Implementación Multi-GPU del método Smoothed Particle Hydrodynamics

J.M. Domínguez, A.J.C. Crespo, A. Barreiro y M. Gómez-Gesteira[1]

Resumen—**En este trabajo se presenta una novedosa implementación Multi-GPU a partir del código Single-GPU llamado DualSPHysics que es un método numérico para la resolución de problemas de dinámica computacional que ha demostrado ser eficiente, robusto y preciso. Esta implementación permite combinar la potencia de múltiples GPUs mediante el uso de MPI. Se abordan los problemas típicos como el balanceo dinámico de carga entre distintos dispositivos y el solapamiento entre comunicaciones y cálculo. La eficiencia de la nueva versión Multi-GPU de DualSPHysics es analizada para diferentes tamaños de simulación y diferentes números de GPUs. Finalmente, una simulación con más de 220 millones de partículas es utilizada para mostrar las capacidades de la nueva implementación.**

Palabras clave—**HPC, GPU, Multi-GPU, SPH, Smoothed Particle Hydrodynamics, Dinámica de fluidos.**

I. INTRODUCCIÓN

El modelado numérico se usa como una herramienta muy útil en el campo de la ingeniería y de la ciencia para resolver problemas complejos. La ventaja principal es la capacidad de simular cualquier escenario por muy complejo que éste sea sin la necesidad de construir costosos modelos físicos a escala. Además, la simulación numérica puede proporcionar datos físicos que pueden ser difícilmente, o incluso imposibles de medir en un modelo real. Sin embargo, también es importante que los tiempos de ejecución sean razonables para poder usar estos modelos en problemas de ingeniería de la vida real. En este trabajo se desarrollan nuevas implementaciones de paralelización para los métodos de dinámica de fluidos computacional llamados Smoothed Particle Hydrodynamics (SPH).

SPH es un método lagrangiano de partículas. Fue desarrollado inicialmente para astrofísica [1] en los años 70, aunque posteriormente se ha aplicado a diversas ramas de la ingeniería. SPH es particularmente útil en el estudio de la hidrodinámica de problemas de superficie libre, como pueden ser flujos violentos y la interacción entre olas y estructuras. Sin embargo, uno de los principales inconvenientes del método SPH es su elevado coste computacional al abordar problemas reales de ingeniería donde se requiere un elevado número de partículas. Por ello, es necesario desarrollar implementaciones paralelas del método capaces de combinar los recursos de múltiples máquinas que permitan simular millones de partículas en tiempos razonables. El uso de unidades de procesamiento gráfico (GPU) se ha convertido en una buena opción para acelerar SPH con un coste económico relativamente bajo (en comparación con clústeres de CPUs tradicionales con un rendimiento equivalente). Sin embargo, el uso de una única tarjeta GPU no es suficiente cuando se trata de simulaciones de varios millones de partículas, ya que los tiempos de ejecución son demasiado elevados y la capacidad de memoria de estos dispositivos puede resultar insuficiente. Por lo tanto, para simulaciones de gran tamaño resulta imprescindible combinar los recursos de varias GPUs.

DualSPHysics es una implementación del modelo SPH para estudiar los flujos de superficie libre. Forma parte del proyecto SPHysics [2 y 3], en el que colaboran investigadores de Johns Hopkins University (U.S.A.), Universidad de Vigo (España) y The University of Manchester (Reino Unido). DualSPHysics consiste en un código open-source escrito en C++ con OpenMP (Open Multi-Processing) y CUDA (Compute Unified Device Architecture), capaz de ejecutarse en CPU y GPU. En [4] se describen las optimizaciones aplicadas para sacarle el máximo partido a cada arquitectura. La validación de este código mediante un experimento real puede encontrarse en [5]. Más información sobre el proyecto DualSPHysics puede encontrarse en la web www.dual.sphysics.org.

En este trabajo se describe la implementación Multi-GPU diseñada a partir del código DualSPHysics, aprovechando así el elevado rendimiento de este código en una GPU para extenderlo a varias GPUs. Su eficiencia y rendimiento son presentados y analizados en este trabajo. También se muestra como una simulación de cientos de millones de partículas es posible en un pequeño clúster de GPUs.

