

Rendimiento del algoritmo AES sobre arquitecturas de memoria compartida

Adrián Pousa¹, Victoria Sanz^{1,2}, Marcelo Naiouf¹, and Armando De Giusti^{1,3}

¹ III-LIDI, Facultad de Informática, Universidad Nacional de La Plata, Argentina

² CIC, Buenos Aires, Argentina

³ CONICET, Argentina

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

Resumen Actualmente AES (Advanced Encryption Standard) es uno de los algoritmos de cifrado simétrico más utilizados para encriptar información. El volumen de datos sensibles que se transmiten en las redes se incrementa constantemente y cifrarlos puede requerir un tiempo significativo. Por lo anterior, es importante adaptar este algoritmo para aprovechar la potencia de cómputo de las arquitecturas paralelas emergentes. En este trabajo presentamos un análisis del rendimiento de AES sobre diversas arquitecturas de memoria compartida (multicore Intel E5-2695v4, Xeon Phi 7230 y GPU Nvidia GTX 960), para datos de entrada de distinto tamaño. Los resultados revelan que la GPU es la mejor alternativa para cifrar datos de entrada que no superan los 32MB. Sin embargo, para un volumen mayor de datos, el multicore alcanza el mejor rendimiento, seguido por el Xeon Phi.

Keywords: AES, cifrado de información, multicore, GPU, Xeon PHI

1. Introducción

En la actualidad, el volumen de datos que se transmite por redes públicas se incrementa constantemente. En ocasiones esta información suele ser sensible y por esto es importante cifrarla. AES (Advanced Encryption Standard) [1] es uno de los algoritmos de cifrado simétrico por bloques más utilizados para encriptar información. AES es considerado como un algoritmo seguro por el gobierno de los Estados Unidos para protección nacional de información [2].

El proceso de cifrado y descifrado requiere de un tiempo de cómputo que dependiendo del volumen de datos puede ser significativo. Diversas investigaciones se han enfocado en reducir este tiempo, adaptando distintos algoritmos de cifrado para aprovechar la potencia de cómputo de arquitecturas paralelas emergentes.

Por un lado, varios autores [3,4,5] presentaron distintas estrategias para paralelizar AES sobre GPUs Nvidia, obteniendo buen rendimiento respecto a la versión secuencial provista por OpenSSL [6].

Por otro lado, otros autores paralelizaron AES sobre GPUs Nvidia y sobre CPUs multicore y compararon el rendimiento de ambas versiones. A continuación resumimos los resultados de algunos de estos trabajos.

Ortega y otros [7] paralelizan AES sobre multicore usando OpenMP y sobre GPU usando CUDA. Ejecutan ambos algoritmos sobre dos escenarios hardware: 1) multicore Intel i5 M450, GPU Nvidia Geforce GT 330M, 2) multicore Intel i7 960, GPU Nvidia Quadro FX 1800. Muestran que la GPU alcanza el mejor rendimiento en ambos escenarios para datos de entrada que no superan los 256MB.

Duta y otros [8] paralelizan AES (versión OpenSSL) con CUDA y OpenMP. Reportan que obtienen mejor rendimiento con la GPU para datos de entrada de hasta 1.4GB. Sin embargo, no describen las arquitecturas utilizadas.

Nishikawa y otros [9] paralelizan AES (OpenSSL) sobre un multicore (Intel Core i7-920) y dos GPUs Nvidia (Tesla C2050 y GTX 285) y muestran que las GPUs alcanzan el mejor rendimiento para datos menores a 256MB.

En [10] comparamos el rendimiento de dos versiones de AES: la primera realiza cómputo intensivo [11] y la segunda, basada en OpenSSL, reduce el volumen de cómputo reemplazando operaciones por accesos a datos pre-calculados almacenados en memoria (es decir, realiza acceso intensivo a memoria). Los resultados revelan que la segunda versión alcanza mejor rendimiento. Paralelizamos esta versión y la ejecutamos sobre una GPU Nvidia GTX 560TI y una máquina multicore compuesta por dos Intel Xeon E5405 (total 8 cores). Mostramos que el mejor rendimiento es alcanzado sobre la GPU para datos de entrada menores a 256MB.

