat{y}}{\partial x}$ se calculan a partir de (4.6).

El cálculo de la derivada en y de $P_i(x,y)$ se realiza de forma análoga, llegando a la siguiente fórmula:

$$\frac{\partial}{\partial y} P_i(x,y) = \sum_{j=0}^{6} m_{i,j} \sum_{k=0}^{l-1} \alpha_{i,j,k} \left(\frac{\partial}{\partial \hat{x}} \Psi_k(\hat{x},\hat{y}) \frac{\partial \hat{x}}{\partial y} + \frac{\partial}{\partial \hat{y}} \Psi_k(\hat{x},\hat{y}) \frac{\partial \hat{y}}{\partial y} \right).$$

Los valores $\frac{\partial \hat{x}}{\partial x}$, $\frac{\partial \hat{y}}{\partial x}$, $\frac{\partial \hat{x}}{\partial y}$ y $\frac{\partial \hat{y}}{\partial y}$ (4 escalares), dado que no varían durante la simulación, se calcularán al principio para cada volumen y se almacenarán.

4.6. Implementación en CUDA

En esta sección se identifican las fuentes de paralelismo de datos existentes en los esquemas numéricos de alto orden que se han descrito a lo largo de las anteriores secciones. Asimismo, se explica su adaptación a GPU y las implementaciones que se han realizado usando CUDA.

4.6.1. Fuentes de Paralelismo

A continuación exponemos un algoritmo paralelo basado en los esquemas numéricos descritos en las anteriores secciones de este capítulo. Las Figuras 4.5 y 4.6 muestran esquemáticamente las etapas de dicho algoritmo paralelo para los órdenes 2 y 3, respectivamente, donde los pasos principales aparecen etiquetados con un número dentro de un círculo, y las principales fuentes de paralelismo de datos se representan con rectángulos superpuestos.

Al igual que en orden 1, en primer lugar la malla de volúmenes finitos debe construirse e inicializarse. Seguidamente se calcula el incremento de

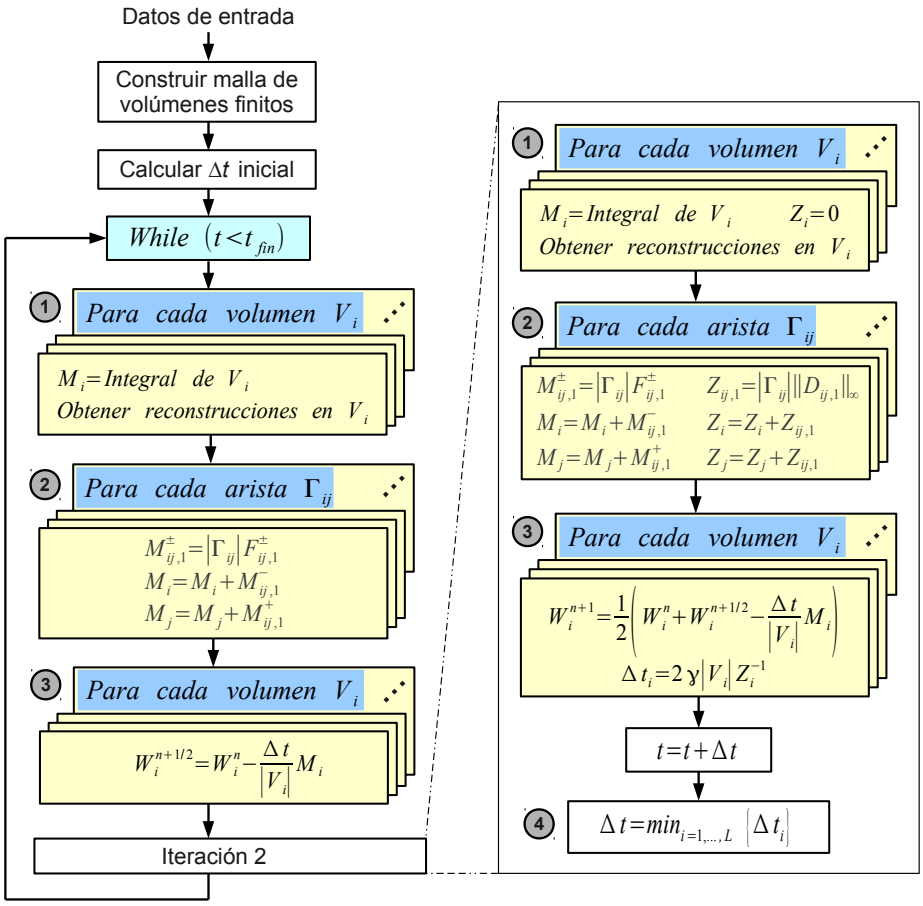


Figura 4.5: Fuentes de paralelismo del esquema de orden 2.

tiempo Δt inicial y se entra en el bucle principal, donde se aplican las iteraciones Runge-Kutta dadas por (4.3) en orden 2 y (4.4) en orden 3 hasta que se alcanza el tiempo final de simulación t_{fin}. Durante el proceso se utilizan los mismos acumuladores M_i (un vector 6×1) y Z_i (un escalar) empleados en orden 1. El cálculo del paso de tiempo inicial es idéntico al que se realizó en los esquemas de orden 1 (ver Figura 3.4). A continuación se describen los pasos principales del algoritmo paralelo:

1. **Obtener integral y reconstrucciones para cada volumen**: Al principio de cada iteración Runge-Kutta, se realizan las siguientes operaciones para cada volumen V_i:

   - Se almacena en el acumulador M_i el valor de las integrales del volumen de (4.5). En la sección 4.5.2 se describió el cálculo de dichas integrales.

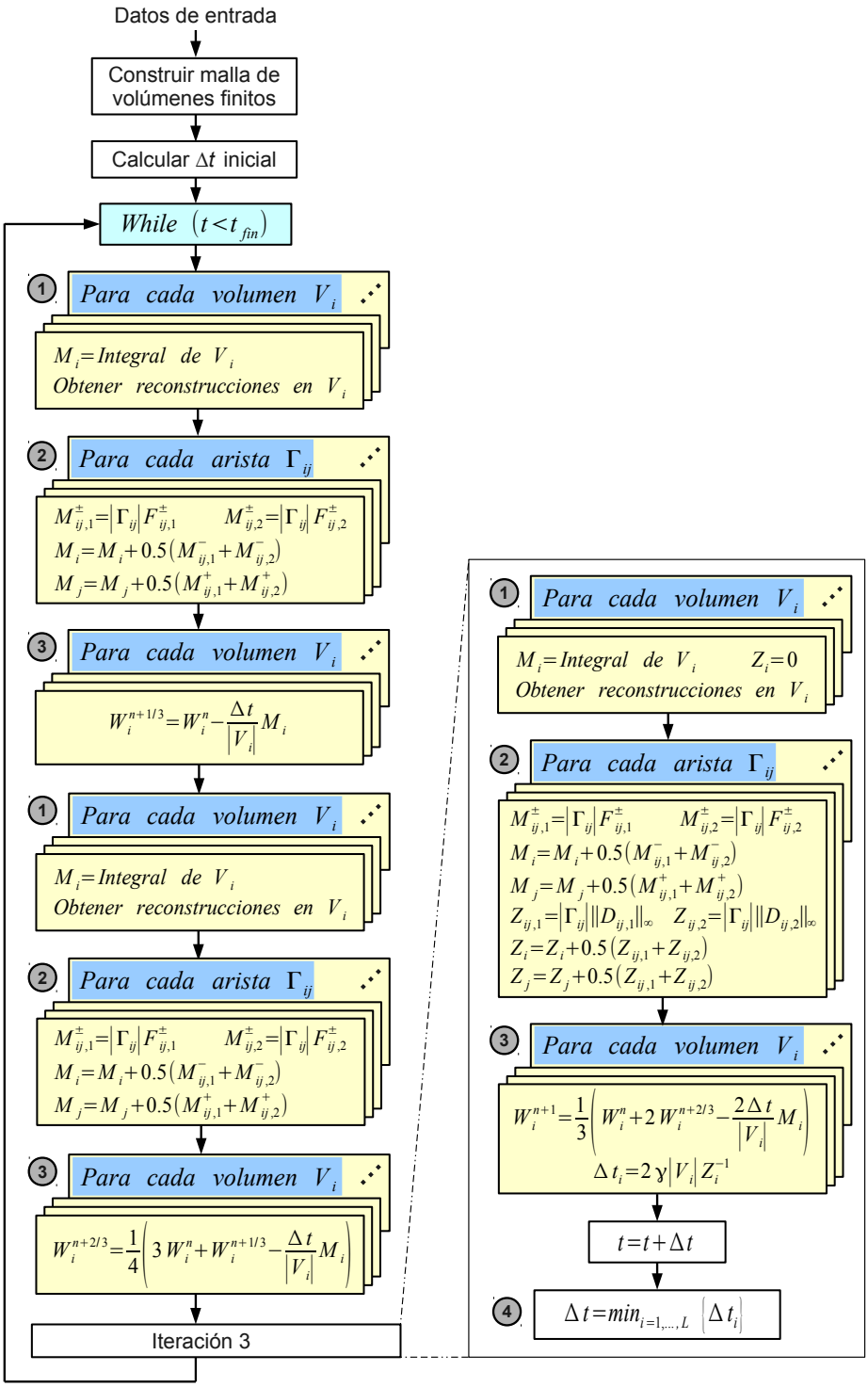


Figura 4.6: Fuentes de paralelismo del esquema de orden 3.

- Se inicializa a cero el acumulador Z_i.
- Para cada punto de cuadratura k de cada arista de V_i se obtienen las reconstrucciones $W^{\pm}_{ij,k}$ y $H^{\pm}_{ij,k}$ asociadas al volumen, tal y como se explicó en la sección 4.4.

② **Procesamiento de las aristas**: Para cada arista Γ_{ij} común a dos volúmenes adyacentes V_i y V_j se realizan los siguientes cálculos:

- Para cada punto de cuadratura k de la arista ($k = 1$ en orden 2, y $k = 1, 2$ en orden 3) se calcula el vector $M^{\pm}_{ij,k} = |\Gamma_{ij}| \, F^{\pm}_{ij,k}$, donde $F^{\pm}_{ij,k}$ representa el cálculo de $\mathcal{F}^{PVM^{\pm}}_{ij}$ (ver sección 3.4) considerando los valores reconstruidos $W^{\pm}_{ij,k}$ y $H^{\pm}_{ij,k}$ asociados al punto de cuadratura k-ésimo de la arista. Las contribuciones finales $M^{\pm}_{ij}$ de la arista se obtienen ponderando todos los $M^{\pm}_{ij,k}$ con los pesos de cuadratura correspondientes. Estas contribuciones deben sumarse a los acumuladores M_i y M_j asociados a V_i y V_j, respectivamente.
- Para cada punto de cuadratura k de la arista se calcula el valor $Z_{ij,k} = |\Gamma_{ij}| \, \|\mathcal{D}_{ij,k}\|_{\infty}$, donde $\mathcal{D}_{ij,k}$ representa la matriz $\mathcal{D}_{ij}$ de la Ecuación (3.5) considerando los valores $W^{\pm}_{ij,k}$ asociados al punto de cuadratura k-ésimo de la arista. La contribución final Z_{ij} de la arista se obtiene ponderando todos los $Z_{ij,k}$ con los pesos de cuadratura correspondientes. Esta contribución debe sumarse a los acumuladores Z_i y Z_j asociados a V_i y V_j, respectivamente.

Para cada arista se procesan sus puntos de cuadratura de forma secuencial, y el procesamiento de todas las aristas puede realizarse en paralelo, ya que los cálculos propios de cada arista son independientes de los cálculos efectuados para las restantes aristas.

③ **Cálculo del siguiente estado y del valor local Δt_i para cada volumen**: Para cada volumen V_i se realizan los siguientes cálculos:

- El siguiente estado $W_i^{n+1/2}$, $W_i^{n+1/3}$, $W_i^{n+2/3}$ o W_i^{n+1} (dependiendo del orden y de la iteración Runge-Kutta en la que nos encontremos) se obtiene a partir del valor del acumulador M_i aplicando la expresión correspondiente de (4.3) en orden 2 y de (4.4) en orden 3.
- El valor local Δt_i se obtiene a partir del valor del acumulador Z_i del siguiente modo (ver Ecuación (3.5)): $\Delta t_i = 2\gamma \, |V_i| \, Z_i^{-1}$.

Al igual que en orden 1, el procesamiento de los volúmenes también puede hacerse en paralelo, ya que los cálculos asociados a cada volumen son independientes de los realizados para los restantes volúmenes.

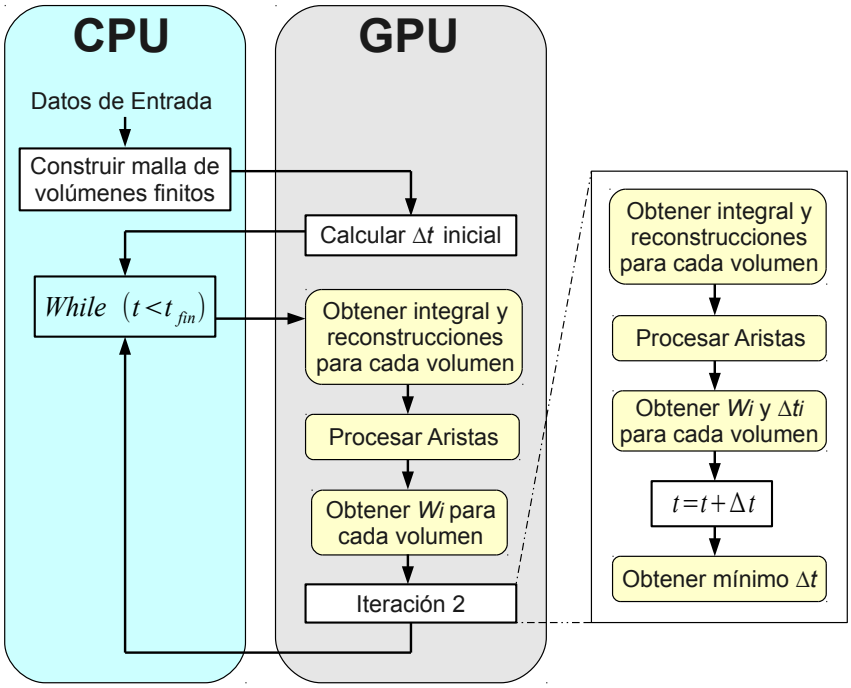


Figura 4.7: Algoritmo paralelo implementado en CUDA para el orden 2.

④ **Obtención del mínimo** Δt: Este paso es idéntico al utilizado en los esquemas de orden 1 (ver sección 3.5.1).

Nótese que en los pasos ①, ② y ③ sólo se realizan las operaciones estrictamente necesarias dependiendo de la correspondiente iteración Runge-Kutta. Concretamente, los cálculos asociados a la obtención del Δt local de cada volumen sólo se efectúan en la última iteración Runge-Kutta (ver Figuras 4.5 y 4.6).

Al igual que los esquemas vistos en el capítulo 3, estos esquemas de alto orden también presentan un alto grado de paralelismo de datos, por lo que son adecuados para ser implementados en arquitecturas CUDA.

4.6.2. Detalles de la Implementación

En esta sección describiremos los aspectos más destacados de las implementaciones del algoritmo paralelo expuesto en la sección 4.6.1 que se han realizado usando el entorno de programación CUDA. Los pasos genéricos del algoritmo implementado se muestran en la Figura 4.7 para el orden 2, y en la Figura 4.8 para el orden 3. Cada paso que se ejecuta en la GPU se asigna a un kernel de CUDA. A continuación describimos los pasos del algoritmo:

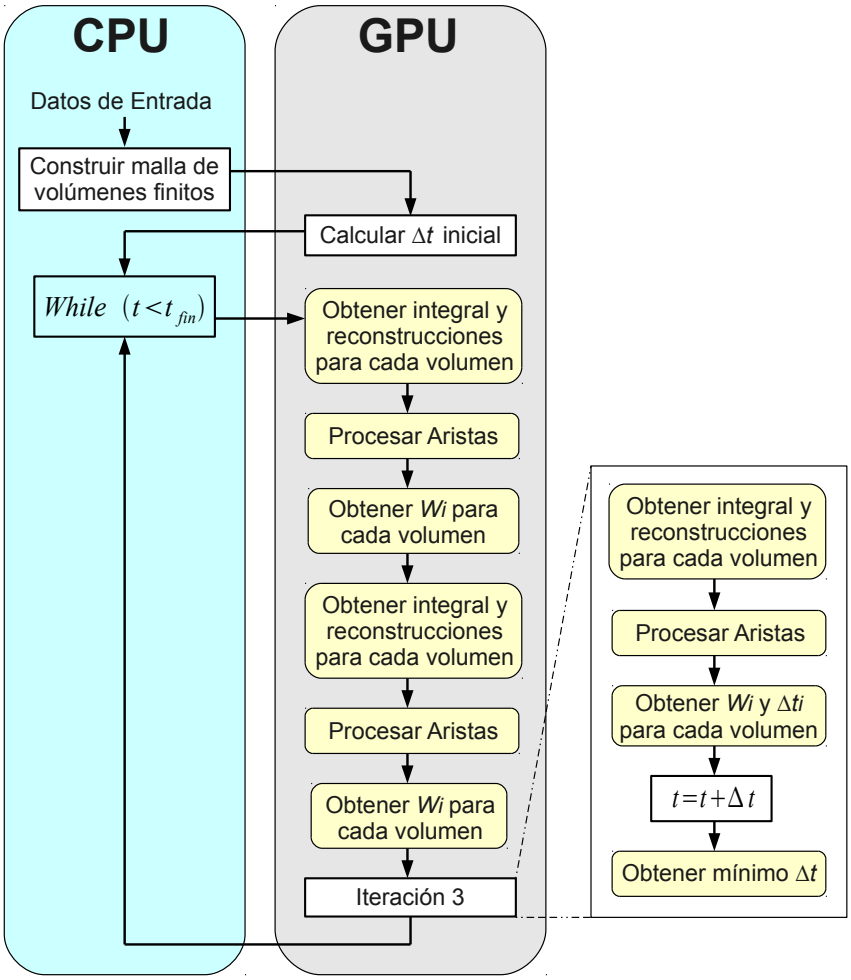


Figura 4.8: Algoritmo paralelo implementado en CUDA para el orden 3.

- **Construir malla de volúmenes finitos**: En este paso se construye la estructura de datos que se usará en GPU. Para cada volumen V_i almacenamos la siguiente información:

  - Al igual que en orden 1, se almacena su estado (h_1, $q_{1,x}$, $q_{1,y}$, h_2, $q_{2,x}$ y $q_{2,y}$), su profundidad H y su área en dos arrays de elementos `float4` como texturas 1D, donde el tamaño de ambos arrays es el número de volúmenes. En la primera textura guardamos h_1, h_2, H y el área, mientras que en la segunda guardamos $q_{1,x}$, $q_{1,y}$, $q_{2,x}$ y $q_{2,y}$. Se ha elegido esta distribución de los datos porque, de este modo, en la reconstrucción de μ_1 y μ_2 (ver sección 4.5.1) sólo será necesario leer de la primera textura.

- Dado que los estados intermedios $W_i^{n+1/2}$ (en orden 2), o $W_i^{n+1/3}$ y $W_i^{n+2/3}$ (en orden 3) se utilizan en la obtención del siguiente estado W_i^{n+1} de un volumen (ver Ecuaciones (4.3) y (4.4)), es necesario almacenar otro estado auxiliar. Por ello, se definen otros dos arrays de elementos `float4` en memoria global, donde el tamaño de cada uno es el número de volúmenes. En este caso no es necesario asociarlos a texturas, ya que cada posición de estos arrays sólo será leída por la hebra asociada a dicha posición en el kernel de obtención del siguiente estado.
- Debemos almacenar el número de stencils completos del volumen y los elementos de cada uno de estos stencils. Los elementos son simplemente índices (posiciones) en los vectores donde se almacenan los estados de los volúmenes. Dado que cada stencil completo tiene 6 elementos en orden 2 y 12 elementos en orden 3, estas posiciones se guardan para todos los volúmenes en un vector de elementos `int2` en orden 2 e `int4` en orden 3, de forma que, para ambos órdenes, cada stencil utiliza tres posiciones de este vector. Se han probado tres tipos de almacenamiento distinto para este vector: como una textura 1D, cargándolo en memoria compartida y cargándolo en un vector local, obteniéndose en la mayoría de los casos mejores tiempos de ejecución con una textura 1D.

  Nótese que el número de stencils completos puede ser distinto para cada volumen. Por ello, también es necesario guardar la posición del vector a partir de la cual se almacenan los stencils del volumen. Sea c el número de stencils completos del volumen, y p la posición del vector donde empiezan sus stencils. Dado que $1 \leq c \leq 7$, podemos ahorrar memoria almacenando ambos valores como un único entero de la forma $8\,p + c$. De este modo, usamos un vector de elementos `int` para guardar este entero para cada volumen, y p y c se obtienen como el resultado y el resto, respectivamente, de dividir dicho entero entre 8.
- También se almacena para cada stencil completo la matriz $(A_i^T A_i)^{-1} A_i^T$ necesaria para obtener los coeficientes de cada stencil (ver sección 4.3). Para todos los volúmenes y stencils, esta matriz se almacena por filas en un vector de elementos `float` en memoria global. La posición en este vector donde empiezan las matrices de un determinado volumen se obtiene a partir de la posición p anterior.
- Los valores reconstruidos en cada punto de cuadratura de una arista del volumen se almacenan en dos vectores de elementos `float4` en memoria global. En el primero se guardan las reconstrucciones de μ_1, $q_{1,x}$, $q_{1,y}$ y H, mientras que en el segundo se guardan las reconstrucciones de μ_2, $q_{2,x}$ y $q_{2,y}$. Las reconstruccio-

nes en los puntos asociados a cada volumen se almacenan consecutivamente en dichos vectores, de forma que para cada volumen se usan 3 posiciones en orden 2, y 6 posiciones en orden 3.

- Los valores de H reconstruidos en cada punto de cuadratura de un volumen necesarios para la obtención de las integrales del volumen (ver sección 4.5.2) se almacenan en un vector de elementos `float4` en memoria global. En orden 2 (un punto de cuadratura) sólo se usará el primer `float`, mientras que en orden 3 (4 puntos de cuadratura) se usarán los cuatro.
- Los cuatro escalares $\frac{\partial \hat{x}}{\partial x}$, $\frac{\partial \hat{y}}{\partial x}$, $\frac{\partial \hat{x}}{\partial y}$ y $\frac{\partial \hat{y}}{\partial y}$ necesarios para la obtención de las integrales del volumen (ver sección 4.5.2) se almacenan en un vector de elementos `float4` en memoria global, donde cada posición del vector corresponde a un volumen.

Por su parte, para almacenar los datos de cada arista Γ_{ij} utilizamos las mismas estructuras de datos que en los esquemas de orden 1, es decir, guardamos su normal $(\eta_{ij,x}, \eta_{ij,y})$ en un array de elementos `float2`, y los datos relativos a acumuladores y volúmenes adyacentes en otro array de elementos `int4`, ambos en memoria global (ver sección 3.5.2 para más detalles).

- **Obtener integral y reconstrucciones para cada volumen**: En este paso, cada hebra representa un volumen y calcula la integral del volumen V_i y las reconstrucciones $W^{\pm}_{ij,k}$ en cada punto de cuadratura k de cada arista de V_i del modo descrito en la sección 4.6.1. Nótese que las reconstrucciones $H^{\pm}_{ij,k}$ se calcularon al principio.

- **Procesar aristas**: En el procesamiento de aristas, cada hebra representa una arista y calcula la contribución de la arista a sus volúmenes adyacentes tal y como se describió en la sección 4.6.1.

  La sincronización entre las aristas (hebras) se realiza de forma idéntica a como se hizo en los esquemas de orden 1 y usando los mismos acumuladores (ver sección 3.5.2).

- **Obtener siguiente estado y Δt_i para cada volumen**: En este paso, cada hebra representa un volumen y obtiene el siguiente estado $\left(W_i^{n+1/2}, W_i^{n+1/3}, W_i^{n+2/3} \text{ o } W_i^{n+1}\right)$ y el Δt_i local del volumen V_i tal y como se describe en el paso 3 del algoritmo paralelo expuesto en la sección 4.6.1.

  Las contribuciones finales M_i y Z_i se obtienen del mismo modo que en orden 1.

- **Obtener el mínimo Δt**: Este paso es idéntico al descrito para los esquemas de orden 1 (ver sección 3.5.2).

También se han hecho implementaciones de este algoritmo en CUDA usando doble precisión para ambos órdenes 2 y 3. Las diferencias más destacadas respecto a las implementaciones en simple precisión son las siguientes:

- Los datos de los volúmenes, en lugar de almacenarse en dos arrays de elementos `float4`, se almacenan en cuatro arrays de elementos `double2`. Del mismo modo, los estados auxiliares de los volúmenes también se almacenan en cuatro arrays de elementos `double2`.
- En lugar de seis acumuladores de elementos `float4`, se utilizan nueve acumuladores de elementos `double2` (donde se almacenan las contribuciones a W_i) y tres acumuladores de elementos `double` (donde se almacenan las contribuciones al Δt local de cada volumen).
- Los valores reconstruidos en cada punto de cuadratura de una arista se almacenan en tres arrays de elementos `double2` (donde se guardan las reconstrucciones de μ_1, $q_{1,x}$, $q_{1,y}$, μ_2, $q_{2,x}$ y $q_{2,y}$), y un array de elementos `double` (para la reconstrucción de H), en vez de dos arrays de elementos `float4`.
- Los valores de H reconstruidos en cada punto de cuadratura de un volumen se almacenan en dos arrays de elementos `double2` en vez de un array de elementos `float4`. Lo mismo ocurre con los cuatro escalares necesarios para el cálculo de la integral del volumen.

4.7. Resultados Experimentales

En esta sección ejecutaremos las implementaciones CUDA descritas en la sección 4.6.2 utilizando distintos problemas de ejemplo, y analizaremos los resultados obtenidos, tanto en términos de precisión como de eficiencia.

4.7.1. Análisis de Precisión

Test Cuantitativo En primer lugar realizaremos un análisis cuantitativo de la precisión obtenida con los esquemas de alto orden presentados en este capítulo. Para ello, se considera un problema bicapa que representa un vórtice que gira sobre sí mismo en el centro del dominio $[-5, 5] \times [-5, 5]$, con parámetro CFL $\gamma = 0.9$, $r = 0.9$ y condiciones de contorno de tipo pared. El ejemplo está extraído de [DCPT09] y su estado inicial se muestra en la Figura 4.9. Idealmente, el estado no debe variar durante la simulación. Para todos los órdenes y tamaños de malla hasta 1024000 volúmenes, simularemos 1 segundo usando las implementaciones OpenMP en doble precisión (al igual que en orden 1, se han realizado implementaciones de todos los esquemas de alto orden en CPU para cuatro hebras usando OpenMP, doble precisión numérica y la librería Eigen). Las Tablas 4.1, 4.2 y 4.3 muestran la norma L^1

de la diferencia entre la solución inicial y la obtenida en el tiempo 1 segundo para los órdenes 1, 2 y 3, respectivamente.

Podemos ver que, para todos los tamaños de malla, la diferencia entre la solución inicial y la final es menor cuanto mayor es el orden, lo que refleja la mayor precisión alcanzada al aumentar el orden del esquema numérico.

Test Cualitativo En este punto realizaremos un análisis cualitativo de la precisión obtenida con los esquemas de alto orden explicados en este capítulo. Para ello, para los dos ejemplos descritos en la sección 3.6.1, obtendremos una solución de referencia con la malla de 1024000 volúmenes usando el esquema de orden 2 para un tiempo de simulación dado. A continuación obtendremos, para el mismo tiempo de simulación, distintas soluciones con la malla de 64000 volúmenes empleando todos los órdenes (1, 2 y 3).

Consideraremos los mismos tiempos de simulación utilizados en el análisis de precisión de los esquemas de orden 1, es decir, 1 segundo para el ejemplo 1 y 4 segundos para el ejemplo 2. Todas las ejecuciones se realizarán con las implementaciones CUDA de doble precisión. Las Figuras 4.10 y 4.11 muestran las vistas desde arriba obtenidas para los distintos órdenes en los ejemplos 1 y 2, respectivamente, con la malla de 64000 volúmenes. En ambos ejemplos se aprecia que el esquema de orden 1 es el que ha introducido mayor difusión y que, cuanto mayor es el orden, las discontinuidades aparecen mejor definidas, llegándose incluso a detectar algunas ondas que con un orden menor no se aprecian. Esto último puede comprobarse en la capa 2 del ejemplo 1, donde el esquema de orden 1 no ha sido capaz de detectar pequeñas ondas del fluido que rebotan en los bordes del dominio, mientras que con los órdenes 2 y 3 sí se han detectado.

En las Figuras 4.12 y 4.13 se muestran cortes del plano $y = 0$ con las soluciones obtenidas para todos los órdenes en los ejemplos 1 y 2. Como era de esperar, en ambos ejemplos el esquema de orden 3 es el que más se ha aproximado a la solución de referencia, seguido del esquema de orden 2 y finalmente el de orden 1, siendo la diferencia entre las soluciones obtenidas con los órdenes 1 y 2 mayor que la diferencia entre las obtenidas con los órdenes 2 y 3.

4.7.2. Análisis de Eficiencia

En esta sección estudiaremos la eficiencia computacional de las distintas implementaciones CUDA explicadas en la sección 4.6.2. Para ello, ejecutaremos dichos programas CUDA para cada uno de los dos ejemplos descritos en la sección 3.6.1. Del mismo modo que en orden 1, para cada esquema de alto orden se han realizado dos implementaciones en CPU, una para una sola hebra y otra para cuatro hebras usando OpenMP. Ambas están implementadas en C++, utilizan doble precisión numérica y hacen uso de la librería matemática Eigen.

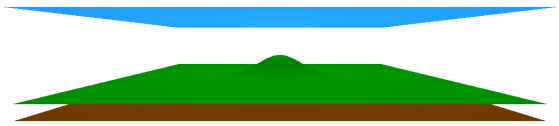


Figura 4.9: Estado inicial del ejemplo del vórtice.

Volúmenes	Norma L^1 $(h_1$	$q_{1,x}$	$q_{1,y}$	h_2	$q_{2,x}$	$q_{2,y})$
4000	$(3.5{\cdot}10^{-1}$	$7.9{\cdot}10^{-1}$	$7.7{\cdot}10^{-1}$	$3.1{\cdot}10^{-1}$	$9.7{\cdot}10^{-2}$	$8.7{\cdot}10^{-2})$
16000	$(1.9{\cdot}10^{-1}$	$4.3{\cdot}10^{-1}$	$4.2{\cdot}10^{-1}$	$1.6{\cdot}10^{-1}$	$5.2{\cdot}10^{-2}$	$4.7{\cdot}10^{-2})$
64000	$(9.5{\cdot}10^{-2}$	$2.3{\cdot}10^{-1}$	$2.3{\cdot}10^{-1}$	$8.3{\cdot}10^{-2}$	$2.8{\cdot}10^{-2}$	$2.5{\cdot}10^{-2})$
256000	$(4.9{\cdot}10^{-2}$	$1.2{\cdot}10^{-1}$	$1.2{\cdot}10^{-1}$	$4.2{\cdot}10^{-2}$	$1.4{\cdot}10^{-2}$	$1.3{\cdot}10^{-2})$
1024000	$(2.5{\cdot}10^{-2}$	$6.4{\cdot}10^{-2}$	$6.5{\cdot}10^{-2}$	$2.1{\cdot}10^{-2}$	$7.3{\cdot}10^{-3}$	$6.6{\cdot}10^{-3})$

Tabla 4.1: Norma L^1 de la diferencia entre la solución inicial y la obtenida en el tiempo 1 seg para el ejemplo del vórtice con el esquema de orden 1.

Volúmenes	Norma L^1 $(h_1$	$q_{1,x}$	$q_{1,y}$	h_2	$q_{2,x}$	$q_{2,y})$
4000	$(7.4{\cdot}10^{-2}$	$1.9{\cdot}10^{-1}$	$1.9{\cdot}10^{-1}$	$6.1{\cdot}10^{-2}$	$3.4{\cdot}10^{-2}$	$3.3{\cdot}10^{-2})$
16000	$(1.0{\cdot}10^{-2}$	$3.9{\cdot}10^{-2}$	$4.2{\cdot}10^{-2}$	$8.1{\cdot}10^{-3}$	$5.7{\cdot}10^{-3}$	$5.9{\cdot}10^{-3})$
64000	$(1.8{\cdot}10^{-3}$	$7.8{\cdot}10^{-3}$	$8.4{\cdot}10^{-3}$	$1.4{\cdot}10^{-3}$	$1.2{\cdot}10^{-3}$	$1.3{\cdot}10^{-3})$
256000	$(4.2{\cdot}10^{-4}$	$1.9{\cdot}10^{-3}$	$2.1{\cdot}10^{-3}$	$3.3{\cdot}10^{-4}$	$3.1{\cdot}10^{-4}$	$3.3{\cdot}10^{-4})$
1024000	$(1.1{\cdot}10^{-4}$	$5.2{\cdot}10^{-4}$	$5.6{\cdot}10^{-4}$	$8.8{\cdot}10^{-5}$	$8.4{\cdot}10^{-5}$	$9.1{\cdot}10^{-5})$

Tabla 4.2: Norma L^1 de la diferencia entre la solución inicial y la obtenida en el tiempo 1 seg para el ejemplo del vórtice con el esquema de orden 2.

Volúmenes	Norma L^1 $(h_1$	$q_{1,x}$	$q_{1,y}$	h_2	$q_{2,x}$	$q_{2,y})$
4000	$(1.5{\cdot}10^{-2}$	$3.8{\cdot}10^{-2}$	$3.8{\cdot}10^{-2}$	$1.3{\cdot}10^{-2}$	$9.6{\cdot}10^{-3}$	$9.6{\cdot}10^{-3})$
16000	$(2.0{\cdot}10^{-3}$	$5.7{\cdot}10^{-3}$	$5.8{\cdot}10^{-3}$	$1.7{\cdot}10^{-3}$	$1.1{\cdot}10^{-3}$	$1.1{\cdot}10^{-3})$
64000	$(2.9{\cdot}10^{-4}$	$1.4{\cdot}10^{-3}$	$1.4{\cdot}10^{-3}$	$2.6{\cdot}10^{-4}$	$2.5{\cdot}10^{-4}$	$2.8{\cdot}10^{-4})$
256000	$(6.6{\cdot}10^{-5}$	$4.1{\cdot}10^{-4}$	$4.2{\cdot}10^{-4}$	$5.3{\cdot}10^{-5}$	$7.4{\cdot}10^{-5}$	$8.4{\cdot}10^{-5})$
1024000	$(3.1{\cdot}10^{-5}$	$1.6{\cdot}10^{-4}$	$1.7{\cdot}10^{-4}$	$2.0{\cdot}10^{-5}$	$2.8{\cdot}10^{-5}$	$3.1{\cdot}10^{-5})$

Tabla 4.3: Norma L^1 de la diferencia entre la solución inicial y la obtenida en el tiempo 1 seg para el ejemplo del vórtice con el esquema de orden 3.

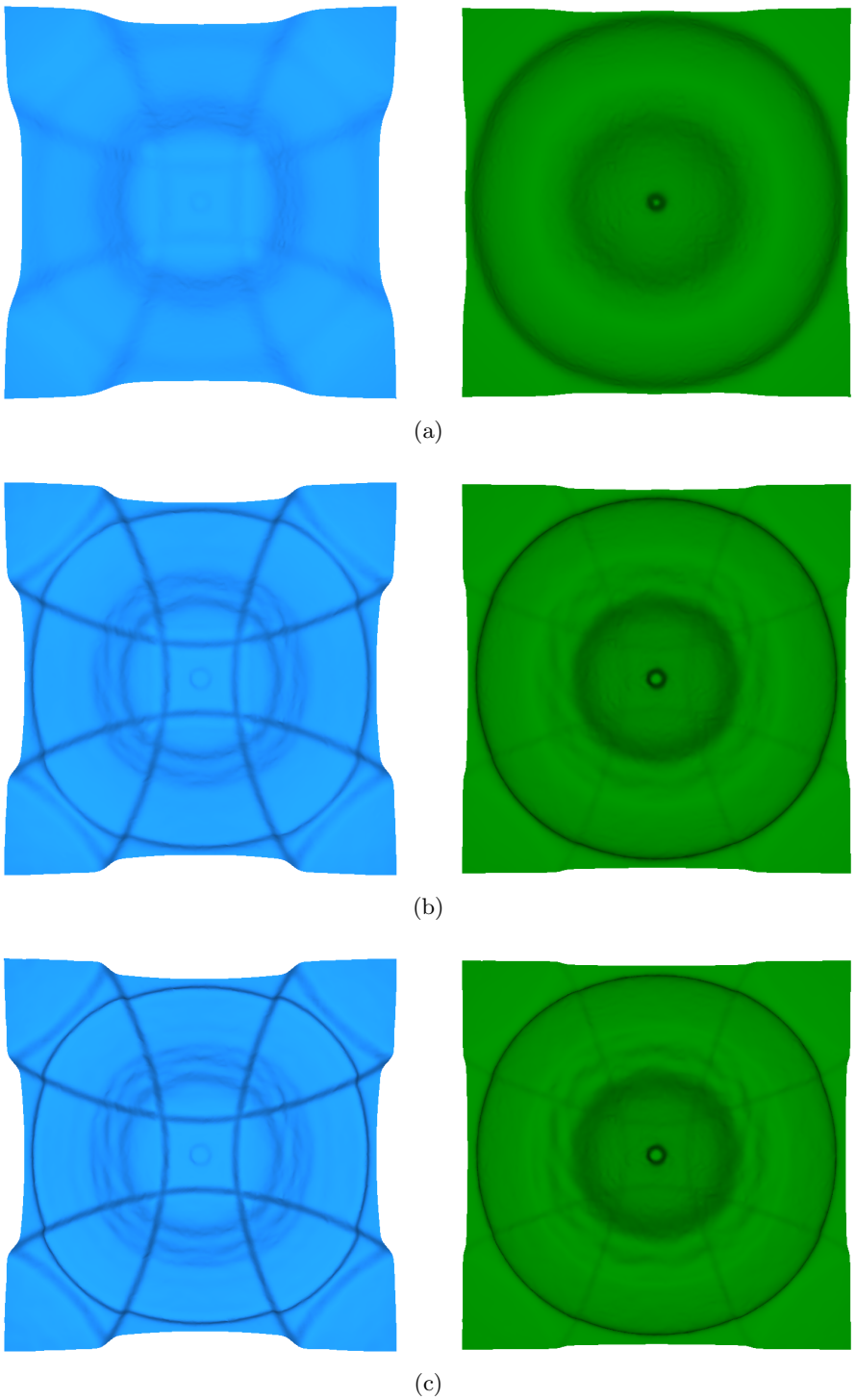
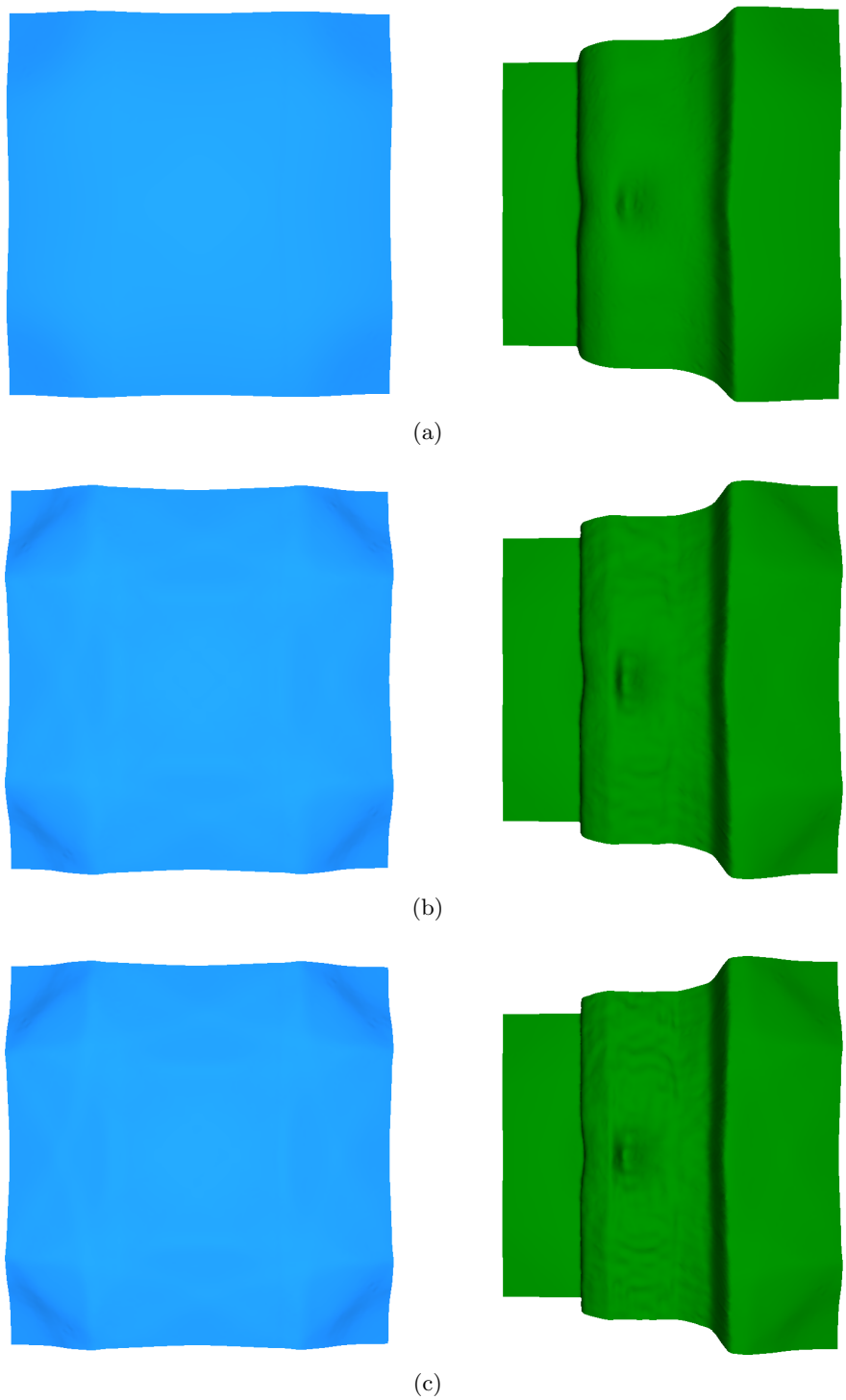


Figura 4.10: Vista desde arriba del estado obtenido para los tres órdenes con la malla de 64000 volúmenes para el ejemplo 1 en el tiempo 1 seg. Izquierda: capa 1. Derecha: capa 2. (a) Orden 1; (b) Orden 2; (c) Orden 3.

