Resumen.

En este proyecto de investigación está enfocado primeramente en desarrollar técnicas de optimización basadas en información de bajo nivel obtenida directamente del hardware de las unidades de procesamiento gráfico (conocido como GPU), que puedan ser aplicadas a todo tipo de algoritmos científicos en una o varias computadoras con GPU conectadas en red.

Se investigó cómo los indicadores de hardware de la GPU proveen dicha información, investigamos cuál es el efecto de cada indicador por separado o de varios en conjunto sobre el rendimiento de los algoritmos ejecutados. Para esto utilizamos herramientas de profiling provistas por los fabricantes de las GPU. Luego se probaron optimizaciones en distinto algoritmos científicos tales como el N-Body para simulaciones gravitatorias, Schönhage-Strassen para multiplicación de enteros, y SGP4/SDP4 para predicción de posición orbital de satélites artificiales, BLAS en matrices dispersas, entre otros. Se evaluó en la etapa final cuales son los indicadores y técnicas de optimización comunes a los diferentes algoritmos, y una guía de optimización utilizando indicadores de hardware de GPU que pueda ser utilizada por científicos y programadores de cualquier área tanto para una sola computadora, como para un clúster de computadoras con GPU conectadas por red.

Palabras Clave: Profiling de hardware, GP-GPU, Optimización, Matrices Dispersas

1 Estructura

En este apartado se presenta la estructura del presente informe la cual toma como base la propuesta en forma general de la guía de informes finales, realizando agregados (indicándose con una –A– después del título) o quitando aquellos títulos que no aplican en la presente temática (indicándose con una –NA– después del título).

2 Introducción.

2.1 Selección del Tema.

La computación científica es utilizada en los más variados y vastos de campos incluyendo la ingeniería, la medicina, la física, la astronomía y la matemática. Estas disciplinas requieren siempre contar con la máxima potencia de procesamiento posible para realizar sus cálculos. Cuanto menos tiempo demore una computadora en ejecutar un algoritmo científico, mejores resultados se podrán obtener de su uso.

En las últimas décadas se han comenzado a desarrollar nuevas arquitecturas de procesamiento que dejan atrás el enfoque clásico de optimización de algoritmos secuenciales. En ellos la mejora, de la prestación, es lograda por un gran esfuerzo de parte del grupo de desarrolladores en realizar la menor cantidad acciones (ejecutar menos instrucciones) posible, logrando el mismo resultado. Gracias a los avances en la tecnología han logrado introducir arquitecturas que permiten la ejecución de algoritmos con niveles de paralelismo nunca antes visto. Por lo tanto, el esfuerzo en la búsqueda de optimizar algoritmos, también debe enfocarse en poder utilizar el potencial de arquitecturas paralelas.

La evolución de los sistemas con multiprocesadores ha seguido dos líneas de desarrollo: las arquitecturas *multicore* (multi-núcleos) y las arquitecturas *many-cores* (muchos núcleos o muchos *cores*). En el primer caso incorporaron varios núcleos de procesamiento, tomando la idea de las supercomputadoras ya existentes y surgieron las computadoras de 2, 3, 4, 8 y más procesadores por unidad central con el objetivo de acelerar las aplicaciones.

En el segundo caso los avances en la tecnología han permitido desarrollar arquitecturas denominadas *many-core* en las que son reemplazados los complejos núcleos convencionales del procesador, por una gran cantidad de núcleos simples y se centran en optimizar el desempeño de las aplicaciones, dentro de este tipo se encuentran las Unidades de Procesamiento Gráfico [1]. Entre estas arquitecturas se encuentra el modelo de GP-GPU (*General Purpose GPU*), utilizada en esta investigación.

Entre los años 1990 y 2000 Nvidia introduce la primera GPU o unidad de procesamiento gráfico del mercado, luego científicos e investigadores de disciplinas como el diagnóstico por imagen o electromagnetismo comenzaron a utilizar las GPU para aplicaciones de cálculo de propósito general y descubrieron un gran rendimiento en operaciones de punto flotante, obteniendo un gran aumento de velocidad de ejecución en una gran variedad de aplicaciones científicas, naciendo ahí el concepto de GP-GPU. Sobre éstas últimas existen guías de cómo aprovechar el uso de estas arquitecturas para obtener el mejor rendimiento, sin embargo ninguna de ellas contempla aprovechar la información provista por los contadores de hardware para mejorar la respuesta de los algoritmos científicos en las áreas que detallamos anteriormente. Nuestra investigación propone sentar nuevas bases para ampliar el conjunto de recomendaciones (guía de buenas prácticas) sobre su uso, basándose en la información provista por los contadores de hardware.

2.2 Definición del Problema.

Ha existido una gran cantidad de publicaciones referidas a la utilización del paradigma de paralelismo convencional (es decir, una o más computadoras con varios núcleos convencionales) que buscan establecer un paradigma de programación que permita determinar qué prácticas arrojarán siempre mejores resultados. Sin embargo, el problema con las arquitecturas paralelas es su imprevisibilidad. No es posible conocer a priori cómo se ejecutaran las tareas, aun si el código es el mismo. Por lo tanto, respecto al paralelismo convencional, sólo se han recopilado guías de buenas prácticas que, adaptadas a cada problema, pueden optimizar la ejecución de un algoritmo.

2.3 Justificación del Estudio

Dada que la tecnología es relativamente nueva, los fabricantes de dispositivos GPU, así como investigadores independientes han publicado varias experiencias (historias de éxito), las mejores prácticas y guías de optimización para ayudar a los desarrolladores para obtener el máximo rendimiento del programa [2] [3].Sin embargo, todavía hay poca información acerca de las técnicas aplicadas para posibles optimizaciones que pueden aprovecharse mediante el análisis de los contadores de rendimiento de hardware, provisto para las arquitecturas GPUs,y además estos varían de acuerdo al fabricante. En este trabajo se trata de sentar las bases para que científicos y desarrolladores de todas las áreas puedan contar con nuevas técnicas de optimización de algoritmos para las arquitecturas GPU, y de esta manera lograr ejecuciones más eficientes cuando éstos tengan acceso a herramientas de *profiling* y mediciones de indicadores de hardware.

2.4 Limitaciones.

Está investigación se limita a las herramientas de perfilado de hardware provistas por los dos fabricantes más importantes, en placas de procesamiento gráfico similares, compatibles con las Geforce GTX650 y ATI Radeon R7 serie 200, permitiendo la generación de una guía de optimizaciones acordes al hardware utilizado, sin la utilización de los indicadores que posee la CPU.

2.5 Alcances del Trabajo

Los indicadores de hardware analizados en la presente investigación, son únicamente los que proveen los procesadores gráficos presente en las computadoras del laboratorio. Se detallan en el ANEXO F.

La guía se incluyen aquellos contadores hardware que mide un comportamiento y puede ser mejorado, por optimizaciones, en forma repetible.

La guía no cuantifica cuanto es la mejora, ya que eso depende del comportamiento de cada procesador gráfico y volumen de datos a procesar.

Las pruebas en *cluster* de los lineamientos de guía está acotada a conseguir el equipamiento disponible en el Laboratorio 266.

2.6 Objetivos

Los objetivos propuestos de la investigación son:

- Investigar sobre los diferentes indicadores de hardware y herramientas de *profiling* existentes para unidades de procesamiento gráfico (GPU).

- Confeccionar una guía de optimización de algoritmos científicos que utilizan las unidades de procesamiento gráfico (GPU).

- Realizar la extrapolación de los ítems de la guía de optimización aplicándolas en un entorno de clusters de dos o más computadoras con GPU interconectadas por red.

2.7 Hipótesis

Se han desarrollado guías sobre "buenas prácticas" para mejorar el rendimiento de algoritmos de ejecución paralela sobre GPU [2] y técnicas específicas para mejorar algoritmos de cierto tipo (por ejemplo: [4] [5] para algoritmos de manejo de grandes cantidades de datos en GPU, y para algoritmos tipo N-Body [5] [6]). Existen muy pocas publicaciones en la actualidad que buscan establecer técnicas de optimización que puedan mejorar el desempeño de algoritmos científicos en general, basándose exclusivamente en los contadores de hardware provistos por herramientas de *profiling*, que proveen un gran potencial para la optimización de algoritmos en GPU ya que las herramientas de *profiling* son agnósticas del problema [7]

3 Desarrollo

3.1 Material y método.

3.1.1 Estudio de la problemática y metodología

Los desarrollos científicos dependen de la potencia de cálculo para resolver o simular problemas científicos y están enfocando su atención sobre tecnologías de múltiples núcleos, la que ha seguido dos líneas de desarrollo: Las arquitecturas multicore (multi-núcleos) y las arquitecturas many-cores (muchos-núcleos o muchos-cores), las primeras se centraron el desarrollo de mejoras, con el objetivo de acelerar las aplicaciones, que dedican más transistores para permitir instrucciones complejas, predicción de saltos, y el paralelismo a nivel de instrucción para cada núcleo de procesamiento. Debido a los límites alcanzados por las computadoras personales, surgieron las computadoras de 2, 4 ,8 y más procesadores por unidad Central, Los multicores comenzaron con sistemas de 2 núcleos y continuaron con procesadores como el Intel Xeon Phi que posee más de 60 núcleos.

Las arquitecturas many-core, los desarrollos están centrados en optimizar el desempeño de las aplicaciones. Las Unidades de Procesamiento Gráfico (Graphics Processing Unit, GPU). Las arquitecturas many-core [8] le llevaron ventaja a los procesadores de propósito general principalmente respecto a las mejoras de velocidad que partir del 2009 fue de 10 a 1 (GPU.CPU), 1 TB versus 100GB.

La diferencia de rendimiento tan grande se debe a la filosofía de diseño de ambos figura 1-, las multi-core están orientadas al mejor desempeño de las soluciones secuenciales, es por ello que la optimización se basan en proveer lógica de control compleja para ejecución paralela de código secuencial o la inclusión de memorias caché más rápidas para disminuir la latencia de las instrucciones. En cambio en las many-core la idea subyacente es optimizar el desempeño de muchos threads ejecutando en paralelo de manera tal que si alguno de ellos está esperando por la finalización de una operación , se le asigne trabajo y no permanezca ocioso. Poseen memorias caché pequeñas, cuya función es ayudar a mantener el ancho de banda definido para todos los threads paralelos. Lo que determina que esta arquitectura está en su mayor parte dedicada a cómputo y no a técnicas para disminuir la latencia**.**

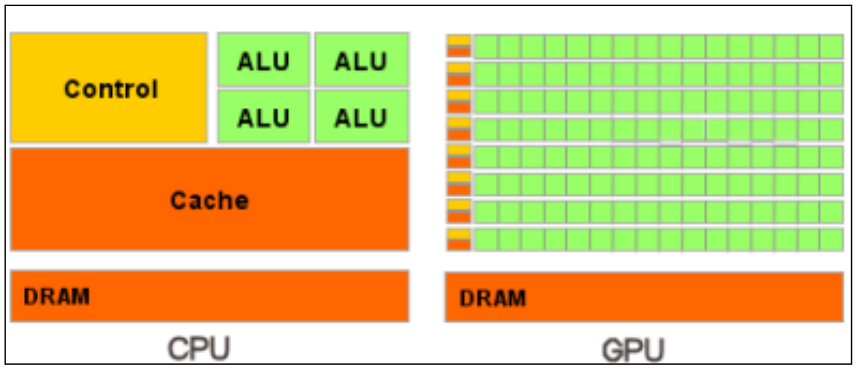


Figura 1 - Diseño de las arquitecturas.

Todo lo expuesto anteriormente fueron mejoras realizadas en el hardware para mejorar el rendimiento teórico en los procesadores, pero además es necesario utilizar un paradigma de programación paralela conjuntamente con Pthread o OpenMP, por lo que resulta necesario una metodología de diseño e implementación de aplicaciones paralelas.

En procesadores *multi-core* aplicaciones pueden escalar casi linealmente con un número bajo de *cores* (2-4-8) a medida que aumenta el número de cores deja de ser lineal, debido a factores como:

- Overhead por la creación o eliminación de threads.

- Desbalanceo (en las aplicaciones), debido a que se produce una incorrecta distribución del volumen de cómputo por Thread. Algunos terminan su trabajo antes que otros, teniendo que esperar a la finalización del resto. Lo que supone un coste de procesador desaprovechado y por lo tanto un overhead. El cual se puede reducir pero no eliminar totalmente asignado trabajo dinámicamente.

- Las comunicaciones entre las memorias de los cores, dependiendo si ejecutan en el mismo procesador o en procesadores diferentes.

Herramientas de Software para programación paralela. OpenMP

Es una Api que soporta programación paralela multithread en arquitecturas de memoria compartida. Puede ser utilizada con Fortran (77, 90, 95), C, C++. Define un modelo portátil, escalable con una interfaz sencilla y flexible para el desarrollo de aplicaciones paralelas en las plataformas desde computadoras de escritorio hasta el supercomputadoras
Está basado en paralelismo a nivel de Thread utilizando memoria compartida
Utiliza el paradigma fork-join, donde un programa OpenMP comienza con un único thread (el master thread) que realiza ejecución secuencial hasta llegar a la región paralela, donde creará un conjunto de threads (fork), que se ejecutarán en paralelo. Cuando el conjunto de threads finaliza el trabajo se sincronizan con (join), quedando únicamente el master thread.
Es un modelo explícito en el cual el programador tiene el control total sobre la paralelización.
Basado en directivas de compilación que se incluyen en el código fuente.
Soporta :
Paralelismo a nivel de bucle (paralelismo de Datos)

Paralelismo Funcional
Regiones Críticas (Exclusión Mutua)
Regiones ejecutas por un único Thread (Regiones Single)
Variables públicas y privadas.
Sincronizaciones explicitas.
Reparto estático y dinámico de trabajos.
[i], [ii], [iii]
- ANEXO C-

OpenMPI

Se trata de una API de código abierto desarrollada para facilitar la programación paralela y/o distribuida que:
● Implementa el estándar MPI.
●Permite la distribución de procesos de forma dinámica.
●Alto rendimiento.
●Tolerancia a fallos: capacidad de recuperarse de forma transparente de los fallos de los componentes (errores en el envío o recepción de mensajes, fallo de un procesador o nodo).
●Soporta redes heterogéneas: permite la ejecución de programas en redes de computadoras con distinto número de nodos y de procesadores.
●Una única biblioteca soporta todas las redes.
●Portable: funciona en los sistemas operativos Linux, OS-X, Solaris y en un futuro próximo en Windows.
●Modificable por los instaladores y usuarios finales: presenta opciones de configuración durante la instalación de la API, la compilación de programas y su ejecución).
[iv] **Ver -ANEXO C-**

PAPI (Performance Application Programming Interface)

PAPI es una API para acceder a los contadores de rendimiento de Hardware, disponibles en la mayoría de los procesadores actuales. Estos contadores son un conjunto de registros que cuentan la ocurrencia de determinados eventos en el procesador.
Dependiendo del sistema operativo empleado, puede ser necesario, aplicar algún parche (PerfCtr o perfmon2) y recompilar el kernel.
Al incorporar PAPI a una sección de código, es posible pasar como parámetro a la aplicación, los nombres de los contadores de hardware de los que se quiere obtener información.
Lista de algunos contadores más utilizados está en el -**ANEXO C-.** y en Página Oficial [vi]

CUDA - Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo)
En el año 2006 NVIDIA presentó una tecnología que propone una filosofía integradora con un lenguaje de programación genérico como el C y la arquitectura paralela de una GPU. La tecnología CUDA permite considerar a la GPU como una arquitectura paralela para la resolución de problemas de propósito general [9]. Este desarrollo de aplicaciones paralelas es posible debido a dos razones:
Las tarjetas gráficas NVIDIA, son una componente común en las computadoras personales actuales.
Es de fácil aprendizaje para los programadores que conocen C o C++, pudiendo mediante wrappers usar Python, Fortran y Java en vez de C/C++, a los que se está por incorporar OpenGL y Direct3D.
Página Oficial de CUDA [vii]

OpenCL ™ - Open Computing Language

OpenCL (Open Computing Language) es un estándar abierto y libre para la programación de propósito general paralelo a través de CPUs, GPUs y otros procesadores, dando a los desarrolladores de software de acceso portátil y eficiente al poder de estas plataformas de procesamiento heterogéneos.

OpenCL compatible con una amplia gama de aplicaciones, que van desde software embebido y de los consumidores, a través de soluciones HPC de bajo nivel, de alto rendimiento y abstracciones portables. Mediante la creación de una interfaz de programación eficiente, close-to-the-metal, OpenCL formará la capa base de un ecosistema de computación paralela de plataforma independiente de herramientas, middleware y aplicaciones.

OpenCL consiste en una API para la coordinación de cómputo paralelo a través de procesadores heterogéneos; y un lenguaje de programación multiplataforma con un entorno de computación bien especificado.

El estándar OpenCL:

- Soporta tanto los modelos de programación paralela de datos y basados en tareas.
- Utiliza un subconjunto de ISO C99 con extensiones para el paralelismo.
- Define requisitos numéricos consistentes basados en IEEE 754.
- Define un perfil de configuración para dispositivos de mano y embebidos
- Eficientemente interopera con OpenGL, OpenGL ES, y otras APIs gráficas

La especificación se divide en una especificación básica que cualquier aplicación compatible con OpenCL debe soportar; un perfil de mano / incrustado que relaja los requisitos de cumplimiento de OpenCL para dispositivos manuales y embebidos; y un conjunto de extensiones opcionales que pueden pasar a la especificación principal en revisiones posteriores de la especificación OpenCL se encuentran en la Página Oficial [viii]

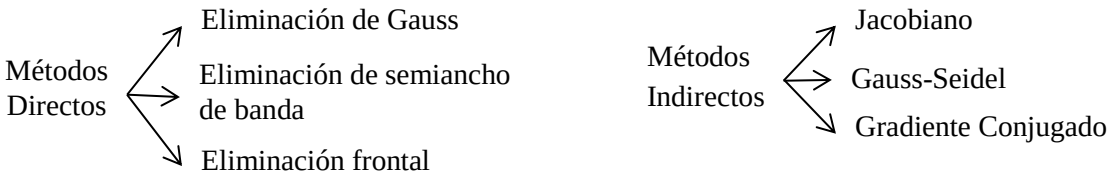
Algoritmos

Matrices dispersas

Las matrices dispersas son utilizadas para representar elementos de análisis en campos de aplicación como computación científica, en análisis estructural y de circuitos, sobre grafos, redes de interconexión, elementos finitos, dinámica computacional de fluidos, entre otros.

El método de elementos finitos es una técnica numérica que resuelve problemas de energía, ecuaciones de diferenciales complejas, que pueden representar problemas físicos como transferencia de calor, flujos de fluidos, campos magnéticos, distribución de esfuerzo, etc.

Estos campos de aplicación suelen requerir de ecuaciones lineales para obtener un resultado [10].



Resumiendo estos métodos, el de *eliminación de gauss* es uno de los más empleados por su simplicidad, su desventaja consiste en su alto consumo de memoria. El método de *gradiente conjugado* reduce el consumo de memoria, sacrificando la precisión en el resultado (utiliza aproximación) y posee un mayor tiempo de resolución por el anidamientos de ciclos programa.
Como solución se utilizan matrices dispersas (también pueden encontrarse en la literatura como matrices ralas). Son matrices que cumplen las siguientes características:
Matrices relativamente grandes de *MxN*. Donde M representa a las columnas y N a las Filas.
La mayoría de los elementos son cero o valor nulo.
Los elementos no cero (ENC) representan una conexión lógica de adyacencia.

Tipos de matrices dispersas

Como se mencionó anteriormente en las matrices dispersas posee la mayoría de datos en cero, ya que representan conexiones de elementos débilmente acoplados. Dado un ENC como a_{ij} representa la conexión del elemento *i* con j [11].
Estas conexiones suelen representar distintos patrones con los elementos de la matriz. Como se explica a continuación.

Matriz dispersa diagonal

La matriz diagonal posee los elementos alineados en forma oblicua ya sea a izquierda o derecha. Suele representar la interconexión de grafos.

En la Figura 2 se visualiza una típica matriz dispersa del tipo diagonal y su grafo que la representa. Esta posee cuatro diagonales.

Primera diagonal: Está formada por los ENC {a32, a43}.

Segunda diagonal: Es la que corresponde a la diagonal principal y se forma con los ENC {a11, a22, a33, a44, a55}

Tercera diagonal: Se forma por los ENC {a12, a23}.

Cuarta diagonal: La forman los ENC {a14, a25}.

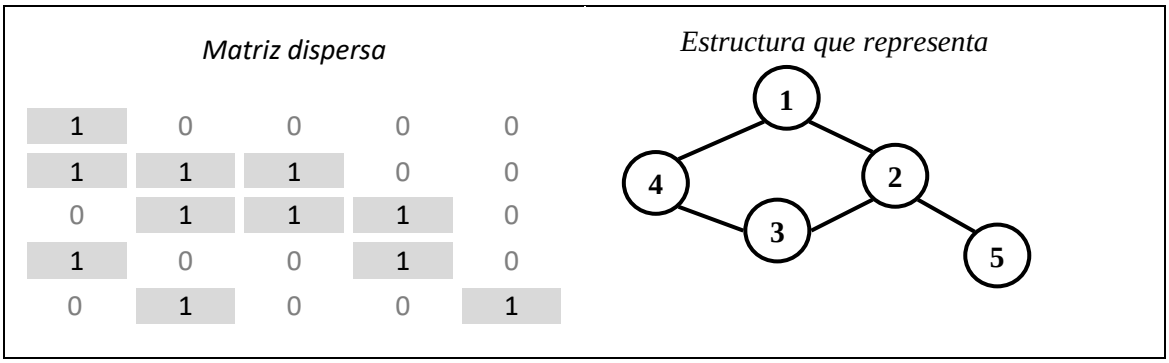


Figura 2 - Matriz dispersa diagonal con su correspondiente grafo.

