Análisis de rendimiento de un algoritmo de criptografía simétrica sobre arquitecturas multicore

Adrian Pousa[1], Victoria Sanz[1], Armando De Giusti[1]

[1] Instituto de Investigacion en Informatica LIDI – Facultad de Informatica – Universidad Nacional de La Plata, Argentina
{apousa, vsanz, degiusti}@lidi.info.unlp.edu.ar

Abstract. En este trabajo se presenta un análisis del rendimiento del algoritmo de cifrado simétrico AES (Advanced Encryption Standard) sobre distintas arquitecturas multicore. Para ello se realizaron tres implementaciones, basadas en el lenguaje C, que utilizan herramientas de programación paralela OpenMP, MPI y CUDA, para ser ejecutadas sobre procesadores multicore, cluster de multicore y GPU respectivamente. Se muestra la eficiencia obtenida por la implementación CUDA del algoritmo a medida que aumenta el tamaño de los datos de entrada.

Keywords: arquitecturas multicore, programación paralela, AES, OpenMP, MPI, CUDA, GPGPU.

1 Introducción

El surgimiento de las arquitecturas multicore [1] [2] impulsa el uso de herramientas de programación paralela [3] [4], tales como OpenMP [5] y MPI [6], de modo de aprovechar la potencia que estas arquitecturas proveen.

En los últimos años, las GPU (Graphics Processing Units) [7] han ganado importancia debido al alto rendimiento alcanzado en aplicaciones de propósito general.

Por otro lado, el volumen de datos que se trasmiten en las redes se ha incrementado considerablemente, y en ocasiones suelen ser información sensible, por lo tanto es importante codificarlos para enviarlos por una red pública como lo es InterNet de manera segura. El encriptado y desencriptado de datos requiere un tiempo de cómputo adicional, que dependiendo de su tamaño puede ser considerable.

AES (Advanced Encryption Standard), es un algoritmo de cifrado simétrico por bloques que se ha convertido en estándar en 2002 [8], y actualmente es el más ampliamente usado para codificar información. En 2003, el gobierno de los Estados Unidos anuncio que el algoritmo era lo suficientemente seguro y que podía ser usado para protección nacional de información [9]. Hasta el momento no se conocen ataques

eficientes, los únicos conocidos son los denominados ataques de canal auxiliar[1][10] [11] [12].

Este algoritmo se caracteriza por ser simple, rápido y por consumir pocos recursos. Sin embargo el tiempo de cifrar y descifrar grandes cantidades de datos es importante por lo que es oportuno aprovechar las posibilidades que brindan las arquitecturas multicores para reducir este tiempo.

El propósito de este trabajo es mostrar la aceleración del cómputo de encriptar información con el algoritmo AES aprovechando distintas arquitecturas multicore:

- Procesadores multicore compartiendo memoria [13]. Para este caso se realizó una implementación del algoritmo que utiliza OpenMP.
- Cluster de multicore [13]. La herramienta que se utilizó en este caso es MPI.
- GPU. El algoritmo fue implementado utilizando CUDA [14] [15].

A continuación se presenta un análisis del rendimiento que muestra la eficiencia de la implementación realizada para GPU.

2 Descripción del algoritmo AES

AES (Advanced Encryption Standard) es un algoritmo de cifrado simétrico que se transformó en estándar en el año 2002, convirtiéndose en uno de los algoritmos más utilizados en la actualidad.

Se caracteriza por ser un algoritmo de cifrado por bloques. Los datos a encriptar se dividen en bloques de tamaño fijo (128 bits), donde cada bloque se representa como una matriz de 4x4 bytes llamada *estado* como se muestra en la figura 1.

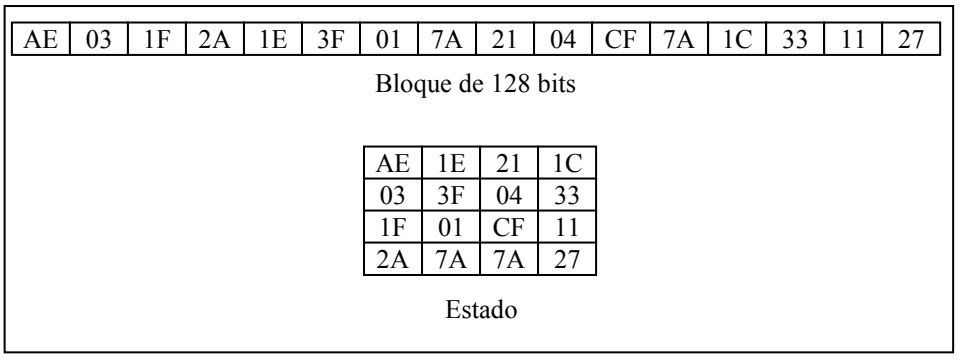

Fig. 1. Estado AES.

