Comparación de rendimiento de algoritmos de cómputo intensivo y de acceso intensivo a memoria sobre arquitecturas multicore. Aplicación al algoritmo de criptografía AES.

Adrian Pousa[1], Victoria María Sanz[1,2], Armando E. De Giusti[1,2]

[1] III-LIDI Facultad de Informática UNLP La Plata, Argentina
[2] CONICET, Argentina
{apousa,vsanz,degiusti}@lidi.info.unlp.edu.ar

Abstract. En este trabajo se presenta una comparación de rendimiento de dos versiones del algoritmo de criptografía AES. La primera versión, AES-CI (AES Cómputo Intensivo), se caracteriza por ser intensiva en cómputo. La segunda versión, AES-AIM (AES Acceso Intensivo a Memoria), reduce la cantidad de cálculo reemplazando operaciones por acceso a datos pre-calculados almacenados en memoria. Se llevaron a cabo tres implementaciones de cada versión del algoritmo, desarrolladas con diferentes herramientas paralelas (OpenMP, MPI, CUDA), para ser ejecutadas sobre una máquina multicore, un cluster de máquinas multicore y una GPU respectivamente. El trabajo experimental muestra que las implementaciones AES-AIM reducen el tiempo de ejecución con respecto a los obtenidos por las implementaciones AES-CI. Del estudio se observa que las características del algoritmo determinan la arquitectura a utilizar para obtener el mejor rendimiento.

Keywords: Algoritmos de Cómputo Intensivo; Algoritmos de Acceso Intensivo a Memoria; AES; arquitecturas multicore.

1 INTRODUCCIÓN

En general las arquitecturas paralelas (clusters, multicores, GPUs, etc.) [1] [2] se han utilizado para resolver diversos problemas en menor tiempo. 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] y permiten explotar las ventajas de estas arquitecturas.

En los últimos años, las GPUs (Graphics Processing Units) [8] han ganado importancia debido al alto rendimiento que alcanzan al ejecutar aplicaciones de propósito general. Sin embargo, estas arquitecturas presentan algunas limitaciones relacionadas con la memoria: (a) La capacidad de memoria global es de un tamaño fijo y no se puede incrementar, a diferencia de otras arquitecturas (b) Las GPUs poseen una jerarquía de memoria de varios niveles y con distintas características, dependiendo del nivel que se acceda se tendrá menor o mayor latencia. Una técnica

que permite ocultar la latencia consiste en ejecutar una mayor cantidad de hilos, de esta forma, mientras un conjunto de hilos espera que se resuelva un acceso a memoria, otro conjunto de hilos ejecuta en los procesadores. Sin embargo, esto implica tener menor cantidad de recursos por hilo (registros, fracción de memoria compartida que le corresponde a cada hilo).

Por otro lado, el volumen de datos que se trasmiten en las redes se ha incrementado considerablemente, y en ocasiones suelen representar 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 [9] y actualmente es ampliamente usado para codificar información. En 2003 el gobierno de los Estados Unidos anunció que el algoritmo era lo suficientemente seguro y que podía ser usado para protección nacional de información [10]. Hasta el momento no se conocen ataques eficientes, los únicos conocidos son los denominados ataques de canal auxiliar [11].

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

En trabajos anteriores [12] [13] se presentaron tres implementaciones de una versión del algoritmo AES que realiza cómputo intensivo. Éstas fueron desarrolladas con las herramientas de programación paralela OpenMP, MPI y CUDA, para ser ejecutadas sobre una máquina multicore, cluster de multicore y GPU respectivamente. El trabajo experimental mostró que la implementación de AES para GPU alcanza una mayor eficiencia tanto en tiempo de cómputo como en consumo energético, en comparación con las implementaciones del algoritmo AES para las arquitecturas restantes.

Asimismo, en [14] se analizó el rendimiento de AES sobre una máquina con una GPU y sobre un cluster de GPU, para casos en que la memoria requerida por el algoritmo supere la memoria de una GPU. Para esto se realizaron dos implementaciones: una utilizando CUDA que se ejecuta sobre una única GPU dividiendo en fragmentos los datos a cifrar e invocando varias veces al kernel AES, y otra que utiliza una combinación de MPI y CUDA para ser ejecutada sobre un cluster de GPU. El trabajo experimental mostró que las comunicaciones en el cluster de GPU tienen un impacto negativo en el tiempo total de cómputo del algoritmo. Por lo tanto, para este algoritmo y arquitectura disponible, es más adecuado dividir los datos en fragmentos y cifrar cada uno utilizando una única GPU, que utilizar un cluster de GPU.

