Análisis de rendimiento de algoritmos de resolución de problemas de matrices dispersas para GP-GPU Computing mediante el uso de profiling de hardware.

Graciela De Luca, Waldo Valiente, Federico Díaz
Nicanor Casas, Daniel Giulianelli, Sergio Martín

Universidad Nacional de la Matanza

Departamento de Ingeniería e Investigaciones Tecnológicas

Dirección: *Florencio Varela 1703* - Código Postal: 1754

{gdeluca, wvaliente, fdiaz, ncasas dgiulian, smartin}@ing.unlam.edu.ar

Resumen

Las arquitecturas many-core proveen un gran potencial, para la optimización de algoritmos científicos gracias a su alto nivel de paralelismo y simplicidad. En este caso particular se utilizará programación de procesamiento general para clusters de computadoras con placas GPU[1].

Primeramente el proyecto se enfoca en el análisis de matrices dispersas, dónde la mayor parte de los datos que contiene, no son relevantes al cálculo final deseado. Por lo tanto utilizando ciertas técnicas de representación en las matrices, se puede evitar realizar operaciones sobre datos a los cuales se les puede determinar su resultado a priori.

En la segunda etapa, se utilizan y analizan implementaciones de biblioteca que aprovechen diferentes sistemas de representación de estas matrices, buscando realizar las optimizaciones oportunas de ser posible, siendo necesario realizar pruebas en las dos tecnologías de GPU más reconocidas buscando identificar posibles deficiencias en la implementación de cada solución.

Finalmente mediante contadores de hardware se analizan en forma precisa, si las optimizaciones a realizar, y las ya realizadas, realmente producen una diferencia apreciable en los tiempos de ejecución y uso de recursos de las bibliotecas.

En este artículo se presentan las tres etapas que componen este proyecto de investigación.

Palabras clave: GP-GPU, Profiling, matrices dispersas, bibliotecas numéricas, representación de matrices

Contexto

Esta Línea de Investigación es parte del proyecto "Aplicación de GP-GPU

[1] Graphics Processing Unit

Computing para la optimización de algoritmos científicos mediante el uso de profiling de hardware ", dependiente de la Unidad Académica del Departamento de Ingeniería e Investigaciones Tecnológicas perteneciente al programa de Investigaciones PROINCE de la Universidad Nacional de La Matanza, el cual es continuación de otros proyectos, entre ellos del proyecto de "Utilización de tecnologías adaptativas para la optimización del uso de recursos y eficiencia energética en clúster de servidores GPU y CPU" [5].

Introducción

En la actualidad, existen un gran número de problemas científicos, que utilizan conceptos asociados a matrices dispersas. Estas matrices, tienen como particularidad, que la densidad de datos no utilizables o no relevantes, es elevada.

Cuando un algoritmo utiliza como dato de entrada, una matriz de este tipo, si no realiza un procesamiento especializado, realizará una gran cantidad de cómputo innecesario. Los algoritmos deben tener cierto conocimiento del contexto de datos donde están operando.

En base a esta problemática, se desarrollaron técnicas matemáticas específicas para el tratamiento de matrices dispersas (también llamadas ralas). Estas matrices, son muy utilizadas en el ámbito científico, ya que tratan de problemas de localización de la información a procesar, es decir, muestras pequeñas, pero correspondientes a un espacio muestral elevado.

Estas matrices son comúnmente utilizadas en Astronomía, Física de materiales, Estadística, Inteligencia artificial, entre otras. Usualmente son matrices de un tamaño elevado, donde los datos útiles son realmente pocos (muchas veces por debajo del 10% del total de los elementos de la matriz).

Las técnicas de representación [8] pueden producir interesantes resultados a la hora de procesar las matrices. Entre los más representativos se encuentran los enumerados en la tabla 1.

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

Tabla 1. Diferentes formatos de representación de matrices dispersas

Los más comúnmente utilizados de la lista son:

ELL: el método de conversión del formato ELL consiste en conocer a priori la máxima cantidad de datos útiles o elementos no cero (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.

COO: El formato de almacenamiento por coordenadas consiste en almacenar la información que guarda cada ENC. Para esto utiliza 3 vectores alineados que contienen la información de valor, fila y columna de los ENC de la matriz dispersa.