Desde nuestro conocimiento, al momento no existen trabajos que realicen una comparación de rendimiento de AES sobre GPU, multicore y Xeon Phi. Asimismo, los trabajos de otros autores citados con anterioridad utilizan multicores de escritorio que no superan los 4 u 8 cores.

En este trabajo presentamos un análisis del rendimiento de AES sobre diversas arquitecturas de memoria compartida (multicore Intel E5-2695v4, Xeon Phi 7230 y GPU Nvidia GTX 960), para datos de entrada de distinto tamaño (hasta 2GB). Para esto, desarrollamos dos implementaciones paralelas de AES: AES-CUDA desarrollada con CUDA para ser ejecutada sobre GPUs Nvidia, y AES-OMP desarrollada con OpenMP para ser ejecutada sobre multicores y Xeon Phi. Los resultados revelan que la GPU es la mejor alternativa para cifrar datos de entrada que no superan los 32MB. Sin embargo, para un volumen mayor de datos, el multicore alcanza el mejor rendimiento, seguido por el Xeon Phi.

El resto del paper está organizado de la siguiente manera. La Sección 2 repasa las arquitecturas multicore y manycore actuales y sus características. La Sección 3 describe el algoritmo AES. La Sección 4 introduce nuestras implementaciones paralelas de AES para ser ejecutadas sobre multicore, Xeon Phi y GPUs Nvidia. La Sección 5 muestra los resultados experimentales. Por último, la Sección 6 presenta las principales conclusiones y algunas ideas para investigación futura.

2. Arquitecturas multicore y manycore emergentes

En la actualidad existen arquitecturas paralelas de memoria compartida muy diversas. Por un lado, las CPUs multicore incorporan decenas de núcleos com-

plejos que operan a alta frecuencia. Por ejemplo: las microarquitecturas Intel Broadwell e Intel Skylake (línea de escritorio i9) poseen 22 y 18 núcleos, respectivamente.

Por otro lado, las arquitecturas many-core tales como las GPUs y el Intel Xeon Phi integran gran cantidad de cores simples que operan a menor frecuencia. Ambas arquitecturas utilizan multithreading por hardware para acelerar los cambios de contexto [12,13]. En una CPU convencional el cambio de contexto es costoso ya que el SO debe almacenar en memoria el estado del programa (el contador de programa, el valor de los registros y la información del stack). Sin embargo, en las arquitecturas many-core el costo del cambio de contexto es menor debido al soporte hardware nativo: la arquitectura dispone de un banco de registros más amplio. Una aplicación hará uso efectivo de esta tecnología si soporta alto grado de paralelismo a nivel de hilo. Un gran número de hilos permite ocultar latencias de acceso a memoria y de operación.

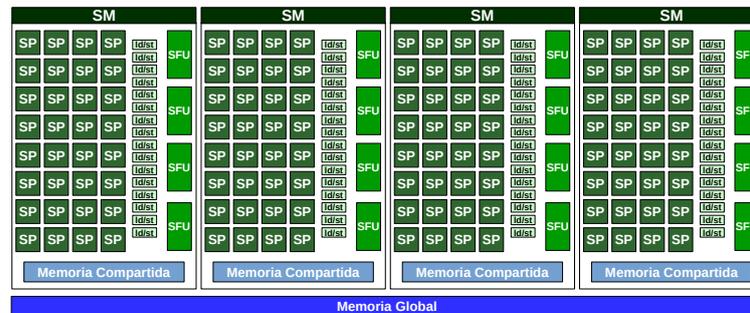


Figura 1. Estructura de una GPU Nvidia.

Las GPUs (Graphics Processing Units) han ganado importancia debido a que alcanzan alto rendimiento para aplicaciones de propósito general. Una GPU actúa como un coprocesador (*device*) conectado a una CPU convencional (*host*) vía PCIe. El cómputo sobre una GPU requiere realizar transferencias desde la memoria del host a la memoria del device (host-to-device o H2D) y viceversa (device-to-host o D2H). La Figura 1 muestra los componentes principales de una GPU Nvidia. En general, una GPU Nvidia está compuesta por un conjunto de multiprocesadores llamados Streaming Multiprocessors (SMs). Cada SM posee varios cores simples denominados Streaming Processors (SPs), un conjunto de unidades para resolver accesos a memoria (unidades load/store) y unidades para funciones especiales (SFUs). Asimismo, las GPUs poseen una jerarquía de memoria de varios niveles, que se diferencian en tamaño y latencia de acceso: la memoria global (mayor latencia), la memoria de constantes (latencia intermedia) y una memoria compartida ubicada en cada SM (menor latencia). Para

programar una GPU Nvidia es necesario utilizar extensiones al lenguaje C, como OpenCL o CUDA, y compilar los programas con un compilador específico.