La siguiente imagen (Figura 3) muestra una matriz dispersa diagonal de [12] llamado "espectro Alfven en magneto hidrodinámica" compuesta por 1,280 columnas y filas, danto un total de 1.6 millones de elementos, con 22,778 ENC.

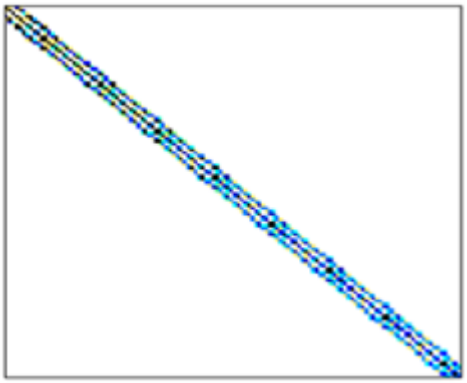


Figura 3 - Matriz dispersa diagonal.

Matriz dispersa triangular

La matriz triangular posee algunos valores desde la diagonal hacia un extremo y su opuesto posee elementos en cero. Cuando los ENC son sobre la diagonal se la llama *matriz triangular superior* y cuando es por debajo, se llama *matriz triangular inferior*

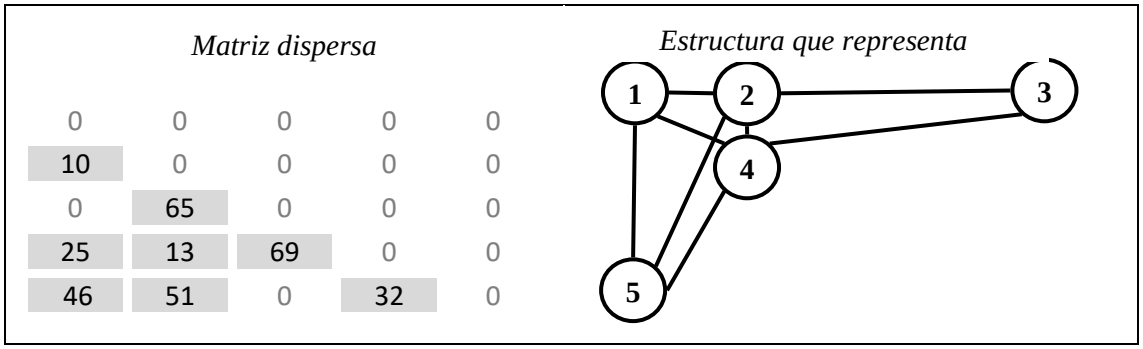


Figura 4 - Matriz dispersa triangular inferior.

En la figura 4 la matriz de ejemplo puede ser el caso de una simplificación estructural en forma de "L". En la que cada ENC expresa una distancia entre sus puntos de apoyo principales.

El siguiente gráfico (Figura 5) muestra a una matriz dispersa triangular inferior llamada BCSSTK32, que representa "Módulo estático de un chasis de automóvil" [ix]. Con columnas y filas de 44,609, por un total de casi 2 mil millones de elementos, con 110 millones ENC.

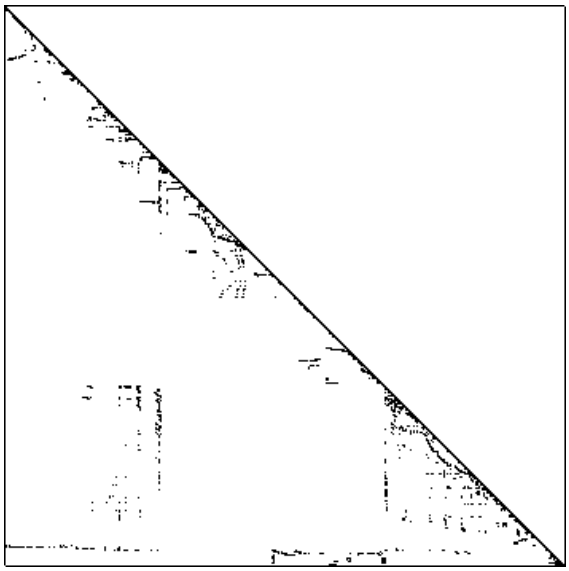


Figura 5 - Matriz dispersa triangular inferior.

Matriz dispersa simétrica

Las matrices simétricas representan redes de nodos balanceadas de interconexión. Estas pueden ser simétricas en forma diagonal o horizontal/vertical.

Como ejemplo en la Figura 6, representa a una interconexión de red topología estrella. Como ejemplo el nodo 1 puede representar a un conmutador central y su interconexión a las repetidoras.

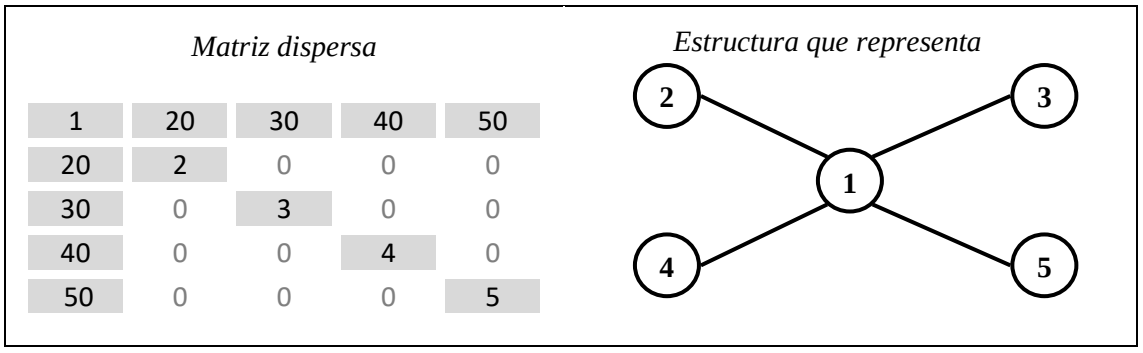


Figura 6 - Matriz dispersa simétrica por diagonal principal.

A continuación mostramos una matriz simétrica BCSPWR10 (Figura 7) llamada “Patrones de redes eléctricas” [ix] de 5300 columnas y filas por total de elementos 28 millones de elementos, y el total de ENC 21842

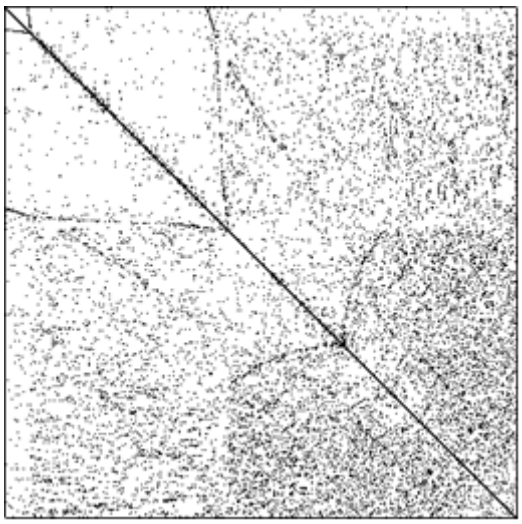


Figura 7 - Matriz dispersa simétrica.

Matriz dispersa aleatoria

En la matriz aleatoria sus ENC no representan un patrón. Representan organizaciones de conexiones de nodos de distintas índoles.

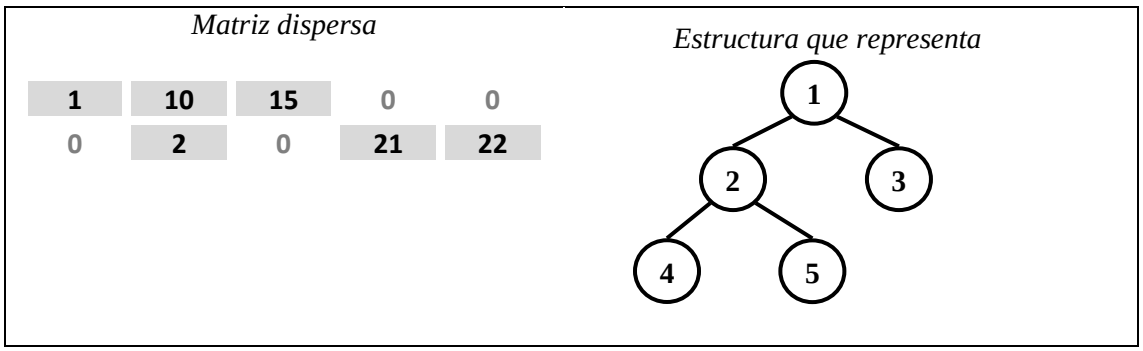


Figura 8 - Matriz dispersa Aleatoria.

En la figura 8 se muestra una matriz dispersa aleatoria de 5x2, que representa a la jerarquía de un árbol. Donde los ENC de su diagonal son los nodos que poseen hijos, el resto son nodos hojas.
En la figura 9 se encuentra la matriz dispersa en forma aleatoria llamada “Carreteras de Estados Unidos” [x] por su forma, consta de 23.947.347 columnas y filas, por un total de elementos 5.734 billones de elementos, con 57.708.624 ENC

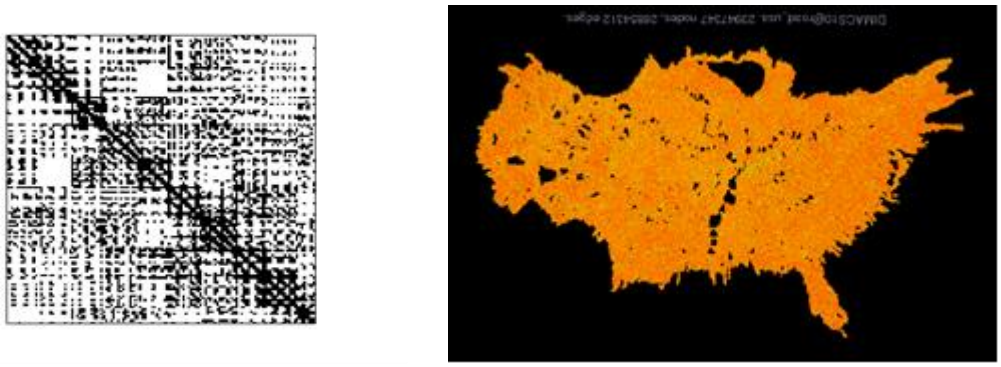


Figura 9 - Matriz dispersa aleatoria con su representación.

Formas de Almacenamiento

La particularidad de las matrices dispersas, de tener su mayoría de elementos no cero. Hace que se desperdicie lugar y tiempo de cómputo con los ENC. Por consiguiente se desarrollaron una variedad de formato de almacenamiento entre otros mostrado en la Tabla 1

Tabla 1 - Formatos de almacenamiento de matrices dispersas.

Alias	Nombre	Alias	Nombre
DNS	Denso	ELL	Ell pack
BND	Packe con banda	DIA	Diagonal
COO	Coordenadas	BSR	Bloque por fila
CRS	Comprimido por filas	SSK	Línea simétrica
CCS	Comprimido por columnas	BSR	Línea no simétrica
MSR	CRS modificado	JAD	Jagged Diagonal
LIL	Lista enlazada	JSA	Vector disperso de JAVA

Formatos de almacenamiento más utilizados

Almacenamiento Denso

Es la representación de la matriz, formada por *MxN* (columnas por filas) elementos. Es alineada consecutivamente como se muestra en la figura 10.

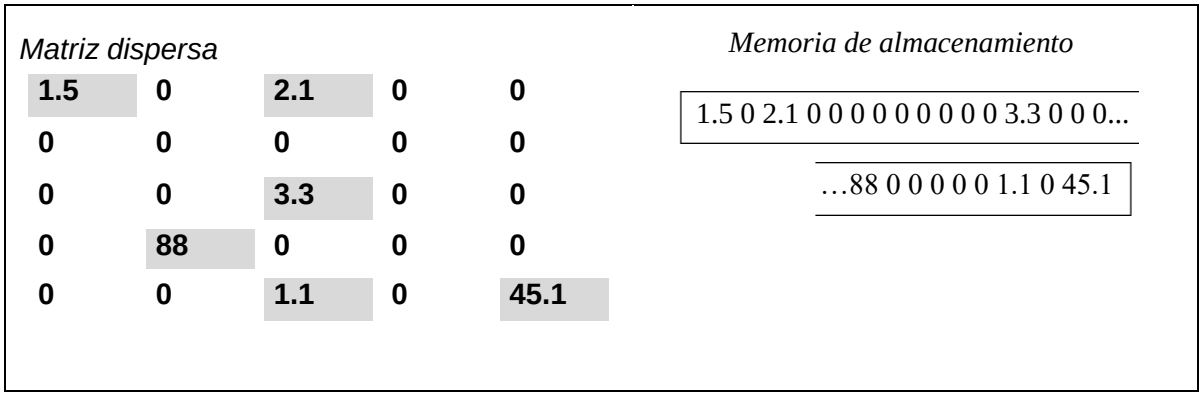


Figura 10 - Matriz dispersa con formato almacenamiento denso.

Como la matriz posee 5 columnas y 5 filas se almacenan 25 elementos y si estos son expresados en punto flotante (4 bytes por elemento) suman 100 bytes. Donde del total

de bytes que ocupa la matriz, los ENC solo utilizan 24 bytes, teniendo un 76% de información innecesaria.
El acceso a un elemento particular está dado por la ecuación *fila* por *total columnas matriz dispersa* más *columna* de ese elemento. Por ejemplo si se quiere acceder al valor 88 de la columna 1 y fila 3 el índice debe ser 16 (fila 3 por 5 columnas en total más 1 columna).
Mientras más grande sea la matriz, mayor será la cantidad de recursos ocupados que no serán útiles. Además de ocupar recursos los algoritmos que la utilizan desperdician tiempo al calcular sobre elementos en cero, ya que los mismos no modifican el resultado final.
El formato es independiente a la cantidad de ENC, siempre ocupará lo mismo. Es por eso que existen distintas formas de representar a las matrices dispersas para poder realizar cálculos sobre ellas desperdiciando la menor cantidad de recursos posibles. Estas formas aprovechan las características de cada tipo de matriz dispersa.

Almacenamiento por puntero

Existen infinidad de formas de representar los ENC de la matriz dispersa. En el almacenamiento por puntero se engloban a todas las representaciones (lista enlazada, cola, pila, lista doblemente enlazada, etc.) Que utilizan punteros a nodos para unir los distintos ENC.

Como se explica en [13] una estructura dinámica formada por lista enlazada, poseerá en cada nodo la información de los ENC de la matriz dispersa. El nodo contiene el valor del elemento, su ubicación absoluta en la matriz (el par columna/fila) y el puntero al siguiente nodo.

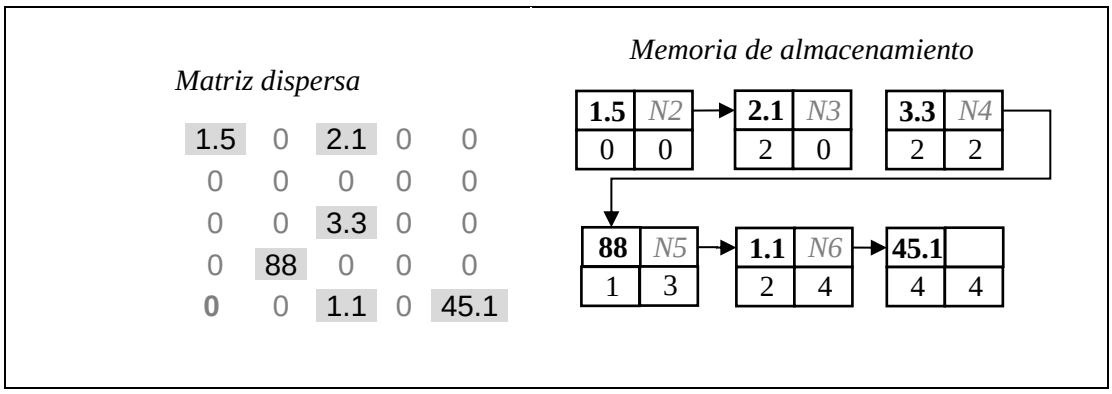


Figura 11 - Matriz dispersa con formato lista enlazada.

En la figura 11 se observan los 6 ENC enlazados por nodos, donde cada uno ocupa 16 bytes (4 bytes para cada valor/casillero del nodo). Siendo un total de 96 bytes de consumo. En principio se reduce el consumo un 4%, comparado con el formato denso. Pero si aumenta la cantidad de ENC aumentará el uso de memoria llegando a superarlo. Para buscar un elemento en particular requiere recorrer los nodos de la lista, evaluando el par (columna, fila) hasta hallarlo. Si se quiere acceder al elemento 88, se debe ir recorriendo la lista evaluando el par (1,3) hasta ubicarlo.
Como es un formato dinámico suele utilizarse en la etapa de creación de la matriz dispersa, ya que pueden insertarse elementos, facilitando el ordenamiento. Para realizar los cálculos se pasa a otro formato de mejor desempeño, porque en cálculos científicos las matrices dispersas no suelen tener modificaciones (agregarles ENC en medio de algún algoritmo).

Almacenamiento COO

El formato de almacenamiento por coordenadas consiste en almacenar la información que guarda cada nodo, visto anteriormente, sin el puntero al siguiente nodo. Para esto utiliza 3 vectores alineados, donde cada elemento de los vectores contiene la información de los ENC de la matriz dispersa.

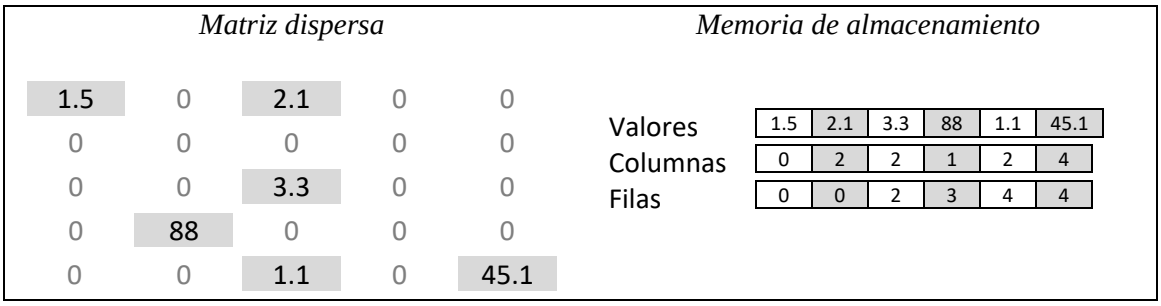


Figura 12 - Matriz Dispersa con formato COO.

En la figura 12 se muestra como estos vectores contienen los datos de los valores, columnas y filas [14] .Cada uno de los vectores tendrá tantos elementos como cantidad de ENC total de la matriz dispersa. Siendo el consumo de memoria un total de 72 bytes (6 ENC por 4 bytes por los 3 vectores).
El acceso a un elemento determinado, dada la posición de columna y fila en particular, requiere recorrer los vectores de columnas y filas. Si se quiere acceder al elemento 88 hay que llevar un índice que recorra los vectores de columnas y filas, hasta hallar su respectiva columna 1 fila 3. Luego ese índice es utilizado como entrada en el vector de valores y se obtiene el dato.
Estos vectores pueden ordenarse según conveniencia (por vector de columna o de fila) para realizar búsquedas binarias, mejorando el tiempo de respuesta en la búsqueda.

Almacenamiento CRS/CCS

El formato CRS viene por el nombre en inglés “Compressed Row Sparse” y CCS “Compressed Columns Sparse”. Ambos representan a la matriz dispersa en forma comprimida, para el primero la compresión es dada por fila y en el segundo por columna. Como indica [14] el formato CRS (figura 13) es similar al formato COO, ya que mantiene los vectores de valores y columnas. La diferencia es que el tercer vector contiene índices a las filas.
En vector de índices para una matriz de MxN tiene un tamaño de M+1. Para los M elementos almacena el índice i que indica donde comienzan los elementos de cada fila (de la matriz dispersa) en los vectores de valores y columnas. El último elemento del vector de índices posee la cantidad total de ENC de la matriz dispersa.

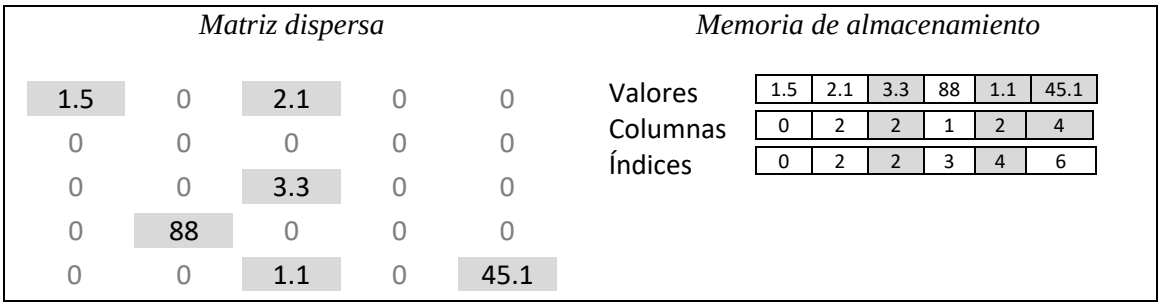


Figura 13 - Matriz dispersa con formatos de CRS.

