Simulación de Dinámica de Fluidos Computacional (CFD), con el método de Hidrodinámica de Partículas Suavizadas (SPH) en Arquitecturas Computacionales con múltiples unidades de procesamiento gráfico (GPU)

Nicolás Gerardo Gutiérrez Carreño

Trabajo de Grado para Optar al Título de Ingeniero de Sistemas

Director

Carlos Jaime Barrios Hernández

Doctor en Informätica, Univesité Nice-Sophia Antipolis, Francia

Codirector

Jorge Luis Chacón Velasco

PhD en ingeniería por la UPV

Asesor

Sergio Augusto Gélvez Cortés

Universidad Industrial de Santander

Facultad de Ingeniería Físico-mecánicas

Escuela de Ingeniería de Sistemas

Ingeniería de Sistemas

Bucaramanga

2022

Dedicatoria

La escritura de este libro se la debo en gran parte a mis padres y a mi hermana, que me ofrecieron todo su apoyo e incondicionalidad durante todos mis estudios.

Agradecimientos

Expreso mis agradecimientos al grupo de investigación SC3UIS: Supercomputación y Calculo Científico de la Universidad Industrial de Santander, en especial al profesor Carlos Jaime Barrios, a mi codirector Jorge Luis Chacón y a mi asesor Sergio Gelves por acompañarme y aconsejarme en este proceso.

Quiero agradecer también al centro de investigación ABACUS-Cinvestav: Centro de Super computo del Centro de investigación y estudios Avanzados de México, por el apoyo brindado a la realización de este trabajo.

A los encargados y desarrolladores de DualSPHysics que estuvieron propuestos para el asesoramiento necesario y el uso de su tecnología.

A mis amigos cercanos que me han acompañado a lo largo de mi carrera y a lo largo de este proceso y me han brindado apoyo en muchos momentos.

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Glosario

Aceleración: La aceleración es hacer uso de hardware con la finalidad de ejecutar software sobre este y llevar una tarea determinada en una menor cantidad de tiempo.

Arquitectura computacional: modelo conceptual y operacional de la estructura computacional de un sistema de computadoras.

CFD: o mecánica de fluidos computacional (MFC) es el uso de métodos numéricos para resolver problemas de flujos de fluidos.

Clúster: red de computadoras interconectadas entre sí que actúan como un único servidor.

Concurrencia: procesos o instrucciones ejecutados independientemente, pero conectados entre sí.

CPU: unidad central de procesamiento.

GPU: unidad de procesamiento gráfico.

Nodo: punto de conexión de al menos dos o más elementos dentro de un: la red.

Paralelismo: ejecución simultánea de procesos o instrucciones computacionales.

SPH: la hidrodinámica de partículas suavizadas es un método numérico enfocado en la resolución de problemas de flujos de fluidos computacionales.

Simbología

Función de suavizado: $w_{ij} = W(|\vec{r} - \vec{r_0}|, h)$

Dominio de la simulación: Ω

Distancia de suavizado: h

Masa de una partícula de interés: m_j

Densidad de una partícula de interés: ρ_j

Volumen de una partícula de interés: $V_j = \frac{m_j}{\rho_j}$

Velocidad de fluido: $\vec{U}$

Vector distancia entre una partícula i y una partícula vecina j: r_{ij}

Resumen

Título: Simulación de dinámica de fluidos computacional (CFD), con el método de hidrodinámica de partículas suavizadas (SPH) en Arquitecturas con múltiples unidades de procesamiento gráfico (GPU)[*]

Autor: Nicolas Gerardo Gutierrez Carreño[**]

Palabras Clave: GPU, SPH, CFD, clúster, paralelismo, concurrencia, nodos.

El estudio de la dinámica de fluidos CFD es actualmente una de las ramas principales para la investigación de fluidos de alta complejidad, el avance en la aplicación de nuevas metodologías y técnicas novedosas como métodos sin malla ha ejercido una presión positiva en su crecimiento. Como parte de ellas, el método SPH ha tomado una posición importante en los adelantos de CFD, debido a que su estructura sin malla hace posible resolver problemas complejos como la ruptura de olas, el choque de fluidos y otros problemas que complejos que no pueden ser usados fácilmente con los métodos basados en mallas. Sin embargo, unas de las preocupaciones de SPH es su alto coste computacional, para esto se han definido varias soluciones, una de ellas es la implementación de formas de paralelización en sus instrucciones. Con el crecimiento de la tecnología y el desarrollo de las unidades de procesamiento gráfico, es posible llevar problemas numéricos para ser resueltas en estos, debido a que su nivel de procesamiento supera por mucho a la tradicional CPU.

Basado en lo anterior es posible deducir que la paralelización de la implementación del método sobre estructuras con múltiples GPU puede conllevar a una aceleración de sus resultados y por consiguiente una disminución en los tiempos de ejecución de este, contribuyendo con una de las alternativas más interesantes para el problema principal de SPH y a su vez contribuyendo con el desarrollo sobre estas tecnologías.

---

[*] Trabajo de Grado

[**] Facultad de Ingeniera Físico-mecánicas. Escuela de Ingeniería de Sistemas. Ingeniería de Sistemas. Director: Carlos Jaime Barrios Hernández. Doctor en Informätica, Univesité Nice-Sophia Antipolis, Francia. Codirector: Jorge Luis Chacón Velasco. Ph.D en ingeniería UPV

Abstract

Title: Computational Fluid Dynamics (CFD) simulation with the Smoothed Particle Hydrodynamics (SPH) method on Multi-Graphics Processing Unit (GPU) architectures[*]

Author(s): Nicolas Gerardo Gutierrez Carreño[**]

Key Words: GPU, SPH, CFD, cluster, parallel, concurrent, nodes.

The study of CFD fluid dynamics is currently one of the main branches for the investigation of highly complex fluids, the advancement in the application of new methodologies and novel techniques such as mesh-free methods has exerted a positive pressure on its growth.
As part of these, the SPH method has taken an important position in CFD advances, because its meshless structure makes it possible to solve complex problems such as wave breaking, fluid shocks and other complex problems that cannot be easily used with mesh-based methods. However, one of the concerns of SPH is its high computational cost, for this several solutions have been defined, one of them is the implementation of forms of parallelization in its instructions. With the growth of technology and the development of graphic processing units, it is possible to carry numerical problems to be solved in these, because their processing level exceeds by far the traditional CPU.

Based on the above, it is possible to deduce that the parallelization of the implementation of the method on structures with multiple GPUs can lead to an acceleration of its results and therefore a decrease in the execution times of this, contributing with one of the most interesting alternatives for the main problem of SPH and in turn contributing with the development of these technologies.

---

[*] Degree Work

[**] Physical-Mechanical Engineering Faculty. Informatic and Systems Engineering School. Director: Carlos Jaime Barrios Hernández. Ph.D Informätica, Université Nice-Sophia Antipolis, Francia. Codirector: Jorge Luis Chacón Velasco. Ph.D engineering from UPV

Introducción

En el estudio de fluidos, la dinámica de fluidos computacional CFD, ha tomado un lugar habitual en la rama de investigación académica para investigar flujos de alta complejidad. El avance continuo de recursos hardware computacionales ha llevado al desarrollo y aplicación de técnicas basadas en mallas, con métodos de elementos finitos, discretizaciones de volúmenes finitos y métodos de diferencias finitas. También a su vez han aparecido en los últimos años métodos sin mallas o métodos lagrangianos, los cuales han crecido en popularidad. Dichos métodos pueden aplicarse a problemas no lineales, en geometrías arbitrarias complejas, los cuales serían un problema por su complejidad para los métodos convencionales basados en mallas. (Crespo, Domínguez, Barreiro, Anxo, Gómez-Gesteira, Moncho, Rogers & Benedict, 2011)

El método de hidrodinámica de partículas suavizadas (o SPH: Smoothed Particle Hydrodinamics) es uno de estos métodos lagrangianos sin malla, que ha tomado una posición importante y es cada vez más usado para una amplia gama de aplicaciones en el campo de los fluidos computacionales, dicho método representa el flujo con partículas que interactúan con estructuras y pueden presentar grandes deformaciones con fronteras en movimiento. La técnica de SPH se acerca a una etapa de madurez con mejoras y actualizaciones continuas y modificaciones. Esto significa que la precisión, estabilidad y fiabilidad del modelo alcanzará un nivel aceptable para las aplicaciones prácticas de ingeniería. (Crespo, Domínguez, Rogers, Gómez-Gesteira, Longshaw, Canelas, Vacondio, Barreiro, García-Feal, 2015)

Los estudios CFD tienen una historia de búsqueda y exigencia en el rendimiento computacional que cada vez es mayor. En el pasado esta búsqueda se ha suplido en parte con velocidades de CPU más rápidas. Pero esta era ha llegado a una meseta, debido a problemas con

la disipación de calor en los procesadores. El paralelismo de datos o paralelismo de tareas puede aumentar el rendimiento de los cálculos sin aumentar la velocidad del reloj de las CPU. El avance en los últimos años de las unidades de procesamiento gráfico o GPUs han aparecido también como una alternativa a los procesadores. Una GPU es un procesador especializado en la representación de gráficos. Su naturaleza es la visualización y el renderizado altamente paralelo de muchos pixeles, lo cual impulsó el desarrollo de hardware paralelo con alta capacidad de cálculo. Tal como su aparición y su desarrollo constante, continuamente se desarrollan lenguajes para expresar problemas informáticos no gráficos como tareas de programación gráfica, lo que permitirá usar la GPU en otros entornos. (Jespersen, D. C., 2010).

Los adelantos tanto para las simulaciones CFD tanto como para el método SPH y los avances en las tecnologías de procesamiento gráfico GPU deja en evidencia de que es la vía por seguir para conseguir mejorar la calidad de los estudios con este método. Ya que demostrado que la implementación paralela de este sobre GPU ofrece una mejoría reflejada en los tiempos de procesamiento. Por lo que es fácil imaginar que una mejoría en el desarrollo y aceleración de SPH podría darse sobre arquitecturas con múltiples GPUs.

En este libro se encontrará el desarrollo de la investigación tratada a lo largo de este trabajo, así, en el capítulo 1, se presentará un resumen del proyecto, donde se encontrará el planteamiento general y la justificación del problema, la metodología propuesta y la esquematización de los procesos que se acompañaran para el tratamiento de este. En el capítulo 2, se expondrán los objetivos propuestos en la búsqueda de respuestas y soluciones. En el capítulo 3, se hará una profundización en los conceptos claves necesarios para la elaboración de este trabajo. Se argumentarán los resultados obtenidos de la investigación acerca de las dinámicas de fluidos y su implementación computacional y sobre la aplicación del método de hidrodinámica de partículas

suavizadas en la simulación CFD, se obtendrá un vistazo de la física del problema y las aplicaciones de este en problemas de la ingeniería. En el capítulo 4, se presentarán los resultados obtenidos durante el desarrollo de la metodología del trabajo, se definirán los detalles de la investigación conceptual de las tecnologías y herramientas que se utilizarán, los estudios de trabajos previos y antecedentes en el desarrollo de simulación CFD, como implementaciones de algoritmos de diferentes autores en búsqueda de proyectos que implementen el método SPH y que exploten en ellos la concurrencia y la paralelización de procesos, posteriormente una profundización en las características de la paralelización del método a arquitecturas GPU y el despliegue de la simulación en arquitecturas con múltiples GPU, se encontrara a su vez un análisis de los resultados alcanzados. En el capítulo 5, se presentará como cierre conclusiones acerca de los resultados obtenidos, recomendaciones y lineamientos propuesto por el autor acerca de las implementaciones SPH sobre arquitecturas multi-GPU y propuestas de trabajos futuros.

1. Resumen del Proyecto

1.1 Planteamiento y justificación del problema

¿Es posible acelerar un código de simulación de dinámicas de fluidos computacionales que implemente el método SPH con el uso de arquitecturas con múltiples GPUs?

Según Valdez, Domínguez, Rogers y Crespo (2013), las implementaciones de simulaciones de flujos de fluidos están limitadas por restricciones computacionales que representan un problema para la dinámica de fluidos computacional basadas en partículas, el primero está relacionada directamente con el tiempo que tarda la simulación, la segunda resistencia es el tamaño del sistema, ya que para obtener una información físicamente significativa el sistema simulado debe tener un tamaño propicio por un espacio de tiempo prolongado.

Una de las soluciones sugiere el uso de técnicas de aceleración, las cuales se clasifican en tres grupos en función del hardware usado, la primera se basa en el uso de computación de alto rendimiento (HPC) y el uso de los nodos de computación. Un segundo enfoque implica el uso de Field – Programmable Gate Arrays (FPGAs). Y un tercer enfoque que orienta al uso de las unidades de procesamiento gráfico (GPU), estas últimas técnicas han crecido en popularidad gracias a los avances en su alto rendimiento y a la aparición de lenguajes específicos que permiten llevar problemas numéricos para ser tratados exclusivamente en estas unidades de procesamiento (Valdez, Domínguez, Rogers, Crespo, 2013).

Basándonos en resultados de trabajos realizados anteriormente es posible afirmar que el uso de GPU es un camino viable para la aceleración de códigos para la simulación de dinámicas de fluidos computacionales con implementaciones SPH. Los objetivos de este proyecto buscan explorar la eficacia en términos de aceleración en la obtención de resultados de implementaciones del método usando el paralelismo en múltiples GPU.

1.2 Metodología

La metodología del proyecto estuvo basada en procesos, los cuales fueron esenciales para crear una curva de aprendizaje sobre los temas tratados a lo largo de este, dichos procesos se organizaron de la siguiente forma:

1.2.1 Procesos

1.2.1.1 Estudio e investigación teórica y conceptual. Búsqueda de herramientas óptimas. Recopilación y estudio de todo el material teórico y conceptual requerido para la estructuración y el entendimiento del proyecto, así como la selección y capacitación de herramientas y tecnologías óptimas para el desarrollo de este. Dicho estudio se basa primariamente en el estudio del método SPH (su aplicabilidad en problemas de la ingeniería y su implementación en dinámicas de fluidos computacionales), con la finalidad de acelerar su ejecución y detectar evidencias de paralelismo y concurrencia.

Actividades:

- **a.** Estudio teórico de las dinámicas de fluidos computacionales
- **b.** Estudio y análisis del método SPH
- **c.** Comprensión e investigación del uso de SPH en problemas de fluidos computacionales.
- **d.** Investigación de los fundamentos teóricos para la paralelización y aceleración de algoritmos

1.2.1.2 Búsqueda, investigación y estudio de algoritmos que exploten concurrencia y paralelismo con implementación del método SPH. En esta etapa se indaga principalmente sobre algoritmos de código abierto, que implementen el uso del algoritmo SPH para problemas de CFD. En búsqueda de explotación de concurrencia y paralelismo, basados en esta investigación se

seleccionará un algoritmo apropiado para la aplicación de los procesos próximos; a su vez se realizará la capacitación en los lenguajes y herramientas necesarios.

Actividades:

a. Capacitación en la implementación de lenguajes y herramientas para la implementación del proyecto.

b. Investigación sobre algoritmos que implementen SPH, en búsqueda de concurrencia y paralelismo

1.2.1.3 Selección, estudio e implementación del algoritmo SPH sobre arquitecturas multi GPU. Este proceso va ligado al anterior en la investigación de algoritmos; teniendo las bases de los conceptos teóricos de CFD y SPH, se evaluará el algoritmo apartado sobre ciertas arquitecturas de interés en pro de determinar ciertas características que permitan cumplir la finalidad del proyecto. En esta etapa del proyecto el uso de clústeres de computación será vital, ya que nos proporcionará resultados sobre la escalabilidad del código.

Actividades:

a. Estudio a fondo del código, pruebas de ejecución sobre distintas arquitecturas GPU, ejecución del algoritmo sobre clústeres de super cómputo, perfilación del algoritmo, pruebas de speedup para medir la escalabilidad del algoritmo.

b. Despliegue del método SPH sobre arquitecturas computacionales con múltiples GPUs.

c. Planteamiento de experimentos en búsqueda de una aceleración en el código.

1.2.1.4 Experimentación y evaluaciones de la implementación. Una vez cumplido el proceso anterior es necesario evaluar la implementación realizada en varios aspectos para comprobar la hipótesis central del proyecto. Se buscará compilar el algoritmo sobre al menos dos

clústeres con arquitecturas diversas, realizando valoraciones de tiempos de ejecución, rendimiento y un estudio de la ejecución del algoritmo sobre dicha arquitectura puntual. Para este punto con base en los resultados se plantearán los lineamientos para la implementación del método sobre multi GPUs.

