Análisis de consumo energético en Cluster de GPU y MultiGPU en un problema de alta demanda computacional

Trabajo Final Integrador para la carrera de

Especialización en Cómputo de Altas Prestaciones y Tecnología Grid

Montes de Oca Erica Soledad

emontesdeoca@lidi.info.unlp.edu.ar

Dirección: Dr. Naiouf Marcelo

Codirección: Dra. De Giusti Laura Cristina

Índice

Objetivos

El objetivo general es realizar investigación y desarrollo en algoritmos paralelos sobre arquitecturas basadas en GPU (Unidad de Procesamiento Gráfico o *Graphic Processing Unit*). En particular, estudiar y desarrollar un algoritmo de Alta Demanda Computacional, como lo es el problema de los N Cuerpos, en las plataformas Cluster de GPU y MultiGPU.

Se busca comparar performance y consumo energético, a fin de obtener conclusiones respecto de la conveniencia de las soluciones en relación fundamentalmente al tiempo de ejecución y consumo energético. Por esto, se estudiarán conceptos básicos de la física, tales como la corriente eléctrica, unidades de medida y equipos de medición. Además, se investigarán los conceptos de Green Computing y eficiencia energética.

Introducción

Los grandes avances tecnológicos en el ámbito de la información y de la comunicación, producidos en las últimas décadas, han generado profundos cambios que inciden en nuestra vida cotidiana. La generalización en el uso de las computadoras, la proliferación de redes informáticas y el desarrollo de las nuevas tecnologías de la información y comunicación (denominadas TICS), cambiaron la manera de comunicarnos y transformaron la economía y la cultura para siempre, originando un sistema económico y social en el que la generación, procesamiento y distribución del conocimiento y la información son la principal fuente de la productividad, poder y prosperidad. La informática y la industria en general no han ahorrado esfuerzos para desarrollarse rápidamente, pero en la mayoría de los casos, a costa del deterioro ambiental. Sin embargo, las TICS pueden ser un aliado en la lucha contra el cambio climático a través de procesos denominados Tecnologías Verdes, que se inscriben en el concepto de economía verde como contexto del desarrollo sustentable [PM15].

En la actualidad, el procesamiento de grandes cantidades de datos, el alcance de la máxima cantidad de transistores por microchip, y la demanda de la reducción de los tiempos de repuesta de las aplicaciones, han hecho que el procesamiento paralelo sea, en muchos casos, la única solución para tratar algunos problemas computables [GGKK03] [Amd11].

El procesamiento paralelo se define como la ejecución de la solución de un problema de manera concurrente o simultánea sobre diferentes componentes físicos. La ejecución del algoritmo se realiza de manera coordinada y cooperante, buscando realizar el procesamiento de manera eficiente en menor tiempo [HB84] [DGT98].

El tiempo de ejecución de un algoritmo paralelo se ve incrementado a su vez por las que se denominan fuentes de overhead. Pueden identificarse básicamente tres:

- Creación de los procesos y scheduling;
- Comunicación;
- Sincronización.

Estos tiempos “no útiles” son generalmente inevitables, aunque sí es deseable reducirlos [And00]. En un sistema paralelo, el algoritmo se encuentra muy fuertemente ligado a la arquitectura paralela subyacente, por lo tanto, las optimizaciones y el rendimiento de la aplicación están íntimamente relacionadas a la arquitectura para la cual se desarrolla.

Hoy en día, la GPU se presenta como una opción de aceleramiento de cómputo para problemas con paralelismo de datos. La reducción de los tiempos de procesamiento en estas arquitecturas, sumado al menor costo y a la facilidad de adaptar los problemas a las mismas (con el empleo de plataformas de programación como CUDA u OpenCL), han hecho de las GPU una opción altamente viable a la hora de elegir una arquitectura paralela [MDDN12] [Mdo13].

Sin embargo, la aceleración del cómputo trae aparejada una problemática: la cantidad de energía consumida [FG08]. Hoy por hoy, el uso de las computadoras es una necesidad, pero su uso demanda ciertos grados de responsabilidad de los usuarios para reducir el impacto ambiental. El uso de la computadora, sus accesorios y sus recursos está catalogado en la actualidad como uno de los responsables del calentamiento global. Incrementar la performance mientras se reduce el consumo energético y la emisión de dióxido de carbono (CO_2), es el objetivo principal. El diseño de todos los sistemas de cómputo, desde dispositivos móviles hasta los sistemas de cómputo de altas prestaciones, está siendo impactado por el consumo energético.

En sus comienzos, los sistemas de cómputo de altas prestaciones tenían sólo como objetivo incrementar la velocidad de procesamiento. Al dueño de la supercomputadora, sólo le importaba la relación precio/prestación [BMRD+++16]. Las supercomputadoras se volvieron tan grandes que llegaban a consumir tanta electricidad como una ciudad. En 2002, el Dr. Eric Schmidt, CEO de Google, dijo "*lo que más importa a los diseñadores de computadoras de Google no es la velocidad sino el consumo energético, porque los centros de datos pueden consumir tanta electricidad como una ciudad*" [ML02].

Hasta el año 2005, no sólo se incrementó el número de transistores, sino que también el consumo energético. A partir de este año, se siguió incrementado la cantidad de transistores de los chips de los procesadores, pero el consumo energético se mantuvo, debido al nacimiento de los procesadores multicore que trabajan a una menor frecuencia de reloj.

Algunas estrategias que reducen el consumo energético son [BMRD+++16]:

- Explotar el paralelismo de los múltiples cores de los procesadores actuales;
- Nueva tecnología de los transistores y reordenamiento de puertas lógicas;
- Rediseño de buses y redes de interconexión;
- Optimización de memoria;
- Arquitecturas adaptables e hibernación de recursos;

- Integración on-chip;
- Uso de GPU;
- Rediseño de algoritmos de aplicaciones;
- Planificación de tareas y asignación de tareas a recursos hardware;
- Escalado dinámico de frecuencia y tensión, entre otras.

Los últimos años, se ha comenzado a tener conciencia por parte de los programadores de software, en desarrollar algoritmos más eficientes que reducen el número de recursos necesarios para completar una función, y, por lo tanto, reducen al mismo tiempo el consumo energético. La virtualización ha emergido como una solución en la que múltiples procesos se ejecutan sobre un mismo conjunto de recursos físicos de hardware, lo que ahorra espacio y el consumo de refrigeración del equipamiento. El consumo de energía de los centros de datos tiene un enorme impacto en entornos de procesamiento de datos. La tecnología de máquina virtual se ha aplicado ampliamente en entornos de centros de datos, debido a sus características de fiabilidad, flexibilidad, y la facilidad de gestión.

Desde el punto de vista económico, reducir el consumo eléctrico reduce los costos energéticos, ya que el costo de la energía se incrementa gradualmente año tras año. Para los grandes centros de datos los costos en energía eléctrica insumen grandes partes de sus ganancias. Consideraciones que van desde el diseño del sistema eléctrico hasta la utilización de sistemas de almacenamiento de datos eficientemente son algunos de los pasos que una organización de esta envergadura debe tener en cuenta.

Otro pilar del Green Computing es el reciclaje. Cada vez más, los consumidores descartan un gran número de computadoras, monitores y otros equipos electrónicos, dos o tres años después su compra, y la mayor parte termina en los rellenos sanitarios, contaminando la tierra y el agua. El aumento del número de las computadoras utilizadas, junto con su reemplazo frecuente, hacen que el impacto medioambiental de las TICs cobre mayor importancia. En consecuencia, cada vez hay más presión sobre esta industria, y sobre los individuos, para que sea el medio ambiente amable en todo su ciclo de vida [Val14] [ANAA+17].

Actualmente las empresas tienen más disposición para incluir en sus presupuestos, inversiones en las iniciativas Green Computing, para conseguir una reducción de costos significativos, y rendimiento medioambiental óptimo. Varias empresas han implementado algunas prácticas que están ayudando a mejorar la situación ambiental global, a incrementar la productividad, e incluso a ahorrar dinero, logrando una disminución en el consumo de recursos energéticos e insumos. [Val14].

La Ciencia de la Computación juega un rol crucial en la investigación y la promoción de las técnicas de Green Computing. Utilizar eficientemente un sistema de cómputo actualmente suma un aspecto significativo, ya sea desde la fabricación del hardware como

en la implementación del software; no sólo se debe promover un avance tecnológico sino un uso responsable del mismo, lo que implica que desde cualquiera de las dos partes deben minimizarse los gastos innecesarios en el consumo energético [MDCD+14] [TW09] [DACC++16].

En este sentido, este trabajo presenta un estudio y análisis, del consumo energético para las soluciones implementados en arquitectura de Cluster de GPU y MultiGPU para el problema de alta demanda computacional N Cuerpos.

Capítulo 1

Conceptos de Arquitectura GPU

1.1 Características de la Arquitectura GPU

En sus comienzos, las GPUs (Unidades de Procesamiento Gráfico) eran utilizadas sólo como un coprocesador de la CPU para la generación de imágenes interpretadas de un plano tridimensional a un plano bidimensional [Lue08].

La industria de los video juegos ocasionó que este hardware, acelerador de gráficos, evolucionara hasta el punto de ser utilizado en ámbitos académicos e industriales como una forma de acelerar cómputo. De este modo nació el concepto de GPGPU (Computación de Procesamiento General en Unidades de Procesamiento Gráfico).

Al principio, la programación era compleja, y se utilizaban lenguajes tales como GLSL, Cg, o HLSL, que requerían un alto grado de conocimiento sobre renderizado de imágenes por lo que el desarrollo de aplicaciones para este tipo de arquitectura paralela consumía mucho tiempo y esfuerzo.

A partir de 2004, los esfuerzos académicos centrados en la explotación, la investigación y el desarrollo en GPU lograron que emerja con ímpetu, a tal punto que se deja de discutir sobre su rendimiento. En adelante, las compañías fabricantes de GPU, comenzaron a modificar sus procesadores de streams para que no sólo ejecutaran operaciones gráficas sino también cómputo de propósito general.

Nvidia resolvió los problemas de la complejidad de programación de su hardware lanzando CUDA (Compute Unified Device Architecture) en 2007, y AMD/ATI hizo lo propio con Book+ (actualmente sin soporte) y más tarde con OpenCL [FX10] [CGef18].

GPGPU inicialmente fue resistido por muchos, ya que siendo procesadores con un propósito específico se ejecutaban en ellos aplicaciones totalmente diferentes para la que habían sido diseñados. Sin embargo, la investigación y el desarrollo en esta arquitectura se ha abierto paso, y puede afirmarse que con los resultados de performance alcanzados el desarrollo de estas seguirá creciendo [Nvi03] [KH10] [Pic11] [SKHZ+++16].

La GPU es una arquitectura manycore cuyo enfoque se basa en la ejecución de aplicaciones paralelas. Por eso, nacieron desde su inicio con una gran cantidad de cores, los cuales se doblan en número aproximadamente cada nueva generación [Ngpu18].

Han evolucionado incrementando la cantidad de cálculo en punto flotante por segundo, ya que su diseño fue pensado desde el principio para realizar cálculo masivo en punto flotante, lo que la convierte en una arquitectura paralela desde su nacimiento.

La GPU, al igual que la CPU, es un procesador segmentado; las características particulares son:

- No fluyen las instrucciones, sino los datos.
- Los operadores son unarios, lo que evita las dependencias y maximiza el paralelismo.
- Se dispone de un menor número de etapas dada la baja frecuencia.
- Se fomenta el procesamiento vectorial y superescalar, buscando aprovechar el paralelismo de datos.

La industria del video juego ejerce una gran presión económica sobre los vendedores de GPU haciendo que aumenten sus ganancias, generando arquitecturas que cada vez ejecutan mayor cantidad de operaciones en punto flotante por segundo. Consiguen un mayor rendimiento haciendo que el hardware aproveche la gran cantidad de threads que se puedan crear para solapar tiempos de espera de cómputo, es decir, mientras algunos threads están en espera se oculta la latencia de acceso a memoria ejecutando otros.

La caché que utiliza para disminuir los tiempos de acceso a memoria es pequeña, lo que permite que haya más lugar dentro del chip dedicado al cálculo de punto flotante. Para compensar la cantidad limitada de memoria caché, la GPU cuenta con una jerarquía de memoria expuesta al programador. La latencia de acceso a la memoria global (off chip) es ocultada con el uso de la memoria shared (on chip).

La GPU incrementa la velocidad de cómputo a través de la ejecución de un grupo de instrucciones de un conjunto de threads planificados por hardware, y con context switching muy eficiente teniendo disponibilidad de una gran cantidad de threads activos que superan el número de recursos de cómputo [Par10].

Por estos y otros motivos la GPU se ha convertido en una de las arquitecturas paralelas más vendidas. Su costo económico las hace muy atractivas para el desarrollo de aplicaciones con gran requerimiento de cálculo [KH10].

1.2 Plataforma CUDA

La arquitectura de una GPU-Nvidia CUDA está organizada en Streams Multiprocessors (SMs), los cuales tienen un determinado número de Streams Processors (SPs). Los SPs comparten la lógica de control y la caché de instrucciones.

Nvidia [ND10] varia la cantidad de SMs en cada nueva generación de GPUs logrando que las mismas sean escalables en performance, además de variar el número de DRAM para lograr escalar el ancho de banda de memoria y la capacidad. Cada SM provee los suficientes threads, cores y memoria shared para ejecutar uno o más bloques de threads CUDA. Los que realizan verdaderamente la ejecución son los SPs, que ejecutan múltiples threads concurrentemente. Los threads se organizan en bloques de hilos, y existen uno o más bloques ejecutándose concurrentemente en SM. La organización de los threads puede verse por nivel o jerárquicamente (Fig. 1.1) de la siguiente manera:

- Grid: nivel superior, está conformado por bloques de threads.
- Bloque: nivel medio, está conformado por threads.
- Threads: nivel inferior, son los que realizan el trabajo.

Los bloques de hilos en un Grid pueden ser de una, dos o tres dimensiones. A su vez, cada bloque está dividido en threads de una o dos dimensiones. En un bloque de threads, los hilos están organizados en warps, los cuales, por lo general, agrupan 32 hilos. Todos los hilos de un warp son planificados juntos durante la ejecución, y de manera independiente [Lue08].

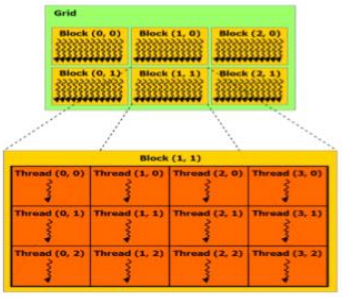

Fig. 1.1 Jerarquía de thread de una GPU [Nvi11]

Comentado [M1]: OJO CON LAS FIGURAS. Falta la 1.1 (mencionada arriba), en ese caso esta sería 1.2 sino no...

Comentado [M2]: Si esta figura la sacaste de algún lado deberías citar de donde.

Los hilos de un bloque pueden comunicarse con otro hilo a través de la memoria compartida. Deben ser ejecutados en el mismo multiprocesador. De esta forma, los hilos pueden sincronizarse entre ellos. Cada thread tiene un identificador que le permite diferenciarse de los demás, direccionar memoria y tomar decisiones de control. La cantidad de bloques de hilos por multiprocesador depende de la memoria compartida y los registros alocados para cada bloque. Por lo cual, cuanta más memoria compartida y más registros son asignados por cada bloque, menor es la cantidad de bloques de hilos por multiprocesador. La creación de los threads, así como su ejecución y terminación, es automática y manejada por la GPU. El usuario solo debe especificar el número de hilos por bloque y el número de bloques por Grid [Hwu18].

1.3 Modelo GPU-CUDA

La GPU [Nvi03] no es un procesador serial sino un procesador de stream. En el modelo de programación de stream, todos los datos son representados como stream, definiéndolos como un conjunto de datos de algún tipo de dato. El tipo de dato puede ser simple (streams de enteros o punto flotante) o complejos (streams de puntos o triángulos o matrices de transformación). Los streams pueden tener cualquier longitud, pero cuanto mayor es la longitud de este más eficientes son las operaciones que se realizan sobre ellos. Los procesadores de stream trabajan de manera diferente a los seriales, ya que ejecutan una función (fragmento de programa) sobre un conjunto de registros de entrada (fragmentos), produciendo un conjunto de registros de salida (pixeles) en paralelo. Los procesadores de stream se refieren a esta función como kernel y al conjunto de registros como stream. Los datos fluyen en el procesador, y este opera sobre los mismos a través de la función kernel, enviando los resultados a la memoria.

El kernel (Fig. 1.2) opera sobre entidades stream de elementos no sobre elementos individuales. Además de evaluar una función sobre cada elemento del stream de entrada, el kernel, también realiza operaciones tales como expansión (más de un elemento de salida es producido por un elemento de entrada), reducción (más de un elemento de entrada es combinado para producir un elemento de salida), o filtrado (un subconjunto de los elementos de entrada se convierte en elementos de salida). Se dice que se tiene un uniform streaming cuando se aplica algún kernel a todos los elementos del stream.