II. IMPLEMENTACIONES PARALELAS

Esta sección describe las distintas implementaciones paralelas aplicadas al código DualSPHysics. En todas ellas se trata de repartir el tiempo de cálculo entre las distintas unidades de procesamiento disponibles adaptándose a las características propias de cada hardware. El código está estructurado en tres pasos fundamentales: creación de lista de vecinos (LV), cómputo de fuerzas de la interacción entre partículas (CF) y cálculo de las magnitudes de cada partícula en el siguiente instante de la simulación (CP). Siendo CF el paso con mayor coste computacional (más del 90%).

[1] EPHYSLAB Environmental Physics Laborarory, Universidad de Vigo, Campus As Lagoas s/n 32004 Ourense, e-mail: jmdominguez@uvigo.es

A. *OpenMP*

La primera aproximación paralela consiste en repartir el cómputo entre los distintos núcleos de una misma CPU. Como todos ellos comparten el mismo espacio de memoria se utiliza OpenMP. Esta técnica presenta importantes ventajas ya que no requiere perder tiempo en el intercambio de datos entre núcleos. El balanceo dinámico de carga es sencillo y no supone un coste importante. Y su implementación mediante directivas no complica el código.

B. *CUDA*

La arquitectura de cálculo paralelo desarrollada por Nvidia es utilizada para sacar partido de las capacidades de cómputo de la arquitectura GPU. En este caso, las tareas que es necesario realizar sobre las partículas se ejecutan usando el elevado número de hilos disponibles en la GPU. En esta aproximación, los datos de las partículas son almacenados en la memoria de la GPU de forma permanente, permitiendo reducir las transferencias entre memoria de GPU y CPU al mínimo. De esta forma, los tres pasos en que se organiza el código son implementados en GPU. Se implementó el mismo tipo de lista de vecinos que en CPU (descrita en [6]), haciendo uso del algoritmo radixsort [7] proporcionado por CUDA para ordenar las partículas de forma eficiente. La interacción entre partículas se implementó en GPU de forma que cada hilo calcula la fuerza de una partícula. Así, cada hilo busca sus partículas vecinas entre las celdas adyacentes y obtiene la fuerza final evaluando todos los pares de interacciones.

C. *MPI*

MPI (Message Passing Interface) consiste en un conjunto de directivas que hacen posible la comunicación entre distintos dispositivos. Esto permite combinar los recursos de varias máquinas que estén conectadas por red. Así, la capacidad de cómputo puede ser incrementada añadiendo más maquinas a la red. Sin embargo, la distribución de trabajo entre distintos dispositivos supone un aumento del tiempo de ejecución debido a: (i) nuevos tareas para el reparto de trabajo, (ii) intercambio de datos y (iii) sincronización.

III. NUEVA IMPLEMENTACIÓN MPI

En esta sección se describe con más detalle la implementación MPI propuesta en este trabajo y que da lugar a interesantes mejoras y a buenos resultados (sección IV), aunque también presenta cierta pérdida de eficiencia (sección V).

El dominio de la simulación se divide entre los distintos procesos MPI. Así, cada proceso sólo necesita asignar recursos para gestionar una fracción del número total de partículas, permitiendo que el tamaño de la simulación escale con el número de máquinas utilizadas.

Las dos causas principales que provocan pérdida de eficiencia al aumentar el número de procesos MPI son el intercambio de información entre procesos y la sincronización. En una implementación previa [8] puede verse como la comunicación supone un elevado porcentaje del tiempo de ejecución y como éste aumenta significativamente al incrementarse el número de procesos. Para evitar este problema es fundamental dividir el dominio del caso de forma que nos permita solapar las comunicaciones con el cálculo.

En cada iteración del método SPH, es necesario calcular el tiempo de paso en función de unos valores que sólo se obtienen una vez haya finalizado el cálculo de fuerzas de todas las partículas. Por tanto, todos los procesos deben haber completado el cálculo de fuerzas antes de poder calcular el tiempo de paso necesario para actualizar el estado de las partículas. Esto supone un punto de sincronización donde todos los procesos deben esperar hasta que termine el más lento. Esto puede provocar una pérdida de eficiencia importante porque el número de pasos necesarios para completar una simulación es muy elevado. Para minimizar este problema es necesario mantener el dominio repartido de forma equilibrada entre todos los procesos, reduciendo al mínimo la diferencia entre el tiempo de cómputo del más rápido y el más lento.

