Análisis de la escalabilidad y el consumo energético en soluciones paralelas sobre cluster de multicores y GPU para un problema con alta demanda computacional

Erica Montes de Oca[1], Laura De Giusti[1], Ismael Rodriguez[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] CONICET
{emontesdeoca,ldgiusti,degiusti,mnaiouf}@lidi.info.unlp.edu.ar

Resumen. Este trabajo realiza un estudio de la escalabilidad y el consumo energético, en el uso de un cluster de multicore y una placa de GPU con 384 cores, teniendo como caso de aplicación el problema de los N-body. Se implementaron una solución paralela en memoria compartida para CPU usando Pthread, una solución en memoria compartida para GPU usando CUDA y una solución en memoria distribuida en CPU utilizando MPI. Se presentan y analizan los resultados obtenidos, que muestran en este problema que el uso de la GPU no solo logra acelerar el cómputo sino también, reducir el consumo energético.

Palabras claves: multicore, cluster de multicores, GPU, N-body, escalabilidad, Green Computing, consumo energético.

1. Introducción

La tecnología ha permitido mejorar la calidad de vida mundial. Los avances en los procesadores en los últimos tiempos, han logrado acelerar la solución muchos problemas de la vida real. La llegada de los multicores, no sólo ha permitido disminuir los tiempos de cómputo de varias aplicaciones [1], sino que también ha logrado reducir el consumo energético de los procesadores; ya que los mismos están formados por más procesadores pero mucho más simples.

Los multicores pueden agruparse formando clusters en los que se combina la memoria compartida por los núcleos de un multicore con la memoria distribuida para la comunicación entre multicore [2], dando lugar a un esquema híbrido. Por otra parte, en las últimas décadas, una nueva plataforma de propósito específico se ha abierto paso como una alternativa para el Cómputo de Altas Prestaciones: la GPU [3] [4] [5] [6].

Sin embargo, la aceleración del cómputo trae aparejada la problemática de la gran cantidad de energía consumida, que se ha convertido en un aspecto significativo tanto en la fabricación del hardware como en las implementaciones software.

Desde una perspectiva actual, es nuestra responsabilidad proveer no solo un avance tecnológico sino un uso responsable del mismo [7]. Ya sea desde el hardware o el

software, debe pensarse en reducir gastos innecesarios en el consumo energético. En ese sentido, este trabajo presenta una comparación de soluciones al problema de la atracción gravitacional de los cuerpos celestes utilizando cluster de multicores y GPU.

El trabajo está organizado de la siguiente manera: en la Sección 2 se introduce al concepto de Green Computing; la Sección 3 plantea un breve comentario del problema de los N-body, mientras que en la Sección 4 se muestran los resultados experimentales obtenidos. La Sección 5 presenta las conclusiones y trabajos futuros.

2. Green Computing

Green Computing comprende el uso eficiente de los recursos. Su objetivo es reducir el uso de materiales peligrosos, maximizar la eficiencia de la energía durante el ciclo de vida de producción, y promover el reciclado o biodegrabilidad de los productos y desechos fabriles [8].

En muchos casos los consumidores no toman en cuenta el impacto ecológico a la hora de comprar sus computadoras, solo prestan atención a la velocidad de sus prestaciones y al precio. Sin embargo, a mayor velocidad de procesamiento se requiere mayor poder energético, trayendo el problema de la disipación del calor, que necesita más energía eléctrica para mantener al procesador en la temperatura normal de trabajo. Los diseñadores de hardware ya han planeado varias estrategias para colaborar con la reducción del consumo energético, desde la fabricación del equipo hasta su reciclado [9].

Un gran avance se ha realizado con el paso del monoprocesador a los procesadores multicore, ya que estos últimos suelen ser más simples, por lo que hacen un uso más eficiente de la energía. Las grandes compañías (Intel y AMD), han tomado conciencia sobre esta necesidad del uso eficiente de los recursos en la producción de los mismos [10] [11].

En los últimos tiempos, la eficiencia energética se ha transformado en uno de los factores influyentes en el desarrollo de aplicaciones. En el campo del Computación de Altas Prestaciones (HPC, High Performance Computing), se están realizando investigaciones orientadas no solo a disminuir la energía consumida, sino también en un sistema de gestión de energía que dada una aplicación y un plataforma HPC presente alternativas de ejecución dependientes de: el consumo energético, la potencia máxima (capacidad de la infraestructura eléctrica y el equipo de refrigeración) y el rendimiento [12]. En HPC la eficiencia energética es un factor que limita el desarrollo de aplicaciones, ya que la cantidad de energía necesaria para procesar las grandes cantidades de datos se ha convertido en un problema al cual cada vez se debe prestar más atención.

