Análisis de uso de un algoritmo de balanceo de carga estático en un Cluster Multi-GPU Heterogéneo

Erica Montes de Oca[1], Laura De Giusti[1], Franco Chichizola[1],
Armando De Giusti[1,2], Marcelo Naiouf [1]

[1] Instituto de Investigación en Informática LIDI (III-LIDI)
Facultad de Informática – Universidad Nacional de La Plata
La Plata, Buenos Aires, Argentina

[2] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)

{emontesdeoca, ldgiusti, francoch, degisuti, mnaiouf}@lidi.info.unlp.edu.ar

Resumen. En este trabajo se realiza un análisis de la influencia de usar un mecanismo de distribución de tareas estático en un cluster heterogéneo de GPUs, basado en la potencia de cómputo de las mismas. Se utiliza como caso de estudio el problema de atracción gravitacional de los cuerpos en el espacio (*N*-Body). Se presentan dos soluciones empleando una combinación de MPI-CUDA, que se diferencian en la forma de distribuir el trabajo: homogénea y heterogénea. Se detallan y analizan los resultados experimentales, mostrando un porcentaje de mejora de aproximadamente veinticinco por ciento al tener en cuenta las características de la arquitectura.

Palabras claves: GPU, Balance de Carga, Cluster Heterogéneo, MPI-CUDA, N-Body.

1. Introducción

En los últimos años, el avance de la GPU ha alcanzado a un conjunto de aplicaciones de propósito general, logrando disminuir sus tiempos de ejecución. El diseño de su arquitectura y su modo de procesamiento, han hecho de las mismas una opción a tener en cuenta a la hora de acelerar la ejecución de aquellas aplicaciones adaptables a estas plataformas [1][2]. La cantidad de transistores dedicados al cálculo intensivo de datos son la clave para que las GPUs sean más rápidas que las CPUs [3][4].

En las GPUs, la organización de los threads puede verse por nivel o jerárquicamente de la siguiente manera: grilla o g rid (formado por bloques de threads), bloque (conformado por threads) y threads. En un bloque, los hilos están organizados en warps (todos los hilos de un warp son planificados juntos durante la ejecución). Por otro lado, el sistema de memoria de la GPU está compuesto por distintos tipos de memorias: Memoria Global, Memoria de Textura, Memoria Constante, Memoria Shared o Compartida, Memoria Local y Registro; los cuales se diferencian en la forma de acceso de los hilos, las limitaciones en las operaciones permitidas y su ubicación dentro del dispositivo [5][2].

En busca de lograr aún mayor potencia de cómputo pueden utilizarse más de una GPU en una misma arquitectura. En ese sentido surgen dos variantes: una solución orientada a arquitecturas de memoria compartida (múltiples GPUs en una máquina) y otra a arquitecturas distribuidas (cluster de múltiples máquinas cada una con GPU) [6]. También pueden combinarse ambas y formar un cluster de máquinas con múltiples GPUs. Este tipo de cluster en sí mismo es una arquitectura híbrida: el subsistema conformado por varios cores CPUs (y su sistema de memoria y entrada/salida) y el subsistema GPU (con sus memorias on y off chip) [7]. Estos dos subsistemas se comunican por medio de un bus PCI-E con una velocidad de ancho de banda baja comparada con las velocidades de comunicación de los sistemas de memoria. Por lo cual, el cuello de botella es la comunicación que existe entre dichos subsistemas [8] [9][10]. Si dejamos de lado el componente CPU en la arquitectura antes mencionada, podemos hablar de cluster heterogéneo cuando las características de las diferentes GPUs que intervienen son diferentes.

La aceleración que se logra con el uso de más de una GPU en la arquitectura varía dependiendo de la aplicación desarrollada. Por ejemplo, en [5] se compara el procesamiento del cifrado AES (Advanced Encryption Standard) en una GPU y en un cluster de GPU; en este caso no se obtiene una ventaja por el uso del cluster. En cambio en [11], donde se resuelve el problema N-Body, se logra una ventaja significativa al usar más de una GPU (todas con iguales características).

En este trabajo se presenta la resolución del problema N-Body utilizando un Cluster heterogéneo de máquinas con múltiples GPUs de diferentes características, y se analiza la mejora obtenida al distribuir el trabajo teniendo en cuenta la diferencia de potencia entre las GPUs. La Sección 2 plantea el problema a estudiar. En la Sección 3 se describen las soluciones implementadas en este trabajo. La Sección 4 contiene los resultados experimentales. La Sección 5 presenta las conclusiones y los trabajos futuros.