1.2.1.5 Documentación. La documentación del proyecto se plantea como un proceso sincrónico para cada etapa de este, donde se recopilará la información sobre el procedimiento, análisis y trabajo realizado para cada segmento, así como los resultados, información y avances del estudio.

Durante este capítulo se presentó un resumen del proyecto con aspectos importantes como el planteamiento y la justificación del problema, la metodología que se acompañara a lo largo del trabajo y la esquematización de procesos. A continuación, se discutirá acerca los objetivos a conseguir en la búsqueda de soluciones y respuestas al tratamiento del problema planteado.

2. Objetivos

2.1 Objetivo General

Implementar el método de hidrodinámica de partículas suavizadas (SPH –Smoothed Particle Hydrodinamics) en una simulación de fluidos computacional (CFD – Computational Fluid Dynamics) en arquitecturas con múltiples unidades de procesamiento gráfico (GPU – Graphics Processing Unit), con el objetivo de acelerar su ejecución.

2.2 Objetivos Específicos

Analizar el método SPH para comprender su uso en problemas de dinámicas de fluidos computacionales.

Analizar los algoritmos de implementación de SPH para identificar procesos que exploten concurrencia y paralelismo.

Estudiar la ejecución de una implementación SPH sobre arquitecturas GPUs para identificar las características de procesamiento.

Implementar el método SPH sobre arquitecturas computacionales con múltiples GPUs para alcanzar una mayor aceleración en la ejecución

Evaluar la ejecución del método SPH, sobre múltiples arquitecturas GPUs para proponer lineamientos de implementación.

Una vez presentados los objetivos del trabajo, se discutirán los aspectos conceptuales sobre la dinámica de fluidos computacionales y la hidrodinámica de partículas suavizadas, por otro lado, se detallarán aspectos sobre las tecnologías y las herramientas necesarias para el desarrollo.

3. Marco de Referencia

3.1 Marco Conceptual

3.1.1 Dinámicas de fluidos computacionales

La dinámica de fluidos computacional (CFD) es la ciencia que estudia la simulación de los fenómenos de flujos de fluidos implementando la solución de las ecuaciones de la conservación de la masa, momento, ecuaciones de Navier Stokes y energía, leyes de conservación de masa, momento y energía. Las técnicas CFD implementan el uso de computadores de alto rendimiento para dar soluciones aproximadas a algoritmos de ecuaciones numéricas.

Los resultados de CFD se pueden obtener basándose en distintas variables de un flujo, como su geometría, sus propiedades físicas y condiciones iniciales del campo del flujo, entre otros. La dinámica de fluidos computacional ha tenido un crecimiento importante en cuanto a la aparición de nuevos métodos numéricos para simular flujos de fluidos que buscan atacar el principal problema de sus resultados, ya que nunca son completamente exactos y es importante interpretar los resultados con cautela. (Woo, 2012)

3.1.2 Hidrodinámica de Partículas Suavizadas (SPH – Smoothed Particle Hydrodinamics)

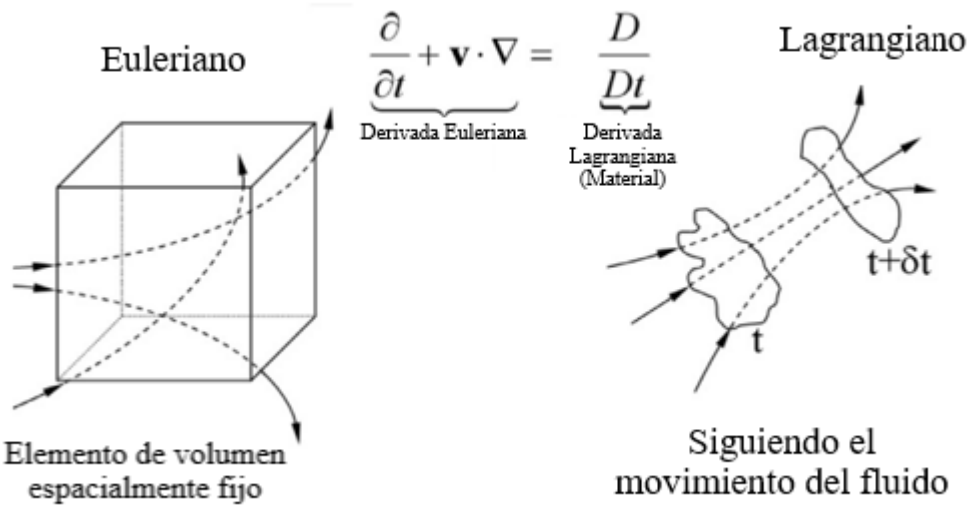
Tanto los métodos basados en malla o los métodos de partículas (o sin malla) son propuestos con el fin de aportar soluciones a problemas que suelen ser tratados con ecuaciones integrales. Su principal diferencia se encuentra en la implementación, los métodos sin malla

presentan una solución que usa un conjunto de nodos o partículas distribuidas arbitrariamente, las cuales están sujetas a fuerzas físicas y la interacción entre ellas en el paso del tiempo.

En la figura 1 se muestra una representación gráfica de cómo se comportaría el enfoque euleriano definido por un elemento de volumen espacialmente fijo y la representación del enfoque lagrangiano el cual se interesa más en las propiedades de cada partícula y el movimiento del flujo.

Figura 1

Representación euleriana y lagrangiana de las ecuaciones del flujo de fluidos



Nota. El gráfico presenta las diferencias en las ecuaciones de flujos de fluidos Eulerianos o basados en mallas y las ecuaciones Lagrangiana sin malla. De Smoothed particle hydrodynamics method for fluid flows, towards industrial applications: Motivations, current state, and challenges (p. 12), Por Shadloo, M. S., Oger, G., Le Touzé, D., 2016, Computer and fluids.

La hidrodinámica de partículas suavizadas (SPH) es un método que explota estas características, al ser un método sin malla, hace uso de la técnica de discretizar el continuo en un conjunto de nodos denominados partículas. Para la simulación de las dinámicas de fluidos se localiza una partícula de interés y se hace uso de las ecuaciones de Navier-Stroke, según las propiedades físicas de las partículas circundantes. Estas partículas vecinas se determinan mediante una función basada

en una distancia de suavizado comúnmente denotada como h. Los métodos sin malla se caracterizan en como las partículas se mueven en función de nuevas propiedades calculadas con cada paso de tiempo. Por otro lado, para la simulación basada en partículas se adaptan las leyes de conservación de la dinámica de fluidos del continuo a una forma adecuada usando ecuaciones integrales con base en funciones de interpolación, las cuales dan estimaciones de los valores en un punto específico. (Crespo, 2020).

El método presenta visibles ventajas en comparación con los métodos tradicionales eulerianos, en cuanto al modelado de problemas de superficies libres, o en deformaciones muy grandes como la ruptura de olas, ya que la interacción de las fuerzas de las partículas se genera en el tiempo. Según Liu, M. B. Liu, G. R. 2010, el método SPH es comparativamente más fácil en la implementación numérica y es más natural para desarrollar modelos numéricos tridimensionales que los métodos basados en mallas.

3.1.2.1 Aproximación Kernel. Como se habló anteriormente en SPH las partículas están dotadas de propiedades de la materia como masa, momento, temperatura, entre otras propiedades hidrodinámicas. El enfoque del método asume que los campos de la partícula de interés ‘r’ se interpolan a partir de todas las partículas del continuo. Con el fin de reducir complejidad y tiempo de cálculo, se reduce la contribución de partículas muy lejanas, SPH incluye efecto de partículas vecinas o adyacentes, haciendo implementación del uso de la distancia de suavizado *kh*, o también llamado dominio de soporte, donde *k* es una constante y *h* la longitud o distancia de suavizado. El dominio de soporte es una región localizada sobre la cual el núcleo es distinto de cero. Si la función delta de dirac en

$\int_\Omega f(\vec{r})\,\delta(|\vec{r}-\vec{r_0}|)d\vec{r}$ se reemplaza por una función de suavizado escrita como $W(|\vec{r}-\vec{r_0}|,h)$, la integral estimada o la aproximación del núcleo a una función arbitraria $f(\vec{r_0})$ se daría de la siguiente manera.

$$f(\vec{r}_0) \approx \langle f(\vec{r_0})\rangle \equiv \int_\Omega f(\vec{r})W(|\vec{r}-\vec{r_0}|,h)d\vec{r} \qquad (1)$$

Donde "⟨ ⟩" denota la aproximación del núcleo, $d\vec{r}$ es un elemento de volumen diferencial y Ω representa el volumen total del dominio.

La función núcleo debe satisfacer las siguientes condiciones,

- Para la normalización de la función se requiere que $\int_\Omega w(|\vec{r}-\vec{r_0}|,h)\,d\vec{r}=1$.
- A medida que la longitud de suavizado sea cero la función tiende a $\lim_{h\to 0} w(|\vec{r}-\vec{r}_0|,h) = \delta(|\vec{r}-\vec{r_0}|)$.
- Propiedad de soporte compacto del núcleo dado que $w(|\vec{r}-\vec{r}_0|,h)=0\ cuando|\vec{r}-\vec{r}_0|>kh$.
- El núcleo debe ser una función simétrica par donde $w(|\vec{r}-\vec{r}_0|,h)=w(-|\vec{r}-\vec{r_0}|,h)$. (Shadloo, M.S., Oger, G., Le Touzé, D. 2016. p 13).

3.1.2.2 Aproximación de partículas. De acuerdo con Shadloo, Oger y Le Touzé, (2016), una aproximación discreta de la ecuación (1), se obtiene de la sumatoria del conjunto de partículas vecinas así:

$$\langle f(\vec{r}_i)\rangle = \sum_j f(\vec{r}_j)W_{ij}V_j \qquad (2)$$

Donde $W(|\vec{r}_i-\vec{r}_j|,h)$, es reemplazado por W_{ij} y el volumen V_j es definido como $V_j = \frac{m_j}{\rho_j}$.

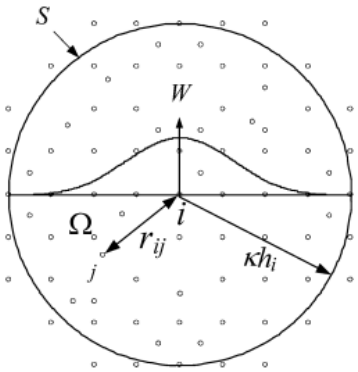
Y una aproximación SPH al gradiente de una función $f(\vec{r})$ se obtendría como

$$\left(\frac{\partial f(\vec{r}_i)}{\partial x_i^k}\right) = \sum_j f(\vec{r}_i)\frac{\partial w_{ij}}{\partial x_i^k}V_j \quad (3)$$

En la figura 2 se muestra una aproximación de partículas SPH en un dominio bidimensional con una superficie S. Donde W es la función núcleo usada para aproximar las variables de campo en la partícula de interés i, usando las sumas promediadas de las partículas que hacen parte del dominio de soporte que tiene una distancia kh_i.

Figura 2

Aproximaciones de partículas SPH



Nota. De Smoothed particle hydrodynamics (SPH): An overview and recent developments (p. 30), Por S Liu, M. B. Liu, G. R., 2010, Archives of Computational Methods in Engineering.

3.1.2.3 Ecuaciones de gobierno. En SPH se usan las ecuaciones de balance de masa y momento dadas en forma lagrangiana con notación directa, respectivamente dadas

$$\frac{D\rho}{Dt} = -\rho\nabla\cdot\vec{U} \quad (4)$$

Y

$$\frac{D\vec{U}}{Dt} = -\frac{\nabla p}{\rho} + \nabla\cdot T + \vec{f}^b + \vec{f}^S \quad (5)$$

Donde $\vec{U}$ es el vector velocidad del fluido, ρ la densidad de este, p es la presión, $\vec{f}^b$ y $\vec{f}^S$ son las fuerzas volumétricas del cuerpo y de la superficie respectivamente, $\frac{D}{Dt}$ es el operador de la derivada temporal del material.

Una forma discreta y básica de representar las ecuaciones de balance de masa y momento lineal se darían de la siguiente forma

$$\frac{D\rho_i}{Dt} = \rho i \sum_j \left(\overrightarrow{U_i} - \vec{U}_j\right) \cdot \nabla_i W_{ij} v_j \qquad (6)$$

Y

$$\frac{D\vec{U}_i}{Dt} = -\frac{1}{\rho_i} \sum_j \left(p_i \vdash p_j\right) \nabla_i w_{ij} v_j \qquad (7)$$

Donde W_{ij} se refiere a la función de suavizado y V_j es el volumen nuevamente.

En estas ecuaciones despreciamos algunas fuerzas como tensiones viscosas, fuerzas de superficie y del cuerpo. (Shadloo, M.S., Oger, G., Le Touzé, D. 2016. p 14).

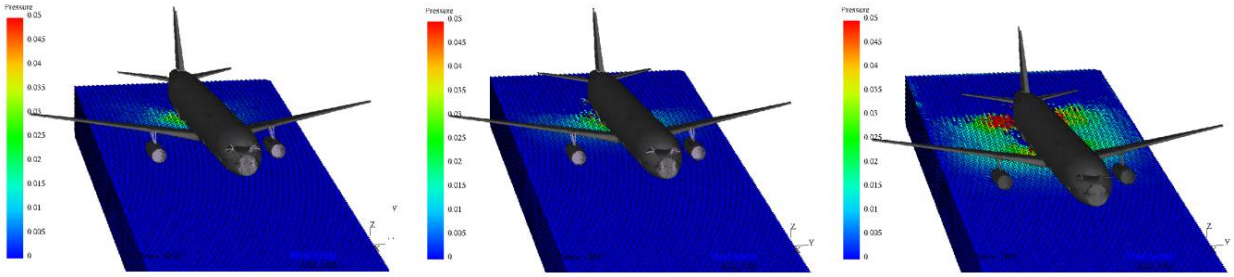
3.1.2.4 Aplicaciones. El objetivo inicial del método SPH era atacar los modelos de fenómenos astrofísicos, como el estudio de supernovas, colisiones estelares, simulaciones de estrellas, entre otros. Sin embargo, su uso se ha extendido a otras áreas como el estudio de problemas de mecánicas de flujos de fluidos y sólidos, esto es debido a sus ventajas en cuanto a los métodos basados en mallas, como la flexibilidad, debido a la falta de mallas provoca que el método sea más adecuado para la simulación de sistemas con geometrías más complejas, como grandes deformaciones y limites móviles. Su robustez es otra característica a resaltar, SPH es menos sensible a errores en las condiciones iniciales simplificando el modelo del problema lo que lo hace más seguro frente a problemas como la incertidumbre o al problema de una entrada de número n con un comportamiento dinámico algo muy similar las

simulaciones de partículas basadas en el método de Montecarlo (Kalos, 2007), sumado a otras características como su escalabilidad ya que puede paralelizarse fácilmente permitiendo que pueda ser usado para sistemas de gran escala en sistemas de alto rendimiento y la precisión que contiene para capturar choques y características criticas influyeron que el método se desplazara a otras áreas del área de la física.

En la figura 3 se muestra un ejemplo de uso de SPH ilustrando varios instantes de tiempos de la simulación del aterrizaje forzoso de un avión A321 FEM tras el contacto con el agua SPH. El agua esta inicialmente en reposo y el avión posee unas velocidades de impacto horizontal y vertical de 60m/s y 1.5 m/s respectivamente con 8 grados iniciales de cabeceo positivo.

Figura 3

Simulación de ruptura de presas usando el método SPH



Nota. Tomada de Structural Loading of a Complete Aircraft Under Realistic Crash Conditions: Generation of a Load Database for Passenger Safety and Innovative Design. De Ortiz, R., Charles, J. L., & Sobry, J. F. (2004)

3.1.2.5 Costos computacionales. Uno de los principales problemas de SPH es su alto costo computacional, esto se debe a diversos factores como la gran cantidad de partículas que debe ser simulada para un flujo y la constante interacción entre ellas. Las soluciones adaptadas para mitigar estos problemas van desde limitar el flujo a un número determinado de partículas o la

implementación de medidas de paralelismo en los procesos SPH como la portabilidad a unidades de procesamiento grafico o a arquitecturas que permitan el procesamiento masivamente paralelo.

3.2 Marco Tecnológico

3.2.1 C++

C++ es un lenguaje de programación, conocido como una extensión del lenguaje C que fue creado principalmente para el uso de clases e implementación de la programación orientada a objetos. Una de las características principales de C++ es la implementación de punteros y su compatibilidad con bibliotecas. (Stroustrup, Bjarne, 1997)

El lenguaje de programación C++ es ampliamente utilizado en la industria y el mundo académico y está definido por el estándar ISO/IEC 14882:2020 (Standard, 2020).