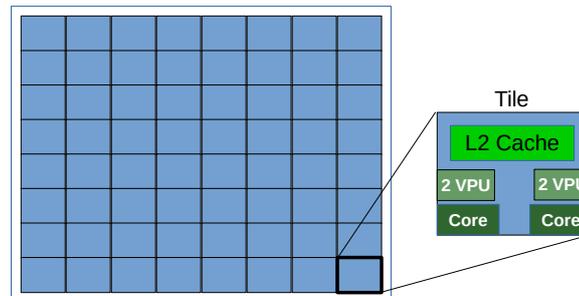


Figura 2. Estructura de un Xeon Phi de segunda generación.

El primer modelo de Xeon Phi conocido como Knights Corner se comercializó como coprocesador, por ello estaba sujeto a limitaciones tales como un tamaño de memoria limitado y podía utilizarse sólo como *device* conectado a un procesador *host* vía PCIe, requiriendo transferencias H2D y D2H. La segunda generación de Xeon Phi conocida como Knights Landing se comercializa además como procesador: puede utilizarse como una CPU independiente en lugar de ser sólo una placa complementaria, por esto no posee las limitaciones del modelo anterior. [14]

El Xeon Phi Knights Corner está compuesto por unos pocos procesadores basados en una modificación de la tecnología Intel Pentium Core (P54C) conectados a través de un bus en anillo. El Xeon Phi Knights Landing está compuesto por procesadores basados en una modificación de la tecnología Silvermont Atom en 14nm. La Figura 2 ilustra su estructura. Específicamente, este modelo está compuesto por varios *tiles* conectados a través de una interconexión en malla 2D. Cada tile posee dos cores, cada core está conectado a dos unidades de procesamiento vectorial (VPUs) AVX-512 y a una caché L2 compartida de 1MB. Cada core soporta cuatro hilos hardware (multithreading por hardware).

Los Xeon Phi son compatibles con procesadores Intel convencionales, es decir pueden ejecutar binarios creados para arquitecturas x86 o x86-64, sin requerir modificaciones en el código fuente. Asimismo, a diferencia de las GPUs Nvidia, no requieren utilizar extensiones a lenguajes ni compiladores específicos. Sin embargo, hacer uso del compilador propietario de Intel puede llevar a obtener un mejor rendimiento.

3. Algoritmo AES

AES (Advanced Encryption Standard) [1] es un algoritmo de cifrado por bloques, es decir los datos a cifrar se dividen en bloques de tamaño fijo (128 bits) y cada bloque se representa como una matriz de 4x4 bytes llamada *estado*, como se muestra en la Figura 3.

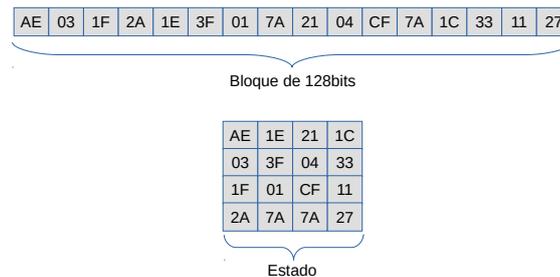


Figura 3. Estado AES

El algoritmo aplica a cada estado 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*. AES es un algoritmo simétrico, por esto utiliza la misma clave para cifrar y descifrar los datos, cuyo tamaño es de 128 bits, según lo indica el estándar. Esta clave es conocida como *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 única operación:

- **AddRoundKey:** realiza un XOR byte a byte entre el estado y la clave inicial.

Cada una de las siguientes nueve rondas (rondas estándar) aplican 4 operaciones en este orden:

- **SubBytes:** reemplaza cada byte del estado por otro de acuerdo a una tabla de sustitución de bytes con valores predeterminados. Este valor resultante es obtenido 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:** rota cíclicamente a izquierda los bytes de las filas del estado de la siguiente manera: una vez en la segunda fila, dos veces en la tercera y tres veces en la cuarta. Notar que la primera fila del estado no se modifica.