[1] Un ataque de canal auxiliar no ataca al algoritmo de cifrado sino que aprovecha vulnerabilidades de las implementaciones que pueden revelar datos a medida que se realiza el cifrado.

A cada *estado* se le aplican once rondas, cada una está compuesta por un conjunto de operaciones. Las once rondas se pueden clasificar en tres tipos: una ronda inicial, nueve rondas estándar y una ronda final, como detalla la figura 2.

Por ser AES un algoritmo simétrico, utiliza la misma clave para cifrar y descifrar los datos, cuyo tamaño es de 128 bits según lo indica el estándar. A esta clave se la denomina *clave inicial*, y a partir de ella se generan diez claves más mediante un procedimiento matemático. Las diez claves resultantes junto con la *clave inicial* son denominadas *subclaves* y cada una es utilizada en una de las rondas.

La ronda inicial realiza una sola operación:

AddRoundKey: se hace un XOR byte a byte entre el *estado* y la clave inicial.

En cada una de las siguientes nueve rondas, denominadas *estándar*, se aplican 4 operaciones en este orden:

SubBytes: se reemplaza cada byte del estado por otro de acuerdo a una tabla de sustitución de bytes con valores predeterminados. Este valor resultante se obtiene accediendo a la tabla tomando como índice de fila los primeros 4 bits del byte a reemplazar y como índice de columna los últimos 4 bits. El tamaño de la tabla es de 16x16 bytes.

ShiftRows: a excepción de la primera fila del estado, que no se modifica, los bytes de las filas restantes se rotan cíclicamente a izquierda: una vez en la segunda fila, dos veces en la tercera y tres veces en la cuarta.

MixColumns: a cada columna del estado se le aplica una transformación lineal y es reemplazada por el resultado de esta operación.

AddRoundKey: es igual a la ronda inicial pero utilizando la siguiente subclave.

La ronda final consiste de 3 operaciones:

SubBytes: de la misma forma que se aplica a las rondas estándar.

ShiftRows: de la misma forma que se aplica a las rondas estándar.

AddRoundKey: al igual que las rondas anteriores pero utilizando la última subclave.

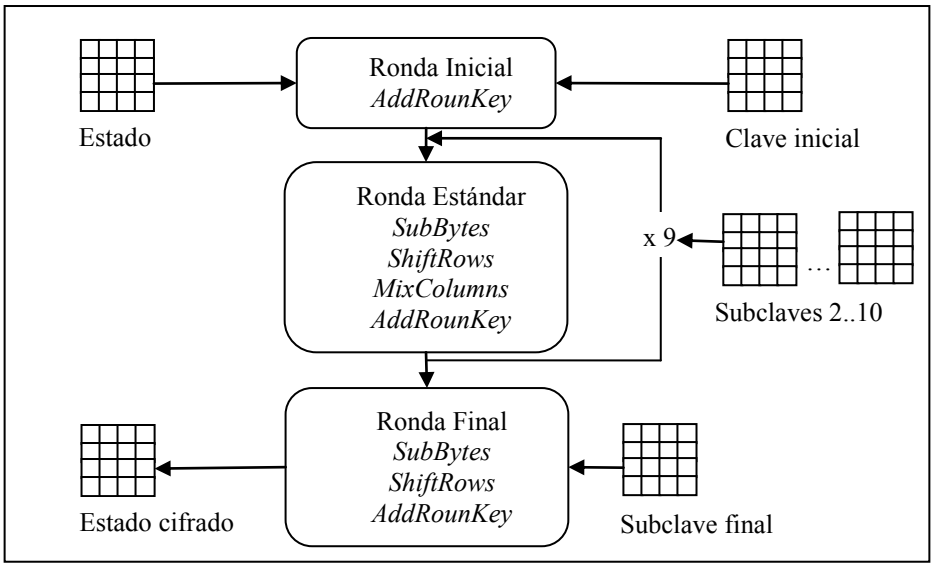

Fig. 2. Rondas del algoritmo AES sobre un estado.