3.2.2 CUDA

CUDA (Compute Unified Device Architecture - Arquitectura Unificada de Depósitos de Cómputo) es una plataforma de computación en paralelo y modelo de programación, su desarrollo está orientado principalmente al uso de unidades de procesamiento gráfico (GPU) donde se ejecutará parte del cómputo intensivo de la aplicación a acelerar. Sus lenguajes soportados se extienden a C, C++, Fortran (Backus, 1957), Python y MATLAB dentro de los cuales se expresa el paralelismo con el uso de palabras clave básicas. (CUDA, 2021)

El kit de desarrollo CUDA incluye bibliotecas aceleradas, compilador, herramientas de desarrollo y herramientas de medición e instrumentos de análisis de ejecución.

3.2.2.1 Perfilador Visual Profiler, esta herramienta viene incluida en el toolkit de CUDA, básicamente ofrece la información importante que ayuda en la optimización de las aplicaciones desplegadas con CUDA C/C++, su función es la creación de perfiles de rendimiento. (NVIDIA, 2021)

Al momento de la escritura de este libro está disponible el visualizador NVIDIA Nsight System, al igual que el Visual Profiler es una herramienta de análisis de rendimiento diseñada para visualizar oportunidades de optimización de algoritmos. (NVIDIA, 2022)

3.2.3 Clústeres

Un clúster es un conjunto de múltiples computadores interconectados llamadas nodos que pueden trabajar unidas en pro de un problema informático de alto rendimiento. Suelen usarse con propósitos científicos a gran escala en campos donde es necesario procesar grandes cantidades de datos en un corto tiempo. Cada nodo es un computador independiente con su propio procesador, memoria y almacenamiento, esta arquitectura permite integrar tarjetas gráficas (GPUs) y sistemas de procesamiento masivamente paralelo para el procesamiento de dichas tareas. (Foster, 1995)

3.2.4 Visual Studio Code

Editor de código fuente ligero desarrollado por Microsoft que se ejecuta en el escritorio, soportado en Linux, Windows y macOS. Cuenta con múltiples extensiones para lenguajes como C, C++, C#, Java, Python, entre otros. Su objetivo es dar comodidad al desarrollo, escritura y lectura de código. (VisualStudio, 2021)

3.2.5 MobaXTerm

Es una terminal mejorada para Windows, que ofrece ciertas herramientas que hace más fáciles las conexiones remotas a servidores remotos (MobaXterm 2008). Será usada para facilitar la conexión a los clústeres y líneas para la ejecución del algoritmo que usaremos en el proyecto.

Una vez abarcados los conceptos fundamentales para el desarrollo de este trabajo, en el siguiente capitulo se introduce al estudio y despliegue de simulaciones CFD con el método SPH,

se indaga acerca de herramientas y trabajos previos y se trabajara sobre estos para el estudio de procesos SPH y la paralelización de procesos.

4. Desarrollo de la metodología

4.1 Estudio conceptual y elección de tecnologías

Para el desarrollo del proyecto fue necesario adquirir conocimientos tanto teóricos como prácticos en ciertas áreas que son de importancia. Inicialmente se estudiaron los conceptos sobre paralelismo y concurrencia. También se examinaron y seleccionaron las tecnologías, lenguajes y herramientas que se adaptaban a las necesidades de los objetivos trazados. A su vez se estudiaron los conceptos sobre SPH y su aplicación en la ingeniería.

Como se ha presentado, los problemas de las simulaciones de dinámicas de fluidos computacionales provienen del alto flujo de datos que se deben procesar para obtener resultados de una simulación con fuerzas físicas de características significativas, para esto el uso de las GPU parece ser una de las soluciones óptimas gracias a la tecnología que ofrecen y su alto nivel de procesamiento. Con este fin es necesario encontrar herramientas que sirvan para llevar el problema de los métodos basados en partículas como lo es SPH a un entorno de múltiples GPU y evaluar su ejecución, rendimiento y aceleración.

En el campo de la aceleración y optimización de algoritmos surgen dos conceptos fundamentales que hay que tener en cuenta en el diseño e implementación de los algoritmos, si se quiere alcanzar un rendimiento mayor al momento de la ejecución. Por una parte, se conoce como concurrencia en la computación, a la característica que tiene un código o algoritmo que permite que una serie de instrucciones se ejecuten por separado sin llegar a afectar su resultado, esta característica es crucial en la aceleración de códigos, ya que en función de ella es posible hacer

uso de otro concepto clave en esta tarea. El paralelismo, que hace referencia a la ejecución de tareas de manera simultánea hablando de múltiples hilos o múltiples procesadores. Existen herramientas calificadas en la computación en paralelo como la API OpenMP que se dedica a al uso de memoria compartida o estándares como MPI que define sintaxis para el paso de mensajes en múltiples procesadores, estos son muy usados en la investigación (Tejaxún Xicón, 2019)

Para la computación en paralelo sobre GPU CUDA se centra en el uso de estas en conjunto con las unidades de procesamiento central, donde la GPU ejecuta la parte más costosa computacionalmente de la aplicación. (Pérez, Nebreda & Ortega, 2016)

Al tener soporte de uno de los principales fabricantes de unidades de procesamiento gráfico, se hace interesante para el desarrollo del proyecto, estudiar y analizar la forma y resultados que ofrece, por lo que fue una de las herramientas seleccionadas durante el progreso del trabajo.

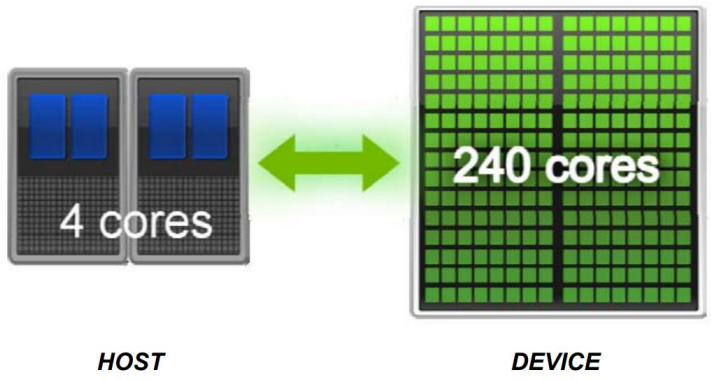
Para el desarrollo de los objetivos, se hace fundamental el uso de clústeres de super cómputo, en los que su arquitectura pueda ofrecer el manejo de al menos dos GPUs en un mismo nodo. Igualmente es elemental evaluar el desempeño de la implementación en un segundo clúster, en el cual la arquitectura de sus GPUs difiera del primero, con esto se pretende valorar el rendimiento que puede ofrecer la solución en función de las unidades de procesamiento gráfico.

En la elección de herramientas y tecnologías, y teniendo en cuenta el despliegue sobre arquitecturas con múltiples GPU, se hace preciso disponer de hardware que supla esta necesidad. Para lograr este objetivo se hace uso de computadores de alto rendimiento, clústeres o modelos de computación heterogéneos, es decir que usan más de un tipo de procesador, en este caso CPU y GPU. Y para evaluar las posibles diferencias que pueda ofrecer en torno al despliegue y aceleración un cambio de esta arquitectura se valoraran dichas ejecuciones en más de una de estas arquitecturas.

En la figura 4 se tiene una representación de una unidad CPU y GPU llamados como host y device respectivamente.

Figura 4

Modelo de computación en sistemas heterogéneos



Nota. De Introducción a la programación en CUDA *(*p. 5), Por Pérez, C. R., Nebreda, J. M. C., & Ortega, P. L. S. 2016.

4.2 Estudio de algoritmos y antecedentes

El enfoque dado a la investigación en este trabajo está dado a la paralelización de procesos mediante el uso de arquitecturas computacionales con el uso de GPUs, para esto fueron estudiados previamente los conceptos básicos de herramientas y tecnologías. Es importante para la investigación analizar estos términos en acción, para esto se investigará acerca de implementaciones de algoritmos en búsqueda esencialmente en la implementación del método SPH y la explotación de concurrencia y tecnologías en paralelización.

Para esto se seleccionaron algunos criterios dentro de los algoritmos en busca de características puntuales en las tecnologías con las que se implementaron, sus avances en el paralelismo y su uso en problemas de la ingeniería. Como primer acercamiento se planteó un

estándar básico para probar el funcionamiento y la utilización de recursos de cada código, se obtuvieron códigos fuentes de los repositorios, y se realizó un despliegue en una arquitectura bastante básica, pero con la arquitectura necesaria para este primer estudio, tabla 1.

Tabla 1

Especificaciones arquitectura primer despliegue.

Hardware	Especificaciones
CPU	Intel Core i5 – 8300H (4 núcleos a 2,30 GHz)
GPU	Nvidia Geforce GTX 1050 – 4GB
RAM	12 GB (2400 MHz)
Sistema Operativo	Windows 10

Para hacer una primera valoración del estado del proyecto y la viabilidad de uso para los propósitos de este trabajo, se obtendrá el código fuente, se instalará sobre esta máquina y se realizará un despliegue de prueba. Adicionalmente se añadió peso a la contribución que tiene cada proyecto a la comunidad científica SPH y su uso en estudios sobre tema.

4.2.1 DualSPHysics

DualSPHysics es un algoritmo de código libre, desarrollado principalmente por la universidad de Vigo y la Universidad de Manchester, el código hace implementación del método de hidrodinámica de partículas suavizadas y es usado para la simulación de dinámicas de fluidos

computacionales. DualSPHysics implementa el uso principalmente de dos suavizados de kernel cubic spline y Quintic. (DualSPHyscis, 2022).

Este código es uno de los más apoyados por la comunidad científica y es muy usado para la resolución de problemas actuales de la ingeniería, como uso en extracción de hidrocarburos, problemas multi físicos, rebosamiento de presas y problemas clásicos como colisiones contra estructuras costeras, simulación de oleaje.

4.2.1.1 Tecnologías. DualSPHysics implementa el método SPH sobre el lenguaje C++ y usa la plataforma CUDA para la paralelización de procesos sobre GPU, esto facilita su uso tanto en CPU como en GPU. En cuanto a la administración de memoria y gestión de memoria se apoya con implementaciones de OpenMP para la administración de memoria y MPI para las ejecuciones en multinúcleo.

4.2.1.2 Documentación.

La documentación complementa mucha información de la implementación del código, aunque se queda desactualizada con algunas de las implementaciones más recientes, aunque dicha información se puede encontrar en los foros de la comunidad del proyecto. Aquí se encuentra toda la información para la utilización de este y la configuración para desarrolladores e instrucciones para la creación de nuevos problemas. También contiene información detallada de los módulos desarrollados en el proyecto para el uso de CPU y GPU, además de toda la formulación SPH para la implementación del método, ecuaciones de gobierno, viscosidad, ecuaciones de estado, ecuaciones de densidad y las ecuaciones de suavizado o función kernel.

4.2.1.3 Código fuente. Se tomó la versión 5.0.2.2 del repositorio en GitHub, el proyecto contiene preconfigurados algunos ejecutables con problemas CFD dentro de su código fuente,

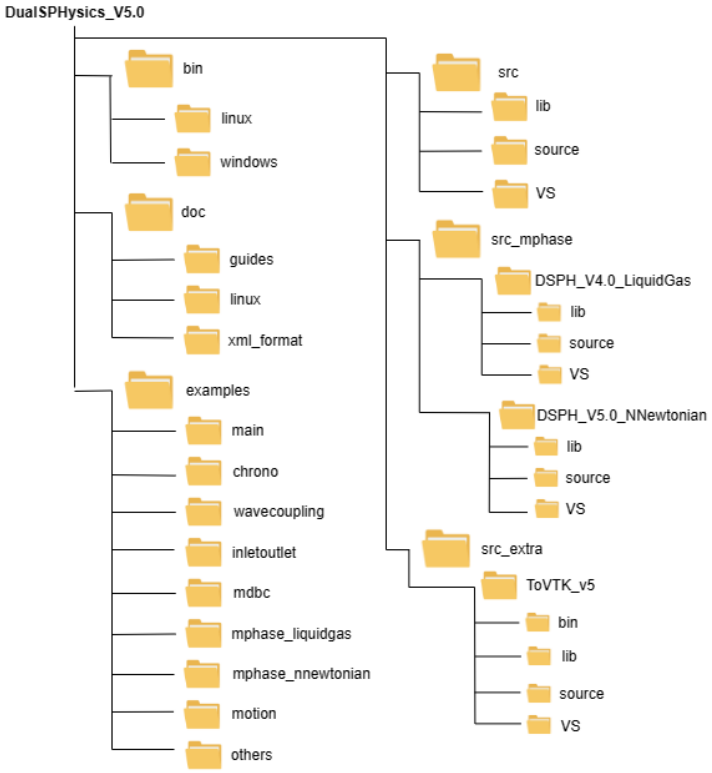
como primer ejercicio se usará un ejemplo clásico de ruptura de presas. (Ramos, Pérez-Sánchez 2018).

Es necesario mencionar que DualSPHysics es un codigo de libre acceso y su codigo fuente se puede encontrar en github.com/DualSPHysics/DualSPHysics.

La figura 5 ilustra la estructura organización de las carpetas del código DualSPHysics.

Figura 5

Estructura de carpetas del código fuente



Adaptada de DualSPHysics wiki (s.f)

4.2.1.4 Despliegue y resultados. Para esta ejecución no se requiere una compilación previa del proyecto DualSPHysics, el código contiene ejecutables para el ejemplo mencionado

configurados para ejecutar el problema CFD mencionado anteriormente sobre CPU y GPU para Microsoft Windows.

El proyecto se despliega en dos partes, en la primera se contiene toda la parte matemática fuerte del método SPH, en este se definen las ecuaciones, se leen los parámetros de la simulación, se calculan listas de vecinos y se realizan todas las operaciones de interacción entre partículas. La segunda parte de la ejecución pertenece a un post procesado que como resultado arroja los archivos de visualización de la simulación calculada.

Se realizará un despliegue sencillo con un tiempo limitado y sin tener en cuenta el número de partículas a simular ni las dimensiones de esta. Se ejecutará durante el mismo periodo de tiempo sobre la CPU y paralelo a GPU para realizar una comparación simple entre ambos resultados.

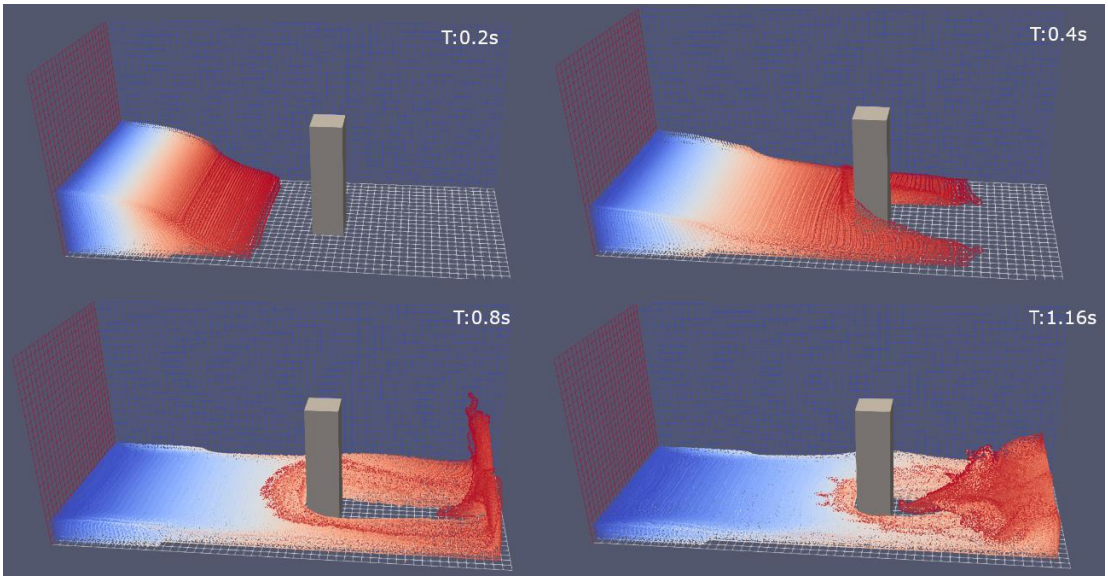
4.2.1.4.1 Resultados CPU. El procesamiento sobre CPU fue exitoso, se limitó en este caso el tiempo a quince minutos ya que es posible que los tiempos de ejecución sobre estas unidades sean demasiado extensos. Durante este tiempo se procesó un 10% de la simulación completa.

4.2.1.4.2 Resultados GPU. La ejecución del proyecto sobre GPU del mismo modo fue exitosa, los resultados fueron mucho más eficaces que en la unidad de procesamiento central, durante un periodo más corto de seis minutos la ejecución paralela pudo completar el 100% de la simulación y se obtuvieron los archivos de visualización correctamente.