- **MixColumns:** aplica a cada columna del estado una transformación lineal y la reemplaza por el resultado de esta operación.
- **AddRoundKey:** opera de la misma manera que en la ronda inicial pero utilizando la siguiente subclave.

La ronda final consiste de 3 operaciones:

- **SubBytes** y **ShiftRows:** operan de la misma manera que en las rondas estándar.
- **AddRoundKey:** opera de la misma manera que en las rondas anteriores pero utilizando la última subclave.

La Figura 4 muestra un esquema del algoritmo.

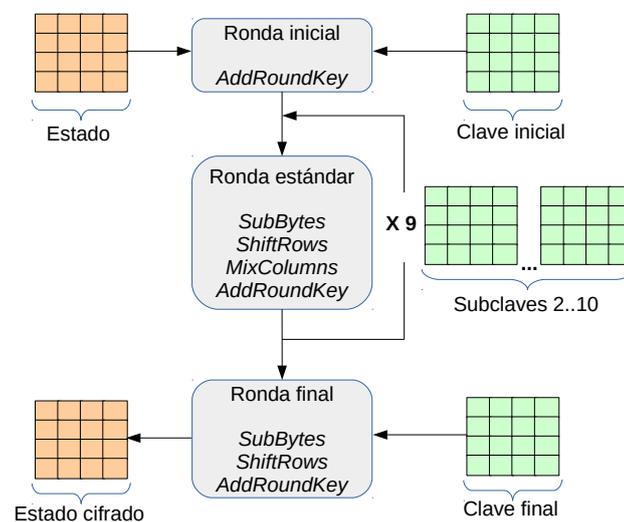


Figura 4. Rondas del algoritmo AES sobre un estado

4. Implementaciones AES

Todas nuestras implementaciones están basadas en el algoritmo AES secuencial provisto por la biblioteca de código abierto OpenSSL [6]. OpenSSL reúne un conjunto de herramientas de administración y bibliotecas relacionadas con criptografía, que proporcionan funciones criptográficas a distintos servicios (OpenSSH, HTTPS, etc).

Cabe destacar que no paralelizamos el proceso de generación de subclaves debido a que su tiempo de ejecución es despreciable.

4.1. AES-OMP

OpenMP es una API para los lenguajes C, C++ y Fortran que permite escribir y ejecutar programas paralelos utilizando memoria compartida. Esta API permite crear un conjunto de hilos que trabajan concurrentemente para aprovechar la potencia de cómputo de arquitecturas multicore y manycore Xeon Phi.

La implementación paralela de AES con OpenMP (AES-OMP) considera a los datos de entrada como bloques consecutivos de 16 bytes. Cada uno de los hilos toma proporcionalmente un conjunto de bloques consecutivos (estados) y aplica el proceso de cifrado.

4.2. AES-CUDA

El algoritmo AES sobre GPU (AES-CUDA) realiza los siguientes pasos. El host copia en la memoria de constantes del device las subclaves y la tabla de sustitución de bytes, ya que ambas son de sólo lectura; luego copia los datos a cifrar en la memoria global del device (transferencia host-to-device o H2D) e invoca al kernel.

Al igual que AES-OMP, AES-CUDA considera los datos de entrada como bloques consecutivos de 16 bytes (estados). Los hilos pertenecientes a un mismo bloque CUDA trabajan sobre estados consecutivos. Para esto, cooperan para cargar de manera eficiente la información a cifrar sobre la memoria compartida de alta velocidad. Luego cada hilo se encarga de cifrar un estado. Por último, los hilos cooperan para trasladar los datos desde la memoria compartida a la memoria global.

Por último, el host recupera los datos cifrados desde la memoria global del device (transferencia device-to-host o D2H).

5. Análisis experimental

Ejecutamos AES-OMP sobre una máquina con dos procesadores Intel Xeon E5-2695v4 y 128GB de RAM. Cada procesador cuenta con 18 cores que operan a 2.10GHz, por lo tanto la máquina posee 36 cores en total. Asimismo, ejecutamos AES-OMP sobre un Intel Xeon Phi 7230 (Knights Landing) y 128GB de RAM. El Xeon Phi posee 64 cores que operan a 1.30GHz y permite ejecutar 4 hilos hardware por core (256 hilos en total). Utilizamos gcc para compilar el código sobre estas plataformas.

