Consumo Energético en Arquitecturas Multicore. Análisis sobre un Algoritmo de Criptografía Simétrica.

Fernando Romero[1], Adrian Pousa[1], Victoria Sanz[1 2], Armando De Giusti[1]

[1] Instituto de Investigacion en Informatica LIDI – Facultad de Informatica – UNLP
[2] Becaria CONICET

{fromero, apousa, vsanz, degiusti}@lidi.info.unlp.edu.ar

Abstract. En este trabajo se presenta un análisis de consumo energético del algoritmo de criptografía AES (Advanced Encryption Standard) sobre dos tipos de arquitecturas multicore: cluster de multicore y GPU. Para ello se utilizaron dos implementaciones del algoritmo: la primera hace uso de la librería de pasaje de mensajes MPI para ser ejecutada sobre un cluster de multicore y la segunda utiliza la herramienta de programación CUDA para ser ejecutada sobre una GPU. Se muestra a la GPU como una arquitectura que tiene una mejor relación FLOP/Watt que un cluster de multicore.

Keywords: Consumo energético, Cluster de multicore, GPU, AES.

1 Introducción

En general las arquitecturas paralelas (clusters, multicores, GPUs etc.) [1] [2] se han utilizado para resolver problemas en menor tiempo de ejecución respecto a los obtenidos por una máquina secuencial. Para tal fin, se han desarrollado herramientas, tales como MPI [3], OpenMP [4], CUDA [5], etc., que facilitan la programación de algoritmos paralelos [6] [7] para explotar las ventajas de estas arquitecturas.

En el último tiempo ha adquirido importancia el consumo energético [8] [9] debido a la gran cantidad de dispositivos electrónicos que se utilizan para cómputo y comunicación, lo que ha llevado a incrementar los Kw/h que las empresas de energía deben proveer. En el caso particular de los sistemas paralelos, para disminuir el tiempo de ejecución de un algoritmo se suele incrementar el número de procesadores hasta cierto límite, y esto implica un incremento en el consumo, que interesa sea lo menor posible en relación con la mejora de speed-up alcanzada.

Por lo anterior, para un algoritmo determinado no sólo es importante evaluar su rendimiento en cuanto a tiempo de ejecución sino también en función del consumo energético; en particular se pueden evaluar estos factores sobre distintas arquitecturas de manera de utilizar aquella que provea la mejor relación Flop/Watt [10] [11].

En un trabajo previo [12] se realizó un análisis de rendimiento en función del tiempo de ejecución del algoritmo criptográfico AES (Advanced Encryption Standard) [13]

sobre distintas arquitecturas multicore, mostrando la eficiencia de la GPU con respecto a una máquina multicore y a un cluster de multicore.

Este trabajo tiene como objetivo presentar un análisis de consumo energético entre una GPU y un cluster de multicore, utilizando como caso de estudio el mismo algoritmo criptográfico. Se muestra a la GPU como una arquitectura que no sólo tiene un alto rendimiento en cuanto a tiempo de ejecución sino también en función del consumo energético, comparada con el cluster de multicore.

2 Algoritmo AES. Implementaciones.

El análisis de consumo energético se realizó teniendo en cuenta la ejecución del algoritmo AES sobre dos arquitecturas paralelas: Cluster de multicore y GPU.

El algoritmo a ser ejecutado en el cluster de multicore se implementó utilizando MPI mientras que el algoritmo a ser ejecutado en la GPU se implementó con CUDA.

A continuación se detalla una breve descripción del algoritmo y las implementaciones paralelas realizadas.

2.1 Algoritmo AES

AES (Advanced Encryption Standard) es un algoritmo de cifrado simétrico por bloques establecido como estándar, siendo uno de los más utilizados en la actualidad.

Los datos a encriptar o desencriptar se dividen en bloques de tamaño fijo (128 bits), donde cada bloque se representa como una matriz de 4x4 bytes llamada *estado*. A cada *estado* se le aplican un conjunto de operaciones a partir de una clave inicial de 128 bits y diez subclaves generadas a partir de la primera.

Una forma de paralelizar este algoritmo es que cada hilo o proceso del programa trabaje sobre uno o varios *estados* en forma paralela.