Para la presentación de resultados es necesario aclarar que se garantizó la repetibilidad para realizar una verificación de los resultados alcanzados en cada despliegue. Por otra parte, es importante mencionar que la simulación completa de rompimiento de olas contiene un total de 160 archivos de visualización, en base a esto se calcularon los porcentajes presentados anteriormente

Figura 6

Simulación visual de ruptura de presas.



Nota. Instantes de tiempo simulación ruptura de presas con DualSPHysics sobre GPU.

La figura 6 muestra distintos instantes de tiempo de la visualización alcanzada con los archivos generados por el código DualSPHsycs.

4.2.2 PySPH

Es un modelo SPH de simulación para la hidrodinámica de partículas suavizadas de código libre, para simulaciones de flujos de fluidos con implementación del método de hidrodinámicas de partículas suavizadas (SPH), el código está desarrollado sobre Python puro, las partes críticas del método se implementan sobre Cython que es un compilador para escritura de extensiones de C o C++ de forma nativa sobre python, el código utiliza implementaciones PyOpenCL para el acceso a la API OpenCL de computación paralela. (PySPH documentation, s. f.)

4.2.2.1 Tecnologías. PySPH está desarrollado sobre Python puro que se convierte a Cython para soporte sobre C y C++ y OpenCL ya que utiliza PyOpenCL para la implementación de tareas

en paralelo en multiprocesadores, Es posible configurar el proyecto para usar OpenMP, OpenCL y MPI. (PySPH documentation, s. f.)

4.2.2.2 Documentación. La documentación de PySPH contiene la información fundamental para la instalación y despliegue del proyecto, información sobre toda la formulación SPH usada en el desarrollo del método, también información sobre las dependencias del proyecto, una de las características negativas del código es que el uso de OpenCL aún es experimental y la información de cómo usar y compilar el proyecto es algo escasa, detalles de la configuración de módulos y métodos para nuevas simulaciones.

En cuanto a apoyo científico el proyecto ha sido usado en diferentes campos de aplicación de CFD. Aunque es de destacar que el fuerte del proyecto no son sus implementaciones paralelas en GPU.

4.2.2.3 Código fuente. El código fuente se puede encontrar en GitHub, y se tomó la única versión disponible 1.0b1, el proyecto tiene configurados algunos problemas CFD clásicos de la ingeniería con los que se puede realizar despliegues rápidos. (Ihme, Colonius, 2015)

Vale la pena señalar que PySPH es un codigo abierto y la última versión del codigo se puede encontrar en github.com/pypr/pysph

4.2.2.4 Despliegue y resultados.

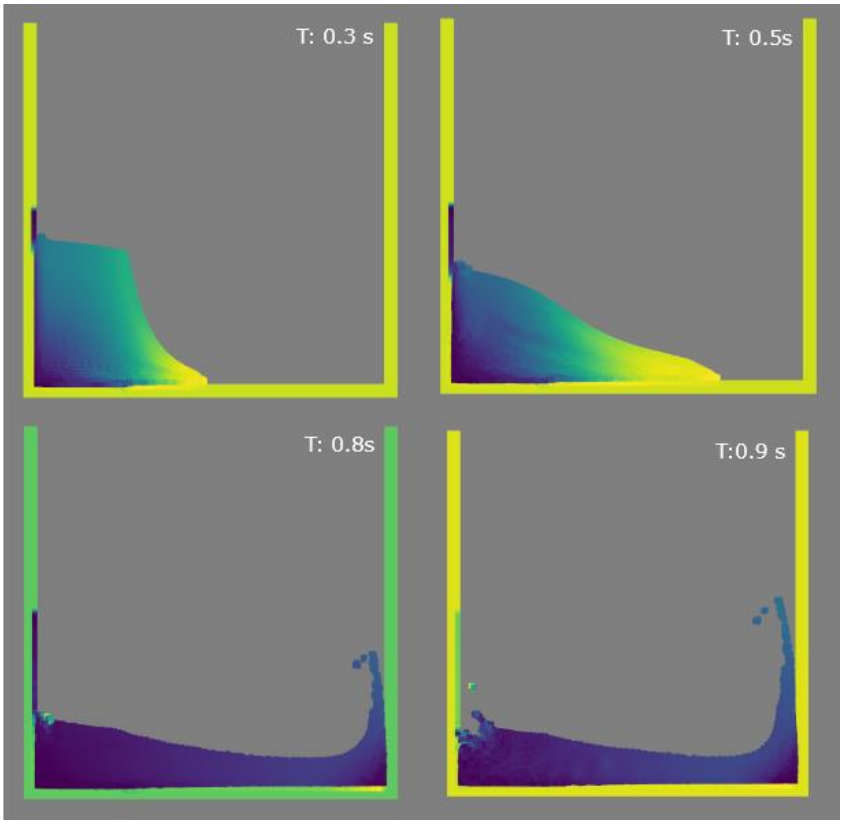
Como ya se mencionó el despliegue sobre GPU de este código es experimental, por lo que la ejecución sobre GPU no está del todo implementada y no se encuentra la documentación acerca de ello. El código fuente contiene preconfigurados algunos ejemplos CFD clásicos del método SPH, se utilizará el problema rompimiento de olas 2D como primer acercamiento con el código. El despliegue se realizó haciendo uso únicamente de la CPU de la máquina y se obtuvieron tiempos de respuesta bastante elevados en torno a los quince minutos de procesado. El código arroja como

archivos de salida los archivos con todos los cálculos SPH, así como los archivos para la visualización de la simulación.

En la figura7 podemos ver la visualización de distintos instantes de tiempo de la simulación de ruptura de olas alcanzada con el codigo PySPH

Figura 7

Simulación visual de ruptura de presas 2D.



Nota. Instantes de tiempo simulación ruptura de presas 2D PySPH.

4.2.3 SPlishSPlasH

SPlishSPlasH es una biblioteca de código abierto que se basa en el método SPH, para lograr simulaciones físicas de fluidos computacionales. Adicionalmente SPliSPlasH integra

solucionadores de presiones poderosos como (WCSPH, PCISPH, PBF, IISP, DFSPH, PF) para simular flujos incomprensibles, en los cuales la densidad a lo largo del flujo es aproximadamente la misma, esto hace que el volumen no varie a lo largo del paso del tiempo. (SPlisHSPlasH documentation, s. f.).

4.2.3.1 Tecnologías. SPlishSplash está implementado sobre Python y C++, a su vez ejerce el uso de librerías propias como cuNSearch implementada en C++/CUDA exclusivamente para la búsqueda de vecinos en la GPU. Otro de sus agregados es la implementación openMP para ejecuciones multinúcleo.

4.2.3.2 Documentación. En la documentación no se encuentra la formulación SPH usada para el proyecto. Se encuentran guías generales de instalación y compilación, información sobre la arquitectura del software y sus clases, guías de creación de escenarios e información sobre cómo usar las diferentes herramientas que integra el proyecto

4.2.3.3 Código fuente. Se tomó la versión 2.11.0 alojada en el repositorio de Github, esta versión tiene algunos escenarios configurados, con el fin de realizar una prueba similar en los proyectos estudiados se usará el ejemplo clásico de ruptura de presas 3D. (Schäfer, Düvel 2015)

El software SPlishSPlasH es un codigo abierto y su codigo fuente está disponible en github.com/InteractiveComputerGraphics/SPlisHSPlasH

4.2.3.4 Despliegue y resultados. El proyecto cuenta con una interfaz gráfica que permite variar de manera intuitiva algunos parámetros tanto de la simulación y formulación de SPH como del modelo del fluido, dentro de estos se encuentra la elección de una función núcleo específica con la que se ejecuta el método, algunos parámetros de simulación, el tamaño mínimo y máximo de iteraciones, un máximo de error de densidad, es posible variar la viscosidad del fluido, los

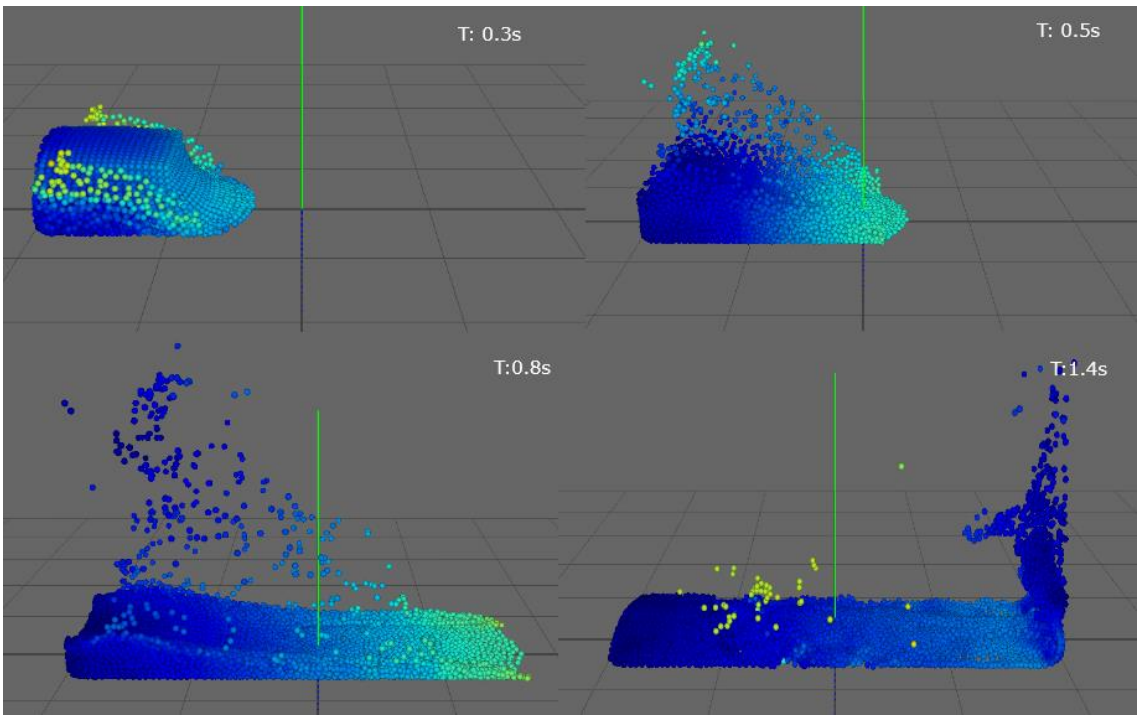
colores del mapeo influyendo en que característica se quiere visualizar en la renderización de la visualización como ejemplo velocidad, densidad, entre otras propiedades.

El proyecto se hace mucho más dinámico al momento de la renderización de la simulación, ya que hace posible jugar con ciertos parámetros y los modelos del fluido, permitiendo aplicar fuerzas externas de una manera fácil y modelos de estructuras precargadas en el código fuente para que influyan directamente en la estructura del problema.

En la figura 8 se muestra distintos instantes de tiempo de la simulación de ruptura de olas desplegada con el codigo SPlishSPlasH.

Figura 8

Simulación ruptura de presa



Nota. Instantes de tiempo en simulación ruptura de presas del código SPliSPlasH.

4.3 Estudio, análisis e implementación del algoritmo SPH sobre arquitecturas multi GPU.

La justificación sobre la que se basó este trabajo es el estudio de los problemas que se presentan en las simulaciones de dinámicas de fluidos, primordialmente desde la implementación del método de hidrodinámica de partículas suavizadas. Como se ha discutido a lo largo de este trabajo el uso de unidades de procesamiento gráfico se propone como una potente herramienta para acelerar las simulaciones sobre SPH. El uso de GPUs habilita la posibilidad de realizar simulaciones con un número más elevado de partículas en tiempos relativamente más cortos. Aunque esta implementación no es del todo sencilla y se requiere de una comprensión del método y de las partes críticas que se deben paralelizar para la obtención de mejores resultados.

Los trabajos propuestos anteriormente son algoritmos que presentan avances en la implementación de simulaciones SPH sobre GPU, es decir que explotan la concurrencia y la paralelización de procesos en su desarrollo. A continuación, se enfocan los esfuerzos en el estudio y análisis de las implementaciones del proyecto DualSPHysics, su desarrollo, las tecnologías aplicadas y la técnica de paralelización del método SPH sobre una única GPU, por último, se discuten los factores y herramientas necesarias para la implementación de este código sobre arquitecturas multi-GPU. El criterio de elección de DualSPHysics para este trabajo se baso en los resultados obtenidos y presentados anteriormente, a su vez, las tecnologías que actualmente usa el proyecto son de interés para el campo de investigación de este trabajo e igual de importante su implementación de SPH es bastante compacta y completa, lo que puede traer beneficios a esta investigación.

4.3.1 DualSPHysics

Una vez recopilados datos fundamentales de algunos antecedentes en implementaciones SPH sobre GPUs, y habiendo usado algunos de estos códigos de manera simple, durante este

trabajo se discutirá acerca de las características y estrategias que fueron usadas para ello. Destinado a este objetivo se tomará el algoritmo DualSPHysics presentado anteriormente y que contiene algunas de las implementaciones más completas acerca de este tema, para estudiar y analizar su despliegue y encontrar factores fundamentales a la hora de la aceleración del código basándonos en las tareas realizadas por el método SPH. También es necesario discutir características necesarias al momento de llevar este método a arquitecturas multi-GPU.

4.3.1.1 Estructura del código.

De forma general se analiza la estructura del código del proyecto, aunque no se hará ningún desarrollo de características nuevas al proyecto es vital entender como fue implementado. DualSPHysics presenta una estructura de código orientada a objetos, que lo hace más limpio, entendible, y fácil de mantener y de entender. En cuanto a la organización de los archivos, el código fuente de DualSPHysics tiene una serie de implementaciones básicas como funciones matemáticas, implementaciones de matrices, métodos de implementaciones de partículas, implementaciones de clases para el control de tiempo, entre otras. Véase tabla 2.

Tabla 2

Archivos comunes DualSPHysics

Archivos	Descripción
Functions(.h .cpp)	Declara funciones básicas/generales para toda la aplicación
FunctionsMath (.h .cpp)	Declara/implementa funciones matemáticas generales/básicas
JDsphConfig(.h .cpp)	Declara/implementa la clase que carga la configuración desde DphConfig.xml.
JException (.h .cpp)	Declara/implementa la clase que define excepciones con la información de la clase y el método.

JMatrix4.h	Declara la plantilla para una matriz 4x4 utilizada para la transformación geométrica de puntos en el espacio.
JMeanValues(.h .cpp)	Declara/implementa la clase que calcula el valor medio de una secuencia de valores.
JObject (.h .cpp)	Declara/implementa la clase que define objetos con métodos que arrojan excepciones.
JParticlesDef.h	Define el tipo de partículas y los métodos básicos para las partículas.
JPeriodicDef.h	Define el tipo de condiciones periódicas y las funcionalidades básicas.
JRadixSort(.h .cpp)	Declara/implementa la clase que implementa el algoritmo RadixSort.
JRangeFilter(.h .cpp)	Declara/implementa la clase que facilita el filtrado de valores dentro de una lista.
JReadDatafile (.h .cpp)	Declara/implementa la clase que permite leer datos en archivos ASCII.
JSaveCsv2 (.h .cpp)	Declara/implementa la clase que crea los archivos CSV.
JSpaceCtes (.h .cpp)	Declara/implementa la clase que administra la información de las constantes del archivo XML de entrada.
JSpaceEParms (.h .cpp)	Declara/implementa la clase que administra la información de los parámetros de ejecución del archivo XML de entrada.
JSpaceParts(.h .cpp)	Declara/implementa la clase que administra la información de las partículas del archivo XML de entrada.
JSpaceProperties(.h .cpp)	Declara/implementa la clase que administra las propiedades asignadas a las partículas en el archivo XML
JTimeControl(.h .cpp)	Declara/implementa la clase que administra la información del tiempo de ejecución en procesos largos.
JTimer.h	Declara la clase que define una clase para medir intervalos de tiempo cortos.

JTimerClock.h	Declara la clase para medir intervalos de tiempo con precisión de reloj.
JXml(.h .cpp)	Declara la clase que ayuda a administrar el documento XML mediante la biblioteca TinyXML.
TiposDef.h	Declara tipos y funciones generales para toda la aplicación.

Nota. Los archivos relacionados en esta tabla contienen implementaciones básicas fundamentales. Por Crespo, A. J. C. 2020, DualSPHysics documentation.

En la tabla 3 encontramos el corazón de la implementación SPH, así como implementaciones para el uso de MPI, datos iniciales de partículas, implementaciones de atributos compartidos y atributos para métricas. Estos módulos están diseñados para ser usados tanto en CPU como en GPU.