Se dice que es similar al *formato COO*, ya que si lo compara cuando esta ordenado por filas. El *formato CRS* comprime y guarda las filas implícitamente como índices.

Para acceder a un elemento determinado hay que utilizar el *vector de índices* y la fila del elemento a buscar. Luego hay que buscar la columna en el *vector de columnas,* hasta el valor indicado en el *vector de índices* más uno.
Por ejemplo si se quiere buscar el valor 2.1 que está ubicado en la columna 2 y fila 0. El número de fila "0" en *vector de índices* indica donde comienzan los elementos de esta fila en los *vectores valores y columnas* Mientras que él valor "0"+1 indica hasta donde llega la fila cero.
Aplicando esto se tiene que la fila 0 comienza en la posición 0 y llega hasta –no incluyéndola- la posición 2 (dos primeros valores del *vector índice*). Indicando que los primeros 2 elementos de los *vectores valores* y *columnas* corresponden a la primer fila de la matriz discreta comprimida en formato CRS. Así se logra indexar por fila acotando la búsqueda de los elementos a lo necesario.
El formato CCS tiene el mismo principio [14] [15], salvo que hace la compresión por columna. Manteniendo los *vectores de valores* y *filas*, del *formato COO*. Ahora el *vector de índices* indicará las columnas en forma comprimida (figura 14).

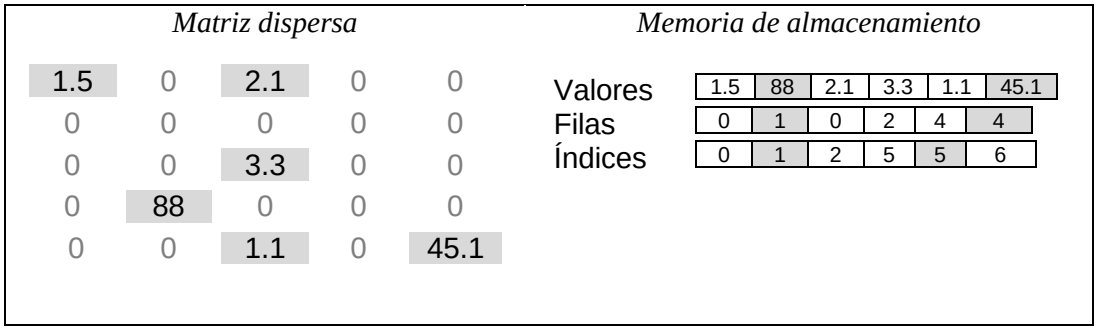


Figura 14 - Matriz dispersa con formato de CCS.

La mecánica de acceso es similar, aplicando la búsqueda anterior el valor 2.1 en la columna 2 y fila 0. Se obtiene que la columna 2 comience en la posición 2 y llega hasta antes que la posición 5 (elementos 2 y 3 del vector de índices). En este rango hay que buscar la fila 0 en el vector de filas. Una vez encontrada, en esa posición, se accede al vector de valores y se obtiene el dato buscado.

Almacenamiento ELL
El método de conversión del formato ELL [11] [14] (Figura 15) consiste en conocer a priori la máxima cantidad de ENC por fila. Luego se generan 2 vectores: uno de *valores ELL* con los ENC de la matriz dispersa y otro utilizado como *índice ELL* de la posición en las columnas.
Los *vectores de valores ELL* e *índices ELL* poseen sub-vectores, cuya longitud equivale a la cantidad de valores no nulos por fila. Cuando es menor la cantidad de ENC en la fila el sub-vector correspondiente se debe rellenar con un valor distinto, indicando esta situación.

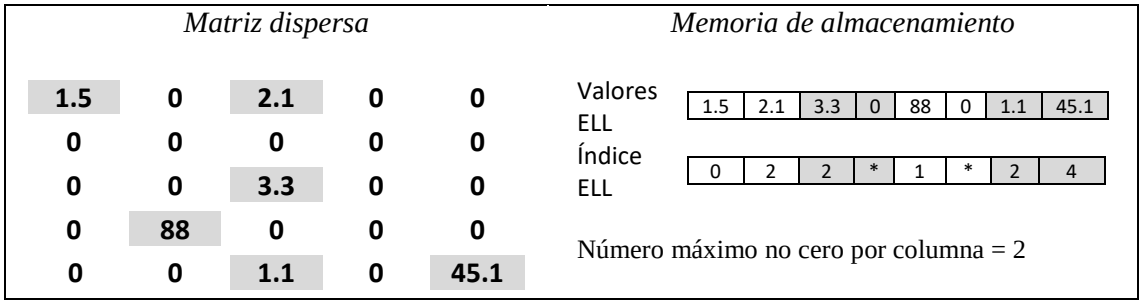


Figura 15 - Matriz dispersa con formato de ELL.

El tamaño que ocupan los vectores está dado por la cantidad máxima de ENC por columna multiplicado por la cantidad de filas con ENC por los bytes de cada elemento. En este caso un total de 64 bytes (2 ENC max por columna x 4 filas con ENC x 2 vectores x 4 bytes).
Por ejemplo si se quiere buscar el valor 2.1 que está ubicado en la columna 2 y fila 0. El número de fila "0" se lo multiplica por los ENC máximo por columna y se obtiene el índice del sub-vector donde comienza la fila en los vectores de valores +ELL e índices ELL. Se busca dentro del sub-vector índice ELL el número de columna. Cuando se encuentra se tiene la posición dentro del vector de valores ELL que contiene el dato buscado.

Lineamientos del trabajo
Durante este trabajo se utilizaran los repositorios [ix] y [x]. Para poder compararlo con otros trabajos de investigación que utilizaron las mismas matrices. Aplicándoles varios formatos de almacenamiento en distintos algoritmos, evaluándolos en una arquitectura clúster de CPU-GPU. Se tratará de realizar el análisis de desempeño y posibles mejoras de performance.

El problema de los N cuerpos (N-Body)

Tenemos un sistema formado por un conjunto de cuerpos. El estado inicial está determinado por las posiciones y velocidades de estos cuerpos.
El problema N-Body consiste en simular la evolución del sistema con el paso del tiempo.
Entre cada par de cuerpos interactúa una fuerza que dependerá del tipo de sistema, ya se astrofísico, plasma o molecular. Cada cuerpo variará su posición y su velocidad con el paso del tiempo en función de la fuerza resultante que reciba en cada momento.
Si los cuerpos son planetas o galaxias, dependiendo de la escala, la fuerza que actúa es la gravedad (Ley de Newton), en el caso de la física del plasma son iones, electrones, etc. y la fuerza que interactúa es la eléctrica (Ley de Coulomb). Si el ámbito de trabajo es la Dinámica molecular, los cuerpos son átomos y moléculas, y la fuerza que interactúa es la electrostática (Ley de Coulomb).
La diferencia de resolución está dada por la fórmula que determina la fuerza interactuante.

Descripción n-body partícula a partícula
Para obtener la posición y velocidad de cada cuerpo mediante este método de resolución, es necesario calcular todas las fuerzas que interactúan entre **todos** los cuerpos que forman el sistema.
Para obtener las posiciones y velocidades de los cuerpos del sistema, durante un período de tiempo T, se deben realizar una serie de k simulaciones, donde se calcula el estado del sistema durante los tiempo t_1, t_2, t_3, …,t_k, siendo t_0 el instante inicial. Se calcula la evolución del sistema durante un período dt para las k simulaciones – dt es el tiempo entre dos instantes consecutivos- y k=T/dt
Cada simulación de las k se realiza en dos pasos:
Se calcula la fuerza para cada cuerpo que ejercen sobre él el resto de los cuerpos, en el instante t.
Se calcula la velocidad y la posición en el instante t+dt de todos los cuerpos a partir de las fuerzas obtenidas en el paso anterior.
En la Figura.16 se muestra una representación simplificada del algoritmo de simulación para el n-body.

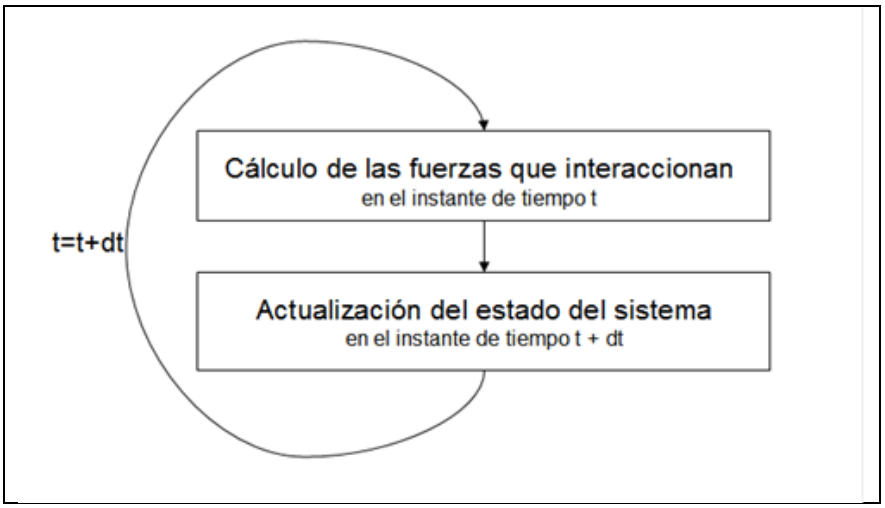


Figura 16 - Representación simplificada del algoritmo n-body.

Este algoritmo partícula-partícula conocido como método directo o método básico posee un orden de complejidad $O(n^2)$ en la cantidad de operaciones que debe realizar para cada iteración y magnitud n – número de partículas - en cuanto a las necesidades de memoria [16]

Descripción de n- body con tipo de fuerzas gravitacional.
Este es el caso de los cuerpos que son planetas o galaxias y cuya simulación se desea. Cada uno de los cuerpos posee una masa y un estado inicial formado por su posición y velocidad iniciales, y entre los cuerpos interactúan fuerzas gravitacionales.

Descripción matemática.
Cada cuerpo i del sistema tiene una masa m_i, y en el instante t se encuentra ubicado en el espacio en la posición que determina un vector $P_i^t = (Px_i^t, Py_i^t, Pz_i^t)$ y mantiene una velocidad descrita por un vector de velocidad $V_i^t = (Vx_i^t, Vy_i^t, Vz_i^t)$. El estado inicial del sistema es conocido y está determinado por P_i^{t0}, V_i^{t0} y m_i para cada una de las n partículas.
Variables utilizadas para el cálculo:

Pit= (Pxit, Pyit, Pzit)	: Vector posición del cuerpo i en el instante de tiempo t
Vit= (Vxit, Vyit, Vzit).	: Vector de Velocidad del cuerpo i en el instante de tiempo t
Fit= (Fxit, Fyit, Fzit).	: Vector de Fuerza que incide sobre el cuerpo i en el instante t
mi	: Masa del cuerpo i
dt	: Intervalo de tiempo de cada iteración
k	: Número de iteraciones a simular
G	: Constante Gravitacional.

La evolución del sistema – posición y velocidad de cada uno de los cuerpos- se simula durante el período de tiempo T, dividido en k pasos de tiempo dt, donde k= T/dt.
Considerando que actúan fuerzas gravitacionales se tiene la siguiente descripción matemática del problema:

$$Fx_i^t = \sum_{j=1,\, j\neq i}^{n} G \times \frac{m_i \times m_j}{\left(\sqrt{\left(Px_j^t - Px_i^t\right)^2 + \left(Py_j^t - Py_i^t\right)^2 + \left(Pz_j^t - Pz_i^t\right)^2}\right)^3} \times \left(Px_j^t - Px_i^t\right)$$

$$Fy_i^t = \sum_{j=1,\, j\neq i}^{n} G \times \frac{m_i \times m_j}{\left(\sqrt{\left(Px_j^t - Px_i^t\right)^2 + \left(Py_j^t - Py_j^t\right)^2 + \left(Pz_j^t - Pz_i^t\right)^2}\right)^3} \times \left(Py_j^t - Py_i^t\right)$$

$$Fz_i^t = \sum_{j=1,\, j\neq i}^{n} G \times \frac{m_i \times m_j}{\left(\sqrt{\left(Px_j^t - Px_i^t\right)^2 + \left(Py_j^t - Py_i^t\right)^2 + \left(Pz_j^t - Pz_i^t\right)^2}\right)^3} \times \left(Pz_j^t - Pz_i^t\right)$$

$$V_i^{t+1} = V_i^t + \frac{F_i^t}{m_i} \times dt$$

$$P_i^{t+1} = P_i^t + V_i^{t+1} \times dt$$

Estas fórmulas se aplican de manera consecutiva para calcular los sucesivos valores de P^{ti} y V^{ti}, dónde t= t1,t2,t3,…,tk .
Esta descripción matemática es el punto de partida para el algoritmo n-body y sus modificaciones [6] [17].

Mercury N-Body

En lugar de utilizar implementaciones de algoritmo puramente teóricas, el grupo de investigación optó por utilizar una implementación del algoritmo de N-Body que fuera utilizada en forma regular por equipos científicos en el mundo. Es de esta manera que se decidió por utilizar la implementación llamada "Mercury" desarrollada por el Dr. John Chambers.
Esta implementación se encuentra escrita en código Fortran, y es en concreto, un conjunto de algoritmos llamados "integradores numéricos", que realizan aproximaciones numéricas para resolver el problema de los n-cuerpos. El software, es altamente configurable, y se pueden seleccionar diferentes casos a ejecutar, siempre utilizando la misma entrada.
Esto permite realizar diferentes observaciones, o atacar un problema con ciertas condiciones iniciales, utilizando diferentes tipos de análisis, según lo que se necesite estudiar. Esto posee como agregado, que Mercury, tendrá un tiempo de ejecución diferente, según que algoritmo se decida utilizar, o que tipo de problema se intente resolver.
Particularmente nuestro equipo de investigación, atacara el problema específico de los N-Cuerpos, utilizando encuentros cercanos y colisiones.
En este caso, ocurre que cuando dos cuerpos se encuentran en orbitas muy cercanas, la aceleración a calcular, responde a valores elevados respecto a distancias mayores, con lo cual aparecen 3 casos particulares, como se ve en la figura 17.

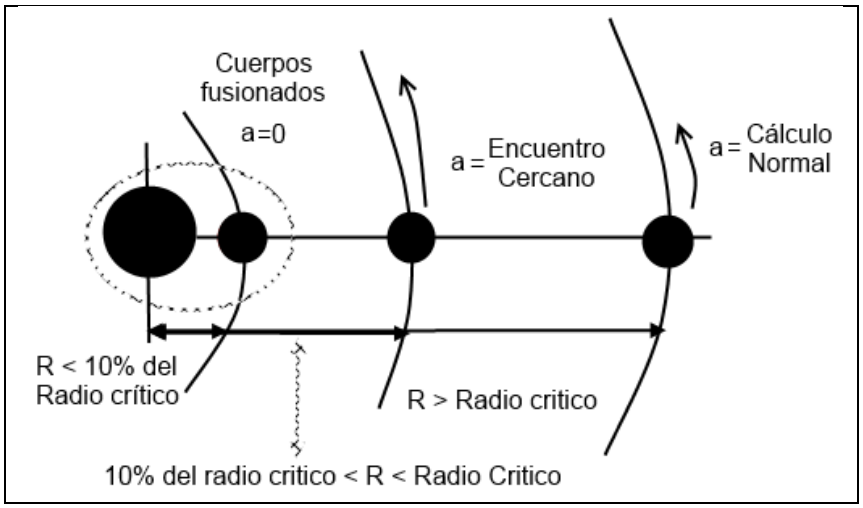


Figura 17 - Posibles casos de cálculo de la aceleración de un cuerpo según su distancia de orbita respecto de otro cuerpo.

Se puede apreciar, que los llamados encuentros cercanos, se producen cuando un cuerpo se encuentra orbitando a una distancia ubicada entre un llamado "Radio crítico" (R, ingresado como parámetro al software), y el 10% de este mismo Radio. Los cuerpos ubicados por fuera de este radio, utilizan la aceleración que se obtiene de resolver el sistema numérico mencionado en el apartado anterior, mientras que los cuerpos ubicados por debajo de este margen, son "fusionados", utilizando como base el cuerpo de mayor masa.
Sin embargo los cuerpos que se ubican en el margen mencionado, reciben un tratamiento especial, el cual requiere de fórmulas más complejas que en los demás casos. Es en este apartado especial, que el grupo de investigación encontró posibles mejoras para los casos de prueba utilizados.

3.2 Lugar y Tiempo de la Investigación

El equipo de investigación llevó a cabo el desarrollo y pruebas en el laboratorio 266, en el que se encuentra el Grupo S.O.D.I.U.M. (Grupo de Investigación y formación en Sistemas Operativos), el que pertenece al Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de La Matanza. Este laboratorio está equipado con las máquinas para desarrollo de software con sistemas operativos Windows, Linux y S.O.D.I.U.M. para la realización de pruebas de instalación, acceso a Internet, acceso a la biblioteca digital del MINCyT y máquinas virtuales. Se encuentran instaladas las herramientas de desarrollo y pruebas en dos computadoras con procesadores Intel Core i5 , dos GPUs Geforce 650 y una ATI Radeon Graphics Card R7 250, que fueron intercambiadas para la realización del proyecto, el cual se llevó a cabo esde el comienzo del año 2014 hasta el fin del año 2015.
Los tiempos de las tareas se llevaron a cabo en base al GANTT (ver tabla 2 y 3) diseñado en el momento de la presentación del protocolo.

3.3 Diseño de la Investigación
Programa de actividades

Tabla 2 - Gantt de tareas previstas para el Primer Año-2014

Actividades / Responsables 1er Año	Mes 1	Mes 2	Mes 3	Mes 4	Mes 5	Mes 6	Mes 7	Mes 8	Mes 9	Mes 10	Mes 11	Mes 12
Etapa Inicial												

Elaboración inicial del proyecto	x	x	x	x	x	x						
Relevamiento de bibliografía sobre algoritmos para GPU	x	x	x									
Puesta en contacto con académicos de otras universidades para asesoría en optimización de algoritmos	x	x	x									
Obtener información de instalación y programación de hardware GPU	x	x	x	x								
Puesta a punto del laboratorio de investigación	x	x	x									
Realizar cursos de computación de alta performance con cluster	x	x	x	x	x	x						
Poner en licitación los requerimientos de hardware					x	x						
Instalación y pruebas iniciales de la arquitectura GPU							x	x	x	x	x	x
Configuración del hardware existente y el adquirido							x	x				
Elaboración de casos de prueba							x	x				
Puesta a punto de las herramientas y entorno de medición									x			
Pruebas iniciales de control para comparaciones futuras									x	x		
Aplicación y prueba técnicas básicas de optimización existentes y medición de resultados										x	x	x

Tabla 3 - Gantt de tareas previstas para el Primer Año-2015

Actividades / Responsables 2do Año	Mes 1	Mes 2	Mes 3	Mes 4	Mes 5	Mes 6	Mes 7	Mes 8	Mes 9	Mes 10	Mes 11	Mes 12
Etapa de Avance												
Elaboración y prueba de los algoritmos a ser optimizados	x	x	x	x	x	x						
Creación y evaluación estrategias de aplicación de técnicas combinadas a priori	x	x	x									
Profiling de hardware de GPU y registración de datos	x	x	x									
Correr pruebas y comparar con los resultados de control		x	x									
Elaborar relación entre resultados/mediciones de hardware			x	x	x	x						
Probar el desempeño de cada optimización con otros algoritmos científicos				x	x	x						
Aplicación de los dispositivos en el campo de aplicación							x	x	x	x	x	x
Simulación de aplicación entornos de trabajo científico							x	x	x			
Elaboración de guía de optimización de GPU utilizando mediciones de hardware							x	x	x			
Etapa Final												
Comparación de Resultados y elaboración de conclusiones										x	x	
Divulgación de resultados y publicaciones												x

En el Gantt (Tabla 2 y Tabla 3) se pueden ver las etapas de la investigación y sus tareas correspondientes las que se detallan a continuación.

3.4 ETAPAS

Etapa Inicial

Investigación bibliográfica.

Se realizó una búsqueda bibliográfica continua de los temas que nos afectan, tales como algoritmos N-BODY, paralelismo en CPU y GPU durante todo el año, Matrices Dispersas, algoritmos de transformación para paralelizar matrices dispersas.

Se incorporó un becario que colaboró con el procesamiento de la bibliografía.

Trabajo con otros Académicos.

Se realizó un trabajo en conjunto con la Universidad Nacional de La Plata, con la colaboración del Dr Fernando Tinetti, el que consistió en la optimización de algoritmos de astrofísica, relacionados con la resolución de ecuaciones diferenciales parciales, para atmósferas estelares, junto con optimizaciones de algoritmos que resuelven el problema de los N-Cuerpos. Estos últimos realizan computo intensivo, porque además del cálculo tradicional, tienen en cuenta lo que se conoce como "encuentros cercanos" y Colisiones de cuerpos.