3 Implementaciones del algoritmo AES

Se realizaron cuatro implementaciones de este algoritmo, una implementación secuencial y las restantes utilizando distintas herramientas de programación paralela tales como OpenMP, MPI y CUDA.

3.1 Implementación secuencial

La implementación secuencial del algoritmo genera las subclaves a partir de la clave inicial. A continuación, para cada *estado* de 16 bytes de los datos a cifrar aplica las rondas utilizando en las subclaves generadas inicialmente.

3.2 Implementaciones paralelas

Las implementaciones paralelas consideran a los datos de entrada como bloques consecutivos de 16 bytes. Además se tiene una cantidad determinada de procesos o hilos, y cada uno se encargará de cifrar un conjunto de bloques. La distribución de los bloques es proporcional a la cantidad de procesos o hilos, es decir si el tamaño de los datos de entrada es de N bytes, la cantidad de bloques es B = N/16. Si se tienen P procesos o hilos, cada uno deberá cifrar B/P bloques.

La generación de las subclaves se realiza secuencialmente en todos los casos por ser un proceso muy simple y el tiempo de ejecución de este cálculo es despreciable.

Una vez generadas las subclaves todos los procesos o hilos las utilizan para el proceso de cifrado de los bloques.

3.2.1 Implementación usando OpenMP

OpenMP es una API para los lenguajes C, C++ y Fortran que permite escribir y ejecutar programas paralelos utilizando memoria compartida, da la posibilidad de crear un conjunto de hilos que trabajan concurrentemente y así aprovechar las ventajas de las arquitecturas multicores.

La implementación de AES propuesta con OpenMP genera en forma secuencial las subclaves a partir de la clave inicial. Luego se crean un conjunto de hilos, tantos como cores provea la arquitectura, cada uno de los hilos tomará un conjunto consecutivo de bloques de 16 bytes y le aplicará el proceso de encriptación.

3.2.2 Implementación usando MPI

MPI (Message Passing Interface) es una especificación de una API para programación con memoria distribuida, con implementaciones para los lenguajes C, C++ y Fortran. Permite ser usada tanto en máquinas multicore, en arquitecturas tipo cluster donde se tienen varias máquinas conectadas por medio de una red, así como también en una combinación de ambas de forma de aprovechar todos los cores que estas arquitecturas proveen.

La implementación de AES propuesta con MPI, parte de tener una cantidad determinada de procesos, tantos como procesadores se tengan. Uno de ellos genera secuencialmente las subclaves a partir de la clave inicial y las comunica. Luego distribuye proporcionalmente los bloques de 16 bytes entre los procesos, incluyéndose a sí mismo. Cada proceso encriptará los bloques que le correspondan y retornará los bloques cifrados.

3.2.3 Implementación usando CUDA

En los últimos años las placas graficas o GPU (Graphics Processing Units) fueron motivo de estudio debido a su alto rendimiento, por este motivo se empezaron a utilizar para otro tipo de propósito distinto al procesamiento grafico a lo que se llamó GPGPU (General Purpose GPU). A partir de esto, una de las empresas fabricantes de placas gráficas 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 placas gráficas y aprovechar la potencia de cómputo que estas tienen.

Las GPU están compuestas por un conjunto de Streaming Multiprocessors (SMs), cada uno posee cores simples, denominados Streaming processors (SP). Cada SM es capaz de ejecutar simultáneamente una gran cantidad de hilos (el límite depende de la arquitectura), lo cual permite que los SP estén siempre realizando trabajo útil, aun cuando parte de dichos hilos están esperando por accesos a memoria.

El sistema de cómputo para un programador CUDA está compuesto por la CPU, también llamada *host,* y una o más GPUs llamadas *devices*. Un programa CUDA se divide en fases a ejecutar en el *host* y fases a ejecutar en el *device*. El código a ejecutar en el *device* recibe el nombre de *kernel,* y la invocación al mismo por parte del *host* dará lugar a la creación de los hilos que se agrupan en un *grid,* el cual se divide en *bloques* de hilos. Los hilos que componen un bloque serán asignados a un SM para su ejecución.