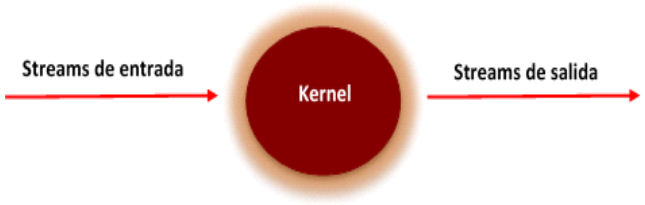

Fig. 1.2 Modelo de programación GPU (tomado de [Nvi03])

Cada elemento es procesado independientemente, es decir, sin existir ninguna dependencia entre los elementos. Esto le permite ejecutar el programa en paralelo sin la necesidad de expresar las unidades paralelas o ninguna instrucción paralela por parte del programador. Esta restricción de independencia entre los datos tiene dos ventajas: primero, los datos requeridos por el kernel para la ejecución son completamente conocidos cuando el kernel es escrito (o compilado). El kernel puede ser muy eficiente con los elementos de entrada y los datos intermedios computados que son almacenados localmente o son cuidadosamente controlados con referencias globales. Segundo, la independencia de los elementos permite que el kernel sea una función de cómputo serial sobre un hardware de datos paralelos. En el modelo de programación de streams, las aplicaciones son construidas como una tubería de múltiples kernel juntos. Por ejemplo, en un pipeline gráfico se podría escribir un kernel que compute vértices, un kernel en assembly de triángulos, un kernel clipping, y uno que conecte la salida de un kernel al otro.

Este modelo hace que la comunicación entre los kernels sea explícita, tomando las ventajas de la localidad de los datos entre los kernels en el pipeline gráfico.

El modelo de programación de streams, realiza cómputo de manera eficiente por las siguientes razones:

• Las entidades streams pueden ser procesadas paralelamente gracias al paralelismo hardware de datos. Los streams con una gran cantidad de elementos utilizan de manera más eficiente el paralelismo a nivel de datos.

• Las aplicaciones son construidas a partir de múltiples kernels, los cuales forman un pipeline y procesan en paralelo haciendo uso del paralelismo a nivel de tarea. La combinación del paralelismo a nivel de datos y tareas permite que la GPU use docenas de unidades funcionales simultáneamente.

1.4 Programación con CUDA

El desafío actual, se encuentra en desarrollar aplicaciones de software paralelas que se adapten de manera transparente a la cantidad creciente de núcleos. El modelo de programación paralela CUDA fue diseñado para cumplir este desafío, y al mismo tiempo mantener una curva de aprendizaje baja para los programadores familiarizados con lenguajes de programación estándar como C [Nvi03] [Lue08] [CCCT10].

CUDA [ND10] comprende el hardware y el software para el cómputo paralelo sobre las GPUs de Nvidia para ejecutar programas con C, C++, Fortran, OpenCL, DirectCompute y otros lenguajes. CUDA preserva el modelo de los lenguajes y los extiende agregándoles una mínima lista de abstracciones para expresar paralelismo. Por lo tanto, el programador se enfoca en las características importantes del paralelismo utilizando un lenguaje familiar [Nvi18]. El entorno de desarrollo de CUDA crece gracias al aporte de compañías que ofrecen servicios y soluciones para las GPUs de NVIDIA. Las tarjetas gráficas NVIDIA son una nueva tecnología con una arquitectura de computación extremadamente multithreaded.

El lenguaje CUDA es una extensión del lenguaje C [Par10] que permite al programador definir funciones C (denominadas kernels), las cuales al ser llamadas se ejecutan paralelizadas n veces en n threads diferentes. Esto contrasta con una función C clásica, que se ejecuta en uno solo thread cada vez que es llamada.

Nvidia introduce el concepto de thread CUDA, los cuales son independientes entre sí y capaces de ejecutar en paralelo, por lo que pueden ser vistos como unidades de paralelismo [Par10]. La computación intensiva o partes del problema con paralelismo de datos son implementadas a través de kernels o núcleos CUDA que son ejecutados por todos los threads que componen un bloque. Múltiples bloques pueden ser ejecutados en un mismos Procesador de Streams (SM), pero solo un bloque puede ser ejecutado a través de los diferentes SMs [FX10].

El núcleo CUDA encierra tres abstracciones clave: la jerarquía de grupos de threads, la memoria compartida y las barreras de sincronización (que el programador las ve como simples extensiones del lenguaje). Los problemas pueden dividirse en subproblemas que se resuelven independientemente en paralelo en bloques de threads y cada sub-problema se resuelve también en paralelo de manera cooperativa entre los threads asignados. La escalabilidad es directa ya que el mismo código compilado puede correr en GPUs con diferente número de núcleos, beneficiándose del administrador o thread scheduler.

La ejecución paralela está expresada en la función kernel que se ejecuta en la GPU o device [Pic11] [Nvi10] [Nvi12]. El código de la función kernel se ejecuta en un solo hilo, y cada hilo ejecuta esta misma función. Por lo cual, se trata de una arquitectura de tipo SIMD (Single Instruction Multiple Data). La función kernel puede invocar código secuencial para ser ejecutado en la CPU. El kernel debe ser configurado con el número de hilos en cada bloque y el número de hilos por Grid. Para declarar un Grid y un bloque

de hilos en CUDA se utiliza un tipo de dato predefinido dim3, un vector de enteros que especifica las dimensiones de la Grid y el bloque de hilos. En la función kernel el llamado a las variables grid y bloque son escritas de la siguiente manera: <<< grid, block >>>. A partir de esta invocación el grid y el bloque de hilos es creado dinámicamente. Los threads son planificados a nivel hardware. La función kernel retorna siempre void, y con el __global__ se indica que dicha función se ejecuta en la GPU.

El paradigma CUDA [CUDA18] provee variables construidas cuyas estructuras son muy eficientes, permitiendo acceder al identificador de un bloque de threads con la variable blockIdx que puede tomar un valor desde cero a la dimensión del grid menos uno. Para acceder a la dimensión del bloque se utiliza la variable blockDim, y gridDim para la dimensión de un grid. Cada hilo individualmente es identificado por la variable threadIdx, la cual puede tomar un valor entre cero y la dimensión del bloque menos uno. WarpSize especifica el tamaño de warp. Todas estas variables son definidas en la función kernel.

CUDA provee una función que sincroniza a los hilos mediante una barrera: __syncthreads() (comunicación intra-SM). Sin embargo, no existe una forma de comunicación explícita entre los threads de diferentes bloques (comunicación inter-SM) debido a la carencia de soporte de comunicación entre los SMs. Por este motivo, dicha comunicación ocurre a través de la memoria global por lo que son necesarias barreras de sincronización implementadas en el host o CPU para la terminación del kernel en ejecución y el lanzamiento de uno nuevo [FX10]. Como los threads son planificados por hardware, está función está implementada en hardware. Los threads esperarán en el punto de sincronización hasta que todos hayan llegado a dicho punto. La sincronización entre threads sólo es posible a nivel de bloque. Como el kernel se ejecuta en el device, la memoria debe ser alocada en el device antes de que la función kernel sea invocada y si el kernel utiliza algún dato, este debe ser copiado desde la memoria del host a la memoria del device. La memoria del device puede ser alocada como una memoria lineal o un arreglo CUDA. La API CUDA provee funciones para alocar y desalocar memoria en el device en tiempo de ejecución: cudaMalloc(), cudaFree(), etc. Una vez ejecutada la función kernel, los datos de la memoria del device deben ser copiados a la memoria del host. Algunas funciones provistas para ello son: cudaMemCpyToSymbol(), cudaMemCpyFromSymbol(), cudaMemCpy(), etc.

La estructura básica de un programa en CUDA consta de los siguientes pasos:

- Alocar memoria en el device y en el host por separado.
- Copiar los datos desde el host al device a través de las funciones provistas por la API.
- Ejecutar la función kernel en paralelo por cada hilo.
- Copiar los datos resultantes desde el device al host con las funciones provistas por la API.

1.5 Cluster de GPU

Algunas de las computadoras más rápidas del mundo son Cluster de computadoras. En lista más reciente del top 500 de las computadoras más rápidas del mundo la gran mayoría de las supercomputadoras cuentan con GPUs Nvidia [NDev18] [Top18].

Un cluster es la combinación de equipos independientes que conforman un sistema unificado [Thi05]. Cada nodo de un Cluster cuenta con hardware y software propio. Por esto, un Cluster puede estar conformado por nodos iguales o diferentes, lo que conformaría un Cluster Homogéneo o Heterogéneo, respectivamente.

En el caso particular de un Cluster GPU (como se observa en la Fig. 1.3), se hace referencia a una arquitectura heterogénea en la cual, por un lado, se tiene un subsistema conformado por varios cores CPUs y su sistema de memoria y E/S, mientras, por otro lado, se encuentra el subsistema GPU con sus memorias on y off chip.

Estos dos sistemas se comunican a través de PCI-E con una velocidad de ancho de banda baja comparada con las velocidades de comunicación de los sistemas de memoria. Por lo que, el cuello de botella es la comunicación que existe entre dichos subsistemas [MDCD+14] [MDCD+16] [Nvi11].

Si bien el uso de un Cluster de GPU permite escalar el problema, la aceleración obtenida con un Cluster de GPU es dependiente de la aplicación. En problemas en los que la comunicación de los datos no logra solaparse con el procesamiento de los mismo, es conveniente utilizar una GPU, en lugar de un Cluster, repitiendo varias instancias del problema [PSD13].

Por lo tanto, el uso de un Cluster de GPU requiere disminuir la cantidad de datos a comunicar entre la CPU y la GPU, en aplicaciones con gran cantidad de datos a procesar.

Comentado [M3]: AGREGARIA UNA FIGURA QUE TENGA UN ESQUEMA DE CLUSTER DE GPU Y UNO DE MULTIGPU Y LO REFERIRÍA EN ESTA SECCION Y EN LA SIGUIENTE.

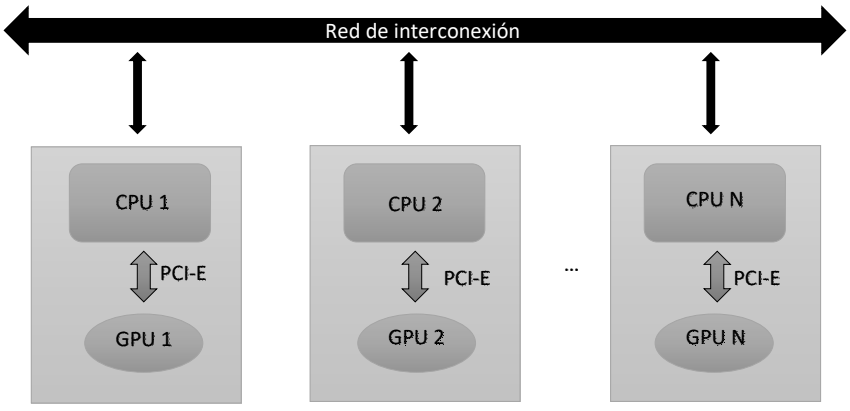

Fig. 1.3 Cluster de GPU

1.6 MultiGPU

La arquitectura MultiGPU (Fig. 1.4) se refiere al uso de una o más tarjetas gráficas en una misma PC. En este caso las GPUs son fácilmente programables a través de CUDA utilizando SLI (Scalable Link Interface) provista por Nvidia. En el caso de los Cluster de GPU, es necesario utilizar MPI+CUDA [Sha17].

El uso tanto de un Cluster de GPU o como de MultiGPU, permiten resolver problemas de alta demanda computacional a gran escala, multiplicando la potencia de la GPU al combinarlas en cualquiera de las dos formas [MDCD+16] [NTs18].

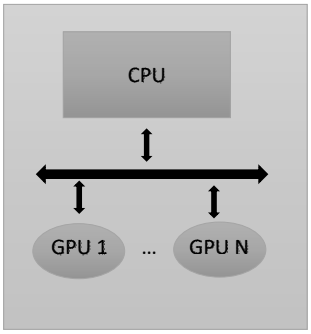

Fig. 1.4 MultiGPU

Capítulo 2

Problema de Alta Demanda Computacional: N Cuerpos

2.1 Descripción del problema

El problema de los N Cuerpos (N-Body) es clásico en el campo científico y ha sido muy estudiado por su adaptabilidad a distintas aplicaciones del mundo real [CGSP04]. Carece de una solución analítica, por lo que lo convierte en uno de los primeros problemas que se buscó resolver mediante una computadora [Bru11].

Esencialmente, el problema consiste en simular el comportamiento (en cuanto al movimiento) de N cuerpos que componen un espacio de trabajo. En particular, el presente trabajo se orienta al campo de la astrofísica, que busca calcular la fuerza de atracción gravitacional de los cuerpos en el espacio [TK10].

Newton descubrió que la gravitación es un fenómeno universal que no se restringe a nuestro planeta. Conociendo tres posiciones de un cuerpo astronómico sobre su órbita, es posible obtener la ecuación de su movimiento. A partir de los datos que le facilitó el astrónomo John Flamsteed, Newton fue capaz de obtener mediante geometría analítica una ecuación para predecir el movimiento de un planeta y determinar sus propiedades orbitales: posición, diámetro orbital, periodo y velocidad orbital. Independientemente de este hecho, Newton y otros físicos pronto descubrieron en el curso de unos pocos años, que aquellas ecuaciones del movimiento no pronosticaban en forma totalmente correcta algunas órbitas. Newton comprendió que las fuerzas gravitatorias mutuas entre todos los planetas afectaban al conjunto de sus propias órbitas [DS01].

Este descubrimiento fue directamente al centro de la cuestión respecto al significado físico exacto del problema de los n-cuerpos: como Newton advirtió, no es suficiente con especificar la posición inicial y la velocidad, o tampoco tres posiciones orbitales, para determinar con certeza la órbita de un planeta: *las fuerzas gravitatorias interactivas tienen que ser conocidas también*.

Así llegaron el interés y las primeras reflexiones sobre el "problema de los n-cuerpos" a comienzos del siglo XVII. Estas fuerzas atractivas gravitatorias se ajustan a las leyes del movimiento de Newton y a su *Ley de la Gravitación Universal*, pero la complejidad de la interacción entre "n-cuerpos" hizo históricamente intratable la obtención de cualquier solución exacta. Irónicamente, esta evidencia dirigió muchos esfuerzos al hallazgo de aproximaciones incorrectas.

Inicialmente, el problema de los n-cuerpos no fue planteado correctamente *porque no se incluía el efecto de las fuerzas interactivas gravitatorias.*

Antes de Newton, nadie había sospechado que la gravitación es un fenómeno inherente a todos los cuerpos del Universo. Muy por el contrario, durante la Edad Media y aún hasta tiempos de Newton, se aceptaba el dogma de que los fenómenos terrestres y los fenómenos celestes son de naturaleza completamente distinta. La gravitación se interpretaba como una tendencia de los cuerpos a ocupar su "lugar natural", que es el centro de la Tierra. Es justo mencionar que, antes de Newton, el intento más serio que hubo para explicar el movimiento de los planetas se debe al científico inglés Robert Hooke, contemporáneo de Newton. En 1674, Hooke ya había escrito:

...todos los cuerpos celestes ejercen una atracción o poder gravitacional hacia sus centros, por lo que atraen, no sólo, sus propias partes evitando que se escapen de ellos, como vemos que lo hace la Tierra, sino también atraen todos los cuerpos celestes que se encuentran dentro de sus esferas de actividad.

Sin esa atracción, prosigue Hooke:

...los cuerpos celestes se moverían en línea recta, pero ese poder gravitacional modifica sus trayectorias y los fuerza a moverse en círculos, elipses o alguna otra curva.

Todos los cuerpos en el Universo se atraen entre sí gracias a la fuerza de atracción gravitacional. Newton descubrió que la fuerza de atracción entre dos cuerpos es proporcional a sus masas e inversamente proporcional al cuadrado de la distancia que los separa. Concluyó a través de su 3ª Ley que "según esta Ley, todos los cuerpos tienen que atraer cada cual a los otros." Esta última declaración, que implica la existencia de fuerzas interactivas gravitatorias, es clave.

Cada cuerpo cuenta con una masa, una posición inicial y una velocidad. La gravedad hace que los cuerpos se aceleren y se muevan, provocando que los cuerpos se atraigan unos con otro. Este tipo de fuerza es denominada de acción a distancia.

El peso y la masa son cantidades íntimamente relacionadas entre sí. Cualquier cosa que tiene peso y ocupa un lugar en el espacio tiene masa. La masa es una cantidad de medida más importante que el peso. La masa de un cuerpo no puede cambiar. Sin embargo, el peso depende de la atracción de la gravedad. Si el cuerpo fuera movido a un punto en el espacio donde no existiera fuerza de gravedad, el peso de este sería cero, pero tendría la misma masa.