Ejecutamos AES-CUDA sobre una GPU Nvidia Geforce GTX 960 compuesta por 1024 cores y 2GB GDDR5 de memoria. Los cores operan a una frecuencia normal de 1127Mhz. Generalmente, los algoritmos CUDA pueden alcanzar distinto rendimiento según la cantidad de hilos por bloque CUDA utilizada. Nuestra implementación obtiene el mejor rendimiento con 256 hilos por bloque CUDA.

Para realizar el análisis de rendimiento ciframos datos de distinto tamaño 16MB, 32MB, 64MB, 128MB, 256MB, 512MB, 1GB y 2GB. En este trabajo mostramos sólo los resultados del proceso de cifrado, ya que el proceso de descifrado presenta un rendimiento similar. Para cada escenario tomamos el tiempo de ejecución promedio de 100 ejecuciones.

Cabe destacar que nuestro análisis considera el tiempo de transferencia de datos (H2D y D2H) en AES-CUDA ya que representa una porción significativa del tiempo total de ejecución. Para demostrar esto, comparamos AES-CUDA considerando las transferencias H2D y D2H con AES-CUDA sin considerar dichas transferencias. La Figura 5 muestra el tiempo normalizado de ejecución de ambas versiones.

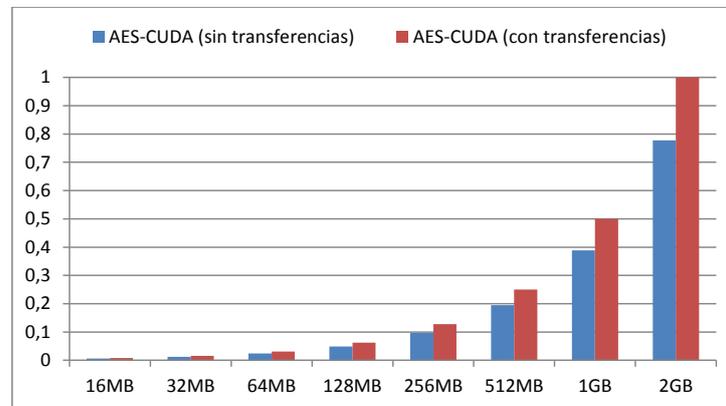


Figura 5. Tiempo de ejecución normalizado de AES-CUDA.

La Figura 6 muestra el Speedup⁴ de AES-CUDA sobre la GPU con 1024 cores, AES-OMP sobre el Xeon Phi con 256 hilos y AES-OMP sobre el multicore Intel E5 con 36 hilos/cores. Calculamos el Speedup con respecto al algoritmo secuencial ejecutado sobre 1 core del Intel E5.

Los resultados revelan que el speedup alcanzado por AES-OMP sobre Intel E5 aumenta a medida que incrementa el tamaño de los datos de entrada. Lo mismo ocurre con el speedup alcanzado por AES-OMP sobre el Xeon Phi, aunque el factor de crecimiento es menor.

⁴ El Speedup es definido como $\frac{T_s}{T_p}$, donde T_s es el tiempo de ejecución del algoritmo secuencial y T_p es el tiempo de ejecución del algoritmo paralelo.

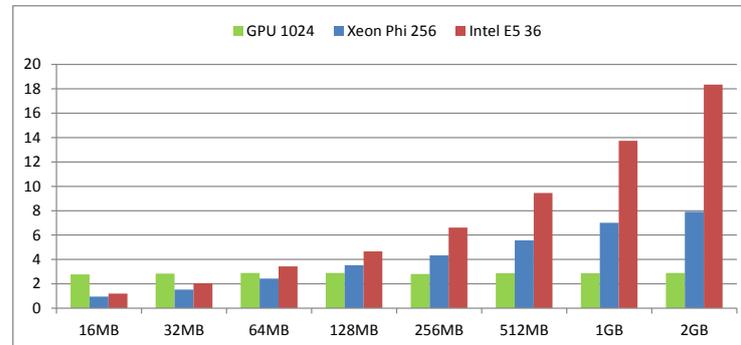


Figura 6. Speedup alcanzado por AES-OMP y AES-CUDA.