A. *Subdivisión del dominio*

Como ya se explicó antes, el dominio del caso es dividido en bloques de partículas que son asignados a los distintos procesos MPI. La división puede realizarse en cualquier dirección (X, Y o Z) para adaptarse a la naturaleza del caso de simulación. De esta forma, cada dominio (excepto los extremos) tiene un dominio vecino a cada lado. Por tanto, cada proceso para calcular las fuerzas ejercidas en sus partículas necesita las partículas de los procesos vecinos que estén dentro de la distancia de interacción (*2h*), a estas partículas les llamamos *halo* del proceso/subdominio o *borde* del proceso/subdominio vecino.

La figura 1 muestra la división de un dominio en tres subdominios (0, 1 y 2). Las partículas verdes pertenecen al subdominio1 pero algunas de ellas, las que están en el borde izquierdo a una distancia menor de *2h* del dominio 0, forman el halo del subdominio 0. Mientras que las del borde derecho forman el halo del subdominio 2.

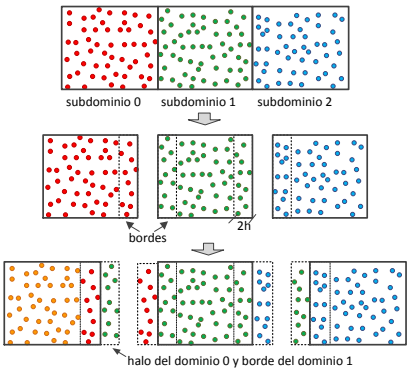

Fig. 1. Ejemplo de subdivisión de un dominio (halos y bordes).

A diferencia de lo presentado en [8], las partículas del dominio nunca se mezclan con las de los halos. Las partículas con sus datos son ordenadas en celdas durante la construcción de la lista de vecinos [6]. El orden puede ser XYZ, XZY o YZX según el eje de división sea Z, Y o X. De esta forma, las partículas siempre están ordenadas en rebanadas de ancho *2h*. El dominio asignado a cada proceso tiene un ancho mínimo de *6h*, para que exista una distancia mínima de *2h* entre los bordes del dominio.

Este método de subdivisión del dominio aporta una serie de importantes ventajas que permiten mejorar el rendimiento:

1. Las partículas de un dominio no se mezclan con las partículas de los halos. Así se evita perder tiempo en reordenar todas las partículas al recibir los halos antes del cálculo de fuerzas, y en volver a separarlas después. También reduce el consumo de memoria, ya que, del halo sólo se necesitan guardar los datos básicos de las partículas (posición, velocidad y densidad).
2. Los procesos pueden ajustar el tamaño del dominio de sus partículas a los límites de sus partículas fluido. Así, se reduce el tiempo de ejecución y la cantidad de memoria necesaria.
3. Los datos de las partículas son almacenados en rebanadas. Permitiendo enviar los datos de los bordes a los procesos vecinos más rápido, ya que los datos están agrupados en posiciones de memoria contiguas.
4. Este sistema mantiene agrupadas las partículas que interactúan con los halos. Así, es posible solapar el tiempo de recepción de los halos mientras se calcula la interacción del resto de partículas. Por ejemplo, las partículas amarillas en la figura 1 que pertenecen al subdominio 0, pueden calcular sus fuerzas mientras se espera el halo (partículas verdes) y una vez recibido se calcula su interacción con el borde (partículas rojas).

B. *Comunicación entre procesos*

Reducir el tiempo de intercambio de datos entre los distintos procesos MPI es fundamental para aumentar el número de procesos sin que la eficiencia disminuya drásticamente. La única forma de conseguir esto es solapar el tiempo de comunicación con el tiempo de cálculo mediante el uso de comunicaciones asíncronas. Se utilizan envíos de tipo asíncrono y recepciones de tipo síncrono. Así, cada proceso envía la información disponible de forma inmediata y continúa su ejecución sin esperar a que el envío concluya. Sólo cuando un proceso necesita información para continuar realiza una recepción síncrona.

La figura 2 muestra el intercambio de datos que se produce en cada paso de tiempo de la simulación con 3 procesos MPI. Las flechas azules representan el intercambio de información relativa a las partículas que cambian de dominio, las flechas verdes representan el envío de las partículas del borde de cada proceso a sus vecinos para el cálculo de fuerzas. Las flechas rojas indican un punto de sincronización donde la fuerza de todas las partículas debe haber sido ya computada para calcular el tiempo de paso (*dt*). Los procesos pertenecientes al cómputo de fuerza están marcados en gris.