Además de la eficiencia energética, se debe tener en cuenta: la escalabilidad de los sistemas y el precio de la energía eléctrica. La gran cantidad de energía consumida reduce la escalabilidad de los sistemas, lo que los hace menos útiles a largo plazo. Por lo tanto, no solo se debe estudiar y analizar la escalabilidad con respecto al problema y la arquitectura, sino también se debe orientar la misma al consumo total final de las aplicaciones, para que por un lado no supere la cantidad de energía eléctrica que se nos suministra y por el otro, esté acorde a los gastos económicos disponibles.

3. Caso de estudio: problema de los N-body

El problema de los N-body es clásico en el cómputo científico y ha sido muy estudiado por su adaptabilidad a distintas aplicaciones del mundo real [13]. El presente trabajo está centrado en la aplicación de la fuerza de atracción gravitacional, basada en la teoría de Newton sobre su Ley de Atracción, que enuncia: "La fuerza de gravedad entre dos cuerpos es proporcional a sus masas e inversamente proporcional al cuadrado de sus distancias" [14].

El modelado del problema requiere del conocimiento de la siguiente información: la masa, la velocidad, la posición y la fuerza de atracción inicial de cada cuerpo. La Ecuación (0) es el cálculo central de todo el procesamiento, y se basa en la fuerza de magnitud de cada cuerpo [15].

$$F = (G \times m_i \times m_j) / r^2 \qquad \textbf{(0)}$$

con r = distancia, G = cte gravitacional $(6{,}67 \times 10^{11})$

El algoritmo secuencial en el que se basan las soluciones paralelas es denominado *all pair* o *simulación directa*. Todos los cuerpos que conforman el espacio del problema calculan su fuerza de atracción gravitacional con el resto. Por lo que la complejidad del problema es de $O(N^2)$ [16]. Para optimizar el acceso a la memoria caché y reducir el tiempo de ejecución, se realiza un procesamiento de los datos por bloques, siendo el tamaño de cada bloque el óptimo para la arquitectura empleada. Los algoritmos paralelos implementados para memoria compartida (en Pthread) y memoria distribuida (en MPI), también hacen uso del procesamiento por bloques para reducir los tiempos de ejecución al disminuir los fallos de caché.

En el caso del algoritmo de memoria compartida, un thread principal es el encargado de realizar la inicialización de los datos, y luego repartir a cada uno de los T-1 threads que conforman la simulación, bloques de datos del tamaño óptimo de la caché. Para la resolución del problema, se crean T-1 threads especificados por el usuario, ya que el hilo principal también realiza trabajo. Una vez que todos los threads tienen delimitado los datos a procesar, comienza el cálculo de la fuerza de atracción gravitacional para cada cuerpo de su conjunto de datos. Este cálculo se repetirá por cada paso de simulación, teniendo que esperar cada thread en una barrera de sincronización antes de comenzar un nuevo paso, para que todos los demás hilos puedan disponer de los datos actualizados del paso anterior.

Para la solución desarrollada con MPI, la implementación del algoritmo es similar con la excepción de que se trata de procesos y no de threads, y que una vez que el proceso terminó de realizar los cálculos para dicho paso, deberá comunicar los resultados obtenidos, a los demás procesos, antes de comenzar un nuevo paso de simulación.

En el algoritmo desarrollado para la GPU en CUDA, la inicialización de los datos se realiza en la CPU, y luego los mismos son comunicados por medio de PCI-E a la GPU. Una vez que la GPU tiene los datos copiados en su memoria global, cada thread que conforma la ejecución copiará los datos correspondientes para realizar los cálculos de la fuerza de atracción gravitacional a la memoria shared, para que de esta forma se optimice el acceso a la memoria. Como la memoria compartida o shared

tiene un tamaño reducido [17] [18] [4] [19], la copia de los datos desde la memoria global a la misma se hace por bloques del mismo tamaño del bloque de thread. Dicha transferencia de datos entre las memorias se lleva a cabo tantas veces hasta que todos los datos necesarios hayan sido copiados para realizar los cálculos correspondientes. Una vez realizado el procesamiento, la GPU enviará por medio de PCI-E los resultados a la CPU.