Las GPU poseen distintos tipos de memoria: la memoria global, accesible por el host y todos los hilos para su lectura y escritura; la memoria de constantes que permite lectura y escritura por parte del host y solo lectura por los hilos; una memoria compartida ubicada en cada chip SM, de acceso más rápido que la memoria global y mediante la cual los hilos de un mismo *bloque* pueden cooperar; registros internos a cada SM, así como también otras memorias de constantes de texturas. [16].

3.2.3.1 Algoritmo AES en GPU

El cálculo de las subclaves del algoritmo AES se realiza en el host, ya que el tiempo de ejecución es despreciable, dejando al *device* solo el procedimiento de cifrado.

El host copia en la memoria de constantes del *device* las subclaves y la tabla de sustitución de bytes, dado que ambas solo serán leídas por los hilos. Luego copia los datos a cifrar en la memoria global del *device*.

A continuación, invoca al kernel especificando tanto la cantidad de bloques, como la cantidad de hilos por bloque.

Los hilos pertenecientes a un mismo bloque CUDA trabajarán sobre *estados* consecutivos, cada uno se encargará de cifrar un *estado* (bloque de 16 bytes). Dado que el acceso a memoria global es costoso, previo a la etapa de cifrado del estado, cada hilo cooperará con los hilos de su mismo bloque para cargar a memoria *shared* la información que los mismos deben cifrar. Estos accesos se hacen de manera coalescente. Una vez terminada la etapa de cifrado, los hilos cooperan de la misma forma para trasladar los datos desde la memoria *shared* a la memoria *global.*

4 Resultados

El algoritmo secuencial fue ejecutado en una máquina con arquitectura Intel Xeon E5405 [17] con 2GB de memoria RAM. El algoritmo de memoria compartida que utiliza OpenMP fue ejecutado en una máquina con 2 procesadores Intel Xeon E5405 con 4 cores cada uno, con 2GB de memoria RAM; mientras que el algoritmo en MPI fue ejecutado utilizando un cluster de 4 maquinas con la arquitectura anteriormente mencionada conectados a 1Gbit Ethernet utilizando 32 cores.

El algoritmo CUDA fue ejecutado en una tarjeta gráfica Nvidia Geforce GTX 560TI [18] con 1GB de RAM que posee 384 SPs, distribuidos en 8 SMs, cada uno siendo capaz de ejecutar un máximo de 768 hilos; se utilizaron bloques de 256 hilos, por lo tanto el máximo número de bloques que podrá ejecutar un SM será 3 y la cantidad de bloques depende del tamaño de los datos a cifrar. CUDA permite crear

65535 bloques de hilos para un grid unidimensional, en el caso que la cantidad de bloques supere al máximo que puede ejecutar cada SM, los bloques de hilos se irán asignando a los SMs a medida que otros van terminando su ejecución.

Los tiempos de ejecución presentados corresponden solo al tiempo de cifrado, para distinto tamaño de datos de entrada. El tiempo de descifrado no se tuvo en cuenta por ser similar.

Tabla 1. La siguiente tabla muestra los tiempos de ejecución promedio en segundos de las distintas implementaciones para los distintos tamaños de datos de entrada.

	1KB	512KB	1MB	15MB	128MB	255MB
Secuencial (Intel)	0,002221	1,133039	2,266163	33,99241	290,034037	577,988805
OMP (8 cores)	0,00054	0,155369	0,29914	4,358485	37,128006	73,94553
MPI (8 cores)	0,00028	0,142951	0,286033	4,29618	36,646296	72,971179
MPI (16 cores)	0,000141	0,072217	0,143635	2,146643	18,313296	36,528222
MPI (32 cores)	0,000071	0,035668	0,071336	1,073768	9,162032	18,248819
CUDA	0,000146	0,002551	0,005033	0,067806	0,572375	1,139361

Tabla 2. La siguiente tabla muestra el speedup de las distintas implementaciones con respecto a la implementación secuencial ejecutada en la arquitectura Intel.