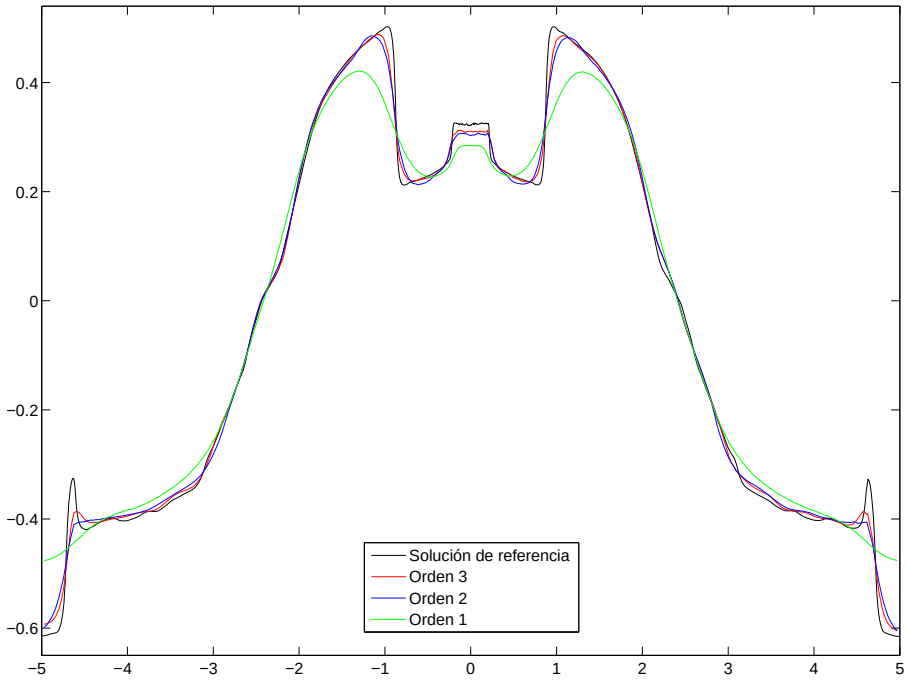


(a)

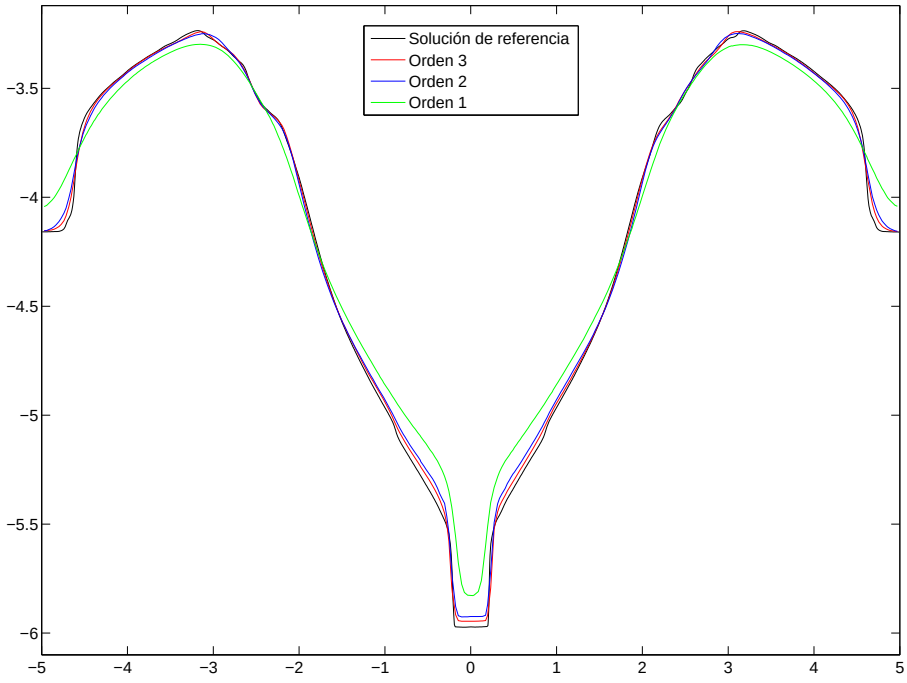
(b)

(c)

Figura 4.11: Vista desde arriba del estado obtenido para los tres órdenes con la malla de 64000 volúmenes para el ejemplo 2 en el tiempo 4 seg. Izquierda: capa 1. Derecha: capa 2. (a) Orden 1; (b) Orden 2; (c) Orden 3.

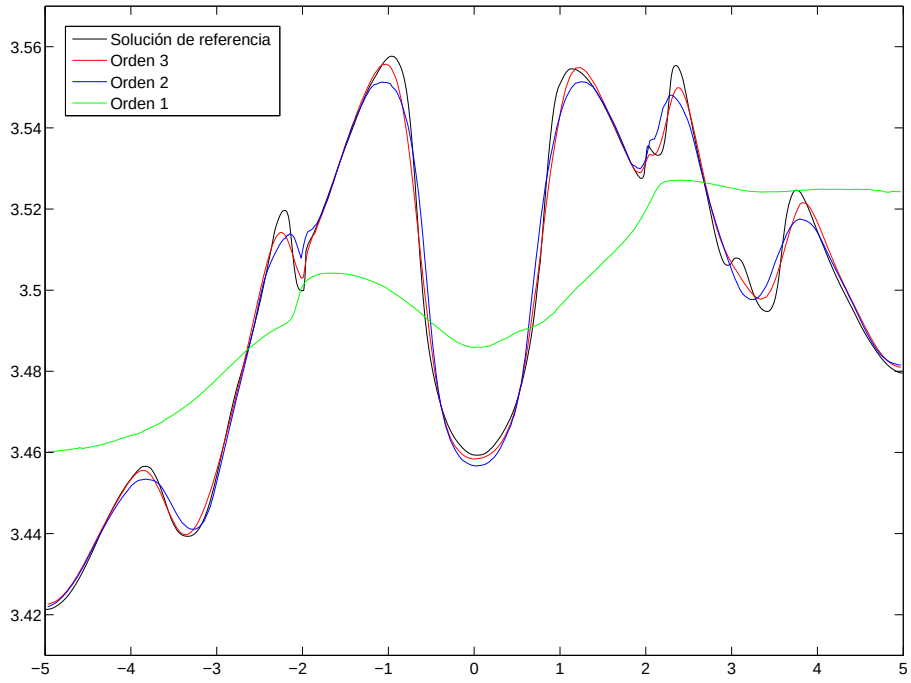


(a)

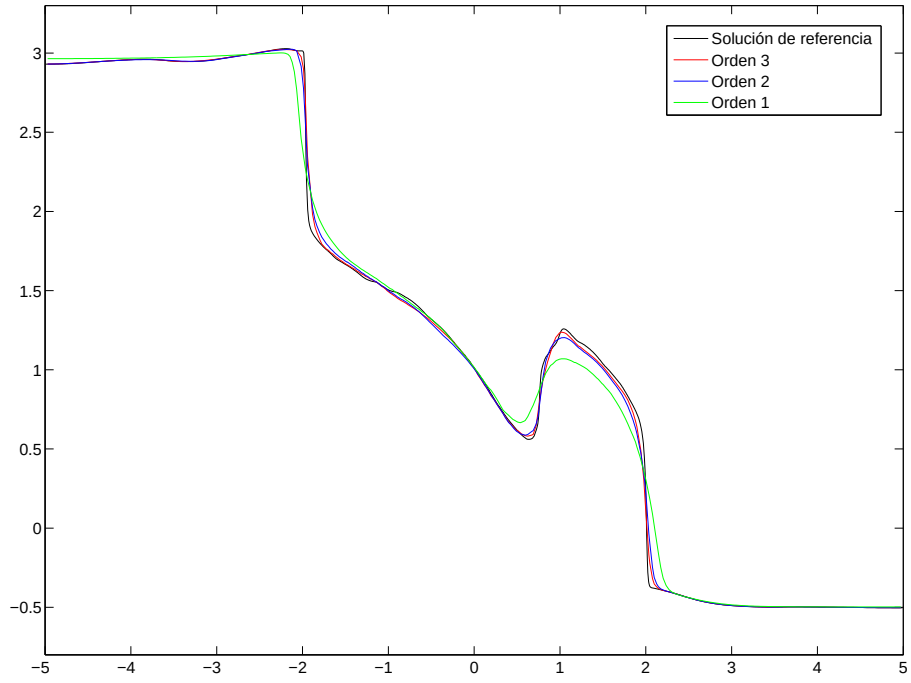


(b)

Figura 4.12: Corte del plano $y = 0$ con las soluciones obtenidas para los tres órdenes con el ejemplo 1 en el tiempo 1 seg: (a) Capa 1; (b) Capa 2.



(a)



(b)

Figura 4.13: Corte del plano $y = 0$ con las soluciones obtenidas para los tres órdenes con el ejemplo 2 en el tiempo 4 seg: (a) Capa 1; (b) Capa 2.

Volúmenes	Iteraciones	CPU 1 hebra	CPU 4 hebras	GTX 580 CUDA SP+DP	GTX 580 CUDA DP
4000	23	2.42	0.81	0.024	0.048
16000	45	19.00	6.42	0.14	0.32
64000	90	152.7	48.14	1.07	2.41
256000	179	1212.3	357.7	8.07	19.03
1024000	357	9710.5	2966.7	62.42	152.3

Tabla 4.4: Tiempos de ejecución en segundos e iteraciones realizadas con el esquema de orden 2 para el ejemplo 1.

Volúmenes	Iteraciones	CPU 1 hebra	CPU 4 hebras	GTX 580 CUDA SP+DP	GTX 580 CUDA DP
4000	23	6.21	2.04	0.087	0.21
16000	45	49.28	17.23	0.73	1.62
64000	90	396.2	131.6	5.19	12.08
256000	179	3164.7	969.2	39.42	92.43

Tabla 4.5: Tiempos de ejecución en segundos e iteraciones realizadas con el esquema de orden 3 para el ejemplo 1.

Para evitar problemas de inestabilidad numérica, en las implementaciones CUDA en simple precisión de los esquemas de alto orden ha sido necesario realizar los calculos dados por (4.11) y (4.12) en doble precisión.

Al igual que con los esquemas de orden 1, todos los programas se ejecutarán en un Core i7 920 con 4 GB de memoria RAM, y la tarjeta gráfica que se usará es una GeForce GTX 580. El tiempo de simulación es de 0.1 segundos para ambos ejemplos. Se ha elegido el valor umbral 4096 para decidir entre llevar a cabo la obtención del mínimo Δt en CPU o en GPU. Se han asignado tamaños de 48 KB y 16 KB a la caché L1 y a la memoria compartida de GPU, respectivamente, para los kernels de reconstrucción y de procesamiento de aristas. El tamaño de un bloque de hebras para los kernels de reconstrucción, de procesamiento de aristas y de obtención del siguiente estado es de 64 hebras.

Las Tablas 4.4 y 4.5 muestran las iteraciones realizadas y los tiempos obtenidos en segundos para todas las implementaciones y tamaños de malla para los órdenes 2 y 3, respectivamente, en el ejemplo 1. Las Tablas 4.6 y 4.7 muestran la misma información para el ejemplo 2. En el caso del orden 3 no ha habido memoria GPU suficiente para la malla de 1024000 volúmenes. Las Figuras 4.14 y 4.15 muestran gráficamente todos los tiempos de ejecución obtenidos con la malla de 256000 volúmenes en los ejemplos 1 y 2, respectivamente. Las Figuras 4.16 y 4.17 representan gráficamente la ganancia obtenida con los programas CUDA en simple precisión para los órdenes 1, 2 y 3 respecto a las dos versiones CPU en los ejemplos 1 y 2,

Volúmenes	Iteraciones	CPU 1 hebra	CPU 4 hebras	GTX 580 CUDA SP+DP	GTX 580 CUDA DP
4000	20	2.08	0.57	0.020	0.042
16000	39	16.39	4.36	0.12	0.28
64000	78	131.7	34.29	0.90	2.09
256000	155	1047.0	271.4	6.88	16.48
1024000	310	8383.2	2185.6	53.79	132.3

Tabla 4.6: Tiempos de ejecución en segundos e iteraciones realizadas con el esquema de orden 2 para el ejemplo 2.

Volúmenes	Iteraciones	CPU 1 hebra	CPU 4 hebras	GTX 580 CUDA SP+DP	GTX 580 CUDA DP
4000	20	5.37	1.45	0.075	0.18
16000	39	42.70	11.23	0.63	1.40
64000	78	343.9	88.93	4.48	10.46
256000	155	2746.1	707.1	34.09	80.02

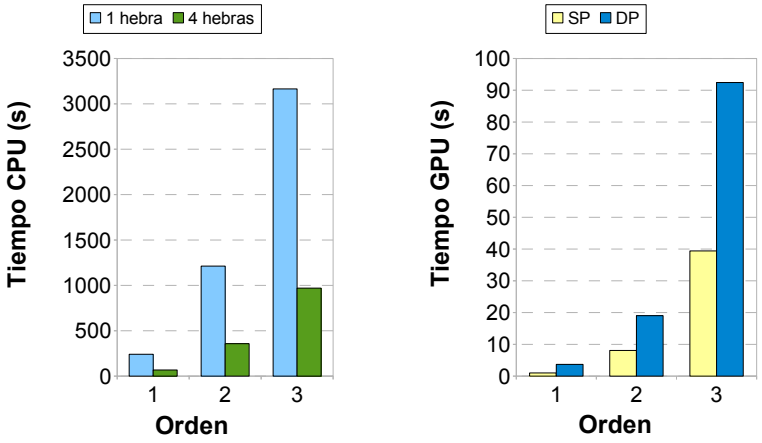
Tabla 4.7: Tiempos de ejecución en segundos e iteraciones realizadas con el esquema de orden 3 para el ejemplo 2.

respectivamente. En dichas Figuras, el esquema de orden 1 se corresponde con el esquema PVM-IFCP visto en el capítulo 3.

Podemos ver que, en ambos ejemplos, las implementaciones CUDA en simple precisión para los órdenes 2 y 3 han obtenido tiempos de GPU aproximadamente 8 y 40 veces más lentos, respectivamente, que los obtenidos en orden 1 para las mallas de mayor tamaño. La mayor carga computacional que se tiene al aumentar el orden del esquema numérico ha tenido mucha más influencia negativa en orden 3 que en orden 2 sobre los tiempos de GPU obtenidos.

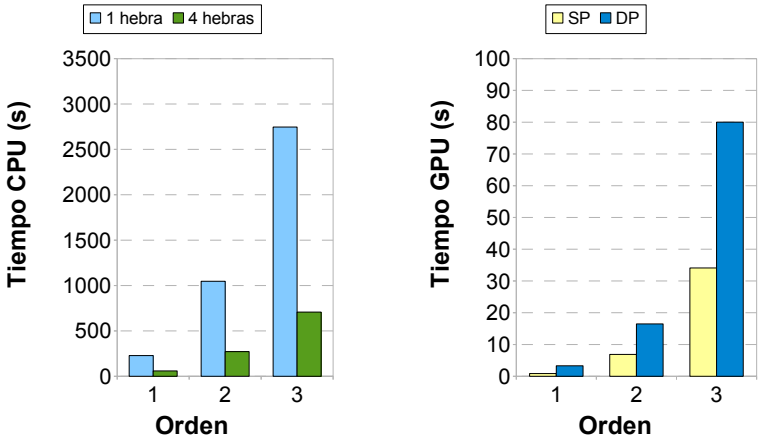
En todos los casos, las implementaciones CUDA de doble precisión han sido entre 2 y 2.5 veces más lentas que las versiones CUDA equivalentes en simple precisión. El hecho de que los cálculos dados por (4.11) y (4.12) se deban realizar en doble precisión ha influido en que esta ganancia no haya sido algo superior, como sucede en los esquemas de orden 1.

Respecto a la ganancia obtenida, la implementación CUDA en simple precisión para el esquema de orden 2 ha alcanzado una ganancia de aproximadamente 150 y 45 respecto a las versiones CPU de 1 y 4 hebras, respectivamente, en ambos ejemplos, mientras que en orden 3 estas ganancias han sido de 80 y 24, respectivamente. Al igual que en los esquemas de orden 1, las implementaciones CPU de 4 hebras han obtenido ganancias inferiores a 4 respecto a las versiones secuenciales en todos los casos, y próximas a 4 para las mallas de mayor tamaño.



(a) (b)

Figura 4.14: Tiempos de ejecución para todos los órdenes con la malla de 256000 volúmenes en el ejemplo 1: (a) CPU; (b) GPU.



(a) (b)

Figura 4.15: Tiempos de ejecución para todos los órdenes con la malla de 256000 volúmenes en el ejemplo 2: (a) CPU; (b) GPU.

La Figura 4.18 representa gráficamente el número de GFLOPS obtenidos con las implementaciones CUDA en simple precisión de los esquemas de orden 1, 2 y 3 para ambos ejemplos, donde el esquema de orden 1 se corresponde con el esquema PVM-IFCP visto en el capítulo 3. Vemos que, en los dos ejemplos, las implementaciones de orden 2 y 3 han alcanzado alrededor de 50 y 90 GFLOPS, respectivamente, frente a los menos de 40 GFLOPS

Kernel	% Tiempo de ejecución	
	Orden 2	Orden 3
Obtener integral y reconstrucciones	74.19	85.65
Procesar aristas	19.88	12.52
Obtener W_i y Δt_i	5.87	1.82
Obtener mínimo Δt	0.06	0.01

Tabla 4.8: Porcentaje del tiempo de ejecución empleado en cada kernel para la malla de 256000 volúmenes con las implementaciones CUDA de simple precisión de los esquemas de orden 2 y 3 para el ejemplo 1 usando una GeForce GTX 580.

Kernel	% Tiempo de ejecución	
	Orden 2	Orden 3
Obtener integral y reconstrucciones	75.34	86.51
Procesar aristas	18.60	11.64
Obtener W_i y Δt_i	6.00	1.84
Obtener mínimo Δt	0.06	0.01

Tabla 4.9: Porcentaje del tiempo de ejecución empleado en cada kernel para la malla de 256000 volúmenes con las implementaciones CUDA de simple precisión de los esquemas de orden 2 y 3 para el ejemplo 2 usando una GeForce GTX 580.

que se obtuvieron con la implementación de orden 1, lo que refleja el mejor aprovechamiento de la GPU cuanto mayor es el orden del esquema numérico debido a la mayor intensidad computacional. Sin embargo, como se ha visto a lo largo de esta sección, al aumentar el orden también aumentan los tiempos de ejecución.

Las Tablas 4.8 y 4.9 muestran el porcentaje del tiempo de ejecución empleado en cada kernel para la malla de 256000 volúmenes con las implementaciones CUDA de simple precisión para los esquemas de orden 2 y 3 con ambos ejemplos. En orden 2, el kernel de obtención de la integral y reconstrucciones para cada volumen ha consumido aproximadamente el 75 % del tiempo de ejecución, mientras que en orden 3 este porcentaje ha aumentado al 86 %.

Al igual que en el capítulo anterior, hemos comparado las soluciones obtenidas en CPU y en GPU calculando la norma L^1 de la diferencia entre las soluciones obtenidas con la versión CPU de 1 hebra y los programas CUDA para todos los tamaños de malla. Los órdenes de magnitud de la norma L^1 obtenidos con las implementaciones CUDA de simple precisión oscilan entre 10^{-4} y 10^{-6} en orden 2, y 10^{-2} y 10^{-4} en orden 3. Por su parte, los órdenes de magnitud de la norma L^1 obtenidos con las implementaciones

CUDA de doble precisión oscilan entre 10^{-13} y 10^{-14} en orden 2, y 10^{-12} y 10^{-14} en orden 3. Esto refleja, por un lado, la distinta precisión alcanzada en GPU usando simple y doble precisión numérica, y por otro lado, la mayor precisión numérica obtenida con las implementaciones de orden 2 respecto a la conseguida en orden 3.

4.8. Conclusiones

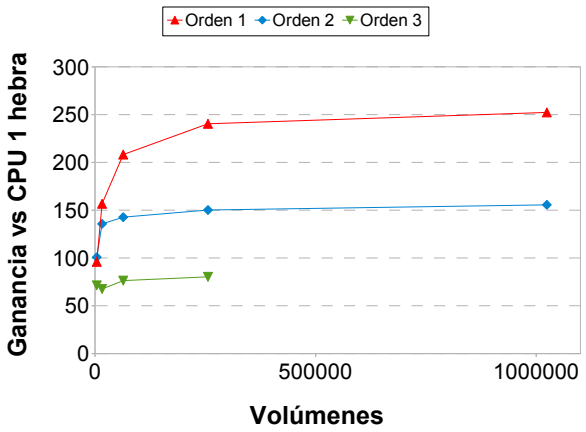
En este capítulo se han adaptado e implementado en GPU dos esquemas numéricos de alto orden para el sistema de aguas someras bicapa en mallas triangulares. Concretamente, hemos estudiado la adaptación a GPU del operador de reconstrucción introducido en [DK07] y lo hemos aplicado, junto con el esquema PVM-IFCP descrito en el capítulo 3, para obtener soluciones de orden 2 y 3.

Hemos visto que, al aumentar el orden del esquema numérico, se obtienen resultados cada vez más precisos. Sin embargo, el método de reconstrucción empleado es muy costoso computacionalmente, lo que hace que los tiempos de ejecución aumenten notablemente al incrementar el orden. En particular, los tiempos de GPU obtenidos con implementaciones en simple precisión de los esquemas de orden 2 y 3 han sido aproximadamente 8 y 40 veces más lentos, respectivamente, que los obtenidos con el esquema de orden 1. En orden 2 se han alcanzado ganancias de aproximadamente 150 y 45 en una GeForce GTX 580 respecto a implementaciones CPU de 1 y 4 hebras, respectivamente, ejecutadas en un Core i7 920. En orden 3 estas ganancias han sido de 80 y 24, respectivamente. En ambos casos, estas ganancias han sido notablemente inferiores a las que se obtuvieron en orden 1.

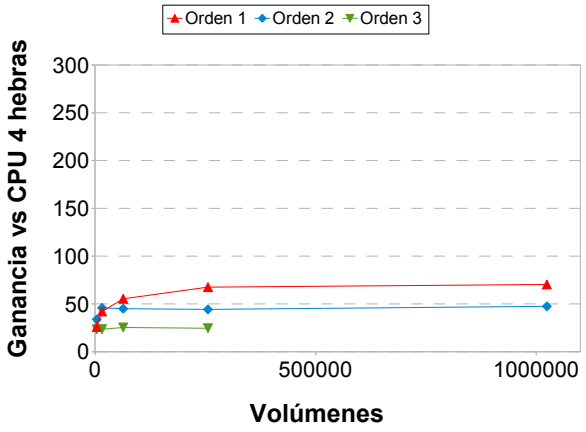
También hemos comprobado que, cuanto mayor es el orden del esquema numérico, mejor se aprovecha la tarjeta gráfica en términos de rendimiento absoluto medido en GFLOPS debido a la carga computacional cada vez más elevada, obteniéndose aproximadamente 50 y 90 GFLOPS con los esquemas de orden 2 y 3, respectivamente, en una GeForce GTX 580, frente a los menos de 40 GFLOPS que se alcanzaron con el esquema de orden 1.

Igualmente, hemos medido el porcentaje de tiempo de ejecución empleado por cada etapa del proceso en las implementaciones CUDA en simple precisión de los esquemas de orden 2 y 3 usando una malla de 256000 volúmenes. Para ambos órdenes, la etapa más costosa ha sido el cálculo de la integral y las reconstrucciones para cada volumen, empleando aproximadamente el 75 % del tiempo de ejecución en orden 2, y el 86 % en orden 3.

Finalmente, hemos visto que, cuanto mayor es el orden del esquema numérico, más precisión se pierde con respecto a la obtenida con la implementación CPU equivalente en doble precisión, siendo esta pérdida más acusada en la implementación de orden 3 en simple precisión.

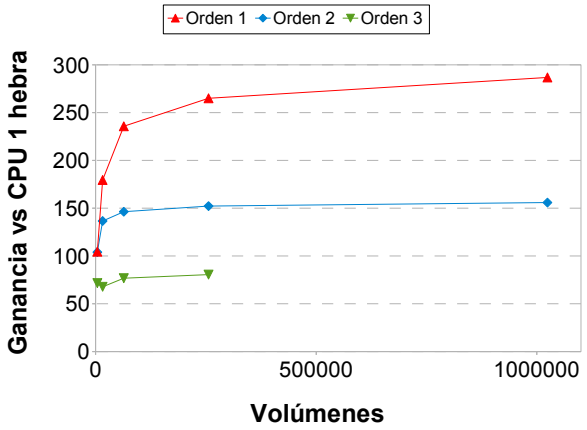


(a)

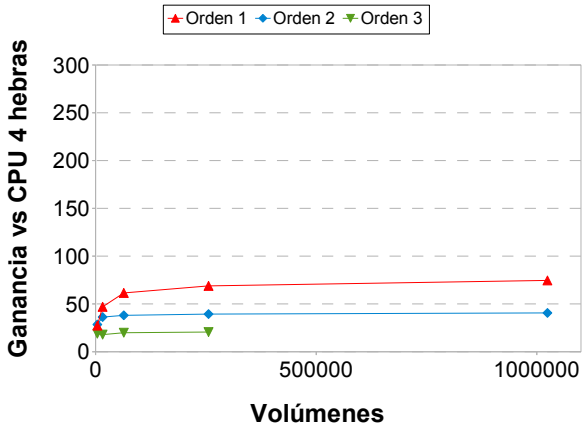


(b)

Figura 4.16: Ganancia obtenida con las implementaciones CUDA de orden 1, 2 y 3 en simple precisión en una GeForce GTX 580 para el ejemplo 1: (a) Respecto a la versión CPU de 1 hebra; (b) Respecto a la versión CPU de 4 hebras.

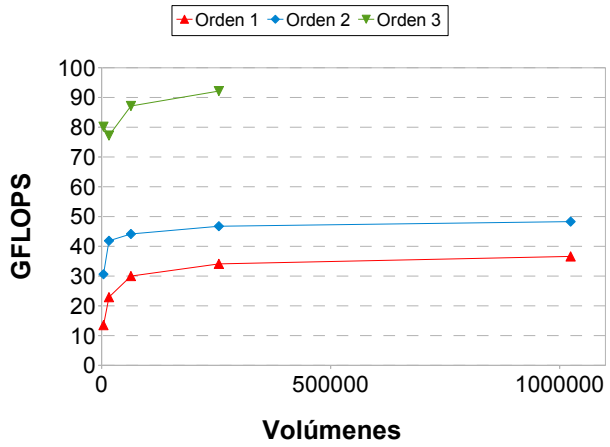


(a)

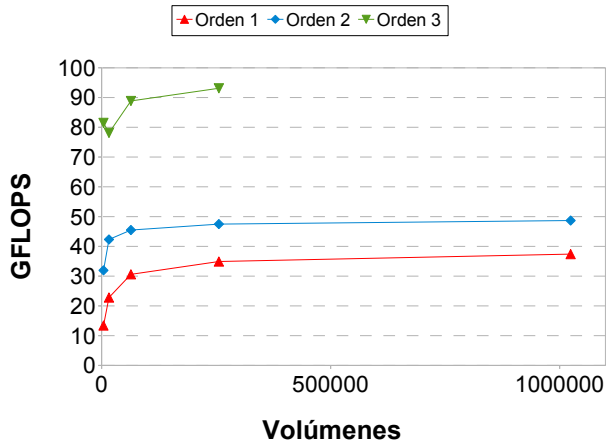


(b)

Figura 4.17: Ganancia obtenida con las implementaciones CUDA de orden 1, 2 y 3 en simple precisión en una GeForce GTX 580 para el ejemplo 2: (a) Respecto a la versión CPU de 1 hebra; (b) Respecto a la versión CPU de 4 hebras.



(a)



(b)

Figura 4.18: GFLOPS obtenidos con las implementaciones CUDA en simple precisión para los órdenes 1, 2 y 3 en una GeForce GTX 580: (a) Ejemplo 1; (b) Ejemplo 2.

Capítulo 5

Simulación en Clusters de GPUs

Resumen: En este capítulo se describen dos implementaciones multi-GPU del esquema PVM-IFCP visto en el capítulo 3 para la resolución numerica del sistema de aguas someras bicapa usando mallas triangulares. Se efectúan varios experimentos y se analizan los resultados obtenidos en términos de eficiencia computacional y escalabilidad. El contenido de este capítulo ha sido parcialmente publicado en [dlAMCFN11, dlAMCFN12].

5.1. Implementación del Esquema PVM-IFCP para el Sistema de Aguas Someras Bicapa en un Cluster de GPUs

En esta sección se presenta una extensión multi-GPU de la implementación CUDA del esquema PVM-IFCP para el sistema de aguas someras bicapa descrita en la sección 3.5.2. Básicamente, la malla triangular se divide en varias submallas conexas y disjuntas utilizando una herramienta de particionamiento de mallas como las citadas en la sección 2.4 (en la Figura 5.1a se muestra un ejemplo de malla triangular dividida en cuatro submallas). A continuación, cada submalla se asigna a un proceso de CPU que, a su vez, utiliza una GPU para llevar a cabo los cómputos asociados a su submalla. Para la comunicación entre procesos se usa MPI [MPI]. De aquí en adelante utilizaremos el término *submalla* para referirnos indistintamente a la submalla o al proceso de CPU asociado a ella.

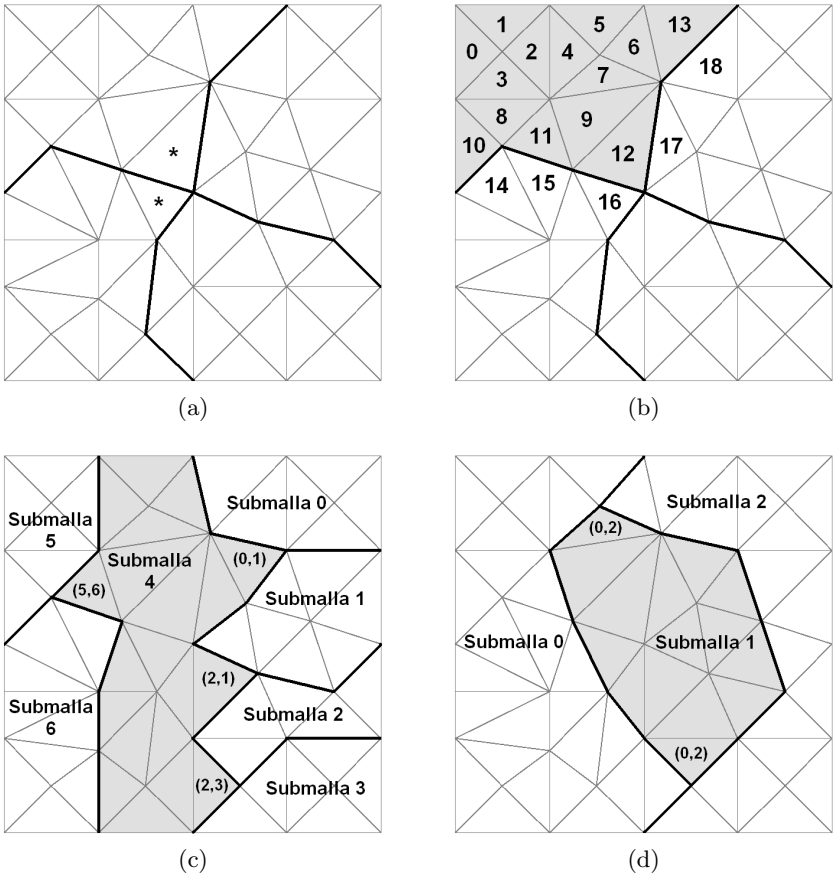


Figura 5.1: Submallas en una malla triangular: (a) Tipos de submallas; (b) Indexación de volúmenes; (c) Secuencia de pares; (d) Caso no permitido.

5.1.1. Definiciones Previas

Introducimos en este punto algunas definiciones de términos relacionados con particionamiento de mallas que usaremos a lo largo de este capítulo.

Arista de comunicación Arista cuyos volúmenes adyacentes pertenecen a submallas distintas. En la Figura 5.1a las aristas de comunicación se denotan en negrita.

Volumen de comunicación Volumen que tiene, al menos, una arista de comunicación. En la Figura 5.1b los volúmenes 10, 11, 12 y 13 son los volúmenes de comunicación de la submalla de color gris.

Submalla adyacente Una submalla es adyacente a otra si ambas comparten, al menos, una arista de comunicación. En la Figura 5.1c la submalla 5 tiene dos submallas adyacentes: la 4 y la 6.

Nótese que cada submalla, para poder procesar sus aristas de comunicación, también necesita almacenar los datos de los volúmenes de comunicación correspondientes de sus submallas adyacentes. Así, en la Figura 5.1b, para que la submalla de color gris pueda procesar sus aristas de comunicación, necesita los datos de los volúmenes del 14 al 18, que pertenecen a otras submallas. Por tanto, en cada iteración del proceso, cada submalla deberá enviar sus volúmenes de comunicación correspondientes a sus submallas adyacentes.

A continuación describimos cómo se crean y almacenan en GPU los datos asociados a cada submalla.

5.1.2. Creación de los Datos de las Submallas

Para la creación de los datos, consideraremos dos tipos de submallas:

- **Tipo 1**: Aquellas que tienen volúmenes que deben ser enviados a dos submallas distintas.
- **Tipo 2**: Aquellas que no cumplen la condición anterior.

Por ejemplo, en la Figura 5.1a las dos submallas de la izquierda son de tipo 1, y las dos submallas de la derecha son de tipo 2. En dicha figura, los volúmenes que deben ser enviados a dos submallas se denotan con un asterisco. Los dos tipos de submalla considerados utilizan las mismas estructuras de datos en GPU para almacenar los datos de sus volúmenes y aristas, que se describieron en la sección 3.5.2, con la diferencia que ahora los dos arrays que almacenan los estados de los volúmenes se dividen en tres bloques ordenados consecutivamente:

- **Bloque 1**: En primer lugar, se almacenan los volúmenes de la submalla que no son de comunicación.
- **Bloque 2**: En segundo lugar se almacenan los volúmenes de comunicación de la submalla. A su vez, estos volúmenes se ordenan en grupos de volúmenes que son adyacentes a una submalla concreta.
- **Bloque 3**: Finalmente, se almacenan los volúmenes de comunicación de otras submallas que son adyacentes a nuestra submalla. A su vez, estos volúmenes se ordenan en grupos de volúmenes que pertenecen a una submalla concreta.

La Figura 5.1b muestra una posible indexación de volúmenes en la submalla de color gris, donde el bloque 1 está formado por los volúmenes del 0 al 9, el bloque 2 por los volúmenes del 10 al 13, y el bloque 3 por los volúmenes del 14 al 18.

5.1.2.1. Creación de Datos en Submallas de Tipo 1

La creación de los datos en submallas de tipo 2 es sencilla. Sin embargo, para submallas de tipo 1 es más complicado, ya que para este tipo de submallas debemos ordenar sus volúmenes de comunicación con objeto de solapar los envíos que se realizan a las submallas adyacentes. Para ello, todos los volúmenes de comunicación que sean adyacentes a una submalla concreta deben aparecer consecutivamente en los arrays, teniendo en cuenta los volúmenes que sean adyacentes a dos submallas distintas. Por ejemplo, en la Figura 5.1b los volúmenes del 10 al 12 deben enviarse a la submalla inferior, mientras que los volúmenes 12 y 13 deben enviarse a la submalla derecha, solapando de este modo los envíos.

Para llevar a cabo esta ordenación, en primer lugar construimos una lista de volúmenes de comunicación para cada submalla adyacente. Por ejemplo, en la Figura 5.1b tendríamos dos listas: [10, 11, 12] y [12, 13].

A continuación, para cada volumen de comunicación de la submalla que debe enviarse a dos procesos MPI, construimos un par (p_1, p_2), significando que el volumen debe enviarse a los procesos p_1 y p_2. La Figura 5.1c muestra un ejemplo considerando la submalla 4, donde se especifican todos los pares. Una vez se ha construido esta secuencia de pares, la reordenamos (junto con sus elementos, si es necesario), de forma que obtengamos una lista de procesos consecutivos. Por ejemplo, en la Figura 5.1c, los pares se reordenan obteniendo: (0,1), (1,2), (2,3) y (5,6). Esto nos da la lista consecutiva de procesos 0, 1, 2, 3, 5 y 6.