En [20] se describe con mayor detalle el resto de las ecuaciones utilizadas en las soluciones, así como una exposición más minuciosa de las soluciones paralelas implementadas.

4. Resultados experimentales

El entorno de prueba está compuesto por las siguientes arquitecturas:

- Un Cluster de Multicore con procesadores Quad Core Intel i5-2300 de 2,8 GHz, con caché de 6MB, y Sistema Operativo Debian de 64 bits.
- Una GPU Geforce TX 560TI, con 384 procesadores con una cantidad máxima de threads de 768 y 1 GB de memoria RAM.

La cantidad de cuerpos para cada simulación es variada entre 65535, 128000 y 256000 cuerpos para dos pasos de la simulación. Los datos obtenidos son el resultado de un promedio de varias ejecuciones.

Para la solución en memoria distribuida se realizó la ejecución utilizando una máquina por proceso (por lo que la comunicación de los mismos se realiza por medio de la red). Además, se llevó a cabo una ejecución de todos los procesos en una sola máquina.

En la Tabla 1 se muestran los tiempos de ejecución en segundos para dos pasos de simulación del problema, de las soluciones en CPU, tanto en memoria compartida como en memoria distribuida, utilizando 2 y 4 procesadores. En la Tabla 2 se presentan los tiempos de ejecución (en segundos) obtenidos para la solución en GPU, empleando CUDA, con un tamaño de bloque de threads de 256, que es el tamaño óptimo para la arquitectura utilizada, y dos pasos de simulación.

Tabla 1. Tiempos de ejecución en segundos para los algoritmos de MPI (para las dos formas de ejecución) y Pthreads en CPU. Con P = cantidad de procesos y T = cantidad de threads.

Cantidad de cuerpos de la simulación	Pthread (T = 2)	MPI (P = 2)	MPI en una maq. (P = 2)	Pthread (T = 4)	MPI (P = 4)	MPI en una maq. (P = 4)
65535	101,68	96,63	92,11	53,87	50,56	53,91
128000	397,39	352,39	352,93	213,94	175,66	182,08
256000	1572,81	1417,61	1427,42	810,17	717,80	728,45

Tabla 2. Tiempos de ejecución en segundos para el algoritmo de CUDA en GPU. Con T = cantidad de threads por bloque.

Cantidad de cuerpos de la simulación	CUDA (T = 256)
65535	1,04
128000	3,97
256000	15,75

Para la medición del consumo energético se empleó un osciloscopio digital, con una resolución de 8 bits con dos entradas, una para capturar la información de la tensión y la otra para la corriente. Ésta última proviene de una pinza trasductora con sensibilidad de ajuste a los siguientes valores: 1A/100mV, 1A/10mV y 1A/1mV.

La tensión se midió directamente de la línea eléctrica a la cual se encuentra conectado el cluster de multicores. La información recogida por el osciloscopio digital es enviada a otro equipo para ser analizada. La información de la corriente es obtenida del cable de entrada de las fuentes de energía de la arquitectura utilizada.

El osciloscopio digital coloca los datos calculados en buffers de 10240 muestras (10 KB). Cada buffer representa un tiempo aproximado de 40 milisegundos, lo que da un intervalo de muestreo de 40 ms / 10 KB = 3,9 μs.

En la Tabla 3 se muestran los resultados medidos en Joules totales consumidos por la aplicación para las distintas configuraciones y soluciones en la arquitectura CPU, para dos pasos de simulación con un tamaño de problema de 65535, 128000 y 256000 cuerpos.

Tabla 3. Joules totales consumidos por los algoritmos de MPI y Pthread en CPU. Con P = cantidad de procesos y T = cantidad de threads.

Cantidad de cuerpos de la simulación	Pthread (T = 2)	MPI (P = 2)	MPI en una maq. (P = 2)	Pthread (T = 4)	MPI (P = 4)	MPI en una maq. (P = 4)
65535	740,74	660,05	699,72	497,14	970,46	428,23
128000	2858,60	2415,74	2840,93	1953,98	3237,15	1798,35
256000	11290,59	9235,81	10040,86	7971,86	13123,58	6394,27

La Tabla 4 presenta el consumo total en Joules del algoritmo en GPU para 65535, 128000 y 256000 cuerpos en dos pasos de la simulación. Las mediciones de consumo energético para la arquitectura GPU se realizaron de la misma forma que para la arquitectura en CPU. La información de consumo obtenida y analizada es resultado de medir en el cable de entrada de la fuente del equipo que contiene la placa GPU.