En este trabajo se realiza una comparación de rendimiento de dos versiones de AES sobre arquitecturas multicore. La primera versión es aquella que se presentó en el trabajo [12], a la cual se denomina AES-CI (AES Cómputo Intensivo). La segunda versión, AES-AIM (AES Acceso Intensivo a Memoria), reduce la cantidad de cálculo reemplazando operaciones por accesos a datos pre-calculados almacenados en memoria. Esto es posible porque AES se basa en un álgebra cerrada, por lo tanto el

resultado de ciertas operaciones se puede mantener en tablas. Esta versión de AES está disponible en la librería OpenSSL[15].

Para cada versión se realizaron tres implementaciones que utilizan herramientas de programación paralela OpenMP, MPI y CUDA, para ser ejecutadas sobre una máquina multicore, cluster de máquinas multicore y GPU respectivamente.

El trabajo experimental muestra que las implementaciones de AES-AIM reducen en todos los casos el tiempo de ejecución respecto a AES-CI. Sin embargo, las implementaciones de AES-AIM secuencial, OpenMP y MPI alcanzan una mejora similar con respecto a AES-CI, mientras que AES-AIM CUDA exhibe un menor grado de mejora respecto a AES-CI CUDA, debido a las limitaciones de latencia de memoria propias de la arquitectura GPU.

2 DESCRIPCIÓN DEL ALGORITMO AES

AES (Advanced Encryption Standard) 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 Fig. 1.

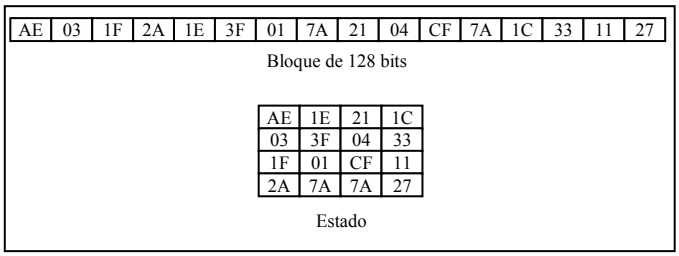

Fig. 1. Estado AES.

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.

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.

En la Fig. 2 se muestra un esquema del algoritmo.

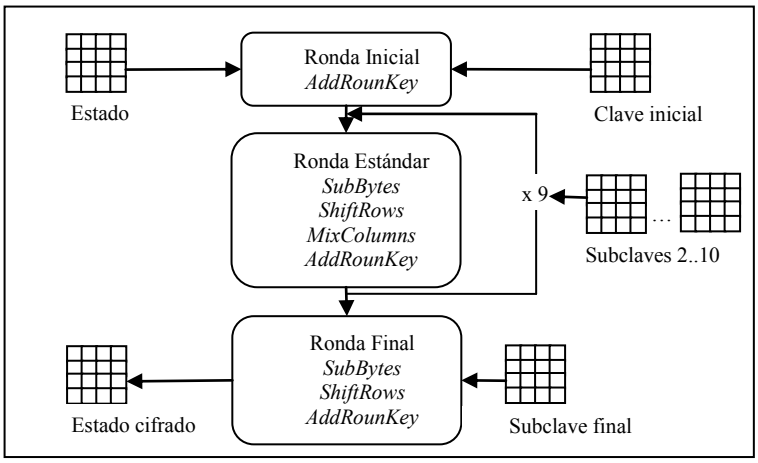

Fig. 2. Rondas del algoritmo AES sobre un estado.

Hay dos formas de implementar el algoritmo:

- La primera consiste en realizar intensivamente todos los cálculos de cada operación que componen los tres tipos de rondas. A esta implementación se le denominó *AES-CI* (AES Cómputo Intensivo).
- La segunda consiste en generar tablas que almacenan datos pre-calculados, que no dependen de la entrada y que se van a utilizar en algunas operaciones. De esta forma, se reemplaza cálculo por acceso a memoria. A esta implementación se le denominó *AES-AIM* (AES Acceso Intensivo a Memoria).

3 MODELO DE PROGRAMACIÓN CUDA

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.

Cuando se invoca a un kernel se debe especificar la cantidad de hilos a utilizar en la ejecución y su organización. CUDA permite organizar los hilos en Grids. Un Grid es un conjunto de bloques de hilos.

Una GPU posee un conjunto de procesadores, llamados Streamming Multiprocessors (SM), y una jerarquía de memoria compuesta por: la memoria global, la memoria de constantes y una memoria compartida ubicada en cada SM. La memoria global es la que posee mayor latencia en la jerarquía, mientras que la memoria compartida es la que posee menor latencia. La memoria de constantes tiene una latencia intermedia.