2. Problema de alta demanda computacional: *N*-Body

Como caso de estudio se presenta el problema *N*-Body, el cual consiste en simular el comportamiento (en cuanto al movimiento) de *N* cuerpos que componen el espacio de trabajo. Este problema es muy estudiado y aplicado en distintos campos de la ciencia tales como la química (por ejemplo, cálculo de la atracción de partículas subatómicas) o la astronomía (estudio de la atracción de los cuerpos en el espacio) [12][13]. En particular, el presente artículo se basa en la aplicación de la Ley de Atracción de Newton [14].

Inicialmente se cuenta con información de la masa (*m*), la velocidad (*v*) y la posición (*p*) de cada cuerpo. Posteriormente, en cada paso de simulación se debe calcular la nueva posición y velocidad de cada uno de los elementos en el espacio de simulación. Para esto es fundamental determinar la fuerza de atracción gravitacional entre cada par de cuerpos [13][15][2].

Por medio de la Ecuación (0) se calcula la *magnitud de* la *fuerza de atracción (F)* entre el par de cuerpos *i* y *j*. Este es el cálculo central de procesamiento.

$$F(i,j) = \frac{G \times m_i \times m_j}{r^2} \qquad (0)$$

siendo r *= distancia calculada a partir de* p_i *y* p_j
G *= constante gravitacional*

La magnitud y la dirección (calculadas en base a la posición de ambos cuerpos) determinan la fuerza ejercida por el cuerpo *j* sobre el *i*. La *fuerza total* sobre un cuerpo *i* es la suma de las fuerzas ejercidas por cada uno de los otros cuerpos sobre él. A partir de la *fuerza total* ejercida sobre un cuerpo *i* y la información actual del mismo (velocidad, masa y posición) se calcula su nueva posición y velocidad.

El cálculo de la fuerza de atracción gravitacional es independiente para cada cuerpo en un paso de simulación, por lo que el problema es fácilmente adaptable a la arquitectura GPU [1].

En trabajos anteriores, se ha mostrado que la resolución de este problema usando una GPU se ve beneficiada de manera significativa al compararla con las versiones paralelas desarrolladas en memoria distribuida, memoria compartida y en la combinación de ambas para la arquitectura CPU [2]. Además, se ha planteado en [3][16], el uso de un cluster homogéneo de GPU obteniendo buenos resultados. En el presente trabajo se utiliza un cluster heterogéneo de GPUs, y se analiza cómo influye la forma de distribuir el trabajo en la solución.

3. Descripción de la solución

En este trabajo las soluciones implementadas se basan en el algoritmo *Fast N-Body Simulation*, el cual es apto para ser ejecutado en la GPU por la independencia de cómputo en el cálculo de la fuerza de atracción gravitacional de los cuerpos. Cada cuerpo debe resolver su fuerza gravitacional dependiendo de las posiciones de los demás sin modificar otro dato que no sea su propia fuerza. Del mismo modo, la actualización de posiciones y velocidades se realiza de manera independiente [17].

Para las implementaciones en las GPUs de este trabajo se utiliza CUDA, la cual permite definir una función denominada *kernel* que es ejecutada *n* veces, siendo *n* la cantidad de threads de la aplicación. Como el kernel se ejecuta en la GPU, la memoria en ella debe ser alocada antes de que la función sea invocada y los datos que requiera utilizar sean copiados desde la memoria de la CPU. Una vez ejecutada la función kernel, los datos de la memoria de la GPU deben ser copiados a la memoria de la CPU [8][9].

3.1. Solución con una GPU

En primera instancia, se comunican a la GPU los datos de los cuerpos previamente inicializados por la CPU a través del PCI-E. A continuación, la CPU delega la ejecución a la GPU, y queda en espera de la respuesta de la misma.

La información comunicada se almacena en la memoria global de la GPU en estructuras de datos utilizadas para mantener la velocidad, posición y masa de cada cuerpo, y para la fuerza de gravedad que se calcula.

En el kernel o función que calcula la fuerza de atracción gravitacional, cada thread calcula su posición en la memoria global que le permite acceder al cuerpo para el que tiene que calcular dicha fuerza. Se utilizan cuatro vectores con dimensión igual a la cantidad de threads por bloque, tres de los cuales son para almacenar una porción del vector de posiciones (una para cada coordenada), y el cuarto para una porción del vector de masas. Eso permite computar por bloque reduciendo el acceso a memoria global.