La relación entre masa y peso se puede deducir del Principio de Masa de Newton3 que dice que Fuerza es el producto de la masa por la aceleración:

$$F = m * a \tag{2.1}$$

donde F es la cantidad de fuerza
m es la masa
a es la aceleración

La aceleración debido a la gravedad es el porcentaje de variación de cambio de la velocidad del cuerpo. Se puede calcular dicha aceleración como:

$$a = F / m \tag{2.2}$$

Durante un pequeño intervalo de tiempo, llamémosle dt, la aceleración del cuerpo i denominada a_i es aproximadamente constante, entonces el cambio de velocidad de los cuerpos será:

$$dv_i = a_i \,.\, \mathrm{dt} \tag{2.3}$$

El cambio de la posición del cuerpo puede ser calculada como la integral de la velocidad y la aceleración en el intervalo de tiempo dt, que es aproximadamente:

$$dp_i = v_i * dt + \frac{a_i}{2} * dt^2 = \left(v_i + \frac{dv_i}{2}\right) * dt \tag{2.4}$$

También, puede calcularse la magnitud de la fuerza de gravedad entre dos cuerpos i y j a través de la siguiente fórmula expresada por Newton:

$$F = \frac{G * m_i * m_j}{r^2} \quad (2.5)$$

Siendo

m la masa,

r la distancia,

G constante gravitacional (cuyo valor es 6,67 x 10-11)

Si los cuerpos se hallan en un plano espacial de dos dimensiones, las posiciones de los cuerpos pueden ser representadas como puntos coordenados en el plano tal como: (p_i. x, p_i. y) y (p_j. x, p_j. y). El plano espacial donde se hallan los cuerpos se lo denomina espacio euclídeo. La ecuación que permite medir la distancia euclídea entre dos puntos es la siguiente:

$$\sqrt[2]{(p_i.x - p_j.x)^2 + (p_i.y - p_j.y)^2} \quad (2.6)$$

Si la distancia de dos cuerpos es pequeña significa que los mismos están muy cerca de colisionar.

Puede haber dos tipos de singularidades en el problema de los n-cuerpos:

- Colisiones de dos o más cuerpos, pero en las que **q**(*t*) (posiciones de los cuerpos) sigan siendo finitas. (En términos matemáticos, una "colisión" significa que dos cuerpos puntuales ocupan idéntica posición en el espacio).
- Singularidades en las que no ocurre una colisión, pero que **q**(*t*) no permanece finita. En este escenario, los cuerpos divergen hacia el infinito en un tiempo finito, mientras que al mismo tiempo su separación tiende hacia cero (se produce una colisión imaginaria "en el infinito").

Este supuesto se denomina *conjetura de Painlevé* (con singularidades sin colisiones). Su existencia ha sido conjeturada para $n > 3$ por Painlevé. Ejemplos de este comportamiento para $n = 5$ se han construido por Xia [Xia92] y un modelo heurístico para $n = 4$ por Gerver [Ger03]. Donald G. Saari ha demostrado que para 4 o menos cuerpos, el conjunto de datos iniciales que da lugar a estas singularidades tiene medida de Lebesgue cero [Saa77].

Se considerará que se producen choques entre los cuerpos, que es lo que sucede naturalmente con los cuerpos en el espacio. Los meteoritos que se mueven por el espacio son atraídos por la fuerza de gravedad de estrellas y planetas, produciendo finalmente una

colisión si la fuerza de gravedad de la estrella o el planeta es mayor que la del meteorito, si no, el mismo se desviará o será repelido continuado su trayectoria.

La cuestión de encontrar la solución general al problema de los n-cuerpos fue considerada muy importante y desafiante. De hecho, en el siglo XIX tardío el rey Óscar II de Suecia [Ste01], aconsejado por Gosta Mittag-Leffler, estableció un premio para quien pudiese encontrar la solución al problema. El anuncio era bastante concreto:

Dado un sistema arbitrario de muchos puntos de masa que se atraen entre ellos de acuerdo con la ley de Newton, bajo la suposición de que no hay dos puntos que alguna vez choquen, trátese de encontrar una representación de las coordenadas de cada punto como una serie en una variable que sea una función conocida del tiempo, y que para todos los valores la serie ***converja uniformemente****.*

En caso de que el problema no pudiera ser solucionado, cualquier otra contribución importante a la mecánica clásica sería entonces considerada para recibir un premio digno. El premio fue otorgado al matemático francés Henri Pincaré, aunque no solucionó el problema original (la primera versión de su contribución incluso contuvo un error serio). La versión finalmente impresa contenía muchas ideas importantes dirigidas al desarrollo de la teoría del caos. El problema con su planteamiento original fue finalmente solucionado por Karl F. Sundman para n = 3.

2.2 Solución al problema en Arquitectura GPU

Al problema de los N Cuerpos se lo identifica como un problema de alta demanda computacional. El movimiento que realiza un cuerpo es simulado a través de pasos en instantes discretos de tiempo. En cada paso se calcula la fuerza de atracción de cada cuerpo y se modifican las posiciones y velocidades de estos. Este problema ha sido estudiado a lo largo de la historia de la computación [HYNN++09] [YB10] [BGP12].

Para realizar la simulación, se utilizar un vector de fuerza, un vector de velocidades, un vector de posiciones, y un vector de masas. Cada uno de los vectores con un tamaño N.

El algoritmo (*all pair*) realiza las siguientes acciones:

1. Calcula la fuerza del cuerpo i, la cual se encuentra determinada por los N – i cuerpos. A su vez, todos los cuerpos N – i al estar influenciados por el cuerpo i se les modificará su fuerza de gravedad.

2. Se modifica la posición y la velocidad del cuerpo i, dependiendo de su fuerza de gravedad.

3. Se repite 1 y 2, tantas veces como pasos de simulación se requiera.

Comentado [M4]: QUE PASO ACA QUE DESAPARECIO LO QUE HABIA ANTES DE LA PRÓXIMA SECCION??????
ME PARECE QUE HAY QUE VOLVER A PONERLO

La solución all-pair se la podría describir como una simulación por fuerza bruta, en la que todos los cuerpos interactúan con todos. Es uno de los métodos más simples, pero no muy usado por su demanda computacional. La complejidad de dicho algoritmo es O(N2), lo que lo hace útil sólo para sistemas de tamaño moderado, además de demandar gran cantidad de memoria.

Muchas aplicaciones utilizan el algoritmo de los N Body en sus versiones jerárquicas. Para el caso gravitacional solo las partículas más próximas contribuyen significativamente a la dinámica del sistema, las más lejanas hacen aportes menores. La idea básica de los métodos de tipo árbol es establecer una relación jerárquica que permita definir "cercano" y "lejano" para poder calcular una solución eficiente [VR08].

Uno de ellos es Treecode, que divide recursivamente el espacio en subregiones hasta que se alcanza un criterio dado, por ejemplo, hasta que dicho espacio contenga k cuerpos. Cada espacio conforma una celda de cuerpos que interactúan. Cada celda ejecuta el método all-pair, en una o más capas de celdas vecinas. El resultado es que muchos cuerpos en una celda, así como una celda con otra ejecutan la solución all-pair para calcular las fuerzas. El problema que tiene esta aproximación es que cuanto más profundo se llega en la jerarquía más trabajo hay que realizar en cuerpos que pertenecen a celdas muy alejadas [Béd07] [VR08] [Bar12].

Otro algoritmo jerárquico utilizado es el Fast Multipole Method (FMM), el cual, usa también una estructura tipo árbol y el cálculo de la fuerza gravitacional se realiza celda a celda, en lugar de partícula a celda como en el algoritmo mencionado en el párrafo anterior. El FMM reduce la complejidad de la simulación de los N Body a O(N) [HYNN+ +09].

El algoritmo directo para la resolución de problemas de N Body se suele utilizar y tiene buenos resultados en tamaños pequeños del problema. A gran escala los algoritmos mencionados anteriormente son claramente más óptimos por su capacidad de reducir la complejidad del problema significativamente [Nak09].

2.2.1. Solución en una GPU

La solución planteada es apta para ser ejecutada 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 las posiciones y velocidades se realiza de manera independiente. En la Fig.2.1. se muestra el modelo de bloques de threads utilizado para resolver el problema de los N Cuerpos en el presente trabajo.

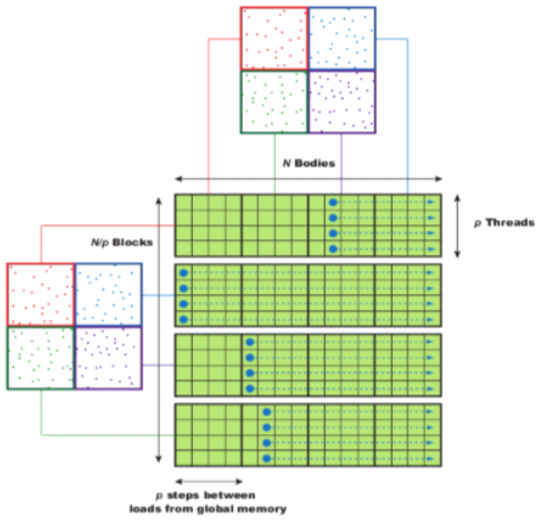

Fig2.1 Modelo de bloque de thread problema N Body en GPU (tomado de [Nvi03])

Como se mencionó en el capítulo anterior, el flujo de un programa CUDA, comprende una reserva de memoria en la GPU, una comunicación desde la CPU a la GPU, la ejecución de una función kernel sobre los datos comunicados, y finalmente una comunicación de los datos calculados desde la GPU a la CPU. Por lo tanto, se almacenan en la memoria global de la GPU las mismas estructuras utilizadas por la CPU: los vectores de fuerza, velocidad, posición y masa. Por convención se suele denominar a las variables igual a las usadas en la CPU, pero con una letra d (indicando device) al inicio. Por ejemplo, si se declara la estructura fuerza en la CPU, para la GPU se denominará d_fuerza.

El paso siguiente es la comunicación de los datos previamente inicializados por la CPU a través del PCI-E. A continuación, la CPU delegará la ejecución a la GPU, y quedará en espera a la respuesta de esta.

En el kernel o función que calcula la fuerza de atracción gravitacional, cada thread calculará su posición en la memoria global que le permitirá acceder al cuerpo para el que tendrá que calcular dicha fuerza. Se declararán tres vectores con dimensión igual a la cantidad de threads por bloque, dos de ellos se utilizarán para almacenar una porción del vector de posiciones, y el tercero para las masas. Estos vectores permiten computar por tile o bloque. Como el acceso a memoria global es muy costoso, es necesario disminuir los accesos haciendo uso de la memoria compartida.

Además de su identificador en la memoria global, cada thread calculará su identificador dentro del vector en memoria compartida. Cada thread traerá de memoria global la posición del cuerpo que le corresponde, por lo tanto, esto significa 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 correspondiente a su identificador dentro de la memoria shared. Antes de comenzar a calcular la fuerza de atracción, los thread son sincronizados, para garantizar que todas las posiciones de los vectores de la memoria compartida hayan sido cargadas [Lue08].

Luego, cada thread realizará 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 haya sido utilizado (Fig. 2.2.). Es necesario destacar que en el proceso de cálculo de la fuerza de atracción se utilizó otra de las optimizaciones de código recomendadas en la literatura [Pic11][Nvi12] de la arquitectura: desenrollo de código. A partir de pruebas realizadas se obtuvo que un desenrollo del loop interno del cálculo de la fuerza de atracción gravitacional de cuatro (4) instrucciones es el óptimo para la presente solución ([Mdo13]).

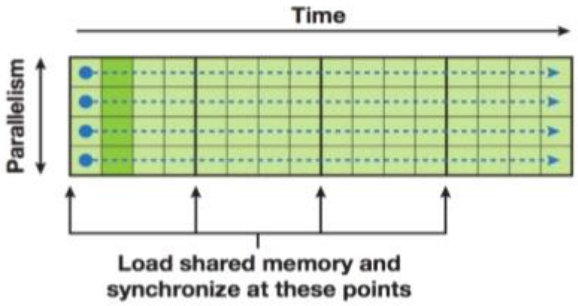

Fig. 2.2 Carga de los cuerpos de memoria global a memoria shared en tile (tomado de [Coo11])

Como se puede apreciar en la Fig. 2.3. el cálculo de la fuerza de atracción gravitacional de un cuerpo con el resto es inevitablemente un proceso secuencial que no puede ser paralelizado.

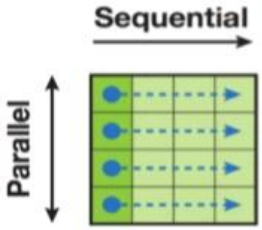

Fig. 2.3 Ejecución del cálculo de atracción gravitacional (tomado de [Coo11])

El paralelismo se obtiene al ejecutar *N* threads que se corresponden con los N Cuerpos que conforman la simulación.

Finalmente, cada thread calculará la velocidad y la posición del cuerpo que le corresponde. El control del flujo de ejecución es devuelto a la CPU una vez completados todos los pasos de simulación en el kernel de la GPU. Por último, la GPU envía los resultados calculados a la CPU por PCI-E [MDDN12] [Mdo13].

2.2.2. Solución en un Cluster de GPU

Basada en la solución del apartado anterior, se realiza la adaptación de esta para ser ejecutada en un Cluster GPU homogéneo. La ejecución en un Cluster de GPU requiere la utilización de una combinación de MPI con CUDA. Por cada proceso MPI creado se tiene asociada una GPU.

Utilizando un esquema master/slave, el proceso master es el encargado de inicializar la información de los cuerpos a distribuir entre los procesos esclavos. Los *N* cuerpos se distribuyen en *P* procesos, por lo que cada proceso debe computar un total de *N/P* elementos. A su vez, cada proceso se encarga de enviar por medio de PCI-E dichos datos a sus 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 *N* datos a la GPU. Una vez que la GPU ha calculado la fuerza de atracción de los cuerpos que le correspondían, enviará dichos datos a la CPU. Paso siguiente, el proceso MPI, comunicará dichos datos al resto de los procesos que conforman el problema; recibiendo al mismo tiempo los datos calculados por los demás. Dicho procesamiento se repetirá *t* pasos, con *t* definido por el usuario [MDCD+14] [MDCD+16].

2.2.3. Solución en MultiGPU

En la solución con MultiGPU, se adaptó la implementación presentada en el apartado 2.2.1. Las GPUs utilizadas son iguales (dos Tesla), por lo cual, al ser una arquitectura homogénea se reparte en partes iguales la cantidad de datos a procesar por cada GPU.

CUDA provee de funciones para setear la afinidad de las GPUs de una PC [Devt18].

```
cudaSetDevice(i);
```

donde $i \leq N$

donde N es la cantidad de GPUs en una PC

La CPU invocará una función la cual realizará el seteo y le comunicará los datos a procesar a la GPU. Una vez realizado esto, el cálculo de la fuerza de atracción gravitacional de cada uno de los cuerpos que componen el espacio de trabajo se realiza del mismo modo que en la solución explicada para una GPU.

Capítulo 3

Green Computing

3.1 Problemática ambiental: el calentamiento global

A partir de la Revolución Industrial, alrededor del siglo XVIII, el hombre con todo su adelanto tecnológico permitió el avance científico e industrial, por un lado, y por el otro inició la degradación ambiental del planeta. El desequilibrio de los Gases de Efecto Invernadero (GEI) en la atmósfera pueden ocasionar grandes cambios en el planeta. Cuanto más alta es su concentración, la temperatura promedio del planeta es mayor [CS07][Guh08][Val14].

El impacto negativo de la actividad humana sobre el planeta, provocando el calentamiento global, fue confirmado en el año 2001 en un reporte realizado por más de 1300 científicos expertos. Al mismo tiempo, nace el protocolo de Kioto, un convenio que busca regular y reducir las emisiones de los gases que incrementan el efecto invernadero, de los cuales el más reconocido es el CO_2 (Dióxido de Carbono) [NU98] [NA11].

"Calentamiento global no solo significa que habrá un leve aumento en el promedio de las temperaturas. Cambiaría radicalmente el sistema en el que la tierra se maneja" (Mark Lynas, autor de "Six Degrees").

La temperatura promedio ha aumentado entre un 0.5 a 1ºC en estos últimos años. El aumento de la temperatura actual del planeta sería un evento totalmente catastrófico, que cambiaría el mundo tal y como lo conocemos [NG08].

Los cinco sectores que constituyen los mayores generadores de emisiones de CO_2 a través del consumo de combustibles fósiles son: la generación de electricidad, la industria, el transporte, el comercio y las residencias [US11]. Según un informe de la

Energy Information Administration, el 98% de todas las emisiones de CO_2 son causadas por el consumo energético [EI07].

El incremento de la población mundial trae aparejado un mayor consumo energético agotando aún más los recursos primarios y ocasionando un mayor daño al medio ambiente [GPVA14].

Muchas organizaciones hablan sobre su deseo de cooperar de una manera "green", y a muchas otras se les aplica impuestos sobre su producción en busca de la reducción del consumo de energía.

Por su parte, en el 2017 el gobierno lanzó convenios para el uso eficiente de la energía en empresas y Pymes, enfocados en el fomento e implementación de los sistemas de gestión de la energía en los procesos productivos y no productivos mejorando el desempeño energético, la competitividad y la capacidad de ahorro y eficiencia de las empresas, contribuyendo también al cuidado del medio ambiente. [ArGob18].

A nivel nacional el Programa Nacional de Uso Racional y Eficiente de la Energía (Decreto P.E.N. N° 140/2007) y la creación del Ministerio de Energía y Minería junto con la constitución de la Subsecretaria de Ahorro y Eficiencia Energética (Decreto P.E.N. N° 231/2015), buscan promover la inclusión de la eficiencia energética y uso responsable de la energía en los programas de carreras universitarias vinculadas a la materia. [ArMin17].

Para este año, el gobierno llevará a cabo el Premio Argentina Eficiente [ArPre18], desde la Secretaría de Ahorro y Eficiencia Energética, con el objetivo de reconocer a las organizaciones públicas y privadas de nuestro país que demuestren un compromiso con el uso eficiente de la energía, certificadas bajo la norma ISO 50.001.

Sin lugar a duda, la reducción del consumo de energía incide de manera directa sobre el estilo de vida de la población y sobre el estado financiero de las organizaciones. Claramente, son estas los mayores centros de consumo de energía contribuyendo a la emisión de greenhouse gas.