Dado que el acceso a memoria global es costoso, los hilos suelen trabajar en memoria compartida. Para esto los hilos deben traer todos los datos que necesiten desde la memoria global a la memoria compartida, esta transferencia de datos implica una gran una cantidad de accesos a memoria global, pero CUDA permite utilizar la coalescencia como técnica de optimización para minimizar la cantidad de accesos. Esta técnica permite mover varios datos desde memoria global a memoria compartida (y viceversa) en un único acceso.

4 IMPLEMENTACIONES DE AES-CI

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

4.1 Implementación secuencial (AES-CI-SEC)

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 las subclaves generadas inicialmente.

4.2 Implementaciones paralelas

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

La generación de las subclaves se realiza secuencialmente en todos los casos por ser una tarea muy simple y el tiempo de ejecución de este cálculo es despreciable. Una vez generadas las subclaves, los procesos o hilos las utilizan para el proceso de cifrado de sus estados.

4.2.1 Implementación usando OpenMP (AES-CI-OMP)

La implementación de AES-CI 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 estados y le aplicará el proceso de encriptación.

4.2.2 Implementación usando MPI (AES-CI-MPI)

La implementación de AES-CI 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 estados entre los procesos, incluyéndose a sí mismo. Cada proceso encriptará los estados que le correspondan.

4.2.3 Implementación usando CUDA (AES-CI-CUDA)

El cálculo de las subclaves se realiza en el host, ya que el tiempo de ejecución es despreciable, dejando al device sólo 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 la organización del Grid, por lo tanto se debe indicar la cantidad de bloques de hilos y la cantidad de hilos por cada bloque.

Los hilos pertenecientes a un mismo bloque CUDA trabajarán sobre estados consecutivos, cada uno se encargará de cifrar un estado. 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 en memoria compartida la información que deben cifrar. Estos accesos se hacen de manera coalescente. Una vez terminada la etapa de cifrado, los hilos cooperan para trasladar los datos desde la memoria compartida a la memoria global de manera coalescente.

5 IMPLEMENTACIONES DE AES-AIM

Se realizaron cuatro implementaciones de AES-AIM, una implementación secuencial (AES-AIM-SEC) y tres implementaciones paralelas denominadas AES-AIM-OMP, AES-AIM-MPI y AES-AIM-CUDA, desarrolladas con OpenMP, MPI y CUDA respectivamente.

Las implementaciones de AES-AIM realizan invocaciones a la librería OpenSSL para realizar el cifrado o descifrado de datos. En particular, las implementaciones paralelas de AES-CI y AES-AIM son similares en la forma en que distribuyen los datos entre los procesos o hilos.

La librería OpenSSL no está pensada para utilizarse en algoritmos paralelos por lo que fue necesario hacer una adaptación para ser usada con OpenMP, MPI y CUDA. En el caso de OpenMP y MPI la librería no requiere de grandes modificaciones. Sin embargo, en el caso de CUDA es necesario indicar que las variables, constantes y funciones deben conocerse dentro de la GPU para que la ejecución del algoritmo de cifrado se lleve a cabo en el device.

6 TRABAJO EXPERIMENTAL

El algoritmo secuencial fue ejecutado en una máquina con arquitectura Intel Xeon E5405 [16] y 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, y 2GB de memoria RAM. El algoritmo MPI fue ejecutado utilizando un cluster de 4 máquinas con la arquitectura anteriormente mencionada, conectadas por una red 1Gbit Ethernet y utilizando 8, 16 y 32 cores.

El algoritmo CUDA fue ejecutado en una tarjeta gráfica Nvidia Geforce GTX 560TI [17] con 1GB de RAM que posee 384 Scalar Processors, distribuidos en 8 Streaming Multiprocessors.

Generalmente, los algoritmos CUDA pueden alcanzar distinto rendimiento según la cantidad de hilos por bloques especificada. Tanto para el algoritmo AES-CI-CUDA como AES-AIM-CUDA el mejor rendimiento se obtiene con 256 hilos por bloque.

Para realizar el análisis de rendimiento se cifraron datos de tamaño 512KB, 1MB, 15MB, 128 MB y 255MB.

En este trabajo, sólo se tiene en cuenta el proceso de cifrado para los distintos tamaños de datos de entrada. El proceso de descifrado no se tuvo en cuenta por tener un rendimiento similar.