2.2 Implementación paralela MPI sobre cluster de multicore

Un cluster es un conjunto de máquinas interconectadas a través de una red, que permite disminuir el tiempo de ejecución de las aplicaciones al incorporar la potencia de cómputo de máquinas individuales. La aparición de los multicore permitió agregar a los clusters mayor capacidad de cómputo al incrementar el número de procesadores de cada máquina conectada. Una de las herramientas más utilizadas para programar aplicaciones paralelas sobre un cluster es MPI. [14]

En la implementación de AES propuesta con MPI, se tiene la misma cantidad de procesos que procesadores, donde cada proceso trabaja en paralelo sobre un conjunto de *estados*.

Como se mostró en el trabajo previo, al incrementar la cantidad de procesadores se reduce el tiempo de ejecución. Por razones antes expuestas, es de interés cuantificar el incremento en consumo de energía a medida que la cantidad de procesadores aumenta.

2.3 Implementación paralela CUDA sobre GPU

En los últimos años las placas gráficas o GPU (Graphics Processing Units) se han estudiado debido a su alto rendimiento, por este motivo se empezaron a utilizar para otro propósito distinto al procesamiento gráfico a lo que se llamó GPGPU (General Purpose GPU) [15]. Una de las empresas fabricantes de estos dispositivos desarrolló un compilador y un conjunto de herramientas de desarrollo llamadas CUDA (Compute Unified Device Architecture) permitiendo a los programadores utilizar una variación del lenguaje C para programar de forma más sencilla las tarjetas gráficas y aprovechar la potencia de cómputo que son capaces de alcanzar.

Las GPU están compuestas por una gran cantidad de procesadores simples denominados Streaming Processors (SP), agrupados en lo que se denomina Streaming Multiprocessors (SMs), donde cada uno de los SM es capaz de ejecutar simultáneamente una gran cantidad de hilos.

En la implementación de AES propuesta con CUDA, se crea una cantidad determinada de hilos, tantos como *estados* se tengan, por lo tanto cada hilo trabajará sobre un solo *estado*. Esta distribución tiene relación con el diseño y funcionamiento de la GPU, que permite obtener mejor rendimiento cuando se tienen más hilos que SPs, esto se debe a que las latencias en el acceso a memoria suelen ser altas, por lo tanto se aprovechan los SPs ejecutando otro conjunto de hilos listos.

El trabajo anterior demostró el buen rendimiento de esta implementación ejecutada sobre una GPU en cuanto a tiempo de ejecución; es importante evaluar como se traduce ese rendimiento en consumo energético.

3 Metodología para la medición de consumo

La energía tanto del cluster como de la GPU fue medida en Joules. Un Joule es un Watt por segundo, donde el Watt es la unidad de potencia de un Ampere y una diferencia de potencial de un Volt y se expresa según (1) donde W simboliza la potencia medida en Watt, I simboliza la corriente medida en Ampere y V simboliza la diferencia de potencial medida en Volt.

$$W = I \cdot V \quad (1)$$

Un Joule o Watt por segundo equivale entonces a la energía que desarrolla una corriente de un Ampere durante un segundo, siendo la corriente impulsada por una diferencia de potencial de un Volt.

Las medidas fueron obtenidas indirectamente, es decir, no se midió potencia de forma directa sino que se midió en forma separada corriente y voltaje, y del producto de estos dos valores se obtuvo la potencia.

El voltaje se midió directamente de la línea eléctrica que alimenta el equipo a ser medido, enviando la señal eléctrica a un osciloscopio digital que permitió su posterior análisis.

La corriente se midió utilizando una pinza transductora en el cable de entrada de la fuente de energía de los equipos, enviando la señal de la pinza al osciloscopio para ser luego analizada. [16][17][18][19]

El osciloscopio digital captura las muestras de corriente y voltaje, y las agrupa en dos buffers de 10240 muestras (10KB), uno corresponde a los valores de la corriente y el otro corresponde a los valores del voltaje, ambos buffers se envían a un equipo que analiza los resultados. Un buffer de 10240 muestras representa aproximadamente 40 milisegundos, por lo tanto el intervalo de muestreo es de 40ms/10KB = 3,9µs.