Como puede verse en la figura 2, para solapar cálculo y comunicaciones se intercalan algunas tareas entre el envío y la recepción de datos. Por ejemplo, al iniciar el cómputo de fuerzas, cada proceso envía sus bordes a los procesos vecinos y mientras, calcula la interacción de las partículas que no interactúan con el halo. Después recibe los datos de un halo y realiza la interacción con ese halo, y finalmente repite el proceso con el segundo halo si lo hubiera.

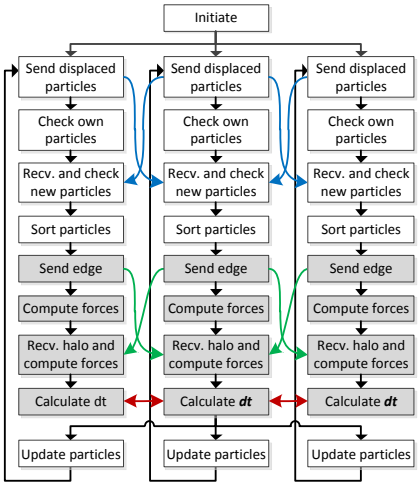

Fig. 2. Esquema de comunicaciones entre tres procesos MPI.

C. *Balanceo dinámico de carga*

Debido a la naturaleza lagrangiana del método SPH, las partículas se desplazan durante la simulación, por lo que es necesario redistribuir las partículas cada cierto número de pasos para mantener la carga de cómputo balanceada entre los procesos y minimizar el tiempo de sincronización. La mayor parte del tiempo de ejecución es consumido por el cálculo de fuerzas que depende del número de partículas fluidas. Por esta razón, el dominio debe ser dividido en subdominios con el mismo número de partículas fluidas (incluyendo los halos).

El balanceo dinámico de carga en función del número de partículas fluidas es la mejor opción cuando todos los procesos son ejecutados en máquinas con el mismo rendimiento. Sin embargo, otra aproximación basada en el tiempo de cálculo requerido por cada proceso fue implementada, permitiendo así combinar GPUs de distinta potencia. Este segundo tipo de balanceo dinámico utiliza una media ponderada de los últimos tiempos de ejecución en cada proceso MPI para lograr un balanceo de carga equilibrado en clústeres heterogéneos.

El balanceo dinámico de carga es comprobado cada cierto número de pasos y sólo se aplica cuando es necesario.

IV. Resultados

El experimento de Yeh and Petroff de la Universidad de Washington, también descrito en [9], fue utilizado como caso de estudio para mostrar el rendimiento de la implementación descrita aquí. Consiste en el colapso de un volumen de agua debido a la gravedad y su interacción con una estructura. La figura 3 muestra distintos instantes de la simulación del caso con 5 millones de partículas para simular 1.5 segundos físicos. Para este trabajo se utilizó el kernel Wendland [10] y el algoritmo de paso de tiempo Verlet [11].

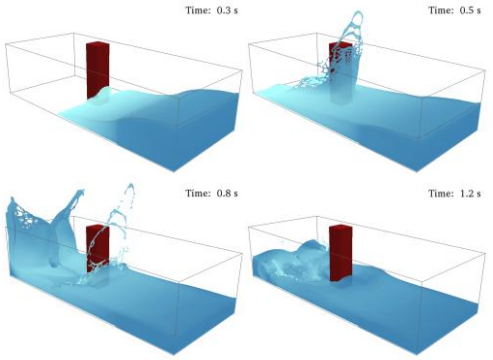

Fig. 3. Diferentes instantes de la evolución del fluido en el caso de estudio.

Las simulaciones se realizaron en dos sistemas hardware distintos, pertenecientes a las universidades de Vigo y Manchester cuyas características se muestran en la tabla I. La principal diferencia entre ambos consiste en que el sistema #1 está formado por un único nodo, mientras que el sistema #2 consiste en 7 nodos conectados mediante Infiniband (40 Gbit/s).

TABLA I

CARACTERÍSTICAS DE LOS SISTEMAS HARDWARE EMPLEADOS.

Sistema #1 **Universidad de Vigo**	**Sistema #2** **Universidad de Manchester**
1 nodo	7 nodos
2x CPU: Intel Xeon E5620 4 cores a 2.4 GHz 4x GPU: GTX480 480 cores a 1.37 GHz 1.5 GB GDDR5	2x CPU: AMD Opteron 6 cores 2x GPU: Tesla M2050 448 cores a 1.15 GHz 3 GB GDDR5

A. Aplicando el balanceo de carga

Un balanceo de carga eficaz es fundamental para maximizar el rendimiento de cualquier aplicación con MPI. En este trabajo se utilizó primero un balanceo dinámico de carga en función del número de partículas. El balanceo es comprobado cada 50 pasos de cálculo, pero sólo es aplicado cuando un nuevo reparto reduzca la diferencia de partículas entre los procesos en más de un 3%.