En este trabajo participan además diversos Astrofísicos de la Universidad Nacional de La Plata, como consultores expertos en el área.

Informe de Instalación

Se registraron los pasos necesarios para tener una instalación funcional de ambas tecnologías tanto en el sistema operativo Windows 7 como en el sistema operativo GNU/Linux - Distribución Ubuntu 12.04. Posteriormente continuaremos con la descripción de los diferentes modelos de programación sobre hardware GPU propuestos por ambas tecnologías**.** Ver **-ANEXO A -** ,

Puesta a punto del laboratorio y configuración del Hardware

Durante los primeros meses se trabajó en la instalación y puesta a punto de las bibliotecas de código abierto para la programación de GPU OpenCL[1] y CUDA, que constan de una interfaz de programación de aplicaciones y de un lenguaje de programación. Permiten crear aplicaciones con paralelismo a nivel de datos y de hilos GPU que pueden ejecutarse tanto en unidades centrales de procesamiento como unidades de procesamiento gráfico, CUDA[2] que hace referencia tanto a un compilador como a un conjunto de herramientas de desarrollo creadas por NVIDIA que permiten a los programadores usar una variación del lenguaje de programación C para codificar algoritmos en GPU de NVIDIA., mientras que OpenCL ha sido creada por Apple, quien creó la especificación original y desarrollada en conjunto con AMD, IBM, Intel y NVIDIA. Apple la propuso al Grupo Khronos para convertirla en un estándar abierto y libre de derechos Estas herramientas intentan explotar las ventajas de las GPU frente a las CPU de propósito general utilizando el paralelismo que ofrecen. Por esto, las aplicaciones diseñadas utilizando numerosos hilos que realizan tareas independientes pueden conseguir un rendimiento mucho mayor en su ejecución sobre una GPU ya que el paralelismo es su forma natural de trabajo. Dichos procedimientos se dejaron registrados en **- ANEXO A –**

Capacitación del grupo de Investigación y cursos realizados

[1] OpenCL sigla de Open Computing Language (en español lenguaje de computación abierto)
[2] CUDA sigla de Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo)

Un grupo de Investigadores asistieron a cursos en otras universidades, Workshops y/o congresos

BIGDATA en Escuela de Posgrado de La Universidad Nacional de La Plata

Predicción de performance y ejecución eficiente en sistemas de cómputos de altas prestaciones Cluster y Cloud, Escuela de Posgrado de La Universidad Nacional de La Plata.

Se participó en los siguientes eventos:

Segundas Jornadas de Cloud Computing.

Cluster Grid y Cloud Computing en la Universidad Nacional de La Plata

Prof: José A. Olivas –Universidad de Castilla-La Mancha

Prof: Armando de Giusti y Marcelo Naiof Universidad Nacional de La Plata

Modelado y Simulación en Ciencia Computacional. Técnicas de Modelado y Simulación de Altas Prestaciones.

Prof: Emilio Luque y Remo Suppi - Universidad Autónoma De Barcelona -.

Ver programas en el **- ANEXO B –**

Elaboración de casos de Pruebas

Matrices dispersas

Para el caso de matrices dispersas, se utiliza un generador de matrices, al cual se le indica una densidad de datos distintos de cero, los cuales siempre son inferiores al 10% del tamaño de la matriz. Luego se realizan operaciones algebraicas sencillas de multiplicación entre matriz-vector, y matriz-matriz, estudiando las posibles soluciones.
Se decidió utilizar bibliotecas estándar de matrices dispersas para realizar pruebas, con el objetivo de realizar optimizaciones sobre estas mismas bibliotecas, siempre y cuando fuere posible.
Las bibliotecas seleccionadas para los casos de matrices dispersas fueron las siguientes:

- PARALUTION - [xi]
- VexCL - [xii]
- VIENNACL - [xiii]

Para correr las pruebas, se utilizó el hardware del laboratorio, adquirido con el presupuesto del proyecto.

Problema de los N Cuerpos

Las pruebas utilizadas para el problema de los N-Cuerpos, fueron proporcionadas por investigadores de la Facultad de Ciencias Astronómicas y Geofísicas de La Plata. Estos casos de prueba, corresponden a cantidades pequeñas de cuerpos, que son simulados utilizando encuentros cercanos, por un lapso de 100 millones de años, con un paso de simulación de 15 días.
Para simular estas pruebas, se utilizaron servidores HPC de Microsoft Azure, que permitieron agilizar la realización de las mismas en un tiempo reducido. Así mismo, las pruebas consistieron en configuraciones de 15 cuerpos iniciales, de distintas masas, y en distintas posiciones.

Realización de Pruebas Iniciales

Matrices dispersas

Las pruebas consistieron en la multiplicación de matriz por vector, utilizando matrices con nombre " VanVelzen std1_Jac2_db“ de 21982x21982 con 498771 elementos no nulos o cero (ENC), y realizando 1024 veces la misma operación, obteniendo promedios de ejecución. Los resultados se pueden observar en las Tabla 4, Tabla 5 y Tabla 6

Tabla 4 - Resultados de la ejecución de la biblioteca Parulation.

Operación Matriz*Vector	Paralution AMDGPU		Paralution NVIDIAGPU	
Formato	**Tiempo**	**Gflop/Sec**	**Tiempo**	**Gflop/Sec**
CSR	0,528166	1,75707	0,408654	2,27093
MCSR	0,51826	1,79065	0,411844	2,25334
ELL	0,236363	3,94393	0,278202	3,35080
COO	0,683236	1,35828	0,658819	1,40862
HYB	0,529866	1,75538	0,41667	2,23226

Tabla 5 - Resultados de ejecución de la biblioteca Vexcl.

Operación Matriz*Vector	VECXCL AMDGPU	VEXCL NVIDIAGPU
GFlops	2,12969	22,7719
Tiempo (s)	0,46702	0,0436771

Tabla 6 - Resultados de ejecución de la biblioteca ViennaCL.

Operación Matriz*Vector	Vienna AMDGPU		Vienna NVIDIAGPU	
Formato	**Tiempo**	**Gflop/Seg**	**Tiempo**	**Gflop/Seg**
COO	7,42336	0,12501	7,44757	0,12461
ELL	16,123	0,05756	15,9269	0,05827
HYB	16,0783	0,05718	16,8232	0,05516
Sliced ELL	7,74153	0,11988	8,0137	0,11581

Mercury N-Body

En el caso de Mercury, se realizó una optimización secuencial de la biblioteca, y se procedió a correr las 5 pruebas mencionadas. La configuración inicial de Mercury para todas las pruebas fue la mencionada en la figura 18.

```
)-----------------------------------------------------------------------
) Important integration parameters:
)-----------------------------------------------------------------------
 algorithm (MVS, BS, BS2, RADAU, HYBRID etc) = hyb
 start time (days)= 0.
 stop time (days) = 365.25d8
 output interval (days) = 365.25d4
 timestep (days) = 15.
 accuracy parameter=1.d-12
)-----------------------------------------------------------------------
) Integration options:
)-----------------------------------------------------------------------
 stop integration after a close encounter = no
 allow collisions to occur = yes
 include collisional fragmentation = no
 express time in days or years = years
 express time relative to integration start time = yes
 output precision = medium
 < not used at present >
 include relativity in integration= no
 include user-defined force = no
```

Figura 18 - Configuración inicial de Mercury.

Tabla 7 - Tiempos de ejecución de Mercury.

Nbody Mercury	Tiempo Seg	Gflop/Seg
Tiempo prueba 1	6788,625	0.60925445
Tiempo prueba 2	5109,625	0.80945275
Tiempo prueba 3	8683,375	0.47631249
Tiempo prueba 4	6853,25	0.60350928
Tiempo prueba 5	6883	0.60090077

A modo de muestra, se agrega la información de salida, de la prueba número 5 (Tabla 7), donde se observan diferencias significativas en los valores que deberían ser idénticos (esto se observa en todos los casos de prueba)

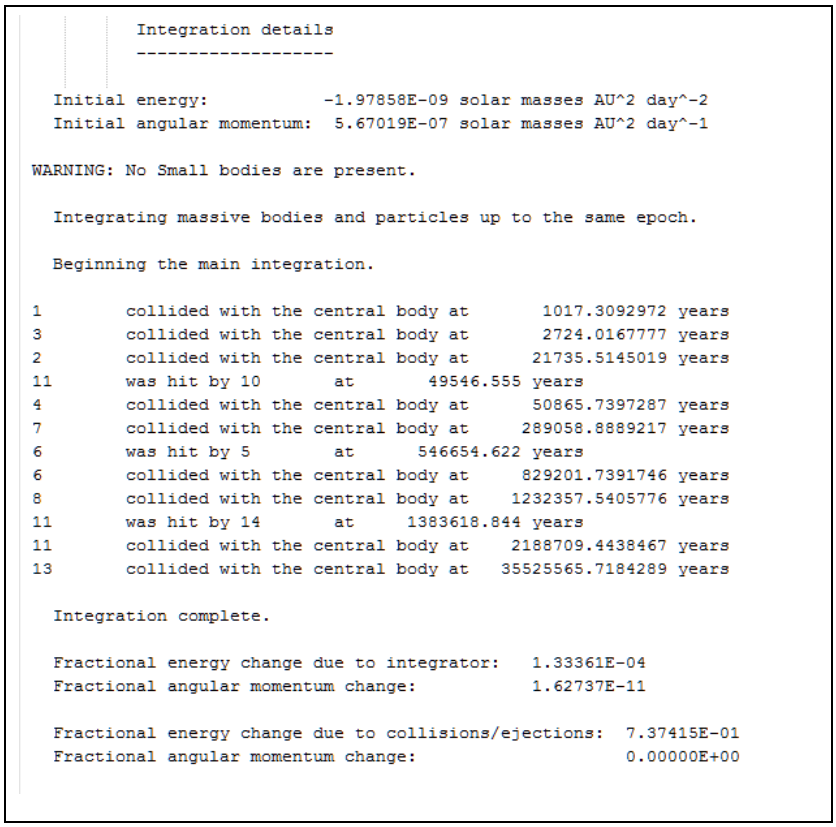


Figura 19 - Salida de ejemplo de la prueba número 5, punto de control.

Aplicación y prueba técnicas básicas de optimizaciones existentes y medición de resultados

Al comenzar el proyecto se ejecutaron diferentes algoritmos científicos utilizando lenguaje de programación CUDA C para diferentes estructuras multidimensionales (parámetros típicos para bloques *BlockSize* con *BlockDim*, y para grillas *GridSize* con *GridDim*) junto con los parámetros de compilación. Como base de conocimiento se utilizaron las buenas prácticas propuestas por NVIDIA [3], que evalúan el rendimiento en función del tiempo de ejecución y de la transferencia entre memorias. Al mismo tiempo, se utilizará una herramienta de *profiling* (NVIDIA Nsight) para registrar todos los indicadores de hardware provistos por la GPU utilizada.

Una vez obtenidos los datos de rendimiento e indicadores de hardware para cada algoritmo y cada ejecución, se investigará la relación entre ambos para determinar una relación entre los indicadores y el tiempo tomado de ejecución. También se evaluarán posibles relaciones de ganancia/pérdida entre diferentes indicadores, especialmente aquellos que compiten por memoria intra-chip. Los impactos de relaciones se evaluarán para cada caso, obteniendo una función de rendimiento.

Etapa de Avance

Durante la segunda etapa del proyecto, Se aplicaron diferentes técnicas de optimización desarrolladas e investigadas durante el proyecto de investigación precedente sobre el algoritmo N-Body [5] para simulaciones gravitatorias, el algoritmo Schönhage-Strassen [17] para multiplicación de enteros, y el algoritmo SGP4/SDP [18] para predicción de posición orbital de satélites artificiales. Luego de aplicar dichas optimizaciones, se registraran los datos obtenidos de la herramienta de profiling. Una vez obtenidos dichos datos, se evaluará la eficiencia de cada técnica de optimización para solucionar los

problemas de rendimiento visibles en los indicadores de hardware estudiados en la primera etapa de esta investigación. En la etapa final del proyecto, basados en el estudio hecho sobre el impacto de las optimizaciones sobre los indicadores de hardware, se establecerán técnicas de optimización de algoritmos científicos únicamente basadas en mediciones de hardware agnósticas del código. Se evaluarán estas técnicas para la ejecución de los algoritmos en una única GPU. Para la elaboración de una guía final basados en los resultados obtenidos en las pruebas.

Elaboración y Prueba de los Algoritmos a ser Optimizados.

Para lograr recolectar la información provista por las herramientas de *profilling*. Se disponía de dos opciones, realizarlo gráficamente o mediante reportes.
El hacerlo gráficamente brinda la posibilidad de realizar un análisis profundo del comportamiento y uso de recurso del GPU, pero en contrapartida encontramos la dificultad para realizar comparación entre distintas ejecuciones.
La utilización de reportes permite expresarlos en archivos de texto plano. La información puede filtrarse y agruparse —con scripts[3]— armando planillas de datos y procesándolas en hojas de cálculos, las cuales pueden llegar a graficar las mediciones. Encontramos desventajas significativas: no hay un estándar de salida, por lo que el formato es dependiente de la herramienta utilizada, además la información generada está limitada a una porción de la ejecución total y al ser archivos de texto plano, ocupan gran volumen en el almacenamiento disco.
Por lo expuesto anteriormente, se eligió utilizar el análisis mediante reportes. Esto permitió armar *scripts* para la ejecución de casos de prueba. Los reportes de *profiling* de cada fabricante fueron enviados a distintos archivos. Luego los resultados se normalizaron y se armaron nuevos archivos con información combinada (Figura 20).

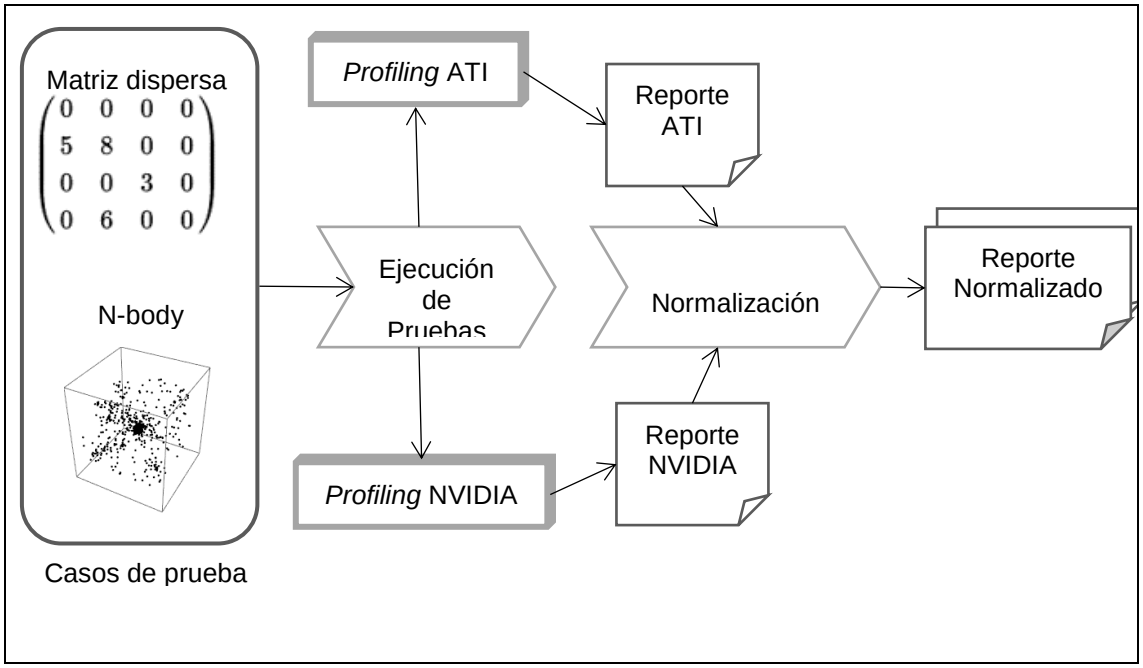


Figura 20- Proceso de obtención y normalización de reportes de *profiling*.

Creación y evaluación estrategias de aplicación de técnicas combinadas a priori.

[3] Scrip: archivo de procesamiento por lotes.

La elección de los algoritmos científicos se realizó siguiendo modelo de evaluación Bibliométricos. Permitió dar una idea global de la actualidad del tema de investigación. La elección de la forma de realizar las pruebas y recolección de datos se realizó por el método de costo-beneficio, comparando los métodos de interfaz gráfica con el de reportes. Optándose por este último, gracias a la posibilidad que brindan los reportes en la automatización a la hora de armar las ejecuciones y comparar los resultados de las pruebas.

Profiling de hardware de GPU y registración de datos.

Puede encontrarse mayor detalle de este punto en el **ANEXO G**, donde se enumeran y califican a todos los contadores de hardware que recopilan las GPUs .
Al profundizar en este tema se encontró una gran disparidad en *Contadores de Hardware*.
Por un lado se encuentra diferencias en la información provista por cada fabricante, para el modelo ATI el *profilling* solo muestra dieciocho *Contadores de Hardware* mientras que para NVIDIA muestra un total de doscientos diecisiete. Otra gran diferencia es la forma de medir de los *Contadores de Hardware*, los únicos con información similar son los que miden transferencia de datos y tiempos.
Los *Contadores de Hardware* para ATI solo proseen métricas enfocadas en tres categorías o contextos:

- General: son valoraciones para promedios de distinto tipos de instrucciones/ *wavefronts*[4] y medición de tiempos.
- Memoria local: Instrucciones o tiempos de acceso a la memoria local (interna al chip GPU).
- Memoria global. Instrucciones o tiempos de acceso a la memoria Global (externa al chip GPU).

Los *Contadores de Hardware* para NVIDIA se dividen en dos tipos:

Métricas

- Memoria: Se enfocada en las transacciones de memoria local, compartida y global.
- Instrucción: Recolecta información de operaciones para distinto nivel de bifurcación, de uso simple o doble precisión.
- Multiprocesador: Información de los *CUDA CORE*.
- Caché: Recompila toda la información de las tasas de acceso, acierto y pérdida de memoria cache para los niveles 1 y 2.
- Textura: Informa sobre los datos de acceso a la memoria de textura.

Eventos

- Instrucción: Muestra información en cantidades sobre *kerne*[5]*l*, *warps*[6] e instrucciones.
- Memoria: Diversos accesos a memoria causados por conflictos de bancos, replanificación de instrucciones del warp por accesos repetidos a memoria global (Cantidad de instrucciones de lectura/escritura desde 8 a 128 bits).
- Caché: Detalla la cantidad de solicitudes, pérdidas y escrituras de memoria cache.

[4] Wavefront: es la unidad básica de planificación para los procesadores gráficos ATI.
[5] Kernel: kernel en “C for CUDA”, es una función la cual al ejecutarse lo hará en N distintos hilos.
[6] Warp: es la unidad básica de planificación para los procesadores gráficos NVIDEA

- Disparadores genéricos: Definidos por el desarrollador, para recompilar información tantas veces como sea invocado.

La nivelación de mediciones constituyó otra dificultad, para lograr recolectar la información de ambos *profilling* expresados en la misma magnitud. Ya que NVIDIA expresa las transferencias en bytes, mientras que ATI lo hace en una magnitud acorde a la cantidad de datos, según corresponda. El tiempo es expresado, en NVIDIA en lapsos transcurrido, mientras que ATI, marca el comienzo y fin expresado unidades de tiempo de GPU.

Correr pruebas y comparar con los resultados de control.

La finalidad de las distintas pruebas es tomar indicadores desde el *profiler* (medidor de rendimiento), comparando los eventos y contadores de hardware en cada GPU.

Se utilizaron bibliotecas BLAS que poseen ciertas limitaciones, para la biblioteca ViennaC*L* en ATI el uso del *profiler* fue exitoso. Este no fue el caso en NVIDIA, la versión del *profiler* no detecta actividad para OpenCL, porque no está implementado para esta tecnología, con lo cual, no pudo ser utilizado. La biblioteca *Magma*, en su implementación para OpenCL solo soporta matrices densas, a diferencia de la versión CUDA que sí soporta matrices dispersas, mientras que la única biblioteca que funcionó en ambas arquitecturas fue Paralution.

Matrices dispersas utilizadas

Las matrices utilizadas están almacenadas en formato de MM y son del repositorio de la Universidad de Florida [19]. Con el fin de lograr obtener datos comparativos las matrices son del mismo tipo de datos y procesado en punto flotante con doble precisión.

Al estar almacenadas en formato COO previamente serán convertidas al formato CRS antes de ser utilizada por el algoritmo que hará la multiplicación de matriz por vector SPMV (Sparse Product Matrix Vector). La elección se debe a su importancia [20] siendo parte central de otros, como el cálculo de gradiente conjugado.

La comparación y elección de matrices está dada por la cantidad de EDC, ya que es lo que da la característica en cuanto a transferencias y cálculos de la matriz. La siguiente Tabla 8 detalla matrices elegidas:

Tabla 8 - Características de Matrices utilizadas.