La máquina que analiza los resultados va recibiendo, durante la ejecución de la aplicación, buffers de a pares corriente-voltaje y realiza el producto uno a uno entre los valores de ambos según (2), siendo B_I el buffer de corriente y B_V el buffer de voltaje, dando como resultado un nuevo buffer B_W con la misma cantidad de elementos, cuyos valores corresponden a la potencia de cada muestra y al igual que los dos buffers anteriores representa 40ms de muestras.

$$B_W[i] = B_I[i] \,.\, B_V[i] \quad \forall\, i \in Z, i \in [0,10240] \tag{2}$$

Para cada buffer de potencia B_W se calcula el valor eficaz f_{rms} según (3) empleando el método de los trapecios para el cálculo de la integral. El intervalo T2 – T1 = 40ms y f(t) representa cada valor en el buffer de valores de potencia B_W.

$$f_{rms} = \sqrt{\frac{1}{T_2 - T_1} \int_{T_1}^{T_2} f(t)^2 dt} \tag{3}$$

Por último se suman los valores eficaces f_{rms}, que se obtuvieron a partir del procesamiento de cada buffer B_W, dando como resultado un valor aproximado de la cantidad de Joules consumidos por la aplicación.

4 Resultados

El algoritmo secuencial fue ejecutado sobre una máquina con un procesador Intel i5 2300 [20] con 4 cores físicos (sin Hyperthreading) utilizando sólo uno de ellos, y 8GB de memoria RAM.

El algoritmo paralelo MPI fue ejecutado sobre un cluster de máquinas, cada una posee una arquitectura igual a la utilizada para el algoritmo secuencial, conectadas a través de una red Gigabit Ethernet.

El algoritmo CUDA fue ejecutado sobre una tarjeta gráfica Nvidia Geforce GTX 560TI [21] con 1GB de RAM que posee 384 SPs, distribuidos en 8 SMs (48 SPs cada uno).

Para hacer las mediciones se utilizó un osciloscopio digital con una resolución de 8 bits que posee dos entradas, una se utilizó para capturar voltaje, y la otra para capturar la corriente que provenía de una pinza transductora con una sensibilidad 1A/100mV, 1A/10mV y 1A/1mV.

La aplicación hace un cifrado y un descifrado de datos, ambos tienen un tiempo de ejecución similar. El descifrado se realiza a fin de verificar el buen funcionamiento del algoritmo. Los tiempos de ejecución presentados en este trabajo tienen en cuenta la ejecución de la aplicación completa a diferencia del trabajo anterior donde sólo se consideró el tiempo de cifrado, esto se debe a que las condiciones no estaban dadas para discriminar las partes de la aplicación al medir consumo.

La *Tabla 1* muestra los Joules totales consumidos por el algoritmo, ejecutado sobre cada arquitectura, para distintos tamaños de datos de entrada.

Tabla 1. Joules consumidos por el algoritmo.

MegaBytes	Secuencial	MPI 4 cores	MPI 8 cores	MPI 16 cores	MPI 32 cores	CUDA
16	1073,8454	417,0014	532,4198	743,0065	1000,9787	69,6309
32	2219,3307	801,7640	992,2515	1382,4791	2067,1985	111,0463
64	4375,8023	1634,6110	2113,6083	2979,2766	4232,7257	161,8118
128	8418,3768	3066,8609	4033,8559	5994,1951	8769,7185	294,5490
255	17685,4759	7050,1483	8513,3998	12159,6573	17305,2101	509,1929

Como se mencionó anteriormente cada buffer B_W representa aproximadamente 40ms, si se multiplica este valor por la cantidad de buffers obtenidos durante la ejecución de la aplicación, el resultado es un valor aproximado al tiempo en que tarda en ejecutarse la aplicación. La *Tabla 2* muestra este tiempo en segundos.

Tabla 2. Tiempos de muestra, en segundos, durante la ejecución de la aplicación.

MegaBytes	Secuencial	MPI 4 cores	MPI 8 cores	MPI 16 cores	MPI 32 cores	CUDA
16	12,92	3,6	2,56	1,76	1,24	0,44
32	27	6,76	4,44	3,16	2,52	0,68
64	53,52	13,84	9,32	6,64	5,16	0,96
128	103,32	25,84	17,52	13,04	10,48	1,68
255	215,84	57,32	37,08	26,52	20,64	2,92

El cociente entre los valores de Tabla 1 y Tabla 2 da la cantidad de Watts que consume cada arquitectura durante un segundo, esto se puede ver en *Tabla 3*.