Por lo cual, las mismas están utilizando la siguiente ecuación:

Reducir el consumo de la energía =

reducir la emisión de greenhouse gas

= reducir el costo operacional de los centros de datos y negocios [Cur08].

2.3 ¿Qué es Green Computing?

Frente al desafío ambiental los científicos continúan en la búsqueda de diversas alternativas que no dañen el ambiente, poniendo énfasis en tecnologías que sean limpias [MP15]. "*Green*" se ha convertido en un término popular para describir las cosas que son

buenas para el medio ambiente. La comunidad informática, especialmente los usuarios de computadoras han popularizado [TW09]: *Green Computing,* el cual 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.

Se puede mencionar como alguna de las tecnologías verdes propuestas: desmaterialización de documentos (aludiendo al mercado electrónico que produce la disminución del uso del papel físico), la optimización del papel, el reciclaje de equipos; el teletrabajo (como una alternativa laboral, en crecimiento con el uso de las nuevas tecnologías, a distancia, donde el sitio de trabajo puede ser: el domicilio del trabajador u otros espacios que no sean el domicilio de la organización para la cual trabaja), la virtualización , y dentro de ella: Cloud Computing y Grid [MP15].

La sustentabilidad ha emergido como término y tema, y aparecen definiciones para la misma, que la presentan como: "*el desarrollo que satisface las necesidades del mundo actual, sin comprometer la capacidad de las generaciones futuras para satisfacer sus propias necesidades*". El enfoque actual, dispuesto por muchas organizaciones TI, es satisfacer la sustentabilidad, desde las perspectivas económica, social y ambiental, simultáneamente.

Por lo tanto, Green Computing ha surgido como una nueva disciplina para hacer frente a la preocupación de la sustentabilidad cuando TI está involucrado [Val14].

En promedio una PC requiere 85 watts aun en estado ocioso con el monitor apagado, produciendo 1500 pounds (libras) de CO_2 anualmente a la atmósfera. Un árbol absorbe entre 3 y 15 pounds por año, lo que significaría que se necesitan más de 500 árboles para absorber la cantidad de CO_2 que produce una computadora en un año. Aún cuando los dispositivos están apagados, siguen consumiendo energía en estado standby, por lo que se los suele denominar “vampire energy loss”. Estos representan entre el 5% y el 8% de la electricidad consumida durante un año por una residencia. A escala mundial, la energía consumida en estado standby representa el 1% de las emisiones de CO_2 del mundo [SE08] [FZ08].

Los consumidores no toman en cuenta el impacto ecológico a la hora de comprar sus computadoras, sólo prestan atención a la velocidad de sus prestaciones y al precio. Sin embargo, sabemos que a mayor velocidad de procesamiento se requiere mayor poder energético, lo cual nos pone frente al problema de la disipación del calor que, por su lado, 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 el medio ambiente desde la fabricación del equipo hasta su reciclado como, por ejemplo, remplazar el plástico por bioplástico (con polímeros a base de compuestos vegetales), ya que requieren menos petróleo y menos energía para su producción, o el uso de memorias flash, en lugar de discos rígidos cuyas partes mecánicas consumen mayor cantidad de energía, etc. [Ver07].

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 de procesadores (Intel y AMD), han tomado conciencia sobre esta necesidad del uso eficiente de los recursos y la producción de estos disminuyendo no sólo la producción de CO_2, sino también, otras toxinas como el plomo, el cadmio, el mercurio, entre otras, que concentrados en sangre pueden causar varias enfermedades en el corazón, riñones, intestinos, y órganos reproductores [Nic10] [Amd11].

La evolución de las CPUs a las APUs (Unidades de Procesamiento de Aplicaciones), que integran en un mismo chip la CPU y la GPU (Unidad de Procesamiento Gráfico), permite reducir el consumo energético permitiendo la ejecución de aplicaciones de alta calidad gráfica 2D y 3D.

Por su parte, los fabricantes de GPUs han venido mejorando sustancialmente la eficiencia energética de sus placas gráficas aumentando incluso su velocidad [NPas18]. En el año 2013, Nvidia comunicaba que la supercomputadora "Europa" (que presta servicios a todos los miembros del consorcio europeo PRACE (Partnership for Advanced Computing in Europe) y las principales instituciones de investigación italiana), configurada con GPU Tesla, basadas en la arquitectura de alta demanda computacional Kepler, establecía un nuevo récord de eficiencia energética. Europa, construida por Eurotech e instalada en el centro de supercómputo Cineca de Bolonia, registró 3150 megaflops de velocidad sostenida por vatio, una cifra que superaba en un 26% la de la primera supercomputadora de la lista del Green500[1] de ese mismo año. Las Tesla K20 usaban una tecnología de refrigeración por agua caliente de Aurora convirtiendo a dicha supercomputadora en un sistema más eficiente y pequeño que los habituales sistemas de refrigeración por aire. Además, el uso de la tecnología de refrigeración de Eurotech reducía o eliminaba la necesidad de utilizar aire acondicionado en climas templados como el italiano. La energía térmica que producía podía ser utilizada para calentar edificios, alimentar sistemas de aire acondicionado basados en refrigeración por absorción o implantar sistemas de trigeneración, es decir, de producción conjunta de electricidad, calor y frío.

Según explicaba Sumit Gupta, director del área de aceleración Tesla en Nvidia, "*las GPU son, por su naturaleza, más eficientes desde el punto de vista energético que las CPU y, en el caso de los aceleradores Tesla K20, la diferencia es aún mayor. La eficiencia energética se ha convertido en un elemento decisivo para medir el rendimiento computacional. Las GPU permiten a CPD de todos los tamaños (desde pequeños cluster*

[1] En el 2007, nace una nueva entidad denominada Green500, surgida como una alternativa del TOP500, que publica su primera lista que clasifica a las computadoras de mayor eficiencia energética del mundo, considerando el rendimiento por W (FLOPS/W) de cada computadora ejecutando un determinado benchmark [Green18] [Top18].

a los futuros sistemas de cálculo a exaescala) conseguir los objetivos de rendimiento dentro de unos márgenes de consumo económicamente viables." [NTK18].

Todos éstos, son ejemplos de los esfuerzos que se están realizando desde el hardware para contribuir con el medio ambiente.

Sin embargo, Green Computing no sólo debe comprender al hardware sino también al software [KJF11]. Dentro de los factores influyentes en el desarrollo de aplicaciones, se suma la eficiencia energética. Estudiar el consumo energético de los algoritmos que desarrollamos, debe formar parte de los puntos considerados a la hora de ejecutar nuestras aplicaciones. El consumo energético en HPC (High Performance Computing) ha estado en segunda consideración; sin embargo, la escalabilidad de los sistemas ha hecho que cobre importancia [Hem10]. 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 una 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 [BUSRL11]. En HPC la eficiencia energética es un factor limitante. La cantidad de energía necesaria para procesar las grandes cantidades de datos de las aplicaciones que hoy se desarrollan se ha convertido en un problema importante al cual cada vez se le presta más atención. A la hora de implementar nuestras aplicaciones, tenemos en cuenta varios factores tales como: fiabilidad, escalabilidad, portabilidad, eficiencia, etc. Sin embargo, el factor ambiental o la eficiencia del consumo energético no se toma en cuenta. Contar con equipos que realizan un menor consumo de energía, pero utilizarlos mal, nos lleva nuevamente al problema inicial de una cantidad de emisiones injustificadas de dióxido de carbono a la atmósfera. Un ejemplo son aquellas aplicaciones que se ejecutan sobre procesadores multicore pero que solo hacen uso eficiente de un solo núcleo. Este es el caso de muchas aplicaciones que fueron desarrolladas para monoprocesadores y que ahora se ejecutan en computadoras multicore. Aunque también es típico el caso de aplicaciones desarrolladas para arquitecturas multicore que no hacen uso eficiente de los recursos ni las capacidades del hardware [FG08].

Dos consideraciones más se suman al consumo energético, además de la eficiencia energética. Una es la escalabilidad de los sistemas y otra el precio de la energía eléctrica. La gran cantidad de energía consumida reduce la escalabilidad de los sistemas de cómputo. Los sistemas no escalables no son sistemas útiles a largo plazo. Ahora no sólo se debe prever la escalabilidad del problema y la arquitectura, sino también si el aumento de cualquiera de los anteriores influirá de tal manera que el consumo final supere la cantidad de electricidad que se nos puede suministrar, así como también el presupuesto económico disponible para tales gastos.

3.3 Principios fundamentales de la física

Para satisfacer completamente cualquier experimento científico, o para comparar resultados de un experimento con respecto a otros, deben utilizarse formas estándares de medición. En el caso de la medición del consumo energético, se debe medir la cantidad de watts por hora que consume la aplicación que se está ejecutando sobre una arquitectura en particular. Para entender claramente qué es lo que se está midiendo, se deben introducir algunos términos y conceptos básicos de física, electricidad y electrónica, los cuales, serán desarrollados en las siguientes secciones.

3.3.1 Conceptos fundamentales de la Corriente Eléctrica

En la actualidad, nuestra vida depende de la electricidad. Su estudio tiene una larga historia, la cual comienza mucho antes que apareciese la primera lámpara eléctrica. Las primeras observaciones de la atracción eléctrica fueron realizadas por los antiguos griegos. Estos observaron que, al frotar el ámbar, esta atraía pequeños objetos. Por lo que, la palabra eléctrico proviene del vocablo griego elektron, que significa ámbar. La electricidad está en un proceso continuo de estudio, investigación y búsqueda de nuevos usos [TM10].

La Física es una de las ciencias que más ha contribuido al conocimiento de la electricidad y la electrónica. Gracias a la física hoy podemos conocer principios fundamentales de la estructura atómica de la materia.

La estructura básica de todos los materiales que se ven o usan diariamente concuerdan con un modelo conocido como estructura atómica de la materia. La materia es cualquier cosa que tiene masa y ocupa un lugar en el espacio, pudiéndose presentar en diversos estados: líquido, gaseoso, o sólido.

La partícula más pequeña que puede reducirse de un compuesto (sustancia que no puede ser cambiada por procedimientos químicos y no se encuentra combinada o mezclada con otros compuestos) sin perder sus características originales es un átomo. De forma similar al sistema solar, el átomo consiste en un cuerpo central relativamente grande con pequeños cuerpos girando en órbitas alrededor de él. El cuerpo central se lo denomina núcleo, y las partículas que giran alrededor se los llama electrones. El núcleo está formado de partículas con carga positiva denominadas protones, mientras que los electrones tienen carga negativa. La palabra carga implica una fuerza potencial. Cuando un átomo determinado es neutro, o balanceado, las cargas negativas y positivas se encuentran equilibradas, y la carga neta del átomo es cero.

Benjamín Franklin propuso un modelo de electricidad en el cual explicaba que cuando dos objetos se frotan entre sí, parte de la electricidad se transfiere de un cuerpo al otro, y uno de ellos queda con un exceso de carga, mientras que en el otro se produce una deficiencia de carga de igual valor. La suma de la carga total de los dos objetos no cambia,

es decir, que se conserva. La ley de la conservación de carga es una ley fundamental de la naturaleza.

Formalmente, se puede definir a la corriente eléctrica *como un flujo de carga eléctrica que recorre un circuito cerrado que va desde un polo negativo a un polo positivo*. Por lo tanto, la energía eléctrica es una energía en tránsito [QRAM+05] [Gan13] [Pur88].

Históricamente, la corriente eléctrica se definió como un flujo de cargas positivas y se fijó el sentido convencional de circulación de la corriente, como un flujo de cargas desde el polo positivo al negativo. Sin embargo, posteriormente se observó, gracias al efecto Hall, que en los metales los portadores de carga son negativos, es decir los electrones, los cuales fluyen en sentido contrario al convencional.

En el 1800, se logró por primera vez tener un movimiento constante de carga cuando el físico italiano Alessandro Volta inventó la primera pila eléctrica [Gan13].

3.3.2 Unidades de medición

Tres magnitudes invisibles y fundamentales están presentes en todo circuito eléctrico: la tensión, la corriente y la resistencia. A continuación, se presentan algunas de las unidades fundamentales que permiten la medición de estas magnitudes.

3.3.2.1 Joule

Un joule o julio es el trabajo necesario para mover una carga eléctrica de un coulomb a través de una tensión (diferencia de potencial) de un voltio. Es decir, el producto de un voltio por un coulomb. El julio es una unidad de energía muy pequeña para la vida corriente. Aproximadamente, un julio es la cantidad de energía necesaria para levantar 1 kg una altura de 10 cm en la superficie terrestre. Una patada de un deportista puede tener una energía de unos 200 J; una lamparita de bajo consumo de 20 W durante 8 horas gasta unos 600.000 J; y el consumo eléctrico de una familia media durante un mes puede ser de 1.000.000.000 J (unos 278 kWh). Por eso es más frecuente utilizar la unidad kWh (kilovatio hora), en lugar del MJ (megajoule) o el GJ (gigajoule), como debería hacerse.

3.3.2.2 Volt

Es la unidad de medida de potencial eléctrico. Se dice que entre dos puntos existe una diferencia de potencial de un volt cuando, al circular un coulomb entre dos puntos, produce un movimiento de electrones de un joule a través de un conductor entre ambos puntos.

$$1\ volt = \frac{1\ joule}{1\ coulomb} \tag{3.3}$$

La diferencia de potencial, la carga eléctrica y el volt, son modos de expresar la fuerza producida entre cuerpos cargados eléctricamente. Esta fuerza puede llamarse también fuerza electromotriz o tensión. Hay muchos métodos para producir tensión. Algunos de estos métodos son: fricción, energía calorífica, energía luminosa, energía química y energía mecánica. La energía química y la mecánica es la que se utilizan como principales medios para producir energía eléctrica en forma constante para mantener un flujo continuo de electrones para el funcionamiento de aparatos eléctricos.

3.3.2.3 Ampere

El ampere es la unidad de medida eléctrica que mide la cantidad de flujo de electrones que fluye a través de un material o medio. Se dice que la intensidad de flujo de corriente en un conductor es de un ampere, cuando por un punto del conductor, fluye un coulomb por segundo. Expresando la intensidad de flujo la ecuación de esta es:

$$I = \frac{Q}{T} \tag{3.4}$$

I = corriente en ampere
Q = cantidad de carga eléctrica
T = tiempos en segundos

En condiciones normales los electrones libres en un conductor no fluyen en ninguna dirección especial; esta actividad de los electrones es una acción casual, desordenada, donde los electrones son liberados por efecto de la temperatura ambiente, y simplemente saltan de un átomo a otro, sin rumbo determinado. Sin embargo, cuando un conductor está conectado a los terminales de una fuente de tensión, esta fuerza producirá un movimiento de los electrones libres, desde el terminal negativo de la fuente hasta el terminal positivo.

El movimiento de los electrones es el resultado del movimiento al azar más el movimiento producido por la tensión aplicada. Como resultado, todos los electrones simultáneamente son guiados dentro del conductor por acción de una tensión sostenida. Las colisiones entre ellos en el conductor contrarrestan el incremento de velocidad de estos. Se ha determinado que la velocidad de desplazamiento de la corriente de electrones correspondiente al ampere alcanza aproximadamente 1 cm por segundo, según las dimensiones del conductor, el material y otros factores.

Para lograr que la energía eléctrica pueda recorrer muchos kilómetros de distancia para suministrarla, deben producirse los siguientes movimientos de los electrones: cuando se cierra la llave del circuito, aparece una tensión. El terminal positivo atrae electrones dejando con déficit de electrones al punto A (como se observa en la Fig. 3.4). Entonces, el punto A atrae electrones del punto B, y este mismo efecto se repite a lo largo del conductor, C a D y E a F. Casi en el mismo instante en que el terminal positivo atrae un electrón del punto A, el terminal negativo suministra un electrón al punto F. De esta manera, pese a que los electrones se mueven a una pequeña velocidad, el efecto de los cambios de posiciones de los electrones se propaga a lo largo del conductor casi instantáneamente. La velocidad de propagación es aproximadamente la de la luz (300.000 km por segundo).

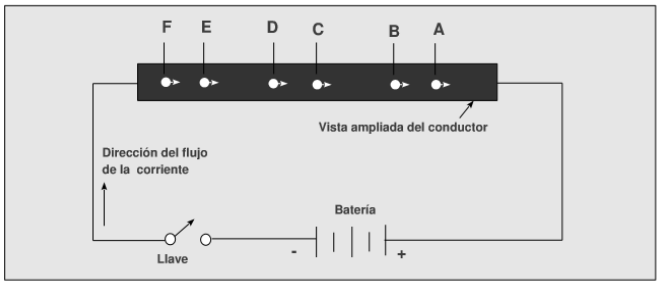

Fig. 3.4. Movimiento resultante de electrones a través de un conductor

3.3.2.4 Watt

El vatio o watt [Sap53] [RAE18], es la unidad de potencia de energía. Se puede decir que la potencia es el ritmo al que se usa (o genera) la energía o se realiza trabajo. La diferencia de energía potencial entre dos puntos se le denomina voltaje. Esta tensión puede ser vista como si fuera una "presión eléctrica" debido a que cuando la presión es uniforme no existe circulación de cargas y cuando dicha "presión" varía se crea un campo eléctrico que a su vez genera fuerzas en las cargas eléctricas. Matemáticamente, la diferencia de potencial eléctrico entre dos puntos A y B es la integral de línea del campo eléctrico:

$$V(A) - V(B) = -\int_{B}^{A} \vec{E}\,\overrightarrow{dt} \tag{3.5}$$

Siendo $\vec{E}$ el campo eléctrico

Generalmente se definen los potenciales referidos a un punto inicial dado. A veces se escoge uno situado infinitamente lejos de cualquier carga eléctrica. Cuando no hay campos magnéticos variables, el valor del potencial no depende de la trayectoria usada

para calcularlo, sino únicamente de sus puntos inicial y final. Se dice entonces que el campo eléctrico es conservativo.