Tabla 3

Archivos comunes GPU y CPU

Archivos	Descripción
main.cpp	Archivo principal del proyecto que ejecuta el código en CPU o GPU
JCfgRunBase(.h .cpp) JDsGaugeItem (.h .cpp) JDsGaugeSystem(.h .cpp) JPartsLoad4 (.h .cpp)	• Recopilar parámetros de ejecución por la línea de comandos. • Implementa la clase común de los diferentes tipos de medidores. • Implementa la clase que administra la lista de indicadores configurados. • Declara la clase que administra la carga inicial de datos de partículas.
OmpDefs.h	Define constantes para usar MPI
TypesDefs.h	Define tipos específicos para la aplicación SPH

JSph (.h .cpp)	Declara/implementa la clase que define todos los atributos y funciones que comparten las simulaciones de CPU y GPU.

Nota. Archivos compartidos para simulaciones en CPU y GPU. Por Crespo, A. J. C. 2020, DualSPHysics documentation.

Para terminar en la tabla 4 se sintetizan los archivos donde se implementa el despliegue de las simulaciones sobre CPU o la paralelización en GPU. Analizando el código aquí se encuentran las configuraciones de los kernels CUDA y asignaciones de memoria que van a ser lanzados durante la simulación.

Tabla 4

Archivos de implementación en GPU

Archivos	Descripción
JArraysGPU (.h .cpp)	Declara/implementa la clase que administra matrices con memoria asignada en GPU.
JCellDivGpu(.h .cpp)	Declara/implementa la clase responsable de generar la lista de vecinos (NL) en la GPU.
JcellDivGpu_ker(.h .cu)	Declara/implementa funciones y kernels CUDA para generar el NL en GPU.
JcellDivGpuSingle(.h .cpp)	Declara/implementa la clase que define la clase responsable de generar el NL en Single-GPU.
JcellDivGpuSingle_ker(.h .cu)	Declara/implementa funciones y kernels CUDA para calcular operaciones del NL.
JdsGauge_ker (.h .cu)	Declara/implementa funciones y kernels CUDA para clases que administran medidores.
JsphGpu (.h .cpp)	Declara/implementa la clase que define los atributos y funciones utilizados solo en las simulaciones de GPU.

JsphGpu_ker(.h .cu)	Declara/implementa funciones y kernels CUDA para la Interacción de Partículas y actualización del sistema.
JsphGpuSingle(.h .cpp)	Declara/implementa la clase que define los atributos y funciones utilizados solo en una sola GPU.
JsphGpuSingle_inOut(.h .cu)	Implementa las funciones inOut de la clase JsphGpuSingle
JSphTimersGPU.h	Mide los intervalos de tiempo durante la ejecución de la GPU.
JdsAccInput_ker(.h .cu)	Implementa un conjunto de funciones y kernels CUDA para fuerzas externas (JdsAccInput) en la GPU.
JsphShifting(.h .cu)	Implementa un conjunto de funciones y kernels CUDA para la corrección del desplazamiento en la GPU

Nota. Archivos implementados para el despliegue de simulaciones sobre GPU. Por Crespo, A. J. C. 2020, DualSPHysics documentation.

4.3.1.2 Implementación DualSPHysics en CPU y GPU. Para entender cómo se desarrolló e implementó DualSPHysics sobre GPU se deben estudiar los procesos necesarios para el funcionamiento del método SPH e identificar aquellos que se puedan beneficiar con la paralelización. Comúnmente se trata de partes del código donde se calcula la interacción entre las partículas. Los procesos de SPH se pueden dividir de esta manera:

4.3.1.2.1 Generación de la lista de vecinos. El dominio se divide en bloques de tamaño 2h (o el tamaño del dominio del núcleo), se genera una lista de partículas ordenadas según el dominio al que pertenecen.

4.3.1.2.2 Calculo de fuerzas resultado de la interacción entre partículas. Las partículas cercanas en un radio de 2h son candidatas a vecinas y cada partícula individual interactúa con todas sus partículas vecinas.

4.3.1.2.3 Paso de tiempo. Se actualizan las propiedades físicas de las partículas en cada paso de tiempo, esta información se almacena localmente en momento definidos durante la ejecución. (Crespo, 2020)

4.3.1.3 Implementaciones en paralelo. Una vez identificados aquellos procesos que requieren de una carga computacional elevada es posible entrar a la aplicación de técnicas para la paralelización de estos.

4.3.1.3.1 OpenMP. Una de las técnicas desarrolladas en DualSPHyics es el cálculo de las interacciones de partículas sobre CPU multinúcleo. Con el uso de OpenMP DuaSPHysics se puede aprovechar los múltiples núcleos de la unidad de procesamiento central, distribuyendo la carga de trabajo entre los núcleos disponibles. Adicionalmente también se permite la paralelización de otras partes de código como las operaciones de entrada/salida, condiciones de contorno y el postprocesado de la aplicación, pero su uso principal es el cálculo de interacciones entre partículas. (Domínguez, Crespo & Gómez-Gesteira. S.f)

4.3.1.3.2 MPI. DualSPHysics implementa MPI para la paralelización del cálculo de interacciones de partículas en múltiples nodos de computación permitiendo la ejecución como si se tratara de una sola máquina para que la simulación aproveche los recursos computacionales de cada nodo, esto proporciona una aceleración significativa a la simulación haciendo posible realizar simulaciones con mayor número de partículas. (Crespo, 2020)

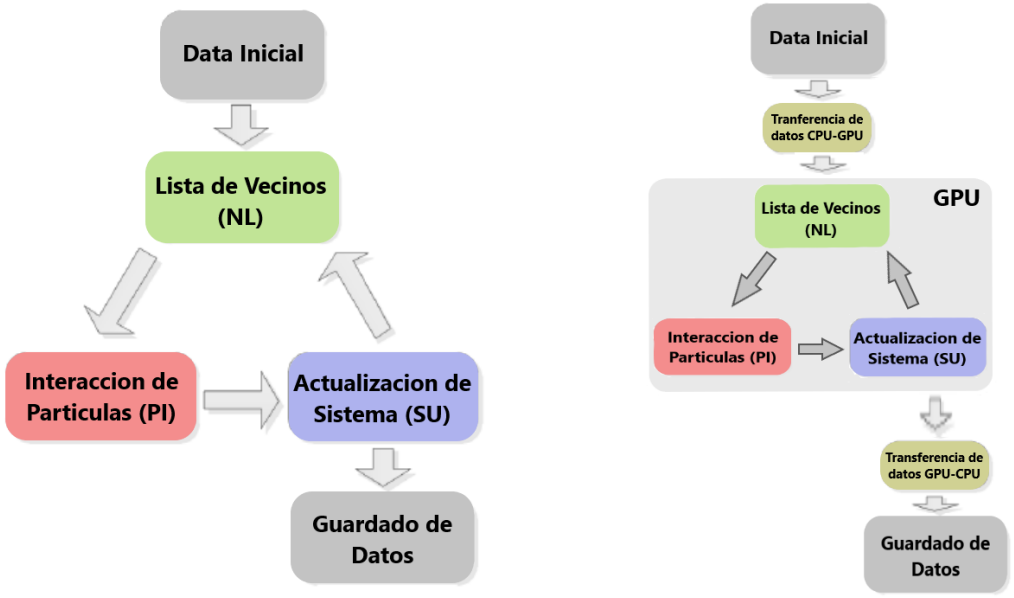
4.3.1.3.3 CUDA. Las implementaciones en paralelo con GPUs vienen de la mano con la plataforma desarrollada con NVIDIA. Dichos avances en GPU están enfocados en que la simulación y todas sus operaciones complejas se ejecuten completamente en la GPU. Para esto los datos se almacenan en el dispositivo en todo momento y se hacen transferencias a la CPU solo

cuando sea necesario guardar resultados de la simulación, así se minimizan al máximo las transferencias de datos CPU y GPU. (Crespo, 2020)

La figura 9 muestra un esquema de cómo se implementaron los procesos SPH sobre DualSPHysics sobre CPU, también nos muestra una idea de la transferencia de memoria CPU-GPU y la paralelización de dichos procesos en la GPU.

Figura 9

Distribución de procesos SPH sobre CPU y GPU respectivamente



Nota. Bloques fundamentales de SPH implementados sobre DualSPHysics en CPU y GPU respectivamente. De *DualSPHysics wiki*.

4.3.2 Despliegue de DualSPHysics

Es momento de analizar las ejecuciones del proyecto sobre CPU y GPU, de estas últimas el objetivo es realizar indagar sobre su funcionamiento para comprender las características del despliegue, como se desarrolla en ciertas arquitecturas y realizar estimados de aceleración en comparación a su uso sobre CPU.

4.3.2.1 Clúster GUANE-1. Para las ejecuciones de DualSPHysics se dispondrá principalmente del clúster GUANE-1. Lo que se busca lograr para efectos del proyecto es aprovechar la estructura de nodos en la que están basados los clústeres, donde cada nodo dispone de una cantidad N de GPUs asignadas. Durante este trabajo no se realizarán despliegues de simulaciones de más de un nodo, pero si se aprovecharán las unidades de procesamiento gráfico asignadas a cada uno de estos. Se analizará el comportamiento de la simulación con una serie de implementaciones en los que se medirán tiempos de ejecución, el uso de recursos mediante un perfilamiento de la aplicación y la aceleración del código al ser ejecutada sobre una GPU.

La arquitectura de GUANE facilita el uso de los recursos de las unidades de procesamiento gráfico. Posee la arquitectura general de un clúster el cual tienen nodos interconectados a otros por una misma red y cada uno de estos tiene a su disposición un determinado número de GPUs. Véase tabla 5.

Tabla 5

Especificaciones GUANE

Hardware	Especificaciones
CPU	Intel Xeon E5645 (6 núcleos a 2,40 GHz)
GPU	Tesla M2075
RAM	104 GB
Sistema Operativo	CentOS Linux release 7.9.2009 (Core)

Nota. Especificaciones de un nodo GUANE.

4.3.2.1.1 Despliegue. Los nodos de GUANE ofrecen una arquitectura de GPU clásica, esta familia contiene una capacidad de cálculo reducida, con una baja capacidad de memoria en la tabla 6 podemos ver las características de estas GPU. Lo que se quiere es tener un acercamiento al despliegue del proyecto DualSPHysics sobre estos nodos. Gracias a las especificaciones es posible desplegar el proyecto compilado bajo la versión de CUDA 8.0.

Tabla 6

Especificaciones Tesla M2075

Especificaciones Tesla M2075	
Fabricante	NVIDIA
Modelo	Tesla M2075
Arquitectura	Fermi 2.0
Especificaciones de memoria	
Tamaño de memoria	6144 MB
Tipo de memoria	GDDR5
Reloj de memoria	783 MHz
Ancho de banda de memoria	150.34 GB/s
Especificaciones CUDA	
CUDA	2.0
CUDA cores	448

Nota. De NVIDIA Tesla M2075 specs. Por TechPowerUp, 2022

Como se comentó el objetivo del despliegue sobre GUANE y sobre esta familia de GPUs, es reconocer el despliegue del código sobre clústeres y analizar su comportamiento sobre este hardware, las ejecuciones iniciales tienen como objetivo hacer una medición de los tiempos de ejecución de una simulación, tanto en las unidades de procesamiento CPU, y probar las herramientas que contiene implementadas DualSPHysics sobre simulaciones en GPU, así

confirmar con una medición simple la aceleración que tiene efecto los despliegues en paralelo, pasando de ejecutar solo en CPU a un despliegue sobre una única GPU.

Como se especificó, la compilación del código se realizó sobre la versión 8.0 de CUDA, los pasos para esto son sencillos y se pueden encontrar en la documentación del proyecto. Es necesario modificar el archivo Makefile. Y es necesario identificar la versión de controlador de NVIDIA, como la familia de GPU tiene una capacidad de cómputo 2.0, es necesario usar arquitecturas al compilar que permitan usar estas características.

Tabla 7

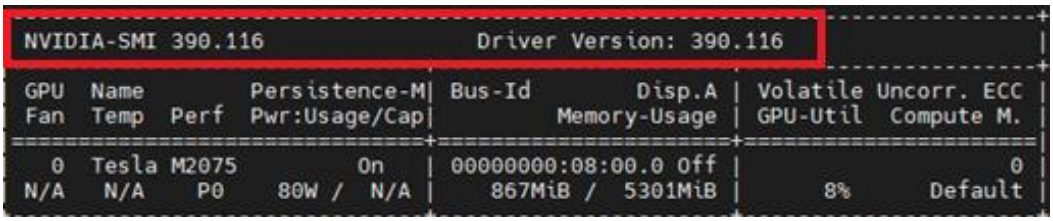
Versiones de controladores compatibles para cada toolkit CUDA.

CUDA Toolkit	Toolkit Driver Version Linux x86_64 Driver Version
CUDA 9.2	>= 396.26
CUDA 9.1	>= 390.46
CUDA 8.0	>= 367.48

Nota. De CUDA Toolkit Documentation. (s.f)

Figura 10

Versión del controlador NVIDIA



Nota. Información obtenida de GUANE mediante el comando nvidia-smi.

De la información contenida en la tabla 7 y la figura 10 concluimos que se puede hacer uso de las versiones de CUDA 9.1 e inferiores. Por lo que es pertinente en este caso compilar el

proyecto sobre la versión de CUDA 8.0. El último paso para el despliegue sobre este clúster es configurar la variable del archivo Makefile que contiene la ruta del toolkit de CUDA que se va a utilizar.

Figura 11

Configuración del archivo Makefile para la compilación de DualSPHysics

```
#============== CUDA selection ===============
CUDAVER=80

#============== CUDA toolkit directory (make appropriate for local CUDA installation) ===============
ifeq ($(CUDAVER),80)
  DIRTOOLKIT=/opt/ohpc/pub/devtools/cuda/8.0
endif
ifeq ($(CUDAVER),75)
  DIRTOOLKIT=/exports/opt/NVIDIA/cuda-7.5
endif
ifeq ($(CUDAVER),91)
  DIRTOOLKIT=/exports/opt/NVIDIA/cuda-9.1
endif
ifeq ($(CUDAVER),92)
  DIRTOOLKIT=/exports/opt/NVIDIA/cuda-9.2
endif

#============== Select GPU architectures ===============
ifeq ($(CUDAVER),80)
  GENCODE:=$(GENCODE) -gencode=arch=compute_20,code=\"sm_20,compute_20\"
  GENCODE:=$(GENCODE) -gencode=arch=compute_30,code=\"sm_30,compute_30\"
```

Nota. Configuración del archivo Makefile.

4.3.2.1.2 Experimentación Lo primero es establecer puntos de partida para algunas métricas y evaluar el desempeño del proyecto ejecutando una simulación sobre CPU y tener una comparación entre tiempos y rendimiento del despliegue sobre una única GPU.

Para efectos prácticos no se realizará un problema CFD desde cero, se aprovecharán los ejemplos contenidos en el código fuente, en este caso se utilizó el ejemplo de rompimiento de olas.

Para obtener resultados de las mediciones se tiene apoyo de las herramientas que ofrece el toolkit de CUDA para perfilado y análisis de aplicaciones. El primer despliegue que se lanzará será una medición de tiempos de ejecución en CPU contra los tiempos de ejecución en paralelo con GPU, contará con un número bajo de partículas que tanto la CPU, como la GPU puedan controlar.

4.3.2.1.3 Resultados de la simulación. Para esta simulación no se quiere sobrepasar los límites de CPU, ni la memoria de la GPU, ya que simular un número elevado de partículas representa una carga muy grande para esta unidad, lo que claramente se traduce en tiempo de espera o errores de memoria respectivamente. En la tabla 8 se sintetizan los datos de la simulación desplegada.

Tabla 8

Características de simulación

Características de la simulación	
Número de partículas	171496
Límites de la simulación	Rango X: 0.001 a 1.599 [m] Rango Y: 0.001 a 0.6725 [m] Rango Z: 0.001 a 0.4515 [m]
Valores de la gravedad	X = [0] Y = [0] Z = [-9.81]

Se harán dos ejecuciones, la primera únicamente en un nodo CPU y un despliegue en paralelo con GPU; se buscan resultados en tiempos de ejecución real de la simulación

Tabla 9

Resultados de tiempo de la simulación GUANE-1.