Ahora recorremos la secuencia de pares ordenada y vamos procesando cada par. Concretamente, para cada elemento (p_1, p_2) de la secuencia de pares, se realizan las siguientes acciones: en la lista de volúmenes adyacentes a la submalla p_1, ponemos el volumen compartido con la submalla p_2 al final. Asimismo, en la lista de volúmenes adyacentes a la submalla p_2, ponemos el volumen compartido con la submalla p_1 al principio.

Aplicando este algoritmo al ejemplo que estamos considerando, tenemos que la secuencia de pares anterior se procesa del siguiente modo:

1. *Procesamiento del elemento* (0,1): En la lista de volúmenes adyacentes a la submalla 0, ponemos el volumen compartido con la submalla 1 al final. Asimismo, en la lista de volúmenes adyacentes a la submalla 1, ponemos el volumen compartido con la submalla 0 al principio.

2. *Procesamiento del elemento* (1,2): En la lista de volúmenes adyacentes a la submalla 1, ponemos el volumen compartido con la submalla 2 al final. Asimismo, en la lista de volúmenes adyacentes a la submalla 2, ponemos el volumen compartido con la submalla 1 al principio.

3. Continuamos procesando del mismo modo los restantes elementos de la secuencia de pares.

Finalmente, unimos todas las listas de volúmenes de comunicación adecuadamente para obtener el bloque final de volúmenes de comunicación. En el ejemplo de la Figura 5.1b, este bloque quedaría: [10, 11, 12, 13].

Nótese que una submalla debe saber cómo están ordenados los volúmenes de comunicación que recibe de otra submalla. Por este motivo, durante el proceso de creación de los datos, todas las submallas envían a sus submallas adyacentes esta información. Concretamente, dicha información consiste en los intercambios de posiciones de las listas que ha realizado la submalla durante la ordenación de sus volúmenes de comunicación.

Nótese también que este algoritmo no funciona cuando en la secuencia de pares se forma un ciclo con sus elementos. La Figura 5.1d muestra un ejemplo de esta situación, donde una submalla tiene dos volúmenes que deben enviarse a las mismas submallas. Sin embargo, siempre se puede realizar una descomposición de la malla donde esto no ocurra. Tampoco funciona cuando una submalla está formada por un único volumen, pero este caso nunca se va a dar en un problema real.

5.1.3. Código Multi-GPU

Se han implementado dos algoritmos multi-GPU: uno que realiza envíos y recepciones MPI bloqueantes, y otro que solapa las comunicaciones MPI con procesamiento de kernels y transferencias de memoria entre CPU y GPU.

El Algoritmo 1 muestra los pasos generales de la implementacion sin solapamiento. En dicho algoritmo, VC denota los volúmenes de comunicación. Suponemos que cada submalla tiene n submallas adyacentes numeradas de 1 a n. En lo sucesivo, hablaremos de líneas del algoritmo como sinónimo de las sentencias que se ejecutan en dichas líneas. En primer lugar, las líneas 2 y 3 obtienen el Δt local de cada submalla y el Δt inicial global de todas las submallas aplicando una reducción MPI, respectivamente. En las líneas 5–8 enviamos a cada submalla adyacente los volúmenes de comunicación adyacentes a ella. A continuación, en las líneas 9–12 recibimos de las mismas submallas sus volúmenes de comunicación adyacentes a nuestra submalla. Hemos usado la función `MPI_Bsend`, que permite redimensionar el buffer de envío, para evitar situaciones de interbloqueo al trabajar con mallas de gran tamaño. Las líneas 13–14 copian los volúmenes de comunicación recibidos a memoria GPU. La línea 15 realiza el procesamiento de todas las aristas de la submalla. En la línea 19 se obtiene el nuevo Δt global de todas las submallas mediante reducción MPI, y las líneas 20–21 copian de GPU a CPU los nuevos estados de los volúmenes de comunicación de nuestra submalla.

El Algoritmo 2 muestra los pasos generales de la implementacion con solapamiento. Las líneas 5–8 (la recepción de los volúmenes de comunicación de las submallas adyacentes) pueden solaparse con las líneas 9–10 (la copia de los nuevos estados de los volúmenes de comunicación de GPU a CPU) y

Algoritmo 1 Algoritmo multi-GPU sin solapamiento

```
1: n ← Número de submallas adyacentes
2: Δt ← calcularDeltaTInicial(...)
3: MPI_Allreduce(Δt, mín Δt, ...)
4: mientras (t < t_fin) hacer
5:   para i = 1 hasta n hacer
6:     MPI_Bsend(Capa 1 de los VC a la submalla adyacente i)
7:     MPI_Bsend(Capa 2 de los VC a la submalla adyacente i)
8:   fin para
9:   para i = 1 hasta n hacer
10:    MPI_Recv(Capa 1 de los VC de la submalla adyacente i)
11:    MPI_Recv(Capa 2 de los VC de la submalla adyacente i)
12:  fin para
13:  CudaMemcpy(Capa 1 de los VC de CPU a GPU)
14:  CudaMemcpy(Capa 2 de los VC de CPU a GPU)
15:  procesarAristas<<<malla, bloque>>>(...)
16:  obtenerEstadoYDeltaTVolumenes<<<malla, bloque>>>(...)
17:  t ← t + mín Δt
18:  Δt ← obtenerMinimoDeltaT(...)
19:  MPI_Allreduce(Δt, mín Δt, ...)
20:  CudaMemcpy(Capa 1 de los VC de GPU a CPU)
21:  CudaMemcpy(Capa 2 de los VC de GPU a CPU)
22: fin mientras
```

con la línea 15 (el procesamiento de las aristas que no son de comunicación, ya que dichas aristas no necesitan datos externos para ser procesadas). Las líneas 11–14 (el envío a cada submalla adyacente de los volúmenes de comunicación adyacentes a ella) también pueden solaparse con la línea 15. En las líneas 16–17 esperamos a que lleguen los volúmenes de comunicación de las submallas adyacentes, y en las líneas 18–19 los copiamos a memoria GPU. En la línea 20 sólo se procesan las aristas de comunicación.

Con objeto de reducir el número de envíos, se ha probado otra implementación para ambos algoritmos, consistente en empaquetar los dos mensajes correspondientes a cada capa en uno solo mediante `MPI_Pack`, realizar un único envío y desempaquetarlo en el destino con `MPI_Unpack`. Sin embargo, se han obtenido peores tiempos de ejecución que con los algoritmos propuestos.

5.2. Resultados Experimentales

En esta sección ejecutaremos las dos implementaciones realizadas de los algoritmos multi-GPU mostrados en la sección 5.1.3 para los dos ejemplos descritos en la sección 3.6.1 y analizaremos los resultados obtenidos. En particular, estudiaremos la escalabilidad fuerte y débil que se obtiene. Para medir la escalabilidad fuerte, se aumenta el número de GPUs manteniendo

Algoritmo 2 Algoritmo multi-GPU con solapamiento

```
1: n ← Número de submallas adyacentes
2: Δt ← calcularDeltaTInicial(...)
3: MPI_Allreduce(Δt, mín Δt, ...)
4: mientras (t < t_fin) hacer
5:   para i = 1 hasta n hacer
6:     MPI_Irecv(Capa 1 de los VC de la submalla adyacente i)
7:     MPI_Irecv(Capa 2 de los VC de la submalla adyacente i)
8:   fin para
9:   CudaMemcpy(Capa 1 de los VC de GPU a CPU)
10:  CudaMemcpy(Capa 2 de los VC de GPU a CPU)
11:  para i = 1 hasta n hacer
12:    MPI_Isend(Capa 1 de los VC a la submalla adyacente i)
13:    MPI_Isend(Capa 2 de los VC a la submalla adyacente i)
14:  fin para
15:  procesarAristas<<<malla, bloque>>>(Aristas de no comunicación)
16:  MPI_Waitall (Capa 1 de los VC de las submallas adyacentes)
17:  MPI_Waitall (Capa 2 de los VC de las submallas adyacentes)
18:  CudaMemcpy(Capa 1 de los VC de CPU a GPU)
19:  CudaMemcpy(Capa 2 de los VC de CPU a GPU)
20:  procesarAristas<<<malla, bloque>>>(Aristas de comunicación)
21:  obtenerEstadoYDeltaTVolumenes<<<malla, bloque>>>(...)
22:  t ← t + mín Δt
23:  Δt ← obtenerMinimoDeltaT(...)
24:  MPI_Allreduce(Δt, mín Δt, ...)
25: fin mientras
```

constante el tamaño del problema (en nuestro caso, el número de volúmenes), mientras que para medir la escalabilidad débil, el tamaño del problema se aumenta de forma proporcional al número de GPUs (esto es, el tamaño de problema por GPU se mantiene constante).

Hemos empleado la utilidad Chaco [HL94] para descomponer una malla en distintas submallas del mismo tamaño, y la implementación OpenMPI [GFB+04]. Todos los programas se ejecutarán en un cluster formado por dos servidores Intel Xeon E5620 conectados a través de un switch Ethernet de un Gigabit. Cada servidor tiene 8 GB de memoria RAM y dos tarjetas GeForce GTX 480. Para una GPU se usará la implementación CUDA del esquema PVM-IFCP que se describió en el capítulo 3. En las ejecuciones con 2 GPUs consideraremos una GPU por cada nodo. El tiempo de simulación física en este caso es de 1 segundo para ambos ejemplos.

Las Tablas 5.1 y 5.2 muestran los tiempos de ejecución en segundos para todas las mallas usando 1, 2 y 4 GPUs en los ejemplos 1 y 2, respectivamente. La Tabla 5.3 especifica el número de iteraciones que se han realizado para ambos ejemplos. La Figura 5.2 representa las ganancias obtenidas con las implementaciones multi-GPU respecto a una GTX 480 en ambos ejemplos.

Volúmenes	GTX 480	2 GTX 480		4 GTX 480	
		Sin solap.	Con solap.	Sin solap.	Con solap.
4000	0.050	0.078	0.075	0.076	0.083
16000	0.27	0.26	0.24	0.19	0.19
64000	1.59	1.15	1.02	0.79	0.73
256000	11.34	6.97	6.30	6.34	3.64
1024000	85.27	47.20	44.39	45.14	23.54
2080560	316.6	180.1	163.3	167.3	85.86

Tabla 5.1: Tiempos de ejecución en segundos de las implementaciones multi-GPU para el ejemplo 1.

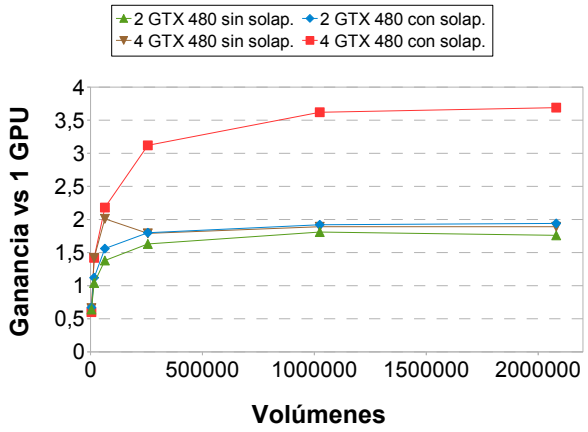
Volúmenes	GTX 480	2 GTX 480		4 GTX 480	
		Sin solap.	Con solap.	Sin solap.	Con solap.
4000	0.043	0.067	0.065	0.066	0.072
16000	0.22	0.22	0.20	0.16	0.16
64000	1.33	0.94	0.85	0.65	0.61
256000	9.45	5.85	5.27	5.26	3.02
1024000	71.43	39.55	37.20	37.76	19.73
2080560	263.1	148.5	135.7	138.4	72.10

Tabla 5.2: Tiempos de ejecución en segundos de las implementaciones multi-GPU para el ejemplo 2.

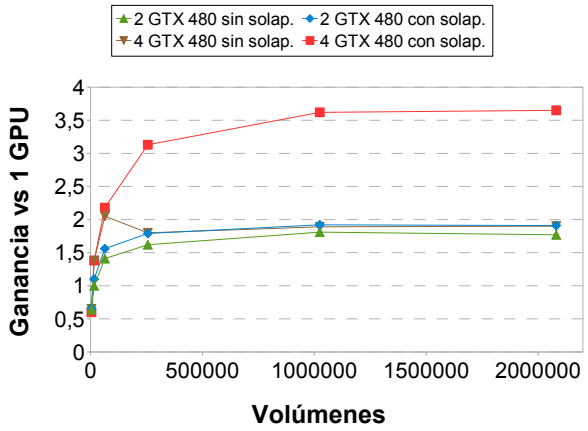
Como era de esperar, la implementación multi-GPU con solapamiento ha mejorado los tiempos de ejecución de la versión sin solapamiento a pesar de tener un kernel adicional. En ambos ejemplos, esta mejora ha sido de casi el 10 % para 2 GPUs y del 50 % para 4 GPUs en las mallas de mayor tamaño.

Con el objetivo de reflejar más claramente los beneficios de la implementación con solapamiento, las Tablas 5.4 y 5.5 muestran los tiempos de ejecución empleados en distintos pasos del proceso usando 4 GPUs con las dos mallas más grandes para los ejemplos 1 y 2, respectivamente. Concretamente, se muestran los tiempos empleados en envíos, recepciones y esperas de mensajes MPI (T_{ComMPI}), el procesamiento de aristas ($T_{\text{ProcesAristas}}$) y la reducción de tiempo conseguida con el Algoritmo 2 con respecto al Algoritmo 1. Como se puede ver, en todos los casos esta reducción de tiempo ha sido cercana a T_{ComMPI} para el Algoritmo 1 (líneas 5–12). En el Algoritmo 2, este coste de las comunicaciones MPI se enmascara en gran parte con el procesamiento de las aristas que no son de comunicación.

La Figura 5.3 está asociada a la escalabilidad fuerte de la versión con solapamiento y representa la evolución de la ganancia (respecto a la versión para una GPU) conforme aumentamos el número de GPUs para todos los



(a)



(b)

Figura 5.2: Ganancia obtenida con las implementaciones multi-GPU respecto a una GTX 480: (a) Ejemplo 1; (b) Ejemplo 2.

Volúmenes	Ejemplo 1	Ejemplo 2
4000	227	195
16000	455	391
64000	910	781
256000	1822	1562
1024000	3647	3124
2080560	6750	5726

Tabla 5.3: Iteraciones realizadas para todos los tamaños de malla en ambos ejemplos.

Volúmenes	T_{ComMPI}		$T_{\text{ProcesAristas}}$		$T_{\text{Alg1}} - T_{\text{Alg2}}$
	Alg1	Alg2	Alg1	Alg2	
1024000	22.81	0.80	16.71	16.94	21.60
2080560	86.12	13.53	62.26	63.48	81.44

Tabla 5.4: Tiempos de ejecución en segundos de distintos pasos en 4 GPUs para el ejemplo 1.

Volúmenes	T_{ComMPI}		$T_{\text{ProcesAristas}}$		$T_{\text{Alg1}} - T_{\text{Alg2}}$
	Alg1	Alg2	Alg1	Alg2	
1024000	19.23	0.69	13.91	14.29	18.03
2080560	70.25	12.93	51.21	52.36	66.30

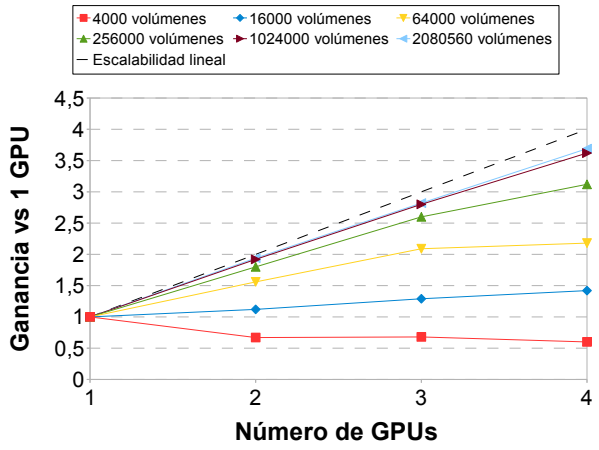
Tabla 5.5: Tiempos de ejecución en segundos de distintos pasos en 4 GPUs para el ejemplo 2.

tamaños de malla en ambos ejemplos. Como puede verse, la escalabilidad es más cercana a la lineal cuanto mayor es el tamaño de la malla. Concretamente, para la malla de 2080560 volúmenes hemos obtenido unas ganancias de aproximadamente 1.9 y 3.7 usando 2 y 4 GPUs, respectivamente. La escalabilidad no es buena para mallas pequeñas debido a que el tamaño pequeño de las submallas impide obtener el máximo rendimiento en cada GPU, y a que las comunicaciones MPI tienen un mayor peso porcentual en el tiempo de ejecución que con mallas más grandes. Para las mallas de más de un millón de volúmenes, se ha alcanzado una escalabilidad fuerte cercana a la lineal usando hasta 4 GPUs.

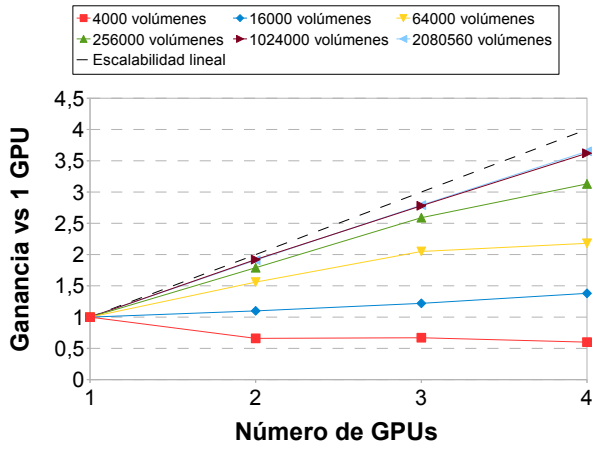
La Figura 5.4, por su parte, está asociada a la escalabilidad débil de la versión con solapamiento y representa la evolución de la eficiencia[1] al aumentar el número de GPUs manteniendo un tamaño de malla constante por GPU e igual a 256000 volúmenes. Para ello, se ha creado otra malla triangular de 512084 volúmenes. De este modo, hemos ejecutado la implementación multi-GPU con solapamiento con 256000, 512084 y 1024000 volúmenes usando 1, 2 y 4 GPUs, respectivamente, en los dos ejemplos. Podemos ver que la eficiencia obtenida en ambos ejemplos ha sido muy próxima a 1. Concretamente, 0.94 y 0.91 usando 2 y 4 GPUs, respectivamente.

Un modo de aumentar aún más las escalabilidades fuerte y débil sería usar una red de mayor velocidad, como una InfiniBand [Inf]. Otra forma sería reducir el número de comunicaciones MPI mediante un solapamiento de las submallas, de forma que cada submalla pudiera realizar varias iteraciones

[1]Definimos la eficiencia como el resultado de dividir la ganancia obtenida respecto a una GPU entre el número de GPUs utilizadas.



(a)



(b)

Figura 5.3: Ganancia obtenida con la implementación multi-GPU con solapamiento respecto a una GTX 480: (a) Ejemplo 1; (b) Ejemplo 2.

antes de intercambiar sus volúmenes de comunicación con sus submallas adyacentes. En la literatura se han publicado resultados satisfactorios de esta última técnica en mallas estructuradas, tanto en CPUs [DH01] como en GPUs [BS10, MLF+12].

5.3. Conclusiones

En este capítulo se han implementado y analizado dos algoritmos multi-GPU distribuidos para el esquema PVM-IFCP visto en el capítulo 3 para la resolución numerica del sistema de aguas someras bicapa usando ma-

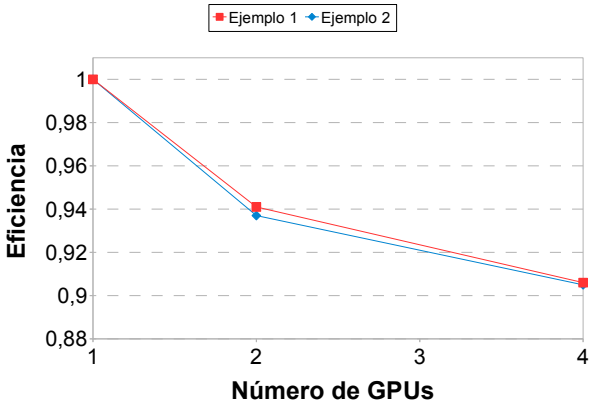


Figura 5.4: Eficiencia obtenida con la implementación multi-GPU con solapamiento asignando un tamaño de malla constante por GPU e igual a 256000 volúmenes.

llas triangulares. Ambas implementaciones se han realizado usando MPI y CUDA. La implementación que solapa comunicaciones MPI con procesamiento de kernels y transferencias de memoria entre CPU y GPU ha alcanzado una escalabilidad fuerte cercana a la lineal para mallas de más de un millón de volúmenes usando hasta 4 GPUs, y una escalabilidad débil próxima a 1 también usando hasta 4 GPUs. El objetivo de dicho solapamiento es reducir la sobrecarga de las comunicaciones MPI, y ha demostrado ser un factor muy relevante para mejorar la eficiencia de este resolvedor.

Capítulo 6

Simulación de Tsunamis en GPUs

Resumen: En este capítulo se adapta e implementa en GPU una modificación del esquema PVM-IFCP descrito en el capítulo 3 para la simulación de tsunamis generados por avalanchas submarinas y subaéreas en mallas estructuradas y triangulares. Se presenta además la herramienta de visualización que se ha implementado en OpenGL. Finalmente, se realizan varios experimentos y se analizan los resultados obtenidos. El contenido de este capítulo ha sido parcialmente publicado en [GCdlA+11, GdlAC+11, GCSdlA12, SCGdlA12, GdlAC+12, dlAMCG12, MFG+12].

6.1. Introducción

La historia de la humanidad está salpicada de numerosos episodios catastróficos que han ocasionado cambios importantes en las ubicaciones de los asentamientos urbanos, así como modificaciones en los usos del territorio. Gran parte de estos episodios tienen su origen en los movimientos sísmicos que suceden en la corteza terrestre y en los procesos que desencadenan. En algunas ocasiones son mayores los desastres originados por los fenómenos que desencadena un terremoto, que los efectos del propio movimiento sísmico.

Un buen ejemplo de esto último se dio en la erupción del volcán Vesubio el 24 de agosto del año 79 d.C., que sepultó las ciudades de Pompeya y Herculano, cubriéndolas de cenizas volcánicas abrasadoras. Sin embargo, lo que resultó letal para los habitantes que huían del desastre fue el tsunami que se originó como consecuencia de los seísmos que acompañaron la expulsión de lavas y cenizas volcánicas. Según narra Plinio el Viejo, las olas que invadie-

ron las costas fueron tremendamente destructivas e hicieron naufragar las embarcaciones que evacuaban a los habitantes que habían logrado escapar de los efectos del volcán.

Más recientemente, encontramos los terremotos de Sumatra (26 de diciembre de 2004, con magnitud 9.1), Chile (27 de febrero de 2010, con magnitud 8.8) y Japón (11 de marzo de 2011, con magnitud 9.0). Todos estos terremotos vinieron acompañados de devastadores tsunamis que aumentaron aún más los daños ocasionados por los seísmos. Particularmente trágicas fueron las consecuencias del terremoto de Sumatra y su posterior tsunami en cuanto a pérdida de vidas humanas y zonas afectadas, con más de 180000 muertos y 40000 desaparecidos en 14 países [USG].

La palabra *tsunami* deriva de un vocablo japonés que significa "ola de puerto" (*tsu*: puerto, y *nami*: ola). Un tsunami es una serie de olas provocadas por el desplazamiento de un gran volumen de agua, típicamente en un océano o un gran lago. Estas olas pueden alcanzar dimensiones muy variadas (desde centímetros a decenas de metros) y pueden originarse por diversas causas, como terremotos, erupciones volcánicas, avalanchas o, raras veces, por explosiones submarinas (tests nucleares) o impactos de meteorito.

6.1.1. Características de los Tsunamis

Existen diferencias sustanciales entre las olas de los tsunamis (de traslación) y las que se producen habitualmente en la superficie de los mares y océanos (de oscilación), tanto en el mecanismo que las genera como en su propagación. Los tsunamis se producen principalmente por la dinámica de la corteza terrestre, de forma que un fuerte desplazamiento vertical en la superficie del fondo marino provoca un colapso de la superficie marina que genera una onda de traslación. Esta onda se propaga en toda la columna de agua, desde la superficie del mar hasta el fondo, se desplaza a gran velocidad (entre 300 y 500 km/h, pudiendo alcanzar velocidades extremas de 1000 km/h) y tiene un fuerte poder destructivo cuando impacta con la costa. Por su parte, el oleaje que se produce en la superficie del mar está ocasionado por la intervención de vientos, corrientes y mareas.

En mar abierto, los tsunamis tienen una baja amplitud (altura de la ola), que no suele superar los 2 metros, y una gran longitud de onda (a menudo cientos de kilómetros), motivo por el cual suelen pasar desapercibidos para los barcos que navegan en alta mar. Cuando los tsunamis se acercan a las costas alcanzando aguas poco profundas, la altura de las olas aumenta y su velocidad disminuye (ver Figura 6.1). Un gran tsunami puede estar formado por múltiples olas llegando en un periodo de varias horas, con un tiempo significativo entre cada una de ellas.

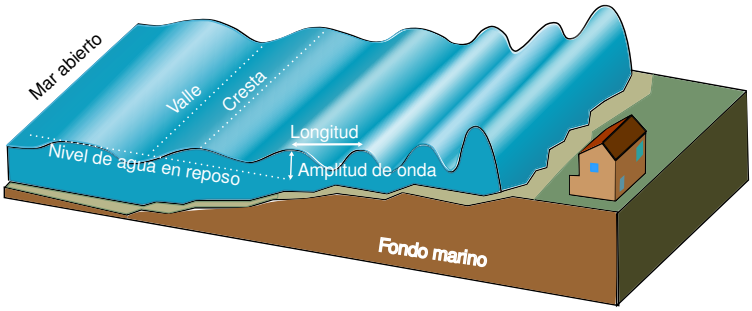


Figura 6.1: Olas producidas por un tsunami.

6.1.2. Utilización de GPUs

La necesidad del ser humano de adaptarse al medio en el que habita, en ocasiones hostil, le ha obligado a tener que aprender de los desastres naturales ocurridos en el pasado para tratar de mitigar sus efectos devastadores. Es precisamente en este punto donde entran en juego los modelos matemáticos. La simulación numérica mediante ordenadores se ha convertido en una herramienta de predicción muy potente y precisa. Sin embargo, la resolución numérica de estos modelos sobre escenarios reales requiere un poder de cómputo muy elevado debido a las grandes dimensiones espaciales y temporales de estos problemas. Como se vio en el capítulo 2, las GPUs han demostrado ser un potente recurso para acelerar simulaciones computacionalmente intensivas.

En este capítulo se describen dos implementaciones en GPU (para mallas estructuradas y triangulares, respectivamente) de un esquema numérico de primer orden basado en volúmenes finitos para la simulación de tsunamis generados mediante avalanchas submarinas y subaéreas (esto es, situadas fuera del agua) usando el entorno de programación CUDA.

6.2. Sistema de Aguas Someras Bicapa de Tipo Savage-Hutter

En esta sección describiremos el sistema de ecuaciones en derivadas parciales que usaremos para la simulación de tsunamis generados mediante avalanchas. Este modelo fue inicialmente propuesto en [FNBB$^+$08]. Nosotros usaremos una versión simplificada del mismo, descrita en [Sa11] para el caso 1D, y que hemos extendido a problemas bidimensionales.

Consideramos un medio estratificado formado por una capa de fluido no viscoso homogéneo con densidad constante ρ_1 (agua), y una capa de material granular con densidad ρ_s y porosidad ψ_0. Suponemos que ambas capas son

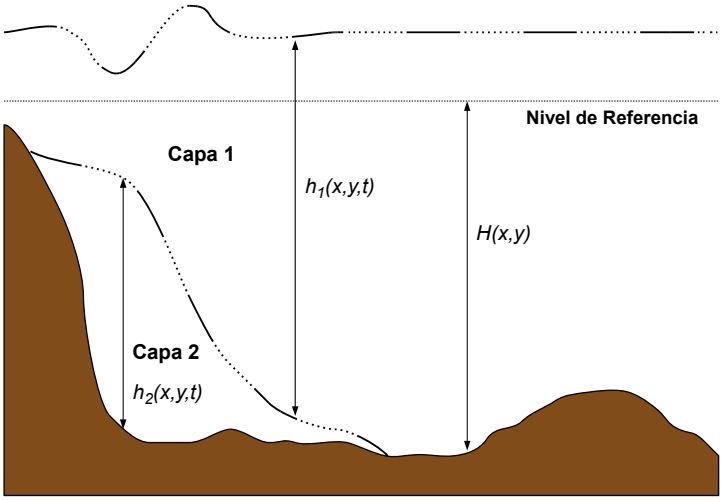


Figura 6.2: Esquema del modelo de simulación de tsunamis. Relación entre h_1, h_2 y H.

inmiscibles y que la densidad media de la capa de material granular viene dada por $\rho_2 = (1 - \psi_0)\rho_s + \psi_0\rho_1$. El sistema tiene la siguiente forma:

$$\begin{cases} \dfrac{\partial h_1}{\partial t} + \dfrac{\partial q_{1,x}}{\partial x} + \dfrac{\partial q_{1,y}}{\partial y} = 0 \\ \dfrac{\partial q_{1,x}}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_{1,x}^2}{h_1} + \dfrac{g}{2}h_1^2\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_{1,x}q_{1,y}}{h_1}\right) = -gh_1\dfrac{\partial h_2}{\partial x} + gh_1\dfrac{\partial H}{\partial x} + S_{f_1}(W) \\ \dfrac{\partial q_{1,y}}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_{1,x}q_{1,y}}{h_1}\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_{1,y}^2}{h_1} + \dfrac{g}{2}h_1^2\right) = -gh_1\dfrac{\partial h_2}{\partial y} + gh_1\dfrac{\partial H}{\partial y} + S_{f_2}(W) \\ \dfrac{\partial h_2}{\partial t} + \dfrac{\partial q_{2,x}}{\partial x} + \dfrac{\partial q_{2,y}}{\partial y} = 0 \\ \dfrac{\partial q_{2,x}}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_{2,x}^2}{h_2} + \dfrac{g}{2}h_2^2\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_{2,x}q_{2,y}}{h_2}\right) = -grh_2\dfrac{\partial h_1}{\partial x} + gh_2\dfrac{\partial H}{\partial x} + S_{f_3}(W) + \tau_x \\ \dfrac{\partial q_{2,y}}{\partial t} + \dfrac{\partial}{\partial x}\left(\dfrac{q_{2,x}q_{2,y}}{h_2}\right) + \dfrac{\partial}{\partial y}\left(\dfrac{q_{2,y}^2}{h_2} + \dfrac{g}{2}h_2^2\right) = -grh_2\dfrac{\partial h_1}{\partial y} + gh_2\dfrac{\partial H}{\partial y} + S_{f_4}(W) + \tau_y \end{cases} \tag{6.1}$$

En estas ecuaciones, el subíndice 1 hace referencia a la capa superior, y el subíndice 2 a la capa inferior. $h_i(x, y, t)$, $H(x, y)$, $q_i(x, y, t)$, $u_i(x, y, t)$, g y r se definen del mismo modo que para el sistema (3.1). En la Figura 6.2 se muestra gráficamente la relación entre h_1, h_2 y H.

Los términos $S_{f_i}(W)$, $i = 1, \ldots, 4$, modelan los distintos efectos de la fricción dinámica, mientras que $\boldsymbol{\tau} = (\tau_x, \tau_y)$ modela la fricción estática. $S_{f_i}(W)$, $i = 1, \ldots, 4$, vienen dados por:

$$S_{f_1}(W) = S_{c_x}(W) + S_{a_x}(W) \qquad S_{f_3}(W) = -r\, S_{c_x}(W) + S_{b_x}(W)$$
$$S_{f_2}(W) = S_{c_y}(W) + S_{a_y}(W) \qquad S_{f_4}(W) = -r\, S_{c_y}(W) + S_{b_y}(W)$$

$S_c(W) = \big(S_{c_x}(W),\, S_{c_y}(W)\big)$ parametriza la fricción entre las dos capas, y se define como:

$$\begin{cases} S_{c_x}(W) = m_f\, \dfrac{h_1 h_2}{h_2 + r h_1}\, (u_{2,x} - u_{1,x})\, \|u_2 - u_1\| \\ S_{c_y}(W) = m_f\, \dfrac{h_1 h_2}{h_2 + r h_1}\, (u_{2,y} - u_{1,y})\, \|u_2 - u_1\| \end{cases}$$

donde m_f es una constante positiva.

$S_a(W) = \big(S_{a_x}(W),\, S_{a_y}(W)\big)$ parametriza la fricción entre el fluido y el fondo fijo, y viene determinado por una ley de Manning:

$$\begin{cases} S_{a_x}(W) = -g h_1\, \dfrac{n_1^2}{h_1^{4/3}}\, u_{1,x}\, \|u_1\| \\ S_{a_y}(W) = -g h_1\, \dfrac{n_1^2}{h_1^{4/3}}\, u_{1,y}\, \|u_1\| \end{cases}$$

donde $n_1 > 0$ es el coeficiente de Manning.

$S_b(W) = \big(S_{b_x}(W),\, S_{b_y}(W)\big)$ parametriza la fricción entre el medio granular y el fondo, y al igual que en el caso anterior, viene determinado por una ley de Manning:

$$\begin{cases} S_{b_x}(W) = -g h_2\, \dfrac{n_2^2}{h_2^{4/3}}\, u_{2,x}\, \|u_2\| \\ S_{b_y}(W) = -g h_2\, \dfrac{n_2^2}{h_2^{4/3}}\, u_{2,y}\, \|u_2\| \end{cases}$$

donde $n_2 > 0$ es el correspondiente coeficiente de Manning.

Nótese que $S_a(W)$ sólo está definido en aquellas zonas del dominio donde $h_2(x, y, t) = 0$. En este caso, $m_f = 0$ y $n_2 = 0$. Del mismo modo, si $h_1(x, y, t) = 0$ suponemos que $m_f = 0$ y $n_1 = 0$.

Por último, $\boldsymbol{\tau} = (\tau_x,\, \tau_y)$ parametriza los efectos de la fricción estática. Consideramos dos posibles leyes de fricción: la ley de Pouliquen [PF02] o la de Coulomb. En el caso de la ley de Pouliquen, el vector de fricción $\boldsymbol{\tau}$ se obtiene a partir del grosor de la capa de material granular y de su velocidad media, mientras que para la ley de Coulomb, $\boldsymbol{\tau}$ se define del siguiente modo:

$$\text{Si } \|\boldsymbol{\tau}\| \geq \sigma^c \Rightarrow \begin{cases} \tau_x = -g(1-r)h_2 \dfrac{q_{2,x}}{\|q_2\|} \tan(\alpha) \\ \tau_y = -g(1-r)h_2 \dfrac{q_{2,y}}{\|q_2\|} \tan(\alpha) \end{cases}$$

$$\text{Si } \|\boldsymbol{\tau}\| < \sigma^c \Rightarrow \; q_{2,x} = 0, \quad q_{2,y} = 0$$

donde $\sigma^c = g(1-r)h_2 \tan(\alpha)$, siendo α el ángulo de reposo o ángulo de fricción de Coulomb.

El sistema (6.1) puede escribirse como un sistema de leyes de conservación con términos fuente y productos no conservativos del siguiente modo:

$$\begin{aligned} \frac{\partial W}{\partial t} + \frac{\partial F_1}{\partial x}(W) + \frac{\partial F_2}{\partial y}(W) \; = \; & B_1(W)\frac{\partial W}{\partial x} + B_2(W)\frac{\partial W}{\partial y} \\ & + S_1(W)\frac{\partial H}{\partial x} + S_2(W)\frac{\partial H}{\partial y} + S_F(W) \end{aligned} \tag{6.2}$$

donde W, $F_1(W)$, $F_2(W)$, $B_1(W)$, $B_2(W)$, $S_1(W)$ y $S_2(W)$ se definieron en (3.3). Nótese que la Ecuación (6.2) es igual que (3.2) con la adición del término $S_F(W)$, que contiene los términos de fricción y se define del siguiente modo:

$$S_F(W) = \begin{pmatrix} 0 \\ S_{f_1}(W) \\ S_{f_2}(W) \\ 0 \\ S_{f_3}(W) + \tau_x \\ S_{f_4}(W) + \tau_y \end{pmatrix}.$$

6.3. Esquema Numérico

La discretización del sistema (6.2) es similar a la descrita en la sección 3.2 para el sistema (3.2). En el caso de trabajar con mallas estructuradas, los volúmenes son cuadrados o rectángulos, todos ellos del mismo tamaño, y $N_i \subset \mathbb{R}^2$ indica el centro del volumen V_i (ver la Figura 6.3).

El esquema numérico que usaremos es el PVM-IFCP (ver sección 3.4), particularizado para la simulación de tsunamis generados mediante avalanchas submarinas. Si denotamos:

$$R_{ij} = \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},j}) - \mathcal{F}_1(W_{\boldsymbol{\eta}_{ij},i}) - \mathcal{B}_{ij}\left(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i}\right) - \mathcal{S}_{ij}\left(H_j - H_i\right)$$

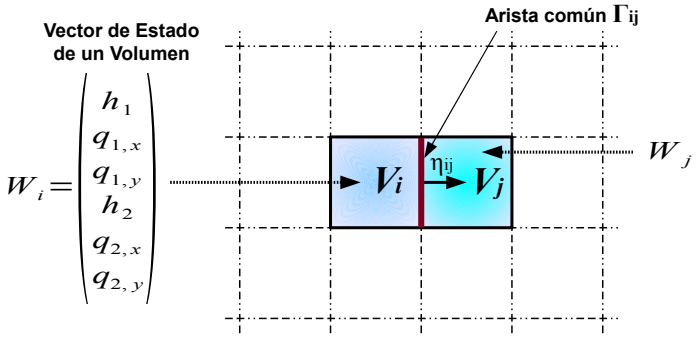


Figura 6.3: Volúmenes finitos para mallas estructuradas.

y tenemos en cuenta la definición de la matriz de viscosidad Q_{ij} para el esquema PVM-IFCP, podemos reescribir la Ecuación (3.10), una vez cancelados los flujos numéricos finales, como:

$$\Phi^{\pm}_{\boldsymbol{\eta}_{ij}} = \frac{1}{2}\Big(R_{ij} \pm \Big(\alpha_0\big(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i} - \mathcal{A}_{ij}^{-1}\mathcal{S}_{ij}\left(H_j - H_i\right)\big) + \alpha_1 R_{ij} + \alpha_2 \mathcal{A}_{ij} R_{ij}\Big)\Big) \tag{6.3}$$

Si la matriz $\mathcal{A}_{ij}$ tiene un autovalor cero, entonces $\mathcal{A}_{ij}$ no es invertible y, por tanto, la expresión (6.3) no está bien definida. Para que el esquema numérico esté definido en cualquier situación, sustituimos $\mathcal{A}_{ij}^{-1}$ por $\left(\mathcal{A}^*_{ij}\right)^{-1}$, donde $\mathcal{A}^*_{ij}$ es la matriz $\mathcal{A}_{ij}$ construida a partir de los estados $T\big(W_{\boldsymbol{\eta}_{ij},i}\big)$ y $T\big(W_{\boldsymbol{\eta}_{ij},j}\big)$, siendo

$$T(W) = T\begin{pmatrix} h_1 \\ q_{1,\boldsymbol{\eta}} \\ h_2 \\ q_{2,\boldsymbol{\eta}} \end{pmatrix} = \begin{pmatrix} h_1 \\ 0 \\ h_2 \\ 0 \end{pmatrix}.$$

Nótese que, en una situación de agua y material granular en reposo, $T(W) = W$.