	1KB	512KB	1MB	15MB	128MB	255MB
OMP (8 cores)	4,112962	7,292568	7,575593	7,799134	7,811732	7,816413
MPI (8 cores)	7,932142	7,926065	7,922732	7,912240	7,914416	7,920782
MPI (16 cores)	15,75177	15,68936	15,77723	15,83514	15,83734	15,82307
MPI (32 cores)	31,281690	31,766261	31,767452	31,657127	31,656082	31,672669
CUDA	15,2123288	444,154841	450,260878	501,318615	506,720309	507,292074

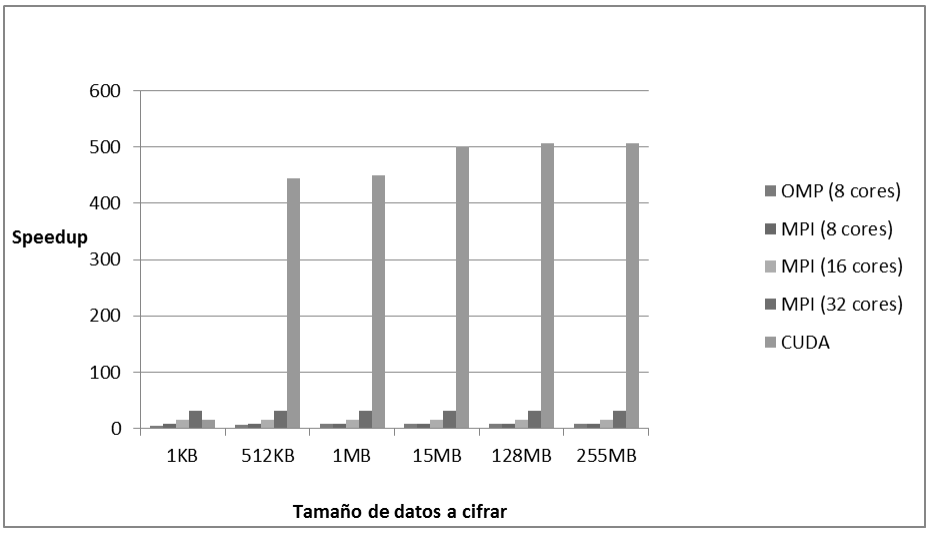

Fig. 3. Diferencias en el speedup de las distintas implementaciones con respecto a la implementación secuencial ejecutada en la arquitectura Intel.

Como se puede apreciar los tiempos del algoritmo que utiliza OpenMP y MPI (8 cores) son similares con excepción de la prueba con un tamaño de datos de 1KB donde MPI se comporta mejor. En la versión MPI del algoritmo se puede observar que los tiempos de ejecución escalan linealmente al aumentar la cantidad de cores y el tamaño de los datos de entrada. De todos modos no alcanza a mejorar al nivel de la implementación sobre la GPU utilizando CUDA, considerablemente más bajos con respecto a las demás implementaciones.

Para poder ejecutar en GPU es necesario copiar los datos a cifrar a la memoria del *device* y una vez que finaliza la ejecución del *kernel* se deben recuperar los datos cifrados. Estas copias memoria *host-device* y *device-host* suelen introducir un cierto overhead.

Tabla 3. La siguiente tabla muestra los tiempos promedio de transferencia de datos entre el *host* y el *device*, el tiempo de cifrado y el tiempo total resultante.

	1KB	512KB	1MB	15MB	128MB	255MB
Copia *host-device*	0,000007	0,000286	0,000996	0,022862	0,208334	0,404399
Copia *device-host*	0,000016	0,000965	0,001956	0,029013	0,247463	0,500004
Tiempo de Cifrado	0,000145	0,001819	0,003594	0,046689	0,392180	0,780433
Tiempo total	0,000168	0,00307	0,006546	0,098564	0,847977	1,684836

El tiempo total de ejecución, que tiene en cuenta el tiempo de cifrado y el tiempo de transferencia de datos, se sigue manteniendo por debajo de los tiempos de ejecución de los demás algoritmos a excepción de la ejecución con 1KB en MPI con 16 y 32 cores donde la diferencia es mínima.