Tabla 4. Joules totales consumidos por el algoritmo de CUDA en GPU. Con T = cantidad de threads por bloque.

Cantidad de cuerpos de la simulación	CUDA (T = 256)
65535	13,67
128000	60,94
256000	239,12

Las Figuras 1 y 2 muestran el consumo total obtenido para los algoritmos en CPU de memoria compartida y distribuida, para los distintos tamaños del problema (65535, 128000 y 256000) en dos pasos de simulación, con una configuración de 2 y 4 procesadores respectivamente. En la Figura 3, se presenta el consumo total resultante de ejecutar la aplicación en GPU, utilizando bloques de 256 threads para 65535, 128000 256000 cuerpos en dos pasos de simulación.

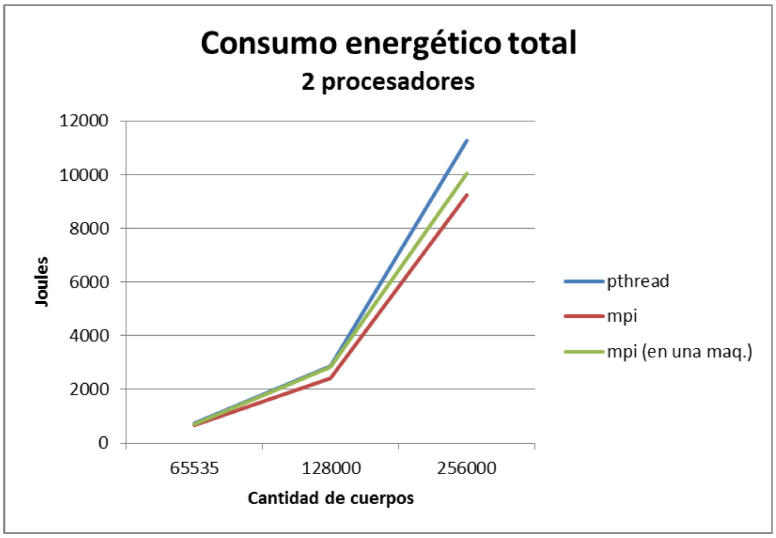

Fig. 1. Consumo total en Joules de los algoritmos MPI y Pthread con dos procesadores, para 65535, 128000 y 256000 cuerpos para dos pasos de simulación.

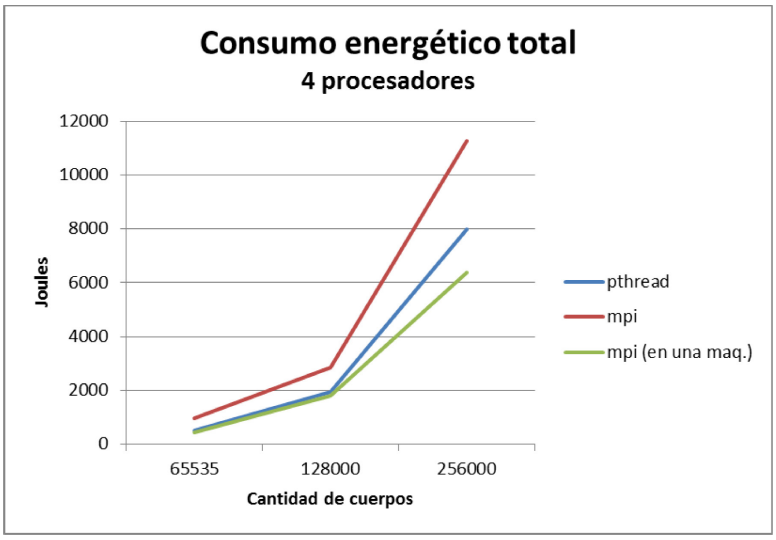

Fig. 2. Consumo total en Joules de los algoritmos MPI y Pthread con cuatro procesadores, para 65535, 128000 y 256000 cuerpos para dos pasos de simulación.