Nombre	Columnas	Filas	EDC
Oberwolfach piston	2,025	2,025	100,015
VanVelzen std1_Jac2_db	21,982	21,982	498,771
Goodwin rim	22,560	22,560	1,014,951
Mallya lhr71c	70,304	70,304	1,528,092
CEMW/tmt_sym	726,713	726,713	5,080,961

Tabla 9 - Especificación de los Procesadores Gráficos Utilizados.

Características	NVIDIA	ATI
Modelo	GeForce GTX 650	R7 250
Frecuencia	1058 MHz	1050Mhz
Memoria total	1GiB DDR5	1GiBDDR5
Memoria cache	262k bytes	16k bytes
Divisiónconfigurable de dimensiones de hilos por placa	{1024, 1024, 64}	{ 256, 256, 256}
Grupo de hilos	32 warps	32 wavefront
Versión de plataforma	CUDACapability 3.0.	OpenCL1.2
Profiler	CUDA profiler 1.6	AMD CodeXL 1.7
Contadores de hardware	217	18

Las pruebas consistieron en ejecutar el algoritmo SPMV para las matrices, arriba indicadas. De manera iterativa se realizaron diez ejecuciones por vez, desde las herramientas de *profiler* de cada GPU en dos equipos (**Tabla 9**).

Para la ejecución de pruebas se siguió un lineamiento de modelo denominado V que originalmente divide las etapas de desarrollo de componentes que son probados funcionalmente. En caso de encontrar alguna diferencia, ese componente vuelve a una etapa que corresponda a su mismo nivel. En esta adaptación el desarrollo se basa en la mejora de desempeño y los resultados de las pruebas que están dados por la medición resultante del *profilling*. Si un desarrollo no obtuvo mejora, vuelve a su etapa correspondiente.
El ciclo comienza definiendo los algoritmos a utilizar, se realiza la selección de que parte del algoritmo puede llegar a mejorarse, se realiza la especificación de las mejoras seleccionadas, y se codifica la optimización.
Al ejecutar la prueba, da como resultado la medición de los contadores de Hardware normalizados. Si en este punto no logro medirse o tiene que hacer correcciones vuelve a la etapa de especificación.
Continúa con la de extrapolación de los resultados de las distintas mediciones, verificando si éstos son los esperados. Cualquier ajuste requiere que se revea la etapa --y sus siguientes—de diseño.
La etapa dice si la mejora obtenida en un algoritmo, puede llegar a ampliarse a otros casos (surge un patrón repetible). Tanto correcciones como extensiones a más algoritmos requieren de volver a trabajar sobre el conjunto de funciones a optimizar. Esto se explica en la siguiente figura.

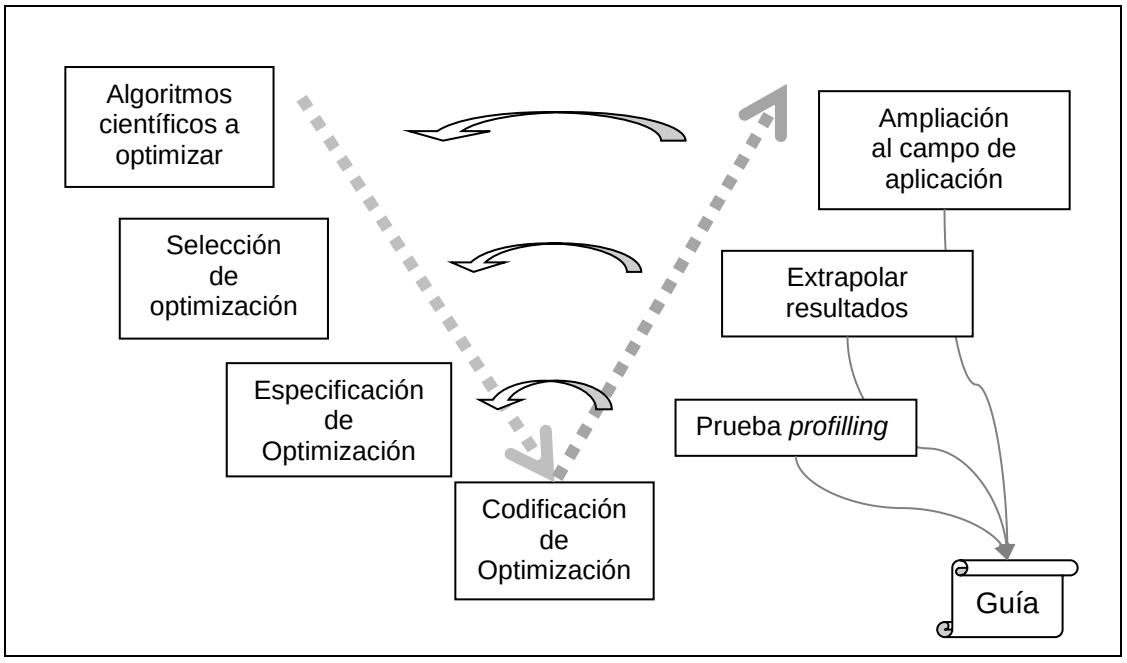


Figura 21 - Proceso de ejecución de pruebas, recolección y afinación de mediciones.

Todas las etapas de prueba, resultados y ampliación brindaron información relevante que nutren la guía de optimización. Esta información está directamente relacionada con la etapa que se encuentra en el mismo nivel. Por ejemplo si hubo una optimización y esta fue medida en un conjunto de Contadores de Hardware, quedan relacionados en la guía. Siempre que sea repetible.

Pruebas realizadas

La recolección de información utilizando herramientas de *profilling* resulto tan ardua, ya que se enfocó en distintas temáticas. Esta necesidad surgió, por un lado en la diferencia de los tiempos de cálculo. Para el algoritmo Mercury (n-body) los tiempos de respuesta son en miles de segundos, mientras que para Blas (matrices dispersas) están en el orden de mili segundos. Además las ejecuciones de las bibliotecas BLAS, detectamos discrepancias en los contadores de hardware para el mismo caso de prueba. Otro punto tenido en cuenta fue que en N-body, se utiliza un único desarrollo del algoritmo Mercury, no tiene comparaciones entre diferentes bibliotecas, pero brinda una gran respuesta en la optimización.

Se procedió con el análisis de optimización, para luego detallar las diferencias detectadas por los contadores de hardware al comprar las bibliotecas BLAS.

Análisis resultados optimización

El caso del algoritmo Mercury se realizó como principal optimización la no secuencialidad de las etapas internas. Se procedió a ejecutar 5 pruebas, comparando los resultados, con los obtenidos en la etapa de control. La configuración inicial del algoritmo Mercury para las pruebas se encuentra en la figura 22.

```
        Integration details
        -------------------

  Initial energy:           -1.97858E-09 solar masses AU^2 day^-2
  Initial angular momentum:  5.67019E-07 solar masses AU^2 day^-1

WARNING: No Small bodies are present.

  Integrating massive bodies and particles up to the same epoch.

  Beginning the main integration.

2         collided with the central body at       1355.6733334 years
1         collided with the central body at      15784.1096344 years
8         was hit by 6         at       36624.705 years
5         collided with the central body at      43866.4367034 years
3         collided with the central body at     155380.5282150 years
7         collided with the central body at     174963.8680615 years
10        was hit by 9         at      602262.983 years
11        collided with the central body at     870603.0340116 years
8         collided with the central body at    1150632.8465510 years
15        was hit by 12        at     1332006.055 years
14        collided with the central body at    2000060.4357160 years
10        was hit by 13        at     3805273.205 years
4         collided with the central body at    4155425.8355547 years

  Integration complete.

  Fractional energy change due to integrator:   7.77502E-05
  Fractional angular momentum change:           1.48160E-12

  Fractional energy change due to collisions/ejections:  5.50667E-01
  Fractional angular momentum change:                    0.00000E+00
```

Figura 22 - Salida de ejemplo de la prueba número 5, optimizada.

Los resultados de la optimización son expuestos en la tabla 10 y comparados gráficamente en **(Figura 23)**

Tabla 10- Tiempos de ejecución de Mercury.

Nbody Mercury	Original	Optimizado
Tiempo prueba 1 (seg)	6788,625	5452,5
Tiempo prueba 2 (seg)	5109,625	5451,25
Tiempo prueba 3 (seg)	8683,375	6395,75
Tiempo prueba 4 (seg)	6853,25	6685,5
Tiempo prueba 5 (seg)	6883	4721,75

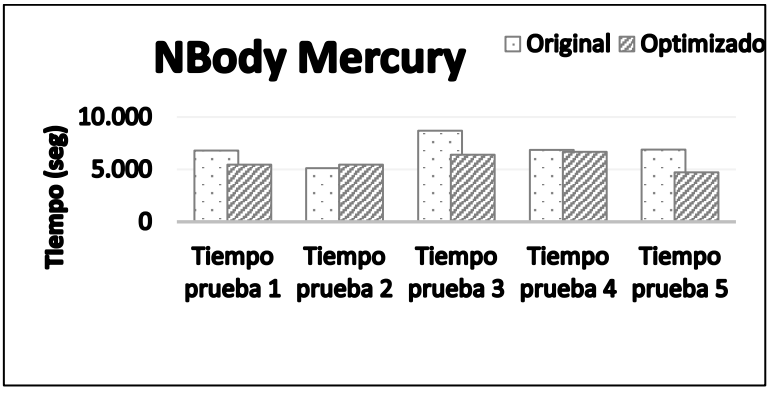


Figura 23 - Comparación prueba control con optimizada para Mercury.

En el caso de la comparación de mejoras en la biblioteca BLAS en matrices dispersas, se trabajó sobre el conjunto de matrices indicadas con la biblioteca PARALUTION, por ser la única en la que se obtuvo contadores de hardware en ambas tecnologías ATI y NVIDIA (**Tabla 11**) y (Figura 24).

Tabla 11 - Tiempos de ejecución biblioteca paralution.

Tecnología	Matriz por EDC	Original	Optimizado
ATI	100015	160.1761667	345.1736667
NVIDIA	100015	57.0775	57.2246
ATI	498771	802.983	852.8043333
NVIDIA	498771	402.6849	402.9667
ATI	1014951	1571.1265	813.3818333
NVIDIA	1014951	694.8663	693.2953
ATI	1528092	1053.76	630.3278333
NVIDIA	1528092	1347.424	1347.185
ATI	5080961	4245.865667	4256.552833
NVIDIA	5080961	4201.554	4201.777

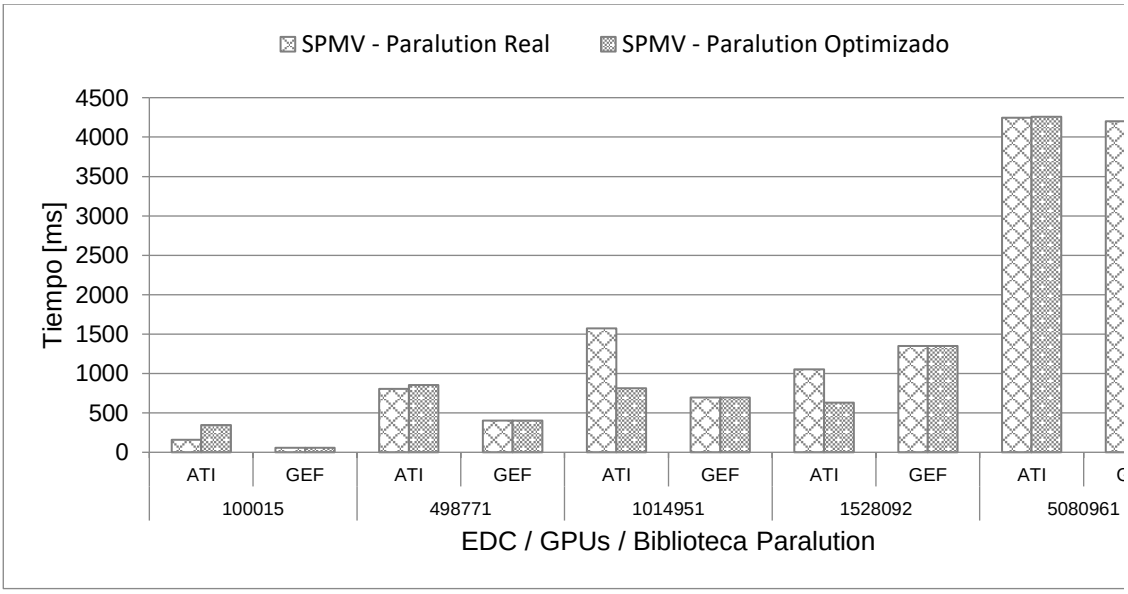


Figura 24 - Comparativa de ejecuciones Paralution en GPUs.

Diferencias en ejecuciones de bibliotecas BLAS

Analizando la información recolectada en (Figura 25) puede verse que los tiempos de los eventos agrupado en la categoría de *escritura GPU,* para *ViennaCL* son muy inferiores y para *Magma* son un poco superiores que el resto. Como las muestras tomadas en los resultados del algoritmo SPMV, fueron iguales para las 3 bibliotecas. Se muestra que las diferencias se encuentran en el volumen de información transmitida.

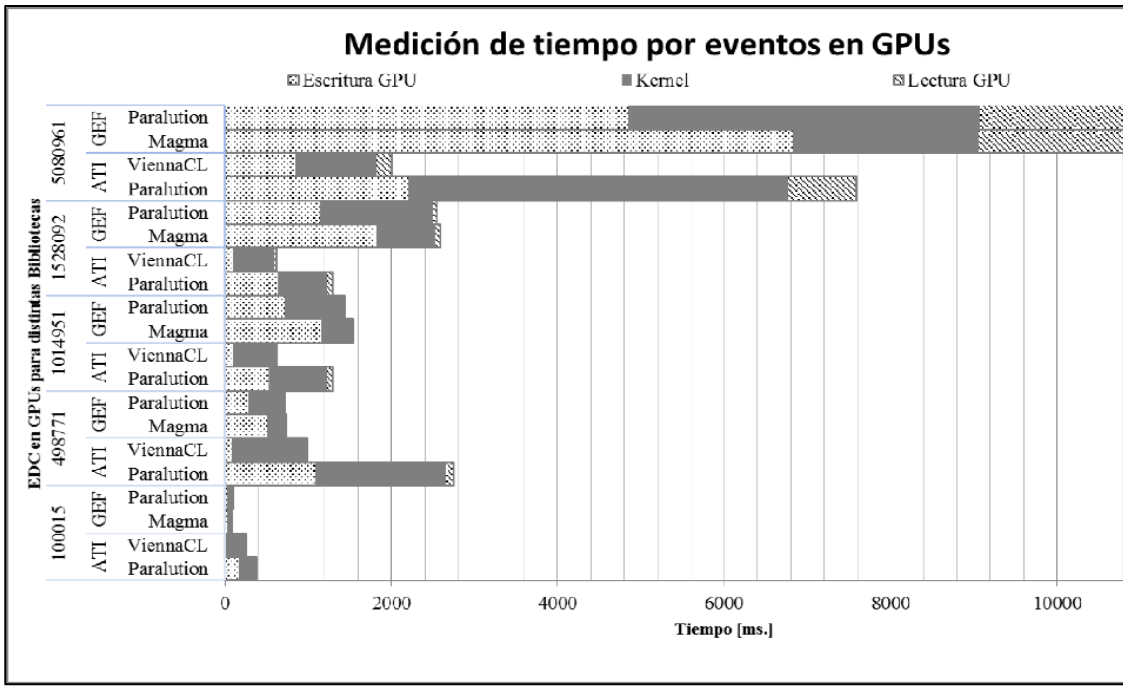


Figura 25 -Mediciones de 3 bibliotecas BLAS que utilizan las GPUs ATI y NVIDIA (GEF), indicando la cantidad de elementos distintos de cero (EDC) por cada fila. Esto indica la matriz utilizada en cada una de las pruebas

Diferencias en Trasferencias

Para analizar dichas diferencias se discrimina la cantidad de bytes transferidos escritos -desde el CPU al GPU- y leídos -desde GPU al CPU-. Para tener elementos de comparación se elaboró un valor mínimo teórico (llamado Escritura Real en la **¡Error! No se encuentra el origen de la referencia.**, que está formado por los volúmenes de datos de la entrada y la salida del algoritmo SPMV.

Para los datos escritos al GPU se estudia el par vector-matriz .Por un lado el vector se compone de tantos elementos como filas tiene la matriz (ver proporciones de matrices en Tabla 8). Por el otro, la matriz está representada en formato CRS, como se vio en la sección de materiales utilizados en la investigación (**¡Error! No se encuentra el origen de la referencia.**). Por ende se establece que está formada por 3 vectores uno de columnas, otro de valores con tamaño igual a la cantidad de EDC, mientras que el tercer vector es el de índices a filas (cuyo tamaño es la cantidad de filas más uno). Tanto el vector de la multiplicación como el vector de valores se expresan en números con formato de doble precisión. Los vectores de columnas e índices a la filas, se compone de números con formato entero. El gráfico de comparación puede verse en la figura 25.

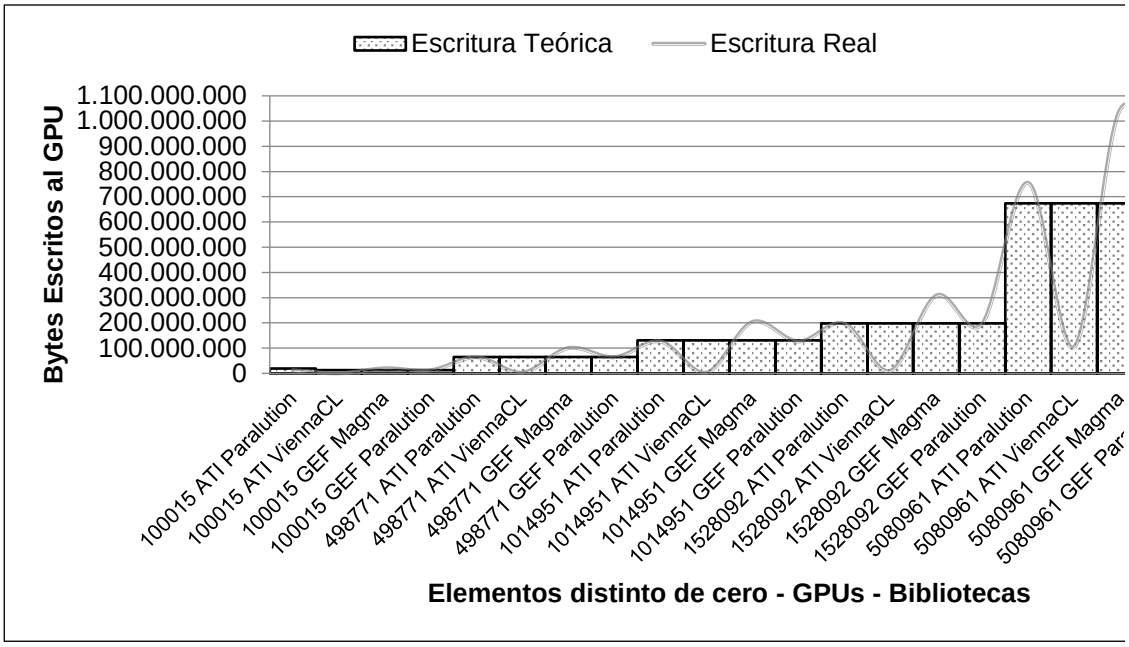


Figura 26 - Bytes escritos al GPU: Real vs Teórico por las bibliotecas BLAS.

Por otro lado, los bytes leídos desde GPU (**¡Error! No se encuentra el origen de la referencia.**), responden al único vector de resultado del algoritmo SPMV. El tamaño de este vector está dado por las filas de la matriz, mientras que el tipo de dato soporta la doble precisión. Para verificarlo también se confecciono un valor teórico esperado (llamado lectura Real), para contrastar con el valor real.

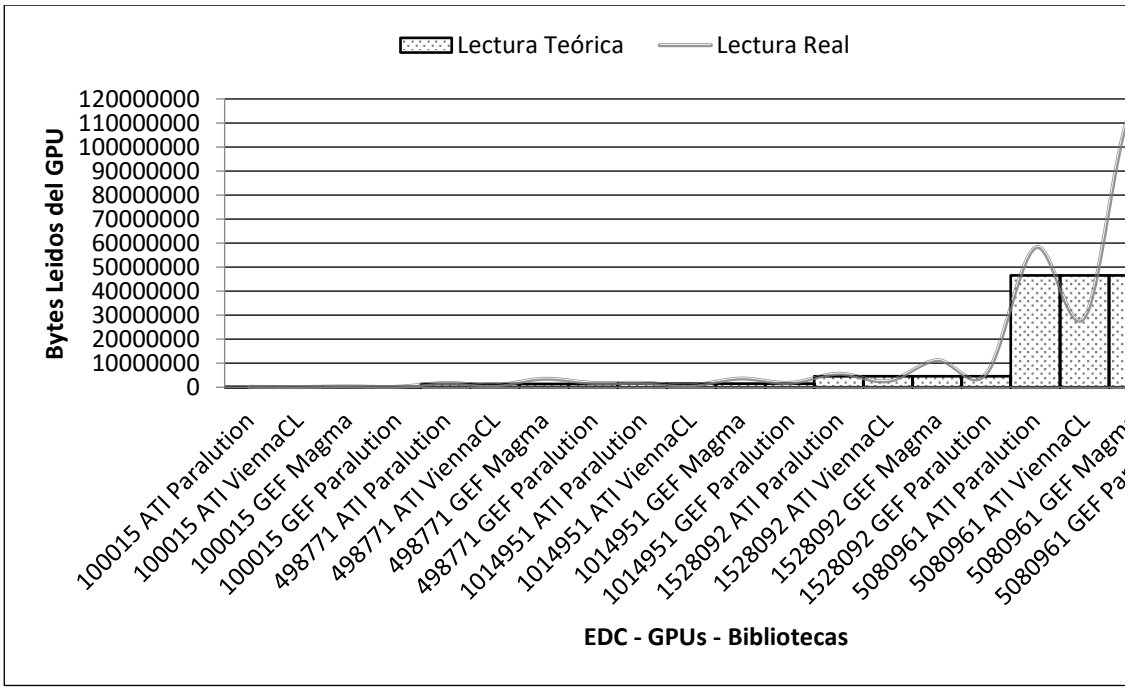


Figura 27 - Bytes leídos del GPU: Real vs Teórico por las bibliotecas BLAS.