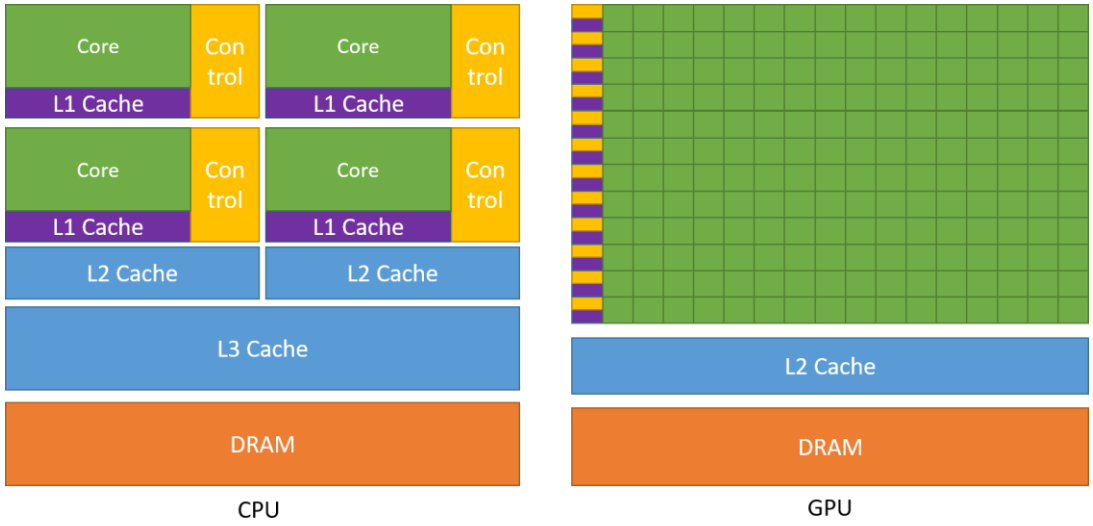
Característica	CPU	GPU
Real	328m53.408s	15m3.0321s
Sistema	2m2.703s	5m13.456s

Realizando un análisis de estos resultados se puede asegurar la diferencia clara que existe entre ambos despliegues, claramente existe una diferencia entre los tiempos entre la ejecución en serie y la ejecución en paralelo. Los resultados afirman que el uso de las unidades de procesamiento gráfico presenta una opción de aceleración de las simulaciones de flujo de fluidos.

Esta desigualdad se debe a la diferencia de diseños de la CPU Y la GPU, mientras que la CPU está diseñada exclusivamente para realizar ejecuciones de secuencias de tareas o subprocesos lo más rápido posible (esta puede procesar sólo algunos de ellos en paralelo). La GPU está diseñada para destacar en la ejecución de miles de ellos en paralelo, es decir, se especializa en cálculos altamente paralelos, lo que conlleva que en su diseño se destinen más transistores al procesamiento de datos en lugar de almacenamientos de caché o de control de flujo. La GPU puede ocultar latencias de acceso a la memoria con computación, en lugar de depender de la memoria en caché de datos y evitar largas latencias de acceso a memoria. Los sistemas están diseñados para combinar GPU y CPU para maximizar el rendimiento general. Véase figura 12. (CUDA C++ Programming Guide, s.f)

Figura 12

Distribución de recursos de chip para CPU frente a GPU

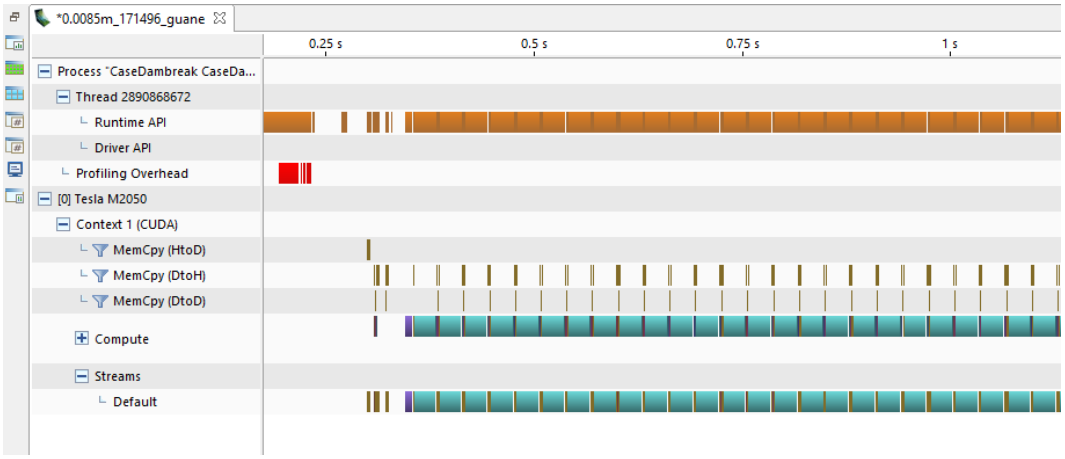


De *C++ Programming Guide*. (s. f.). https://docs.nvidia.com/cuda/cuda-c-programming-guide/

4.3.2.1.4 Perfilación de DualSPHysics. El perfilamiento de un código contribuye altamente al análisis del ¿por qué? del comportamiento del despliegue, es decir, nos va a permitir visualizar de forma más grafica cada detalle del despliegue mediante una línea de tiempo, el lanzamiento de cada kernel de la simulación y como se distribuye el costo computacional en cada uno de ellos. Además de esto aporta información esencial del manejo de memoria en cada paso de tiempo. Debido a toda esta información, la perfilación de un código es una herramienta esencial cuando se quiere estudiar la naturaleza del paralelismo y/o se requiere implementar nuevas características.

Figura 13

Perfilación de DualSPHysics sobre GUANE



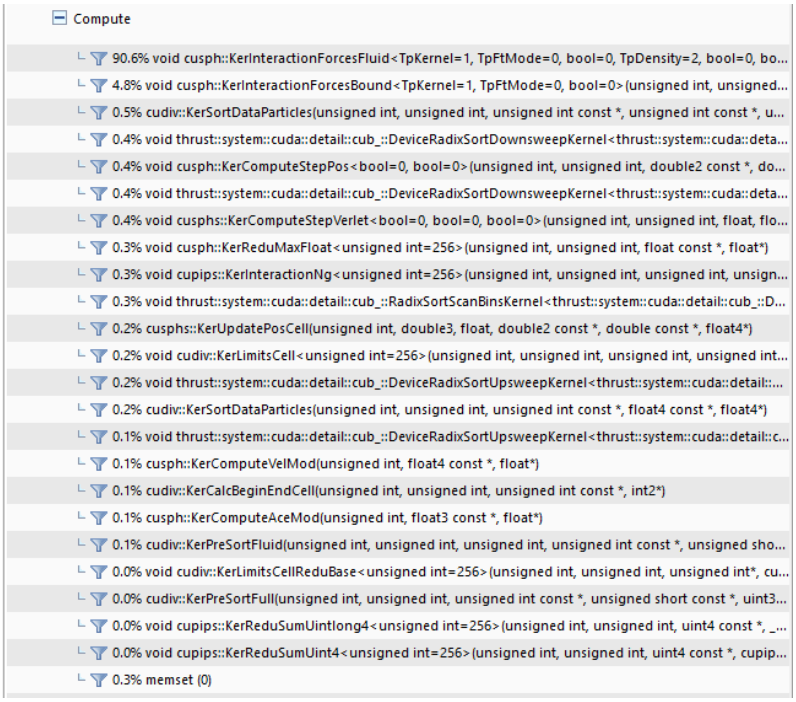
Nota. Perfilación del caso rompimiento de olas sobre GPU serie Tesla M2050 – GUANE

En la figura 13 vemos un resumen de la perfilación de la simulación lanzada, se utilizó la herramienta nvprof del toolkit de CUDA, que proporciona una versión grafica todas estas características mencionadas anteriormente, para el caso del despliegue de DualSPHysics, podemos ver las asignaciones de memorias hechas en cada paso de tiempo, tanto las copias de memoria host a dispositivo como viceversa. Obtenemos información detallada de cada kernel lanzado en todo el despliegue y como se comportó a lo largo del tiempo que duró en ejecución.

Analizando la gráfica, también obtuvimos información sobre los streams lanzados, se evidencia que la implementación de DualSPHysics contempla un sólo stream, conocido como stream default.

Figura 14

Kernels CUDA lanzados durante el despliegue de la simulación sobre una GPU.



En la figura 14 contemplamos un panorama general del comportamiento del despliegue a lo largo de la línea de tiempo, podemos ver los esfuerzos en la paralelización en el código fuente de DualSPHysics, es de destacar que una de las partes fundamentales del método SPH es el cálculo de fuerzas entre partículas vecinas en cada paso del tiempo y también es una de las partes más demandantes computacionalmente. Como podemos evidenciar con los resultados abarca una gran cantidad de los recursos de la GPU realizar esta tarea

4.3.2.2 Clúster ABACUS. Teniendo en cuenta los resultados del despliegue de la simulación sobre la arquitectura GPU del nodo anterior. Es importante para nuestro objetivo analizar las posibles diferencias en el comportamiento del código sobre otras arquitecturas. En este caso se optó por una GPU que ofrece una capacidad de cómputo más completa y con un mejor rendimiento en cuanto a memoria y capacidad, para esto se utilizó el hardware contenido en el

nodo ABACUS. Este clúster pertenece al centro de investigación y de estudios avanzados de Instituto politécnico nacional CINVESTAV ubicado en Ciudad de México.

Tabla 10

Especificaciones nodo ABACUS

Hardware	Especificaciones
CPU	Intel Xeon E5-2697v3 (14 núcleos a 2.6 GHz)
GPU	NVIDIA Tesla K40m
RAM	128 GB (2133 MHz)
Sistema Operativo	CentOS Linux release 7.9.2009 (Core)

4.3.2.2.1 Despliegue. Las especificaciones de estas unidades de procesamiento ofrecen una mejora en cuanto a cantidad de memoria y cantidad de multiprocesadores, esto ofrece la posibilidad de realizar simulaciones con más alto número de partículas para visualizar el comportamiento del proyecto.

Tabla 11

Especificaciones Tesla K40m

Especificaciones Tesla K40m	
Fabricante	NVIDIA
Modelo	Tesla K40m
Arquitectura	Kepler
Especificaciones de memoria	
Tamaño de memoria	12 GB
Tipo de memoria	GDDR5
Reloj de memoria	745 MHz
Ancho de banda de memoria	288.4 GB/s
Especificaciones CUDA	
CUDA	3.5
CUDA cores	2880

Nota. Datos extraídos de NVIDIA Tesla K40m specs. Por TechPowerUp, 2022

Es importante comentar que, por las especificaciones de la arquitectura, la instalación del proyecto se realizó sobre la versión 9.0 de CUDA, la compilación del código fue similar a la instalación sobre GUANE solo con el cambio de la versión de CUDA a usar.

4.3.2.2.2 Experimentación. Los experimentos sobre el nodo ABACUS se centraron en tres ideas principales, los primeros despliegues serán ejecuciones iguales a las planteadas en GUANE con los que se obtendrán resultados de tiempos de ejecución y se hará una perfilación de las ejecuciones de DualSPHysics sobre este nodo para notar posibles diferencias en el despliegue. Y una tercera fase donde se elevarán los números de partículas demandando más recursos a la GPU y aumentando la complejidad de procesamiento, se perfilaron estas ejecuciones para observar sus características.

4.3.2.2.3 Resultados de la simulación. Primeramente, se desplegará la simulación contenida en la tabla 8 sobre esta nueva arquitectura para visualizar posibles diferencias entre

nodos, se obtendrán también resultados de tiempos de ejecución de un despliegue en serie y paralelo respectivamente.

Tabla 12

Resultados de tiempos de ejecución ABACUS.

Característica	CPU	GPU
Real	361m29s	5m45s
Sistema	1m5.703s	1m41.21s

La tabla 12 sintetiza los resultados obtenidos en estos despliegues, vemos un comportamiento similar a la anterior ejecución realizada sobre la arquitectura de GUANE, donde la aceleración de los resultados del problema es muy evidente, una carga más grande sobre la CPU retrasaría la obtención de resultados.

Tabla 13

Contraste de tiempos GUANE-1 y ABACUS.

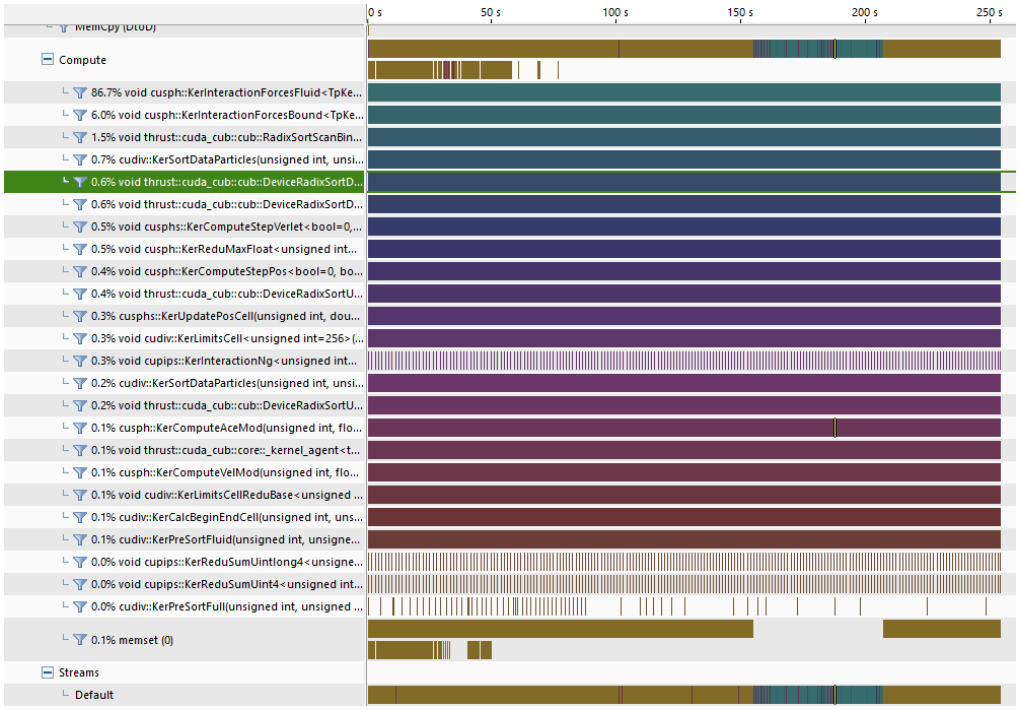
	GUANE-1		ABACUS	
Característica	**CPU**	**GPU**	**CPU**	**GPU**
Tiempo Real (seg)	19733.408	903.0321	21689	345
Tiempo de sistema (seg)	122.703	313.456	65.703	101.21

En la tabla 13 se aprecia una comparativa de los tiempos de despliegue del ejemplo de rompimiento de olas simulado en distintas arquitecturas, primeramente, la arquitectura contenida en GUANE-1 con familias de GPU NVDIA Tesla M2075, y ABACUS con arquitecturas de GPU NVIDIA Tesla k40m.

4.3.2.2.4 Perfilación de DualSPHysics. De igual manera se planteó hacer una perfilación de la ejecución de la simulación, De manera general el despliegue se comporta de manera muy similar comparado a el nodo anterior, la principal característica en la que difieren es en los tiempos de ejecución de los kernels CUDA lanzados durante el despliegue, las asignaciones de memoria al device se hacen en un periodo de tiempo más corto, otra particularidad es que los costos de los kernels cambiaron, en este caso la interacción entre partículas bajo de un 90% a un 86.7& de uso, y el kernel de interacción entre partículas límites de la simulación subió a un 6%.

Figura 15

Perfilación del despliegue de DualPSHysics sobre un nodo de ABACUS



Nota. Perfilación del caso rompimiento de olas sobre GPU serie Tesla K40m – ABACUS

4.3.3 Speedup

El speedup no es más que la aceleración que obtiene un programa al ser ejecutado en su versión paralela, se obtiene del resultado del cociente entre tiempo de ejecución serial y el tiempo de ejecución de la misma tarea en paralelo. Y su objetivo es señalar el tiempo que se ha reducido de un despliegue a otro. Para hacer el cálculo de esta métrica para DualSPHysics se realizó usando un nodo del clúster GUANE, donde los tiempos seriales de la aplicación se computaron usando una sola CPU y los tiempos en paralelo el despliegue sobre una única GPU, se aumentó gradualmente el número de partículas que se simuló. Fue necesario restringir el número de partículas para esta medición con el objetivo de no exceder los límites de memoria de la unidad de procesamiento gráfico, y que los tiempos fueran manejables para las ejecuciones en la unidad de procesamiento central.