Por otro lado, los resultados muestran que el speedup de AES-CUDA no aumenta a medida que incrementa el tamaño de los datos de entrada. Esto se debe a que el tiempo de transferencia de datos entre CPU y GPU crece linealmente con el volumen de datos a transferir.

Como puede observarse, la GPU alcanza el mejor rendimiento cuando el tamaño de los datos a cifrar no supera los 32MB, seguido por el Intel E5 y el Xeon Phi. Para tamaños mayores, el mejor rendimiento lo obtiene el Intel E5, seguido por el Xeon Phi y la GPU.

6. Conclusiones y Trabajo Futuro

En este trabajo presentamos un análisis del rendimiento del algoritmo de cifrado simétrico AES sobre diversas arquitecturas de memoria compartida (multicore Intel E5-2695v4, Xeon Phi 7230 y GPU Nvidia GTX 960), para datos de entrada de distinto tamaño.

Para esto, desarrollamos dos implementaciones paralelas de AES basadas en el algoritmo secuencial provisto por la biblioteca OpenSSL: AES-CUDA desarrollada con CUDA para ser ejecutada sobre GPUs Nvidia, y AES-OMP desarrollada con OpenMP para ser ejecutada sobre multicores y Xeon Phi.

El análisis experimental revela que la GPU es la mejor alternativa para cifrar datos de entrada que no superan los 32MB. Para un volumen de datos mayor, el multicore alcanza el mejor rendimiento, seguido por el Xeon Phi.

Como trabajo futuro, planeamos comparar el rendimiento que proveen estas arquitecturas para aplicaciones con distintos perfiles de cómputo. Asimismo, planeamos utilizar estas arquitecturas de forma conjunta, como un sistema híbrido, para acelerar distintos tipos de aplicaciones.

Referencias

1. Federal Information Processing Standard Publication 197 (FIPS PUB 197): The Official AES Standard, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
2. 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>
3. Manavski, S. A. et al: CUDA compatible GPU as an efficient hardware accelerator for AES cryptography. In: Proc. of the 2007 IEEE International Conference on Signal Processing and Communications (ICSPC'07), pp. 65–68 (2007).
4. Di Biagio, A. et al: Design of a parallel AES for graphics hardware using the CUDA framework. In: Proc. of the 2009 IEEE International Symposium on Parallel & Distributed Processing, pp. 1–8 (2009).
5. Iwai, K. et al: Acceleration of AES encryption on CUDA GPU. *International Journal of Networking and Computing*. 2 (1), pp. 131–145 (2012).
6. The OpenSSL Project: Cryptography and SSL/TLS Toolkit, <https://www.openssl.org>
7. Ortega, J. et al: Parallelizing AES on multicores and GPUs. In: Proc. of the 2011 IEEE INTERNATIONAL CONFERENCE ON ELECTRO/INFORMATION TECHNOLOGY, pp. 1–5 (2011)
8. Duta, C.L. et al: Accelerating Encryption Algorithms Using Parallelism. In: Proc. of the 19th International Conference on Control Systems and Computer Science, pp. 549–554 (2013).
9. Nishikawa, N. et al: High-Performance Symmetric Block Ciphers on Multicore CPU and GPUs. *International Journal of Networking and Computing*. 2(2), pp. 251–268 (2012)
10. Pousa, A. et al: Comparación de rendimiento de algoritmos de cómputo intensivo y de acceso intensivo a memoria sobre arquitecturas multicore. Aplicación al algoritmo de criptografía AES. In: Actas del XXI Congreso Argentino de Ciencias de la Computación (CACIC 2015), pp. 163–172 (2015).
11. Pousa, A. et al: Performance Analysis of a Symmetric Cryptographic Algorithm on Multicore Architectures. In: De Giusti A, Diaz J (eds.) *Computer Science & Technology Series. XVII Argentine Congress of Computer Science Selected Papers*. pp. 57–66. EDULP, Argentina (2012)
12. Hyesoon, K. et al: Performance Analysis and Tuning for General Purpose Graphics Processing Units (GPGPU). Morgan & Claypool, (2012)
13. Rahman, R.: *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*. Apress, Berkeley, CA (2013)
14. Jeffers, J. et al: *Intel Xeon Phi Processor High Performance Programming Knights Landing Edition*. Morgan Kaufmann (2016)