En el trabajo previo [12] se mostró que la implementación de AES-CI para GPU alcanza mayor rendimiento, en comparación con las implementaciones del algoritmo AES-CI para las arquitecturas restantes. En la Fig. 3 se puede ver el alto speedup alcanzado por AES-CI-CUDA con respecto a las otras implementaciones paralelas de AES-CI, para archivos a cifrar de distinto tamaño. El Speedup se calculó en relación al tiempo de AES-CI-SEC sobre una máquina del cluster.

Por lo antes mencionado, si se tiene un algoritmo con las características de AES-CI, donde el cómputo es intensivo, será conveniente utilizar una GPU dado que garantiza un alto rendimiento.

Por otro lado, AES-AIM realiza menor cantidad de cálculo y permite obtener mejor rendimiento que AES-CI. La Fig. 4 muestra el porcentaje de reducción en el tiempo de ejecución del algoritmo secuencial AES-AIM-SEC con respecto al algoritmo secuencial AES-CI-SEC. Como se puede observar, en general AES-AIM-SEC reduce en aproximadamente un 99% el tiempo de AES-CI-SEC.

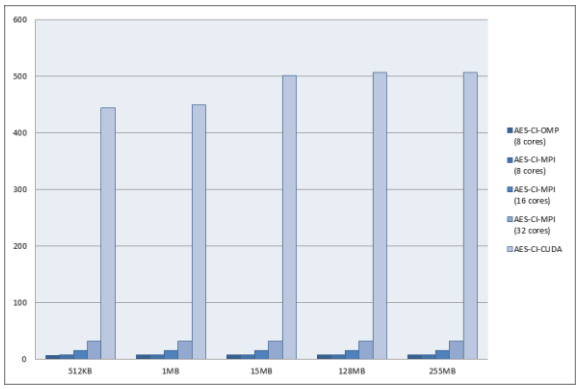

Fig. 3. Speedup de las implementaciones paralelas de AES-CI.

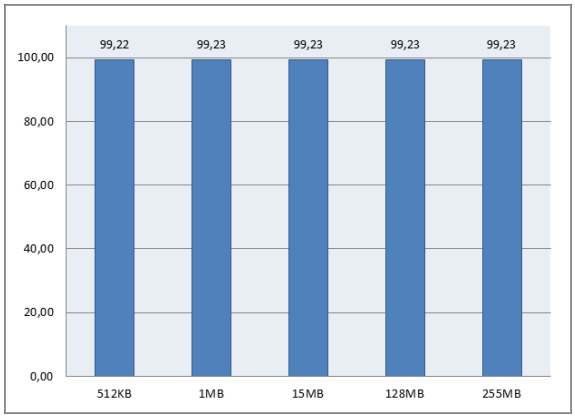

Fig. 4. Porcentaje de reducción de AES-AIM-SEC respecto a AES-CI-SEC

La Fig. 5 muestra el porcentaje de reducción en el tiempo de ejecución de las implementaciones paralelas AES-AIM con respecto las implementaciones paralelas respectivas de AES-CI. En particular, AES-AIM-OMP y AES-AIM-MPI alcanzan una reducción similar con respecto a AES-CI-OMP y AES-CI-MPI respectivamente, siendo la misma aproximadamente de 99%.

En contraste con lo anterior, la reducción obtenida por AES-AIM-CUDA con respecto a AES-CI-CUDA varía entre 56% y 62%.

La Fig. 6 muestra el speedup de las implementaciones paralelas de AES-AIM. El Speedup se calculó en relación al tiempo de AES-AIM-SEC sobre una máquina del cluster. De los resultados se observa que el rendimiento obtenido por AES-AIM-CUDA es menor al obtenido por AES-AIM-MPI cuando utiliza más de una máquina del cluster. El menor rendimiento de la GPU se debe a la manera en la cual estos dispositivos gestionan los hilos para intentar ocultar la latencia de memoria. Cuando un conjunto de hilos hace operaciones de memoria (load o store), la GPU coloca otro conjunto de hilos a procesar mientras se resuelven los accesos a memoria de los primeros. Como AES-AIM prácticamente no realiza procesamiento, existe un punto en el cual la latencia no puede ser ocultada.

Por lo tanto, si se tiene un algoritmo con las características de AES-AIM, el mejor rendimiento se obtiene al utilizar un cluster de máquinas multicore.