La figura 4 muestra distintos instantes de una simulación con 12M de partículas utilizando las 4 GPUs del sistema #1. Los límites de cada subdominio están representados en la figura mediante cajas y se puede observar como su tamaño varía con el número de partículas.

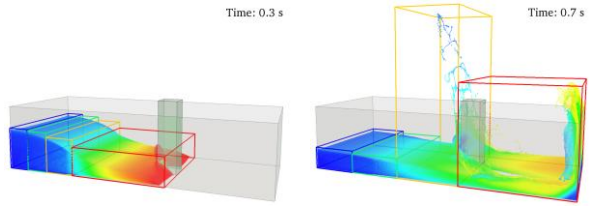

Fig. 4. División del dominio en 4 subdominios.

La simulación de 1.5 segundos físicos necesitó 58,318 pasos, pero el balanceo sólo se comprobó en 1,166 consumiendo un 0.03% del tiempo total. Y finalmente fue aplicado en sólo 93 ocasiones consumiendo un 0.01% del tiempo.

La figura 5 muestra el porcentaje de partículas fluidas (nf) asignadas a cada proceso MPI durante la simulación. Puede verse como los valores están muy próximos al 25%, siendo este el valor óptimo al balancear entre 4 procesos.

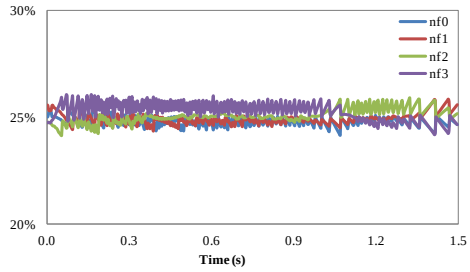

Fig. 5. Distribución de partículas entre los cuatro procesos MPI.

En la figura 6 se muestra el tiempo de ejecución consumido por el cálculo de fuerzas en cada proceso MPI. Como el número de partículas está balanceado de forma equilibrada, los tiempos son también muy similares.

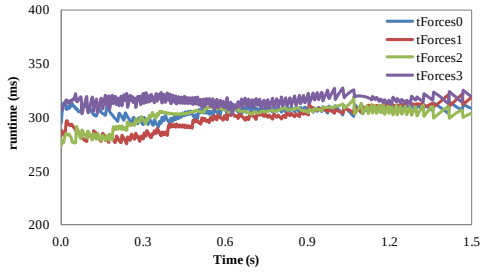

Fig. 6. Tiempo de ejecución del cálculo de fuerzas.

B. Eficiencia y tiempos de ejecución

Se ha medio la eficiencia de la implementación MPI en ambos sistemas hardware. La eficiencia *E* es definida como el ratio entre el número de pasos procesados por segundo con el código Multi-GPU y el número de GPUs empleadas dividido por el número de pasos procesados por la implementación Single-GPU (figura 7).

$$E = \frac{\frac{steps\ per\ second\ of\ multiGPU}{number\ of\ GPUs}}{steps\ per\ second\ of\ singleGPU}$$

Fig. 7. Fórmula empleada para el cálculo de eficiencia.

La eficiencia lograda en el sistema hardware #1 simulando de 1 a 8 millones de partículas (N) es mostrada en la figura 8. A partir de los 4 millones de partículas se ha obtenido una eficiencia del 98% usando 2 GPUs y superior al 92% con 4 GPUs.

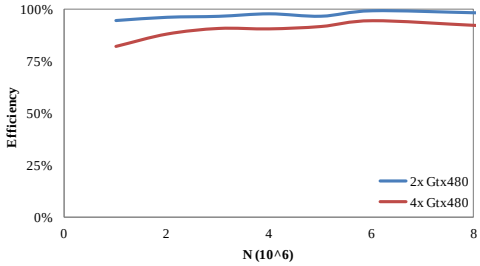

Fig. 8. Eficiencia alcanzada en el sistema hardware #1.