CRS: Se dice que es similar al *formato COO*, para cuando esta ordenado por filas. Esto es porque el *formato CRS* comprime las filas implícitamente y guardando su información 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.

HYB: Método Hibrido. Este método suele combinar diferentes métodos de representación, como por ejemplo combinar CRS con ELL.

En la actualidad existen diversas bibliotecas, que están en desarrollo, y permiten la utilización de los diferentes tipos de representación de matrices, para aprovechar las cualidades de las mismas. Las bibliotecas que se utilizan, son las que más comúnmente se utilizan en investigaciones en universidades de todo el mundo.

Las bibliotecas seleccionadas para los casos de matrices dispersas fueron las siguientes:

- PARALUTION [9] – Se utilizó la versión 1.0.0 publicado el 27 feb. 2015.
- VexCL [10] – Utilizándose la versión 1.3.2 publicada el 24 sep. 2014.
- VIENNACL [11] – Con versión 1.6.2 disponible desde 11 dic 2014.

Cada una de estas bibliotecas, posee diferentes métodos de representación de matrices, y todas poseen la operación básica Matriz x Vector, que es la que nos interesa estudiar en nuestro caso.

Estas se encuentran implementadas para ser utilizadas con las dos tecnologías de GPU existentes, que se utilizan comúnmente. Del fabricante ATI® OpenCL™[1] y del fabricante NVIDIA® CUDA™ [2][7]. Se utilizan las implementaciones de estas bibliotecas para ambas tecnologías. Esto permite detectar posibles deficiencias que resulten en áreas de optimización interesantes para atacar.

Líneas de Investigación, Desarrollo e Innovación

Actualmente, nos encontramos trabajando sobre los siguientes aspectos:

- Análisis de bibliotecas numéricas para resolución de operaciones algebraicas con matrices dispersas.
- Aplicación de optimizaciones a problemas de simulaciones físicas, en conjunto con investigadores de la Universidad Nacional de La Plata [6].
- Utilización de contadores de hardware como método de análisis y validación de potenciales optimizaciones [3][4].

Resultados y Objetivos

Las pruebas consistieron en la multiplicación de matriz por vector (llamada por sus siglas en ingles SpMV *"Sparse Matrix-Vector Multiplication"*), utilizando matrices de 65025x65025 con 453137 elementos no nulos, y realizando 1024 veces la misma operación, obteniendo promedios de ejecución. Los

resultados se pueden observar en las Tablas siguientes

SpMV	Paralution AMDGPU	
Formato	Tiempo	Gflop/Sec
CSR	0,528166	1,75707
MCSR	0,51826	1,79065
ELL	0,236363	3,94393
COO	0,683236	1,35828
HYB	0,529866	1,75538

Tabla 2. Ejecución de la operación Matriz por vector, utilizando la biblioteca Paralution en una placa de tecnología AMD

SpMV	Paralution NVIDIAGPU	
Formato	Tiempo	Gflop/Sec
CSR	0,408654	2,27093
MCSR	0,411844	2,25334
ELL	0,278202	3,35080
COO	0,658819	1,40862
HYB	0,41667	2,23226

Tabla 3. Ejecución de la operación Matriz por vector, utilizando la biblioteca Paralution en una placa de tecnología NVIDIA

SpMV	VEXCL AMDGPU	VEXCL NVIDIAGPU
GFlops	2,12969	2,27719
Tiempo (s)	0,46702	0,43677

Tabla 4. Ejecución de la operación Matriz por vector, utilizando la biblioteca VEXCL en una placa de tecnología AMD y NVIDIA

SpMV	ViennaCL AMDGPU	
Formato	Tiempo	Gflop/Sec
COO	7,42336	0,12501
ELL	16,123	0,05756
HYB	16,0783	0,05718
Sliced ELL	7,74153	0,11988

Tabla 5. Ejecución de la operación Matriz por vector, utilizando la biblioteca ViennaCL en una placa de tecnología AMD