Otra de las formas de expresar la tensión entre dos puntos, es en función de la intensidad de corriente y la resistencia existentes entre ellos.

En el caso de campos no estacionarios el campo eléctrico no es conservativo y la integral de línea del campo eléctrico contiene efectos provenientes de los campos magnéticos variables inducidos o aplicados, que corresponden a una fuerza electromotriz inducida, que también se mide en voltios. Entonces:

$$Watt = I * V \tag{3.6}$$

Siendo
I la corriente
V = volt

Finalmente, también se puede definir al vatio como un julio por segundo. Por ejemplo, si una lámpara de 100 vatios está encendida durante una hora, la energía consumida es de 100 vatios-hora (W·h) o 0,1 kilovatio-hora (kW·h) o (60 × 60 × 100) 360.000 julios (J). La capacidad o potencia de una central energética se mide en vatios, pero la energía generada anualmente se medirá en vatios-hora (W·h), kilovatios-hora (kW·h), megavatios-hora (MW·h), gigavatios-año (GW·año).

3.3.3 Instrumentos de medición

3.3.3.1 Osciloscopio

Se denomina osciloscopio a un instrumento de medición electrónico para la representación gráfica de señales eléctricas que pueden variar en el tiempo. Permite visualizar fenómenos transitorios, así como formas de ondas en circuitos eléctricos y electrónicos. Mediante su análisis se puede diagnosticar con facilidad cuáles son los problemas del funcionamiento de un determinado circuito. Es uno de los instrumentos de medida y verificación eléctrica más versátiles que existen, y se utiliza en una gran cantidad de aplicaciones técnicas.

Un osciloscopio puede medir un gran número de fenómenos. El osciloscopio presenta los valores de las señales eléctricas en forma de coordenadas en una pantalla, en la que normalmente el eje X (horizontal) representa tiempos y el eje Y (vertical) representa tensiones. La imagen así obtenida se denomina oscilograma. Suelen incluir otra entrada, llamada "eje Z" que controla la luminosidad del haz, permitiendo resaltar o apagar algunos segmentos de la traza. El funcionamiento del osciloscopio está basado en la posibilidad de desviar un haz de electrones por medio de la creación de campos eléctricos y magnéticos.

3.3.3.2 Amperímetro

Un amperímetro es un instrumento que sirve para medir la intensidad de corriente que está circulando por un circuito eléctrico. Para efectuar la medida de la intensidad de la corriente circulante, el amperímetro debe ser colocado en serie, para que sea atravesado por dicha corriente. Esto lleva a que el amperímetro debe poseer una resistencia interna lo más pequeña posible, a fin de que no produzca una caída de tensión apreciable. Para ello, en el caso de instrumentos basados en los efectos electromagnéticos de la corriente eléctrica, están dotados de bobinas de hilo grueso y con pocas espiras.

3.3.3.3 Pinza Amperométrica

La pinza amperométrica es un tipo especial de amperímetro que permite obviar el inconveniente de tener que abrir el circuito que se quiere medir la corriente para colocar un amperímetro clásico [Al08].

El funcionamiento de la pinza se basa en la medida indirecta de la corriente circulante por un conductor, a partir de un campo magnético o de los campos que dicha circulación de corriente genera. Recibe el nombre de pinza porque consta de un sensor, en forma de pinza, que se abre y abraza el cable cuya corriente se quiere medir.

Capítulo 4

Trabajo Experimental

El entorno de prueba está compuesto por las siguientes máquinas:

- Cluster de multicore con procesadores Quad Core Intel i5-2310 de 2,9 GHz, con caché de 6 MB: cada nodo cuenta con una GPU GeForce TX 560TI. CUDA 8.0
- Cluster de multicore con procesadores Quad Core Intel i5-4460 de 3.2 GHz, con caché de 6 MB: cada nodo cuenta con una GPU GeForce GTX 960. CUDA 8.0
- Una PC con un procesador Quad Core Intel i5-2310 de 2,9 GHz, con dos GPU Tesla C2075. CUDA 4.2

A continuación, se detallan algunas especificaciones técnicas de cada una de las tarjetas gráficas utilizadas en el presente trabajo.

Especificación Técnica	Tesla C2075	GeForce GTX 560	GeForce GTX 960
Microarquitectura	Fermi	Fermi	Maxwell
Cantidad de cores	448	384	1024
Memoria	6 GB	1 GB	2 GB
Velocidad del procesador	1150MHz	835MHz	1177MHz
Rendimiento de punto flotante	1.03TFLOPS	1.28TFLOPS	2.41TFLOPS
Velocidad de la memoria efectiva	6000MHz	4000MHz	7012MHz
Velocidad de Memoria	1500MHz	1000MHz	1753MHz
Memoria Máxima de Ancho de Banda	144GB/s	128GB/s	112.2GB/s
Capacidad del Bus de Memoria	384bit	256bit	128bit
Potencia de Diseño Técnico	225W	170W	120W
Número de transistores	3000 millones	1950 millones	2940 millones
Tamaño de los semiconductores	40nm	40nm	28nm
Anchura	248mm	229mm	206mm

Todas las pruebas realizadas de la simulación se hicieron para uno y tres pasos. La cantidad de cuerpos (N) se varió entre 128000, 256.000, 512.000 y 1.024.000. En las pruebas se utilizaron bloques de threads de 256 hilos para todas las GPUs.

Para realizar las mediciones se utilizan una pinza amperimétrica (para la medición de corriente, con una sensibilidad 1A/100mv, 1A/10mv y 1A/1mv) y un transformador (que censa la tensión existente en la entrada del suministro de energía). La tensión se midió directamente de la línea eléctrica a la cual se encuentra conectado el cluster de multicore y la MultiGPU. Ambos dispositivos se conectan a los canales de entrada del Osciloscopio y desde este último, se configuran los parámetros de visualización y ajustes de la pantalla según el muestreo que se desee realizar.

El osciloscopio utilizado (Rigol DS1074Z) tiene como características principales para estas mediciones 1GSa/s de tasa de muestreo, “1 giga sample sobre segundo”, esto quiere decir, 1 billón de muestras en 1 segundo en el mejor de los casos para los canales analógicos. Por otro lado, se almacenan en su memoria hasta 24Mpts, lo que permite tener una memoria de 24 millones de muestras (utilizando un único canal, si se utilizan varios canales, dicha cantidad se divide por este valor) [Rigol18]. Para la medición de consumo realizada se utilizaron 2 canales (uno para la corriente y otro para la tensión), por lo que finalmente se almacenaran 12Mpts en memoria.

Además, la tasa de muestreo se establece en base a la relación entre la capacidad de la memoria y otro parámetro que afecta el modo en que se realizará el muestreo: H. El parámetro H está dado por la visualización de las ondas en la pantalla del instrumento. H es la cantidad de tiempo que representa cada fragmento de los 12 visibles dentro de la pantalla del osciloscopio. H toma valores entre 5ns y 50seg., por lo que permite como máximo, obtener una ejecución en memoria de 10 minutos. Al extender el tiempo de ejecución aumentando el valor de H, se ve afectada la tasa de muestreo mediante la siguiente relación (1):

Tasa de muestreo = Capacidad de la memoria / (H * Divisiones de la pantalla) (1)

La Tasa de muestreo o cantidad de puntos que se obtienen en un segundo está dada por la capacidad de la memoria (es el número máximo de puntos que se pueden almacenar según la cantidad de canales utilizada), H la cantidad de segundos por división de la pantalla o resolución horizontal, y la cantidad de divisiones por pantalla que para la serie de osciloscopios DS1000Z es 12 divisiones. Yendo a un ejemplo más práctico, para los diferentes valores de H, con 2 canales se pueden obtener las diferentes tasas de muestreo que se observan en la Tabla 4.1.

Tabla 4.1. Tasas de muestreo del Osciloscopio

H (seg. por división de pantalla)	H*12 (Tiempo total con 12 divisiones)	Muestras/(H*12) (muestras/seg)	Muestras/ms
1	12	1000000	1000
2	24	500000	500
5	60	200000	200
10	120	100000	100
20	240	50000	50
50	600	20000	20

Como se observa en la Tabla 4.1, utilizando valores de H de 1seg. a 50 seg. se obtienen tasas de muestreo de 1MSa/s hasta 20KSa/s. Si se tiene en cuenta el tiempo de ejecución para estos casos van desde 12seg. en pantalla hasta 10min. Como dato adicional, la última columna tiene la cantidad de muestras que se tomarán en cada milisegundo [GM18].

Cabe destacar que la medición se realiza de modo manual, es decir, se debe colocar el osciloscopio en modo ejecución, luego lanzar a ejecutar la aplicación. Una vez finalizada la ejecución del problema, se detiene la medición del osciloscopio y desde la PC que recoge los datos del muestreo se le envía la orden de transferencia de la información. Por lo tanto, los casos probados para el presente trabajo se encuentran limitados por el tiempo de medición del osciloscopio del que se dispone, más el retardo del lanzamiento de la ejecución del problema y el detenimiento del osciloscopio.

Por otra parte, para analizar las muestras obtenidas para cada escenario planteado, se modificó una aplicación de análisis de consumo energético denominada energyAnalyser[2] (La herramienta original fue modificada para satisfacer los requerimientos del presente trabajo).

El primer paso que se lleva cabo es: a partir de los datos de la corriente y la tensión extraídos del osciloscopio, realizar el cálculo de la potencia instantánea (2).

$$P(t) = v(t) * i(t) \quad (2)$$

Siendo v la tensión e i la corriente

[2] La misma fue desarrollada por el Dr. Balladini J. de la Universidad Nacional del Comahue.

La potencia eléctrica representa la cantidad de energía entregada o absorbida por un elemento. Por lo tanto, la potencia instantánea es la cantidad de energía entregada o absorbida por un elemento en un instante determinado de tiempo.

Una vez que se obtiene la potencia instantánea, se calcula la potencia promedio, y la cantidad de Joules consumidos por la ejecución del algoritmo de los N Cuerpos.

Los datos obtenidos son el resultado de un promedio de diez ejecuciones del algoritmo y mediciones de consumo energético. Se plantó un lote de escenarios conformados por los cuatro tamaños de entrada, repitiendo la simulación diez veces.

Comentado [M5]: CUANTAS?????

Cada repetición de la simulación, anteriormente mencionada, se refiere a la cantidad de pasos explicados en el Capítulo 2, apartado 2.2. Cada paso de simulación recibe como entrada una cantidad de cuerpos que se mantiene constante durante el cálculo de la fuerza de atracción gravitacional, dando como resultado la misma cantidad de cuerpos a la salida de dicho paso, generando la entrada del próximo paso de simulación si lo hubiere. La cantidad de cómputo que se realiza en cada paso de ejecución es igual: por cada cuerpo que comprende el espacio del problema se calculará su fuerza de atracción con respecto al resto de los cuerpos que conforman el espacio. Estas dos consideraciones determinan que el tiempo de ejecución del algoritmo no es dependiente de la cantidad de pasos de ejecución, sino del tamaño de la entrada. Por lo tanto, el tiempo del cálculo de la fuerza gravitacional del conjunto de cuerpos que comprenden el espacio de trabajo de la simulación para *x* pasos es igual a el tiempo de ejecución de un paso de la simulación multiplicado por la cantidad de pasos que se quiera repetir la misma.

Por lo tanto, puede concluirse que la cantidad de pasos de la simulación es independiente del tiempo. Por lo que, sabiendo el tiempo requerido para simular el cálculo de la fuerza gravitatoria para un tamaño de entrada N, la estimación del tiempo de la simulación para *x* cantidad de pasos estará dado por (3).

$$\text{Tiempo}(N) = \text{Tiempo\_1Paso}(N) * x + v \qquad (3)$$

Siendo:
N = la cantidad de cuerpos que conforman la simulación
Tiempo(N) = tiempo de ejecución en segundos de la simulación
Tiempo_1Paso(N) = tiempo de ejecución en segundos de la simulación al ejecutar un paso
x = cantidad de pasos
v = término variable, comprende por ejemplo comunicación, retardo de sincronización, etc.

Para corroborar el modelo de estimación propuesto en (3), se evaluarán los tiempos obtenidos mediante medición y el modelo de estimación de tiempo (3) utilizando regresión lineal. El coeficiente de correlación de Pearson (4), permitirá comprobar la relación entre los modelos.

$$r_{xy} = \frac{\sum xy}{\sqrt{\sum x^2}\sqrt{\sum y^2}} \quad (4)$$

Donde

- Si r = 1, existe una correlación positiva perfecta.
- Si 0 < r < 1, existe una correlación positiva.
- Si r = 0, no exise correlación.
- Si r –1 < r < 0, exise una correlación negativa.
- Si r = -1, existe una correlación negativa perfecta.

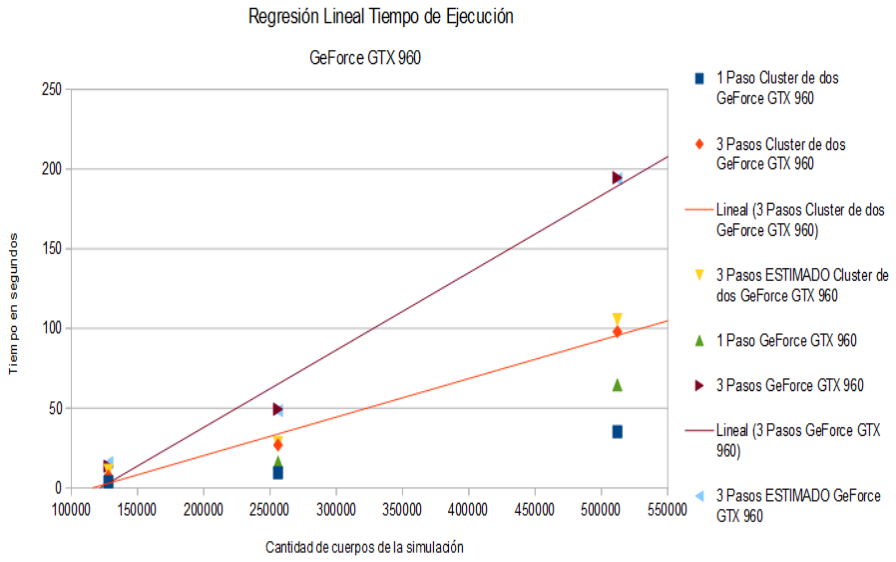

Fig. 4.1 Regresión Lineal de tiempo de ejecución de una GeForce GTX 960 y un Cluster con dos GeForce GTX 960

Como se puede observar en la Fig. 4.1, la tendencia del tiempo ejecución de la simulación es lineal, con un coeficiente de correlación lineal de 0,99.

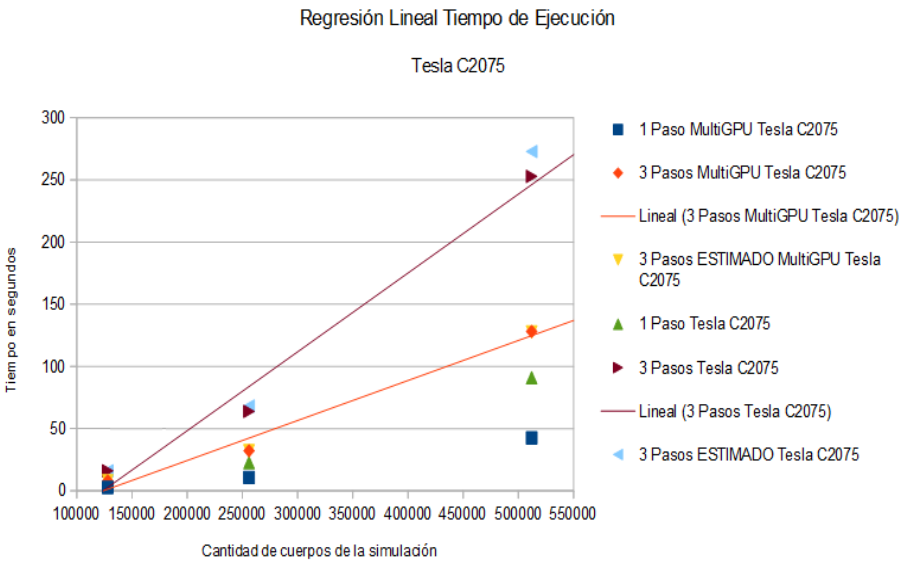

Fig. 4.2 Regresión Lineal del Tiempo de Ejecución de la simulación utilizando una Tesla C2075 y una MultiGPU con dos Teslas C2075

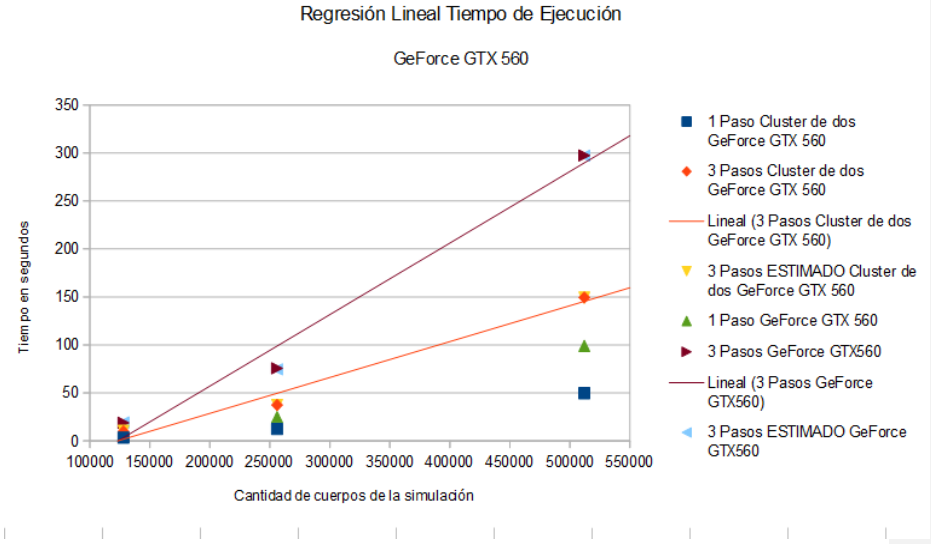

Fig. 4.3 Regresión Lineal de Tiempos de ejecución de la simulación para una GeForce GTX 560 y un Cluster de dos GeForce GTX 560

Como se observa en las Fig. 4.2 y 4.3, el modelo de estimación de tiempo está correlacionado linealmente con los tiempos medidos, con un coeficiente de correlación para ambos casos de 0,99. En el Apéndice A, pueden encontrarse las tablas con los tiempos en segundos medidos y estimados de la simulación para 1 paso y 3 pasos.