Es fácil probar que $\alpha_0\big(W_{\boldsymbol{\eta}_{ij},j} - W_{\boldsymbol{\eta}_{ij},i} - \left(\mathcal{A}^*_{ij}\right)^{-1}\mathcal{S}_{ij}\left(H_j - H_i\right)\big)$ puede escribirse como $\alpha_0\widetilde{I}_{ij}$, donde

$$\widetilde{I}_{ij} = \begin{pmatrix} (\mu_{1,j} - \mu_{1,i}) - (\mu_{2,j} - \mu_{2,i}) \\ q_{1,j,\boldsymbol{\eta}} - q_{1,i,\boldsymbol{\eta}} \\ \mu_{2,j} - \mu_{2,i} \\ q_{2,j,\boldsymbol{\eta}} - q_{2,i,\boldsymbol{\eta}} \end{pmatrix} = \begin{pmatrix} \Delta_{ij}\mu_1 - \Delta_{ij}\mu_2 \\ \Delta_{ij} q_{1,\boldsymbol{\eta}} \\ \Delta_{ij}\mu_2 \\ \Delta_{ij} q_{2,\boldsymbol{\eta}} \end{pmatrix}$$

siendo $q_{k,i,\boldsymbol{\eta}}$, $k = 1, 2$, el valor de $q_{k,\boldsymbol{\eta}}$ en el volumen V_i, y $\mu_{k,i}$, $k = 1, 2$, el valor de μ_k en el volumen V_i, con $\mu_1 = h_1 + h_2 - H$ y $\mu_2 = h_2 - H$.

De este modo, podemos escribir (6.3) como:

$$\Phi^{\pm}_{\boldsymbol{\eta}_{ij}} = \frac{1}{2}\Big(R_{ij} \pm \big(\alpha_0 \widetilde{I}_{ij} + \alpha_1 R_{ij} + \alpha_2 \mathcal{A}_{ij} R_{ij}\big)\Big).$$

Obsérvese que $R_{ij} = F_Q(W_{\boldsymbol{\eta}_{ij},j}) - F_Q(W_{\boldsymbol{\eta}_{ij},i}) + P_{ij}$, donde

$$P_{ij} = \begin{pmatrix} 0 \\ g h_{1,ij} \Delta_{ij} \mu_1 \\ 0 \\ g h_{2,ij}\big(r\Delta_{ij}\mu_1 + (1-r)\Delta_{ij}\mu_2\big) \end{pmatrix}, \qquad F_Q(W) = \begin{pmatrix} q_{1,\boldsymbol{\eta}} \\ \dfrac{q^2_{1,\boldsymbol{\eta}}}{h_1} \\ q_{2,\boldsymbol{\eta}} \\ \dfrac{q^2_{2,\boldsymbol{\eta}}}{h_2} \end{pmatrix}$$

con

$$h_{k,ij} = \frac{h_{k,i} + h_{k,j}}{2}, \quad k = 1, 2.$$

Finalmente, el esquema numérico consiste en la aplicación del método PVM-IFCP, donde la Ecuación (3.10) se sustituye por:

$$\begin{aligned} \Phi^{-}_{\boldsymbol{\eta}_{ij}} &= \frac{1}{2}\Big(R_{ij} - \big(\alpha_0 \widetilde{I}_{ij} + \alpha_1 R_{ij} + \alpha_2 \mathcal{A}_{ij} R_{ij}\big)\Big) + F_Q(W_{\boldsymbol{\eta}_{ij},i})\,, \\ \Phi^{+}_{\boldsymbol{\eta}_{ij}} &= \frac{1}{2}\Big(R_{ij} + \big(\alpha_0 \widetilde{I}_{ij} + \alpha_1 R_{ij} + \alpha_2 \mathcal{A}_{ij} R_{ij}\big)\Big) - F_Q(W_{\boldsymbol{\eta}_{ij},j})\,. \end{aligned} \tag{6.4}$$

6.4. Tratamiento Seco-Mojado

Si aplicamos el esquema numérico anterior en problemas donde aparecen frentes seco-mojado, los resultados no son correctos: el gradiente del fondo genera fuerzas de presión espurias que pueden hacer que el fluido suba por pendientes de forma poco realista. En [CFG+05] se propone una modificación del esquema numérico para el caso de una dimensión con el objetivo de evitar este problema, y que hemos adaptado a dos dimensiones para su aplicación a nuestro esquema. A continuación describimos las modificaciones del esquema numérico que se han llevado a cabo para el tratamiento de frentes seco-mojado.

6.4.1. Redefinición de los Términos de Presión

Para evitar las fuerzas de presión espurias redefinimos los términos relacionados con los efectos de la presión, P_{ij} e $\widetilde{I}_{ij}$, del siguiente modo:

$$P_{ij} = \begin{pmatrix} 0 \\ gh_{1,ij}\widetilde{\Delta}_{ij}\mu_1 \\ 0 \\ gh_{2,ij}\big(r\widetilde{\Delta}_{ij}\mu_1 + (1-r)\widetilde{\Delta}_{ij}\mu_2\big) \end{pmatrix}, \qquad \widetilde{I}_{ij} = \begin{pmatrix} \widetilde{\Delta}_{ij}\mu_1 - \widetilde{\Delta}_{ij}\mu_2 \\ \Delta_{ij}q_{1,\boldsymbol{\eta}} \\ \widetilde{\Delta}_{ij}\mu_2 \\ \Delta_{ij}q_{2,\boldsymbol{\eta}} \end{pmatrix}$$

donde

$$\widetilde{\Delta}_{ij}\mu_1 = \text{máx}(\mu_{1,j} + H_m,\, 0) - \text{máx}(\mu_{1,i} + H_m,\, 0)$$
$$\widetilde{\Delta}_{ij}\mu_2 = \text{máx}(\mu_{2,j} + H_m,\, 0) - \text{máx}(\mu_{2,i} + H_m,\, 0)$$

con $H_m = \text{mín}(H_i,\, H_j)$.

Nótese que $\widetilde{\Delta}_{ij}\mu_1$ y $\widetilde{\Delta}_{ij}\mu_2$ coinciden con $\Delta_{ij}\mu_1$ y $\Delta_{ij}\mu_2$, respectivamente, en el caso de que el fondo no sea emergente. Por tanto, usaremos siempre esta definición a la hora de calcular los términos P_{ij} e $\widetilde{I}_{ij}$.

6.4.2. Imposición de una Condición de Contorno de Tipo Pared

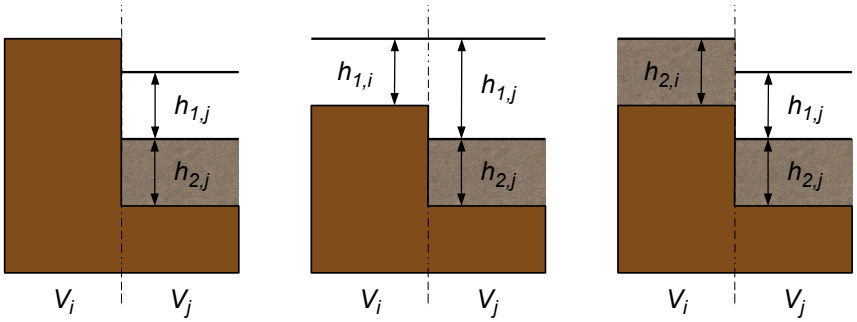
Con el objetivo de imponer que el fondo emergente actúe como una pared, impondremos velocidades nulas en aquellos volúmenes que sean vecinos a otro con fondo emergente. Pueden darse tres situaciones:

- *Desaparición de ambas capas*: Es el caso mostrado en la Figura 6.4a, donde el volumen V_i no tiene ninguna capa ($h_{1,i}+h_{2,i} < \varepsilon_h$) y su fondo sobresale por encima de las dos capas de V_j ($H_j - h_{1,j} - h_{2,j} > H_i$). En la práctica, $\varepsilon_h \approx 10^{-3}$. Entonces, definimos

$$W_{\boldsymbol{\eta}_{ij},i} = \begin{pmatrix} h_{1,i} \\ q_{1,i,\boldsymbol{\eta}} \\ h_{2,i} \\ q_{2,i,\boldsymbol{\eta}} \end{pmatrix}, \qquad W_{\boldsymbol{\eta}_{ij},j} = \begin{pmatrix} h_{1,j} \\ 0 \\ h_{2,j} \\ 0 \end{pmatrix}.$$

- *Desaparición de la capa inferior*: Es el caso mostrado en la Figura 6.4b, donde el volumen V_i no tiene capa 2 ($h_{2,i} < \varepsilon_h$) y su fondo sobresale por encima de la capa 2 de V_j ($H_j - h_{2,j} > H_i$). Entonces, definimos

$$W_{\boldsymbol{\eta}_{ij},i} = \begin{pmatrix} h_{1,i} \\ q_{1,i,\boldsymbol{\eta}} \\ h_{2,i} \\ q_{2,i,\boldsymbol{\eta}} \end{pmatrix}, \qquad W_{\boldsymbol{\eta}_{ij},j} = \begin{pmatrix} h_{1,j} \\ q_{1,j,\boldsymbol{\eta}} \\ h_{2,j} \\ 0 \end{pmatrix}.$$



(a) (b) (c)

Figura 6.4: Situaciones de fondo emergente: (a) Desaparición de ambas capas; (b) Desaparición de la capa inferior; (c) Desaparición de la capa superior.

- *Desaparición de la capa superior*: Es el caso mostrado en la Figura 6.4c, donde el volumen V_i no tiene capa 1 ($h_{1,i} < \varepsilon_h$) y su capa 2 sobresale por encima de las dos capas de V_j ($H_j - h_{1,j} - h_{2,j} > H_i - h_{2,i}$). Entonces, definimos

$$W_{\boldsymbol{\eta}_{ij},i} = \begin{pmatrix} h_{1,i} \\ q_{1,i,\boldsymbol{\eta}} \\ h_{2,i} \\ q_{2,i,\boldsymbol{\eta}} \end{pmatrix}, \qquad W_{\boldsymbol{\eta}_{ij},j} = \begin{pmatrix} h_{1,j} \\ 0 \\ h_{2,j} \\ q_{2,j,\boldsymbol{\eta}} \end{pmatrix}.$$

Si el fondo emergente se da en el volumen V_j, se procede de igual forma.

A continuación, aplicamos el esquema numérico considerando los estados $W_{\boldsymbol{\eta}_{ij},i}$ y $W_{\boldsymbol{\eta}_{ij},j}$ definidos tal y como hemos descrito.

6.5. Tratamiento de los Términos de Fricción

En esta sección veremos cómo queda finalmente el esquema numérico tras discretizar los términos de fricción $S_F(W)$ del sistema (6.2). Los términos $S_{f_1}(W)$, $S_{f_2}(W)$, $S_{f_3}(W)$ y $S_{f_4}(W)$ se discretizarán de forma semi-implícita como se describe en [BMPV03]. La discretización del término de fricción $\boldsymbol{\tau}$ se realizará aplicando el procedimiento que se explica en [FNBB+08].

Dada la aproximación del promedio de la solución W_i^n en el volumen V_i en el instante t^n, el esquema resultante consta de tres pasos, que notaremos del siguiente modo:

$$W_i^n \to W_i^{n+1/3} \to W_i^{n+2/3} \to W_i^{n+1}$$

6.5.1. Paso 1: cálculo de $W_i^{n+1/3}$

Este paso consiste en la aplicación del esquema PVM-IFCP descrito a lo largo de las secciones anteriores, teniendo en cuenta el término de fricción $\boldsymbol{\tau}$ y su influencia en los términos de presión. El esquema queda como sigue:

$$W_i^{n+1/3} = W_i^n - \frac{\Delta t}{|V_i|} \sum_{j \in \aleph_i} |\Gamma_{ij}| \, \mathcal{F}_{ij}^{PVM^-}(W_i, W_j, H_i, H_j) \tag{6.5}$$

Nótese que esta expresión es igual que (3.9) renombrando W_i^{n+1} como $W_i^{n+1/3}$. En el cálculo de $\mathcal{F}_{ij}^{PVM^-}$, (6.4) se sustituye por:

$$\Phi^-_{\boldsymbol{\eta}_{ij}} = \frac{1}{2}\Big(R_{ij} - \big(\alpha_0 \widetilde{I}^{\boldsymbol{\tau}}_{ij} + \alpha_1 R^{\boldsymbol{\tau}}_{ij} + \alpha_2 \mathcal{A}_{ij} R^{\boldsymbol{\tau}}_{ij}\big)\Big) + F_Q(W_{\boldsymbol{\eta}_{ij},i}) \, ,$$
$$\Phi^+_{\boldsymbol{\eta}_{ij}} = \frac{1}{2}\Big(R_{ij} + \big(\alpha_0 \widetilde{I}^{\boldsymbol{\tau}}_{ij} + \alpha_1 R^{\boldsymbol{\tau}}_{ij} + \alpha_2 \mathcal{A}_{ij} R^{\boldsymbol{\tau}}_{ij}\big)\Big) - F_Q(W_{\boldsymbol{\eta}_{ij},j}) \, ,$$

donde

$$\widetilde{I}^{\boldsymbol{\tau}}_{ij} = \begin{cases} \widetilde{I}_{ij} & \text{si } |h_{2,ij}\, u_{2,\boldsymbol{\eta}_{ij}}| > \Delta t \, \sigma^c_{ij} \\ \begin{pmatrix} \widetilde{\Delta}_{ij}\mu_1 \\ \Delta_{ij} q_{1,\boldsymbol{\eta}} \\ 0 \\ \Delta_{ij} q_{2,\boldsymbol{\eta}} \end{pmatrix} & \text{en otro caso} \end{cases}$$

Si se utiliza la ley de fricción de Coulomb, $\sigma^c_{ij} = g(1-r)h_{2,ij}\tan(\alpha)$. En caso de aplicar la ley de fricción de Pouliquen, se utiliza una expresión equivalente a la anterior (ver la función `terminoFriccion-Pouliquen` del Apéndice B).

Por su parte, $R^{\boldsymbol{\tau}}_{ij}$ viene dado por:

$$R^{\boldsymbol{\tau}}_{ij} = \begin{cases} R_{ij} & \text{si } |h_{2,ij}\, u_{2,\boldsymbol{\eta}_{ij}}| > \Delta t \, \sigma^c_{ij} \\ F_Q(W_{\boldsymbol{\eta}_{ij},j}) - F_Q(W_{\boldsymbol{\eta}_{ij},i}) + P^*_{ij} & \text{en otro caso} \end{cases}$$

donde

$$P^*_{ij} = \begin{pmatrix} 0 \\ g h_{1,ij} \widetilde{\Delta}_{ij}\mu_1 \\ 0 \\ r g h_{2,ij} \widetilde{\Delta}_{ij}\mu_1 \end{pmatrix}.$$

6.5.2. Paso 2: cálculo de $W_i^{n+2/3}$

Una vez calculado $W_i^{n+1/3}$, definimos $W_i^{n+2/3}$ como:

$$W_i^{n+2/3} = \begin{pmatrix} h_{1,i}^{n+1/3} \\ u_{1,i,x}^{n+2/3} h_{1,i}^{n+1/3} \\ u_{1,i,y}^{n+2/3} h_{1,i}^{n+1/3} \\ h_{2,i}^{n+1/3} \\ u_{2,i,x}^{n+2/3} h_{2,i}^{n+1/3} \\ u_{2,i,y}^{n+2/3} h_{2,i}^{n+1/3} \end{pmatrix} \tag{6.6}$$

donde $u_{k,i,\alpha}^{n+2/3},\ k = 1, 2,\ \alpha = x, y$, son las soluciones del siguiente sistema:

$$\begin{cases} u_{1,i,x}^{n+2/3} = u_{1,i,x}^{n+1/3} + a\, h_{2,i}^{n+1/3} \left(u_{2,i,x}^{n+2/3} - u_{1,i,x}^{n+2/3} \right) - \dfrac{\Delta t\, g\, n_1^2}{\left(h_{1,i}^{n+1/3} \right)^{4/3}} \left\| u_{1,i}^n \right\| u_{1,i,x}^{n+2/3} \\ u_{1,i,y}^{n+2/3} = u_{1,i,y}^{n+1/3} + a\, h_{2,i}^{n+1/3} \left(u_{2,i,y}^{n+2/3} - u_{1,i,y}^{n+2/3} \right) - \dfrac{\Delta t\, g\, n_1^2}{\left(h_{1,i}^{n+1/3} \right)^{4/3}} \left\| u_{1,i}^n \right\| u_{1,i,x}^{n+2/3} \\ u_{2,i,x}^{n+2/3} = u_{2,i,x}^{n+1/3} - r\, a\, h_{1,i}^{n+1/3} \left(u_{2,i,x}^{n+2/3} - u_{1,i,x}^{n+2/3} \right) - \dfrac{\Delta t\, g\, n_2^2}{\left(h_{2,i}^{n+1/3} \right)^{4/3}} \left\| u_{2,i}^n \right\| u_{2,i,x}^{n+2/3} \\ u_{2,i,y}^{n+2/3} = u_{2,i,y}^{n+1/3} - r\, a\, h_{1,i}^{n+1/3} \left(u_{2,i,y}^{n+2/3} - u_{1,i,y}^{n+2/3} \right) - \dfrac{\Delta t\, g\, n_2^2}{\left(h_{2,i}^{n+1/3} \right)^{4/3}} \left\| u_{2,i}^n \right\| u_{2,i,y}^{n+2/3} \end{cases} \tag{6.7}$$

con

$$a = \Delta t\, \frac{m_f}{r h_{1,i}^{n+1/3} + h_{2,i}^{n+1/3}} \left\| u_{1,i}^n - u_{2,i}^n \right\|.$$

Obsérvese que (6.7) se corresponde con una discretización semi-implícita de los términos de fricción $S_{f_1}(W),\ S_{f_2}(W),\ S_{f_3}(W)$ y $S_{f_4}(W)$.

Este paso se corresponde con la función `discretizacionImplicita` del Apéndice B.

6.5.3. Paso 3: cálculo de W_i^{n+1}

Tras calcular $W_i^{n+2/3}$, finalmente obtenemos el siguiente estado W_i^{n+1} como:

$$W_i^{n+1} = \begin{pmatrix} h_{1,i}^{n+2/3} \\ q_{1,i,x}^{n+2/3} \\ q_{1,i,y}^{n+2/3} \\ h_{2,i}^{n+2/3} \\ q_{2,i,x}^{n+1} \\ q_{2,i,y}^{n+1} \end{pmatrix} \tag{6.8}$$

donde

$$q_{2,i,\alpha}^{n+1} = \begin{cases} \dfrac{\left\| q_{2,i}^{n+2/3} \right\| q_{2,i,\alpha}^{n+2/3}}{\left\| q_{2,i}^{n+2/3} \right\| + \sigma_i \, \Delta t} & \text{si } \left\| q_{2,i}^{n+2/3} \right\| \geq \sigma_i \, \Delta t \\ 0 & \text{en otro caso} \end{cases} , \qquad \alpha = x, y$$

con $\sigma_i = g(1-r)h_{2,i}^{n+2/3}\tan(\alpha)$ si se usa la ley de fricción de Coulomb. Nuevamente, si se aplica la ley de Pouliquen, se utiliza la expresión correspondiente (ver la función `terminoFriccion-Pouliquen` del Apéndice B).

Este paso se corresponde con la función `coulomb` del Apéndice B.

El esquema que acabamos de describir es exactamente bien equilibrado para el agua en reposo, es decir, la solución permanece invariante si $q_{1,x} = q_{1,y} = 0$, μ_1 es constante y μ_2 es constante.

En el Apéndice B se muestran los cálculos matemáticos detallados de este esquema numérico, con algunas modificaciones que también permiten simular tsunamis generados por avalanchas situadas fuera del agua.

6.6. Fuentes de Paralelismo

A continuación presentamos un algoritmo paralelo basado en el esquema numérico para la simulación de tsunamis generados por avalanchas submarinas descrito en las anteriores secciones de este capítulo. La Figura 6.5 muestra esquemáticamente las etapas de dicho algoritmo paralelo, donde los pasos principales aparecen etiquetados con un número dentro de un círculo, y las principales fuentes de paralelismo de datos se representan con rectángulos superpuestos.

Como se puede ver, el algoritmo paralelo es muy similar al que se expuso en la Figura 3.4 del capítulo 3. Las principales diferencias están en la etapa de procesamiento paralelo de los volúmenes. En primer lugar, la malla de

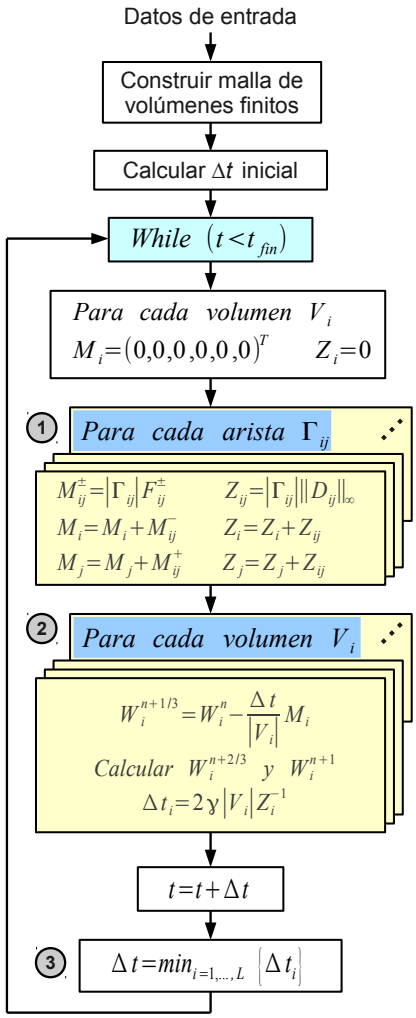


Figura 6.5: Fuentes de paralelismo del esquema de orden 1 para la simulación de tsunamis.

volúmenes finitos se construye e inicializa. Seguidamente se calcula el incremento de tiempo Δt inicial y se entra en el bucle principal, donde se itera el proceso hasta que se alcanza el tiempo final de simulación t_{fin}. El cálculo del paso de tiempo inicial es idéntico al que se describió para los esquemas de orden 1 del capítulo 3 (ver Figura 3.4). A continuación se describen los pasos principales del algoritmo paralelo:

- **Inicialización de los acumuladores**: Al principio de cada iteración del bucle principal, para cada volumen V_i, se inicializan a cero sus acumuladores M_i (un vector 6×1) y Z_i (un escalar). Son los mismos

acumuladores utilizados en los esquemas de orden 1 del capítulo 3 (ver sección 3.5.1).

① **Procesamiento de las aristas**: Para cada arista Γ_{ij} común a dos volúmenes adyacentes V_i y V_j se realizan los siguientes cálculos:

- Se calcula el vector $M_{ij}^{\pm} = |\Gamma_{ij}|\, F_{ij}^{\pm}$, donde $F_{ij}^{\pm}$ se corresponde con $\mathcal{F}_{ij}^{PVM^{\pm}}$ en la Ecuación (6.5). M_{ij}^{-} y M_{ij}^{+} representan las contribuciones de la arista al cálculo de los nuevos estados de V_i y V_j, respectivamente. Esta contribución debe sumarse a los acumuladores M_i y M_j asociados a V_i y V_j, respectivamente.
- Se calcula el valor $Z_{ij} = |\Gamma_{ij}|\, \|\mathcal{D}_{ij}\|_{\infty}$. Z_{ij} representa la contribución de la arista al cálculo de los valores locales Δt de V_i y V_j (ver Ecuación (3.5)). Esta contribución debe sumarse a los acumuladores Z_i y Z_j asociados a V_i y V_j, respectivamente.

Dado que los cálculos que se realizan para cada arista son independientes de los efectuados para las restantes aristas, el procesamiento de las aristas puede hacerse en paralelo,

② **Cálculo del nuevo estado W_i^{n+1} y del valor local Δt_i para cada volumen**: Para cada volumen V_i se realizan los siguientes cálculos:

- El estado $W_i^{n+1/3}$ se obtiene a partir del valor del acumulador M_i del siguiente modo: $W_i^{n+1/3} = W_i^n - \frac{\Delta t}{|V_i|}\, M_i$. Esta expresión se corresponde con la Ecuación (6.5). A continuación se calculan $W_i^{n+2/3}$ y W_i^{n+1} aplicando (6.6) y (6.8), respectivamente.
- El valor local Δt_i se obtiene a partir del valor del acumulador Z_i del siguiente modo (ver Ecuación (3.5)): $\Delta t_i = 2\gamma\, |V_i|\, Z_i^{-1}$.

El procesamiento de los volúmenes también puede hacerse en paralelo, ya que los cálculos asociados a cada volumen son independientes de los realizados para los restantes volúmenes.

③ **Obtención del mínimo Δt**: Este paso es idéntico al descrito en los esquemas de orden 1 del capítulo 3 (ver sección 3.5.1).

Al igual que con los esquemas de orden 1 y alto orden presentados en los capítulos 3 y 4, dado que el esquema de simulación de tsunamis posee un alto grado de paralelismo de datos, es adecuado para ser implementado en arquitecturas CUDA.

6.7. Herramienta de Visualización

En esta sección describiremos brevemente la implementación que se ha realizado en OpenGL [Shr09] para visualizar de forma realista las simulaciones de tsunamis que se han realizado.

Como punto de partida se ha tomado el tutorial de agua realista en OpenGL de GameTutorials [LLC]. Este tutorial implementa la visualización realista de un lago, con las siguientes características:

- Efectos de reflexión y refracción del agua. Para la reflexión se crea una textura que contiene la parte del entorno situada por encima del nivel del agua usando un plano de corte. Para la refracción se crean dos texturas: una con la parte del entorno correspondiente usando también un plano de corte, y otra textura de distorsión (a partir de un mapa dudv) que especifica el desplazamiento que se aplica en cada punto de la imagen reflejada o refractada. Un mapa dudv se crea simplemente como la derivada de un mapa de normales.

- Uso de un skybox para el paisaje del fondo. Un skybox es un método para crear fondos que consiste en situar la escena en el interior de un cubo, y en las caras del cubo se proyecta el paisaje de fondo deseado.

- Uso de técnicas de multitexturing para texturizar el terreno con varias texturas superpuestas. Nótese que puede utilizarse una imagen satélite como textura para representar el terreno.

- Uso de la técnica de mapeado topológico (bump mapping) para simular olas en el agua. Básicamente, esta técnica consiste en modificar las normales de la superficie de un objeto sin cambiar su geometría, con el objetivo de conseguir un determinado efecto. En nuestro caso se crea una textura (a partir de un mapa de normales) que especifica la dirección de la normal en cada punto de la superficie del agua.

- Efectos de cáusticas. Las cáusticas son los rayos de luz que aparecen reflejados o refractados en el fondo del agua. Para ello se utiliza un conjunto de texturas que simulan este efecto. Estas texturas simplemente se pintan en el fondo del agua y se va iterando entre ellas para simular el movimiento de los rayos de luz.

- Efectos de niebla en el agua dependiendo del nivel de profundidad del agua. Para ello se crea una textura de profundidad del agua.

- Para el pintado final del agua se utiliza un shader GLSL (OpenGL Shading Language) [RLKG+09] que hace uso de las texturas de reflexión, refracción, distorsión, normales y profundidad de agua citadas anteriormente.

Tomando como base esta implementación, se han realizado las siguientes modificaciones y ampliaciones:

- Se ha eliminado la niebla del agua.
- Se ha adaptado la superficie del agua a una malla triangular arbitraria.
- Se ha mejorado el aspecto de las cáusticas.
- Se ha añadido una segunda capa correspondiente al material granular. Esta segunda capa tiene asociada otra malla triangular y pueden pintarse sobre ella las cáusticas y otra textura que representa el material granular.
- Se ha implementado otro shader GLSL para el pintado del terreno y sus cáusticas en el caso de estar bajo el agua.
- Se ha añadido la posibilidad de mostrar espuma en el agua, que aparece con mayor intensidad en zonas de agua con poca profundidad y en las crestas de las olas altas. Para ello se utilizan varios valores umbrales y una textura que representa la espuma.
- Para evitar que la textura del terreno se distorsione en los saltos verticales, se ha añadido la posibilidad de aplicar un algoritmo para adaptar la textura a la malla que define el terreno. Para ello se hace uso de la librería OpenNL [INR], que incorpora una implementación en C++ del algoritmo Least Squares Conformal Maps [LPRM02].
- Se ha añadido la posibilidad de representar diversas variables en un mapa de colores en la superficie del agua, como la altura que tiene el agua en cada punto, su velocidad, caudal, etc. Puede seleccionarse cualquiera de los trece mapas de colores que vienen por defecto con MATLAB (jet, HSV, hot, cool, spring, summer, autumn, winter, gray, bone, copper, pink y lines).

La simulación puede visualizarse en tiempo real, es decir, mientras se está ejecutando, o bien a posteriori leyendo datos almacenados en ficheros. También se ha implementado la posibilidad de almacenar en ficheros, al final de la simulación, datos como la altura máxima alcanzada por el agua en cada volumen respecto a la superficie libre inicial ($\mu_1(x, y, 0) = h_1(x, y, 0) + h_2(x, y, 0) - H(x, y)$), el tiempo en el que se ha alcanzado dicha altura, los volúmenes inicialmente secos que han resultado inundados o el tiempo de propagación del tsunami. Todos estos datos pueden visualizarse con mapas de colores mediante otro programa en OpenGL que se ha implementado para tal efecto.

En las secciones 6.8.2 y 6.9.2 se muestran varios ejemplos de visualización de tsunamis en mallas estructuradas y triangulares, respectivamente.

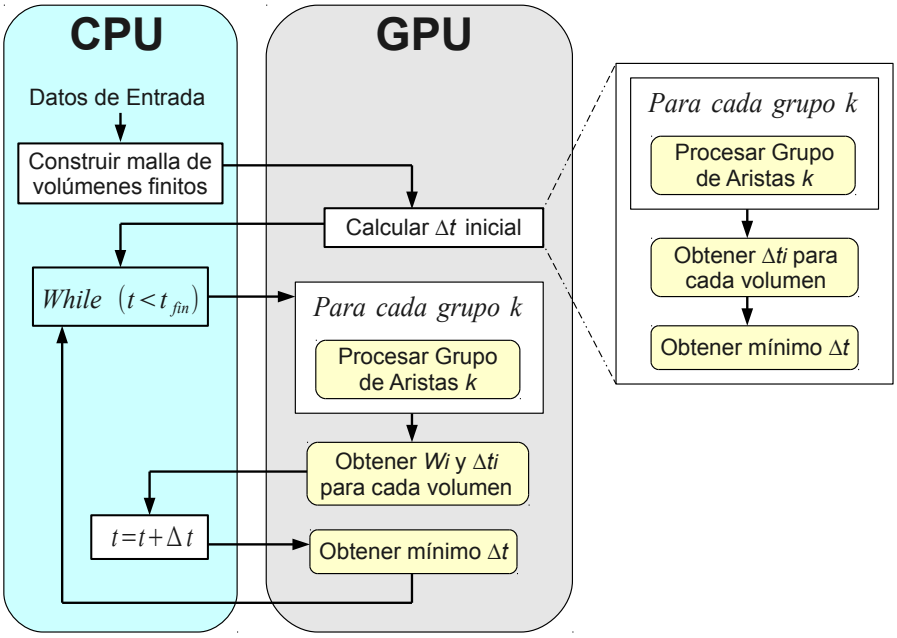


Figura 6.6: Algoritmo paralelo implementado en CUDA para el esquema de simulación de tsunamis.

6.8. Mallas Estructuradas

En esta sección se describe la adaptación a GPU e implementación en CUDA que se ha realizado del algoritmo paralelo expuesto en la sección 6.6 para mallas estructuradas (la estructura del algoritmo, mostrada en la Figura 6.5, es la misma para mallas estructuradas y triangulares). Finalmente, se realiza un experimento usando datos reales y se analizan los resultados obtenidos.

6.8.1. Implementación en CUDA

A continuación describiremos los aspectos más destacados de la implementación del algoritmo paralelo expuesto en la sección 6.6 que se ha realizado para mallas estructuradas usando el entorno de programación CUDA. Los pasos genéricos del algoritmo implementado se muestran en la Figura 6.6. Cada paso que se ejecuta en la GPU se asigna a un kernel de CUDA. Seguidamente describimos cada uno de los pasos del algoritmo:

- **Construir malla de volúmenes finitos**: En este paso se construye la estructura de datos que se usará en GPU. Para cada volumen V_i, almacenamos su estado (h_1, $q_{1,x}$, $q_{1,y}$, h_2, $q_{2,x}$ y $q_{2,y}$) y su profundidad H. Al igual que para los esquemas de orden 1 del capítulo 3, definimos

dos arrays de elementos `float4`, donde el tamaño de ambos arrays es el número de volúmenes. El primer array contiene los parámetros h_1, $q_{1,x}$, $q_{1,y}$ y H, mientras que el segundo array contiene h_2, $q_{2,x}$ y $q_{2,y}$. Dado que una malla estructurada, a diferencia de una triangular, sí sigue un patrón regular, ambos arrays pueden ser accedidos de forma alineada y, por tanto, se almacenan en memoria global.

El área de los volúmenes y la longitud de las aristas horizontales y verticales se precalculan y se pasan como parámetros directamente a los kernels de CUDA que los necesitan.

En el caso de mallas estructuradas, podemos saber en tiempo de ejecución si una arista o volumen es frontera y el valor de la normal $\boldsymbol{\eta}_{ij}$ de una arista comprobando la posición de la hebra en la malla.

- **Procesar aristas**: En este paso, cada hebra representa una arista y calcula la contribución de la arista a sus volúmenes adyacentes tal y como se describió en la sección 6.6. En esta implementación seguimos un enfoque similar al de [dlAMC10] y [dlAMC11], donde el procesamiento de las aristas se dividió en procesamiento de aristas horizontales y verticales, permitiendo que algunos términos del cálculo numérico fueran eliminados y mejorando de este modo la eficiencia. Efectivamente, para las aristas horizontales $\boldsymbol{\eta}_{ij,x} = 0$ y, por tanto, todas las operaciones donde interviene este término pueden descartarse. Del mismo modo, para las aristas verticales $\boldsymbol{\eta}_{ij,y} = 0$ y todas las operaciones donde interviene este término pueden eliminarse.

  Sin embargo, en nuestro caso, debido a la rotación que se aplica a cada arista antes de ser procesada (ver secciones 6.3–6.5), esta división del procesamiento de aristas en horizontales y verticales tiene poca influencia en la eficiencia. El principal motivo por el que realizamos esta división es la introducción de una etapa que llamamos `positividad`. Esta etapa se ejecuta al final de procesar cada arista y consiste en limitar el flujo con el que la arista contribuye a sus volúmenes adyacentes, de forma que se garantice que h_1 y h_2 siempre sean mayores o iguales que cero, manteniendo la conservación de la masa, es decir, que no se cree ni se destruya agua ni material granular durante la simulación. Para ello se realizan una serie de operaciones teniendo en cuenta, entre otros datos, los valores que tengan los acumuladores M_i y M_j en cada momento (ver la función `positividad` del Apéndice B para más detalles). Por este motivo, sólo podemos procesar una arista por volumen en cada instante, lo que nos obliga a dividir el procesamiento de aristas en cuatro kernels:

  - Procesamiento de aristas verticales pares.
  - Procesamiento de aristas verticales impares.

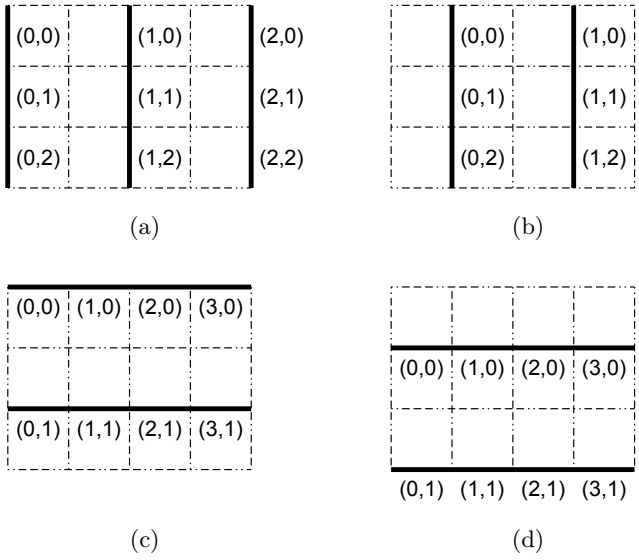


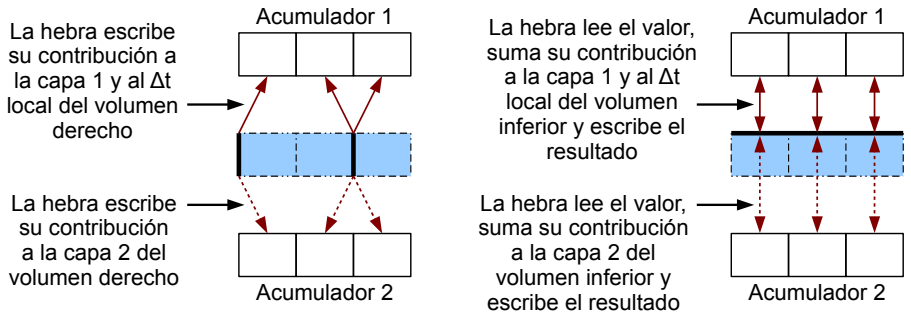
Figura 6.7: Malla 3×4 que muestra la distribución espacial de las hebras en un bloque de hebras. Para cada hebra se indica su posición (x, y): (a) Procesamiento de aristas verticales pares; (b) Procesamiento de aristas verticales impares; (c) Procesamiento de aristas horizontales pares; (d) Procesamiento de aristas horizontales impares.

- Procesamiento de aristas horizontales pares.
- Procesamiento de aristas horizontales impares.

En la Figura 6.6 cada uno de estos kernels se corresponde con el procesamiento de un grupo de aristas k.

La Figura 6.7 muestra una malla de ejemplo de tamaño 3×4 con la distribución espacial de las hebras en un bloque de hebras para cada uno de los cuatro kernels anteriores. Como se comentó en la sección 3.5.2 del capítulo 3, esta división del procesamiento de aristas en varios kernels también implica una reducción considerable de las necesidades de memoria GPU respecto a las implementaciones de los esquemas de orden 1 utilizadas en el capítulo 3, al ser necesarios menos acumuladores para almacenar las contribuciones de las aristas, lo que permite trabajar con tamaños de malla mayores usando los mismos recursos hardware, tanto para mallas estructuradas como triangulares.

Utilizamos dos acumuladores en memoria global, donde cada uno de ellos es un array de elementos `float4` y su tamaño es el número total de volúmenes. Cada elemento del primer acumulador almacena las



(a) (b)

Figura 6.8: Cálculo de la suma de las contribuciones de cada arista para cada volumen en una malla estructurada: (a) Procesamiento de aristas verticales (pares o impares); (b) Procesamiento de aristas horizontales (pares o impares).

contribuciones de las aristas a la capa 1 de W_i (un vector 3×1) y al Δt local del volumen (un valor `float`), mientras que cada elemento del segundo acumulador almacena las contribuciones de las aristas a la capa 2 de W_i (un vector 3×1). La ordenación de los volúmenes en los acumuladores es la misma que en los arrays que almacenan el estado de los volúmenes. La Figura 6.8 muestra el proceso gráficamente.

- **Obtener W_i^{n+1} y Δt_i para cada volumen**: En este paso, cada hebra representa un volumen y obtiene el siguiente estado W_i^{n+1} (a partir de M_i) y el Δt_i local del volumen V_i (a partir de Z_i) tal y como se describe en el paso ② del algoritmo paralelo expuesto en la sección 6.6.