La eficiencia lograda con el sistema hardware #2 se muestra en la figura 9. En este caso se ha simulado de 1 a 16 millones de partículas, utilizando hasta un máximo de 14 GPUs. Se puede ver como la eficiencia aumenta con el número de partículas y disminuye al aumentar el número de GPUs. Con 14 GPUs se ha conseguido una eficiencia del 81% simulando 16 millones de partículas.

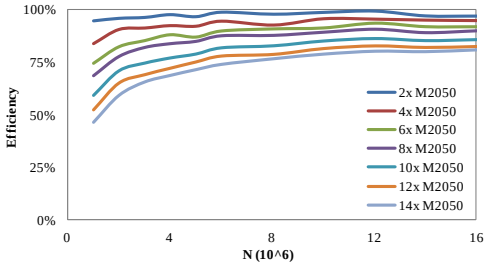

Fig. 9. Eficiencia alcanzada en el sistema hardware #2.

En la figura 10 pueden observarse los tiempos de ejecución para distinto número de partículas. Por ejemplo, se ha simulado 16M con 14 GPUs del sistema #2 en 3.7 horas frente a las 42.6 horas utilizadas con una GPU del mismo sistema. Utilizando las 4 GPUs del sistema #1 se realizó la misma simulación en 8.4 horas, no pudiendo realizarse en una GPU de este sistema por su menor capacidad de memoria. Con 4 GPUs del sistema #1 se pudo simular hasta 24M usando celdas de tamaño *h* y hasta 32 usando celdas de tamaño *2h*.

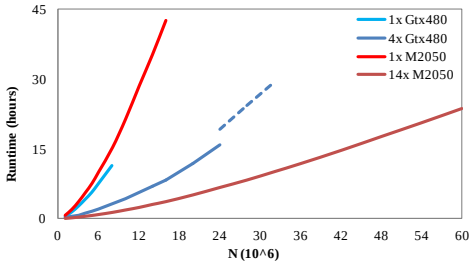

Fig. 10. Tiempos de ejecución para simulaciones de hasta 60M.

Información más detallada del tiempo de ejecución y del límite de cada sistema hardware puede verse en la tabla II.

TABLA II

TIEMPOS DE EJECUCIÓN Y TAMAÑO MÁXIMO DE SIMULACIONES.

GPU	*Cells (h)*		*Cells (2h)*	
	Max. size	**Runtime**	**Max. size**	**Runtime**
1x Gtx480	8M	11.5 h	10M	19.9 h
4x Gtx480	24M	15.8 h	32M	29.4 h
1x M2050	16M	42.6 h	18M	64.2 h

V. PERDIDA DE EFICIENCIA

Esta sección describe los principales inconvenientes que presenta la implementación Multi-GPU. En las figuras 11 y 12 puede verse el tiempo de ejecución dedicado a tareas propias de la versión con MPI. Concretamente la figura 11 muestra el porcentaje del tiempo de ejecución de diferentes tareas cuando simulamos 16M de partículas usando 4, 8 y 12 GPUs del sistema #2. El tiempo dedicado a tareas específicas de Multi-GPU se incrementa con el número de GPUs utilizadas. Por otro lado, la figura 12 muestra el impacto de esas tareas usando 8 GPUs para simular diferentes números de partículas (8, 12 y 16M). Como se esperaba, los porcentajes dedicados a las tareas de comunicación decrecen con el tamaño de la simulación.

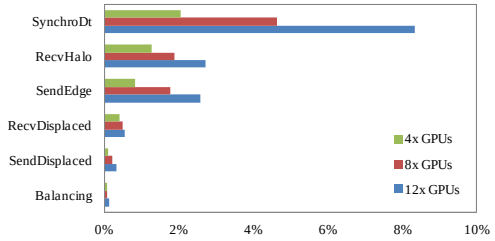

Fig. 11. Porcentaje de tareas propias de MPI simulando 16 en GPUs Tesla 2050 del sistema hardware #2.

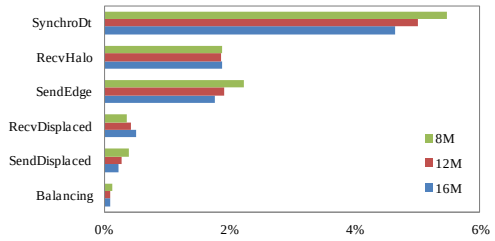

Fig. 12. Porcentaje de tareas propias de MPI usando 8 GPUs Tesla 2050 del sistema hardware #2.