Además de su identificador en la memoria global, cada thread calcula su identificador dentro del vector en memoria compartida. Cada uno trae de memoria global la posición y la masa del cuerpo que le corresponde, por lo que el vector almacenado en memoria compartida es cargado por todos los threads, y la posición del cuerpo que le corresponde será almacenada en la dirección relacionada con su identificador dentro de la memoria shared. Antes de comenzar a calcular la fuerza de atracción, los threads son sincronizados para garantizar que todas las posiciones de los vectores de la memoria compartida hayan sido cargadas.

Luego, cada thread realiza el cálculo correspondiente a la fuerza de atracción para su cuerpo con las posiciones que se encuentran en los vectores de posiciones de la memoria compartida. Cuando todos los threads del bloque hayan utilizado todas las

posiciones de los vectores de la memoria shared, deben sincronizarse para volver a traer una porción más de las posiciones y de las masas de los cuerpos de la memoria global a la memoria compartida. Este ciclo se repetirá hasta que todo el vector de posiciones y masas haya sido utilizado. Finalmente, cada thread calcula la velocidad y posición del cuerpo que tiene asignado. Cuando los pasos de simulación a realizar se han completado, la GPU enviará a la CPU los resultados calculados, y le devolverá el control de la ejecución.

3.2. Solución con un cluster heterogéneo de GPUs

Basada en la solución de la Sección 3.1, se realiza una adaptación para ser ejecutada en un cluster de GPUs. Para esto se requiere la utilización de una combinación de MPI con CUDA. En este caso se utiliza un esquema master/worker de procesos que se comunican por medio de MPI. El *master* es el encargado de inicializar la información de los *N* cuerpos y distribuir el trabajo entre los *workers*, los cuales tienen asociada una GPU cada uno.

Cada worker (después de haber recibido del master la información necesaria) envía por medio de PCI-E dichos datos a su respectiva GPU, siendo éstas las que realmente realizan cómputo, ya que los procesos MPI se limitan a comunicar entre ellos los datos resultantes de cada paso de simulación, para que cada GPU conozca la información calculada por el resto de los procesos.

En la GPU, los datos son almacenados en la *memoria global* (off-chip) y luego son cargados a la *memoria shared* (on-chip) en porciones de 256 datos. La copia de los datos a esta última memoria, así como la reserva de espacio en la misma, es tarea del programador. Cabe destacar que al inicio del algoritmo se le debe enviar los datos de los *N* cuerpos a la GPU. Una vez que la GPU ha calculado la nueva posición y velocidad de los cuerpos que le corresponde, envía dichos datos a la CPU. A continuación, el proceso MPI comunicará esta información al resto de los procesos, recibiendo al mismo tiempo los datos calculados por los demás. Dicho procesamiento se repetirá *t* veces (donde t es la cantidad de pasos de simulación).

Un punto importante a tener en cuenta en esta solución es la forma en que se reparte el trabajo entre los *workers* (subconjunto de cuerpos que debe resolver cada uno), teniendo en cuenta que las GPUs asociadas a cada uno de ellos son diferentes. Para analizar el efecto de la distribución se utilizan dos alternativas para repartir las tareas: *homogénea* y *heterogénea*.

3.2.1. Distribución Homogénea

En el caso de la *Distribución Homogénea*, a todos los *workers* se les asigna la misma cantidad de cuerpos, es decir que cada GPU va a resolver la simulación de *N/p* cuerpos (siendo *p* la cantidad de *workers*, y suponiendo *N* múltiplo de *p*).

3.2.2. Distribución Heterogénea

En la *Distribución Heterogénea* se tiene en cuenta la potencia de cada GPU para repartir el trabajo en base a la misma, y de esta manera conseguir que todas trabajen aproximadamente el mismo tiempo en lugar de hacer la misma cantidad de tareas, buscando de esta manera reducir el desbalance de trabajo y como consecuencia el tiempo final de la aplicación.