  La contribución final M_i se obtiene del siguiente modo: los tres primeros elementos de M_i (contribución a la capa 1) se obtienen leyendo el vector 3×1 almacenado en la posición asociada al volumen V_i del acumulador 1, mientras que los tres últimos elementos de M_i (contribución a la capa 2) se obtienen leyendo el vector 3×1 almacenado en la posición asociada al volumen V_i del acumulador 2 (ver Figura 6.9a).

  La contribución final Z_i se obtiene leyendo el valor `float` almacenado en la posición asociada al volumen V_i del acumulador 1 (ver Figura 6.9b).

- **Obtener el mínimo Δt**: Este paso es idéntico al descrito para los esquemas de orden 1 del capítulo 3 (ver sección 3.5.2).

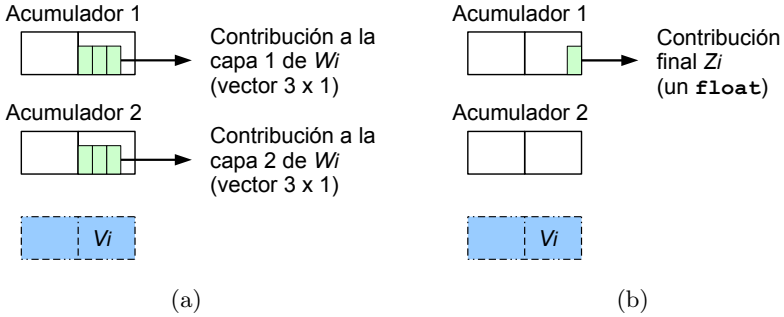


Figura 6.9: Cálculo de la contribución final de las aristas para cada volumen en una malla estructurada: (a) Contribución al cálculo de W_i; (b) Contribución al cálculo de Δt_i.

6.8.2. Simulación de un Paleotsunami en el Mar de Alborán

En esta sección ejecutaremos la implementación CUDA descrita en la sección 6.8.1 sobre un ejemplo consistente en la simulación de un paleotsunami que probablemente ocurrió en el pasado en el Mar de Alborán.

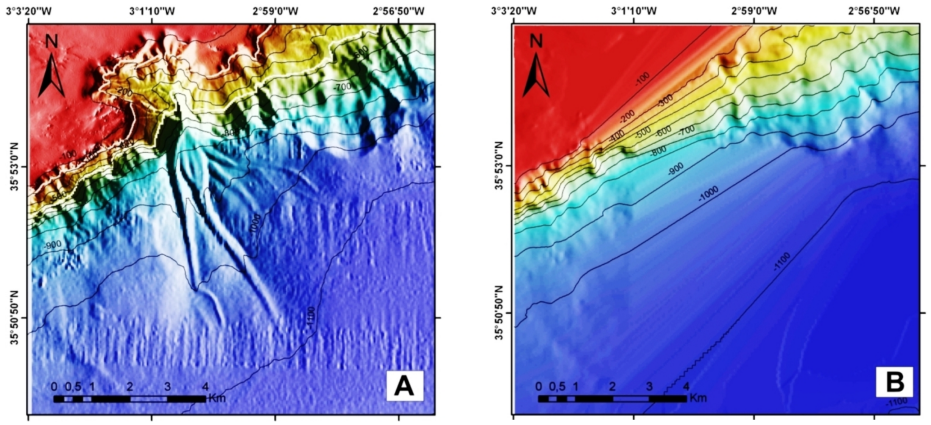
El estudio del fondo marino en la región de la Dorsal de Alborán, situada cerca de la isla de Alborán, revela la existencia de un cañón submarino y un abanico de depósitos a los pies del cañón, denominado sistema cañón-abanico Al-Borani [BVdR+00]. Ello sugiere que dicho cañón pudo originarse como consecuencia de un deslizamiento de materiales que originalmente rellenaban esta zona de la Dorsal. Se calcula que el volumen de material desprendido fue de aproximadamente mil millones de metros cúbicos.

Esta avalancha de materiales tuvo que producir un tsunami cuyos efectos debieron alcanzar las costas del sur de España y del norte de África. En particular, ciudades como Málaga, Fuengirola, Adra o Melilla debieron verse afectadas.

Para realizar esta simulación, por tanto, ha sido necesario llevar a cabo un proceso de reconstrucción del escenario existente antes del deslizamiento de materiales. La Figura 6.10 muestra el cañón-abanico Al-Borani en la actualidad y una vez efectuada esta reconstrucción.

Los datos batimétricos y topográficos del Mar de Alborán y las zonas continentales adyacentes se han obtenido de diversas fuentes (Instituto Español de Oceanografía, General Bathymetric Chart of the Oceans –GEBCO–, y el Modelo Digital del Terreno de Andalucía) y con resoluciones que varían entre 5 metros para la línea de costa española y 2 kilómetros para la zona del norte de África.

Simularemos el tsunami en un dominio de 180 × 190 kilómetros dividido en 3600 × 3800 volúmenes, es decir, un total de 13.680.000 volúmenes



(a) (b)

Figura 6.10: Sistema cañón-abanico Al-Borani en la Dorsal de Alborán: (a) Batimetría actual; (b) Paleobatimetría reconstruida.

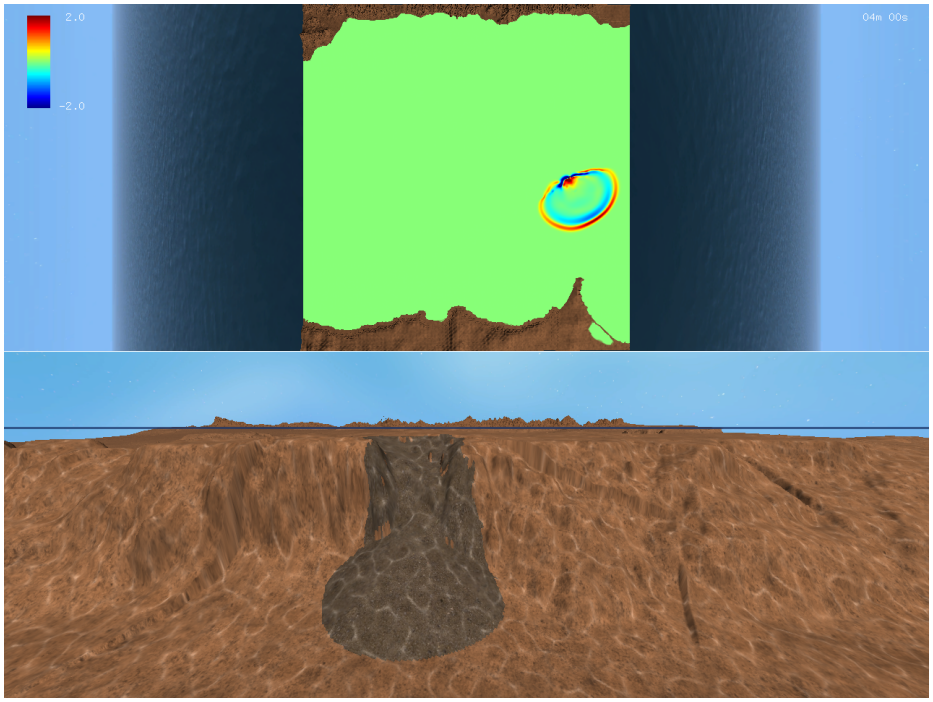
Parámetro	Valor	Parámetro	Valor
γ	0.9	n_1	0.05
r	0.55	n_2	0.4
$\alpha_1, \ldots, \alpha_4$	7º, 10º, 7º, 11º	v_{max1}	12
m_f	10^{-5}	v_{max2}	60

Tabla 6.1: Valores de los parámetros utilizados en la simulación del paleotsunami en el Mar de Alborán.

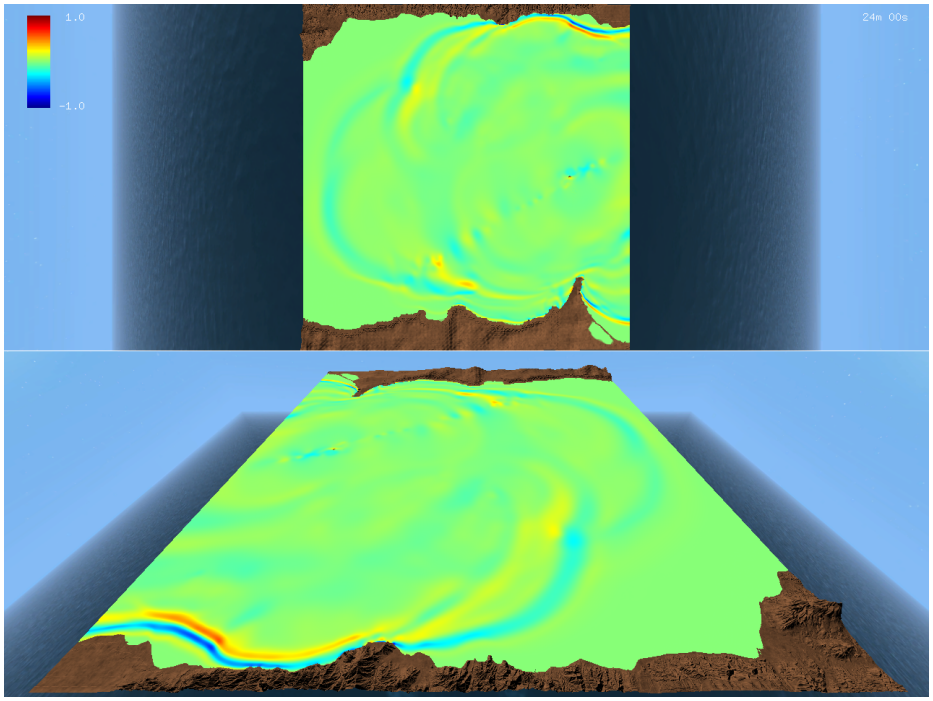
donde cada uno de ellos representa 2500 metros cuadrados. Consideraremos contornos abiertos, una hora de simulación y la ley de fricción estática de Pouliquen. La Tabla 6.1 muestra los valores de todos los parámetros utilizados (ver el Apéndice B para más detalles).

La Figura 6.11 muestra la evolución del tsunami en distintos instantes de tiempo, con la escala vertical aumentada 4 veces. En la Figura 6.11a se puede ver la avalancha submarina, mientras que en la Figura 6.11b se muestra el momento de la llegada del tsunami a las costas de Granada y Almería.

Las Figuras 6.12a y 6.12b representan la altura máxima alcanzada por el agua en cada punto respecto a la superficie libre inicial ($\mu_1(x, y, 0)$) y el tiempo en el que se ha alcanzado dicha altura, respectivamente. La Figura 6.13 muestra una vista detallada del alcance del tsunami en la costa de Málaga. En la zona correspondiente a Torremolinos, por ejemplo, el tsunami ha penetrado unos 300 metros tierra adentro. La Figura 6.14 muestra los tiempos de propagación del tsunami desde su inicio con una resolución de 2 minutos, donde se puede apreciar la influencia que ha tenido el fondo marino

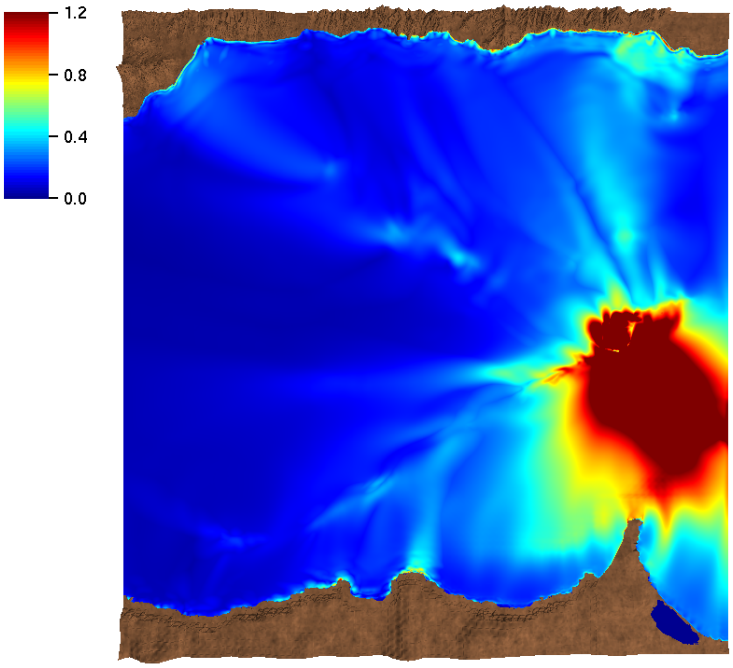


(a)

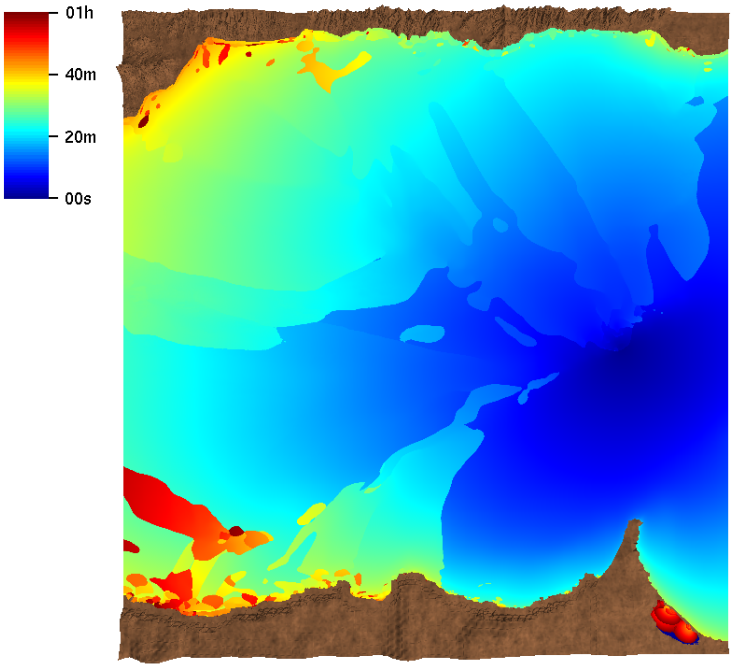


(b)

Figura 6.11: Simulación del paleotsunami en el Mar de Alborán: (a) 4 min; (b) 24 min.



(a)



(b)

Figura 6.12: Altura máxima del agua respecto a la superficie libre inicial en la simulación del paleotsunami en el Mar de Alborán: (a) Altura máxima (metros); (b) Tiempo en el que se ha alcanzado.

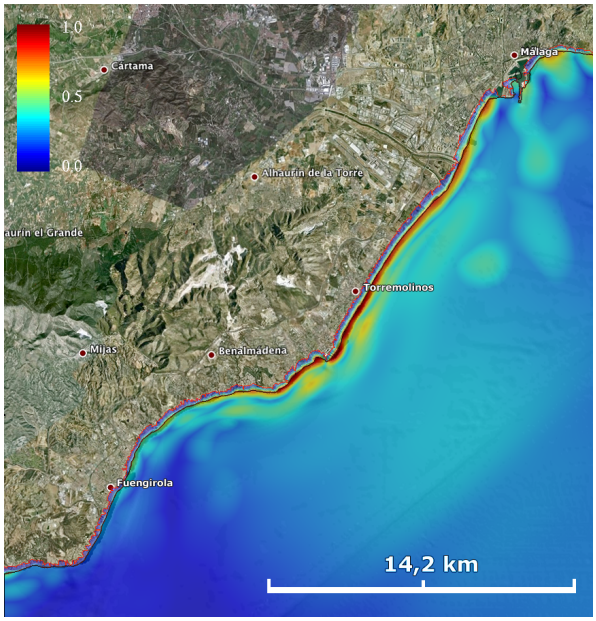


Figura 6.13: Alcance del tsunami en la costa de Málaga en la simulación del paleotsunami en el Mar de Alborán.

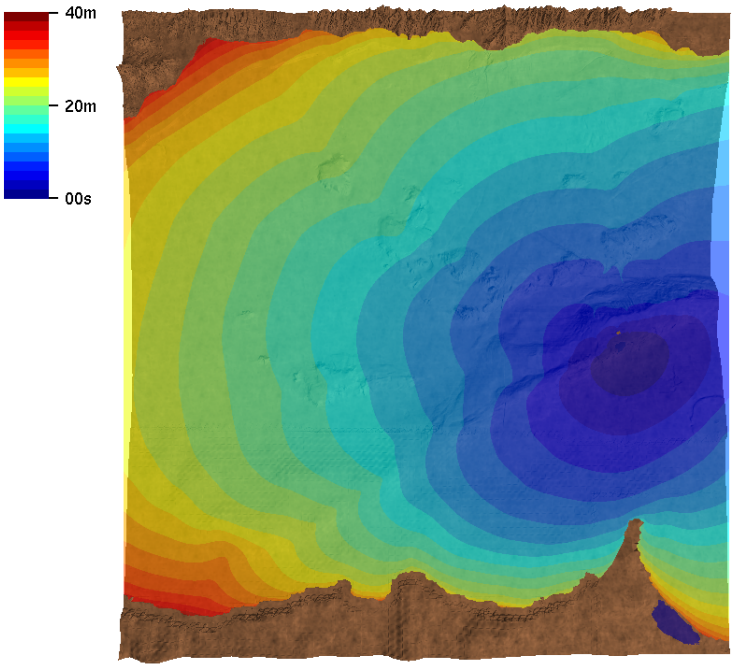


Figura 6.14: Tiempos de propagación del tsunami en la simulación del paleotsunami en el Mar de Alborán.

Volúmenes	Iteraciones	CPU 1 hebra	CPU 4 hebras	GTX 580
13680000	30	4668.2	1393.2	19.35

Tabla 6.2: Tiempos de ejecución en segundos e iteraciones realizadas con el esquema de simulación de tsunamis para la simulación del paleotsunami en el Mar de Alborán en una malla estructurada.

en dicha propagación. Podemos ver que el tsunami tarda aproximadamente 12 minutos en alcanzar el Cabo Tres Forcas, 24 minutos en llegar a las costas de Granada y Almería, y 40 minutos en alcanzar la ciudad de Málaga.

6.8.2.1. Análisis de Eficiencia

Para analizar la eficiencia de la implementación CUDA desarrollada, ejecutaremos 5 segundos de simulación del paleotsunami en el Mar de Alborán y obtendremos medidas como la ganancia respecto a implementaciones del esquema en CPU y los GFLOPS alcanzados. Al igual que con todos los esquemas vistos en capítulos anteriores, se han realizado dos implementaciones en CPU del esquema numérico para la simulación de tsunamis en mallas estructuradas, una para una sola hebra y otra para cuatro hebras usando OpenMP. Nuevamente, ambas están implementadas en C++, utilizan doble precisión numérica y hacen uso de la librería Eigen.

Todos los programas se ejecutarán en un Core i7 920 con 4 GB de memoria RAM, y se usará una tarjeta GeForce GTX 580. Se han asignado tamaños de 48 KB y 16 KB a la caché L1 y a la memoria compartida de GPU, respectivamente, para el kernel de procesamiento de aristas. El tamaño de un bloque de hebras para los kernels de procesamiento de aristas y de obtención del siguiente estado W_i^{n+1} (y del Δt local) es de $8 \times 8 = 64$ hebras en ambos kernels.

La Tabla 6.2 muestra los tiempos obtenidos en segundos para todas las implementaciones, junto con el número de iteraciones realizadas. Vemos que la implementación CUDA ha obtenido ganancias de 240 y 72 respecto a las versiones de CPU para una y cuatro hebras, respectivamente. El número de GFLOPS alcanzado por la versión CUDA ha sido 33.4.

La Tabla 6.3 muestra el porcentaje del tiempo de ejecución empleado en cada kernel con la implementación CUDA, donde se puede ver que los kernels de procesamiento de aristas y de obtención del siguiente estado (y del Δt local) de un volumen han empleado aproximadamente el 81 y el 19 % del tiempo de ejecución, respectivamente.

Al igual que con los esquemas numéricos vistos en los capítulos 3 y 4, hemos comparado las soluciones obtenidas en CPU y en GPU calculando la norma L^1 de la diferencia entre las soluciones obtenidas con la versión

Kernel	% Tiempo de ejecución	
	Alborán (estructurado)	Sumatra (triangular)
Procesar aristas	80.72	80.97
Obtener W_i y Δt_i	19.22	18.91
Obtener mínimo Δt	0.06	0.12

Tabla 6.3: Porcentaje del tiempo de ejecución empleado en cada kernel para las simulaciones de los paleotsunamis en Alborán y Sumatra con las implementaciones CUDA de simple precisión del esquema para la simulación de tsunamis usando una GeForce GTX 580.

CPU de 1 hebra y la implementación CUDA. Los órdenes de magnitud de la norma L^1 obtenidos varían entre 10^{-4} y 10^{-5}.

6.9. Mallas Triangulares

En esta sección se describe, en primer lugar, la adaptación a GPU e implementación en CUDA que se ha realizado del algoritmo paralelo expuesto en la sección 6.6 para mallas triangulares. A continuación, se realiza un experimento usando datos reales y se analizan los resultados obtenidos.

6.9.1. Implementación en CUDA

En esta sección describiremos los aspectos más destacados de la implementación del algoritmo paralelo expuesto en la sección 6.6 que se ha realizado para mallas triangulares usando el entorno de programación CUDA. Los pasos genéricos del algoritmo implementado son los mismos que en mallas estructuradas y se mostraron en la Figura 6.6. A continuación describimos cada uno de los pasos del algoritmo:

- **Construir malla de volúmenes finitos**: Este paso es idéntico al descrito para los esquemas de orden 1 y empleando las mismas estructuras de datos (ver sección 3.5.2).

- **Procesar aristas**: En este paso, cada hebra representa una arista y calcula las contribuciones de la arista a sus volúmenes adyacentes tal y como se describió en la sección 6.6. Al igual que sucedía con la implementación para mallas estructuradas, debido a la fase de positividad sólo podemos procesar una arista por volumen en cada instante. Sin embargo, en este caso la división de aristas en varios conjuntos disjuntos es algo más complicada que en mallas estructuradas. Para efectuar esta división en una malla triangular aplicamos el Algoritmo 3, que es

Algoritmo 3 Algoritmo de división de aristas en grupos

```
1: ngrupos ← 0
2: na ← Número de aristas totales
3: Marcar todas las aristas como no procesadas
4: mientras (Hay aristas sin procesar) hacer
5:   aristas[ngrupos] ← ∅
6:   naristas[ngrupos] ← 0
7:   Marcar todos los volúmenes como no procesados
8:   para k = 1 hasta na hacer
9:     Γ_ij ← Arista k-ésima
10:    si (Γ_ij no ha sido procesada) entonces
11:      si (Ningún volumen adyacente de Γ_ij ha sido procesado) entonces
12:        Añadir Γ_ij al conjunto aristas[ngrupos]
13:        naristas[ngrupos] ← naristas[ngrupos] + 1
14:        Marcar la arista Γ_ij como procesada
15:        Marcar los volúmenes adyacentes a Γ_ij como procesados
16:      fin si
17:    fin si
18:  fin para
19:  ngrupos ← ngrupos + 1
20: fin mientras
21: devolver (ngrupos, aristas, naristas)
22: ▷ ngrupos es el número de grupos de aristas, aristas[i] contiene las aristas
    del grupo i-ésimo, y naristas[i] es el número de aristas del grupo i-ésimo,
    0 ≤ i < ngrupos.
```

bastante autoexplicativo. Nótese que en una malla estructurada siempre resultan 4 grupos de aristas, mientras que en una malla triangular este número es indeterminado a priori, pero normalmente se obtienen 4 o 5 grupos. Cada uno de estos grupos de aristas será procesado por un kernel distinto.

Para la escritura de las contribuciones de las aristas, utilizamos dos acumuladores en memoria global, ambos con las mismas características y forma de uso que los acumuladores que se utilizaron en la implementación para mallas estructuradas (ver sección 6.8.1). La Figura 6.15 muestra el proceso gráficamente.

- **Obtener W_i^{n+1} y Δt_i para cada volumen**: Este paso es idéntico al descrito para el caso de mallas estructuradas (ver sección 6.8.1), con la única diferencia que el cálculo de las contribuciones finales M_i y Z_i se realiza tal y como viene esquematizado por las Figuras 6.16a y 6.16b, respectivamente.

- **Obtener el mínimo Δt**: Este paso es idéntico al descrito para los esquemas de orden 1 (ver sección 3.5.2).

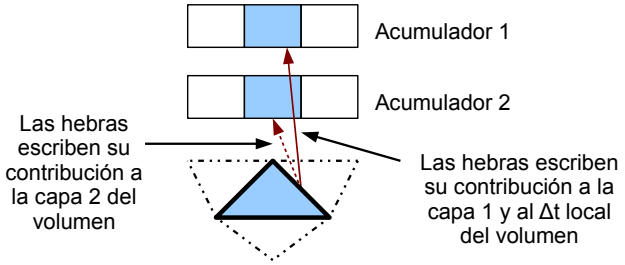


Figura 6.15: Cálculo de la suma de las contribuciones de cada arista para cada volumen con división de aristas en grupos.

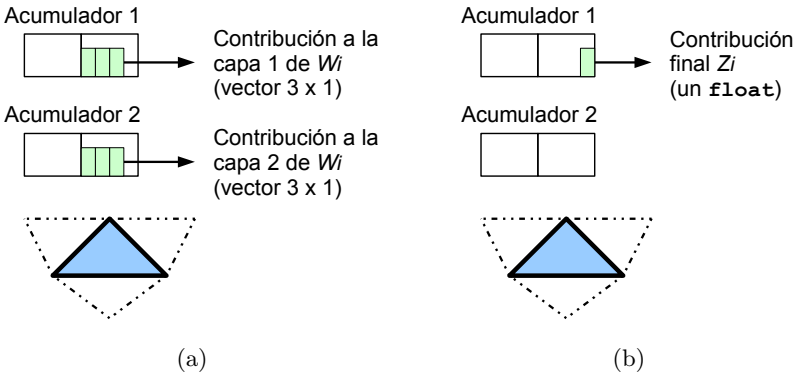


Figura 6.16: Cálculo de la contribución final de las aristas para cada volumen con división de aristas en grupos: (a) Contribución al cálculo de W_i; (b) Contribución al cálculo de Δt_i.

6.9.2. Simulación de un Paleotsunami en Sumatra

En esta sección ejecutaremos la implementación CUDA descrita en la sección 6.9.1 sobre un ejemplo consistente en la simulación de un paleotsunami ocurrido en el año 1797 en Sumatra. En ese año se produjo un fuerte terremoto en Sumatra, cuya magnitud se estima en aproximadamente 8.4 [PSH+10]. Se cree que este terremoto provocó una gran avalancha submarina cerca de la isla Siberut, que originó un tsunami que arrasó la ciudad de Padang, en la isla de Sumatra, y zonas cercanas [NSC+06].

Al igual que en el ejemplo del paleotsunami en el Mar de Alborán de la sección 6.8.2, ha sido necesario llevar a cabo un proceso de reconstrucción del escenario existente antes de la avalancha submarina, localizando la zona más probable donde se produjo el deslizamiento.

Los datos batimétricos y topográficos de la zona, así como la reconstrucción del escenario previo a la avalancha, han sido proporcionados por el Institut de Physique du Globe de París, en un mallado estructurado de

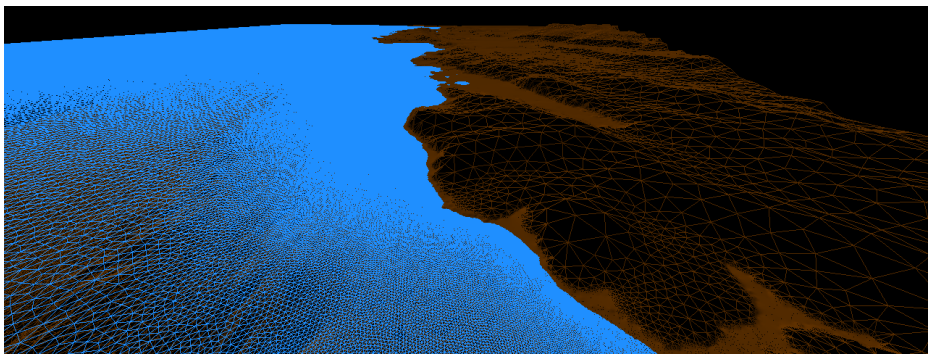


Figura 6.17: Malla triangular utilizada en la simulación del paleotsunami en Sumatra, con distintas resoluciones dependiendo de la zona.

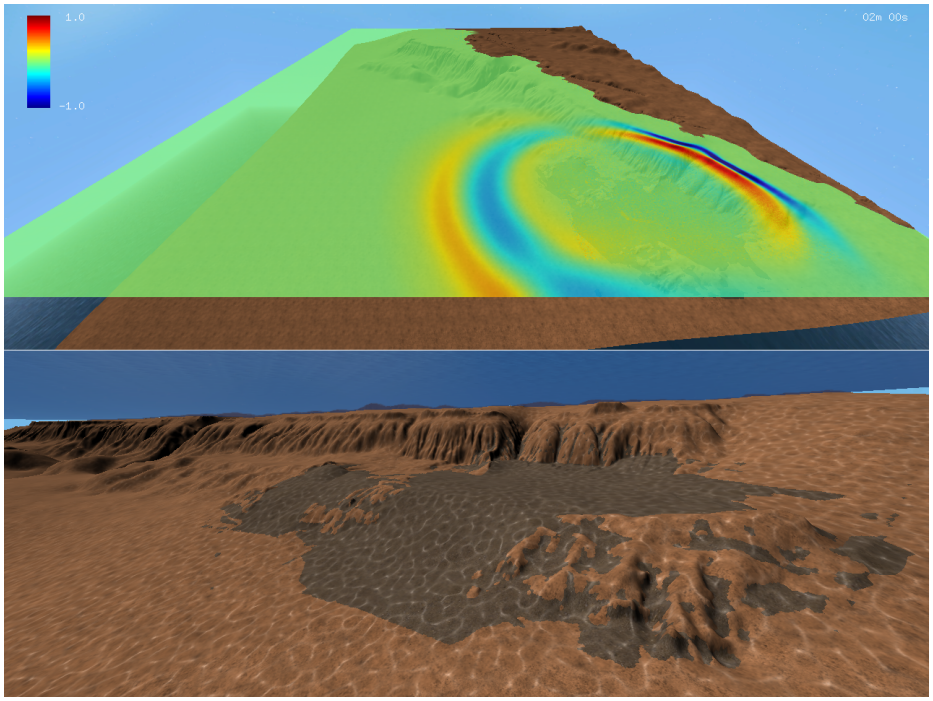
Parámetro	Valor	Parámetro	Valor
γ	0.9	n_1	0.05
r	0.55	n_2	0.4
α	10^o	v_{max1}	12
m_f	10^{-4}	v_{max2}	60

Tabla 6.4: Valores de los parámetros utilizados en la simulación del paleotsunami en Sumatra.

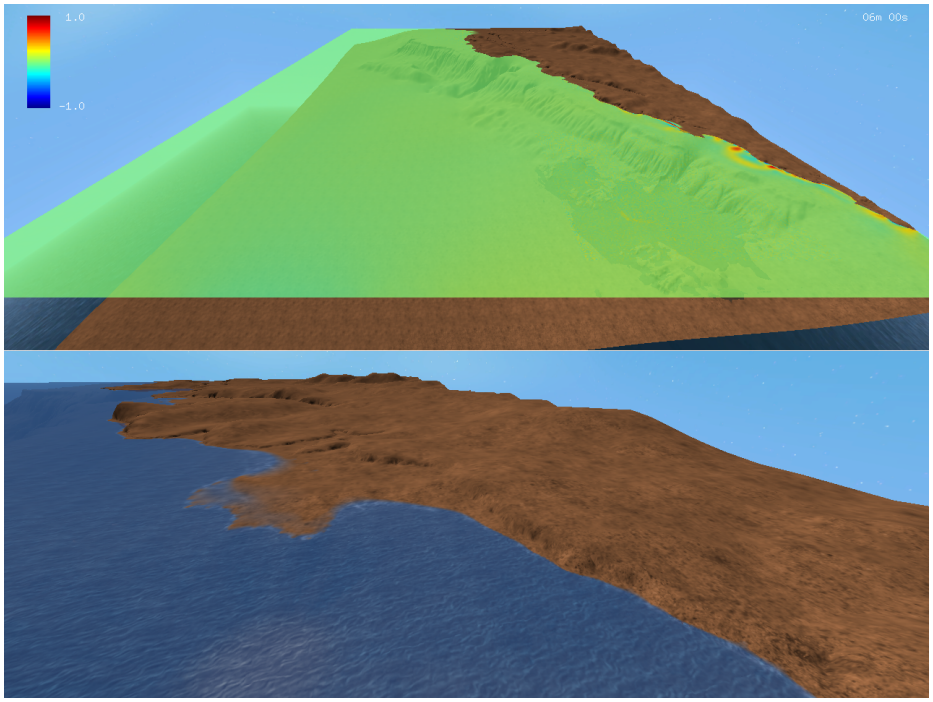
$2410 \times 3092 = 7.451.720$ volúmenes, con una resolución constante de 22 metros. A partir de estos datos, se ha creado e interpolado una malla triangular de 2.793.535 volúmenes que representa los mismos datos con distintas resoluciones dependiendo de la zona del dominio. Concretamente, las costas y la zona de la avalancha están discretizadas con la mayor resolución. A continuación, el resto de regiones que contienen agua se representan con una resolución algo menor. Finalmente, las zonas de tierra elevadas se han discretizado con una resolución muy gruesa, ya que dichas zonas no influyen en la simulación. La Figura 6.17 muestra una parte del dominio donde pueden apreciarse las distintas resoluciones consideradas.

Al aplicar el Algoritmo 3, las aristas de la malla se han particionado en 5 conjuntos disjuntos, lo que conlleva la ejecución de 5 kernels para el procesamiento de las aristas. Consideraremos contornos abiertos, 40 minutos de simulación y la ley de fricción estática de Coulomb. La Tabla 6.4 muestra los valores de todos los parámetros utilizados.

La Figura 6.18 muestra la evolución del tsunami en distintos instantes de tiempo, con la escala vertical aumentada 4 veces. En la Figura 6.18a se aprecia la avalancha submarina, mientras que en la Figura 6.18b se muestra la llegada del tsunami a las costas de la isla Siberut.

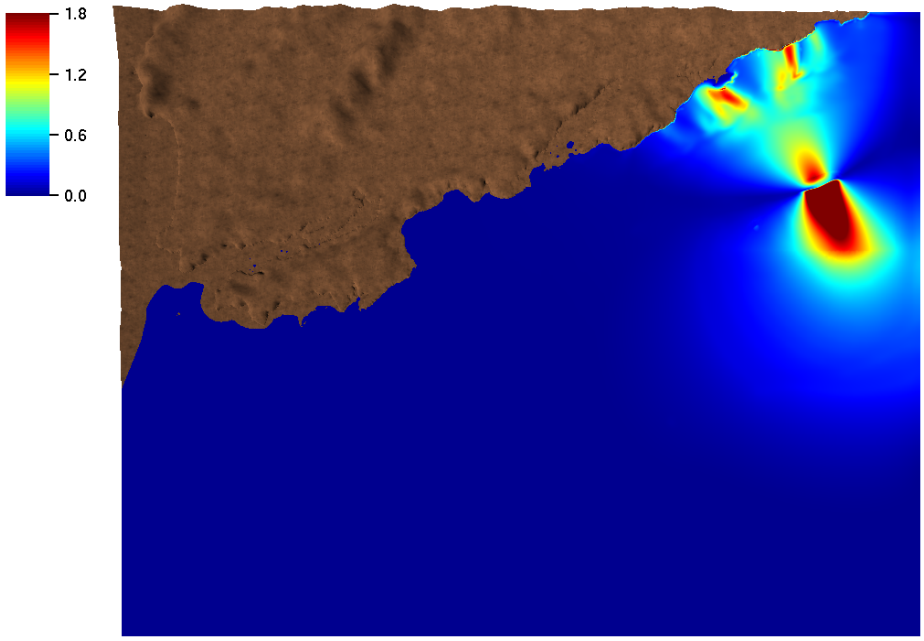


(a)

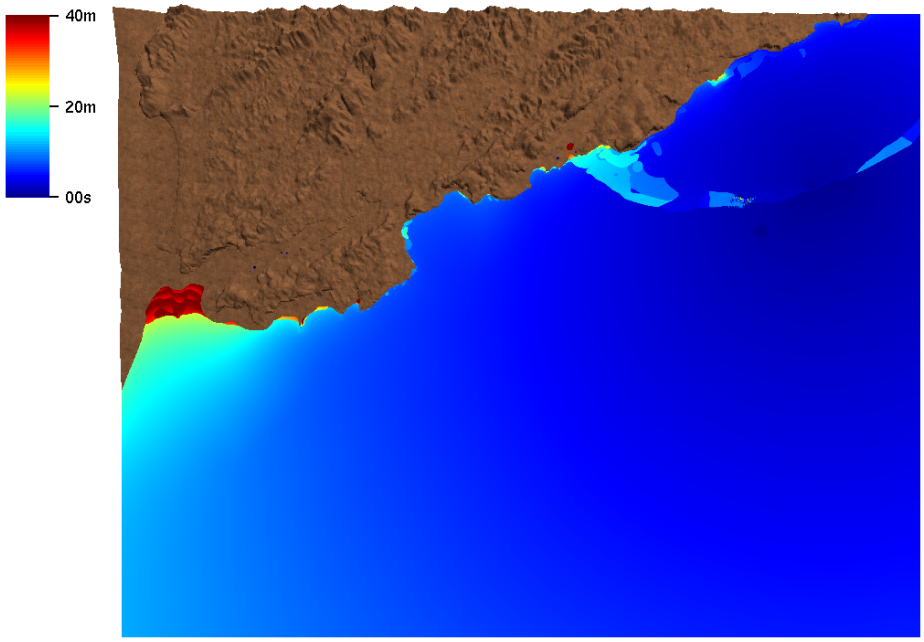


(b)

Figura 6.18: Simulación del paleotsunami en Sumatra: (a) 2 min; (b) 6 min.



(a)



(b)

Figura 6.19: Altura máxima del agua respecto a la superficie libre inicial en la simulación del paleotsunami en Sumatra: (a) Altura máxima (metros); (b) Tiempo en el que se ha alcanzado.

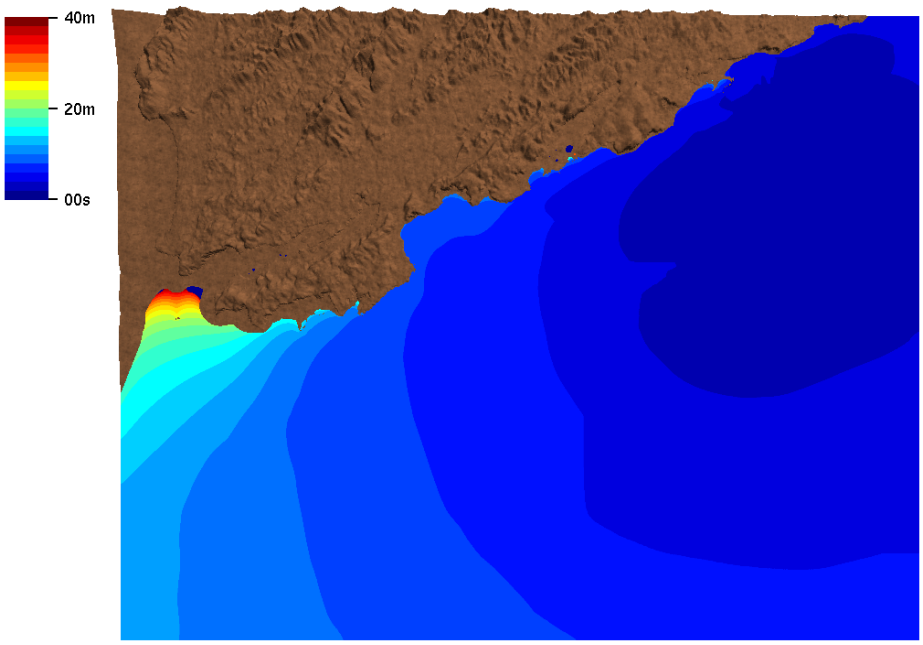


Figura 6.20: Tiempos de propagación del tsunami en la simulación del paleotsunami en Sumatra.