En ambos análisis se comprobó que *Paralution* tiene transferencias de escritura un poco superiores a las teóricas. Esto se debe a transfiere el vector resultado del CPU al GPU,

antes del algoritmo. Verificando que la lógica del algoritmo aplicada en el hilo *kernel* del GPU, a dicho valor no lo reutiliza, sino que lo sobrescribe.

Para *ViennaCL* las mediciones de transferencias en la escritura son muy inferiores, y un poco superiores en el caso de la lectura. Este resultado refleja que internamente, esta biblioteca no utiliza el formato CRS, pudiendo utilizar otro formato que optimiza la compresión de datos (por ej. ELL).

Para el caso de la biblioteca *Magma,* se transfieren volúmenes similares al teórico. Comparándolo con *Paralution* los tiempos son levemente superiores.

Diferencia de configuración de las multi-dimensiones

Otro punto de análisis desarrollado es la comparación de la planificación multidimensional de los hilos *kernel* del GPU. Se encontró con gran disparidad en la misma biblioteca para distintas tecnologías y en algunos casos, hasta con poca eficiencia en la definición de las dimensiones del algoritmo SPMV.

La arquitectura del GPU permite a los hilos *kernel* que se organicen multi/dimensionalmente en dos niveles de abstracción, además en cada nivel se soportan hasta tres dimensiones lógicas. El primer nivel está dado por el agrupamiento de los warps en *bloques*, para el caso de GPU NVIDIA y luego estos en el segundo nivel que se llama *Grilla*. Para la arquitectura ATI, tiene la misma topología con otros nombres, los wavefront se organizan en *grupos locales* –primer nivel- y estos se organizan en el segundo nivel como *grupos globales.* Ambas tecnologías poseen hasta tres dimensiones en el primer nivel.

Por la lógica del algoritmo SPMV implementada en GPU, la cantidad de hilos kernel que se planifican dependen de la cantidad de filas que poseen las matrices utilizadas. Con esta información como punto de comparación con la obtenida por las herramientas de *profiling* con los reportes, se elabora la Tabla 12:

Tabla 12 - Dimensionamiento de SMPV para BLAS.

ATI						
EDC	Hilos teóricos/ Filas	Biblioteca	Elementos de Trabajo	Grupos de Trabajo	Total de hilos planificados	Hilos sobre planificados
100015	2025	Paralution	2048, 1, 1	1, 1, 1	2048	23
		ViennaCL	25600, 1, 1	1, 1, 1	25600	25600
498771	21982	Paralution	22016, 1, 1	1, 1, 1	22016	34
		ViennaCL	132864, 1, 1	1, 1, 1	132864	132864
1014951	22560	Paralution	22592, 1, 1	1, 1, 1	22592	32
		ViennaCL	266496, 1, 1	1, 1, 1	266496	266496
1528092	70304	Paralution	70336, 1, 1	1, 1, 1	70336	32
		ViennaCL	386304, 1, 1	1, 1, 1	386304	386304
5080961	726713	Paralution	726720, 1, 1	1, 1, 1	726720	7
		ViennaCL	1273088, 1, 1	1, 1, 1	1273088	1273088
NVIDIA						
EDC	Hilos Teóricos/ Filas	Biblioteca	Bloques	Grillas	Total de hilos planificados	Hilos sobre planificados
100015	2025	Magma	16, 1, 1	127, 1, 1	2032	7
		Paralution	16, 1, 1	127, 1, 1	2032	7
498771	21982	Magma	16, 1, 1	1374, 1, 1	21984	2
		Paralution	16, 1, 1	1374, 1, 1	21984	2
1014951	22560	Magma	16, 1, 1	1410, 1, 1	22560	0
		Paralution	16, 1, 1	1410, 1, 1	22560	0
1528092	70304	Magma	16, 1, 1	4394, 1, 1	70304	0

		Paralution	16, 1, 1	4394, 1, 1	70304	0
5080961	726713	Magma	32, 1, 1	22710, 1, 1	726720	7
		Paralution	32, 1, 1	22710, 1, 1	726720	7

De la misma puede apreciar que el caso más resonante es el de la biblioteca ViennaCL en ATI, que se sobredimensionan los hilos *kernel*. Mientras que la versión de Paralution, de la misma plataforma, es la correcta. Cabe aclarar que la cuenta de hilos sobreplanificados menos los hilos esperados, no da cero, porque la cantidad de hilos se redondea al tamaño del *wavefront* (32 hilos *kernel*).

Para la definición de ***multidimensionado*** de NIVIDIA ambas bibliotecas definen una dimensión en bloques y otra dimensión para grillas. Esto trae un pequeño costo en el algoritmo, que tiene que calcular la dirección lógica absoluta del hilo *kernel* utilizando los datos de bloques y grillas en ATI solo se utiliza una dimensión-. Otro dato obtenido interesante que la misma biblioteca planifique distinto el mismo algoritmo, con la misma matriz de entrada, dependiendo la arquitectura GPU.

Elaborar relación entre resultados/mediciones de hardware.

Las buenas prácticas para optimizar ejecuciones en GPU [3] se enfocan a ciertos casos bien definidos. Cuando los analizamos como punto de partida de la investigación. Encontramos que pueden ser medibles utilizando un único contador de hardware. Por ejemplo “minimizar la sobrecarga de transferencias” al copiar los mismos datos entre CPU al GPU y viceversa. Puede verse con los contadores de hardware *WriteSize* (en ATI) o *gst_transactions* (en NVIDIA), que cuantifican las escrituras a memoria global. Otro ejemplo la “maximización del ancho de banda de memoria” se centra en utilizar el orden consecutivo la memoria entre hilos consecutivos, por lo que cada vez que los microinstrucciones del GPU deben ir a buscar a memoria global en lugar de traer solo un byte, trae un conjunto, esta técnica se llama acceso *collapsed*. Con esto logra que cuando se planifican los hilos, el conjunto (warps o wavefront dependiendo de la tecnología) va a tener disponible la porción de memoria. Si no fuera así, debería hacer accesos a la memoria --de lenta respuesta—por hilo antes de planificarlos en grupo. Este contador de hardware está disponible para NVIDIA *l1_global_load_hit*.

Estos casos marcaron los puntos de partida para generar la guía de optimización. Con distintas pruebas y readaptación de las optimizaciones, se encontró que la clave es relacionar eventos y contadores de hardware. Esta relación forman el conjunto de pautas que conducen a interpretar si la comparación brinda una mejora o no en la búsqueda de optimizaciones.

No hay forma de comparar contadores de hardware de ATI y NVIDIA. Lo que se realizo fue buscar contadores que den información expresada en similares parámetros. Luego se aplicaron distintas optimizaciones y se analizó la evolución de contadores seleccionados.

La guía se basa en eventos de transferencia en memoria global, dimensionado de kernels y puntos críticos en la ejecución de instrucciones.

Los Contadores de Hardware que dan información sobre los accesos a memoria global en ATI son “FetchSize” y “WriteSize”, miden kilobytes transferidos a memoria global. En NVIDIA no hay un contador que los exprese en esas magnitudes, por lo que a estos valores se los obtiene del parámetro de los eventos “cudamemcpy”.

La información sobre las dimensiones viene de los contadores de eventos, en NVIDIA para bloque, definido en tres campos (BlockX,BlockY,BlockZ) y en la grilla los siguientes tres campos (GridX,GridY,GridZ). En ATI se define las dimensiones como grupo de trabajo “WorkGroupSize’ y al grupo global “GlobalWorkSize”.

A nivel de instrucciones se utilizó el contador de hardware “ipc” en NVIDIA, que indica la cantidad de instrucciones ejecutadas por ciclo.y para ATI un contador de hardware utilizado, que expresa este concepto cómputo-instrucciones en función del tiempo, “VALUBusy” indica el porcentaje de tiempo procesando instrucciones de punto flotante.

Probar el desempeño de cada optimización con otros algoritmos científicos.

Existen diversos problemas científicos que precisan gran intensidad de cálculo, como N-Body, Mercury N-Body que utilizamos para la generación de la guía de optimizaciones y se comenzó a trabajar con SGP4 para el cálculo de basura espacial en orbitas cercanas a la tierra, modificando el trabajo realizado por el equipo publicado en CACIC 2013

Aplicación de los dispositivos en el campo de aplicación.

Los campos de aplicación que tienen los sistemas lineales dispersos, van desde los utilizados en modelados científicos y problemas industriales. Como por ejemplo simulación de medioambiente, procesamientos complejos industriales [19]. Además en su aplicación por método de elementos finitos en ingeniería eléctrica, que fue donde surgió esta rama, modelando tendido eléctrico. También en ingeniería industrial para calcular la resonancia que provocan los transeúntes en puentes peatonales [20]. Otra aplicación es el modelado de sistemas económicos [21]. Resultan de los algoritmos emblemáticos catalogados como uno de los siete más importantes de la actualidad [22].

Simulación de aplicación en entornos de trabajo científico.

Esta guía de optimización, puede ser aplicada en una gran variedad de algoritmos científicos que realizan grandes cantidades de cálculos. Tales como el N-body, el Mercury, el modelo actual utilizado es el SGP4, un derivado del SGP para orbitas cercanas a la tierra, y el SDP4, un derivado del SGP para orbitas superiores a los 5000 km de altura.
Con este trabajo se intenta dar un paso hacia adelante en la dirección de la optimización del modelo predictivo. Al realizar un análisis objetivo desde el punto de vista de las operaciones en punto flotante, y del manejo de cache, esto produce resultados que podrían indicar posible mejoras en una optimización, en la función analizada. Se sugiere que el mayor esfuerzo de análisis, debería concentrarse en el desafío de obtener versiones paralelas de todas las perturbaciones calculadas posibles. Inclusive, la posibilidad de distribuir el trabajo computacional utilizando GP-GPU para este fin.

.

Elaboración de guía de optimización de GPU utilizando mediciones de hardware.

Formatos que descarten datos ociosos.

Los múltiples formatos de representación de matrices dispersas, tienen un doble beneficio. Estas poseen muy pocos elementos distintos de cero (menos al 10%). Los formatos buscan almacenar la información más valiosa minimizando el uso de recursos. Por un lado porque los valores en cero no interfieren en el resultado final del algoritmo, y por otro, se reduce considerablemente el uso de la memoria. En GPU tienen la ventaja que al reducir su consumo de memoria la información a transferir es menor y el acceso a memoria global (lenta), por el hilo *kernel* también disminuye. Como contra partida en

lugar de hacer una sola transferencia (la matriz) se deben transferir los vectores que tienen la información valida.

Agrupar transferencias.

Bajo ciertas condiciones existe la posibilidad de beneficiarse en transferir entre CPU y GPU una sola vez, en lugar de dividir la transferencia en varias más pequeñas. Se transfiere la misma cantidad de información, lo que varía es la forma (**Figura 28**).

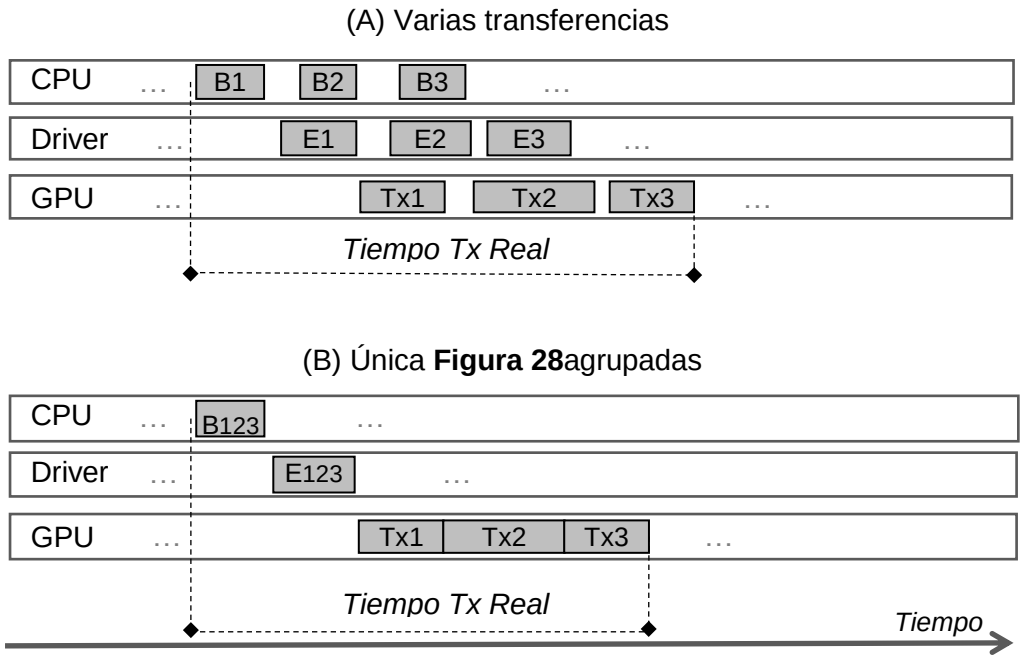


Figura 28 - Comparación de transferencias simple o seriada.

Esta situación se da cuando se preparan las pruebas sobre matrices dispersas enfocadas en los formatos de representación.

El formato CRS está formado por tres vectores y hay otros como el formato de bloques BSR que se compone de cuatro vectores. Cada vector es transferido en eventos separados (Figura 28 A). Mientras que existe el formato ELL con dos vectores, (Figura 28 B). Si se agrupan los vectores y se realiza una sola transferencia, se tiene la ventaja que se desprende de la secuencialidad de la CPU, que va invocando a las llamadas a biblioteca (API y Evento). Según sea la carga de la CPU esta puede generar tiempos de demora.

Reutilización de variables globales en cache.

Existen casos en el algoritmo sus datos están almacenados en memoria global y necesitan ser accedidas continuamente. Si estos son enviados a memoria interna del *kernel*, cuando el planificador no disponga de memoria, esta será enviada a la memoria local, que es de acceso lento. Para evitar esta situación se puede usar la memoria textura, que puede ser accedida por el CPU. La memoria de textura posee la particularidad de tener un nivel de memoria caché dentro del core GPU. Sí el dato almacenado en esta memoria solo es accedido por los hilos *kernel* una vez (**Figura 29**).

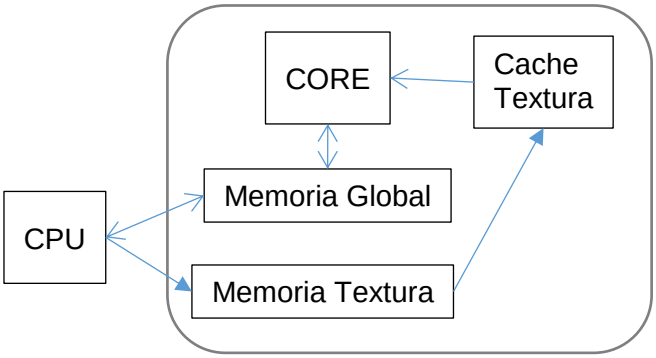


Figura 29 - Diagrama en bloques de memoria de textura

Relación de datos de entrada con lectura en memoria global tendiente a cero.

Este parámetro requiere de un conocimiento de las variables que procesa el algoritmo en GPU (**¡Error! No se encuentra el origen de la referencia.** y **¡Error! No se encuentra el origen de la referencia.**). Se puede trazar una relación entre los accesos de lectura en memoria global, con la cantidad teóricamente calculada de lo que se espera que use el algoritmo con las propiedades de los datos de entrada. Si esta relación es negativa, significa que se accede menos a memoria global de lo estipulado (el algoritmo no estaría procesando todos los elementos). Si la relación es positiva, se está accediendo a memoria global más de lo esperado (Puede indicar que se está utilizando la memoria global para cuentas intermedias). En cero es el caso ideal.

Relación de datos de salida con escritura en memoria global tendiente a cero.

Un caso similar ocurre cuando el lgoritmo escribe menos datos que los planificados. En este caso hay que comparar el valor teórico con el obtenido en los Contadores Hardware. Si es negativo el algoritmo se planifica mal, si es positivo está leyendo más cantidad de datos desde memoria global

En caso de resultar negativo, puede mejorarse con auxiliares (memoria interna) para que solo se acceda una vez finalizado el algoritmo, desde el auxiliar.

Configuración ajuste minimalista de dimensiones.

Por lo analizado en las pruebas de las bibliotecas BLAS. La mala elección del dimensionamiento está fijada por el algoritmo.

La cantidad de hilos creados tiene relación con la configuración de dimensiones y a su vez, con la forma en que toma lógicamente esa topología de bloques/grillas en caso de NVIDIA o Grupo Local/Global en ATI. Una forma de evitar el caso anterior y este, de hilos que no impactan en todos los datos del algoritmo, es alinear al tamaño de warp para NVIDIA o wavefront en ATI.

En la (Figura 30) se sobredimensiona la cantidad de hilos a ejecutar, dando hilos ociosos y desperdicio de recursos.

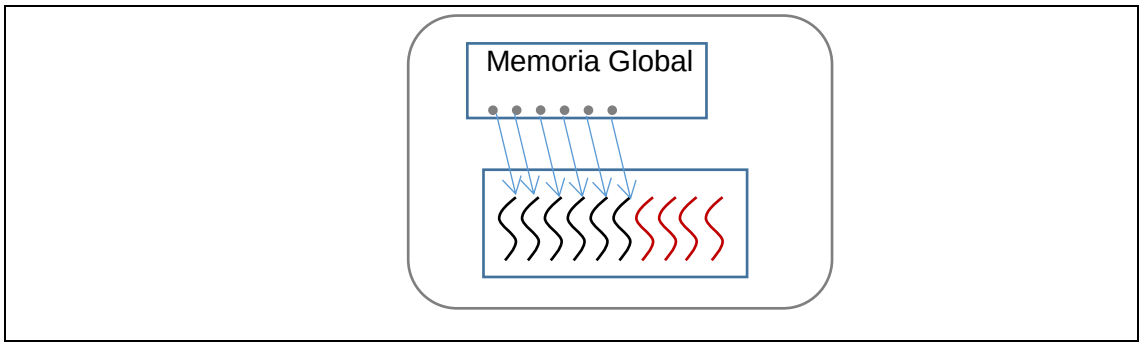


Fig. 30 - Ejemplo de hilos sobre-dimensionados.

Operación aritmética en su forma optima.

La operación aritmética principal del algoritmo SPMV es la multiplicación de un elemento del vector con todos los elementos de la columna de la matriz. Cada multiplicación se acumula en una variable intermedia, interviniendo las operaciones de multiplicación, suma e igualación. Por simplicidad suele escribirse en seudocódigo como:

*Acumulador += Matriz[i][n] * vector[n]*

Se ha verificado que aumenta la cantidad de instrucciones por ciclo si se codifica de esta forma.

*Acumulador = Acumulador + (Matriz[i][n] * vector[n])*

Esta mejora no es conmutativa y en cuanto a microinstrucciones del GPU la siguiente expresión da menos rendimiento.

*Acumulador = (Matriz[i][n] * vector[n]) + Acumulador*

La diferencia surge debido a que la GPU está preparada para procesar en forma matemática con un orden definido de operaciones (igualdad – suma – multiplicación).

Comparación de Resultados y elaboración de conclusiones.

El primer resultado de esta instigación, es el conocimiento adquirido sobre los contadores de hardware para GPU provistas por diferentes fabricantes**.**
El otro objetivos Importante, en esta investigación, es la construcción de la guía de optimización. Su aplicación se centra en la optimización reflejada con la biblioteca PARALUTION. Al incluir la forma que se planificaron las pruebas de rendimiento, junto con las etapas desarrolladas en la recolección de reportes. Plantea un ámbito que permite la extensión de la investigación.
Algunas pautas de la guía de optimización, requieren de un alto nivel de conocimiento del algoritmo a aplicar. Tales como la estructura interna de los datos de entrada y salida del algoritmo científico. Si enfoca la guía a un nuevo tema científico, como puede ser el procesamiento de búsqueda de patrones en genes humanos por GPU. Requiere un análisis profundo el nuevo tema, para lograr aplicar pautas como “Formatos que destacan datos ociosos“.

Este último punto no llega a ser una desventaja, sino una afirmación, ya que la complejidad de los algoritmos científicos crece día a día, necesitando conocerlos en profundidad para lograr mejorar su desempeño.

Divulgación de resultados y publicaciones.