Tabla 3. Watts por segundo de cada arquitectura.

MegaBytes	Secuencial	MPI 4 cores	MPI 8 cores	MPI 16 cores	MPI 32 cores	Cuda
16	83,1149	115,8337	207,9764	422,1627	807,2408	158,2520
32	82,1974	118,6041	223,4800	437,4933	820,3168	163,3033
64	81,7601	118,1077	226,7820	448,6862	820,2956	168,5539
128	81,4786	118,6865	230,2429	459,6775	836,8052	175,3267
255	81,9378	122,9963	229,5954	458,5089	838,4307	174,3811
Promedio	82,0978	118,8456	223,6153	445,3057	824,6178	167,9634

Como se puede observar en la tabla anterior, para una arquitectura en particular los Watts que consume por segundo son generalmente independientes del tamaño de datos de entrada, y tienen una variación poco significativa. La pequeña variación puede estar relacionada a la comunicación de datos por la red en el caso del cluster y a las copias de memoria (CPU-GPU; GPU-CPU) en el caso de la GPU, que no se ve reflejada en el caso secuencial y de 4 cores (1 máquina) por no haber movimiento de datos, por esta razón se utiliza un valor promedio de Watts por segundo.

La máquina que ejecuta el algoritmo secuencial tiene un menor consumo por segundo pero tiene un tiempo de ejecución muy alto. Cuando se ejecuta el algoritmo paralelo utilizando los 4 cores de la máquina, el tiempo de ejecución se reduce a un cuarto, pero se incrementa el consumo en aproximadamente 36 Watts por segundo (44%); duplicar la cantidad de cores (8 cores) implica agregar otra máquina, en este caso el tiempo se reduce a algo más de la mitad pero el consumo aumenta prácticamente al doble (89% aprox.); lo mismo ocurre si se incrementa a 16 cores (un total de 4 máquinas) y 32 cores (un total de 8 máquinas). Como se puede observar, incrementar la cantidad de cores sin agregar una máquina implica un aumento en el consumo, pero hacerlo agregando una máquina más al cluster causa un aumento mas significativo.

La GPU tiene un consumo de 167Ws que está entre una máquina de 4 cores (118Ws) y dos máquinas que suman 8 cores (223Ws), pero se ve claramente que por el buen rendimiento en tiempo de ejecución sobre el algoritmo hace que el consumo total sea mucho menor al del cluster.

En el trabajo previo, donde se hizo un análisis de rendimiento con respecto al tiempo de ejecución sobre el algoritmo AES, se mostró que para alcanzar los tiempos de una GPU se necesita un cluster con una gran cantidad de cores (alrededor de 256), el presente trabajo muestra que el incremento en la cantidad de cores genera un importante incremento de energía, más aún cuando esto implica agregar más máquinas al cluster (notar que la energía total consumida por el algoritmo con 32 cores es similar a la energía total consumida por el algoritmo secuencial).

Por lo tanto, alcanzar los tiempos logrados sobre la GPU con un cluster implica agregar una gran cantidad de cores a un costo energético muy alto, es por eso, que para este algoritmo particular, la GPU es una arquitectura que posee una mejor relación tiempo de ejecución/consumo energético que un cluster de multicore.

5 Conclusiones y trabajo a futuro

En este trabajo se presentó un análisis de consumo energético de dos arquitecturas multicore: GPU y cluster de multicore, utilizando como caso de estudio el algoritmo de criptografía AES.

Se mostró que, para este algoritmo en particular, la GPU tiene mejor rendimiento, no sólo en tiempo de ejecución sino también en lo relacionado al consumo de energía con respecto al cluster de multicore.

Las líneas de investigación futura apuntan a analizar otro tipo de algoritmos sobre estas arquitecturas, teniendo en cuenta aquellos que tienen el mismo rendimiento en cuanto a tiempo de ejecución para una misma entrada.