Para comprobar el modelo de estimación de tiempo (3), se calculará el tiempo de ejecución de la simulación para 6 pasos, y se lo comparará con los tiempos medidos para la misma cantidad de pasos. La Tabla 4.2, muestra los tiempos medidos y estimados en segundos de la simulación para una GeForce GTX 960 y un Cluster de dos GeForce GTX 960.

Tabla 4.2 Tiempo Medido y Tiempo Estimado en segundos para una GeForce GTX 960 y un Cluster de dos GeForce GTX 960

Geforce GTX 960			
	128000	256000	512000
6 Pasos	26,08	97,84	386,59
6 Pasos ESTIMADO	31,38	96,9	387,78
Coeficiente de correlación		0,99	
Cluster dos GeForce GTX 960			
6 Pasos	18,61	55,11	203,5
6 Pasos ESTIMADO	22,14	56,52	211,08
Coeficiente de correlación		0,99	

Las Tabla 4.3 y 4.4, muestran los tiempos medidos y los tiempos estimados en segundos de la simulación para una Tesla C2075 y una MultiGPU con dos Tesla C2075, y GeForce GTX 560 y un Cluster de dos GeForce GTX 560, respectivamente.

Tabla 4.3 Tiempo Medido y Tiempo Estimado en segundos para una Tesla C2075 y una MultiGPU con dos Tesla C2075

Tesla C2075			
	128000	256000	512000
6 Pasos	32,18	127,77	505,76
6 Pasos ESTIMADO	32,16	136,44	545,7
Coeficiente de correlación		0,99	
MultiGPU Tesla C2075			
6 Pasos	16,15	64,41	255,52
6 Pasos ESTIMADO	16,14	64,38	255,48
Coeficiente de correlación		0,99	

Tabla 4.4 Tiempo Medido y Tiempo Estimado en segundos para una GeForce GTX 560 y un Cluster de dos GeForce GTX 560

Geforce GTX 560			
	128000	256000	512000
6 Pasos	38,17	149,47	594,71
6 Pasos ESTIMADO	38,52	149,22	594,48
Coeficiente de correlación		0,99	
Cluster dos GeForce GTX 560			
6 Pasos	17,19	74,87	298,69
6 Pasos ESTIMADO	20,76	74,88	298,5
Coeficiente de correlación		0,99	

Con lo anteriormente mostrado, los experimentos que siguen se realizaron para un paso de simulación.

Partiendo de esta consideración, para conocer una estimación que nos aproxime al tiempo de ejecución para cualquier N como entrada del algoritmo, se debe conocer cuál es su orden de magnitud. A partir de esto, se podrá conocer el comportamiento asintótico del algoritmo, es decir, la tendencia del algoritmo al variar el valor de N.

La solución utilizada para resolver el problema es denominada: all-pair. Se la podría describir como una simulación por fuerza bruta, en la que todos los cuerpos interactúan con todos. La complejidad de dicho algoritmo es $O(N^2)$ [Nak09].

La curva que representa la tendencia de un algoritmo de este orden es un polinomio de grado dos. Utilizando el Método de Regresión Polinomial, se puede hallar un modelo de estimación de tiempo para determinar el tiempo de la simulación.

En la Fig. 4.4, se puede observar la curva de tiempo para los tamaños de entrada medidos (128000, 256000 y 512000) y la tendencia del tiempo por el Método de Regresión Polinomial, para una GPU GeForce GTX 960 y un Cluster de dos GPU GeForce GTX 960.

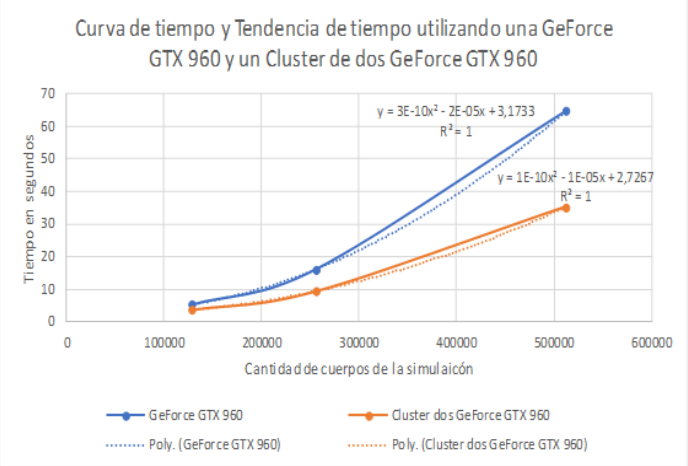

Fig. 4.4 Curva de Tiempo y Tendencia de Tiempo de la simulación utilizando una GeForce GTX 960 y un Cluster de dos GeForce GTX 960

En la Fig. 4.5 se observa la curva de tiempo para los distintos tamaños de entrada probados y la tendencia del tiempo para una GPU Tesla C2075 y una MultiGPU con dos Tesla C2075. Finalmente, se muestra la curva de tiempo y la del tiempo de una GPU GeForce GTX 560 y un Cluster de dos GPU GeForce GTX 560 en la Fig. 4.6.

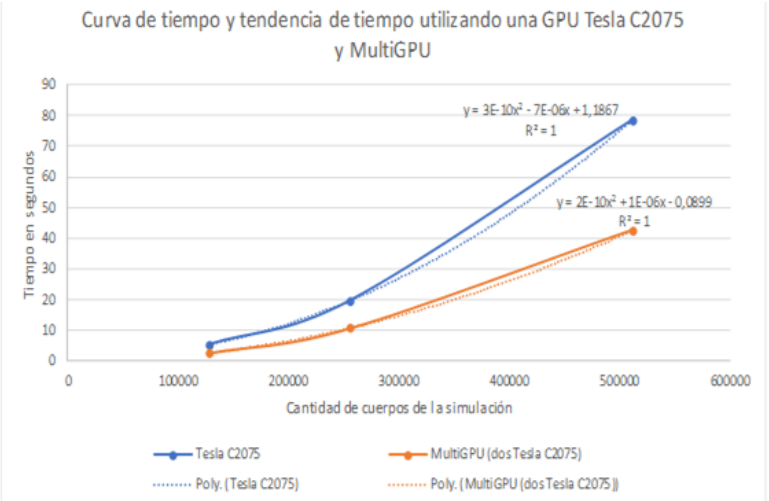

Fig. 4.5 Curva de Tiempo y Tendencia de Tiempo de la simulación utilizando una Tesla C2075 y una MultiGPU con dos Tesla C2075

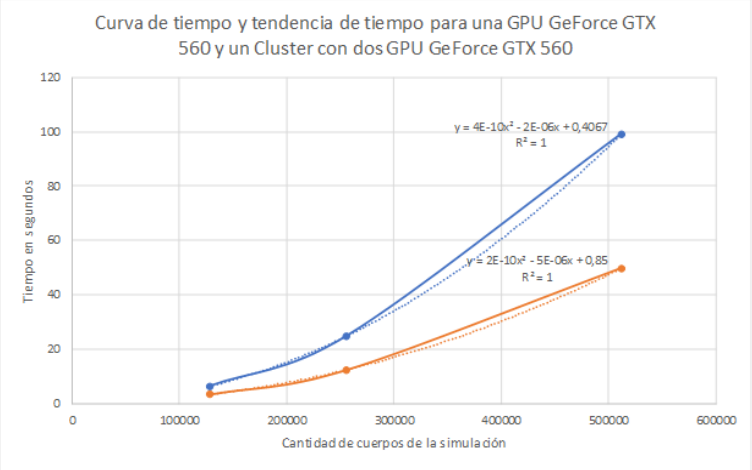

Fig. 4.4 Curva de Tiempo y Tendencia de Tiempo de la simulación utilizando una GeForce GTX 560 y un Cluster de dos GeForce GTX 560

A partir de las regresiones realizadas se obtienen los modelos de estimación de tiempo para cada una de las arquitecturas GPU utilizadas. A continuación, la Tabla 4.5, presenta los tiempos estimados para cada arquitectura utilizando su respectivo modelo de estimación.

El coeficiente de correlación para todas las formas de las arquitecturas probadas es 0,99.

Tabla 4.5 Tiempos Estimados en segundos para cada arquitectura GPU utilizando sus respectivos modelos

una GPU			
	GeForce GTX 960	Tesla C2075	Geforce GTX 560
128000	5,23	5,36	6,42
256000	16,15	22,74	24,88
512000	64,63	90,95	99,09
Cluster GPU/MultiGPU			
	GeForce GTX 960	Tesla C2075	Geforce GTX 560
128000	3,69	2,69	3,46
256000	9,42	10,74	12,48
512000	35,18	42,59	49,75

Se propone probar los respectivos modelos de estimación de tiempo para cada arquitectura GPU con N = 1024000. A continuación, la Tabla 4.6, presenta los tiempos

en segundos medidos y los tiempos en segundos estimados para cada arquitectura empleada utilizando su respectivo modelo de estimación de tiempo. Además, se muestra el coeficiente de correlación para cada uno.

Tabla 4.6 Tiempos Medidos y Tiempos Estimados en segundos de la simulación utilizando las distintas arquitecturas GPU

	GeForce GTX 960	Tesla C2075	GeForce GTX 560
Tiempo Medido	259,11	337,33	396,04
Tiempo Estimado	297,26	308,59	417,78
Coeficiente de correlación		0,99	
	Cluster dos GeForce GTX 960	MultiGPU dos Tesla C2075	Cluster dos GeForce GTX 560
Tiempo Medido	135,62	168,58	198,17
Tiempo Estimado	97,34	210,64	205,44
Coeficiente de correlación		0,99	

Los algoritmos paralelos utilizan como medida de performance el speedup para medir el rendimiento del aumento de cantidad de procesadores comparado con el rendimiento de utilizar sólo uno. Las arquitecturas empleadas en el presente trabajo como se explicó en el Capítulo 1, son inherentemente paralelas; por lo tanto, esta métrica no es lo suficientemente representativa para medir su rendimiento.

Por esto se definirá el concepto de aceleración (5), como métrica para medir performance al utilizar más de una GPU [MDCD+14].

$$\text{Aceleración} = \frac{Tiempo\_1GPU(N)}{Tiempo\_+GPU(N)} \quad (5)$$

Siendo
N = tamaño de entrada de datos
Tiempo_1GPU(N) = tiempo en segundos del algoritmo con N cantidad de datos de entrada utilizando una GPU
Tiempo_+GPU(N) = tiempo en segundos del algoritmo con N cantidad de datos de entrada utilizando más de una GPU

La Fig. 4.7 presenta la aceleración alcanzada por la simulación para cada arquitectura GPU respectivamente.

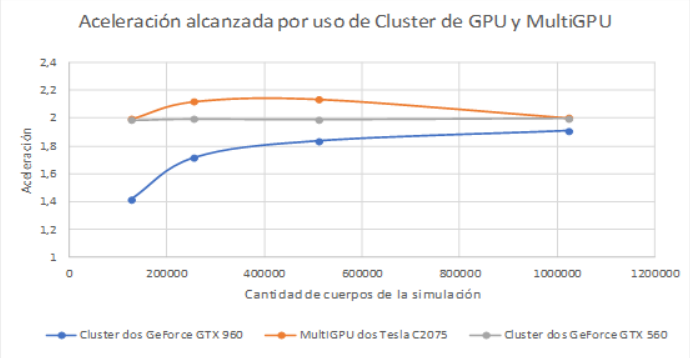

Fig. 4.7 Aceleración alcanzada por la simulación utilizando las distintas arquitecturas GPU propuestas

Para el caso de la MultiGPU con dos Tesla C2075, se puede observar que da como resultado una aceleración superlineal. Esto se debe generalmente a la división de los datos. Si se tiene N cantidad de datos a procesar con una GPU, cuando se trabaja con dos, la división de los datos (como se realizó en el presente trabajo, para el problema propuesto) es N/2, es decir se tiene una menor cantidad de datos a procesar por cada GPU, por lo cual, el tiempo será menor y el resultado será superlineal.

Comentado [M6]: Explicar un poco más, entiendo que se debe al uso de cache u otra memoria??

Hoy por hoy, acelerar el cómputo, teniendo en cuenta sólo performance en la escalabilidad de los sistemas, no alcanza. Un nuevo concepto se ha implantado referido al cuidado del medio ambiente que impacta en el diseño y la fabricación del hardware, así como también en la implementación del software. Un sistema que brinda performance pero que por otro lado consume demasiada energía, no es útil ni sostenible en el tiempo y mucho menos rentable económicamente.

A continuación, se propone un modelo de estimación de consumo de energía (6) para las arquitecturas utilizadas y la simulación desarrolla.

$$\text{Energía} = PMGPU_x * \text{Tiempo}(N) + \upsilon \quad (6)$$

Siendo
$PMGPU_x$ = potencia media de energía consumida por la simulación utilizando la GPU x.
Tiempo(N) = tiempo de ejecución en segundos de la simulación para el tamaño de entrada N
υ = término variable, comprende factores tales como la temperatura, la refrigeración, condiciones ambientales, etc.

Se puede definir a la potencia promedio de energía consumida por una GPU determinada como (7):

$$PPGPU_x = \frac{\sum_{i=0}^{n} P(t)_i}{n} \quad (7)$$

Siendo
$P(t)_i$ = Potencia instantánea en el instante i
n = cantidad de muestras

La potencia promedio de energía es la cantidad de energía promedio entregada o absorbida por un elemento. Lo que implica que la cantidad de energía necesaria para procesar un dato es la misma independientemente del dato que sea y la cantidad de datos a procesar. En la Tabla 4.7 se pueden observar algunos valores medidos de la potencia promedio de energía para una GPU para los distintos tamaños de entrada y varios pasos de la simulación. La Tabla 4.8 muestra los valores probados de potencia promedio de energía para los Cluster de GPU y MultiGPU. En el Apéndice B, apartado B.1, se encuentran todas las tablas referidas al experimento realizados en cuanto a potencia promedio de energía.

Comentado [M7]: Qué significa "probados"??

Tabla 4.7 Potencia Promedio en Watts por segundo consumidos por la simulación para una GPU

pasos	GeForce GTX 960	Tesla C2075	GeForce GTX 560
128000			
1	160,48	276,74	191,87
3	163,75	279,45	196,49
256000			
1	160,91	276,96	193,01
3	166,07	279,26	203,28
5120000			
1	162,36	278,51	202,77
3	164,49	278,17	208,97

Tabla 4.8 Potencia Promedio en Watts por segundo consumidos por la simulación para los Cluster de GPU/MultiGPU

pasos	Cluster dos GeForce GTX 960	MultiGPU dos Tesla C2075	Cluster dos GeForce GTX 560
128000			
1	306,9	376,4	382,29
3	307,61	385,71	383,78
256000			
1	315,62	375,94	390,88
3	320,42	381,4	392,33
5120000			
1	321,14	378,39	394,64
3	322,9	388,37	406,63

Como se puede observar en las Tablas 4.7 y 4.8, la variabilidad de la potencia promedio de energía no cambia significativamente. Las variaciones dadas pueden deberse a los movimientos de los datos en la jerarquía de memoria de la GPU o a la refrigeración, entre otras.

Partiendo de lo anterior, podemos definir la potencia media de energía de una GPU determinada para la simulación de los N Cuerpos (8):

$$PMGPU_x = \frac{\sum_{i=0}^{n}(PPGPU_x)_i}{n} \quad (8)$$

Sea
$(PPGPU_x)_i$ = las potencias promedio de energía para cada tamaño de entrada y pasos del experimento para la GPU x.
n = la cantidad de muestras

La Tabla 4.9, muestra la potencia media de energía para una GPU, para los Cluster de GPU y MultiGPU.