La guía de Optimizaciones generada en este proyecto, puede ser consultada en la dirección de internet: http://www.so-unlam.com.ar/wiki/index.php?title=PUBLICO:Guia_optimizacion_GPU y la divulgación de las mediciones realizadas se encuentran en la dirección de internet http://so-unlam.com.ar/soa/investigacion/C156/.

4 Conclusiones.

Luego de la elaboración de una guía de optimización aplicable en algoritmos científicos que ejecutan en entornos GPU, se logró extender los lineamientos propuestos en las recomendaciones de mejores prácticas de programación en GPU, que es confeccionada por los fabricantes de ATI y NVIDIA. Cada tema expuesto en la guía, tiene como sustento las mediciones recolectadas por herramientas profiling. Relacionando los eventos y contadores hardware, pertenecientes a un contexto determinado, se encuentran las pautas que definen el comportamiento del rendimiento en algoritmos científicos.

La comparación de bibliotecas BLAS pudo realizarse en profundidad gracias a la información de contadores hardware. Las primeras comparaciones fueron expresadas en función de los tiempos de respuesta, al profundizar el análisis, se encontró con diferencias estructurales surgidas por la forma en ser desarrolladas.

Para la biblioteca VIENNACL solo pudo realizar las mediciones de profiller en ATI. Ya que la versión de la herramienta NVIDIA no soporta ejecutar código en lenguaje OpenCL. Esto nos limitó las pruebas planeadas, hubiera sido muy interesante ejecutar el mismo algoritmo con la misma biblioteca OpenCL en GPUs de distintos fabricante. Este es un tema que deberá explorar en próximos trabajos.

Cuando se comenzó a trabajar en la biblioteca Magma aparentaba ser la más completa de todas, ya que es la que posee la mayor cantidad de formatos de representación de matrices dispersas, con un problema, solo disponibles en su versión NVIDIA. Nuevamente quedó trunca la posibilidad de comparar las mejoras en distintas GPUs.

El caso de PARALUTION es la única biblioteca que se logró obtener contadores de hardware en las GPUs de los fabricantes ATI y NVIDIA. En ella enfocamos los objetivos de optimización, comparando cómo impacta en cada GPU. Llegando a verificar que en ATI se logra la mayor mejoría de desempeño.

La preparación de todas las pruebas involucradas requirió de la normalización de la información generada por los reportes de las herramientas de profiler. No se esperó que fuera tan dispareja. Hasta en un principio se planteó la posibilidad de encontrar la relación que permita igualar cada contador de hardware. No se pudo completar porque cada cual uno mide en distinta proporción. Lo que si se llegó a un punto en común, fueron los principales eventos de transferencia y ejecución de hilos. Otros eventos fueron descartados ya que no impactan en el contexto de la ejecución del algoritmo en GPU.

5 Bibliografía

[1] M. F. Piccoli, Computación de alto desempeño en GPU, 1ra ed., vol. 1, M. Sanguinetti, Ed., La Plata, Buenos Aires: Editorial de la Universidad Nacional de La Plata, 2011, p. 168.

[2] N. CUDA™, Programming Guide Versión 5.0, NVIDIA Corporation, 2012.

[3] NVIDIA CUDA, Best Practices Guide Version 5.0, NVIDEA Corporation, 2012.

[4] J. Siegel, J. Ributzka and L. Xiaoming, "CUDA memory optimizations for large data-structures in the Gravir Simulator," in *International Conference on Parallel Porcesing, Workshops*, Viena, Austria, 2009.

[5] R. Belleman, J. Bedorf and S. Zwart, "High Performance Direct Gravitational N-body Simulations on Graphics Processing Units," *New Astronomy,* vol. 13, no. 2, pp. 103-112, 2008.

[6] F. Tinetti y S. Martin, «Sequential optimization and shared and distributed memory parallelization in clusters:NBODY/Particle Simulation.,» de *Parallel and Distributed Computing and Systems (PDCS 2012).*, Las Vegas ,USA, 2012.

[7] NVIDIA Nsight™, Visual Studio Edition 3.0 User Guide, NVIDIA Corporation. 2013., 2013..

[8] Z. Fan, F. F. Qiu, A. Kaufman y S. Yoakum-Stover, «GPU Cluster for High Performance Computing,» de *ACM/IEEE Supercomputing 2004. ACM,*, Washington DC, USA, 2004.

[9] S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk y W. Hwu, «Optimization principles and application performance evaluation of a multithreaded GPU using CUDA,» de *ACM SIGPLAN Symposium on Principles and practice of parallel programming 2008*, New York, USA, 2008.

[10] G. Natale Primero, J. R. García Núñez y A. Correa Arroyave, «Natale Primero, Gino and García Núñez, Jesús Raf Utilización de matrices dispersas en el método de los elementos finitos,» *Ingeniería e Investigación- Universidad Nacional de Colombia,* vol. 1, nº 27, pp. 18-37 , 1992.

[11] I. P. Stanimirovi´c and M. B. Tasi, "Performance comparison of storage formats for sparse matrices," *FACTA UNIVERSITATIS (NIS),* vol. 1, no. 24, pp. 39-51, 2009.

[12] A. Grama, A. Gupta, G. Karypis and V. Kumar, Introduction to Parallel Computing, Pearson Adisson Wessley, 2003.

[13] J. Soto Mejía, G. R. Solarte Martínez y L. E. Muñoz Guerrero, «Matrices dispersas descripción y aplicaciones,» *Sciencia et Technica Universidad Tecnológica de Pereira,* vol. Vol 18, nº 1, 2013.

[14] N. Bell y M. Garland, «Efficient Sparse Matrix-Vector Multiplication on CUDA,» NVIDIA, 2008.

[15] M. J. Martín Santamaría, María J. Martín Santamaría – Factorización de cholesky modificada de matrices dispersas sobre multiprocesadores, Santiago de Compostela: Universidad de santiago de Compostela Facultad de Física, 1999..

[16] F. Diacu, "The solution of the n-body problem," *The Mathematical Intelligencer,* vol. 18, no. 3, pp. 66-70, 1996.

[17] S. Martín, F. Tinetti, N. Casas, G. De Luca and D. Giulianelli, "N-Body Simulation Using GP-GPU: Evaluating Host/Device Memory Transference Overhead," in *XIX Congreso Argentino de Ciencia de la Computación 2013*, Mar del Plata, 2013.

[18] F. Tinetti, S. Martin, F. Frati y M. Méndez, «“Optimization and parallelization experiences using hardware performance counters,» de *International Supercomputing Conference 2013*, Colima, Mexico, 2013.

[19] V. Kindratenko, «Numerical Computations with GPUs,» National Center for Supercomputing Applications (NCSA), Springer, 2014, pp. 4-13.

[20] Z. Reimert, «Human-induced vibrations on footbridges,» Delft University of Technology, 2015, pp. 43-173.

[21] K. K., «Sparse Matrix Algorithms Using GPGPU,» Tartu , University of Tartu , 2012, pp. 3-35.

[22] E. L. Kaltofen, The ‘Seven Dwarfs’ of Symbolic Computation, Raleigh, North Carolina 27695-8205, USA: Dept. of Mathematics, North Carolina State University,, 2014.

Sitios Web

[i] The OpenMP® API specification for parallel programming, http://openmp.org/wp/
[ii] OpenMP Specifications, http://openmp.org/wp/openmp-specifications/
[iii] OpenMP Application Program Interface Manual, http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf
[iv] Open MPI: Open Source High Performance Computing, http://www.open-mpi.org
[v] Open MPI v1.8.8 documentation, http://www.open-mpi.org/doc/v1.8/
[vi] PAPI, Performance Application Programming Interface, http://icl.cs.utk.edu/papi/index.html
[vii] NVIDIA Corporation, http://www.nvidia.com
[viii] The Khronos Group, https://www.khronos.org/

6 Producción científico-tecnológica

6.1 Publicaciones

6.1.1 Artículos

AUTOR (ES): Valiente Waldo, Díaz Federico, De Luca Graciela E., Giulianelli Daniel.
TÍTULO: Medición de rendimiento del algoritmo SPMV utilizando contadores de hardware para
GP GPU en arquitecturas no homogéneas.
FUENTE: Ciencia y Técnica Administrativa CyTA ejournal Técnica Administrativa ISSN
VOLUMEN: 15
TOMO:-.
NÚMERO 2.
PÁGINAS: 4
EDITORIAL Dr. Marcelo Claudio Perissé Director y Editor Responsable
LUGAR: La Matanza, Provincia de Buenos Aires - Argentina
FECHA 15-04-2016

6.1.2 Capítulos de libro

AUTOR (ES): Martín Sergio, Tinetti Fernando G., Casas Nicanor, De Luca Graciela, Giulianelli Daniel.
TÍTULO : N-Body Simulation Using GP-GPU: Evaluating Host/Device Memory Transference Overhead.
FUENTE: COMPUTER SCIENCE & TECHNOLOGY SERIES XIX ARGENTINE CONGRESS OF COMPUTER SCIENCE-SELECTED PAPERS.
ISBN: 978-987-1985-49-4
PÁGINAS inicial y final del trabajo. Ej. : 41-52
PÁGINAS: 325
COLECCIÓN: SELECTED PAPERS – CONGRESO CACIC
EDITORIAL: RED UNCI
FECHA: 2014
LUGAR DE EDICIÓN La Plata, Buenos Aires.

6.1.3 Congresos Internacionales, Nacionales, Simposios, Jornadas, otros

AUTOR (ES):Tinetti Fernando G., Martín Sergio
TÍTULO: Optimizing a CPU Algorithm through hardware Profiling Analysis.

TIPO: Conferencia
REUNIÓN: Computational Science and Computational Intelligence (CSCI), 2014
LUGAR: Las Vegas , USA.
FECHA REUNIÓN: 11-03-2014.
RESPONSABLE: American Council on Science and Education
TIPO DE TRABAJO: articulo completo.
FUENTE:
EDITORIAL IEEE Conference Publications, FECHA 2014, LUGAR DE EDICIÓN Las Vegas, USA

AUTOR (ES): Casas Nicanor, De Luca Graciela E., Giulianelli Daniel, Díaz Federico, Valiente Waldo, Martín Sergio.
TÍTULO: Aplicación de GP-GPU Computing para la optimización de algoritmos científicos mediante el uso de profiling de hardware.
TIPO: comunicación libre con exposición de poster
REUNION: WICC 2014 XVI Workshop de Investigadores en Ciencias de la Computación
LUGAR: Ushuaia, Argentina
FECHA REUNIÓN: 7y 8 de mayo 2014
RESPONSABLE: Red UNCI - Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur
TIPO DE TRABAJO: Artículo completo
FUENTE: WICC 2014 XVI Workshop de Investigadores en Ciencias de la Computación
ISBN 978-950-34-1084-4
PÁGINAS: 311-315.
EDITORIAL Universidad Nacional de Tierra del Fuego, Antártida e Islas del Atlántico Sur
LUGAR DE EDICIÓN: Ushuaia, Tierra del Fuego Argentina.
FECHA 2014.

AUTOR (ES): De Luca Graciela, Valiente Waldo, Díaz Federico, Giulianelli Daniel, Martín Sergio
TÍTULO: Análisis de rendimiento de algoritmos de resolución de problemas de matrices dispersas para GP-GPU Computing mediante el uso de profiling de hardware.
TIPO: comunicación libre con exposición de poster
REUNIÓN: XVII Workshop de Investigadores en Ciencias de la Computación
LUGAR: Salta. Argentina
FECHA REUNIÓN: 16-04-2015/17-04-2015
RESPONSABLE: Red UNCI - Universidad nacional de Salta
TIPO DE TRABAJO: Artículo completo
FUENTE: Anales del XVII Workshop de Investigadores en Ciencias de la Computación
ISBN 978-987-633-134-0
EDITORIAL C.I.D.I.A - Centro de Investigación y Desarrollo en Informática Aplicada Universidad Nacional de Salta, FECHA 16 de Abril de 2015, LUGAR DE EDICIÓN Salta, Argentina

Las publicaciones del trabajo se encuentran en el “ANEXO D Publicaciones”

7 ANEXOS

ANEXO - A - INFORME DE INSTALACIÓN

Obtener información sobre instalación y programación en hardware GPU

En el mercado actual existen dos grandes tecnologías para la programación general en unidades de procesamiento gráfico: OpenCL y CUDA. En este apartado procederemos a especificar en primer término los pasos necesarios para tener una instalación funcional de ambas tecnologías tanto en el sistema operativo Windows 7 como en el sistema operativo GNU/Linux - Distribución Ubuntu 12.04. Describiremos en detalle cada punto a tener en cuenta para poder conseguir el objetivo previamente planteado resaltando aquellas cosas que se nos presentaron como dificultades producto de nuestra experiencia realizando dicha actividad. Posteriormente continuaremos con la descripción de los diferentes modelos de programación sobre hardware GPU propuestos por ambas tecnologías.

Antes de entrar en el detalle específico del cómo instalar cada tecnología en los diferentes entornos nos parece pertinente mencionar que existen para ambas tecnologías implementaciones o adaptaciones para diferentes lenguajes de programación como por ejemplo: C/C++, FORTRAN, Haskell, Python, Ruby, Java, entre otros. De todos los lenguajes disponibles para la programación en hardware GPU cada uno de ellos presenta ventajas y desventajas unos respecto de otros, y que dependen en gran medida de los objetivos del software producido, de los entornos de ejecución del mismo y principalmente por los objetivos del proyecto. En nuestro caso optamos por utilizar el lenguaje de programación C/C++ debido a:

Estandarización estable del lenguaje
Gran variedad de entornos ejecución
Gran disponibilidad de herramientas de desarrollo y compiladores
Conocimiento profundo del lenguaje por parte del equipo de investigación
Fácil transferencia hacia la comunidad científica dedicada al área
Fácil transferencia hacia el aula producto de la familiaridad de los estudiantes con el mismo.
Obtener mayor nivel de detalle sobre el funcionamiento de las tecnologías más allá de las interfaces provistas por los fabricantes debido al bajo/medio nivel de los lenguajes.
Generar optimizaciones agnósticas del lenguaje producto al bajo/medio nivel de los mismos.

Instalación de CUDA sobre entornos con el sistema operativo Windows

Para instalar CUDA sobre entornos con el sistema operativo Windows (XP o superior) únicamente se debe descargar los controladores actualizados para la tarjeta gráfica de nuestro computador provistos por NVIDIA desde http://www.nvidia.es/Download/index.aspx?lang=es y descargar CUDA Toolkit desde https://developer.nvidia.com/cuda-downloads para la versión del sistema operativo instalada. Posteriormente se instalará según lo indicado en la guía oficial NVIDIA disponible en http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-microsoft-windows/#axzz3KjwZG857

Siguiendo los pasos indicados desde la guía oficial provista por el fabricante para instalar el CUDA tuvimos alguno de los siguientes problemas de instalación:

Los controladores de dispositivo provistos desde los repositorios de Ubuntu 12.04 no eran compatibles con la versión de CUDA 5.5 haciendo la instalación obsoleta.
La instalación de los drivers dejaba obsoleta la interfaz gráfica provista por el sistema operativo.
Los drivers eran compatibles con CUDA 5.5 durante la ejecución de programas CUDA luego de un período de tiempo los mismos lanzaban errores fatales producto de una mala instalación de los paquetes sobre el sistema.

Por lo tanto para tener un entorno funciona y completamente libre de errores ejecutando CUDA debimos de realizar los siguientes pasos:

Descargar los drives privativos provistos por NVIDIA de la unidad de procesamiento gráfico utilizada en nuestro computador y compatible con el sistema operativo Linux desde la siguiente URL: http://www.nvidia.es/Download/index.aspx?lang=es.

Descargar CUDA Toolkit (Archivo RUN) para la versión y distribución Linux utilizada, en nuestro caso Ubuntu 12.04 LTS, desde la siguiente URL: https://developer.nvidia.com/cuda-downloads
Descomprimir el archivo descargado en el punto número 1.

Damos permisos de ejecución, lectura y escritura recursivamente al directo previamente descomprimido:

user@machine:~$sudo chmod 777 -R /path/NVIDIA.run*↲

Instalamos los paquetes básicos necesarios para proseguir con la instalación manual de los controladores y del ambiente de ejecución CUDA:

user@machine:~$sudo apt-get install build-essential↲
user@machine:~$sudo apt-get install linux-headers-$(uname -r)↲

Eliminar todo controlador de la tarjeta gráfica desde el sistema utilizando el siguiente comando:

user@machine:~$sudo apt-get remove --purge nvidia*↲

Ejecutar el siguiente comando para eliminar los paquetes indicados:

user@machine:~$sudo apt-get remove --purge xserver-xorg-video-nouveau↲
user@machine:~$sudo apt-get remove --purge unity*↲
user@machine:~$sudo apt-get remove --purge ubuntu-desktop↲

Abrimos el siguiente archivo:

user@machine:~$sudo gedit /etc/modprobe.d/blacklist.conf↲

Nos dirigimos al final del archivo y escribimos

blacklist vga16fb
blacklist nouveau
blacklist rivafb
blacklist nvidiafb
blacklist rivatv

Guardamos los cambios y cerramos el fichero.

Presionamos **CTRL+ALT+F1**, cambiando a una terminal en modo línea de comandos.

Nos logueamos con nuestro usuario y password.

Detenemos el servicio de interfaz gráfica:

user@machine:~$sudo service lightdm stop↲

Cambiamos el nivel de ejecución del sistema

user@machine:~$sudo init 3↲

Procedemos a instalar los controladores

user@machine:~$sudo sh NVIDIA.run *↲

Reiniciamos

user@machine:~$sudo reboot↲

Cambiamos el nivel de ejecución del sistema

user@machine:~$sudo init 3↲

Ejecutar el archivo de instalación de CUDA y continuar según lo indicado por el instalador:

user@machine:~$sudo sh ./cuda_6.5.14_linux_64.run↲

Reiniciamos nuevamente:

user@machine:~$sudo sh reboot↵

Finalmente reinstalamos los paquetes:

user@machine:~$sudo apt-get install ubuntu-desktop↵
user@machine:~$sudo apt-get install unity*↵

Instalación de OpenCL sobre entornos con el sistema operativo Windows

La instalación de OpenCL en entornos Windows va de la mano a la utilizada en CUDA. Ya que el mismo driver de Nvidia para CUDA posee la biblioteca de OpenCL entre otras.

Entrando en la página https://developer.nvidia.com/cuda-downloads se puede obtener descargar CUDA Toolkit. Este se integra al Visual Studio la configuración para los proyectos que utilicen OpenCL es automática.

Instalación de OpenCL sobre entornos con el sistema operativo Linux

La instalación va de la mano a la utilizada en CUDA. Ya que el mismo driver de Nvidia posee la biblioteca de OpenCL.

1. Instalar Ubuntu (la versión utilizada es la 12.04 64bits).
2. Instalar CUDA (ya que el equipo posee placa nvidia) en este caso la versión 5.5.
3. Bajarlo desde “https://developer.nvidia.com/cuda-downloads”.
4. Elegir la opción del sistema operativo Ubuntu 12.04 y tipo “RUN”
5. Luego ejecutarlo de la siguiente forma “sudo sh cuda_5.5.22_linux_64.run”.
6. Luego fue necesario instalar “g++” desde el gestor de paquetes de Ubuntu. Ya que se trataba de una distribución recién instalada.
7. Para configurar la definición de biblioteca que utilizara OpenCL al ejecutar.

Para saber las bibliotecas disponibles, en una terminal ejecutar “locate opencl”. De lo que muestra el resultado marcamos 3 líneas útiles en la configuración:

```
waldo@ubuntu-black:~$ locate opencl
/usr/lib/libnvidia-opencl.so.1
/usr/lib/libnvidia-opencl.so.319.37
/usr/lib32/libnvidia-opencl.so.1
/usr/lib32/libnvidia-opencl.so.319.37
/usr/local/cuda-5.5/include/CL/opencl.h
/usr/share/app-install/desktop/openclipart.desktop
/usr/share/gtksourceview-3.0/language-specs/opencl.lang
waldo@ubuntu-black:~$
```

Biblioteca de 64bits: /usr/lib/libnvidia-opencl.so.319.37

Biblioteca de 32bits: /usr/lib32/libnvidia-opencl.so.319.37.

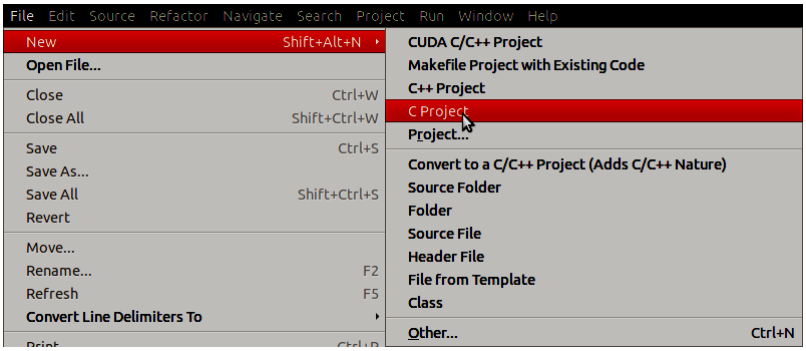
Archivo de definición C: /usr/local/cuda-5.5/include/CL/opencl.h.