Las Figuras 6.19a y 6.19b representan la altura máxima alcanzada por el agua en cada punto respecto a la superficie libre inicial ($\mu_1(x, y, 0)$) y el tiempo en el que se ha alcanzado dicha altura, respectivamente. En la costa de Siberut que se muestra en la mitad derecha de dichas imágenes, las olas han penetrado hasta 100 metros tierra adentro. La Figura 6.20 representa los tiempos de propagación del tsunami desde su inicio con una resolución de 2 minutos. Podemos ver que el tsunami tarda unos 4 minutos en alcanzar las costas de la isla Siberut.

Es interesante destacar que la simulación obtenida con la malla triangular presenta una mayor difusión numérica que si se utiliza la malla estructurada de Sumatra, lo que se traduce en que las olas del tsunami pierden altura más rápidamente conforme avanza la simulación y el tsunami no llega a todas las zonas del dominio. Por este motivo, en las Figuras 6.19b y 6.20 se ha tomado como base la simulación obtenida con la malla estructurada de Sumatra.

6.9.2.1. Análisis de Eficiencia

Al igual que en el ejemplo de la sección 6.8.2, analizaremos la eficiencia de la implementación CUDA ejecutando 5 segundos de simulación del paleotsunami en Sumatra y obtendremos medidas como la ganancia respecto a implementaciones del esquema en CPU y los GFLOPS obtenidos.

Volúmenes	Iteraciones	CPU 1 hebra	CPU 4 hebras	GTX 580
2793535	230	4153.1	1343.7	18.23

Tabla 6.5: Tiempos de ejecución en segundos e iteraciones realizadas con el esquema de simulación de tsunamis para la simulación del paleotsunami en Sumatra en una malla triangular.

Nuevamente, se han realizado dos implementaciones en CPU del esquema numérico para la simulación de tsunamis en mallas triangulares, una para una sola hebra y otra para cuatro hebras usando OpenMP. Ambas están implementadas en C++, utilizan doble precisión numérica y hacen uso de la librería Eigen.

El hardware empleado para la ejecución de todos los programas, y los tamaños de caché L1, de memoria compartida y de bloques de hebras para cada kernel serán los mismos que los utilizados en la sección 6.8.2.

La Tabla 6.5 muestra los tiempos obtenidos en segundos para todas las implementaciones, junto con el número de iteraciones realizadas. Vemos que la implementación CUDA ha obtenido ganancias de 228 y 74 respecto a las versiones de CPU para una y cuatro hebras, respectivamente. El número de GFLOPS alcanzado por la versión CUDA ha sido 30.9.

En la Tabla 6.3 se muestra el porcentaje del tiempo de ejecución empleado en cada kernel con la implementación CUDA, donde se puede ver que se han obtenido porcentajes muy similares a los alcanzados con la implementación para mallas estructuradas. Para ambos tipos de mallado, los kernels de procesamiento de aristas y de obtención del siguiente estado (y del Δt local) de un volumen han empleado aproximadamente el 81 y el 19 % del tiempo total de ejecución, respectivamente.

También hemos comparado las soluciones obtenidas en CPU y en GPU calculando la norma L^1 de la diferencia entre las soluciones obtenidas con la versión CPU de 1 hebra y la implementación CUDA. En este caso, los órdenes de magnitud de la norma L^1 obtenidos varían entre 10^{-3} y 10^{-5}.

6.10. Conclusiones

En este capítulo se ha adaptado e implementado en GPU un esquema numérico de orden uno para la simulación de tsunamis generados por avalanchas submarinas y subaéreas en mallas estructuradas y triangulares. Para ambos tipos de mallado, las implementaciones CUDA desarrolladas se han utilizado para simular paleotsunamis ocurridos en la antigüedad empleando datos topográficos y batimétricos reales. La posibilidad de utilizar mallas triangulares en este tipo de simulaciones permite una mayor flexibilidad al

poder discretizar el dominio con distintas resoluciones espaciales dependiendo de la zona que se representa. Normalmente interesará discretizar las zonas costeras con mayor resolución, ya que uno de los aspectos más importantes del estudio de estas simulaciones es analizar el impacto que tiene el tsunami en las zonas de costa habitadas. Además, la división del procesamiento de aristas en varios kernels ha supuesto una reducción considerable de las necesidades de memoria respecto a las implementaciones de los esquemas de orden 1 del capítulo 3, lo cual resulta muy útil en este tipo de simulaciones, donde las dimensiones del dominio espacial pueden llegar a ser muy grandes.

Con los experimentos llevados a cabo en una tarjeta GeForce GTX 580, se han obtenido ganancias bastante similares usando mallas estructuradas y triangulares respecto a versiones de CPU ejecutadas en un Core i7 920, si bien se ha usado una ley de fricción estática distinta para ambos tipos de mallado y sería necesario realizar más experimentos. Concretamente, la implementación CUDA para mallas estructuradas ha obtenido ganancias de 240 y 72 respecto a versiones de CPU para una y cuatro hebras, respectivamente. En mallas triangulares estas ganancias han sido 228 y 74, respectivamente. Los GFLOPS alcanzados por los programas CUDA han sido aproximadamente 33 en mallas estructuradas, y 31 en mallas triangulares. Las dos implementaciones CUDA son estables en simple precisión.

Al igual que con las implementaciones de los esquemas de orden 1 vistas en el capítulo 3, la etapa de procesamiento de aristas ha resultado ser la más costosa de todo el proceso, consumiendo aproximadamente un 81 % del tiempo de ejecución en GPU tanto para mallas estructuradas como triangulares.

Finalmente, se han implementado en OpenGL varias herramientas que permiten visualizar de forma realista las simulaciones realizadas, así como efectuar análisis a posteriori de los resultados obtenidos a partir de datos almacenados en ficheros, tales como la altura máxima alcanzada por el agua en cada volumen, el tiempo en el que se ha alcanzado dicha altura, las zonas que han sido inundadas o el tiempo de propagación del tsunami.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

En este trabajo hemos llevado a cabo un amplio estudio sobre la mejora que supone utilizar hardware gráfico moderno para la simulación numérica de sistemas de aguas someras bicapa basada en volúmenes finitos sobre mallados triangulares. Exponemos a continuación las principales conclusiones que pueden extraerse:

- Las técnicas de adaptación a GPU que se han utilizado a lo largo de este trabajo han permitido sacar el máximo partido de la arquitectura de las modernas tarjetas gráficas NVIDIA con el objetivo de abordar del modo más eficiente posible la resolución de las ecuaciones de aguas someras en varias aplicaciones relacionadas con la simulación de flujos geofísicos. Los experimentos realizados muestran que, en todos los casos abordados, se obtienen mejoras sustanciales en las prestaciones respecto al uso de CPUs multinúcleo. Además, los métodos de adaptación a GPU empleados en esta tesis son fácilmente reproducibles y aplicables a otros problemas con una estructura similar.

- Respecto a los esquemas de orden uno, hemos visto que las implementaciones de los métodos de Roe (Roe clásico, IR-Roe y PVM-Roe) han proporcionado peores prestaciones en GPU que el resto de esquemas PVM (PVM-0, PVM-1U, PVM-2, PVM-2U, PVM-4 y PVM-IFCP) en términos de tiempo de ejecución y ganancia con respecto a versiones equivalentes en CPU, debido a la mayor intensidad computacional de los esquemas basados en Roe. La ventaja de los esquemas PVM es que tienen una menor demanda computacional al no requerir la descomposición espectral de la matriz de Roe y sus implementaciones son todas ellas estables en simple precisión. De todos los esquemas analizados, el

PVM-IFCP es el que ha proporcionado un mejor equilibrio entre precisión de resultados y eficiencia computacional en GPU, obteniendo resultados casi tan precisos como los métodos de Roe.

- Respecto a los esquemas de alto orden, hemos adaptado e implementado en GPU el operador de reconstrucción introducido en [DK07] y lo hemos aplicado, junto con el esquema PVM-IFCP, para obtener soluciones de orden 2 y 3. El inconveniente de este operador de reconstrucción es que es muy costoso computacionalmente. Al aumentar el orden del esquema numérico se obtienen resultados cada vez más precisos, pero los tiempos de ejecución también aumentan considerablemente. En concreto, los tiempos de GPU obtenidos con implementaciones en simple precisión de los esquemas de orden 2 y 3 han sido aproximadamente 8 y 40 veces más lentos, respectivamente, que los obtenidos con el esquema de orden 1, y las ganancias obtenidas respecto a versiones CPU también han sido inferiores que las alcanzadas en orden 1. Por otra parte, debido a la mayor intensidad computacional de los esquemas de alto orden, cuanto mayor es el orden del esquema numérico, mejor se aprovecha la tarjeta gráfica en términos de rendimiento absoluto medido en GFLOPS.

- Mediante un algoritmo multi-GPU distribuido que solapa las comunicaciones MPI con procesamiento de kernels y transferencias de memoria entre CPU y GPU, aplicado al esquema PVM-IFCP, hemos conseguido una escalabilidad fuerte cercana a la lineal usando hasta 4 GPUs, y una escalabilidad débil próxima a 1 también usando hasta 4 GPUs. El solapamiento de comunicación y computación ha demostrado ser un factor muy importante para mejorar la eficiencia en este tipo de resolvedores.

- También hemos visto que las técnicas empleadas a lo largo de esta memoria pueden aplicarse con éxito a esquemas numéricos que permiten simular escenarios reales. Concretamente, hemos adaptado e implementado en GPU un esquema para la simulación de tsunamis generados mediante avalanchas submarinas y subaéreas sobre mallas estructuradas y triangulares. El uso de mallas triangulares en este esquema resulta útil para poder discretizar con mayor resolución aquellas zonas del dominio donde se requieran resultados más precisos, como por ejemplo las zonas costeras. Además, se han conseguido reducir las necesidades de memoria respecto a las implementaciones de los esquemas de orden 1 del capítulo 3 mediante una división de las aristas en varios conjuntos disjuntos, lo cual resulta muy útil en este tipo de simulaciones, donde las dimensiones del dominio espacial pueden llegar a ser muy grandes. En los experimentos realizados en una tarjeta GeForce GTX 580, se han alcanzado en todos los casos más de 30

GFLOPS y ganancias de dos órdenes de magnitud respecto a versiones CPU secuenciales ejecutadas en un Core i7 920.

- Asimismo, se han implementado en OpenGL varias herramientas de visualización que pueden usarse conjuntamente con el esquema de simulación de tsunamis citado en el punto anterior. Estas herramientas permiten, por un lado, visualizar de forma realista las simulaciones realizadas, y por otro lado, efectuar análisis a posteriori de los resultados obtenidos a partir de datos almacenados en ficheros, como por ejemplo la altura máxima alcanzada por el agua en cada volumen, el tiempo en el que se ha alcanzado dicha altura, las zonas que han sido inundadas o el tiempo de propagación del tsunami.

- Comparando la precisión de los resultados y la eficiencia computacional al usar simple (SP) o doble precisión (DP) numérica en GPU, tenemos que, para los esquemas de orden uno, las implementaciones CUDA en DP han resultado ser entre 2 y 4 veces más lentas que usando SP, mientras que la precisión numérica alcanzada ha sido lógicamente mayor con DP que con SP, y en ambos casos del orden de la precisión de la máquina. En los esquemas de orden 2 y 3, por su parte, debido a que en las implementaciones CUDA en SP se utiliza doble precisión en una fase del cómputo, los tiempos de GPU con DP han sido entre 2 y 2.5 veces más lentos que con SP para ambos órdenes. Al aumentar el orden del esquema numérico se produce una pérdida de precisión con respecto a la obtenida con la versión CPU equivalente en DP, siendo esta pérdida más acusada en la implementación de orden 3 en SP.

- Por último, hemos comprobado que la ordenación de los volúmenes en los arrays que almacenan los datos de los volúmenes en GPU resulta ser un factor influyente en los tiempos de ejecución que se obtienen. Concretamente, aplicando la ordenación de Cuthill-McKee inversa a dichos arrays con el objetivo de reducir el ancho de banda de la matriz de adyacencia, hemos logrado reducir los tiempos de GPU de varios esquemas de orden uno entre un 5 y un 55 % (dependiendo del esquema numérico) con respecto a la ordenación proporcionada por MATLAB.

7.2. Trabajo Futuro

A continuación se proponen algunas líneas de trabajo futuro que se podrían abordar como continuación del trabajo expuesto en esta memoria:

- Los métodos WAF (Weighted Average Flux) [Tor89] son métodos TVD (Total Variation Diminishing) [Shu88] de orden 2 que proporcionan resultados numéricos casi tan precisos como un esquema de orden 2, pero

son computacionalmente más eficientes al requerir una única iteración en tiempo. Estos métodos fueron introducidos por E. F. Toro en [Tor89] y posteriormente han sido aplicados a las ecuaciones de aguas someras en [Tor92, FNNR08, dlACFN$^+$12]. Concretamente, en [dlACFN$^+$12] se aborda la implementación en GPU de un método WAF para el sistema de aguas someras de una capa en mallas estructuradas, realizando asimismo un estudio comparativo con esquemas HLL [HLvL83] de orden 1 y 2 en términos de precisión y eficiencia computacional. Sería interesante realizar un estudio similar con un método WAF para mallas triangulares, analizando su precisión y eficiencia en GPU en comparación con otros esquemas de orden 1 y 2 para mallas triangulares, como por ejemplo algunos de los descritos en esta memoria.

- Como se vio en el capítulo 6 (concretamente en el ejemplo del paleotsunami en Sumatra), los esquemas de orden uno tienen el inconveniente de presentar una alta difusión numérica para bajas resoluciones espaciales. En simulaciones basadas en escenarios reales que requieran soluciones precisas, esto puede suponer un problema importante. Un modo de tratar de solventar dicho problema es aumentando la resolución de los datos. Sin embargo, en la práctica esto no siempre será posible debido a las grandes necesidades de memoria para almacenar todos los datos. Otra alternativa para reducir esta difusión numérica consiste en utilizar esquemas de alto orden. En este sentido, los métodos WAF citados en el punto anterior resultan una excelente opción al incrementar la precisión de los resultados y reducir la difusión numérica respecto a un esquema de orden uno sin aumentar excesivamente las necesidades de memoria ni el tiempo de ejecución. Se propone, por tanto, la implementación en GPU de un método WAF aplicado al esquema de simulación de tsunamis descrito en el capítulo 6, tanto para mallas estructuradas como triangulares.

- Asimismo, se propone la implementación de un algoritmo multi-GPU para el esquema de simulación de tsunamis presentado en el capítulo 6, con el objetivo de poder aplicar dicho esquema en escenarios reales con una resolución espacial mayor que la que se podría utilizar con una sola GPU. Centrándonos en el caso de mallas triangulares, el Algoritmo 1 que se describió en el capítulo 5, correspondiente a la versión multi-GPU sin solapamiento, podría aplicarse sobre este esquema numérico sin necesidad de realizar modificaciones importantes en el algoritmo, a pesar de la división de las aristas en varios conjuntos disjuntos. Sin embargo, en el caso del Algoritmo 2, correspondiente a la versión multi-GPU con solapamiento, sí sería necesario efectuar algunos cambios. Concretamente, las modificaciones más importantes habría que realizarlas en la fase de división de aristas en grupos, iden-

tificando los grupos que contienen aristas de comunicación y los que no, y minimizando asimismo en lo posible el número de grupos con aristas de comunicación.

- Finalmente, se plantea la implementación de una clase en C++ con código CUDA integrado que sea capaz de manejar varios esquemas numéricos como los descritos en esta memoria y usando un número arbitrario de capas. El código de dicha clase debería ser más genérico que los descritos a lo largo de esta documentación (específicamente diseñados para obtener la máxima eficiencia posible resolviendo problemas concretos), lo cual lógicamente conllevaría una pérdida de rendimiento, pero con la ventaja de tener un código más general, integrado en una misma clase y reutilizable para resolver un abanico más amplio de problemas relacionados con la simulación de flujos geofísicos.

7.3. Publicaciones

El trabajo realizado durante el desarrollo de esta tesis ha dado lugar a varias publicaciones, que pasamos a citar a continuación. También se incluyen publicaciones en las que se trabaja con el sistema de aguas someras de una capa.

Libros:

- J. Macías, L. M. Fernández, J. M. González, J. T. Vázquez, M. J. Castro, P. Bárcenas, V. Díaz del Río, T. Morales, M. de la Asunción, C. Parés. *Deslizamientos y tsunamis en Alborán: un riesgo oculto bajo el mar*. Temas de Oceanografía, Instituto Español de Oceanografía, 2013.

Capítulos de libros:

- M. de la Asunción, J. M. Mantas, M. J. Castro. Programming CUDA-based GPUs to simulate two-layer shallow water flows. *Euro-Par 2010*, volumen 6272 de *Lecture Notes in Computer Science*, 353–364. Springer, 2010.

Publicaciones en revistas indexadas en el JCR (Journal Citation Reports):

- M. de la Asunción, J. M. Mantas, M. J. Castro. Simulation of one-layer shallow water systems on multicore and CUDA architectures. *Journal of Supercomputing*, 58(2):206–214, 2011.

- M. J. Castro, S. Ortega, M. de la Asunción, J. M. Mantas, J. M. Gallardo. GPU computing for shallow water flow simulation based on finite volume schemes. *Comptes Rendus Mécanique*, 339(2–3):165–184, 2011.

- J. M. Gallardo, S. Ortega, M. de la Asunción, J. M. Mantas. Two-dimensional compact third-order polynomial reconstructions. Solving nonconservative hyperbolic systems using GPUs. *Journal of Scientific Computing*, 48(1–3):141–163, 2011.

- M. de la Asunción, J. M. Mantas, M. J. Castro, E. D. Fernández-Nieto. An MPI-CUDA implementation of an improved Roe method for two-layer shallow water systems. *Journal of Parallel and Distributed Computing, Special Issue on Accelerators for High-Performance Computing*, 72(9), 1065–1072, 2012.

- M. de la Asunción, M. J. Castro, E. D. Fernández-Nieto, J. M. Mantas, S. Ortega, J. M. González. Efficient GPU implementation of a two waves TVD-WAF method for the two-dimensional one layer shallow water system on structured meshes. *Computers & Fluids*, 2012.

Publicaciones en revistas no indexadas en el JCR:

- M. J. Castro, S. Ortega, M. de la Asunción, J. M. Mantas. On the benefits of using GPUs to simulate shallow flows with finite volume schemes. *Boletín de la Sociedad Española de Matemática Aplicada*, 50:27–45, 2010.

Publicaciones en congresos internacionales:

- M. de la Asunción, J. M. Mantas, M. J. Castro. *Fast simulation of one-layer shallow water systems using CUDA architectures*. 9th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE 2009), Gijón (España), Julio 2009.

- M. de la Asunción, M. J. Castro, S. Ortega, J. M. Mantas. *An efficient implementation of a high order Roe-type finite volume numerical scheme for non-conservative hyperbolic systems using GPUs: Application to shallow flows*. Third Chilean Workshop on Numerical Analysis of Partial Differential Equations (WONAPDE 2010), Concepción (Chile), Enero 2010.

- J. M. González, M. J. Castro, M. de la Asunción, E. D. Fernández-Nieto, J. Macías, C. Parés. *Modelling submarine avalanches and generated tsunamis. Application to tsunami effects forecasting*. European Geosciences Union General Assembly 2011 (EGU 2011), Viena (Austria), Abril 2011.

- M. de la Asunción, J. M. Mantas, M. J. Castro, E. D. Fernández-Nieto. *Two-layer shallow water simulation on clusters of CUDA-enabled GPUs*. Parallel CFD 2011, Barcelona (España), Mayo 2011.

- S. Ortega, M. de la Asunción, J. M. Mantas, M. J. Castro, J. M. González, J. Macías, T. Morales, C. Parés. *Dambreak simulations on shallow flows: An efficient implementation using GPUs*. Parallel CFD 2011, Barcelona (España), Mayo 2011.

- J. M. González, M. de la Asunción, M. J. Castro, E. D. Fernández-Nieto, J. Macías, C. Parés. *Modelling tsunamis generated by submarine landslides. Application to real cases in the Mediterranean*. 13th Plinius Conference on Mediterranean Storms, Savona (Italia), Septiembre 2011.

- J. M. González, M. J. Castro, C. Sánchez, M. de la Asunción. *Simulation of landslide-generated tsunamis with the HySEA platform: Application to the Lituya Bay 1958 tsunami*. European Geosciences Union General Assembly 2012 (EGU 2012), Viena (Austria), Abril 2012.

- C. Sánchez, M. J. Castro, J. M. González, M. de la Asunción. *Simulation of landslide-generated tsunamis using IFCP volume scheme. Application to the Lituya Bay 1958 tsunami*. 1st Joint Conference of the Belgian, Royal Spanish and Luxembourg Mathematical Societies (BSL 2012), Lieja (Bélgica), Junio 2012.

- J. M. González, M. de la Asunción, M. J. Castro, E. D. Fernández-Nieto, J. Macías, T. Morales, C. Parés, C. Sánchez. *Simulation of landslide-generated tsunamis with the HySEA platform: the Lituya Bay 1958 event*. SIAM Conference on Nonlinear Waves and Coherent Structures, Seattle (EE.UU.), Junio 2012.

- M. de la Asunción, J. M. Mantas, M. J. Castro, J. M. González. *Multi-GPU simulation of tsunamis generated by submarine landslides*. XIX International Conference on Computational Methods in Water Resources (CMWR 2012), Urbana-Champaign (EE.UU.), Junio 2012.

- M. de la Asunción, J. M. Mantas, M. J. Castro. *Evaluating the impact of cell renumbering of unstructured meshes on the performance of finite volume GPU solvers*. 12th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE 2012), La Manga (España), Julio 2012.

Adicionalmente, también ha dado lugar a la siguiente charla invitada en un congreso internacional:

- M. de la Asunción, J. M. Mantas, M. J. Castro. *Improving PVM finite volume schemes using GPUs. Application to shallow flows.* SIAM Conference on Mathematical & Computational Issues in the Geosciences, Long Beach (EE.UU.), Marzo 2011.

Apéndice A

Cálculos de los Esquemas PVM

Resumen: Este apéndice detalla los cálculos matemáticos que se realizan en los esquemas PVM de orden 1 explicados en la sección 3.4.

A.1. Introducción

Presentamos a continuación un algoritmo secuencial que detalla todos los cálculos matemáticos que se efectúan en los esquemas PVM de orden 1 explicados en la sección 3.4. t_{fin} es el tiempo total de simulación, γ es el parámetro CFL, r es el ratio de densidades, $g = 9.81$ es la gravedad, $\varepsilon = 10^{-10}$ y $\varepsilon_h = 10^{-3}$.

Como se explicó en la sección 3.5.1, cada volumen V_i tiene asociados dos acumuladores: M_i (un vector 6×1) y Z_i (un escalar). Ver dicha sección para más detalles.

Para procesar una arista Γ_{ij} es necesario disponer de dos volúmenes V_i y V_j. En el caso de una arista frontera[1], la aproximación más sencilla consiste en reflejar su volumen adyacente V_i, de forma que $|V_i| = |V_j|$ y $H_i = H_j$. Las condiciones de contorno se implementan del siguiente modo: un borde abierto se consigue asignando $W_j = W_i$, mientras que un borde de tipo pared se consigue asignando $W_j = (h_{1,i}, \quad 0, \quad 0, \quad h_{2,i}, \quad 0, \quad 0)^{\mathrm{T}}$, o bien $W_j = (h_{1,i}, \quad -q_{1,x}, \quad -q_{1,y}, \quad h_{2,i}, \quad -q_{2,x}, \quad -q_{2,y})^{\mathrm{T}}$.

[1] Una arista es frontera si sólo tiene un volumen adyacente

A.2. Función Principal

Función principal

1: $t \leftarrow 0$
2: $\Delta t \leftarrow$ `obtenerDeltaTInicial`()
3: **mientras** $(t < t_{fin})$ **hacer**
4: **para cada** volumen V_i **hacer**
5: $M_i \leftarrow (0,\, 0,\, 0,\, 0,\, 0,\, 0)^{\mathrm{T}}$
6: $Z_i \leftarrow 0$
7: **fin para**
8: **para cada** arista Γ_{ij} **hacer**
9: `procesarArista`(Γ_{ij})
10: **fin para**
11: **para cada** volumen V_i **hacer**
12: $W_i^{n+1} \leftarrow W_i^n + \dfrac{\Delta t}{|V_i|}\, M_i$
13: $W_i^{n+1} \leftarrow$ `filtroEstado`$\big(W_i^{n+1},\ \Delta t\big)$
14: **fin para**
15: $t \leftarrow t + \Delta t$
16: $\Delta t \leftarrow$ `obtenerSiguienteDeltaT`()
17: **fin mientras**

A.3. Obtención de Δt

`obtenerDeltaTInicial`

1: **para cada** volumen V_i **hacer**
2: $Z_i \leftarrow 0$
3: **fin para**
4: **para cada** arista Γ_{ij} **hacer**
5: `procesarAristaDeltaT`(Γ_{ij})
6: **fin para**
7: $\Delta t \leftarrow 10^{30}$
8: **para cada** volumen V_i **hacer**
9: $\Delta t_i \leftarrow 2\,\gamma\,|V_i|\,Z_i^{-1}$
10: $\Delta t \leftarrow$ mín$(\Delta t,\, \Delta t_i)$
11: **fin para**
12: **devolver** Δt

`procesarAristaDeltaT`(Γ_{ij})

$W_i^{rot} \leftarrow$ `rotarEstado`$(W_i,\ \boldsymbol{\eta}_{ij})$
$W_j^{rot} \leftarrow$ `rotarEstado`$(W_j,\ \boldsymbol{\eta}_{ij})$
$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n}) \leftarrow$ `obtenerNormal`$\left(W_i^{rot},\ W_j^{rot}\right)$
$D \leftarrow$ `aproximarAutovalores1D`$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n})$
$\lambda_{max} \leftarrow \text{máx}\left(\left|D_{[1]}\right|,\ \left|D_{[4]}\right|,\ |u_{1,ij,n}|,\ |u_{2,ij,n}|\right)$
si $(\lambda_{max} < \varepsilon_h)$ **entonces**
    $\lambda_{max} \leftarrow \lambda_{max} + \varepsilon_h$
fin si
$Z_i \leftarrow Z_i + |\Gamma_{ij}|\,\lambda_{max}$
$Z_j \leftarrow Z_j + |\Gamma_{ij}|\,\lambda_{max}$

`rotarEstado`$(W,\ \boldsymbol{\eta}_{ij})$

$a \leftarrow W_{[2]}\,\eta_{ij,x} + W_{[3]}\,\eta_{ij,y}$
$b \leftarrow W_{[5]}\,\eta_{ij,x} + W_{[6]}\,\eta_{ij,y}$
$W^{rot} \leftarrow \left(W_{[1]},\quad a,\quad W_{[4]},\quad b\right)^{\mathrm{T}}$
devolver W^{rot}

`aproximarAutovalores1D`$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n})$

$q \leftarrow h_{1,ij}\,u_{1,ij,n} + h_{2,ij}\,u_{2,ij,n}$
$h \leftarrow h_{1,ij} + h_{2,ij}$
$hd \leftarrow \sqrt{h^4 + \left(\text{máx}(h,\,\varepsilon_h)\right)^4}$
$u \leftarrow \dfrac{q\,h\,\sqrt{2}}{hd}$
$uu \leftarrow \dfrac{(h_{1,ij}\,u_{2,ij,n} + h_{2,ij}\,u_{1,ij,n})\,h\,\sqrt{2}}{hd}$
$aux \leftarrow 1 - \dfrac{(u_{1,ij,n} - u_{2,ij,n})^2\,h\,\sqrt{2}}{g\,(1-r)\,hd}$
$cg \leftarrow \sqrt{\dfrac{g\,(1-r)\,h_{1,ij}\,h_{2,ij}\,h\,|aux|\,\sqrt{2}}{hd}}$
$D \leftarrow \left(u - \sqrt{gh},\quad uu - cg,\quad uu + cg,\quad u + \sqrt{gh}\right)^{\mathrm{T}}$
devolver D

`obtenerNormal`$\left(W_i^{rot},\, W_j^{rot}\right)$

1: $h_{1,ij} \leftarrow \dfrac{\left(W_i^{rot}\right)_{[1]} + \left(W_j^{rot}\right)_{[1]}}{2}$

2: $u_{i,n} \leftarrow \dfrac{\left(W_i^{rot}\right)_{[1]} \left(W_i^{rot}\right)_{[2]} \sqrt{2}}{\sqrt{\left(\left(W_i^{rot}\right)_{[1]}\right)^4 + \left(\text{máx}\left(\left(W_i^{rot}\right)_{[1]}, \varepsilon_h\right)\right)^4}}$

3: $u_{j,n} \leftarrow \dfrac{\left(W_j^{rot}\right)_{[1]} \left(W_j^{rot}\right)_{[2]} \sqrt{2}}{\sqrt{\left(\left(W_j^{rot}\right)_{[1]}\right)^4 + \left(\text{máx}\left(\left(W_j^{rot}\right)_{[1]}, \varepsilon_h\right)\right)^4}}$

4: $u_{1,ij,n} \leftarrow \dfrac{u_{i,n}\sqrt{\left(W_i^{rot}\right)_{[1]}} + u_{j,n}\sqrt{\left(W_j^{rot}\right)_{[1]}}}{\sqrt{\left(W_i^{rot}\right)_{[1]}} + \sqrt{\left(W_j^{rot}\right)_{[1]}} + \varepsilon}$

5: $h_{2,ij} \leftarrow \dfrac{\left(W_i^{rot}\right)_{[3]} + \left(W_j^{rot}\right)_{[3]}}{2}$

6: $u_{i,n} \leftarrow \dfrac{\left(W_i^{rot}\right)_{[3]} \left(W_i^{rot}\right)_{[4]} \sqrt{2}}{\sqrt{\left(\left(W_i^{rot}\right)_{[3]}\right)^4 + \left(\text{máx}\left(\left(W_i^{rot}\right)_{[3]}, \varepsilon_h\right)\right)^4}}$

7: $u_{j,n} \leftarrow \dfrac{\left(W_j^{rot}\right)_{[3]} \left(W_j^{rot}\right)_{[4]} \sqrt{2}}{\sqrt{\left(\left(W_j^{rot}\right)_{[3]}\right)^4 + \left(\text{máx}\left(\left(W_j^{rot}\right)_{[3]}, \varepsilon_h\right)\right)^4}}$

8: $u_{2,ij,n} \leftarrow \dfrac{u_{i,n}\sqrt{\left(W_i^{rot}\right)_{[3]}} + u_{j,n}\sqrt{\left(W_j^{rot}\right)_{[3]}}}{\sqrt{\left(W_i^{rot}\right)_{[3]}} + \sqrt{\left(W_j^{rot}\right)_{[3]}} + \varepsilon}$

9: **devolver** $(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n})$

`obtenerSiguienteDeltaT`

1: $\Delta t \leftarrow 10^{30}$
2: **para cada** volumen V_i **hacer**
3: $\quad \Delta t_i \leftarrow 2\,\gamma\,|V_i|\,Z_i^{-1}$
4: $\quad \Delta t \leftarrow \text{mín}(\Delta t,\, \Delta t_i)$
5: **fin para**
6: **devolver** Δt

A.4. Procesamiento de Aristas

procesarArista(Γ_{ij})

$W_i^{rot} \leftarrow$ `rotarEstado`$(W_i,\ \boldsymbol{\eta}_{ij})$
$W_j^{rot} \leftarrow$ `rotarEstado`$(W_j,\ \boldsymbol{\eta}_{ij})$
$(h_{1,ij},\, u_{1,ij,n},\, h_{2,ij},\, u_{2,ij,n}) \leftarrow$ `obtenerNormal`$\big(W_i^{rot},\, W_j^{rot}\big)$
si $(h_{1,ij} \geq \varepsilon)$ **o** $(h_{2,ij} \geq \varepsilon)$ **entonces**
  $P \leftarrow$ `terminosPresion1D`$\big(W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ h_{2,ij}\big)$
  $D \leftarrow$ `aproximarAutovalores1D`$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n})$
  $I \leftarrow$ `modificacionIdentidad`$\big(W_i^{rot},\ W_j^{rot}\big)$
  $\lambda_{max} \leftarrow \text{máx}\left(\left|D_{[1]}\right|,\ \left|D_{[4]}\right|,\ |u_{1,ij,n}|,\ |u_{2,ij,n}|\right)$
  $(F,\, S) \leftarrow$ `PVM-x`$\big(P,\ D,\ I,\ \lambda_{max},\ W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n}\big)$
  $\Phi^- \leftarrow F - S+$ `flujo1DC`$\big(W_i^{rot}\big)$
  $\Phi^+ \leftarrow F + S-$ `flujo1DC`$\big(W_j^{rot}\big)$
  $(p_1,\, p_2) \leftarrow$ `obtenerParametros`$\big(W_i,\, W_j,\, W_i^{rot},\, W_j^{rot},\, \boldsymbol{\eta}_{ij},\, \Phi^-\big)$
  $\mathcal{F}_{ij}^- \leftarrow$ `inversaRotarEstado`$(\Phi^-,\ \boldsymbol{\eta}_{ij},\ p_1,\ p_2)$
  $\mathcal{F}_{ij}^+ \leftarrow$ `inversaRotarEstado`$(\Phi^+,\ \boldsymbol{\eta}_{ij},\ -p_1,\ -p_2)$
  si $(\lambda_{max} < \varepsilon_h)$ **entonces**
    $\lambda_{max} \leftarrow \lambda_{max} + \varepsilon_h$
  fin si
  $M_i \leftarrow M_i - |\Gamma_{ij}|\, \mathcal{F}_{ij}^-$
  $M_j \leftarrow M_j - |\Gamma_{ij}|\, \mathcal{F}_{ij}^+$
  $Z_i \leftarrow Z_i + |\Gamma_{ij}|\, \lambda_{max}$
  $Z_j \leftarrow Z_j + |\Gamma_{ij}|\, \lambda_{max}$
fin si

flujo1DC(W)

si $\big(W_{[1]} < \varepsilon\big)$ **entonces**
  $a \leftarrow 0$
si no
  $$a \leftarrow \frac{W_{[1]}\, W_{[2]}\, W_{[2]}\, \sqrt{2}}{\sqrt{\big(W_{[1]}\big)^4 + \Big(\text{máx}\big(W_{[1]},\, \varepsilon_h\big)\Big)^4}}$$
fin si
si $\big(W_{[3]} < \varepsilon\big)$ **entonces**
  $b \leftarrow 0$
si no
  $$b \leftarrow \frac{W_{[3]}\, W_{[4]}\, W_{[4]}\, \sqrt{2}}{\sqrt{\big(W_{[3]}\big)^4 + \Big(\text{máx}\big(W_{[3]},\, \varepsilon_h\big)\Big)^4}}$$
fin si
$F \leftarrow \big(W_{[2]},\quad a,\quad W_{[4]},\quad b\big)^{\mathrm{T}}$
devolver F

`terminosPresion1D`$\left(W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ h_{2,ij}\right)$

$H_m \leftarrow \text{mín}(H_i,\ H_j)$
$h_0 \leftarrow \text{máx}\left(\left(W_i^{rot}\right)_{[1]} + \left(W_i^{rot}\right)_{[3]} - H_i + H_m,\ 0\right)$
$h_1 \leftarrow \text{máx}\left(\left(W_j^{rot}\right)_{[1]} + \left(W_j^{rot}\right)_{[3]} - H_j + H_m,\ 0\right)$
$deta_1 \leftarrow h_1 - h_0$
$h_0 \leftarrow \text{máx}\left(\left(W_i^{rot}\right)_{[3]} - H_i + H_m,\ 0\right)$
$h_1 \leftarrow \text{máx}\left(\left(W_j^{rot}\right)_{[3]} - H_j + H_m,\ 0\right)$
$deta_2 \leftarrow h_1 - h_0$
$P \leftarrow \left(0, \quad g\, h_{1,ij}\, deta_1, \quad 0, \quad g\, h_{2,ij}\, ((1-r)\, deta_2 + r\, deta_1)\right)^{\mathrm{T}}$
devolver P

`modificacionIdentidad`$\left(W_i^{rot},\ W_j^{rot}\right)$

$H_m \leftarrow \text{mín}(H_i,\ H_j)$
$h_0 \leftarrow \text{máx}\left(\left(W_i^{rot}\right)_{[1]} + \left(W_i^{rot}\right)_{[3]} - H_i + H_m,\ 0\right)$
$h_1 \leftarrow \text{máx}\left(\left(W_j^{rot}\right)_{[1]} + \left(W_j^{rot}\right)_{[3]} - H_j + H_m,\ 0\right)$
$deta_1 \leftarrow h_1 - h_0$
$h_0 \leftarrow \text{máx}\left(\left(W_i^{rot}\right)_{[3]} - H_i + H_m,\ 0\right)$
$h_1 \leftarrow \text{máx}\left(\left(W_j^{rot}\right)_{[3]} - H_j + H_m,\ 0\right)$
$deta_2 \leftarrow h_1 - h_0$
$dq_1 \leftarrow \left(W_j^{rot}\right)_{[2]} - \left(W_i^{rot}\right)_{[2]}$
$dq_2 \leftarrow \left(W_j^{rot}\right)_{[4]} - \left(W_i^{rot}\right)_{[4]}$
si $\left(\left(W_i^{rot}\right)_{[1]} \geq \varepsilon_h\right)$ **y** $\left(\left(W_j^{rot}\right)_{[1]} \geq \varepsilon_h\right)$ **entonces**
    $I \leftarrow \left(deta_1 - deta_2, \quad dq_1, \quad deta_2, \quad dq_2\right)^{\mathrm{T}}$
si no
    $I \leftarrow \left(deta_1, \quad dq_1, \quad deta_2, \quad dq_2\right)^{\mathrm{T}}$
fin si
devolver I

`inversaRotarEstado`$(\Phi,\ \boldsymbol{\eta}_{ij},\ p_1,\ p_2)$

$a \leftarrow \Phi_{[2]}\, \eta_{ij,x} - p_1\, \eta_{ij,y}$
$b \leftarrow \Phi_{[2]}\, \eta_{ij,y} + p_1\, \eta_{ij,x}$
$c \leftarrow \Phi_{[4]}\, \eta_{ij,x} - p_2\, \eta_{ij,y}$
$d \leftarrow \Phi_{[4]}\, \eta_{ij,y} + p_2\, \eta_{ij,x}$
$F \leftarrow \left(\Phi_{[1]}, \quad a, \quad b, \quad \Phi_{[3]}, \quad c, \quad d\right)^{\mathrm{T}}$
devolver F

`obtenerParametros`$\left(W_i,\, W_j,\, W_i^{rot},\, W_j^{rot},\, \boldsymbol{\eta}_{ij},\, \Phi^-\right)$

1: **si** $\left(\left(W_i^{rot}\right)_{[1]} < \varepsilon_h\right)$ **entonces**
2: $\quad u_{1,i,t} \leftarrow 0$
3: **si no**
4: $\quad u_{1,i,t} \leftarrow \dfrac{(W_i)_{[3]}\,\eta_{ij,x} - (W_i)_{[2]}\,\eta_{ij,y}}{\left(W_i^{rot}\right)_{[1]}}$
5: **fin si**
6: **si** $\left(\left(W_j^{rot}\right)_{[1]} < \varepsilon_h\right)$ **entonces**
7: $\quad u_{1,j,t} \leftarrow 0$
8: **si no**
9: $\quad u_{1,j,t} \leftarrow \dfrac{(W_j)_{[3]}\,\eta_{ij,x} - (W_j)_{[2]}\,\eta_{ij,y}}{\left(W_j^{rot}\right)_{[1]}}$
10: **fin si**
11: **si** $\left((\Phi^-)_{[1]} > 0\right)$ **entonces**
12: $\quad u_{1,ij,t} \leftarrow u_{1,i,t}$
13: **si no**
14: $\quad u_{1,ij,t} \leftarrow u_{1,j,t}$
15: **fin si**
16: **si** $\left(\left(W_i^{rot}\right)_{[3]} < \varepsilon_h\right)$ **entonces**
17: $\quad u_{2,i,t} \leftarrow 0$
18: **si no**
19: $\quad u_{2,i,t} \leftarrow \dfrac{(W_i)_{[6]}\,\eta_{ij,x} - (W_i)_{[5]}\,\eta_{ij,y}}{\left(W_i^{rot}\right)_{[3]}}$
20: **fin si**
21: **si** $\left(\left(W_j^{rot}\right)_{[3]} < \varepsilon_h\right)$ **entonces**
22: $\quad u_{2,j,t} \leftarrow 0$
23: **si no**
24: $\quad u_{2,j,t} \leftarrow \dfrac{(W_j)_{[6]}\,\eta_{ij,x} - (W_j)_{[5]}\,\eta_{ij,y}}{\left(W_j^{rot}\right)_{[3]}}$
25: **fin si**
26: **si** $\left((\Phi^-)_{[3]} > 0\right)$ **entonces**
27: $\quad u_{2,ij,t} \leftarrow u_{2,i,t}$
28: **si no**
29: $\quad u_{2,ij,t} \leftarrow u_{2,j,t}$
30: **fin si**
31: $p_1 \leftarrow u_{1,ij,t}\,(\Phi^-)_{[1]}$
32: $p_2 \leftarrow u_{2,ij,t}\,(\Phi^-)_{[3]}$
33: **devolver** $(p_1,\ p_2)$

La función `PVM-x` de la línea 9 de la función `procesarArista` es una de las siguientes: `PVM-0`, `PVM-1U`, `PVM-2`, `PVM-2U`, `PVM-4` o `PVM-IFCP`. Es decir, para ejecutar un determinado esquema PVM sólo hay que considerar la función `PVM-x` correspondiente, siendo el resto del algoritmo igual para todos los esquemas PVM. A continuación se definen estas funciones.