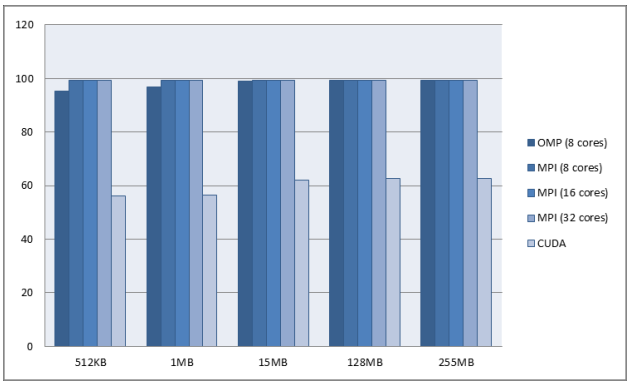

Fig. 5. Porcentaje de reducción de las implementaciones paralelas de AES-AIM vs. AES-CI

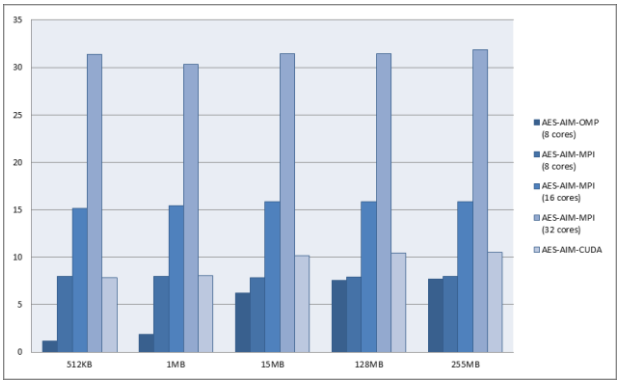

Fig. 6. Speedup de las implementaciones paralelas de AES-AIM.

7 CONCLUSIONES

Este trabajo presentó una comparación del rendimiento obtenido por dos versiones del algoritmo AES sobre distintas arquitecturas multicore (máquina multicore, cluster de máquinas multicore y GPU). Una versión está caracterizada por realizar cómputo intensivo (AES-CI) y otra por realizar acceso intensivo a memoria (AES-AIM), la cual reemplaza cálculo por acceso a datos pre-calculados. Para cada versión del algoritmo se presentaron tres implementaciones paralelas utilizando OpenMP, MPI y CUDA.

El trabajo experimental mostró que las implementaciones de AES-AIM reducen en todos los casos el tiempo de ejecución respecto a AES-CI. Sin embargo, las implementaciones de AES-AIM secuencial, OpenMP y MPI reducen el tiempo de ejecución en un 99% con respecto a AES-CI, mientras que AES-AIM CUDA exhibe

un menor grado de reducción respecto a AES-CI CUDA, debido a las limitaciones de latencia de memoria propias de la arquitectura GPU.

De los resultados experimentales se concluye lo siguiente: si se tiene un algoritmo con las características de AES-AIM, el cual realiza acceso intensivo a memoria, no será conveniente utilizar una GPU frente a un cluster, si estos poseen características similares a las utilizadas en este estudio. Esto contrasta con lo observado para AES-CI, algoritmo que realiza cómputo intensivo, donde se observó la gran eficiencia alcanzada por la versión para GPU respecto a las demás implementaciones paralelas para multicore y cluster de multicore.

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. General-Purpose Computation on Graphics Hardware http://gpgpu.org/.
9. FIPS PUB 197: the official AES Standard http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
10. 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.
11. D.J.Bernstein-Cache-timing attacks on AES (2005) http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.
12. Pousa A., Sanz V., De Giusti A. Performance Analysis of a Symmetric Cryptographic Algorithm on Multicore Architectures. Computer Science & Technology Series - XVII Argentine Congress of Computer Science - Selected Papers. Edulp 2012.
13. Romero F., Pousa A., Sanz V., De Giusti A. Consumo Energético en Arquitecturas Multicore. Análisis sobre un Algoritmo de Criptografía Simétrica. Proceedings del XVIII Congreso Argentino de Ciencias de la Computación. ISBN 978-987-1648-34-4
14. Pousa, A, Sanz, V., De Giusti, A. Performance Analysis of a Symmetric Cryptography Algorithm on GPU and GPU Cluster. VI Latin American Symposium on High Performance Computing HPCLatAm 2013. http://hpc2013.hpclatam.org/papers/HPCLatAm2013-paper-12.pdf
15. OpenSSL. The OpenSSL Project. Cryptography and SSL/TLS Toolkit. https://www.openssl.org/
16. Intel Product Specifications http://ark.intel.com/products/33079/Intel-Xeon-Processor-E5405-(12M-Cache-2_00-GHz-1333-MHz-FSB).
17. Nvidia Geforce GTX 560TI Specifications http://www.nvidia.com/object/product-geforce-gtx-560ti-us.html.