Para realizar la medición de speedup, se usó un enfoque de aumento de carga computacional para cada despliegue realizado, de esta manera es posible obtener los datos necesarios de tiempos de simulación en serie y en paralelo para realizar los respectivos cálculos de aceleración. Adicionalmente se tomó la medida de mantener los volúmenes de la simulación de forma estándar, así se asegura un entorno controlado en el que se aumentará la carga progresivamente en función de la cantidad de partículas simuladas. Para esto es necesario disminuir el espacio inicial entre partículas vecinas, así es posible simular mayor cantidad de estas manteniendo las dimensiones del ejercicio.

4.3.3.1 Resultados. Para conseguir una medición del speedup del proyecto se utilizó el enfoque de aumentar la carga computacional simulando distintas cantidades de partículas, para esto se realizaron mediciones para distintas distancias iniciales entre ellas, lo que permite aumentar

o disminuir la carga, para cada despliegue se tomaron marcas de tiempo de cada simulación y posteriormente se efectuó el cálculo para obtener la aceleración. Véase tabla 13 y figura 16.

Tabla 14

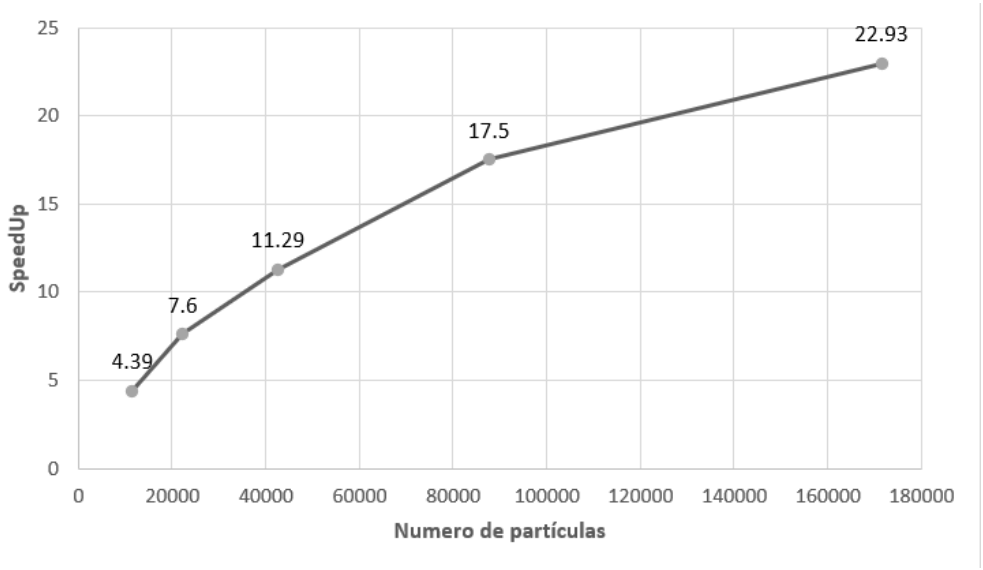
Resultados speedup DualSPHYsics GUANE.

Distancia entre partículas (m)	Número de partículas	Tiempo de ejecución en serie (seg)	Tiempo de ejecución en paralelo (seg)	SpeedUp
0.0085	171496	21677.4	945.37	22.93
0.0108	87670	6789.6	388.25	17.49
0.014	42608	2776.2	246.12	11.28
0.018	22286	1368	180.52	7.58
0.024	11549	632.4	144.4	4.38

Con estos resultados se sintetizó la gráfica para así poder analizar estos de una mejor forma.

Figura 16

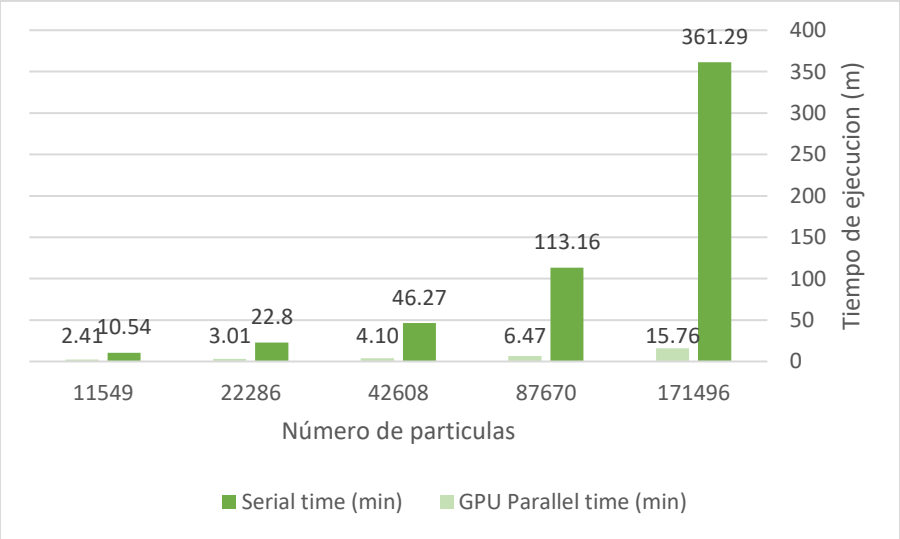
Gráfico SpeedUp DualSPHysics sobre GUANE



La grafica pertenece a un tipo de escalabilidad débil donde se aumenta la carga de trabajo con un mayor número de partículas simuladas. La gráfica de speedup muestra un comportamiento típico y representa la aceleración de la aplicación al ejecutarse en paralelo haciendo uso de GPU.

Figura 17

Contraste en tiempos de ejecución paralelo y serial DualSPHysics.



La figura 17 es una gráfica extraída de los datos obtenidos con las mediciones anteriores aquí se refleja una comparación en términos de tiempos de ejecución entre el despliegue de la aplicación sobre CPU y sobre la GPU. En esta se puede observar claramente como el aumento en la carga computacional recae directamente en la capacidad de cómputo de las tradicionales unidades de procesamiento, a una carga muy grande los tiempos de resultados serán demasiado grandes, y es posible que se exceda la capacidad de memoria de estas.

4.3.4 Arquitecturas Multi GPU

La implementación del método SPH en múltiples GPU pretende principalmente paralelizar el cálculo de las interacciones entre partículas en una simulación de fluidos computacional, para esto se han propuesto distintas soluciones para realizar este cálculo. Una de ellas consiste en la partición del dominio de la simulación en subdominios, donde cada uno es asignado a una GPU distinta. Una vez seccionado el dominio de la simulación cada GPU se encarga del cálculo de las interacciones entre partículas de su subdominio asignado y por último los resultados se combinan

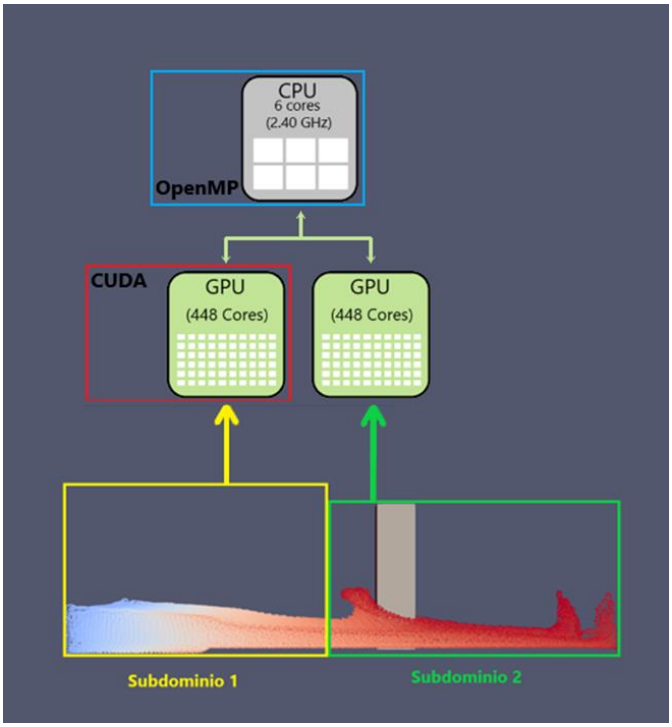
y sincroniza en todas las GPU compartiendo información entre dispositivos. El siguiente método se conoce como descomposición de datos. Aquí las partículas se mueven libremente por los subdominios. Este método requiere más comunicación entre GPU para garantizar que las interacciones entre partículas se calculen de forma correcta. Además, también es importante optimizar el uso de la memoria en los dispositivos y minimizar la transferencia de datos entre host y dispositivo en cualquiera de las dos implementaciones.

Para este trabajo el despliegue se propuso implementar sobre el método de división de dominios, para esto no se tendrán en cuenta factores como el balanceo de carga entre los subdominios, ni el intercambio de memoria entre dispositivos, se plantean despliegues limpios considerando las implementaciones actuales del proyecto DualSPHysics. Una vez se obtengan los resultados se realizará un análisis donde finalmente se propondrán lineamientos y consideraciones al llevar a cabo la implementación del método en estas arquitecturas.

4.3.4.1 Arquitectura. La arquitectura propuesta para el despliegue se compone únicamente de un nodo que tiene a disposición dos unidades de procesamiento gráfico GPU, figura 18, esta es una arquitectura básica en la que se busca omitir el intercambio de memoria de host a host al no ser necesario.

Figura 18

Arquitectura multi-GPU y subdivisión de dominios



Nota. Adaptado de Mokos (2017)

A su vez la figura 18 ilustra un ejemplo de la partición del dominio de la simulación en dos subdominios, siguiendo la arquitectura propuesta, cada subdominio pretende asignarse a una GPU dedicada exclusivamente a los cálculos de las interacciones entre partículas del subdominio asignado.

Como se mencionó anteriormente no se definirá un balanceo de cargas al momento de realizar la división del dominio, por lo que la carga computacional asignada a cada GPU puede no ser equitativa, se analizaran los resultados de estos despliegues y se determinará si es necesario la aplicación de esta técnica.

4.3.4.2 Despliegue. Las implementaciones se desplegaron en el clúster Guane sobre un nodo que ofrece prestaciones de hardware superiores a las utilizadas recientemente en el mismo

servidor y con esto una mayor capacidad de cálculo y memoria. Como configuración de DualSPHysics se mantuvo la versión de CUDA 8.0 usada sobre este clúster anteriormente.

Tabla 15

Hardware multi-GPU

Hardware	Especificaciones
CPU	Intel Xeon E5645 (6 núcleos a 2,40 GHz)
GPU	GeForce GTX TITAN X x2
RAM	104 GB
Sistema Operativo	CentOS Linux release 7.9.2009 (Core)

Tabla 16

Especificaciones y arquitectura de la GPU Geforce GTX TITAN X

Especificaciones Geforce GTX TITAN X	
Fabricante	NVIDIA
Modelo	Geforce GTX TITAN X
Arquitectura	Maxwell 2.0
Especificaciones de memoria	
Tamaño de memoria	12213 MB
Tipo de memoria	GDDR5
Reloj de memoria	1753 MHz
Ancho de banda de memoria	336.6 GB/s
Especificaciones CUDA	
CUDA	5.2
CUDA cores	3072 (128 CUDA cores/MP)

En la tabla 15 se sintetizan las prestaciones que ofrece la GPU en cuanto a arquitectura, algunas de las características a destacar de esta unidad es su capacidad de memoria, lo cual va a permitir realizar simulaciones con mayor número de partículas, de la misma manera la cantidad de núcleos CUDA es un factor fundamental a la hora de realizar dichas al hacer el uso de esta tecnología.

4.3.4.3 Resultados. El problema CFD que es utilizado es el ejemplo contenido en el código DualSPHysics conocido como rompimiento de olas, esto para mantener un estándar en cada fase del trabajo. Al igual que en las implementaciones anteriores se realizaron diferentes despliegues para obtener las mediciones pertinentes para llevar a cabo un análisis y posteriormente realizar el planteamiento de lineamientos para la implementación del método SPH sobre estas arquitecturas multi-GPU.

4.3.4.3.1 SpeedUp. En la tabla 16 se esquematizan los despliegues realizados con su respectivo cálculo de speedup para cada medición de tiempo realizada. En los resultados es posible apreciar la aceleración obtenida al paralelizar el código, como es de notar dicha diferencia crece en función de la carga computacional aplicada, es decir que, a mayor carga, se obtiene una mayor aceleración en cada despliegue en paralelo. Esto, es notable también al analizar los resultados del cálculo de speedup, este incrementa a la vez que sube la cantidad de partículas simuladas, siendo el mayor registrado en el despliegue con mayor número de estas.

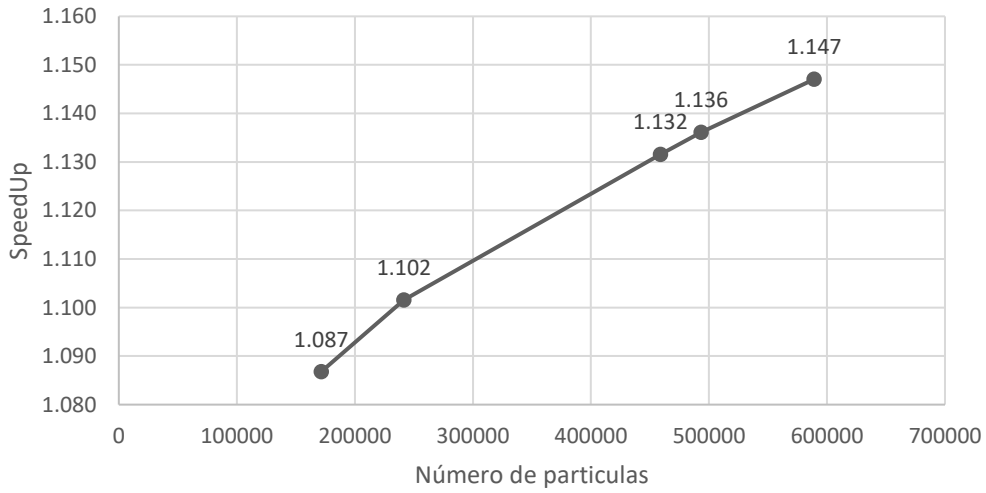
Tabla 17

Resultados SpeedUp multi-GPU.

Distancia entre partículas (m)	Número de partículas	Tiempo de ejecución en serie – Una sola GPU (seg)	Tiempo de ejecución en paralelo – multi-GPU (seg)	Speedup	Aceleración %
0.0055	589245	982.302	856.37	1.147	14.71
0.0058	493316	776.126	683.155	1.136	13.61
0.006	458920	698.25	617.056	1.132	13.16
0.0075	241672	348.105	316.018	1.102	10.15
0.0085	171496	222.645	204.865	1.087	8.69

Figura 19

Gráfico SpeedUp DualSPHysics multi-GPU.

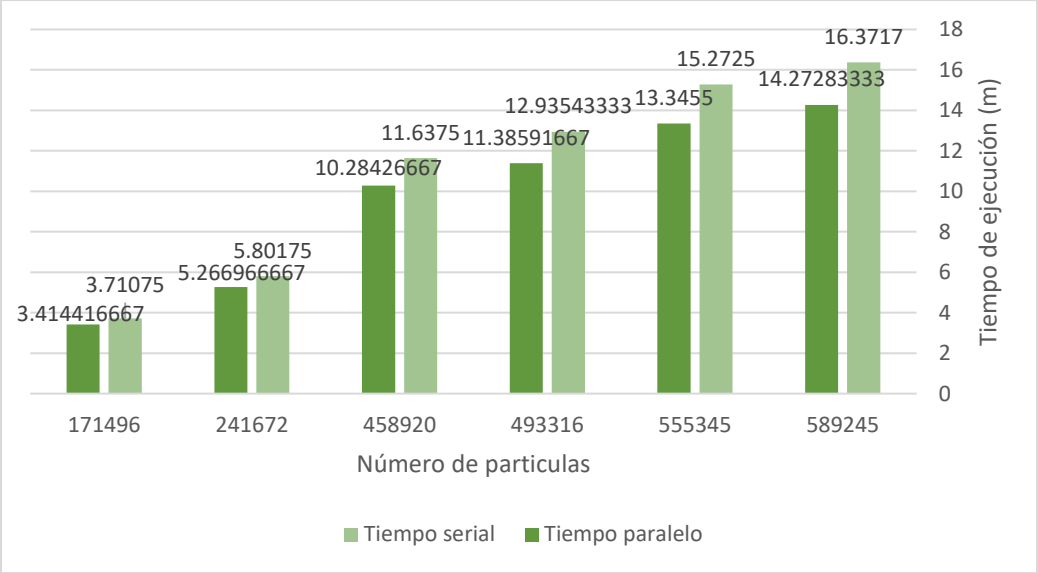


En una descomposición de la gráfica presentada en la figura 19 es notable que algunas de las expectativas puestas en un despliegue multi-GPU no se cumplen en su totalidad. Las posibles causas de estos resultados pueden ser diversas y es necesario realizar un diagnóstico del código e identificar factores que estén afectan negativamente la aceleración de su implementación. Es

importante diagnosticar debidamente antes de realizar un veredicto sobre las capacidades de esta tecnología.