Se plantea analizar también el uso intensivo de memoria, CPU, comunicaciones y/o entrada-salida. Para esto último se deben ajustar las mediciones sobre cada aspecto a observar, ya sea CPU, memoria, disco, entre otros, dado que las mediciones realizadas se hicieron observando los equipos como un todo al medir en la entrada de la fuente de alimentación. Una posibilidad es medir consumo en la salida de la fuente de energía, donde se puede discriminar mejor cada aspecto pero se requieren herramientas con mayor precisión.

Además de lo anterior, se pretende analizar distintas arquitecturas multicore y cluster que incluyan una mayor cantidad de cores por máquina.

Por último, es de interés realizar un análisis de tiempo y consumo con más de una GPU por máquina [22] [23] como también sobre un cluster de máquinas que poseen GPUs [24].

6 Referencias

1. Chapman B., The Multicore Programming Challenge, Advanced Parallel Processing Technologies; 7th International Symposium, (7th APPT'07), Lecture Notes in Computer Science (LNCS), Vol. 4847, p. 3, Springer-Verlag (New York), November 2007.
2. Suresh Siddha, Venkatesh Pallipadi, Asit Mallick. "Process Scheduling Challenges in the Era of Multicore Processors"Intel Technology Journal, Vol. 11, Issue 04, November 2007.
3. MPI Specification http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf.
4. The OpenMP API specification for parallel programming. http://openmp.org/wp/.
5. Cuda Home Page http://www.nvidia.com/object/cuda_home_new.html.
6. Grama A., Gupta A., Karypis G., Kumar V. "Introduction to Parallel Computing". Second Edition. Addison Wesley, 2003.
7. Bischof C., Bucker M., Gibbon P., Joubert G., Lippert T., Mohr B., Peters F. (eds.), Parallel Computing: Architectures, Algorithms and Applications, Advances in Parallel Computing, Vol. 15, IOS Press, February 2008.
8. Cuda Home Page http://www.green500.org/.
9. Wu-chun Feng (2005). "The Importance of Being Low Power in High Performance Computing". CT Watch Quarterly.
10. Green Supercomputing Comes of Age - Wu-chun Feng, Xizhou Feng & Rong Ge http://portal.acm.org/citation.cfm?id=1344283.
11. W. Feng and K. Cameron. The Green500 List: Encouraging Sustainable Supercomputing. Computer, 40(12):50–55, 2007.
12. Adrián Pousa (UNLP), Victoria Sanz, (UNLP), Armando E. De Giusti,(UNLP) "Análisis de rendimiento de un algoritmo de criptografía simétrica sobre arquitecturas multicore" XVII Congreso Argentino de Ciencias de la Computación, Octubre 2011.
13. FIPS PUB 197: the official AES Standard http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
14. T. Rauber, G. Rünger. Parallel Programming: For Multicore and Cluster Systems. ISBN 364204817X, 9783642048173. Springer, 2010.
15. General-Purpose Computation on Graphics Hardware http://gpgpu.org/.
16. Frank, Ernest: "Análisis de Medidas Eléctricas", Mc Graw Hill, 1969.
17. Gray, P. E., Searle, C. L., "Principios de Electrónica", Reverté, 1973.
18. Cooper, William D. y Helfrick, Albert D.: "Instrumentación Electrónica Moderna y Técnicas de Medición", Prentice Hall Iberoamericana, 1991.
19. Wolf, S., Smith, R. F. M., "Guía para mediciones electrónicas y prácticas de laboratorio", Prentice-Hall, 1992.
20. Intel Product Specifications http://ark.intel.com/products/52206/Intel-Core-i5-2300-Processor-(6M-Cache-up-to-3_10-GHz).
21. Nvidia Geforce GTX 560TI Specifications http://www.nvidia.com/object/product-geforce-gtx-560ti-us.html.
22. D. Schaa and D. Kaeli. Exploring the Multi GPU Design Space. In International Symposium on Parallel and Distributed Processing, October 2009.
23. A. Moerschell and J. Owens. Distributed Texture Memory in a Multi-GPU Environment. In Graphics Hardware 2006,2006.
24. Z. Fan, F. Qiu, and A. Kaufman. ZippyGPU: Programming Toolkit for General-Purpose Computation on GPU Clusters. In GPGPU Workshop at Supercomputing, 2006.