Tabla 4.9 Potencia Media de Energía medida en Watts por segundo para una GPU, para los Cluster de GPU y para MultiGPU

GeForce GTX 960	Tesla C2075	GeForce GTX 560
163,01	278,18	199,39
Cluster dos GeForce GTX 960	MultiGPU dos Tesla C2075	Cluster dos GeForce GTX 560
315,76	381,08	391,75

Teniendo definidos todos los componentes del modelo de estimación propuesto, la Fig. 4.8 muestra la energía estimada y la energía medida para una GPU. En la Fig. 4.9 se presenta la energía estimada y la energía medida para los Cluster de GPU y la MultiGPU.

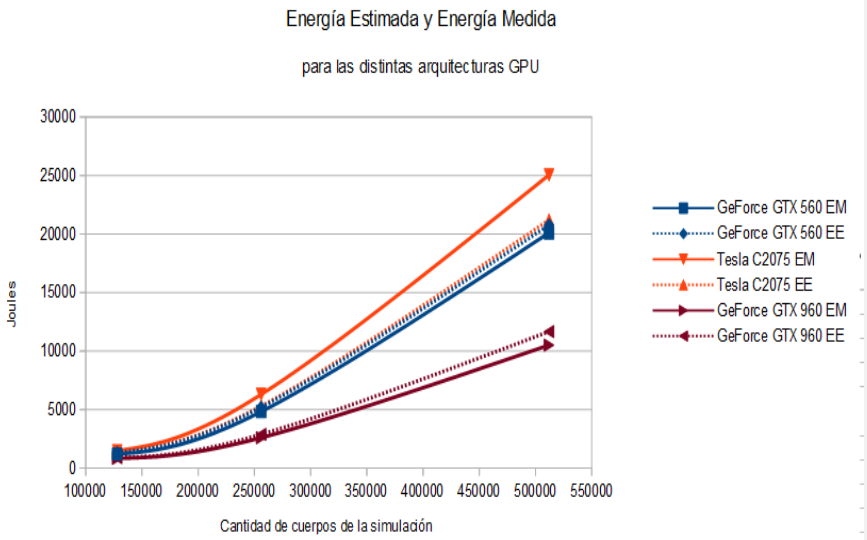

Fig. 4.8 Energía Estimada (EE) y Energía Medida (EM) en Joules por la simulación utilizando una GPU

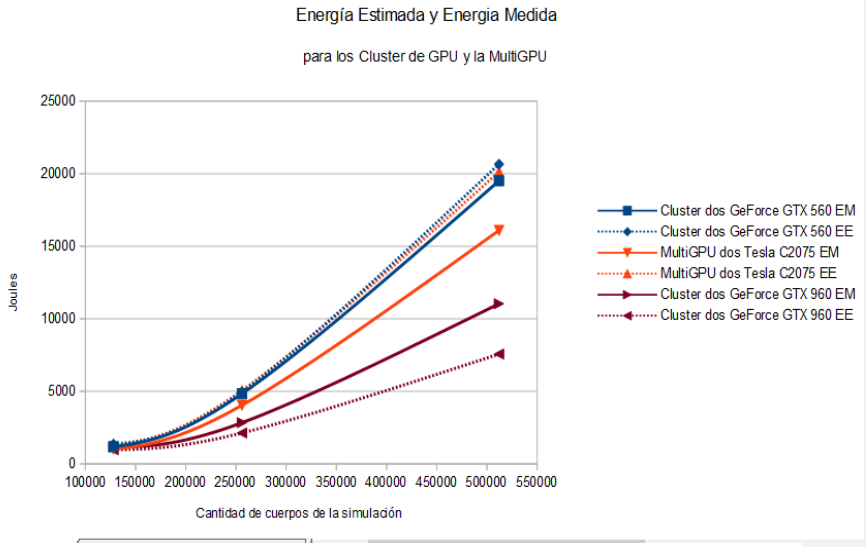

Fig. 4.9 Energía Estimada (EE) y Energía Medida (EM) en Joules por la simulación utilizando los Cluster de GPU y la MultiGPU

La Tabla 4.9, muestra la energía estimada y la medida en Joules y el coeficiente de correlación para cada arquitectura GPU.

Tabla 4.9 Energía Estimada (EE), Energía Medida (EM) en Joules y coeficiente de correlación entre ambas para la simulación para las distintas arquitecturas GPU empleadas.

	N	EM	EE	coeficiente correlacion
GeForce GTX 960	128000	840,15	899,81	0,9999967785
	256000	2600,85	2886,9	
	512000	10512,37	11668,25	
Cluster de dos GeForce GTX 960	128000	1170,61	979,07	0,999982872
	256000	2845,35	2136,15	
	512000	11038,7	7571,9	
Tesla C2075	128000	1483,33	1446,53	0,9999549946
	256000	6298,2	5302,11	
	512000	25053,44	21211,22	
MultiGPU con dos Tesla C2075	128000	1012,52	1261,37	0,9999999996
	256000	4045,53	5056,93	
	512000	16090,3	20132,45	
GeForce GTX 560	128000	1202,64	1335,91	0,9999742865
	256000	4829,86	5204,07	
	512000	20102,91	20784,41	
Cluster de dos GeForce GTX 560	128000	1189,3	1369,29	0,9999125316
	256000	4839,47	4963,47	
	512000	19509,6	20656,97	

A continuación, se desea probar el modelo de estimación con N = 1024000. Además, se compara el resultado con la medición de energía real. En la Tabla 4.10 se muestran la energía estimada y la energía medida junto con el coeficiente de correlación entre ambas para la simulación con las distintas arquitecturas GPU utilizadas.

Tabla 4.10 Energía Estimada y Energía Medida en Joules consumida por la simulación utilizando cada arquitectura GPU propuesta con N = 1024000

	GeForce GTX 960	Tesla C2075	GeForce GTX 560
Tiempo Medido	4238,34	93837,44	78967,01
Tiempo Estimado	48457,35	85843,98	94070,15
Coeficiente de correlación		0,99	
	Cluster dos GeForce GTX 960	MultiGPU dos Tesla C2075	Cluster dos GeForce GTX 560
Tiempo Medido	42826,21	55127,49	33613,37
Tiempo Estimado	30736,08	58590,27	40962,68
Coeficiente de correlación		0,99	

Hasta este punto, no es posible hacer una valoración en cuanto al consumo energético de cuál arquitectura convendría para ejecutar la simulación propuesta. Y esto se debe a la forma de medición disponible, que permite medir desde afuera de la máquina, por lo que, no sólo se está obteniendo el consumo energético de la GPU, sino también del resto de los componentes de la PC.

Por lo tanto, para poder comparar las arquitecturas, se realizaron mediciones en modo de reposo, para obtener de esta manera la potencia estática de cada una. En la Tabla 4.11, se pueden observar las potencias estáticas promedio en Watts por segundo para las arquitecturas utilizadas en la experimentación.

Tabla 4.11 Potencia Estática Promedio en Watts por segundos para las distintas arquitecturas GPU

GPU	una PC	dos PC
GeForce GTX 960	69,4	120,2
Tesla C2075	132	132
GeForce GTX 560	97,6	122,1

Se define la Potencia Estática, como: *la Potencia de energía consumida por una máquina en estado ocioso*.

Las Tablas 4.12 y 4.13, muestran la potencia media en Watts por segundo consumidos por la simulación para para 1 y 3 pasos para las distintas arquitecturas GPU.

Tabla 4.12 Potencia Media de Energía sin la potencia estática medida en Watts por segundo para cada arquitectura GPU

	GeForce GTX 960	Tesla C2075	GeForce GTX 560
128000			
1	91,01	144,74	94,22
3	94,28	147,45	98,84
256000			
1	91,44	144,96	95,36
3	96,6	147,26	105,63
512000			
1	92,89	146,51	105,12
3	95,02	146,17	111,32

Tabla 4.13 Potencia Media de Energía sin la potencia estática medida en Watts por segundo para los Cluster de GPU y la MultiGPU

	Cluster dos GeForce GTX 960	MultiGPU dos Tesla C2075	Cluster dos GeForce GTX 560
128000			
1	186,64	244,4	260,27
3	187,35	253,7	261,76
256000			
1	178,97	243,94	268,86
3	180,44	249,4	270,31
512000			
1	185,45	246,39	272,62
3	183,6	256,37	284,61

Si se normalizan las potencias medias de energía de cada arquitectura con respecto a la potencia estática media respectiva, se puede observar cuánto más consume la ejecución de la simulación para cada plataforma experimental. En promedio una GeForce GTX 960, una Tesla C2075, una GeForce GTX 560 y el Cluster de dos GeForce GTX 960 consume dos veces más. Mientras, que la MultiGPU con dos Tesla C2075 al y el Cluster con dos GeForce GTX 560 consumen tres veces más. La Tabla 4.14 muestra cuánto más consume una GPU ejecutando la simulación. La Tabla 4.15, presenta cuánto más consumen los Cluster de GPU y la MultiGPU ejecutando la simulación.

Tabla 4.14 Normalización de la potencia media de energía consumida por la simulación utilizando una GPU

	GeForce GTX 960	Tesla C2075	GeForce GTX 560
128000			
1	2,31	2,09	1,96
3	3,35	2,11	2,01
256000			
1	2,32	2,1	1,98
3	2,39	2,12	2,08
512000			
1	2,34	2,11	2,08
3	2,37	2,11	2,14

Tabla 4.15 Normalización de la potencia media de energía consumida por la simulación utilizando los Cluster de GPU y la MultiGPU

	Cluster dos GeForce GTX 960	MultiGPU dos Tesla C2075	Cluster dos GeForce GTX 560
128000			
1	2,55	2,85	3,13
3	2,55	2,92	3,14
256000			
1	2,5	2,85	3,2
3	2,52	2,89	3,21
512000			
1	2,56	2,87	3,23
3	2,54	2,94	3,33

Capítulo 5

Conclusiones y Trabajos Futuros

En el presente trabajo se muestra el consumo energético de tres diferentes tarjetas gráficas al ejecutar una solución del problema de Alta Demanda Computacional: N Cuerpos. Por un lado, utilizando una GPU de manera individual, y por otro, con un Cluster de GPU y MultiGPU.

A partir de las mediciones de performance y consumo energético presentadas en el capítulo anterior, se concluye que:

- Se propuso un modelo de estimación de tiempo para calcular el tiempo aproximado de ejecución de la simulación utilizada conociendo el tiempo de ejecución de un paso de la simulación para los distintos N. Se concluyó que el tiempo de ejecución de la simulación es dependiente del tamaño de la entrada. Por lo tanto, repetir p cantidad de veces la simulación es aproximadamente igual a ejecutar la simulación una vez, y si ese tiempo se lo multiplica por p se obtendría una estimación del tiempo que tardaría en ser ejecutada la simulación p veces.
- Referido al tiempo de ejecución, la GPU GeForce GTX 960, es la que ejecuta en un menor tiempo el problema. Sin embargo, cualquiera de las tres GPU, consiguen acelerar el cómputo del algoritmo desarrollado. En trabajo previos, puede observarse la aceleración significativa que se obtiene utilizando GPU comparada con las versiones en memoria compartida, memoria distribuida y memoria híbrida [MDDN12] [MDCD+14].
- La aceleración obtenida en cualquiera de las tres GPU utilizadas se encuentra cerca de la aceleración óptima.
- La Potencia Estática Medida, en la MultiGPU consume por segundo aproximadamente 132 W. Mientras que utilizando una PC con una tarjeta gráfica

GeForce 960, ésta consume por segundo 69 W, en el caso de una PC con una placa GeForce 560 los watts consumidos por segundos son 98 W.

- Para el caso del cluster de GPU con dos PC, la potencia estática media con una placa GeForce GTX 960 por máquina, es de aproximadamente 120 W; y en el Cluster de GPU, con una placa GeForce GTX 560 por máquina es de aproximadamente 122 W/s. Se observa que, aun utilizando dos PC para el caso de los cluster, la potencia estática media consumida por segundo es menor que la de la MultiGPU. En la MultiGPU, la refrigeración necesaria para mantener a la máquina en un estado favorable para el correcto funcionamiento de sus componentes insume gran parte de la energía consumida.

- Además, se pudo comprobar lo demostrado en [PSD13], que la cantidad de Watts promedios consumidos por una GPU, un Cluster de GPU o MultiGPU, es independiente del tamaño de la entrada, y la variación que se observa en las mediciones es insignificante. Dicha variación puede deberse a los movimientos de datos y/o a la refrigeración de las placas.

- A partir de lo anterior, se propuso un modelo de estimación de energía conformado por la información de la potencia media de energía para cada arquitectura empleada y el tiempo estimado de ejecución de la simulación para cada N. Con un cierto grado de error, el modelo propuesto permitiría estimar cuánta energía se consumiría y el costo de energía (conociendo el cuadro tarifario correspondiente al suministro eléctrico de la empresa que presta este servicio), pudiendo de esta manera optar por invertir en equipamiento, o contratar, por ejemplo, un servicio de Cloud Computing que resuelva nuestras necesidades dependiendo de los costos que se está dispuesto a pagar o que se pueden pagar.

- Considerando las siguientes características de cada GPU:
    - GeForce GTX 960: la microarquitectura de esta GPU es Nvidia Maxwell, optimizada en performance y eficiencia energética. Es una placa de gama media [GeForce96018] Su valor en el mercado está entre unos $200 a $250 dólares (según el distribuidor).
    - Tesla C2075: basada en la microarquitectura Fermi, promocionada por Nvidia como capaz de brindar el performance de una supercomputadora para Cómputo de Altas Prestaciones [Tesla18]. Su valor en el mercado es de unos $600 dólares.
    - GeForce GTX 560: microarquitectura de base Fermi. No está disponible a la venta en el mercado [GeForce56018].

A partir de los resultados obtenidos, para el problema de Alta Demanda Computacional de los N Cuerpos, y con los escenarios experimentales llevados a cabo, se puede concluir que el uso de una GPU GeForce GTX 960 o de un Cluster de GPU GeForce GTX 960 consigue tanto en performance como en consumo energético las mejores prestaciones. Si se tienen en cuenta los costos de las distintas GPU utilizadas, la mejor opción sigue siendo la GeForce GTX 960.

- Al normalizar las potencias promedio con la potencia base de cada máquina respectivamente, se puede ver que para el caso del uso de una GPU para todas las placas utilizadas el consumo es dos veces más al ejecutar el algoritmo de los N cuerpos. Para el caso del uso de dos placas el consumo por segundos en watts es 3 veces mayor, en la MultiGPU y en el Cluster de GPU con placas GeForce GTX 560. Para el caso del Cluster de GPU de tarjetas gráficas GeForce GTX 960, el consumo energético es de aproximadamente 2.5 watts por segundo más en promedio.
- Para cada tipo de placa se puede observar que la energía consumida por segundo en Joules escala. Y que la placa que menos consume en líneas generales es la GeForce 960 para todos los tamaños del problema probados.

Comentado [M8]: CREO QUE ANTES DE LOS TRABAJOS FUTUROS HABRIA QUE HACER UNA RELACION ENTRE LOS RESULTADOS OBTENIDOS PRESTACIONAL Y ENERGETICAMENTE…

Se puede llegar a la conclusión que, tanto en performance como en consumo energético, la GeForce GTX 960 y el Cluster con dos GeForce GTX 960 son las plataformas GPU óptimas para la ejecución de la simulación de los N Cuerpos desarrollada para el presente trabajo.

El modelo de estimación de energía propuesto, basado en el tiempo de ejecución de la simulación de los N Cuerpos y la potencia de cada GPU utilizada, logró estimar energía con un coeficiente de correlación de 0,99, para todos los escenarios experimentales dispuestos.

Como trabajos futuros se puede mencionar:

- Utilizando el mismo algoritmo de N cuerpos, medir consumo energético en más de dos nodos para los Cluster de GPU con GeForce GTX 560 y GeForce GTX 960.
- Estudiar y utilizar NVIDIA Management Library (nvml), que permite monitorear y administrar varios estados de los dispositivos NVIDIA GPU. Proporciona un acceso directo a las consultas y comandos expuestos a través de nvidia-smi. Entre las consultas que se pueden realizar, se encuentra la administración de la energía en la placa. La misma está disponible para la Tesla C2075 (no así para las otras placas: GeForce GTX 560 y 960). La consulta devuelve como resultado la potencia instantánea consumida por la tarjeta gráfica.

- Realizar análisis de consumo energético referidas a otro tipo de simulación, tales como simulaciones orientadas a individuos.