Debido a que la potencia de cada GPU depende en gran medida de las características de la aplicación, se ejecuta la solución descrita en la Sección 3.1 en cada GPU, resolviendo un problema de tamaño reducido. A partir de esos tiempos de ejecución se calcula la *potencia de cómputo relativa (pcr)* de cada GPU con respecto a la mejor potencia por medio de la Ecuación (1). En este trabajo, la *pcr* de cada GPU se determina previo a la ejecución de la aplicación, pasando esta información como parámetro en el momento de ejecutarla.

$$pcr_i = \frac{Tiempo_{\,mejor\ GPU}}{Tiempo_i} \tag{1}$$

Cuando comienza la aplicación en el *master* se suman las potencias de cómputo relativa de cada GPU y se obtiene la *potencia de cómputo total (pct)* del cluster. En base a estos valores el *master* distribuye los cuerpos entre los *workers*. La *cantidad de cuerpos (cc)* para cada uno se calcula de acuerdo a la Ecuación (2).

$$cc_i = \frac{pcr_i \times N}{pct} \tag{2}$$

4. Resultados experimentales

En este trabajo, se utilizó un cluster heterogéneo de GPU, compuesto por 3 nodos (CPU) con 2 GPUs en cada uno. Las características de cada nodo son las siguientes:

- Quad Core Intel i5-2310 de 2,9 GHz, con dos GPUs Tesla C2050 con CUDA 4.2 y MPI 1.8.5.
- Dual Core Pentium G2020 de 2,9 GHz, con dos GPUs GeForce GTX 480 con CUDA 6.0 y MPI 1.8.4.
- Quad Core Intel i5-2310 de 2,9 GHz, con dos GPUs GeForce GTX 560 Ti con CUDA 7 y MPI 1.8.5.

En todas las pruebas realizadas se hicieron 10 pasos de simulación, y la cantidad de cuerpos (*N*) varió entre 1.024.000, 2.048.000 y 4.096.000. En las pruebas se utilizaron bloques de threads de 256 hilos para todas las GPUs.

En primera instancia se realizaron las pruebas utilizando una sola GPU (solución presentada en la Sección 3.1) para obtener el tiempo de ejecución de cada una de ellas y poder determinar la potencia de cómputo relativa (*pcr*). A partir de estas pruebas se obtuvieron las *pcr* que se muestran en la Tabla 1.

Tabla 1. Potencia de cómputo relativa (*pcr*) de cada GPU.

GPU	*Pcr*
GPU_0 y GPU_1 (Tesla C2050)	0,77
GPU_2 y GPU_3 (GeForce GTX 480)	1
GPU_4 y GPU_5 (GeForce GTX 560)	0,58

A partir de estas pruebas también se obtiene el mejor tiempo al usar una única GPU, el cual sirve de referencia para determinar la mejora conseguida al usar un cluster de GPUs. En la Tabla 2, se muestra el tiempo en segundos al usar la mejor GPU para los 3 tamaños de *N* (obtenidos como promedio de 15 ejecuciones para cada caso).

Tabla 2. Tiempo en segundos al usar la mejor GPU.

N	*Tiempo (seg)*
1.024.000	2624,64
2.048.000	10501,05
4.096.000	41980,43

En segunda instancia, se utilizó el cluster completo para realizar las pruebas correspondientes a las soluciones de la Sección 3.2.1 (*distribución homogénea*) y Sección 3.2.2 (*distribución heterogénea*). Para el segundo caso, se consideraron las *pcr* obtenidas en las pruebas anteriores (Tabla 1). En la Tabla 3 se muestran los tiempos en segundos obtenidos en estas pruebas (también como promedio de 15 ejecuciones en cada caso). En la Figura 1 se muestran estos valores.

Tabla 3. Tiempo en segundos de las pruebas en el cluster con distribución homogénea y heterogénea.

N	*Tiempo (seg) Distribución Homogénea*	*Tiempo (seg) Distribución Heterogénea*
1.024.000	1262,20	953,52
2.048.000	4986,76	3765,59
4.096.000	19873,14	14938,82

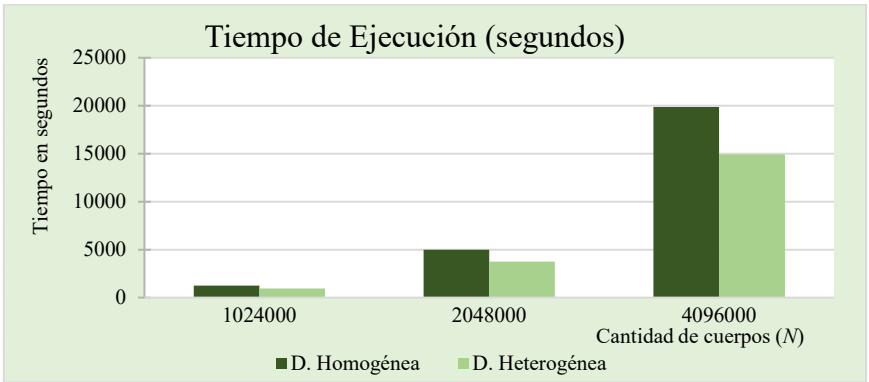

Figura 1. Tiempo en segundos de las pruebas en el cluster con distribución homogénea y heterogénea.