En todos los pasos de tiempo de la simulación, cada proceso tiene que enviar los datos de las partículas de sus bordes a los procesos vecinos (**SendEdge**) y recibir las partículas de sus halos (**RecvHalo**). Esto supone un volumen importante de información que se incrementa con el número de procesos involucrados. Estas transferencias entre procesos pueden ser ocultadas cuando se solapan con el cálculo de fuerzas. Durante la creación de la lista de vecinos, los procesos MPI deben comunicarse entre ellos para enviar y recibir las partículas que cambian de subdominio (**SendDisplaced** y **RecvDisplaced**). En este caso el volumen de información es 3 órdenes de magnitud menor que en el caso de los halos. Antes del cómputo de paso, todos los procesos deben sincronizarse para calcular el nuevo paso de tiempo (**SynchroDt**). Como el balanceo no es perfecto, los procesos más rápidos tienen que esperar por el más lento. El coste computacional de esta sincronización es el mayor de los mostrados en las figuras 11 y 12 y representa una de las principales causas de pérdida de rendimiento. El balanceo dinámico de carga (**Balancing**) supone un coste adicional propio de la versión MPI. Sin embargo, esta tarea consume menos del 0.1% del tiempo total de ejecución y es vital para conseguir una buena eficiencia al reducir el tiempo de sincronización.

VI. APLICACIÓN

Una vez que la implementación MPI ha sido descrita y su eficiencia mostrada, esta nueva versión Multi-GPU de DualSPHysics se ha utilizado para llevar a cabo una simulación de gran tamaño.

La aplicación consiste en la interacción de una gran ola con una plataforma petrolífera usando dimensiones

reales durante 10 segundos de tiempo físico. La configuración del caso y sus dimensiones se muestran en la figura 13. La distancia inicial entre partículas es de 6.2 cm dando lugar a una simulación de 226,307,484 partículas (220,556,367 partículas de fluido) .

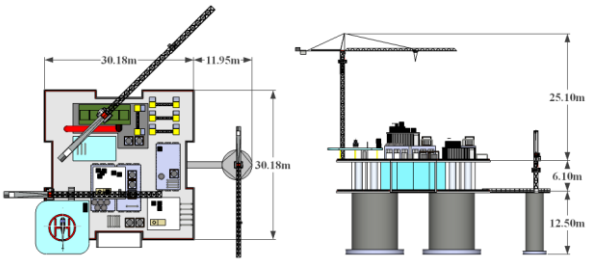

Fig. 13. Configuración inicia de la simulación.

La simulación fue llevada a cabo usando las 14 GPUs Tesla 2050 del sistema #2. Se han ejecutado 153,818 pasos de que han necesitado 82.5 horas. Los datos se han guardado cada 0.05 segundos físicos, lo que supone más de 1,275 GB de datos de salida.

La figura 14 muestra el porcentaje de memoria por GPU. Simulando 226 millones de partículas con 14 GPUs se ha utilizado más del 90% de los recursos disponibles.

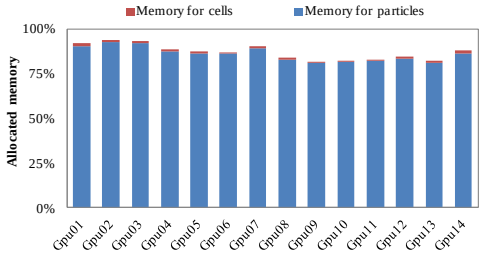

Fig. 14. Porcentaje de memoria reservada por GPU M2050.

Diferentes instantes de la simulación pueden verse en la figura 15. Es importante comentar que la preparación y posterior visualización de una simulación con tal cantidad de partículas no fue sencilla y obligó a adaptar las herramientas de pre-processing y post-processing para mejorar su eficiencia.

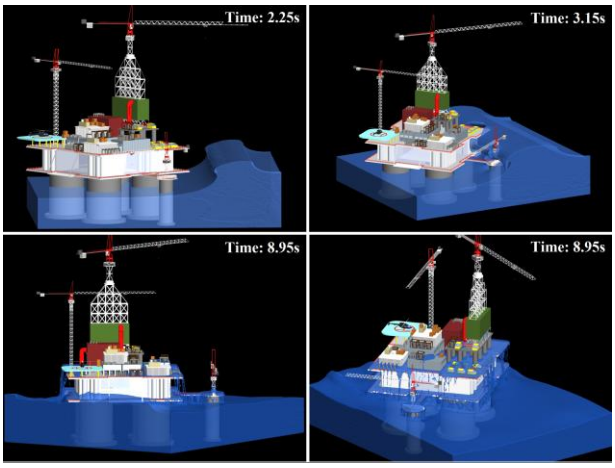

Fig. 15. Simulación de una gran ola impactando con una plataforma petrolífera usando 226 millones de partículas con DualSPHysics.