Como se mencionó anteriormente, la implementación con MPI escala linealmente, es decir, duplicando la cantidad de cores el tiempo de ejecución se reduce a aproximadamente la mitad. Se puede ver que con un tamaño de datos de entrada de 255MB y 32 cores en la implementación MPI, el tiempo de ejecución está en el orden de los 18 segundos. Por lo tanto para lograr alcanzar el orden de los 1,68 segundos con la implementación MPI, al igual que la implementación CUDA, es necesario elevar la cantidad de cores a algo más de 256.

5 Conclusiones y trabajo a futuro

Se presentó una implementación secuencial y tres implementaciones paralelas del algoritmo de cifrado simétrico por bloques AES para aprovechar distintas arquitecturas multicore.

Se realizo un análisis de los tiempos de ejecución resultantes de las distintas implementaciones, observando la eficiencia de la implementación del algoritmo con CUDA.

Se comprobó que el cifrado de gran cantidad de datos con la implementación CUDA del algoritmo AES presenta tiempos muy bajos, por lo tanto es posible reducir el costo de cifrado en una transferencia de datos de manera considerable.

Existe en la actualidad una librería de criptografía de propósito general llamada OpenSSL [19], esta implementa distintos tipos de algoritmos criptográficos incluyendo AES de forma mucho más eficiente, como trabajo futuro se pretende utilizar esta librería para hacer el análisis de rendimiento del algoritmo AES y otros algoritmos de cifrado tanto simétricos como asimétricos.

Otras líneas futuras incluyen el estudio sistemático del rendimiento de algoritmos paralelos (especialmente numéricos [20] [21] [22]) ejecutados sobre arquitecturas basadas en GPUs, en comparación con clusters de multicores. Asimismo se plantea el análisis de la eficiencia energética [23] [24] [25] al escalar los problemas y el número y complejidad de las GPUs que se empleen.

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. Grama A., Gupta A., Karypis G., Kumar V. “Introduction to Parallel Computing”. Second Edition. Addison Wesley, 2003.
4. 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.
5. The OpenMP API specification for parallel programming. http://openmp.org/wp/.
6. MPI Specification http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf.
7. General-Purpose Computation on Graphics Hardware http://gpgpu.org/.

8. FIPS PUB 197: the official AES Standard http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
9. Lynn Hathaway (June 2003). "National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information"http://csrc.nist.gov/groups/ST/toolkit/documents/aes/CNSS15FS.pdf.
10. D.J. Bernstein - Cache-timing attacks on AES (2005) http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.
11. Dag Arne Osvik, Adi Shamir and Eran Tromer - Cache Attacks and Countermeasures: the Case of AES (2005) http://www.wisdom.weizmann.ac.il/~tromer/papers/cache.pdf.
12. A Diagonal Fault Attack on the Advanced Encryption Standard http://eprint.iacr.org/2009/581.pdf.
13. T. Rauber, G. Rünger. Parallel Programming: For Multicore and Cluster Systems. ISBN 364204817X, 9783642048173. Springer, 2010.
14. Buck I. “Gpu computing with nvidia cuda”. ACM SIGGRAPH 2007 courses ACM, 2007. New York, NY, USA.
15. Cuda Home Page http://www.nvidia.com/object/cuda_home_new.html.
16. Cuda best practices guide http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf.
17. Intel Product Specifications http://ark.intel.com/products/33079/Intel-Xeon-Processor-E5405-(12M-Cache-2_00-GHz-1333-MHz-FSB).
18. Nvidia Geforce GTX 560TI Specifications http://www.nvidia.com/object/product-geforce-gtx-560ti-us.html.
19. The official OpenSSL www.openssl.org.
20. Basic Linear Algebra Subprograms BLAS http://www.netlib.org/blas/.
21. Linear Algebra Package LAPACK http://www.netlib.org/lapack/.
22. Automatically Tuned Linear Algebra Software ATLAS http://www.netlib.org/atlas/.
23. Computer Architecture: Challenges and Opportunities For The Next Decade - Tilak Agerwala Siddhartha Chatterjee IBM Research 2004. Published by the IEEE Computer Society
24. Green Supercomputing Comes of Age - Wu-chun Feng, Xizhou Feng & Rong Ge http://portal.acm.org/citation.cfm?id=1344283.
25. Maximizing Power Efficiency with Asymmetric Multicore Systems - Alexandra Fedorova, Juan Carlos Saez, Daniel Shelepov, and Manuel Prieto http://portal.acm.org/citation.cfm?id=1610270.