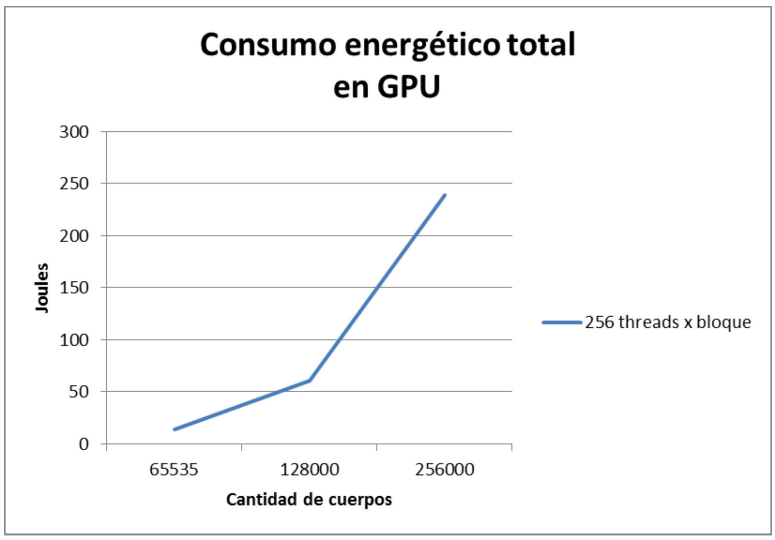

Fig. 3. Consumo total en Joules del algoritmo en CUDA para GPU con un tamaño de bloque de 256 thread, para 65535, 128000 y 256000 cuerpos para dos pasos de simulación.

A partir del entorno de prueba planteado, se obtuvieron los resultados anteriormente presentados. Se puede apreciar, que en cuanto a los tiempos de ejecución la diferencia entre las soluciones paralelas en CPU es poco importante. El tiempo de ejecución del algoritmo desarrollado en MPI para las ejecuciones con todos

los procesos en una máquina, y un proceso por máquina, es muy próximo entre sí. En cuanto a la GPU, notablemente se aprecia que el tiempo de ejecución obtenido para los distintos tamaños del problema es significativamente menor al de las soluciones paralelas desarrolladas para la arquitectura CPU.

Desde el punto de vista del consumo energético total se puede apreciar que el de la solución paralela en GPU es la que presenta un menor consumo, muy notable por la gran aceleración del cómputo obtenida. Mientras, que el consumo mayor alcanzado por la GPU se logra en el tamaño máximo del problema para las pruebas realizadas, el consumo total para el caso de las versiones paralelas en Pthread y MPI en una sola máquina, doblan dicho consumo para el tamaño inferior probado.

Para los casos de las soluciones paralelas en CPU, el aumento de procesadores aproxima el consumo energético total de las implementaciones en Pthread y MPI ejecutando todos los procesos en una sola máquina. Sin embargo, para el caso en el que cada proceso se ejecuta en una máquina distinta, el consumo se dispara al aumentar los procesadores, por razón de que se trata del consumo energético total de todas las máquinas completas.

La escalabilidad de un sistema paralelo refleja la capacidad del mismo de incrementar los recursos de procesamiento efectivamente. Como se puede observar, en los tiempos de ejecución obtenidos, el incremento de procesadores reduce el tiempo de procesamiento de la aplicación para los distintos tamaños del problema. Analizando a su vez, la escalabilidad desde el punto de vista del consumo energético total, para los casos de Pthread y MPI en una sola máquina el aumento de la arquitectura reduce el consumo por reducir el tiempo de ejecución. Mientas que para la GPU, aunque crecer el tamaño del problema implica un mayor consumo energético total, este sigue siendo muy inferior comparado con las soluciones paralelas en CPU.

5. Conclusiones y Trabajos futuros

El paralelismo busca acelerar el cálculo de los datos de las aplicaciones, ya que en algunos casos, los tiempos de ejecución obtenidos con los algoritmos secuenciales suelen ser inaceptables para los tiempos de respuesta requeridos. En particular, en el presente trabajo, se plantea como caso de estudio un problema de alta demanda computacional como N-body. La resolución del problema en su versión secuencial, obtiene tiempos de procesamiento muy elevados [20], y se han implementado soluciones paralelas, en dos arquitecturas de distinto tipo como CPU y GPU. Se ha logrado con éxito demostrar que se puede alcanzar una aceleración significativa con el uso de la GPU, para el caso de estudio.

Por otro lado, la reducción del consumo energético en los últimos años se ha convertido en un factor limitante en el uso de equipamiento tecnológico para la resolución de problemas del mundo real. Ya sea por lograr una escalabilidad de los sistemas en un futuro, como reducir los gastos económicos por el consumo de energía eléctrica, cada vez es mayor la importancia que se le está dando a la eficiencia energética de los equipos y de los algoritmos en el uso de los mismos.