`PVM-0`$\big(P,\ D,\ I,\ \lambda_{max},\ W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n}\big)$

1: $a_0 \leftarrow \lambda_{max}$
2: $F \leftarrow \dfrac{1}{2}\Big(\texttt{flujo1DC}\big(W_j^{rot}\big) - \texttt{flujo1DC}\big(W_i^{rot}\big) + P\Big)$
3: $S \leftarrow \dfrac{1}{2}\, a_0\, I$
4: **devolver** $(F,\, S)$

`PVM-1U`$\big(P,\ D,\ I,\ \lambda_{max},\ W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n}\big)$

1: $a_0 \leftarrow \dfrac{D_{[4]}\,\big|D_{[1]}\big| - D_{[1]}\,\big|D_{[4]}\big|}{D_{[4]} - D_{[1]}}$
2: $a_1 \leftarrow \dfrac{\big|D_{[4]}\big| - \big|D_{[1]}\big|}{D_{[4]} - D_{[1]}}$
3: $F \leftarrow \texttt{flujo1DC}\big(W_j^{rot}\big) - \texttt{flujo1DC}\big(W_i^{rot}\big) + P$
4: $S \leftarrow \dfrac{1}{2}\,(a_0\, I + a_1\, F)$
5: $F \leftarrow \dfrac{1}{2}\, F$
6: **devolver** $(F,\, S)$

`PVM-2U`$\big(P,\ D,\ I,\ \lambda_{max},\ W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n}\big)$

1: **si** $\big(D_{[1]} \geq D_{[4]}\big)$ **entonces**
2: $\quad S_L \leftarrow D_{[1]}$
3: $\quad S_R \leftarrow D_{[4]}$
4: **si no**
5: $\quad S_L \leftarrow D_{[4]}$
6: $\quad S_R \leftarrow D_{[1]}$
7: **fin si**
8: $b \leftarrow (S_R - S_L)^2$
9: $a_1 \leftarrow \dfrac{S_L(|S_L| - |S_R|) + S_R\big(\mathrm{sgn}(S_L)S_R - S_L\mathrm{sgn}(S_R)\big)}{b}$
10: $a_2 \leftarrow \dfrac{S_R\big(\mathrm{sgn}(S_R) - \mathrm{sgn}(S_L)\big)}{b}$
11: $a_0 \leftarrow S_L\, S_L\, a_2$
12: $F \leftarrow \texttt{flujo1DC}\big(W_j^{rot}\big) - \texttt{flujo1DC}\big(W_i^{rot}\big) + P$
13: $a \leftarrow (g\, h_{1,ij} - u_{1,ij,n}\, u_{1,ij,n})\, F_{[1]} + 2\, u_{1,ij,n}\, F_{[2]} + g\, h_{1,ij}\, F_{[3]}$
14: $b \leftarrow r\, g\, h_{2,ij}\, F_{[1]} + (g\, h_{2,ij} - u_{2,ij,n}\, u_{2,ij,n})\, F_{[3]} + 2\, u_{2,ij,n}\, F_{[4]}$
15: $S \leftarrow \big(F_{[2]},\quad a,\quad F_{[4]},\quad b\big)^{\mathrm{T}}$
16: $S \leftarrow \dfrac{1}{2}\,(a_0\, I + a_1\, F + a_2\, S)$
17: $F \leftarrow \dfrac{1}{2}\, F$
18: **devolver** $(F,\, S)$

`PVM-2`$\big(P,\ D,\ I,\ \lambda_{max},\ W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n}\big)$

1: $a_0 \leftarrow \dfrac{\lambda_{max}}{2}$
2: $a_2 \leftarrow \dfrac{1}{2\,\lambda_{max}}$
3: $F \leftarrow \texttt{flujo1DC}\big(W_j^{rot}\big) - \texttt{flujo1DC}\big(W_i^{rot}\big) + P$
4: $a \leftarrow (g\,h_{1,ij} - u_{1,ij,n}\,u_{1,ij,n})\,F_{[1]} + 2\,u_{1,ij,n}\,F_{[2]} + g\,h_{1,ij}\,F_{[3]}$
5: $b \leftarrow r\,g\,h_{2,ij}\,F_{[1]} + (g\,h_{2,ij} - u_{2,ij,n}\,u_{2,ij,n})\,F_{[3]} + 2\,u_{2,ij,n}\,F_{[4]}$
6: $S \leftarrow \big(F_{[2]},\quad a,\quad F_{[4]},\quad b\big)^{\mathrm{T}}$
7: $S \leftarrow \dfrac{1}{2}\,(a_0\,I + a_2\,S)$
8: $F \leftarrow \dfrac{1}{2}\,F$
9: **devolver** $(F,\,S)$

`PVM-4`$\big(P,\ D,\ I,\ \lambda_{max},\ W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n}\big)$

1: $a_0 \leftarrow \dfrac{3\,\lambda_{max}}{8}$
2: $a_2 \leftarrow \dfrac{3}{4\,\lambda_{max}}$
3: $a_4 \leftarrow \dfrac{-1}{8\,(\lambda_{max})^3}$
4: $F \leftarrow \texttt{flujo1DC}\big(W_j^{rot}\big) - \texttt{flujo1DC}\big(W_i^{rot}\big) + P$
5: $a \leftarrow (g\,h_{1,ij} - u_{1,ij,n}\,u_{1,ij,n})\,F_{[1]} + 2\,u_{1,ij,n}\,F_{[2]} + g\,h_{1,ij}\,F_{[3]}$
6: $b \leftarrow r\,g\,h_{2,ij}\,F_{[1]} + (g\,h_{2,ij} - u_{2,ij,n}\,u_{2,ij,n})\,F_{[3]} + 2\,u_{2,ij,n}\,F_{[4]}$
7: $S \leftarrow \big(F_{[2]},\quad a,\quad F_{[4]},\quad b\big)^{\mathrm{T}}$
8: $a \leftarrow (g\,h_{1,ij} - u_{1,ij,n}\,u_{1,ij,n})\,S_{[1]} + 2\,u_{1,ij,n}\,S_{[2]} + g\,h_{1,ij}\,S_{[3]}$
9: $b \leftarrow r\,g\,h_{2,ij}\,S_{[1]} + (g\,h_{2,ij} - u_{2,ij,n}\,u_{2,ij,n})\,S_{[3]} + 2\,u_{2,ij,n}\,S_{[4]}$
10: $T \leftarrow \big(S_{[2]},\quad a,\quad S_{[4]},\quad b\big)^{\mathrm{T}}$
11: $a \leftarrow (g\,h_{1,ij} - u_{1,ij,n}\,u_{1,ij,n})\,T_{[1]} + 2\,u_{1,ij,n}\,T_{[2]} + g\,h_{1,ij}\,T_{[3]}$
12: $b \leftarrow r\,g\,h_{2,ij}\,T_{[1]} + (g\,h_{2,ij} - u_{2,ij,n}\,u_{2,ij,n})\,T_{[3]} + 2\,u_{2,ij,n}\,T_{[4]}$
13: $T \leftarrow \big(T_{[2]},\quad a,\quad T_{[4]},\quad b\big)^{\mathrm{T}}$
14: $S \leftarrow \dfrac{1}{2}\,(a_0\,I + a_2\,S + a_4\,T)$
15: $F \leftarrow \dfrac{1}{2}\,F$
16: **devolver** $(F,\,S)$

`PVM-IFCP`$\left(P,\ D,\ I,\ \lambda_{max},\ W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n}\right)$

$\lambda_2 \leftarrow \text{máx}\left(\left|D_{[2]}\right|,\ \left|D_{[3]}\right|\right)$
si $\left(\text{sgn}\left(D_{[1]} + D_{[4]}\right) < 0\right)$ **entonces**
  $\lambda_2 \leftarrow -\lambda_2$
fin si
$a \leftarrow \dfrac{\left|D_{[1]}\right|}{\left(D_{[1]} - D_{[4]}\right)\left(D_{[1]} - \lambda_2\right)}$
$b \leftarrow \dfrac{\left|\lambda_2\right|}{\left(\lambda_2 - D_{[1]}\right)\left(\lambda_2 - D_{[4]}\right)}$
$c \leftarrow \dfrac{\left|D_{[4]}\right|}{\left(D_{[4]} - D_{[1]}\right)\left(D_{[4]} - \lambda_2\right)}$
$a_0 \leftarrow a\,\lambda_2\,D_{[4]} + b\,D_{[1]}\,D_{[4]} + c\,D_{[1]}\,\lambda_2$
$a_1 \leftarrow -a\left(\lambda_2 + D_{[4]}\right) - b\left(D_{[1]} + D_{[4]}\right) - c\left(D_{[1]} + \lambda_2\right)$
$a_2 \leftarrow a + b + c$
$F \leftarrow \texttt{flujo1DC}\left(W_j^{rot}\right) - \texttt{flujo1DC}\left(W_i^{rot}\right) + P$
$a \leftarrow \left(g\,h_{1,ij} - u_{1,ij,n}\,u_{1,ij,n}\right)F_{[1]} + 2\,u_{1,ij,n}\,F_{[2]} + g\,h_{1,ij}\,F_{[3]}$
$b \leftarrow r\,g\,h_{2,ij}\,F_{[1]} + \left(g\,h_{2,ij} - u_{2,ij,n}\,u_{2,ij,n}\right)F_{[3]} + 2\,u_{2,ij,n}\,F_{[4]}$
$S \leftarrow \left(F_{[2]},\quad a,\quad F_{[4]},\quad b\right)^{\mathrm{T}}$
$S \leftarrow \dfrac{1}{2}\left(a_0\,I + a_1\,F + a_2\,S\right)$
$F \leftarrow \dfrac{1}{2}\,F$
devolver $(F,\ S)$

A.5. Procesamiento de Volúmenes

La función `filtroEstado` aplica un filtrado adicional al estado W_i^{n+1}. En esta función se corrigen los valores h_1 y h_2 en caso de ser negativos, y también se limita la velocidad de cada capa. Esto último se realiza a partir de dos valores proporcionados por el usuario: v_{max1} (velocidad máxima de la capa 1) y v_{max2} (velocidad máxima de la capa 2). La función se define del siguiente modo:

`filtroEstado`$(W,\, \Delta t)$

si $\big(W_{[1]} < 0\big)$ **entonces**
  $W_{[1]} \leftarrow 0$
fin si
si $\big(W_{[4]} < 0\big)$ **entonces**
  $W_{[4]} \leftarrow 0$
fin si
$a = \dfrac{1}{\left(\frac{W_{[1]}}{\varepsilon_h}\right)^4 + \varepsilon}$
$b = e^{-a\,\Delta t}$
$W_{[2,3]} \leftarrow b\, W_{[2,3]}$
$a = \dfrac{1}{\left(\frac{W_{[4]}}{\varepsilon_h}\right)^4 + \varepsilon}$
$b = e^{-a\,\Delta t}$
$W_{[5,6]} \leftarrow b\, W_{[5,6]}$
$W \leftarrow$ `tratamientoComplejos`(W)
si $(v_{max1} > 0)$ **entonces**
  $W_{[1,2,3]} \leftarrow$ `ajustarVelocidadCapa`$\big(W_{[1,2,3]},\, v_{max1}\big)$
fin si
si $(v_{max2} > 0)$ **entonces**
  $W_{[4,5,6]} \leftarrow$ `ajustarVelocidadCapa`$\big(W_{[4,5,6]},\, v_{max2}\big)$
fin si
devolver W

`ajustarVelocidadCapa`$\big(W,\ v_{max}\big)$

$h_m \leftarrow \sqrt{\big(W_{[1]}\big)^4 + \Big(\text{máx}\,\big(W_{[1]},\, \varepsilon_h\big)\Big)^4}$
$u_x \leftarrow \dfrac{W_{[1]}\, W_{[2]}\, \sqrt{2}}{h_m}$
$u_y \leftarrow \dfrac{W_{[1]}\, W_{[3]}\, \sqrt{2}}{h_m}$
$u \leftarrow \sqrt{(u_x)^2 + (u_y)^2}$
si $(u > v_{max})$ **entonces**
  $W_{[2]} \leftarrow \dfrac{W_{[1]}\, u_x\, v_{max}}{u}$
  $W_{[3]} \leftarrow \dfrac{W_{[1]}\, u_y\, v_{max}}{u}$
fin si
devolver W

`tratamientoComplejos`(W)

1: $h_{1m} \leftarrow \sqrt{\left(W_{[1]}\right)^4 + \Big(\text{máx}\left(W_{[1]},\, \varepsilon_h\right)\Big)^4}$
2: $h_{2m} \leftarrow \sqrt{\left(W_{[4]}\right)^4 + \Big(\text{máx}\left(W_{[4]},\, \varepsilon_h\right)\Big)^4}$
3: $u_{1,x} \leftarrow \dfrac{W_{[1]}\, W_{[2]}\, \sqrt{2}}{h_{1m}}$
4: $u_{1,y} \leftarrow \dfrac{W_{[1]}\, W_{[3]}\, \sqrt{2}}{h_{1m}}$
5: $u_{2,x} \leftarrow \dfrac{W_{[4]}\, W_{[5]}\, \sqrt{2}}{h_{2m}}$
6: $u_{2,y} \leftarrow \dfrac{W_{[4]}\, W_{[6]}\, \sqrt{2}}{h_{2m}}$
7: $u_1 \leftarrow \sqrt{(u_{1,x})^2 + (u_{1,y})^2}$
8: $u_2 \leftarrow \sqrt{(u_{2,x})^2 + (u_{2,y})^2}$
9: $du \leftarrow \sqrt{(u_{1,x} - u_{2,x})^2 + (u_{1,y} - u_{2,y})^2}$
10: $h_{mod} \leftarrow W_{[4]} - r\, W_{[1]}$
11: $h_{modm} \leftarrow \sqrt{(h_{mod})^4 + \big(\text{máx}(h_{mod},\, \varepsilon_h\,(1+r))\big)^4}$
12: $cf \leftarrow \sqrt{\dfrac{\left(W_{[1]} + W_{[4]}\right) du^2\, \sqrt{2}}{g\,(1-r)\sqrt{\left(W_{[1]} + W_{[4]}\right)^4 + \Big(\text{máx}\left(W_{[1]} + W_{[4]},\, 2\,\varepsilon_h\right)\Big)^4}}}$
13: **si** $(cf > 1)$ **y** $\left(W_{[1]} > 0\right)$ **y** $\left(W_{[4]} > 0\right)$ **entonces**
14: $\quad c_1 \leftarrow \dfrac{\text{máx}(cf - 1,\, 0)\, W_{[4]}\, h_{mod}\, \sqrt{2}}{h_{modm}}$
15: $\quad c_2 \leftarrow \dfrac{\text{máx}(cf - 1,\, 0)\, r\, W_{[1]}\, h_{mod}\, \sqrt{2}}{h_{modm}}$
16: $\quad det \leftarrow (1 + c_1)\,(1 + c_2) - c_1\, c_2$
17: $\quad u_{1,n} \leftarrow \dfrac{u_1\,(1 + c_2) + c_1\, u_2}{det}$
18: $\quad u_{2,n} \leftarrow \dfrac{u_2\,(1 + c_1) + c_2\, u_1}{det}$
19: $\quad W_{[2]} \leftarrow \dfrac{W_{[1]}\, u_{1,x}\, u_{1,n}}{u_1 + \varepsilon}$
20: $\quad W_{[3]} \leftarrow \dfrac{W_{[1]}\, u_{1,y}\, u_{1,n}}{u_1 + \varepsilon}$
21: $\quad W_{[5]} \leftarrow \dfrac{W_{[4]}\, u_{2,x}\, u_{2,n}}{u_2 + \varepsilon}$
22: $\quad W_{[6]} \leftarrow \dfrac{W_{[4]}\, u_{2,y}\, u_{2,n}}{u_2 + \varepsilon}$
23: **fin si**
24: **devolver** W

Apéndice B

Cálculos del Esquema de Simulación de Tsunamis

Resumen: Este apéndice detalla los cálculos matemáticos que se realizan en el esquema de orden 1 para la simulación de tsunamis descrito en el capítulo 6.

B.1. Introducción

Presentamos a continuación un algoritmo secuencial que detalla todos los cálculos matemáticos que se efectúan en el esquema de orden 1 para la simulación de tsunamis generados por avalanchas submarinas y subaéreas explicado en el capítulo 6. t_{fin} es el tiempo total de simulación, γ es el parámetro CFL, r es el ratio de densidades, g es la gravedad, m_f es el coeficiente de fricción entre las dos capas, n_1 es el coeficiente de Manning para la fricción entre el agua y el fondo, n_2 es el coeficiente de Manning para la fricción entre el sedimento y el fondo, v_{max1} es la velocidad máxima de la capa 1, v_{max2} es la velocidad máxima de la capa 2, $\varepsilon = 10^{-10}$ y $\varepsilon_h = 5 \cdot 10^{-3}$.

En caso de considerar la ley de fricción de Coulomb se utiliza un ángulo de reposo α, mientras que si se considera la ley de fricción de Pouliquen se utilizan cuatro ángulos de reposo $\alpha_1, \ldots, \alpha_4$.

L, H, Q y T son los parámetros de normalización de los datos. L normaliza las coordenadas espaciales de los ejes x e y, H normaliza el eje z (las alturas), Q los caudales y T el tiempo. En caso de normalizar, el usuario debe proporcionar los parámetros L y H.

B.2. Función Principal

Función principal

1: `normalizarDatos`()
2: $t \leftarrow 0$
3: $\Delta t \leftarrow$ `obtenerDeltaTInicial`()
4: **mientras** $(t < t_{fin})$ **hacer**
5: **para cada** volumen V_i **hacer**
6: $M_i \leftarrow (0,\ 0,\ 0,\ 0,\ 0,\ 0)^{\mathrm{T}}$
7: $Z_i \leftarrow 0$
8: **fin para**
9: **para cada** arista Γ_{ij} **hacer**
10: `procesarArista`$(\Gamma_{ij},\ \Delta t)$
11: **fin para**
12: **para cada** volumen V_i **hacer**
13: $W_i^{n+1} \leftarrow W_i^n + \dfrac{\Delta t}{|V_i|}\, M_i$
14: $W_i^{n+1} \leftarrow$ `filtroEstado`$\big(W_i^{n+1},\ \Delta t\big)$
15: $W_i^{n+1} \leftarrow$ `discretizacionImplicita`$\big(W_i^n,\ W_i^{n+1},\ \Delta t\big)$
16: $W_i^{n+1} \leftarrow$ `coulomb`$\big(W_i^{n+1}, \Delta t\big)$
17: **fin para**
18: $t \leftarrow t + \Delta t$
19: $\Delta t \leftarrow$ `obtenerSiguienteDeltaT`()
20: **fin mientras**

B.3. Normalización de Datos

`normalizarDatos`

1: **si** normalizar **entonces**
2: $g \leftarrow 1$
3: $Q \leftarrow \sqrt{9.81\, H^3}$
4: $T \leftarrow \dfrac{L\, H}{Q}$
5: $t_{fin} \leftarrow \dfrac{t_{fin}}{T}$
6: $m_f \leftarrow m_f\, L$
7: $n_1 \leftarrow \dfrac{Q\sqrt{L}}{H^{13/6}}\, n_1$
8: $n_2 \leftarrow \dfrac{Q\sqrt{L}}{H^{13/6}}\, n_2$

$v_{max1} \leftarrow \dfrac{H}{Q}\, v_{max1}$
$v_{max2} \leftarrow \dfrac{H}{Q}\, v_{max2}$
$\varepsilon_h \leftarrow \dfrac{\varepsilon_h}{H}$
para cada volumen V_i **hacer**
  $|V_i| \leftarrow \dfrac{|V_i|}{L^2}$
  $H_i \leftarrow \dfrac{H_i}{H}$
  $(W_i)_{[1,4]} \leftarrow \dfrac{1}{H}\,(W_i)_{[1,4]}$
  $(W_i)_{[2,3,5,6]} \leftarrow \dfrac{1}{Q}\,(W_i)_{[2,3,5,6]}$
fin para
si no
  $g \leftarrow 9.81$
  $L \leftarrow H \leftarrow Q \leftarrow T \leftarrow 1$
fin si

B.4. Obtención de Δt

`procesarAristaDeltaT`(Γ_{ij})

$W_i^{rot} \leftarrow$ `rotarEstado`($W_i,\ \boldsymbol{\eta}_{ij}$)
$W_j^{rot} \leftarrow$ `rotarEstado`($W_j,\ \boldsymbol{\eta}_{ij}$)
$(h_{1,ij},\, u_{1,ij,n},\, h_{2,ij},\, u_{2,ij,n}) \leftarrow$ `obtenerNormal`$\left(W_i^{rot},\, W_j^{rot}\right)$
$D \leftarrow$ `aproxAutovalores1D`$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n},\ 0)$
$\lambda_{max} \leftarrow \text{máx}\left(\left|D_{[1]}\right|,\, \left|D_{[4]}\right|,\, |u_{1,ij,n}|,\, |u_{2,ij,n}|\right)$
si $(\lambda_{max} < \varepsilon_h)$ **entonces**
  $\lambda_{max} \leftarrow \lambda_{max} + \varepsilon_h$
fin si
$Z_i \leftarrow Z_i + |\Gamma_{ij}|\, \lambda_{max}$
$Z_j \leftarrow Z_j + |\Gamma_{ij}|\, \lambda_{max}$

`aproxAutovalores1D`$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n},\ flag)$

si $flag = 0$ **entonces**
  $q \leftarrow h_{1,ij}\, u_{1,ij,n} + h_{2,ij}\, u_{2,ij,n}$
si no
  $q \leftarrow u_{1,ij,n} + u_{2,ij,n}$
fin si
$h \leftarrow h_{1,ij} + h_{2,ij}$
$hd \leftarrow \sqrt{h^4 + \left(\text{máx}(h,\, \varepsilon_h)\right)^4}$
$u \leftarrow \dfrac{q\, h\, \sqrt{2}}{hd}$
si $flag = 0$ **entonces**
  $uu \leftarrow \dfrac{(h_{1,ij}\, u_{2,ij,n} + h_{2,ij}\, u_{1,ij,n})\, h\, \sqrt{2}}{hd}$
  $u_1 \leftarrow u_{1,ij,n}$
  $u_2 \leftarrow u_{2,ij,n}$
si no
  $u_1 \leftarrow \dfrac{(h_{1,ij}\, u_{1,ij,n})\, \sqrt{2}}{\sqrt{\left(h_{1,ij}\right)^4 + \left(\text{máx}\left(h_{1,ij},\, \varepsilon_h\right)\right)^4}}$
  $u_2 \leftarrow \dfrac{(h_{2,ij}\, u_{2,ij,n})\, \sqrt{2}}{\sqrt{\left(h_{2,ij}\right)^4 + \left(\text{máx}\left(h_{2,ij},\, \varepsilon_h\right)\right)^4}}$
  $uu \leftarrow \dfrac{(h_{2,ij}\, u_1 + h_{1,ij}\, u_2)\, h\, \sqrt{2}}{hd}$
fin si
$aux \leftarrow 1 - \dfrac{(u_{1,ij,n} - u_{2,ij,n})^2\, h\, \sqrt{2}}{g\,(1-r)\, hd}$
$cg \leftarrow \sqrt{\dfrac{g\,(1-r)\, h_{1,ij}\, h_{2,ij}\, h\, |aux|\, \sqrt{2}}{hd}}$
$D \leftarrow \left(u - \sqrt{gh},\quad uu - cg,\quad uu + cg,\quad u + \sqrt{gh}\right)^{\text{T}}$
para $i = 1$ **hasta** 3 **hacer**
  para $j = 2$ **hasta** 4 **hacer**
    si $\left(D_{[i]} > D_{[j]}\right)$ **entonces**
      $a \leftarrow D_{[j]}$
      $D_{[j]} \leftarrow D_{[i]}$
      $D_{[i]} \leftarrow a$
    fin si
  fin para
fin para
devolver D

Las funciones `obtenerDeltaTInicial`, `rotarEstado`, `obtenerNormal` y `obtenerSiguienteDeltaT` son idénticas a las del Apéndice A.

B.5. Procesamiento de Aristas

`procesarArista` $(\Gamma_{ij},\ \Delta t)$

$W_i^{rot} \leftarrow$ `rotarEstado`$(W_i,\ \boldsymbol{\eta}_{ij})$
$W_j^{rot} \leftarrow$ `rotarEstado`$(W_j,\ \boldsymbol{\eta}_{ij})$
$\left(W_i^{rot},\ W_j^{rot}\right) \leftarrow$ `tratamientoSecoMojado`$\left(W_i^{rot},\ W_j^{rot}\right)$
$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n}) \leftarrow$ `obtenerNormal`$\left(W_i^{rot},\ W_j^{rot}\right)$
si $(h_{1,ij} \geq \varepsilon)$ **o** $(h_{2,ij} \geq \varepsilon)$ **entonces**
  $P \leftarrow$ `terminosPresion1D`$\left(W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ h_{2,ij}\right)$
  $P_m \leftarrow$ `terminosPresion1DMod`$\left(W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n},\ \Delta t\right)$
  $D_1 \leftarrow$ `aproxAutovalores1D`$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n},\ 0)$
  $D_2 \leftarrow$ `aproxAutovalores1D`$\left(\left(W_i^{rot}\right)_{[1]},\ \left(W_i^{rot}\right)_{[2]},\ \left(W_i^{rot}\right)_{[3]},\ \left(W_i^{rot}\right)_{[4]},\ 1\right)$
  $D_3 \leftarrow$ `aproxAutovalores1D`$\left(\left(W_j^{rot}\right)_{[1]},\ \left(W_j^{rot}\right)_{[2]},\ \left(W_j^{rot}\right)_{[3]},\ \left(W_j^{rot}\right)_{[4]},\ 1\right)$
  $I \leftarrow$ `modificacionIdentidad`$\left(W_i^{rot},\ W_j^{rot}\right)$
  $\lambda_1 \leftarrow \text{mín}\left(\left|(D_1)_{[1]}\right|,\ \left|(D_2)_{[1]}\right|,\ \left|(D_3)_{[1]}\right|\right)$
  $\lambda_4 \leftarrow \text{máx}\left(\left|(D_1)_{[4]}\right|,\ \left|(D_2)_{[4]}\right|,\ \left|(D_3)_{[4]}\right|\right)$
  $\lambda_{max} \leftarrow \text{máx}\left(|\lambda_1|,\ |\lambda_4|,\ |u_{1,ij,n}|,\ |u_{2,ij,n}|\right)$
  $(a_0,\ a_1,\ a_2) \leftarrow$ `obtenerCoeficientes`$(h_{1,ij},\ h_{2,ij},\ D_1,\ \lambda_1,\ \lambda_4,\ \lambda_{max})$
  $F \leftarrow$ `flujo1DC`$\left(W_j^{rot}\right) -$ `flujo1DC`$\left(W_i^{rot}\right)$
  $P_m \leftarrow P_m + F$
  $F \leftarrow \dfrac{1}{2}\left(F + P\right)$
  $a \leftarrow \left(g\, h_{1,ij} - u_{1,ij,n}\, u_{1,ij,n}\right)(P_m)_{[1]} + 2\, u_{1,ij,n}\,(P_m)_{[2]} + g\, h_{1,ij}\,(P_m)_{[3]}$
  $b \leftarrow r\, g\, h_{2,ij}\,(P_m)_{[1]} + \left(g\, h_{2,ij} - u_{2,ij,n}\, u_{2,ij,n}\right)(P_m)_{[3]} + 2\, u_{2,ij,n}\,(P_m)_{[4]}$
  $S \leftarrow \left((P_m)_{[2]},\quad a,\quad (P_m)_{[4]},\quad b\right)^{\mathrm{T}}$
  $S \leftarrow \dfrac{1}{2}\left(a_0\, I + a_1\, P_m + a_2\, S\right)$
  $\Phi^- \leftarrow F - S +$ `flujo1DC`$\left(W_i^{rot}\right)$
  $\Phi^+ \leftarrow F + S -$ `flujo1DC`$\left(W_j^{rot}\right)$
  $(p_1,\ p_2) \leftarrow$ `obtenerParametros`$\left(W_i,\ W_j,\ W_i^{rot},\ W_j^{rot},\ \boldsymbol{\eta}_{ij},\ \Phi^-\right)$
  $\mathcal{F}_{ij}^- \leftarrow$ `inversaRotarEstado`$(\Phi^-,\ \boldsymbol{\eta}_{ij},\ p_1,\ p_2)$
  $\mathcal{F}_{ij}^+ \leftarrow$ `inversaRotarEstado`$(\Phi^+,\ \boldsymbol{\eta}_{ij},\ -p_1,\ -p_2)$
  $(\mathcal{F}_{ij}^-,\ \mathcal{F}_{ij}^+) \leftarrow$ `positividad`$(W_i,\ W_j,\ \mathcal{F}_{ij}^-,\ \mathcal{F}_{ij}^+,\ \Delta t)$
  si $(\lambda_{max} < \varepsilon_h)$ **entonces**
    $\lambda_{max} \leftarrow \lambda_{max} + \varepsilon_h$
  fin si
  $M_i \leftarrow M_i - |\Gamma_{ij}|\,\mathcal{F}_{ij}^-$
  $M_j \leftarrow M_j - |\Gamma_{ij}|\,\mathcal{F}_{ij}^+$
  $Z_i \leftarrow Z_i + |\Gamma_{ij}|\,\lambda_{max}$
  $Z_j \leftarrow Z_j + |\Gamma_{ij}|\,\lambda_{max}$
fin si

`tratamientoSecoMojado`$\left(W_i^{rot},\, W_j^{rot}\right)$

si $\left(\left(W_i^{rot}\right)_{[3]} < \varepsilon_h\right)$ **y** $\left(H_j - \left(W_j^{rot}\right)_{[3]} > H_i\right)$ **entonces**
  $\left(W_j^{rot}\right)_{[4]} \leftarrow 0$
fin si
si $\left(\left(W_j^{rot}\right)_{[3]} < \varepsilon_h\right)$ **y** $\left(H_i - \left(W_i^{rot}\right)_{[3]} > H_j\right)$ **entonces**
  $\left(W_i^{rot}\right)_{[4]} \leftarrow 0$
fin si
si $\left(\left(W_i^{rot}\right)_{[1]} < \varepsilon_h\right)$ **y** $\left(H_j - \left(W_j^{rot}\right)_{[1]} - \left(W_j^{rot}\right)_{[3]} > H_i - \left(W_i^{rot}\right)_{[3]}\right)$ **entonces**
  $\left(W_j^{rot}\right)_{[2]} \leftarrow 0$
fin si
si $\left(\left(W_j^{rot}\right)_{[1]} < \varepsilon_h\right)$ **y** $\left(H_i - \left(W_i^{rot}\right)_{[1]} - \left(W_i^{rot}\right)_{[3]} > H_j - \left(W_j^{rot}\right)_{[3]}\right)$ **entonces**
  $\left(W_i^{rot}\right)_{[2]} \leftarrow 0$
fin si
si $\left(\left(W_i^{rot}\right)_{[3]} + \left(W_i^{rot}\right)_{[1]} < \varepsilon_h\right)$ **y** $\left(H_j - \left(W_j^{rot}\right)_{[1]} - \left(W_j^{rot}\right)_{[3]} > H_i\right)$ **entonces**
  $\left(W_j^{rot}\right)_{[2,4]} \leftarrow (0,\, 0)^{\mathrm{T}}$
fin si
si $\left(\left(W_j^{rot}\right)_{[3]} + \left(W_j^{rot}\right)_{[1]} < \varepsilon_h\right)$ **y** $\left(H_i - \left(W_i^{rot}\right)_{[1]} - \left(W_i^{rot}\right)_{[3]} > H_j\right)$ **entonces**
  $\left(W_i^{rot}\right)_{[2,4]} \leftarrow (0,\, 0)^{\mathrm{T}}$
fin si
devolver $\left(W_i^{rot},\, W_j^{rot}\right)$

`terminosPresion1DMod`$\left(W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n},\ \Delta t\right)$

$H_m \leftarrow \mathrm{mín}(H_i,\, H_j)$
$h_0 \leftarrow \mathrm{máx}\left(\left(W_i^{rot}\right)_{[1]} + \left(W_i^{rot}\right)_{[3]} - H_i + H_m,\, 0\right)$
$h_1 \leftarrow \mathrm{máx}\left(\left(W_j^{rot}\right)_{[1]} + \left(W_j^{rot}\right)_{[3]} - H_j + H_m,\, 0\right)$
$deta_1 \leftarrow h_1 - h_0$
$h_0 \leftarrow \mathrm{máx}\left(\left(W_i^{rot}\right)_{[3]} - H_i + H_m,\, 0\right)$
$h_1 \leftarrow \mathrm{máx}\left(\left(W_j^{rot}\right)_{[3]} - H_j + H_m,\, 0\right)$
$deta_2 \leftarrow h_1 - h_0$
$\mu_c \leftarrow$ `terminoFriccion-x`$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n})$
$f_c \leftarrow 1 - r\left(1 - e^{-\left(\frac{10\, h_{1,ij}}{\varepsilon_h}\right)^2}\right)$
$s_c \leftarrow f_c\, \mu_c\, g\, h_{2,ij}$
si $\left(|h_{2,ij}\, u_{2,ij,n}| < s_c\, \Delta t\right)$ **entonces**
  $P \leftarrow \left(0, \quad g\, h_{1,ij}\, deta_1, \quad 0, \quad g\, h_{2,ij}\, r\right)^{\mathrm{T}}$
si no
  $P \leftarrow \left(0, \quad g\, h_{1,ij}\, deta_1, \quad 0, \quad g\, h_{2,ij}\, \left((1-r)\, deta_2 + r\, deta_1\right)\right)^{\mathrm{T}}$
fin si
devolver P

`obtenerCoeficientes`$(h_{1,ij},\ h_{2,ij},\ D_1,\ \lambda_1,\ \lambda_4,\ \lambda_{max})$

$\lambda_2 \leftarrow \text{máx}\left(\left|(D_1)_{[2]}\right|,\ \left|(D_1)_{[3]}\right|\right)$
si $\left(\text{sgn}(\lambda_1 + \lambda_4) < 0\right)$ **entonces**
  $\lambda_2 \leftarrow -\lambda_2$
fin si
si $\left(|\lambda_1 - \lambda_4| \geq \varepsilon\right)$ **y** $\left(|\lambda_1 - \lambda_2| \geq \varepsilon\right)$ **y** $\left(|\lambda_2 - \lambda_4| \geq \varepsilon\right)$ **entonces**
  si $(h_{1,ij} \geq \varepsilon_h)$ **y** $(h_{2,ij} \geq \varepsilon_h)$ **entonces**
    $a \leftarrow \dfrac{|\lambda_1|}{(\lambda_1 - \lambda_4)\,(\lambda_1 - \lambda_2)}$
    $b \leftarrow \dfrac{|\lambda_2|}{(\lambda_2 - \lambda_1)\,(\lambda_2 - \lambda_4)}$
    $c \leftarrow \dfrac{|\lambda_4|}{(\lambda_4 - \lambda_1)\,(\lambda_4 - \lambda_2)}$
    $a_0 \leftarrow a\,\lambda_2\,\lambda_4 + b\,\lambda_1\,\lambda_4 + c\,\lambda_1\,\lambda_2$
    $a_1 \leftarrow -a\,(\lambda_2 + \lambda_4) - b\,(\lambda_1 + \lambda_4) - c\,(\lambda_1 + \lambda_2)$
    $a_2 \leftarrow a + b + c$
  si no
    $a_0 \leftarrow \dfrac{\lambda_4\,|\lambda_1| - \lambda_1\,|\lambda_4|}{\lambda_4 - \lambda_1}$
    $a_1 \leftarrow \dfrac{|\lambda_4| - |\lambda_1|}{\lambda_4 - \lambda_1}$
    $a_2 \leftarrow 0$
  fin si
si no
  $a_0 \leftarrow \lambda_{max}$
  $a_1 \leftarrow 0$
  $a_2 \leftarrow 0$
fin si
devolver $(a_0,\ a_1,\ a_2)$

`obtenerParametros`$\left(W_i,\, W_j,\, W_i^{rot},\, W_j^{rot},\, \boldsymbol{\eta}_{ij},\, \Phi^-\right)$

$q_{1,i,t} \leftarrow (W_i)_{[3]}\, \eta_{ij,x} - (W_i)_{[2]}\, \eta_{ij,y}$

$u_{1,i,n} \leftarrow \dfrac{\left(W_i^{rot}\right)_{[1]}\, q_{1,i,t}\, \sqrt{2}}{\sqrt{\left(\left(W_i^{rot}\right)_{[1]}\right)^4 + \left(\text{máx}\left(\left(W_i^{rot}\right)_{[1]},\, \varepsilon_h\right)\right)^4}}$