Figura 20

Contraste en tiempos de ejecución paralelo y serial DualSPHysics.



Aunque los resultados no son exactamente los esperados y no demuestran el poder de procesamiento de una técnica multi-GPU, si podemos afirmar resultados positivos para esta investigación. Y es de tener en cuenta que como se ha hablado anteriormente despliegues no se realizó ninguna modificación al código fuente del proyecto, el objetivo es realizar un análisis desde los avances contenidos en DualSPHysics, y a partir de este momento entrar a discutir alternativas para mejorar dichas estadísticas en trabajos futuros.

Para finalizar, en este capítulo se proyectaron los resultados de las investigaciones acerca de la explotación del paralelismo de procesos SPH sobre arquitecturas GPU. Para esto se analizaron las estrategias y herramientas usada por el algoritmo DualSPHysics. Se pudo

profundizar en cada una de sus características, como la visualización de sus propiedades en los distintos despliegues realizados sobre cada arquitectura GPU usada en este trabajo, se realizó una perfilación del codigo en los que se evidencian los kernels CUDA implementados y usados por la simulación y una visualización de las transferencias de memoria entre host y dispositivo en una línea temporal. Por último, se obtuvieron resultados de la aceleración alcanzada tras el despliegue sobre una única GPU comparado con un despliegue en CPU.

También, durante este capítulo, se presentaron los resultados del estudio de técnicas de paralelismo SPH para múltiples GPU, se estudiaron las posibles alternativas para estas implementaciones, y se expuso los resultados alcanzados tras el despliegue de una de ellas, de los cuales se obtuvo una aceleración de entre el 8.69% y el 14.71%. Sin embargo, se identificaron algunas de las limitaciones y oportunidades de mejora de esta, como lo sería la falta de un balanceo de cargas oportuno entre las unidades de procesamiento gráfico.

Con base en los resultados resaltados en este capítulo, a continuación, se presentarán una serie de conclusiones y recomendaciones acerca a la problemática de las dinámicas de fluidos computacionales, los aportes de la solución presentada y se plantearán caminos a seguir en trabajos futuros.

5. Conclusiones

En este trabajo se profundizó e investigó acerca de la problemática de la dinámica de fluidos computacional, más específicamente en cómo es abordada desde el ángulo de la ingeniería computacional. Haciendo énfasis en el método de hidrodinámica de partículas suavizadas y como su funcionamiento y su uso atacaban este problema desde la perspectiva del paralelismo.

Para esto, en una primera parte se investigo acerca de las problemáticas de las dinámicas de fluidos y como el uso del método SPH atacaba ciertas de estas con características como la

flexibilidad al no requerir una malla fija, haciéndolo más eficaz al simular geometrías complejas. La robustez siendo menos sensible a los errores en las condiciones iniciales de un problema, mucho más versátil trabajando en varios campos como fluidos y sólidos y que permite ser escalado fácilmente a arquitecturas altamente paralelas. Cumpliendo de esta manera el objetivo de comprender su uso en los problemas de CFD.

Para el segundo objetivo propuesto se investigó acerca de proyectos que mantuvieran un acercamiento no solo a las simulaciones CFD basadas en este método lagrangiano, sino que también, contuvieran enfoques en la paralelización de procesos y la explotación de la concurrencia. Esta parte del estudio se enfocó en el análisis de distintos algoritmos que atacaran la problemática desde el ángulo propuesto con diferentes herramientas y tecnologías. Como resultado y en beneficio del desarrollo de este trabajo y después de experimentar e identificar dentro de cada algoritmo estos procesos, se adoptó una de las mejores implementaciones que como se evidencia obtuvo los mejores resultados.

Se realizo entonces el estudio de los despliegues SPH sobre GPU, los cuales se caracterizan por tener tiempos de ejecución muy cortos. Esto es posible notarlo con un problema básico como lo es el rompimiento de olas. Mediante esta investigación fue posible comprobar que las simulaciones sobre las convencionales CPU tienden a ser mucho menos eficaces que los despliegues realizados sobre GPUs. Una vez exploradas estas diferencias, se propuso profundizar en algunas de las características de las tecnologías usadas. Con un ejercicio como la perfilación de la ejecución se pudo visualizar como se segmento para este caso, el código DualSPHysics para cada proceso que se requiere paralelizar del método SPH, siendo los más afectados los procesos que contienen más carga computacional, que como se ha mencionado a lo largo de esta investigación, son aquellos procesos que conllevan el cálculo de las fuerzas entre partículas

vecinas en cada paso de tiempo de la simulación. Así también, es posible afirmar que los cambios en las características del hardware afectan positiva o negativamente el despliegue del aplicativo, mostrándose como esencial la cantidad de núcleos y de multiprocesadores contenidos en cada arquitectura usada.

Se realizo entonces una investigación acerca de las alternativas y caminos para esta implementación. De los resultados y despliegues realizados se obtuvieron resultados positivos para el estudio, los cuales fueron presentados durante el capítulo anterior y de los que se resalta una mejoría en la aceleración del codigo entre el 8.69% al 14.71% con respecto a las implementaciones sobre una única GPU, esto a medida que se aumentaba la carga computacional del problema CFD simulado, dicha aceleración es posible verla en las métricas de speedup y visualizarla en la respectiva grafica. Tabla 16 y figura 19. Estos resultados aportan avances a la solución de las limitaciones en la simulación de problemas CFD y al estudio de la hidrodinámica de partículas suavizadas SPH. No obstante, aunque favorables para la investigación los resultados se quedan un poco lejos de las expectativas creadas con base en la alta capacidad de cómputo demostrado por estas unidades de procesamiento y es necesario en implementaciones futuras realizar una revisión de estos factores que pueden estar afectando la aceleración.

6. Recomendaciones

Hay diversos factores que es necesario evaluar al implementar una simulación CFD con SPH sobre arquitecturas con múltiples GPU, en primer lugar, se debe conocer la arquitectura sobre la que se va a desplegar la simulación. Es importante conocer las capacidades y limitaciones del hardware sobre el que se trabajará, a su vez es valioso elegir un método de particionamiento adecuado que se ajuste a la arquitectura específica del dispositivo con tal que se minimice la sobrecarga en la comunicación entre las unidades de procesamiento y así garantizar un buen

desempeño. La gestión de memoria también es fundamental en estas implementaciones es más que necesario disminuir la transferencia de datos entre la memoria del host y la memoria del dispositivo y el intercambio de información entre estos. Aquí juegan factores como el equilibrio de cargas, la sincronización de las simulaciones, la paralelización optimizada y la escalabilidad de la simulación, con esto último se quiere enfatizar a que la paralelización debe de rendir de manera eficiente cuando se aumente el número de GPUs, esto se puede conseguir asegurando un particionamiento adecuado y optimizando la comunicación y la gestión de memoria.

Es necesario discutir entonces sobre las características fundamentales al momento de particionar el dominio de una simulación y en la repartición de estos subdominios a cada GPU asignada. Para iniciar es necesario hablar del tamaño del dominio y la cantidad de partículas. Conocer la cantidad de partículas que va a contener la simulación es un factor clave a la hora de subdividir el dominio en las diferentes GPUs, un gran número de partículas requiere una mayor cantidad de memoria y de recursos, es importante equilibrar dichos subdominios para garantizar un uso eficiente de cada unidad de procesamiento gráfico. Uno de los factores que afecta directamente a la cantidad de partículas es el tamaño del dominio, un dominio muy grande, conllevaría a tener que simular una mayor cantidad de partículas, lo que afecta también a la subdivisión de este. La capacidad del flujo de fluidos es otra de las características que puede afectar a la cantidad de partículas de una simulación, ya que un flujo complejo requerirá la simulación de un mayor número de partículas.

También es importante tener en cuenta otros factores como la cantidad de memoria disponible en cada dispositivo, ya que una mayor cantidad de memoria permite simular una mayor cantidad de partículas para cada subdominio, aunque es importante tener en cuenta y como se mencionaba anteriormente no dejar de lado el intercambio de información entre dispositivos ya que se debe

resaltar que un intercambio de información alto a lo largo del despliegue de la simulación impactara sobre los tiempos de ejecución, una disminución de esta se puede lograr minimizando el paso de partículas entre subdominios y por lo tanto la información que debe transferirse entre estas y a su vez optimizando los algoritmos de comunicación.

Por último, el método de partición debe elegirse de forma que proporcione el mayor aumento de velocidad, como ya se ha discutido esto depende mucho del tipo de simulación a realizar y la capacidad de hardware, ya que por ejemplo un método diseñado para una cantidad pequeña de GPUs puede no ajustarse al aumentar el número de estas. Como conclusión, una implementación eficiente sobre arquitecturas con múltiples GPU requiere de una buena interpretación del problema que se quiere resolver y de los recursos disponibles para ello. Teniendo en cuenta todos estos factores a la hora de la implementación se garantiza que los recursos se usen de forma eficiente y se obtengan resultados positivos en la aceleración del código.

6.2 Trabajo futuro

A partir de este punto es posible disponer de distintas alternativas de trabajo futuro como la exploración del método de partición de datos para la ejecución multi-GPU, la implementación de técnicas de balanceo de cargas dinámicas para la partición del dominio de la simulación, o el uso de tecnologías aplicadas como MPI para evaluar la ejecución de implementaciones multi-GPU entre nodos. También pueden surgir propuestas como usar directivas como OpenACC para ampliar el rango de dispositivos como a AMD o Intel.

Referencias Bibliográficas

ABACUS: Laboratorio de Matemática Aplicada y Cómputo de Alto Rendimiento del Departamento de Matemáticas. (s. f.). ABACUS: Laboratorio de Matemática Aplicada y Cómputo de Alto Rendimiento del Departamento de Matemáticas. https://www.abacus.cinvestav.mx/

About SPlisHSPlasH - SPlisHSPlasH 2.12.1 documentation. (s. f.). https://splishsplash.readthedocs.io/en/latest/about.html

Backus, J. (1957). The FORTRAN Automatic Coding System. IBM Journal of Research and Development, 1(2), 100-109.

Clúster Guane - Supercomputación y Cálculo Científico UIS. (s. f.). http://wiki.sc3.uis.edu.co/index.php/Cluster_Guane

Crespo, A. J. C., Domínguez, J. M., Barreiro, A., Gómez-Gesteira, M., & Rogers, B. D. (2011). GPUs, a new tool of acceleration in CFD: Efficiency and reliability on smoothed particle hydrodynamics methods. PLoS ONE.

DualSPHysics Wiki. (s. f.). GitHub. https://github.com/DualSPHysics/DualSPHysics/wiki

Crespo, A. J. C., Domínguez, J. M., Rogers, B. D., Gómez-Gesteira, M., Longshaw, S., Canelas, R., Vacondio, R., Barreiro, A., & García-Feal, O. (2015). DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH). Computer Physics Communications.

CUDA C++ Programming Guide. (s. f.). https://docs.nvidia.com/cuda/cuda-c-programming-guide/

CUDA Toolkit Documentation. (s. f.). (C) Copyright 2005. https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html

CUDA Zone. (2021, 26 julio). NVIDIA Developer. https://developer.nvidia.com/cuda-zone

Domínguez, J. M., Crespo, A. J. C., & Gómez-gesteira, A. B. M. (s.f.). Implementación Multi-GPU del método Smoothed Particle Hydrodynamics.

DualSPHysics. (2022). DualSPHysics – DualSPHysics: A combined CUDA and OpenMP implementation of the Smoothed Particle Hydrodynamics method based on the advanced SPHysics code. https://dual.sphysics.org/

Foster, I. (1995). Designing and Building Parallel Programs. Addison-Wesley Professional.

Ihme, M., & Colonius, T. (2015). PySPH: A flexible framework for smoothed particle hydrodynamics simulations. Journal of Open Research Software, 3(1), 1-16.

Jespersen, D. C. (2010). Acceleration of a CFD code with a GPU. Scientific Programming.

Liu, M. B. y Liu, G. R. (2010) Smoothed particle hydrodynamics (SPH): An overview and recent developments. Archives of Computational Methods in Engineering

M. H. Kalos, "Monte Carlo methods in the physical sciences," 2007 Winter Simulation Conference, Washington, DC, USA, 2007, pp. 266-271, doi: 10.1109/WSC.2007.4419611.

MobaXterm (2008). MobaXterm. MobaXterm. https://mobaxterm.mobatek.net/

Mokos, A., & Rogers, B. D. (2017). HPC for SPH methods: Multicore, GPU and multiGPU. November, 13–15.

Moler, C. (1983). Numerical Computing with MATLAB. Society for Industrial and Applied Mathematics, Philadelphia, PA.

NVIDIA Nsight Systems. (2022, 13 diciembre). NVIDIA Developer. https://developer.nvidia.com/nsight-systems

NVIDIA Tesla M2075 Specs. (2010, 4 octubre). TechPowerUp. Recuperado 3 de octubre de 2022, de https://www.techpowerup.com/gpu-specs/tesla-m2075.c2025

NVIDIA Tesla K40m Specs. (2010, 25 octubre). TechPowerUp. Recuperado 24 de octubre de 2022, https://www.techpowerup.com/gpu-specs/tesla-k40m.c2529

NVIDIA Visual Profiler. (2021, 14 abril). NVIDIA Developer. https://developer.nvidia.com/nvidia-visual-profiler

Ortiz, R., Charles, J. L., & Sobry, J. F. (2004). Structural Loading of a Complete Aircraft Under Realistic Crash Conditions: Generation of a Load Database for Passenger Safety and Innovative Design. 24th International Congress of the Aeronautical Sciences.

Overview - PySPH 1.0b2.dev0 documentation. (s. f.). https://pysph.readthedocs.io/en/latest/overview.html

Pérez, C. R., Nebreda, J. M. C., & Ortega, P. L. S. (2016). Introducción a la programación en CUDA. 74. https://riubu.ubu.es/bitstream/handle/10259/3933/Programacion_en_CUDA.pdf?sequence=1&isAllowed=y

Ramachandran, P., Bhosale, A., Puri, K., Negi, P., Muta, A., Dinesh, A., Menon, D., Govind, R., Sanka, S., Sebastian, A. S., Sen, A., Kaushik, R., Kumar, A., Kurapati, V., Patil, M., Tavker, D., Pandey, P., Kaushik, C., Dutt, A., & Agarwal, A. (2021). PySPH: A Python-based Framework for Smoothed Particle Hydrodynamics. ACM Transactions on Mathematical Software, 47(4). https://doi.org/10.1145/3460773

Ramos, J. J., & Pérez-Sánchez, C. (2018). DualSPHysics: An open-source code for simulating fluid dynamics using the Smoothed Particle Hydrodynamics method. Journal of Open Research Software, 6(1), 1-18.

Ritchie, D. M. (1972). The C Programming Language (1ª ed.). Prentice-Hall.

Standard. (2011). ISO/IEC 9899:2011 Programming Languages - C. Geneva, Switzerland: International Organization for Standardization.

Schäfer, C., & Düvel, S. (2015). SPLISH-SPLASH: A versatile open-source SPH library. Journal of Open Research Software, 3(1), 1-20.

Standard. (2020). ISO/IEC 14882:2020 Programming Languages - C++. Geneva, Switzerland: International Organization for Standardization.

Stroustrup, Bjarne (1997). «1». The C++ Programming Language (Third edición). ISBN 0201889544. OCLC 59193992.

Shadloo, M. S. Oger, G. y Le Touzé, D. (2016) Smoothed particle hydrodynamics method for fluid flows, towards industrial applications: Motivations, Current state, And challenges. Computers and Fluids

Tejaxún Xicón, E. R. (2019, 1 agosto). Computación paralela a través de CUDA | Decimocuarta Edición - ECYS. Computación paralela a través de CUDA. https://revistaecys.github.io/14Edicion/05-etejaxun.html

VisualStudio (2021, 3 noviembre). Documentation for Visual Studio Code. https://code.visualstudio.com/docs

Valdez-Balderas, D., Domínguez, J. M., Rogers, B. D., & Crespo, A. J. C. (2013). Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters. Journal of Parallel and Distributed Computing, 73(11), 1483–1493. https://doi.org/10.1016/j.jpdc.2012.07.010

Van Rossum, G. (1989). Python (Version 0.9.0). Retrieved from http://www.python.org.

Woo O., H. (2012). Computational Fluid Dynamics, Applied Computational Fluid Dynamics. In Mechanical Engineering Series (Fifth Edit). Elsevier.