Apéndice A

A.1. Tiempos de ejecución medidos en segundos para la simulación utilizando una GPU

Tabla A.1. Tiempos de ejecución medidos en segundos para la simulación utilizando una GPU

Una GPU			
pasos	GeForce GTX 960	Tesla C2075	GeForce GTX 560
128000			
1	5,23	5,36	6,42
3	13,54	16,10	18,91
256000			
1	16,16	22,74	24,88
3	49,33	63,90	75,68
512000			
1	64,64	90,95	99,09
3	194,58	252,87	297,45
1024000			
1	259,12	337,33	396,04
3	777,35	1011,98	1190,70

A.2. Tiempos de ejecución medidos en segundos para la simulación utilizando Cluster de GPU y MultiGPU

Tabla A.2. Tiempos de ejecución medidos en segundos de la simulación utilizando Cluster de dos GPU y MultiGPU

Dos GPU			
pasos	GeForce GTX 960	Tesla C2075	GeForce GTX 560
128000			
1	3,69	2,69	3,46
3	7,52	8,08	9,93
256000			
1	9,42	10,74	12,48
3	26,83	32,21	37,44
512000			
1	35,18	42,59	49,75
3	130,59	85,18	149,44
1024000			
1	135.62	168,58	198,17
3	409,22	506,90	594,71

Apéndice B

B.1 Potencia promedio de energía medidas en Watts por segundo para la simulación utilizando una GPU, Cluster de GPU y MultiGPU

Tabla B.1.1. Potencia promedio de energía en Watts por segundo por la simulación en una y dos GPU, para: GeForce GTX 960

GeForce GTX 960			
Una GPU			
pasos	128000	256000	512000
1	160,48	160,91	162,36
3	163,75	166,07	164,49
Dos GPU			
1	306,895	315,62	321,135
3	307,609	320,42	322,897

Tabla B.1.2. Potencia promedio de energía en Watts por segundo por la simulación en una y dos GPU, para: Tesla C2075

Tesla C2075			
Una GPU			
pasos	128000	256000	512000
1	276,74	276,96	278,51
3	279,45	279,26	278,17
Dos GPU			
1	376,40	375,94	378,39
3	385,71	381,40	388,37

Tabla B.1.3. Potencia promedio de energía en Watts por segundo por la simulación en una y dos GPU, para: GeForce GTX 560

GeForce GTX 560			
Una GPU			
pasos	128000	256000	512000
1	191,87	193,01	202,77
3	196,49	203,28	208,97
Dos GPU			
1	382,29	390,88	394,64
3	383,78	392,33	406,63

Tabla B.1.4. Potencia promedio de energía en Watts por segundo consumida por la simulación en una y dos GPU para las distintas arquitecturas empleadas.

una GPU 1024000		
GeForce GTX 960	Tesla C2075	GeForce GTX 560
163,54	278,37	209,09
Cluster de dos GPU / MultiGPU		
325,86	398,84	407,76

B.2 Energía medida en Joules por segundo para la simulación utilizando una GPU, Cluster de GPU y MultiGPU

Tabla B.2.1. Energía consumida en Joules por la simulación en la GPU GeForce GTX 960

GeForce GTX 960			
Una GPU			
pasos	128000	256000	512000
1	1133,67	2600,85	10512,38
3	2208,02	8045,07	31810,36
Dos GPU			
1	927,35	2528,64	10453,66
3	2133,36	8165,9	59232,72

Tabla B.2.2. Energía consumida en Joules por la simulación en la GPU Tesla C2075

Tesla C2075			
Una GPU			
pasos	128000	256000	512000
1	1483,33	6298,2	25053,44
3	4499,19	17844,18	70290,5
Dos GPU			
1	1012,53	4045,54	16090,3
3	3116,50	12345,77	33628,23

Tabla B.2.3. Energía consumida en Joules por la simulación en la GPU GeForce GTX 560

GeForce GTX 560			
Una GPU			
pasos	128000	256000	512000
1	1202,65	4829,87	20102,91
3	3696,98	3124,33	62266,04
Dos GPU			
1	1189,3	4839,48	19509,61
3	3677,73	14858,75	60597,44

Tabla B.2.4. Energía consumida en Joules por la simulación en las distintas arquitecturas GPU empleadas

una GPU 1024000		
GeForce GTX 960	Tesla C2075	GeForce GTX 560
42396,78	94027,32	82955,45
Cluster de dos GPU / MultiGPU		
44196,52	67227,84	81046,93

Referencias

[Al08] Pablo Alcalde San Miguel, “Electrotecnia”, Ed. Paraninfo, 2008.

[Amd11] amd.com/public, “Meeting the challenges of the future with innovative solutions for public sector IT needs”, 2011.

[ANAA+17] AlMusbahi, Nahhas, AlMuhammadi, Anderkairi, Hemalatha, “Survey on Green Computing: Vision and Challenges”, Interenational Journal of Computer Applicatios. Vol 167, nro 10, junio 2017.

[And00] Andrews Gregory R., “Foundations of Multithreaded, Parallel, and Distributed Programming”, Addison-Wesley, 2000.

[ArGob18] “Convenio para uso eficiente de la energía en empresas”. www.argentina.gob.ar/noticias/convenio-para-uso-eficiente-de-energia-en-empresas-0 fecha de acceso Febrero 2018

[ArMin17] Ministerio de Educación, Ministerio de Energía y Minería, Presidencia de La Nación, “Lineamientos para la mejora de la enseñanza sobre eficiencia energética en carreras estratégicas de Ingeniería y Arquitectura”. Octubre 2017

[ArPre18] www.argentina.gob.ar/premio-argentina-eficiente fecha de acceso febrero 2018

[Béd07] Jeroen Bédorf, “High Performance Direct Gravitational N -body Simulations on Graphics Processing Units”, Universiteit van Amsterdam, 2007.

[BGP12] Bódorf J., Gaburov E., Portegies Zwart S., “Bonsai: A GPU Tree-Code”, Leiden Observatory, Leiden University and Northwestern University, ASP Conference Series, Astronomical Society of the Pacific, 2012.

[BMRD+++16] Baladini Javier, Morán Marina, Rozas Claudia, De Giusti Armando, Suppi Remo, Rexachs Dolores, Luque Emilio, "Consumo energético de sistemas de cómputos de altas prestaciones" 2016.

[Bru11] Bruzzone Sebastian, LFN10, LFN10-OMP y el Método de Leapfrog en el Problema de los N Cuerpos, Instituto de Física, Departamento de Astronomia, Universidad de la República y Observatorio Astronómico los Molinos, Uruguy (2011)

[Bue88] Bueche Frederick “Ciencias Físicas” Ed. Reverté. 1988.

[CS07] Colque Pinelo MaTeresa y Sánchez Campos Víctor E., “Los Gases de Efecto Invernadero: ¿Por qué se produce el Calentamiento Global?”, Asociación Civil Labor / Amigos de la Tierra – Perú. 2007.

[CUDA18] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html fecha de acceso: febrero 2018

[Cur08] Curtis Lewis, “Environmentally Sustainable Infrastructure Design”, The Architecture Journal, págs. 2-8, 2008.

[CGef18] www.geforce.com/hardware/technology/cuda fecha de acceso: febrero 2018.

[CGSP04] 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.

[DACC++16] Díaz Javier, Ambrosi Viviana, Castro Nestor, Candia Damian, Vega Edgar, Rodriguez Anahí, "Experiencia de la enseñanza de Green IT en la currícula de carreras de Informática de la UNLP ". XI Congreso de Educación en Tecnología y Tecnología en Educación. 9 y 10 de junio de 2016

[Devt18]https://devtalk.nvidia.com/default/topic/808106/question-abount-cudasetdevice-and-multiple-host-threads fecha de acceso febrero 2018.

[DGT98] De Giusti Armando y Tinetti Fernando G., “Procesamiento Paralelo. Conceptos de Arquitecturas y Algoritmos”, Primera Edición, 1998.

[DS01] David H. y Stephen P. H. Clark's **The Suppressed Scientific Discoveries of Stephen Gray and John Flamsteed, Newton's Tyranny**, *W. H. Freeman and Co.*, 2001

[EI07] Energy Information Administration, “Emissions of Greenhouse Gases in the United States 2006”, año 2007.

[FG08] Francis Kevin y Richardson Peter, Green Maturity Model for Virtualization, The Architecture Journal, págs. 9-15. 2008

[FZ08] Gerhard Fettweis, Ernesto Zimmermann, “ICT ENERGY CONSUMPTION – TRENDS AND CHALLENGES”. The 11th International Symposium on Wireless Personal Multimedia Communications (WPMC 2008) 2008.

[GeForce56018] https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-560/buy-gpu accedido abril 2018

[GeForce96018] http://www.nvidia.es/object/geforce-gtx-960-es.html#pdpContent=0 accedido abril 2018

[GGKK03] Grama Ananth, Gupta Anshul, Karypis George y Kumar Vipin, “Introduction to Parallel Computing”, Segunda Edición, Adison-Wesley, 2003.

[Gan13] Ganot, Maneuvrier, “Tratado Elemental de Física”. Paris Imprenta de la Vta de C. Bouret. Vigésima Quinta Edición. 1913.

[Ger03] Joseph L. Gerver, Noncollision Singularities: Do Four Bodies Suffice? Exp. Math. (2003), 187-198.

[GM18] Gallo S., Montes de Oca E., "Reporte técnico: Medición de energía con Osciloscopio Rigol DS1074Z “, III LIDI Instituto de Investigación en Informática LIDI, Facultad de Informática, UNLP. 2018.

[GPVA14] Christian González, Ramón Pérez, Carmen Vásquez Stanescu, Gustavo Araujo, “EFICIENCIA ENERGÉTICA USO RACIONAL DE LA ENERGÍA ELÉCTRICA EN EL SECTOR ADMINISTRATIVO”. Ministerio del Poder Popular para la Energía Eléctrica. Municipio Libertador, Distrito Capital República Bolivariana de Venezuela. 2014

[Green18] http://www.green500.org fecha de acceso 2018

[Guh08] Guhl Corpas Andrés, “Aspectos éticos del calentamiento climático global”. Revista Latinoamericana de Bioética, ISSN 1657-4702, Volumen 8, Número 2, Edición 15, Páginas 20-29, 2008.

[HB84] Hwang Kai y Briggs Fayé A., “Computer Architecture and Parallel Proccessing”, MacGraw-Hill Book Company, 1984.

[Hem10] Scott Hemmert, “Green HPC From Nice to Necessity”, Sandia National Laboratories, 1521-9615/10/, IEEE, 2010.

[HYNN++09] Tsuyoshi Hamada, Rio Yokota, Keigo Nitadori, Tetsu Narumi , Kenji Yasuoka, Makoto Taiji , “42 TFlops Hierarchical N-body Simulations on GPUs with Applications in both Astrophysics and Turbulence ” , www.stats.bristol.ac.uk, 2009.

[Hwu18] Wen-Mei W. Hwu, “GPU Computing Gems Esmerald Edition. Applications of GPU Computing” Ed. Morgan Kaufmann Publ Incorporated 2018.

[KJF11] Alexander Kipp, Tao Jiang, Mariagrazia Fugini, “Green Metrics for energy-aware IT systems”, 2011 International Conference on Complex, Intelligent, and Software Intensive Systems, 978-0-7695-4373-4/11, IEEE, 2011

[MDDN12] Montes de Oca E., De Giusti L., De Giusti A., Naiouf M., “Comparación del uso de GPU y cluster de multicore en problemas con alta demanda computacional l”. XII Workshop de Procesamiento Distribuido y Paralelo. CACIC2012. Bahía Blanca, Buenos Aires, Argentina, octubre 2012.

[Mdo13] Montes de Oca Erica, “Comparación del uso de GPGPU y cluster de multicore en problemas con alta demanda computacional”. Tesina de Grado, Facultad de Informática, Universidad Nacional de La Plata. Julio 2013.

[MDCD+14] Montes de Oca E., De Giusti L., Chichizola F., De Giusti A., Naiouf M., “Utilización de Cluster de GPU en HPC. Un caso de estudio”. IVX Workshop de Procesamiento Distribuido y Paralelo. CACIC2014. ISBN 978-987-3806-05-6. La Matanza, Buenos Aires, Argentina, octubre 2014.

[MDCD+16] Montes de Oca E., De Giusti L., Chichizola F., De Giusti A., Naiouf M., “Análisis de uso de un algoritmo de balanceo de carga estático en un Cluster Multi-GPU Heterogéneo”. XV Workshop de Procesamiento Distribuido y Paralelo. CACIC2016. San Luis, San Luis, Argentina, octubre 2016.

[ML02] Markoff John, Lohr Steve, "Intel´s huge bet turns iffy." New York Times, septiembre 29, 2002.

[MP15] Adriana Norma Martínez y Adriana Margarita Porcelli. "Impactos de las tecnologías informáticas en el Ambiente y nuevas tendencias en computación verde ". Diario DPI Suplemento Derecho y Tecnologías Nº 11 -30.12.2015

[NA11] NASA. “Global Climate Change”. http://climate.nasa.gov/causes/

[NDev18] https://devblogs.nvidia.com/how-build-gpu-accelerated-research-cluster fecha de acceso febrero 2018.

[NG08] National Geographic Channel, “Seis grados que podrían cambiar el mundo. Las consecuencias del calentamiento global”, http://www.natgeo.tv/la/especiales/seis-grados/

[Ngpu18] www.nvidia.com/object/what-is-gpu-computing.html fecha de acceso febrero 2018

[NPas18] www.nvidia.es/graphics-cards/geforce/pascal. Fecha de acceso febrero 2018.

[NTs18] la.nvidia.com/object/tesla-feactures-la.hmtl fecha de acceso febrero de 2018

[Nvi18] www.nvidia.es/object/cuda-parallel-computing-es.html fecha de acceso: febrero 2018.

[NTK18] www.nvidia.es/object/tesla-gpu-supercomputer-record-jan-30-2013-es.html fecha de acceso 2018

[NU98] Naciones Unidas, “PROTOCOLO DE KYOTO DE LA CONVENCIÓN MARCO DE LAS NACIONES UNIDAS SOBRE EL CAMBIO CLIMÁTICO”. 1998.

[PM15] Adriana Margarita Porcelli, Adriana Norma Martínez, "La nueva economía del siglo XXI: análisis de los impactos de la informática en el ambiente. Tendencias actuales en tecnologías informáticas verdes, un compromiso con la sustentabilidad". Quaestio

Iuris. vol. 08, nº. 04, Número Especial. Rio de Janeiro, 2015. pp. 2174-2208 DOI: 10.12957/rqi.2015.20953. 2015

[Pur88] Purcell Edward M., "Electricidad y Magnetismo". 2da Edición. Ed. Reverté. 1988

[PSD13] Adrian Pousa, Victoria Sanz, Armando De Giusti. Análisis de rendimiento de un algoritmo de criptografía simétrica sobre GPU y Cluster de GPU. Instituto de Investigación en Informática LIDI, Fac. de Informática, UNLP. HPC La TAM 2013. (2013)

[QRAM+05] Quintela, Redondo, Arévalo, Melchor, Redondo. "Carga de una batería y electricidad, dos términos de utilización confusa" Técnica Industrial, junio 2005.

[RAE18] Real Academia Española, Diccionario de la Real Academia Española, 2018.

[Rigol18] "RIGOL User's Guide MSO1000Z/DS1000Z Series Digital Oscilloscope". http://beyondmeasure.rigoltech.com/acton/attachment/1579/f-050a/1/-/-/-/-/MSO1000Z%26DS1000Z_UserGuide.pdf

[Saa77] Saari, Donald G. (1977). «A global existence theorem for the four-body problem of Newtonian mechanics». *J. Differential Equations* **26**: 80-111. Bibcode:1977JDE....26...80S. doi:10.1016/0022-0396(77)90100-0.

[Sap53] Sapiens, Enciclopedia ilustrada de la lengua castellana, Editoral Sopena Argentina, 1953.

[Sha17] blogonparallelcomputing.wordpress. Multi-GPU or GPU Cluster. Abril 2017.

[SKHZ+++16] MARK SILBERSTEIN, SANGMAN KIM, SEONGGU HUH, XINYA ZHANG, YIGE HU, AMIR WATED, EMMETT WITCHEL, "GPUnet: Networking Abstractions for GPU Programs", Transactions on Computer Systems, Vol. 34, No. 3, Article 9, Publication date: September 2016, ACM 0734-2071/2016/09, 2016

[Ste01] Stewart, I. (2001). *¿Juega Dios a los dados?* Barcelona: crítica. ISBN 978-84-8432-881-0.

[Tesla8] https://www.nvidia.com/docs/IO/43395/NV-DS-Tesla-C2075.pdf accedido abril 2018

[TK10] Tsuyoshi Hamada, Keigo Nitadori: 190 TFlops Astrophysical N-body Simulation on cluster of GPUs. Universidad de Nagasaki. IEEE 978-1-4244-7558-2 (2010)

[TM10] Tipler, Mosca, "Física para la Ciencia y la Tecnología. Volumen 2 Electricidad y Magnetismo/Luz" 6ta Edición. Ed. Reverté. 2010.

[Top18] www.top500.org/list/2017/11 fecha de acceso febrero 2018.

[TW09] Mujtaba Talebi, Thomas Way, “Methods, Metrics and Motivation for a Green Computer Science Program”. SIGCSE’09, March 3–7, 2009, Chattanooga, Tennessee, USA. 2009.

[US11] U.S. Environmental Protection Agency, “INVENTORY OF U.S. GREENHOUSE GAS EMISSIONS AND SINKS: 1990 –2009”, año 2011.

[Val14] Edgar Valdés Castro, "Tecnologías de información que contribuyen con las prácticas de Green IT ". RECIBIDO: ENERO 5, 2014; ACEPTADO: MARZO 15, 2014 Ingenium, 8(19), 11-26

[Xia92] Zhihong Xia. The Existence of Noncollision Singularities in Newtonian Systems. Annals of Mathematics. Second Series, Vol. 135, No. 3 (Mayo, 1992), pp. 411-468

[YB10] Rio Yokota, Lorena A. Barba, “Treecode and fast multipole method for N-body simulation with CUDA”, Boston University, http:// www.bu.edu, 2010.