$q_{1,j,t} \leftarrow (W_j)_{[3]}\, \eta_{ij,x} - (W_j)_{[2]}\, \eta_{ij,y}$

$u_{1,j,n} \leftarrow \dfrac{\left(W_j^{rot}\right)_{[1]}\, q_{1,j,t}\, \sqrt{2}}{\sqrt{\left(\left(W_j^{rot}\right)_{[1]}\right)^4 + \left(\text{máx}\left(\left(W_j^{rot}\right)_{[1]},\, \varepsilon_h\right)\right)^4}}$

si $\left(\left|(\Phi^-)_{[1]}\right| < \varepsilon\right)$ **entonces**

  $u_{1,ij,t} \leftarrow 0$

si no si $\left((\Phi^-)_{[1]} > 0\right)$ **entonces**

  $u_{1,ij,t} \leftarrow u_{1,i,n}$

si no

  $u_{1,ij,t} \leftarrow u_{1,j,n}$

fin si

$q_{2,i,t} \leftarrow (W_i)_{[6]}\, \eta_{ij,x} - (W_i)_{[5]}\, \eta_{ij,y}$

$u_{2,i,n} \leftarrow \dfrac{\left(W_i^{rot}\right)_{[3]}\, q_{2,i,t}\, \sqrt{2}}{\sqrt{\left(\left(W_i^{rot}\right)_{[3]}\right)^4 + \left(\text{máx}\left(\left(W_i^{rot}\right)_{[3]},\, \varepsilon_h\right)\right)^4}}$

$q_{2,j,t} \leftarrow (W_j)_{[6]}\, \eta_{ij,x} - (W_j)_{[5]}\, \eta_{ij,y}$

$u_{2,j,n} \leftarrow \dfrac{\left(W_j^{rot}\right)_{[3]}\, q_{2,j,t}\, \sqrt{2}}{\sqrt{\left(\left(W_j^{rot}\right)_{[3]}\right)^4 + \left(\text{máx}\left(\left(W_j^{rot}\right)_{[3]},\, \varepsilon_h\right)\right)^4}}$

si $\left(\left|(\Phi^-)_{[3]}\right| < \varepsilon\right)$ **entonces**

  $u_{2,ij,t} \leftarrow 0$

si no si $\left((\Phi^-)_{[3]} > 0\right)$ **entonces**

  $u_{2,ij,t} \leftarrow u_{2,i,n}$

si no

  $u_{2,ij,t} \leftarrow u_{2,j,n}$

fin si

$p_1 \leftarrow u_{1,ij,t}\, (\Phi^-)_{[1]}$

$p_2 \leftarrow u_{2,ij,t}\, (\Phi^-)_{[3]}$

devolver $(p_1,\ p_2)$

`positividad`$(W_i,\, W_j,\, \mathcal{F}_{ij}^{-},\, \mathcal{F}_{ij}^{+},\, \Delta t)$

$hp_{1,i} \leftarrow (W_i)_{[1]} + \dfrac{\Delta t}{|V_i|}\,(M_i)_{[1]}$
$hp_{2,i} \leftarrow (W_i)_{[4]} + \dfrac{\Delta t}{|V_i|}\,(M_i)_{[4]}$
$hp_{1,j} \leftarrow (W_j)_{[1]} + \dfrac{\Delta t}{|V_j|}\,(M_j)_{[1]}$
$hp_{2,j} \leftarrow (W_j)_{[4]} + \dfrac{\Delta t}{|V_j|}\,(M_j)_{[4]}$
$a \leftarrow b \leftarrow 10^{30}$
si $\left((\mathcal{F}_{ij}^{-})_{[1]} > 0\right)$ **entonces**
    $a \leftarrow \dfrac{hp_{1,i}\,|V_i|}{(\mathcal{F}_{ij}^{-})_{[1]} + \varepsilon}$
fin si
si $\left((\mathcal{F}_{ij}^{+})_{[1]} > 0\right)$ **entonces**
    $b \leftarrow \dfrac{hp_{1,j}\,|V_j|}{(\mathcal{F}_{ij}^{+})_{[1]} + \varepsilon}$
fin si
$dt_1 \leftarrow \text{mín}(a,\, b)$
$a \leftarrow b \leftarrow 10^{30}$
si $\left((\mathcal{F}_{ij}^{-})_{[4]} > 0\right)$ **entonces**
    $a \leftarrow \dfrac{hp_{2,i}\,|V_i|}{(\mathcal{F}_{ij}^{-})_{[4]} + \varepsilon}$
fin si
si $\left((\mathcal{F}_{ij}^{+})_{[4]} > 0\right)$ **entonces**
    $b \leftarrow \dfrac{hp_{2,j}\,|V_j|}{(\mathcal{F}_{ij}^{+})_{[4]} + \varepsilon}$
fin si
$dt_2 \leftarrow \text{mín}(a,\, b)$
si $(\Delta t \leq dt_1)$ **entonces**
    $f \leftarrow 1$
si no
    $f \leftarrow \dfrac{dt_1}{\Delta t + \varepsilon}$
fin si
$(\mathcal{F}_{ij}^{-})_{[1,2,3]} \leftarrow f\,(\mathcal{F}_{ij}^{-})_{[1,2,3]}$
$(\mathcal{F}_{ij}^{+})_{[1,2,3]} \leftarrow f\,(\mathcal{F}_{ij}^{+})_{[1,2,3]}$
si $(\Delta t \leq dt_2)$ **entonces**
    $f \leftarrow 1$
si no
    $f \leftarrow \dfrac{dt_2}{\Delta t + \varepsilon}$
fin si
$(\mathcal{F}_{ij}^{-})_{[4,5,6]} \leftarrow f\,(\mathcal{F}_{ij}^{-})_{[4,5,6]}$
$(\mathcal{F}_{ij}^{+})_{[4,5,6]} \leftarrow f\,(\mathcal{F}_{ij}^{+})_{[4,5,6]}$
devolver $(\mathcal{F}_{ij}^{-},\, \mathcal{F}_{ij}^{+})$

`modificacionIdentidad`$\left(W_i^{rot},\ W_j^{rot},\ h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n},\ \Delta t\right)$

$H_m \leftarrow \text{mín}(H_i,\ H_j)$
$h_0 \leftarrow \text{máx}\left(\left(W_i^{rot}\right)_{[1]} + \left(W_i^{rot}\right)_{[3]} - H_i + H_m,\ 0\right)$
$h_1 \leftarrow \text{máx}\left(\left(W_j^{rot}\right)_{[1]} + \left(W_j^{rot}\right)_{[3]} - H_j + H_m,\ 0\right)$
$deta_1 \leftarrow h_1 - h_0$
$h_0 \leftarrow \text{máx}\left(\left(W_i^{rot}\right)_{[3]} - H_i + H_m,\ 0\right)$
$h_1 \leftarrow \text{máx}\left(\left(W_j^{rot}\right)_{[3]} - H_j + H_m,\ 0\right)$
$deta_2 \leftarrow h_1 - h_0$
$\mu_c \leftarrow$ `terminoFriccion-x`$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n})$
$f_c \leftarrow 1 - r\left(1 - e^{-\left(\frac{10\,h_{1,ij}}{\varepsilon_h}\right)^2}\right)$
$s_c \leftarrow f_c\,\mu_c\,g\,h_{2,ij}$
$dq_1 \leftarrow \left(W_j^{rot}\right)_{[2]} - \left(W_i^{rot}\right)_{[2]}$
$dq_2 \leftarrow \left(W_j^{rot}\right)_{[4]} - \left(W_i^{rot}\right)_{[4]}$
si $\left(|h_{2,ij}\,u_{2,ij,n}| < s_c\,\Delta t\right)$ **entonces**
  $I \leftarrow \left(deta_1,\quad dq_1,\quad 0,\quad dq_2\right)^{\mathrm{T}}$
si no
  si $\left(\left(W_i^{rot}\right)_{[1]} \geq \varepsilon_h\right)$ **y** $\left(\left(W_j^{rot}\right)_{[1]} \geq \varepsilon_h\right)$ **entonces**
    $I \leftarrow \left(deta_1 - deta_2,\quad dq_1,\quad deta_2,\quad dq_2\right)^{\mathrm{T}}$
  si no
    $I \leftarrow \left(deta_1,\quad dq_1,\quad deta_2,\quad dq_2\right)^{\mathrm{T}}$
  fin si
fin si
devolver I

Las funciones `inversaRotarEstado`, `flujo1DC` y `terminosPresion1D` son idénticas a las del Apéndice A.

La función `terminoFriccion-x` es una de las dos siguientes: `terminoFriccion-Coulomb` o `terminoFriccion-Pouliquen`, que se definen a continuación:

`terminoFriccion-Coulomb`$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n})$

$t \leftarrow |\tan(\alpha)|\ \dfrac{L}{H}$
devolver t

`terminoFriccion-Pouliquen`$(h_{1,ij},\ u_{1,ij,n},\ h_{2,ij},\ u_{2,ij,n})$

$f_c \leftarrow 1 - r\left(1 - e^{-\left(\frac{10\,h_{1,ij}}{\varepsilon_h}\right)^2}\right)$

$f_1 \leftarrow \dfrac{h_{1,ij}\,(u_{1,ij,n})^2\,\sqrt{2}}{g\,f_c\,\sqrt{(h_{1,ij})^4 + \left(\text{máx}(h_{1,ij},\,\varepsilon_h)\right)^4}}$

$f_2 \leftarrow \dfrac{h_{2,ij}\,(u_{2,ij,n})^2\,\sqrt{2}}{g\,f_c\,\sqrt{(h_{2,ij})^4 + \left(\text{máx}(h_{2,ij},\,\varepsilon_h)\right)^4}}$

$f \leftarrow \sqrt{f_1 + f_2 + (1 - f_c)\,f_1\,f_2}$

$\beta \leftarrow 0.136$

$L_2 \leftarrow 8 \cdot 10^{-4}$

$\chi \leftarrow 10^{-3}$

$\mu_{ini} \leftarrow \tan(\alpha_3) + \dfrac{\tan(\alpha_4) - \tan(\alpha_3)}{1 + \dfrac{h_{2,ij}}{L_2}}$

$\mu_{fin} \leftarrow \tan(\alpha_1) + \dfrac{\tan(\alpha_2) - \tan(\alpha_1)}{1 + \dfrac{h_{2,ij}}{L_2}}$

si $(fr > \beta)$ **entonces**

  $\mu_f \leftarrow \tan(\alpha_1) + \dfrac{\tan(\alpha_2) - \tan(\alpha_1)}{1 + \dfrac{\beta\,h_{2,ij}}{f\,L_2}}$

si no si $(|f| < \varepsilon)$ **entonces**

  $\mu_f \leftarrow \mu_{ini}$

si no

  $\mu_f \leftarrow \mu_{ini} + \left(\dfrac{f}{\beta}\right)^{\chi}\,(\mu_{fin} - \mu_{ini})$

fin si

$t \leftarrow \mu_f\,\dfrac{L}{H}$

devolver t

B.6. Procesamiento de Volúmenes

La función `filtroEstado` es muy similar a la del Apéndice A, con la única diferencia que no se realiza el tratamiento de complejos.

`coulomb`$(W,\ \Delta t)$

1: $u_1 \leftarrow \dfrac{W_{[1]}\sqrt{\left(W_{[2]}\right)^2+\left(W_{[3]}\right)^2}\sqrt{2}}{\sqrt{\left(W_{[1]}\right)^4+\Big(\text{máx}\left(W_{[1]},\,\varepsilon_h\right)\Big)^4}}$

2: $u_2 \leftarrow \dfrac{W_{[4]}\sqrt{\left(W_{[5]}\right)^2+\left(W_{[6]}\right)^2}\sqrt{2}}{\sqrt{\left(W_{[4]}\right)^4+\Big(\text{máx}\left(W_{[4]},\,\varepsilon_h\right)\Big)^4}}$

3: $\mu_c \leftarrow$ `terminoFriccion-x`$(W_{[1]},\ u_1,\ W_{[2]},\ u_2)$

4: $f_c \leftarrow 1 - r\left(1 - e^{-\left(\frac{10\,W_{[1]}}{\varepsilon_h}\right)^2}\right)$

5: $s_c \leftarrow f_c\,\mu_c\,g\,W_{[4]}\,\Delta t$

6: $norm_q \leftarrow \sqrt{\left(W_{[5]}\right)^2+\left(W_{[6]}\right)^2}$

7: **si** $(norm_q + s_c \geq \varepsilon)$ **y** $(norm_q \geq s_c)$ **y** $\left(W_{[4]} > 0\right)$ **entonces**

8: $\quad aux \leftarrow \dfrac{norm_q}{norm_q + s_c}$

9: **si no**

10: $\quad aux \leftarrow 0$

11: **fin si**

12: $W_{[5,6]} \leftarrow aux\,W_{[5,6]}$

13: **devolver** W

`discretizacionImplicita`$\left(W^n,\,W^{n+1},\,\Delta t\right)$

1: $h_{1m} \leftarrow \sqrt{\left((W^{n+1})_{[1]}\right)^4+\Big(\text{máx}\left((W^{n+1})_{[1]},\,\varepsilon_h\right)\Big)^4}$

2: $h_{2m} \leftarrow \sqrt{\left((W^{n+1})_{[4]}\right)^4+\Big(\text{máx}\left((W^{n+1})_{[4]},\,\varepsilon_h\right)\Big)^4}$

3: $u_{1,x} \leftarrow \dfrac{(W^{n+1})_{[1]}\,(W^{n+1})_{[2]}\sqrt{2}}{h_{1m}}$

4: $u_{1,y} \leftarrow \dfrac{(W^{n+1})_{[1]}\,(W^{n+1})_{[3]}\sqrt{2}}{h_{1m}}$

5: $u_{2,x} \leftarrow \dfrac{(W^{n+1})_{[4]}\,(W^{n+1})_{[5]}\sqrt{2}}{h_{2m}}$

6: $u_{2,y} \leftarrow \dfrac{(W^{n+1})_{[4]}\,(W^{n+1})_{[6]}\sqrt{2}}{h_{2m}}$

7: $h_{mod} \leftarrow \sqrt{\left((W^{n})_{[1]}\right)^4+\Big(\text{máx}\left((W^{n})_{[1]},\,\varepsilon_h\right)\Big)^4}$

8: $h_{modm} \leftarrow \sqrt{\left((W^{n})_{[4]}\right)^4+\Big(\text{máx}\left((W^{n})_{[4]},\,\varepsilon_h\right)\Big)^4}$

9: $uo_{1,x} \leftarrow \dfrac{(W^n)_{[1]}\,(W^n)_{[2]}\,\sqrt{2}}{h_{mod}}$

10: $uo_{1,y} \leftarrow \dfrac{(W^n)_{[1]}\,(W^n)_{[3]}\,\sqrt{2}}{h_{mod}}$

11: $uo_{2,x} \leftarrow \dfrac{(W^n)_{[4]}\,(W^n)_{[5]}\,\sqrt{2}}{h_{modm}}$

12: $uo_{2,y} \leftarrow \dfrac{(W^n)_{[4]}\,(W^n)_{[6]}\,\sqrt{2}}{h_{modm}}$

13: $du \leftarrow \sqrt{(uo_{1,x} - uo_{2,x})^2 + (uo_{1,y} - uo_{2,y})^2}$

14: $u_1 \leftarrow \sqrt{(uo_{1,x})^2 + (uo_{1,y})^2}$

15: $u_2 \leftarrow \sqrt{(uo_{2,x})^2 + (uo_{2,y})^2}$

16: **si** $\big((W^{n+1})_{[1]} > 0\big)$ **y** $\big((W^{n+1})_{[4]} > 0\big)$ **entonces**

17: $\quad W^{n+1} \leftarrow \texttt{friccionCapas}\big(W^{n+1},\ \Delta t,\ du,\ u_2,\ u_{1,x},\ u_{1,y},\ u_{2,x},\ u_{2,y}\big)$

18: **fin si**

19: **si** $\big((W^{n+1})_{[1]} > 0\big)$ **y** $\big((W^{n+1})_{[4]} < \varepsilon_h\big)$ **entonces**

20: $\quad u_{1,x} \leftarrow \dfrac{(W^{n+1})_{[1]}\,(W^{n+1})_{[2]}\,\sqrt{2}}{h_{1m}}$

21: $\quad u_{1,y} \leftarrow \dfrac{(W^{n+1})_{[1]}\,(W^{n+1})_{[3]}\,\sqrt{2}}{h_{1m}}$

22: $\quad c_1 \leftarrow \dfrac{g\,\Delta t\,(n_1)^2\,u_1}{\big((W^{n+1})_{[1]}\big)^{4/3} + \varepsilon}$

23: $\quad (W^{n+1})_{[2]} \leftarrow \dfrac{(W^{n+1})_{[1]}\,u_{1,x}}{1 + c_1}$

24: $\quad (W^{n+1})_{[3]} \leftarrow \dfrac{(W^{n+1})_{[1]}\,u_{1,y}}{1 + c_1}$

25: **fin si**

26: **si** $\big((W^{n+1})_{[4]} > 0\big)$ **y** $\big((W^{n+1})_{[1]} < \varepsilon_h\big)$ **entonces**

27: $\quad u_{2,x} \leftarrow \dfrac{(W^{n+1})_{[4]}\,(W^{n+1})_{[5]}\,\sqrt{2}}{h_{2m}}$

28: $\quad u_{2,y} \leftarrow \dfrac{(W^{n+1})_{[4]}\,(W^{n+1})_{[6]}\,\sqrt{2}}{h_{2m}}$

29: $\quad c_1 \leftarrow \dfrac{g\,\Delta t\,(n_2)^2\,u_2}{\big((W^{n+1})_{[4]}\big)^{4/3} + \varepsilon}$

30: $\quad (W^{n+1})_{[5]} \leftarrow \dfrac{(W^{n+1})_{[4]}\,u_{2,x}}{1 + c_1}$

31: $\quad (W^{n+1})_{[6]} \leftarrow \dfrac{(W^{n+1})_{[4]}\,u_{2,y}}{1 + c_1}$

32: **fin si**

33: **devolver** W^{n+1}

`friccionCapas`$(W,\ \Delta t,\ du,\ u_2,\ u_{1,x},\ u_{1,y},\ u_{2,x},\ u_{2,y})$

1: $h_{mod} \leftarrow W_{[4]} - r\, W_{[1]}$
2: $h_{modm} \leftarrow \sqrt{(h_{mod})^4 + \left(\text{máx}(h_{mod},\, \varepsilon_h\,(1+r))\right)^4}$
3: $c_1 \leftarrow \dfrac{\Delta t\, W_{[4]}\, m_f\, du\, h_{mod}\, \sqrt{2}}{h_{modm}}$
4: $c_2 \leftarrow \dfrac{r\, \Delta t\, W_{[1]}\, m_f\, du\, h_{mod}\, \sqrt{2}}{h_{modm}}$
5: $c_3 \leftarrow \dfrac{g\, \Delta t\, (n_2)^2\, u_2}{\left(W_{[4]}\right)^{4/3} + \varepsilon}$
6: $det \leftarrow \dfrac{1}{(1+c_1)\,(1+c_2+c_3) - c_1\, c_2}$
7: $W_{[2]} \leftarrow W_{[1]}\left(u_{1,x}\,(1+c_2+c_3) + c_1\, u_{2,x}\right) det$
8: $W_{[3]} \leftarrow W_{[1]}\left(u_{1,y}\,(1+c_2+c_3) + c_1\, u_{2,y}\right) det$
9: $W_{[5]} \leftarrow W_{[4]}\left(u_{2,x}\,(1+c_1) + c_2\, u_{1,x}\right) det$
10: $W_{[6]} \leftarrow W_{[4]}\left(u_{2,y}\,(1+c_1) + c_2\, u_{1,y}\right) det$
11: **devolver** W

Bibliografía

[AMD07] AMD. Brook+ Presentation. ACM/IEEE Conference on Supercomputing 2007 (SC 2007), Reno (EE.UU.), Noviembre 2007.

[Inf] InfiniBand Trade Association. InfiniBand architecture specification, Volume 1, Release 1.2, 2004. `http://www.infinibandta.org`, Accedido Octubre 2012.

[BHLN10] A. R. Brodtkorb, T. R. Hagen, K.-A. Lie, y J. R. Natvig. Simulation and visualization of the Saint-Venant system using GPUs. *Computing and Visualization in Science*, 13(7):341–353, 2010.

[BMPV03] F. Bouchut, A. Mangeney, B. Perthame, y J.-P. Vilotte. A new model of Saint Venant and Savage-Hutter type for gravity driven shallow flows. *Comptes Rendus Mathématique*, 336(6):531–536, 2003.

[BP07] T. Brandvik y G. Pullan. Acceleration of a two-dimensional Euler flow solver using commodity graphics hardware. *Journal of Mechanical Engineering Science*, 221(12):1745–1748, 2007.

[BS10] A. R. Brodtkorb y M. L. Sætra. Shallow water simulations on multiple GPUs. En *PARA 2010: State of the Art in Scientific and Parallel Computing*, Reykjavik (Islandia), Junio 2010.

[BSA12] A. R. Brodtkorb, M. L. Sætra, y M. Altinakar. Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation. *Computers & Fluids*, 55(15):1–12, 2012.

[BVdR+00] P. Bárcenas, J. T. Vázquez, V. Díaz del Río, L. M. Fernández-Salas, O. Tello, y J. L. Sanz. La vertiente meridional del Banco de la isla de Alborán: presencia de dos sistemas cañón-abanico submarino. En *VI Reunión Nacional de Geomorfología*, Madrid (España), Septiembre 2000.

[CFG+05] M. J. Castro, A. M. Ferreiro, J. A. García, J. M. González, J. Macías, C. Parés, y M. E. Vázquez-Cendón. The numerical treatment of wet/dry fronts in shallow flows: applications to one-layer and two-layer systems. *Mathematical and Computer Modelling*, 42(3–4):419–439, 2005.

[CFN12] M. J. Castro y E. D. Fernández-Nieto. A class of computationally fast first order finite volume solvers: PVM methods. *SIAM Journal on Scientific Computing*, 34(4):173–196, 2012.

[CFNF+09] M. J. Castro, E. D. Fernández-Nieto, A. M. Ferreiro, J. A. García, and C. Parés. High order extensions of Roe schemes for two-dimensional nonconservative hyperbolic systems. *Journal of Scientific Computing*, 39(1):67–114, 2009.

[CGGP06] M. J. Castro, J. A. García, J. M. González, y C. Parés. A parallel 2D finite volume scheme for solving systems of balance laws with nonconservative products: Application to shallow flows. *Computer Methods in Applied Mechanics and Engineering*, 195(19–22):2788–2815, 2006.

[CGGP08] M. J. Castro, J. A. García, J. M. González, y C. Parés. Solving shallow-water systems in 2D domains using finite volume methods and multimedia SSE instructions. *Journal of Computational and Applied Mathematics*, 221(1):16–32, 2008.

[CJvdP07] B. Chapman, G. Jost, y R. van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming.* The MIT Press, 2007.

[CM69] E. Cuthill y J. McKee. Reducing the bandwidth of sparse symmetric matrices. En *Proceedings of the 24th National Conference ACM*, páginas 157–172, 1969.

[COdlA+11] M. J. Castro, S. Ortega, M. de la Asunción, J. M. Mantas, y J. M. Gallardo. GPU computing for shallow water flow simulation based on finite volume schemes. *Comptes Rendus Mécanique*, 339(2–3):165–184, 2011.

[DCPT09] M. Dumbser, M. Castro, C. Parés, y E. F. Toro. ADER schemes on unstructured meshes for nonconservative hyperbolic systems: Applications to geophysical flows. *Computers & Fluids*, 38(9):1731–1748, 2009.

[DH01] C. Ding y Y. He. A ghost cell expansion method for reducing communications in solving PDE problems. En

ACM/IEEE Conference on Supercomputing 2001 (SC 2001), Denver (EE.UU.), Noviembre 2001.

[DK07] M. Dumbser y M. Käser. Arbitrary high order non-oscillatory finite volume schemes on unstructured meshes for linear hyperbolic systems. *Journal of Computational Physics*, 221(2):693–723, 2007.

[dlACFN+12] M. de la Asunción, M. J. Castro, E. D. Fernández-Nieto, J. M. Mantas, S. Ortega, y J. M. González. Efficient GPU implementation of a two waves TVD-WAF method for the two-dimensional one layer shallow water system on structured meshes. *Computers & Fluids, DOI: 10.1016/j.compfluid.2012.01.012*, 2012.

[dlAMC10] M. de la Asunción, J. M. Mantas, y M. J. Castro. Programming CUDA-based GPUs to simulate two-layer shallow water flows. En P. D'ambra, M. Guarracino, y D. Talia, editores, *Euro-Par 2010*, volumen 6272 de *Lecture Notes in Computer Science*, páginas 353–364. Springer, 2010.

[dlAMC11] M. de la Asunción, J. M. Mantas, y M. J. Castro. Simulation of one-layer shallow water systems on multicore and CUDA architectures. *Journal of Supercomputing*, 58(2):206–214, 2011.

[dlAMC12] M. de la Asunción, J. M. Mantas, y M. J. Castro. Evaluating the impact of cell renumbering of unstructured meshes on the performance of finite volume GPU solvers. En *12th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE 2012)*, La Manga (España), Julio 2012.

[dlAMCFN11] M. de la Asunción, J. M. Mantas, M. J. Castro, y E. D. Fernández-Nieto. Two-layer shallow water simulation on clusters of CUDA enabled GPUs. En *Parallel CFD 2011*, Barcelona (España), Mayo 2011.

[dlAMCFN12] M. de la Asunción, J. M. Mantas, M. J. Castro, y E. D. Fernández-Nieto. An MPI-CUDA implementation of an improved Roe method for two-layer shallow water systems. *Journal of Parallel and Distributed Computing, Special Issue on Accelerators for High-Performance Computing*, 72(9):1065–1072, 2012.

[dlAMCG12] M. de la Asunción, J. M. Mantas, M. J. Castro, y J. M. González. Multi-GPU simulation of tsunamis generated by

submarine landslides. En *XIX International Conference on Computational Methods in Water Resources (CMWR 2012)*, Urbana-Champaign (EE.UU.), Junio 2012.

[DPRV99] P. Degond, P. F. Peyrard, G. Russo, y P. Villedieu. Polynomial upwind schemes for hyperbolic systems. *Comptes Rendus Académie des Sciences, Series 1*, 328(6):479–483, 1999.

[Eig] Eigen 3.1.1. `http://eigen.tuxfamily.org`, Accedido Octubre 2012.

[Eis] EISPACK. `http://www.netlib.org/eispack`, Accedido Octubre 2012.

[FK03] R. Fernando y M. J. Kilgard. *The Cg tutorial: The definitive guide to programmable real-time graphics.* Addison-Wesley Professional, 2003.

[FN09] E. D. Fernández-Nieto. Modelling and numerical simulation of submarine sediment shallow flows: transport and avalanches. *Boletín de la Sociedad Española de Matemática Aplicada*, 49:83–103, 2009.

[FNBB+08] E. D. Fernández-Nieto, F. Bouchut, D. Bresch, M. J. Castro, y A. Mangeney. A new Savage-Hutter type model for submarine avalanches and generated tsunami. *Journal of Computational Physics*, 227(16):7720–7754, 2008.

[FNCP11] E. D. Fernández-Nieto, M. J. Castro, y C. Parés. On an intermediate field capturing Riemann solver based on a parabolic viscosity matrix for the two-layer shallow water system. *Journal of Scientific Computing*, 48(1–3):117–140, 2011.

[FNNR08] E. D. Fernández-Nieto y G. Narbona-Reina. Extension of WAF type methods to non-homogeneous shallow water equations with pollutant. *Journal of Scientific Computing*, 36(2):193–217, 2008.

[FQKYS04] Z. Fan, F. Qiu, A. Kaufman, y S. Yoakum-Stover. GPU cluster for high performance computing. En *ACM/IEEE Conference on Supercomputing 2004 (SC 2004)*, Pittsburgh (EE.UU.), Noviembre 2004.

[GCdlA+11] J. M. González, M. J. Castro, M. de la Asunción, E. D. Fernández-Nieto, J. Macías, y C. Parés. Modelling submarine avalanches and generated tsunamis. Application to tsunami effects forecasting. En *European Geosciences Union General Assembly 2011 (EGU 2011)*, Viena (Austria), Abril 2011.

[GCSdlA12] J. M. González, M. J. Castro, C. Sánchez, y M. de la Asunción. Simulation of landslide-generated tsunamis with the HySEA platform: Application to the Lituya Bay 1958 tsunami. En *European Geosciences Union General Assembly 2012 (EGU 2012)*, Viena (Austria), Abril 2012.

[GdlAC+11] J. M. González, M. de la Asunción, M. J. Castro, E. D. Fernández-Nieto, J. Macías, y C. Parés. Modelling tsunamis generated by submarine landslides. Application to real cases in the Mediterranean. En *13th Plinius Conference on Mediterranean Storms*, Savona (Italia), Septiembre 2011.

[GdlAC+12] J. M. González, M. de la Asunción, M. J. Castro, E. D. Fernández-Nieto, J. Macías, T. Morales, C. Parés, y C. Sánchez. Simulation of landslide-generated tsunamis with the HySEA platform: the Lituya Bay 1958 event. En *SIAM Conference on Nonlinear Waves and Coherent Structures*, Seattle (EE.UU.), Junio 2012.

[Geo71] J. A. George. Computer implementation of the finite element method. Technical Report STAN-CS-71-208, Universidad de Stanford, 1971.

[GFA+09] C. H. González, B. B. Fraguela, D. Andrade, J. A. García, y M. J. Castro. Programación de alto rendimiento en el procesador Cell: Aplicación a simulación de fluidos. En *XX Jornadas de Paralelismo*, La Coruña (España), Septiembre 2009.

[GFB+04] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barret, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, y T. S. Woodall. Open MPI: goals, concept, and design of a next generation MPI implementation. En *11th European PVM/MPI Users'Group Meeting*, Budapest (Hungría), Septiembre 2004.

[GKKG03] A. Grama, G. Karypis, V. Kumar, y A. Gupta. *Introduction to Parallel Computing.* Addison-Wesley Professional, Segunda edición, 2003.

[GRMG11] M. Geveler, D. Ribbrock, S. Mallach, y D. Göddeke. A simulation suite for Lattice-Boltzmann based real-time CFD applications exploiting multi-level parallelism on modern multi- and many-core architectures. *Journal of Computational Science*, 2(2):113–123, 2011.

[Har07] M. Harris. Optimizing parallel reduction in CUDA. Technical report, NVIDIA, 2007.

[HHL+05] T. R. Hagen, J. M. Hjelmervik, K.-A. Lie, J. R. Natvig, y M. O. Henriksen. Visual simulations of shallow-water waves. *Simulation Modelling Practice and Theory*, 13(8):716–726, 2005.

[HL94] B. Hendrickson y R. Leland. The Chaco user's guide: version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, 1994.

[HLvL83] A. Harten, P. D. Lax, y B. van Leer. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. *SIAM Review*, 25(1):35–61, 1983.

[HSO07] M. Harris, S. Sengupta, y J. D. Owens. Parallel prefix sum (scan) with CUDA. En H. Nguyen, editor, *GPU Gems 3*, páginas 851–876. Addison-Wesley Professional, 2007.

[INR] INRIA. OpenNL 3.2.1. `http://alice.loria.fr/software/OpenNL`, Accedido Octubre 2012.

[JTS10] D. A. Jacobsen, J. C. Thibault, y I. Senocak. An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters. En *48th AIAA Aerospace Sciences Meeting*, Orlando (EE.UU.), Enero 2010.

[KEGM10] D. Komatitsch, G. Erlebacher, D. Göddeke, y D. Michéa. High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster. *Journal of Computational Physics*, 229(20):7692–7714, 2010.

[Khr] Grupo Khronos. OpenCL - The open standard for parallel programming of heterogeneous systems. `http://www.khronos.org/opencl`, Accedido Octubre 2012.

[Kita] Kitware. Paraview. `http://www.paraview.org`, Accedido Octubre 2012.

[Kitb] Kitware. Visualization Toolkit. `http://www.vtk.org`, Accedido Octubre 2012.

[KK99] G. Karypis y V. Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[LAA+09] J. Lobeiras, M. Amor, M. Arenaz, B. B. Fraguela, J. A. García, y M. J. Castro. Simulación de aguas poco profundas en una GPU mediante Brook+. En *XX Jornadas de Paralelismo*, La Coruña (España), Septiembre 2009.

[LHS+09] W.-Y. Liang, T.-J. Hsieh, M. T. Satria, Y.-L. Chang, J.-P. Fang, C.-C. Chen, y C.-C. Han. A GPU-based simulation of tsunami propagation and inundation. En *9th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2009)*, Taipei (Taiwan), Junio 2009.

[LLC] GameTutorials LLC. `http://www.gametutorials.com`, Accedido Octubre 2012.

[LMn+09] M. Lastra, J. M. Mantas, C. Ure na, M. J. Castro, y J. A. García. Simulation of shallow-water systems using graphics processing units. *Mathematics and Computers in Simulation*, 80(3):598–618, 2009.

[LPRM02] B. Lévy, S. Petitjean, N. Ray, y J. Maillot. Least squares conformal maps for automatic texture atlas generation. En *29th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2002)*, San Antonio (EE.UU.), Julio 2002.

[MA09] M. A. Acuña y T. Aoki. Real-time tsunami simulation on a multi-node GPU cluster. ACM/IEEE Conference on Supercomputing 2009 (SC 2009) [Poster], Portland (EE.UU.), Noviembre 2009.

[Map] Maplesoft. Maple 16. `http://www.maplesoft.com/products/Maple`, Accedido Octubre 2012.

[Mat] MathWorks. MATLAB R2012b. `http://www.mathworks.com/products/matlab`, Accedido Octubre 2012.

[MFG+12] J. Macías, L. M. Fernández, J. M. González, J. T. Vázquez, M. J. Castro, P. Bárcenas, V. Díaz del Río, T. Morales, M. de la Asunción, y C. Parés. *Deslizamientos y tsunamis en Alborán: un riesgo oculto bajo el mar*. Temas de Oceanografía. Instituto Español de Oceanografía, 2012.

[Mica] Microsoft. Compute Shader Overview. `http://msdn.microsoft.com/en-us/library/ff476331.aspx`, Accedido Octubre 2012.

[Micb] Microsoft. DirectX Developer Center. `http://msdn.microsoft.com/en-us/directx`, Accedido Octubre 2012.

[MLF+12] M. Viñas, J. Lobeiras, B. B. Fraguela, M. Arenaz, M. Amor, J. A. García, M. J. Castro, y R. Doallo. A multi-GPU shallow-water simulation with transport of contaminants. *Concurrency and Computation: Practice and Experience, DOI: 10.1002/cpe.2917*, 2012.

[MPI] Message Passing Interface Forum: A Message Passing Interface Standard. Universidad de Tennessee, Knoxville, Tennessee.

[NSC+06] D. H. Natawidjaja, K. Sieh, M. Chlieh, J. Galetzka, B. W. Suwargadi, H. Cheng, R. L. Edwards, J.-P. Avouac, y S. N. Ward. Source parameters of the great Sumatran megathrust earthquakes of 1797 and 1833 inferred from coral microatolls. *Journal of Geophysical Research*, 111(B06403), 2006.

[NVIa] NVIDIA. GPU Computing SDK. `http://developer.nvidia.com/cuda/gpu-computing-sdk`, Accedido Octubre 2012.

[NVIb] NVIDIA. NVIDIA Developer Zone. `http://developer.nvidia.com/category/zone/cuda-zone`, Accedido Octubre 2012.

[NVIc] NVIDIA. NVIDIA's next generation CUDA compute architecture: Fermi. `http://www.nvidia.com/object/fermi-architecture.html`, Accedido Octubre 2012.

[NVId] NVIDIA. NVIDIA's next generation CUDA compute architecture: Kepler GK110. `http://www.nvidia.com/object/nvidia-kepler.html`, Accedido Octubre 2012.

[NVIe] NVIDIA. Tuning CUDA applications for Fermi. `http://developer.nvidia.com/cuda/nvidia-gpu-computing-documentation`, Accedido Octubre 2012.

[OHL+08] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, y J. C. Phillips. GPU computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[PC04] C. Parés y M. J. Castro. On the well-balance property of Roe's method for nonconservative hyperbolic systems. Applications to shallow-water systems. *ESAIM: Mathematical Modelling and Numerical Analysis*, 38(5):821–852, 2004.

[PF02] O. Pouliquen y Y. Forterre. Friction law for dense granular flows: application to the motion of a mass down a rough inclined plane. *Journal of Fluid Mechanics*, 453:133–151, 2002.

[PSH+10] H. Permana, S. C. Singh, N. Hananto, A. Chauhan, M. Denolle, A. Handryana, Sumirah, A. W. Djaja, E. Rohendi, C. Sudjana, J. Prihantono, y D. D. Wardhana. Submarine mass movement and localized tsunami potentiality of Mentawai Basin, Sumatera, Indonesia. *Bulletin of the Marine Geology*, 25(2):53–63, 2010.

[RLKG+09] R. J. Rost, B. M. Licea-Kane, D. Ginsburg, J. M. Kessenich, B. Lichtenbelt, H. Malan, y M. Weiblen. *OpenGL Shading Language*. Addison-Wesley Professional, Tercera edición, 2009.

[RS06] M. Rumpf y R. Strzodka. Graphics processor units: new prospects for parallel computing. En A. M. Bruaset y A. Tveito, editores, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volumen 51 de *Lecture Notes in Computational Science and Engineering*, páginas 89–134. Springer, 2006.

[Sa11] C. Sánchez. *Simulación numérica de tsunamis generados por avalanchas submarinas: aplicación al caso de Lituya Bay*. Trabajo Fin de Máster, Universidad de Málaga, 2011.

[SCGdlA12] C. Sánchez, M. J. Castro, J. M. González, y M. de la Asunción. Simulation of landslide-generated tsunamis using IFCP volume scheme. Application to the Lituya Bay 1958 tsunami. En *1st Joint Conference of the Belgian, Royal Spanish and Luxembourg Mathematical Societies (BSL 2012)*, Lieja (Bélgica), Junio 2012.

[Shr09] D. Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*. Addison-Wesley Professional, Séptima edición, 2009.

[Shu88] C.-W Shu. Total-variation-diminishing time discretizations. *SIAM Journal on Scientific and Statistical Computing*, 9(6):1073–1084, 1988.

[TB00] E. F. Toro y S. J. Billet. Centred TVD schemes for hyperbolic conservation laws. *IMA Journal of Numerical Analysis*, 20(1):47–79, 2000.

[Tec] Tecplot. `http://www.tecplot.com`, Accedido Octubre 2012.

[Tor89] E. F. Toro. A weighted average flux method for hyperbolic conservation laws. *Proceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences*, 423(1865):401–418, 1989.

[Tor92] E. F. Toro. Riemann problems and the WAF method for solving the two-dimensional shallow water equations. *Philosophical Transactions of the Royal Society of London, Series A, Physical Sciences and Engineering*, 338(1649):43–68, 1992.

[TS12] J. C. Thibault y I. Senocak. Accelerating incompressible flow computations with a Pthreads-CUDA implementation on small-footprint multi-GPU platforms. *Journal of Supercomputing*, 59(2):693–719, 2012.

[USG] United States Geological Survey. `http://earthquake.usgs.gov/earthquakes/eqinthenews/2004/us2004slav`, Accedido Octubre 2012.

[WAK10] P. Wang, T. Abel, y R. Kaehler. Adaptive mesh fluid simulations on GPU. *New Astronomy*, 15(7):581–589, 2010.

[WC07] C. Walshaw y M. Cross. JOSTLE: Parallel multilevel graph-partitioning software – An overview. En F. Magoules, editor, *Mesh partitioning techniques and domain decomposition techniques*, páginas 27–58. Civil-Comp Ltd., 2007.

[ZT00] O. C. Zienkiewicz y R. L. Taylor. *The Finite Element Method*. Butterworth-Heinemann, Quinta edición, 2000.