Al comparar los tiempos obtenidos con cada tipo de distribución se puede observar que al distribuir de acuerdo a la potencia de cada GPU (para esta aplicación en particular), se logra reducir el tiempo aproximadamente un 25%.

Se puede analizar la mejora conseguida al usar el cluster de GPUs si comparamos estos tiempos (Tabla 3) con los conseguidos al usar una sola GPU (Tabla 2). Esta mejora se calcula por medio de la Ecuación (3).

$$Mejora = \frac{Tiempo\ en\ una\ GPU}{Tiempo\ en\ Cluster} \tag{3}$$

A su vez, teniendo en cuenta que la máxima mejora que se puede obtener en el cluster está dada por la *potencia de cómputo total (pct)*, se puede estudiar la forma en que se aprovecha la arquitectura por medio de la Ecuación (4), siendo *pct = 4,7* en esta arquitectura. En la Tabla 4 se muestra la *mejora* y el *aprovechamiento* al usar el cluster para ambos tipos de distribución.

$$Aprovechamiento = \frac{Mejora}{pct} \tag{4}$$

Tabla 4. *Mejora* y *aprovechamiento* al usar el cluster con distribución homogénea y heterogénea.

N	***Distribución Homogénea***		***Distribución Heterogénea***	
	Mejora	***Aprovechamiento***	***Mejora***	***Aprovechamiento***
1.024.000	2,0794	0,44	2,7526	0,58
2.048.000	2,1058	0,45	2,7887	0,59
4.096.000	2,1124	0,45	2,8102	0,60

En la Figura 2 se muestra el *aprovechamiento* conseguido con ambos tipos de distribución, donde se puede observar una mejora importante al distribuir el trabajo de acuerdo a la potencia de cada GPU.

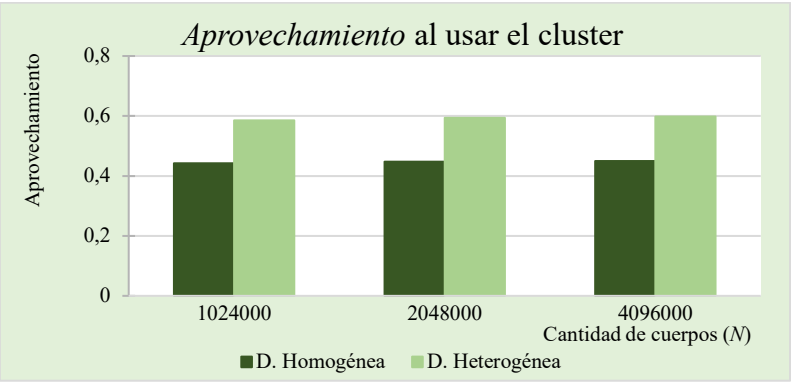

Figura 2. *Aprovechamiento* al usar el cluster con distribución homogénea y heterogénea.

Si bien la distribución heterogénea logra una mejora importante, la misma dista de aprovechar completamente el cluster debido a que en cada paso de simulación todos los procesos MPI deben sincronizar e intercambiar los nuevos datos de los cuerpos. Esto no sólo produce un retardo por el intercambio de mensajes entre los procesos, sino que genera un retardo mayor al tener que copiar nuevamente en la memoria de la GPU la información de todos los cuerpos.

5. Conclusiones y trabajos futuros

En este trabajo se presenta la solución a u n problema con alta demanda computacional (N-Body) utilizando un cluster heterogéneo de CPUs y GPUs, analizando los resultados obtenidos al tomar en cuenta las características de las GPUs para la distribución del trabajo.

Puede observarse que la mejora alcanzada considerando las potencias de cómputo relativas, se encuentra en el orden del 25% para la solución sobre un cluster de 3 CPUs con 2 GPUs. Asimismo, el aprovechamiento de la arquitectura ronda el 60% para la solución mencionada, mientras que si no se consideran las características de las GPUs el mismo es del 45%.