VII. CONCLUSIONES Y TRABAJO FUTURO

Se ha presentado una nueva implementación Multi-GPU del método SPH usando MPI, capaz de combinar la potencia de múltiples GPUs de iguales o distintas características.

Se ha mostrado como la implementación MPI descrita permite simular un gran número de partículas (32M) en una máquina con 4 GPUs y más de 226M en un clúster con 14 GPUs con tiempos de ejecución razonables.

Es necesario seguir trabajando para conseguir solapar completamente el tiempo de comunicación con el tiempo de cálculo, y así poder aumentar el número de procesos sin perder eficiencia.

El balanceo dinámico de carga además de tratar de igualar los tiempos de ejecución de cada proceso debería tener en cuenta las limitaciones de memoria de cada GPU para poder aprovechar toda la memoria disponible si fuese necesario.

AGRADECIMIENTOS

El presente trabajo ha sido financiado por la Xunta de Galicia, Programa de Consolidación e Estructuración de Unidades de Investigación Competitivas (Grupos de Referencia Competitiva) cofinanciado por European Regional Development Fund (FEDER). Los autores también desean agradecer el soporte en temas de hardware proporcionado por Orlando Garcia Feal.

REFERENCIAS

[1] R. A. Gingold and J. J. Monagham, *"Smoothed particle hydrodynamics: theory and application to non- spherical stars"*, Mon Not R Astr Soc 181: 375-389, 1977.

[2] M. Gómez-Gesteira, B.D. Rogers, A.J.C. Crespo, R.A. Dalrymple, M. Narayanaswamyc and J.M. Dominguez, *"SPHysics - development of a free-surface fluid solver - Part 1: Theory and Formulations"*, Computers & Geosciences, doi: 10.1016/j.cageo.2012.02.029, 2012.

[3] M. Gómez-Gesteira, A.J.C. Crespo, B.D. Rogers, R.A. Dalrymple, J.M. Dominguez and A. Barreiro, *"SPHysics - development of a free-surface fluid solver - Part 2: Efficiency and test cases"*, Computers & Geosciences, doi: 10.1016/j.cageo.2012.02.028, 2012.

[4] J.M. Dominguez, A.J.C. Crespo and M. Gómez-Gesteira. *"Optimization strategies for CPU and GPU implementations of a Smoothed Particle Hydrodynamics method"*. Submitted to Computer Physics Communications.

[5] A. J. C. Crespo, J. M. Dominguez, A. Barreiro, M. Gómez-Gesteira and B. D. Rogers, *"GPUs, a new tool of acceleration in CFD: Efficiency and reliability on Smoothed Particle Hydrodynamics methods"*, PLoS ONE, doi: 10.1371/journal.pone.0020685, 2011.

[6] J. M. Dominguez, A. J. C. Crespo, M. Gómez-Gesteira and J. C. Marongiu, *"Neighbour lists in Smoothed Particle Hydrodynamics"*, International Journal for Numerical Methods in Fluids, doi: 10.1002/fld.2481, 2010.

[7] N. Satish, M. Harris and M. Garland, *"Designing efficient sorting algorithms for manycore GPUs"*, Proceedings of IEEE International Parallel and Distributed Processing Symposium 2009, 2009.

[8] D. Valdez-Balderas, J. M. Dominguez, A. J. C. Crespo and B. D. Rogers, *"SPH simulations on multi-GPU clusters"*, Submited to Journal of Parallel and Distributed Computing.

[9] M. Gómez-Gesteira y R. Dalrymple, *"Using a 3D SPH method for wave impact on a tall structure"*, Journal of Waterway, Port, Coastal, and Ocean Engineering, 130(2): 63-69, 2004.

[10] H. Wendland H, *"Piecewiese polynomial, positive definite and compactly supported radial functions of minimal degree"*, Advances in computational Mathematics, 4(1): 389-396, 1995.

[11] L. Verlet, *"Computer experiments on classical fluids"*, I. Thermodynamical properties of Lennard-Jones molecules, Phys Rev, 159: 98-103, 1967.