Los resultados presentados, muestran la disminución del consumo energético al utilizar GPU por su alta aceleración de cómputo, con respecto a las versiones

paralelas en CPU. Para el caso de las soluciones paralelas implementadas para la CPU, se han conseguido mejores resultados en la versión de memoria distribuida en la que la ejecución se realiza en una sola máquina, comparada con la solución en memoria compartida. Sin embargo, la reducción del consumo energético total por la aplicación no es tan significativa entre estas dos soluciones comparadas con la solución en GPU.

Como trabajos futuros se plantea el uso de cluster de GPU y el estudio del modelo híbrido MPI-CUDA para su utilización, analizando los parámetros de escalabilidad y consumo. Asimismo, se busca generalizar la investigación a otras clases de aplicaciones.

Referencias

1. Silva Juliana M. N., Drummond Lúcia, y Boeres Cristina, "On Modelling Multicore Clusters", 22nd International Symposium on Computer Architecture and High Performance Computing Workshops, (2010).
2. Tinetti Fernando G. y Wolfmann Gustavo, "Parallelization Analysis on Clusters of Multicore Nodes Using Shared and Distributed Memory Parallel Computing Models", World Congress on Computer Science and Information Engineering, (2009).
3. Kirk David B. y Hwu Wen-mei W., "Programming Massively Parallel Processors: A Hands-on Approach", Morgan Kaufmann, (2010).
4. Piccoli María Fabiana, "Computación de Alto Desempeño en GPU", XV Escuela Internacional de Informática del XVII Congreso Argentino de Ciencia de la Computación, Editorial de la Universidad Nacional de La Plata, (2011).
5. Nvidia Corporation, "GPU gems", Pearson Education, (2003).
6. Song Jun Park, "An Analysis of GPU Parallel Computing", 2009 DoD Hogh Performance Computing Modernization Program Users Group Conference, publicado en IEEE, (2010).
7. Francis Kevin y Richardson Peter, Green Maturity Model for Virtualization, The Architecture Journal, págs. 9-15. (2008).
8. Schneider Electric, "Go Green, Save Green. The Benefits of Eco-Friendly Computing", (2008).
9. S.S. Verma, "GREEN COMPUTING". Departamento de fisica, SLIET, (2007).
10. Nicolaisen Nancy, "Green Computing with Intel® AtomTM Processor-Based Devices",http://software.intel.com/en-us/articles/green-computing-with-intel-tomprocessor-based-devices/, (2010).
11. amd.com/public, "Meeting the challenges of the future with innovative solutions for public sector IT needs", (2011).
12. J. Balladini, F. Uribe, R. Suppi, D. Rexachs, E. Luque. "Factores influyentes en el consumo energético de los Sistemas de Cómputo de Altas Prestaciones basado en CPUs y GPUs". Facultad de Informática, Universidad Nacional del Comahue, Argentina, y Departamento de Arquitectura de Computadores y Sistemas Operativos, Universidad Autónoma de Barcelona, España. XVII Congreso Argentino de Ciencias de la Computación, (2011).
13. Francisco Chinchilla, Todd Gamblin, Morten Sommervoll, Jan F. Prins , "Parallel N-Body Simulation using GPUs ", Department of Computer Science , University of North Carolina at Chapel Hill , http://gamma.cs.unc.edu/GPGP , Technical Report TR04-032, (2004).

14. Bruzzone Sebastian, “LFN10, LFN10-OMP y el Método de Leapfrog en el Problema de N Cuerpos”, Instituto de Física, Departamento de Astronomia, Universidad de la República y Observatorio Astronómico los Molinos, Uruguay, (2011).
15. Andrews Gregory R., “Foundations of Multithreaded, Parallel, and Distributed Programming”, Addison-Wesley, (2000).
16. Jeroen Bédorf, “High Performance Direct Gravitational N -body Simulations on Graphics Processing Units”, Universiteit van Amsterdam, (2007).
17. Nvidia, “NVIDIA CUDA C Programming Guide”, (2011).
18. Nvidia, “CUDA C BEST PRACTICES GUIDE”, (2012).
19. Perez Cristian y Piccoli M. Fabiana, “Estimación de los parámetros de rendimiento de una GPU”, Mecánica Computacional Vol XXIX, págs. 3155-3167, (2010).
20. Erica Montes de Oca, Laura De Giusti, Armando De Giusti, Marcelo Naiouf, “Comparación del uso de GPU y Cluster de multicore en problemas de alta demanda computacional”, XVIII Congreso Argentino de Ciencias de la Computación, CACIC 2012, pág. 267-275, (2012).