Como trabajos futuros se plantea agregar más nodos en el cluster para analizar el comportamiento al escalar la arquitectura y aumentar la heterogeneidad. Y por otro lado, probar con aplicaciones que tengan características diferentes para realizar un estudio más completo de la influencia de esta distribución de trabajo en un cluster de GPUs.

Referencias

[1] Jeroen Bédorf. *High Performance Direct Gravitational N-body Simulations on Graphics Processing Units*. Universiteit van Amsterdam (2007).

[2] Montes de Oca Erica, De Giusti Laura, De Giusti Armando, Naiuof Marcelo. *Comparación del uso de GPU y cluster de multicores en problemas con alta demanda computacional*. XVIII Congreso Argentino de Ciencias de la Computación, pág. 267-275. (2012).

[3] Montes de Oca Erica, De Giusti Laura, Rodriguez Ismael, De Giusti Armando, Naiuof Marcelo. *Análisis de la escalabilidad y el consumo energético en soluciones paralelas sobre cluster de multicores y GPU para un problema con alta demanda computacional.* XIX Congreso Argentino de Ciencias de la Computación (2013).

[4] Pérez C., Piccoli M. F. *Estimación de los parámetros de rendimiento de una GPU*. Mecánica Computacional. Vol. XXIX, pp. 3155-3167 (2010).

[5] Adrian Pousa, Victoria Sanz, Armando De Giusti. *Análisis de rendimiento de un algoritmo de criptografía simétrica sobre GPU y Cluster de GPU.* HPC La TAM 2013. (2013).

[6] Volodymyr V. Kindratenko, Jeremy J. Enos, Guochun Shi, Michael T. Showerman, Galen W. Arnold, John E. Stone, James C. Phillips, Wen-mei Hwu. *GPU Clusters for High-Performance Computing*. Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on. 978-1-4244-5011-4. (2009).

[7] Lingyuan Wang, Miaoging Huang, Vikram K. Narayana, Tarek El-Ghazawi. *Scaling Scientific Applications on Clusters of Hybrid Multicore/GPU Nodes*. ACM 978-1-4503-0698-0. (2011).

[8] Chao-Tung Yang, Chih-Lin Huang, Cheng-Fang Lin, Tzu-Chieh Chang. *Hybrid Parallel Programming on GPU Clusters*. IEEE 978-0-7695-4190-7/10 (2010).

[9] Nvidia Corporation. *NVIDIA CUDA C Programming Guide*. (2011).

[10] Nvidia Corporation. *CUDA C Best Practices Guide*. (2012).

[11] Montes de Oca Erica, De Giusti Laura, Chichizola Franco, De Giusti Armando, Naiuof Marcelo. *Utilización de un Cluster de GPU en HPC. Un caso de estudio.* XX Congreso Argentino de Ciencias de la Computación (2014).

[12] Peter Berczik, Keigo Nitadori, Shiyan Zhong, Rainer Spurzem, Tsuyoshi Hamada, Xiaowei Wang, Ingo Berentzen, Alexander Veles, Wei Ge. *High performance massively parallel direct N-body simulations on large GPU cluters.* Astronomisches Rechen-Institut, ZAH, Heidelberg, Alemania, Main Astronimical Observatory, NASU, Kyiv, Ucrania, Centre for Astronomy, Jeidelberg University, Heidelberg, Alemania. HPC –UA (2011).

[13] Andrews Gregory R. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley (2000).

[14] Tsuyoshi Hamada, Keigo Nitadori. *190 TFlops Astrophysical N-body Simulation on cluster of GPUs*. IEEE 978-1-4244-7558-2 (2010).

[15] Bruzzone Sebastian. *LFN10, LFN10-OMP y el Método de Leapfrog en el Problema de los N Cuerpos*. Instituto de Física, Departamento de Astronomía, Universidad de la República y Observatorio Astronómico los Molinos, Uruguy (2011).

[16] Pritish Jetley, Lukasz Wesolowski, Filippo Gioachin, Laxmikant V. Kalé, Thomas R. Quinn. *Scaling Hierarchical N-body Simulations on GPU clusters*. IEEE 978-1-4244-7558-2 (2010).

[17] Lasr Nyland, Mark Harris, Jan Prins. *Fast N-body simulation with CUDA.* GPU Gem 3. cap. 31, pp. 677-695. Addison-Wesley (2007).