Generar el archivo de definición de biblioteca, si no existe, "vi /etc/OpenCL/vendors/nvidia.icd". Con el texto plano "/usr/lib/libnvidia-opencl.so.1" (es un link a la biblioteca de 64bits ubicada en /usr/lib). Este archivo es utilizado por el dispositivo gráfico nvidia, para saber el tipo de biblioteca que utilizara el programa en ejecución.

La ubicación de los archivos de definición del lenguaje C. Es útil a la hora de compilar programas que utilicen OpenCL. Para luego la ubicación de las bibliotecas esta especificada en la variable LD_LIBRARY_PATH.

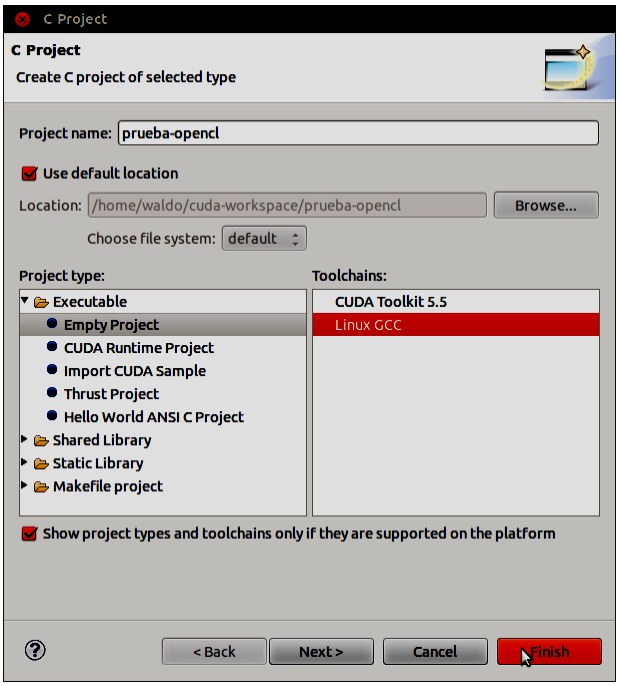
Configuración de OpenCL con eclipse (nsight) sobre entornos con el sistema operativo Linux

Para iniciar Nsight, se encuentra en "/usr/local/cuda-5.5/bin".
Al iniciar pide indicar donde se encuentra el espacio de trabajo
Al iniciar ir a [File]-> [New] → [C Project]

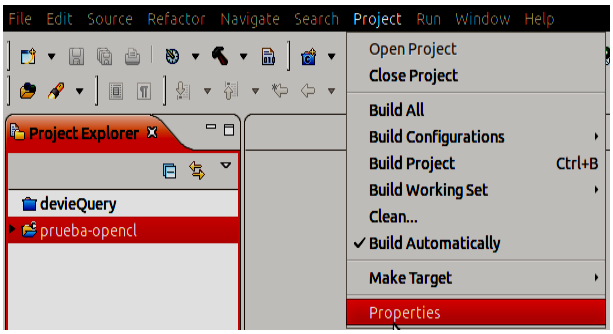


En la configuración del nuevo proyecto indicar:
Project Name: Nombre del proyecto a crear. En este caso "prueba-opencl".
Project type: El tipo de proyecto a crear debe ser vacío "Empty Project".

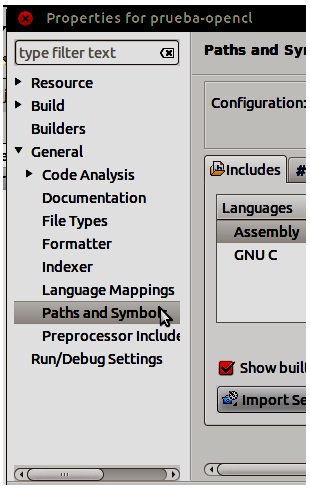
Toolchairs: Compilador que utilizara, se debe indicar el de “Linux GCC”. Ya que OpenCl se compila en tiempo de ejecución.



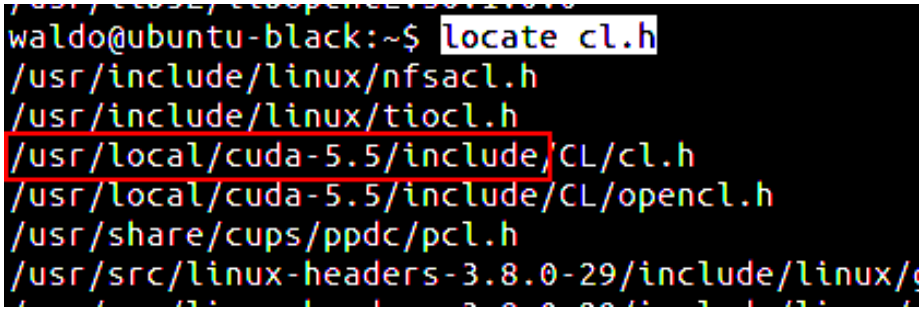
Sobre el proyecto creado se debe indicar las variables de compilación.
Para lo cual seleccionando el proyecto recién creado. Se debe ir a [Project] → [Properties]



En las propiedades del proyecto ir a las opciones [Paths and Symbols]



En la pestaña [Include] para el compilador de c agregar el PATH al directorio donde se encuentra cabecera de openCL.
Para buscar en donde se encuentra el archivo cabecera. En una terminal ejecutar el comando “locate cl.h”



ANEXO B Cursos realizados en otras Universidades

UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

Calle 120 y 50 – 2do piso (1900) La Plata Pág. 1 de 3 http://postgrado.info.unlp.edu.ar

TEL-FAX: (54) 221-4273235 E-Mail: postgrado@lidi.info.unlp.edu.ar

Predicción de performance y ejecución eficiente en sistemas de cómputo paralelos (HPC): clúster y cloud

Carrera: Doctorado en Ciencias Informáticas

Año: 2014

Duración: 70 hs

Profesor a Cargo: Dr Emilio Luque, Dra Dolores Rexachs, Dr Javier Balladini, Dr Álvaro Wong

OBJETIVOS GENERALES:

El objetivo general del curso se orientó hacia la ejecución eficiente de aplicaciones sobre Cluster o Cloud, para ello analizamos la manera de caracterizar el comportamiento de las aplicaciones paralelas de paso de mensajes y utilizar esta información para seleccionar los recursos adecuados para configurar el sistema de cómputo paralelo (Clúster o Cloud), de modo que conseguimos cumplir con las especificaciones de velocidad (tiempos de ejecución), productividad y eficiencia computacional y energética. Como herramienta para el análisis de las aplicaciones utilizamos PAS2P.

PROGRAMA

1. Computación de altas prestaciones (HPC): Sistemas paralelos y cloud computing

a. Computadores Paralelos: sistemas Multicore, Cluster y Multiclusters

b. Cloud computing

c. Escalabilidad de Sistemas Paralelos

2. Aplicaciones paralelas

a. Aplicaciones paralelas: modelos y paradigmas de programación paralela

b. Análisis del comportamiento de las aplicaciones

c. Caracterización de las aplicaciones paralelas

d. Firmas de las aplicaciones paralelas

3. Evaluación de las prestaciones

a. Modelos, métricas y herramientas para la evaluación de prestaciones

b. Benchmarks

c. Predicción de prestaciones: objetivo y usos

d. PAS2P: una herramienta para analizar y predecir la ejecución de aplicaciones paralelas

e. Análisis de la escalabilidad

4. Ejecución eficiente y sintonización en computadores paralelos y cloud

a. Eficiencia computacional y eficiencia energética

b. Metodología para la ejecución eficiente de aplicaciones SPMD

c. Planificación de tareas

d. El método de la “Escalabilidad Inversa”

5. Predicción utilizando Minería de Datos (Big Data)

a. Aplicaciones no determinísticas (Data dependent)

b. Algoritmo paralelo del problema del viajante: ¿Cuál es su Complejidad?

c. Knowledge discovery methodology

6. Sistemas de visualización de ejecuciones paralelas

a. Monitorización en sistemas paralelos y distribuidos. La presentación de datos.

b. Técnicas de visualización de ejecuciones de aplicaciones paralelas

c. Errores frecuentes en visualización de prestaciones

d. Análisis de herramientas de visualización de ejecuciones

BIBLIOGRAFÍA DEL CURSO:

1- Almeida, F.; Giménez, D.; Mantas J.M.; Vidal A.M. (2008). Introducción a la programación paralela. Paraninfo

2- Grama A., Gupta A., Karypis G., Kumar V. (2003). Introduction to Parallel Computing. (2nd Edition). Pearson Addison Wesley.

3- Lilja, David J. (2005). Measuring Computer Performance: A Practitioner's Guide. Cambridge University Press.

Bibliografía específica

Caracterización de las aplicaciones paralelas, Alexandre Otto Strube, Dolores Rexachs, Emilio Luque: Software Probes: A Method for Quickly Characterizing Applications' Performance on Heterogeneous Environments. ICPP Workshops 2009: 262-269

PAS2P: una herramienta para analizar y predecir la ejecución de aplicaciones paralelas, Javier Panadero, Alvaro Wong, Dolores Rexachs, Emilio Luque: A Tool for Selecting the Right Target Machine for Parallel Scientific Applications. ICCS 2013: 1824-1833

Eficiencia computacional y eficiencia energética, Javier Balladini, Remo Suppi, Dolores Rexachs, Emilio Luque: Impact of parallel programming models and CPUs clock frequency on energy consumption of HPC systems. AICCSA 2011: 16-21

Metodología para la ejecución eficiente de aplicaciones SPMD, Muresano, R.; Rexachs, D.; Luque, E. (2010). Methodology for Efficient Execution of SPMD Applications on Multicore Environments. CCGRID pp 185-195. 2010.

Predicción utilizando Minería de Datos (Big Data), Paula Cecilia Fritzsche, Dolores Rexachs, Emilio Luque: Defining Asymptotic Parallel Time Complexity of Data-dependent Algorithms. New Generation Comput. 32(2): 123-144 (2014)-Eduardo Cabrera, Manel Taboada, Ma Luisa Iglesias, Francisco Epelde, Emilio Luque: Simulation Optimization for Healthcare Emergency Departments. ICCS 2012: 1464-1473

Sistemas de visualización de ejecuciones paralelas, Raj Jain (1991). Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling . Wiley Computer Publishing, John Wiley & Sons, Inc. ISBN: 0471503363. (Capítulo 10 The art of data presentation). http://www.ensenadamexico.net/hector/mest/Art_Of_Computer_Systems_Performance_Analysis_Techniques_For_Experimental_Measurements_Simulation_And_Modeling-Raj_Jain.pdf

UNIVERSIDAD NACIONAL DE LA PLATA FACULTAD DE INFORMÁTICA

Calle 120 y 50 – 2do piso (1900) La Plata Pág. 1 de 2
http://postgrado.info.unlp.edu.ar TEL-FAX: (54) 221-4273235 E-Mail: postgrado@lidi.info.unlp.edu.ar

PROCESAMIENTO INTELIGENTE DE BIG DATA: TÉCNICAS, APLICACIONES Y NUEVOS RETOS

Año 2014

Duración: hs 20
Profesor a Cargo***: Olivas, Jose A.***

PROGRAMA

SOME INTELLIGENT ALGORITHMS FOR BIG DATA:

Data and Information Fusion

Genetic Algorithms

Machine Learning

Natural Language Processing

Dimensionality Reduction Techniques

Multidimensional Big Data

SOME INTELLIGENT BIG DATA SEARCH & MINING METHODS:

Data Mining

Social Networks

Data Science

Web Search and Information Mining

Scalable Search Architectures

Cleaning Big Data (noise reduction), Acquisition & Integration

Visualization Methods for Search

Time Series Analysis

Recommendation Systems

Graph Mining and Other Similar Technologies

SOME APPLICATIONS OF BIG DATA:

Applications in Science, Engineering, Healthcare, Visualization, Business, Education, Security, Humanities, Bioinformatics, Health , Informatics, Medicine, Finance, Law, Transportation, Retailing, Telecommunication, all Search-based applications, ...

SOME BIG DATA FUNDAMENTALS:

Computational Science

Computational Intelligence

BIBLIOGRAFÍA

Adriaans, P. W.; Zantinge, D.: Data Mining. Addison-Wesley, 1996.

Berry, M. J. A.; Linoff, G.: Data Mining Techniques. Wiley Computer Publishing. New York, 1996.

Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.: The KDD Process for Extracting Useful Knowledge from Volumes of Data. Communications of the ACM, November 1996/ Vol 39, Nº 11, pp. 27 – 34.

Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.; Uthurusamy, R. (Eds): Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press, Cambridge MA, 1996.

Joyanes, Luis: Big Data - Análisis de grandes volúmenes de datos en organizaciones, Alfaomega, 2013.

Mayer-Schönberger, V.; Cukier, K.: Big data. La revolución de los datos masivos. Turner 2013.

Olivas, J. A.: Búsqueda eficaz de información en la Web, EDULP, 2011.

Piatetsky-Shapiro, G.; Frawley, W.: Knowledge Discovery in Databases. AAAI/MIT Press, Cambridge MA, 1991.

Siegel E.: Analítica predictiva. Predecir el futuro utilizando Big Data. Anaya Multimedia-Anaya Interactiva, 2013.

ANEXO C DETALLES DE BIBLIOTECAS UTILIZADAS.

OpenMP

La sintaxis básica que nos encontramos en una directiva de OpenMP es para C/C++:
En el bloque dónde se incluye una directiva OpenMP esto implica que dicho bloque posee una sincronización obligatoria. Es decir, el bloque de código se marcará como paralelo y se lanzarán hilos según las características que nos dé la directiva, y al final de ella habrá una barrera para la sincronización de los diferentes hilos (salvo que implícitamente se indique lo contrario con la directiva nowait). Este tipo de ejecución se denomina fork-join.

```
# pragma omp <directiva> [cláusula [ , ...] ...]

Para FORTRAN

!$OMP  PARALLEL <directiva> [cláusulas]

    [ bloque de código ]

!$OMP END <directiva>
```

Directivas más destacadas

parallel: indica que parte del código podrá ser ejecutada por varios threads

for: indica que a cada thread se le asigna una parte del for a ejecutar (paralelismo de datos)

sections: indica que cada sección será ejecutadas por un thread(paralelismo funcional)

single: indica que esta sección será ejecutada por un único thread

master: indica que esta sección será ejecutadas por un master-thread

parallel for: combinación de parallel (fork)+ sections (asignación de cada sección a un thread)

critical: indica que ala sección será de exclusión mutua.

Barrier: indica la necesidad de sincronización de los threads en este punto.

Clausulas a destacar

Schedule (static | dynamic | guided | runtime [, chunk]):

Determina de qué forma se realizará la asignación del trabajo a los threads
private (variable [,variable, …])

Indica que las variables que aparecen en la cláusula serán privadas, cada thread tendrá una copia independiente.

Funciones a destacar

omp_set_num_threads: Fija el número de threads que se crearán en los forks de las regiones paralelas de código.

omp_get_num_threads: Devuelve el número de threads activos en la región paralela actual.

omp_get_thread_num: Devuelve el identificador del thread que lo ejecuta.

omp_get_num_procs: Devuelve el número de cores del multi-core en el que se está ejecutando el thread.

OpenMPI

Las aplicaciones de MPI son bibliotecas

Todas las operaciones son ejecutadas realizando invocaciones a funciones

Se debe incluir las siguientes cabeceras (header) de archivos

mpi.h en C

mpif.h para Fortran77 y Fortran90

Encontramos los siguientes elementos:

Los archivos cabecera (headers)

El entorno de comunicación (MPI Communicator)

Formato de las funciones de MPI

Tamaño del entorno de comunicación

El identificador de cada proceso dentro del entorno de comunicación Inicialización y finalización del programa

La lista de comandos disponibles en la última versión se encuentra dentro de la documentación en la página oficial [v]

PAPI

Algunos de los contadores hardware a los que se puede acceder mediante la utilización de PAPI son los siguientes:

Nombre de contador	Descripción
PAPI_L1_DCM	Level 1 data cache
PAPI_L1_ICM	Level 1 instruction cache
PAPI_L2_DCM	Level 2 data cache
PAPI_L2_ICM	Level 2 instruction cache
PAPI_L3_DCM	Level 3 data cache
PAPI_L3_ICM	Level 3 instruction cache
PAPI_L1_TCM	Level 1 cache misses
PAPI_L2_TCM	Level 2 cache misses
PAPI_L3_TCM	Level 3 cache misses
PAPI_CA_SNP	Requests for a snoop
PAPI_CA_SHR	Requests for exclusive access
PAPI_CA_CLN	Requests for exclusive access
PAPI_CA_INV	Requests for cache line
PAPI_CA_ITV	Requests for cache line
PAPI_L3_LDM	Level 3 load misses
PAPI_L3_STM	Level 3 store misses
PAPI_BRU_IDL	Cycles branch units are
PAPI_FXU_IDL	Cycles integer units are
PAPI_FPU_IDL	Cycles floating point units
PAPI_LSU_IDL	Cycles load/store units are
PAPI_TLB_DM	Data translation lookaside buffer
PAPI_TLB_IM	Instruction translation lookaside buffer
PAPI_TLB_TL	Total translation lookaside buffer
PAPI_L1_LDM	Level 1 load misses
PAPI_L1_STM	Level 1 store misses
PAPI_L2_LDM	Level 2 load misses
PAPI_L2_STM	Level 2 store misses
PAPI_BTAC_M	Branch target address cache
PAPI_PRF_DM	Data prefetch cache misses
PAPI_L3_DCH	Level 3 data cache
PAPI_TLB_SD	Translation lookaside buffer shootdowns
PAPI_CSR_FAL	Failed store conditional instructions
PAPI_CSR_SUC	Successful store conditional instructions
PAPI_CSR_TOT	Total store conditional instructions
PAPI_MEM_SCY	Cycles Stalled Waiting for
PAPI_MEM_RCY	Cycles Stalled Waiting for
PAPI_MEM_WCY	Cycles Stalled Waiting for
PAPI_STL_ICY	Cycles with no instruction
PAPI_FUL_ICY	Cycles with maximum instruction
PAPI_STL_CCY	Cycles with no instructions
PAPI_FUL_CCY	Cycles with maximum instructions
PAPI_HW_INT	Hardware interrupts
PAPI_BR_UCN	Unconditional branch instructions
PAPI_BR_CN	Conditional branch instructions
PAPI_BR_TKN	Conditional branch instructions taken
PAPI_BR_NTK	Conditional branch instructions not
PAPI_BR_MSP	Conditional branch instructions mispredicted
PAPI_BR_PRC	Conditional branch instructions correctly
PAPI_FMA_INS	FMA instructions completed
PAPI_TOT_IIS	Instructions issued
PAPI_TOT_INS	Instructions completed

Nombre de contador	Descriptor
PAPI_INT_INS	Integer instructions
PAPI_FP_INS	Floating point instructions
PAPI_LD_INS	Load instructions
PAPI_SR_INS	Store instructions
PAPI_BR_INS	Branch instructions
PAPI_VEC_INS	Vector/SIMD instructions
PAPI_RES_STL	Cycles stalled on any
PAPI_FP_STAL	Cycles the FP unit(s)
PAPI_TOT_CYC	Total cycles
PAPI_LST_INS	Load/store instructions completed
PAPI_SYC_INS	Synchronization instructions completed
PAPI_L1_DCH	Level 1 data cache
PAPI_L2_DCH	Level 2 data cache
PAPI_L1_DCA	Level 1 data cache
PAPI_L2_DCA	Level 2 data cache
PAPI_L3_DCA	Level 3 data cache
PAPI_L1_DCR	Level 1 data cache
PAPI_L2_DCR	Level 2 data cache
PAPI_L3_DCR	Level 3 data cache
PAPI_L1_DCW	Level 1 data cache
PAPI_L2_DCW	Level 2 data cache
PAPI_L3_DCW	Level 3 data cache
PAPI_L1_ICH	Level 1 instruction cache
PAPI_L2_ICH	Level 2 instruction cache
PAPI_L3_ICH	Level 3 instruction cache
PAPI_L1_ICA	Level 1 instruction cache
PAPI_L2_ICA	Level 2 instruction cache
PAPI_L3_ICA	Level 3 instruction cache
PAPI_L1_ICR	Level 1 instruction cache
PAPI_L2_ICR	Level 2 instruction cache
PAPI_L3_ICR	Level 3 instruction cache
PAPI_L1_ICW	Level 1 instruction cache
PAPI_L2_ICW	Level 2 instruction cache
PAPI_L3_ICW	Level 3 instruction cache
PAPI_L1_TCH	Level 1 total cache
PAPI_L2_TCH	Level 2 total cache
PAPI_L3_TCH	Level 3 total cache
PAPI_L1_TCA	Level 1 total cache
PAPI_L2_TCA	Level 2 total cache
PAPI_L3_TCA	Level 3 total cache
PAPI_L1_TCR	Level 1 total cache
PAPI_L2_TCR	Level 2 total cache
PAPI_L3_TCR	Level 3 total cache
PAPI_L1_TCW	Level 1 total cache
PAPI_L2_TCW	Level 2 total cache
PAPI_L3_TCW	Level 3 total cache
PAPI_FML_INS	Floating point multiply instructions
PAPI_FAD_INS	Floating point add instructions
PAPI_FDV_INS	Floating point divide instructions
PAPI_FSQ_INS	Floating point square root
PAPI_FNV_INS	Floating point inverse instructions