SpMV	ViennaCL NVIDIAGPU	
Formato	Tiempo	Gflop/Sec
COO	7,44757	0,12461
ELL	15,9269	0,05827
HYB	16,8232	0,05516
Sliced ELL	8,0137	0,11581

Tabla 6. Ejecución de la operación Matriz por vector, utilizando la biblioteca Vienna en una placa de tecnología NVIDIA

En las pruebas de las bibliotecas Paralution y Vexcl se puede observar que las ejecuciones sobre GPU de tecnología NVIDIA, ofrecen mejores resultados en tiempo de resolución del algoritmo y en eficiencia. Esto nos demuestra que a pesar que las bibliotecas son escritas con la misma semántica, la versión que implementa CUDA logra mejores resultados. Esto nos sugiere que el código, al compilarse, no sufre las mismas optimizaciones para ambos casos. Por lo que podemos concluir que existen una gran cantidad de optimizaciones a realizar manualmente, para lograr resultados similares en ambas tecnologías.

Una optimización a realizar, que se detectó utilizando contadores de hardware relacionados con accesos a memoria global (gld_efficiency, gst_efficiency, gld_requested_throughput, gst_requested_throughput [12]) es el evitar el reiterado acceso a posiciones globales de memoria. Esto es debido a que se consultan variables globales dentro de condiciones de fin de ciclo. Otro caso a mejorar es la escritura de variables globales, las cuales son utilizadas en cálculos intermedios.

Ambos casos se pueden solucionar utilizando variables locales, siempre que los recursos del hardware lo permitan.

Cabe destacar de la biblioteca ViennaCL, que hay casos de formato como COO, HYB y ELL que obtienen mejores tiempos en la GPU ATI vs la GPU NVIDIA. Sin embargo, no supera los valores de tiempo de las bibliotecas restantes. Una potencial causa de esta diferencia temporal, se detectó utilizando medidores de performance, que indican la cantidad de llamadas de cada función. Se observó que en la resolución del algoritmo, la biblioteca invierte demasiado tiempo en la inicialización de OpenCL y CUDA. Esta se realiza en cada llamada a la función SpMV de esta biblioteca, mientras que para las otras, se realiza al principio, fuera de la función de cómputo.

Formación de Recursos Humanos

Se está trabajando en conjunto con los alumnos de la Universidad Nacional de La Matanza para lograr que estos, además de los contenidos teóricos de la materia, puedan adquirir el conocimiento del diseño y construcción de funciones de optimización para el uso de GPU.

La presente línea de investigación forma parte del trabajo que el Ingeniero Federico Díaz se encuentra realizando para su tesis de Maestría y el Ingeniero. Waldo Valiente para su tesis de Maestría

Además se incorporó un becario - beca CIN- para realizar la iniciación en investigación. El que se encuentra cursando las ultimas materias de la carrera de grado.

Referencias

[1] OpenCL Spec, OpenCL 1.2 API and C Language Specification, 2012.

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

[3] G.L.M. Teodoro, R.S. Oliveira, D.O.G. Neto, R.A.C. Ferreira, “Profiling General Purpose GPU Applications”. 21st International Symposium on Computer Architecture and High Performance Computing. October, 2009. Sao Paulo, Brazil.

[4] F. G. Tinetti, S. M. Martin, F. E. Frati, M. Méndez. “Optimization and parallelization experiences using hardware performance counters”. International Supercomputing Conference Mexico. March, 2013. Colima, Mexico.

[5] Sergio Martin, Fernando Tinetti, Nicanor Casas, Graciela De Luca, Daniel Giulianelli. “N-Body Simulation Using GP-GPU: Evaluating Host/Device Memory Transference Overhead”. XIX Congreso Argentino de Ciencia de la Computación. Octubre, 2013. Buenos Aires, Argentina.

[6] Kelso, T. S. "Real-world benchmarking." Satellite Times 3.2 (1996): 80-82.

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

[8] B. Neelima and Prakash S. Raghavendra “Effective Sparse Matrix Representation for the GPU Architectures” (2012) 3-6

[9] http://www.paralution.com/

[10] https://github.com/ddemidov/vexcl

[11] http://viennacl.sourceforge.net/

[12] http://docs.nvidia.com/cuda/profiler-users-guide/#metrics-reference