ALGORITMO DE CIFRADO SIMÉTRICO AES.
ACELERACIÓN DE TIEMPO DE CÓMPUTO SOBRE ARQUITECTURAS MULTICORE.

Trabajo Final presentado para obtener el grado de Especialista en Redes y Seguridad

Autor: Lic. Adrián Pousa.

Director: Lic. Javier Díaz.

Co-Director: Ing. Armando De Giusti.

Facultad de Informática - Universidad Nacional de La Plata

Diciembre de 2011

Contenido

Capítulo 6

Objetivo

El objetivo de este trabajo es mostrar la aceleración en el tiempo de cómputo del algoritmo criptográfico Advanced Encryption Standard (AES) con clave de tamaño 128bits, que se obtiene al aprovechar el paralelismo que proveen las arquitecturas multicores actuales utilizando herramientas de programación paralela.

AES es uno de los algoritmos de criptografía más usados en la actualidad, con el crecimiento de las redes y la información que se maneja hoy en día puede ser necesario cifrar un volumen muy grande de información para lo que se requiere mayor velocidad en los procesadores, pero esto actualmente no es posible debido a que los procesadores han llegado al límite de velocidad por problemas térmicos y de consumo, por esta razón se está incrementando la cantidad de procesadores en los equipos.

Como aporte de la concreción de este trabajo se pretende presentar un análisis de rendimiento que muestre cómo a pesar de las limitaciones de velocidad de los procesadores, es posible, mediante herramientas de programación paralela, aprovechar las arquitecturas multicore para acelerar el cómputo del algoritmo AES y así reducir el tiempo de cifrar información ya sea para almacenarla o enviarla por la red.

Introducción

Desde mucho antes que existieran las computadoras los seres humanos han tenido la necesidad de intercambiar mensajes de forma segura, de manera que solo puedan ser leídos por las personas a quienes van dirigidos. A partir de esta necesidad es que nace la criptografía, esta provee técnicas de codificación y decodificación de información para un intercambio seguro. Desde la aparición de las computadoras y más aun con el crecimiento de las redes, la necesidad de intercambiar información de manera segura fue mayor y es donde aparecen implementaciones de distintos sistemas de cifrado.

Existen dos sistemas de cifrado que involucran distintos tipos de algoritmos, los sistemas simétricos y los sistemas asimétricos, ambos con sus ventajas y sus desventajas.

Dentro de los sistemas simétricos se encuentra el algoritmo Advanced Encryption Standard (AES), que es uno de los algoritmos más utilizados en la actualidad, considerado por el gobierno de los Estados Unidos como un algoritmo seguro para protección nacional de información y del cual, aun no se conocen ataques eficientes que puedan vulnerarlo.

Hoy en día la el volumen de información que se maneja en ocasiones es muy grande y en algunos casos es necesario almacenar esta información o transmitirla en forma segura, AES es un algoritmo simple y rápido, pero aun así, el tiempo de computo de cifrar un gran volumen de información puede ser importante, es necesario en estos casos contar con procesadores más veloces.

Los procesadores han venido siendo más rápido cada año, pero en la actualidad este crecimiento se ve interrumpido por problemas térmicos y de consumo dentro de los procesadores. Por esta razón es que las arquitecturas actuales tienen más de un procesador de manera de poder aprovechar el paralelismo que proveen, y en la medida que se pueda poder acelerar el tiempo de cómputo de las aplicaciones.

Siempre fue común el uso de arquitecturas como multiprocesadores y clusters dentro del ámbito académico-científico, hoy en día los costos de estas arquitecturas han bajado y es posible encontrarlas fuera de este ámbito; también se le ha dado importancia a las placas para procesamiento grafico (GPU) que han sido usadas con mucho éxito para aplicaciones de propósito general.

Estas arquitecturas junto con el uso de herramientas de programación paralela adecuadas para cada caso, como son OpenMP, MPI y CUDA permiten paralelizar algoritmos de manera de acelerar el tiempo de cómputo.

Como muestra este trabajo, el algoritmo AES puede ser implementado con herramientas paralelas y ejecutado sobre arquitecturas multicores de manera de reducir el costo de cifrado y descifrado de datos ya sea para almacenarlos o para hacer una transferencia importante de datos sensibles sobre una red pública.

Capítulo 1

Criptografía

1.1 Definición

La criptografía (del griego oculta y escribir, literalmente escritura oculta) es el arte o ciencia de cifrar (encriptar) y descifrar (desencriptar) información utilizando técnicas que hagan posible el intercambio de mensajes de manera segura que solo puedan ser leídos por las personas a quienes van dirigidos.

Existen dos tipos básicos de sistemas de cifrado:

- Sistemas de cifrado *simétricos* (o sistemas de clave secreta o de clave privada).
- Sistemas de cifrado *asimétrico* (o sistemas de clave pública).

1.2 Sistemas de cifrado simétricos

Los sistemas simétricos utilizan la misma clave para encriptar y desencriptar. Existen dos modos de operación básicos:

- Cifrado en bloques: La información a cifrar se divide en bloques de longitud fija (8,16, ... bytes) y luego se aplica el algoritmo de cifrado a cada bloque utilizando una clave secreta. Ejemplos: DES, AES.
  Existen distintos modos de operación dependiendo de como se mezcla la clave con la información a cifrar:

  - Modo ECB (Electronic Codebook): El texto se divide en bloques y cada bloque es cifrado en forma independiente utilizando la clave. Tiene la desventaja que puede revelar patrones en los datos.

  - Modo CBC (CBC): El texto se divide en bloques y cada bloque es mezclado con la cifra del bloque previo, luego es cifrado utilizando la clave.

  - Modos CFB (Cipher FeedBack) y OFB (Output FeedBack).

- Cifrado de flujo: Para algunas aplicaciones, como el cifrado de conversaciones telefónicas, el cifrado en bloques es inapropiada porque los datos se producen en tiempo real en pequeños fragmentos. Las muestras de datos pueden ser tan pequeñas como 8 bits o incluso de 1 bit. El algoritmo genera una secuencia pseudoaleatoria (secuencia cifrante o keystream en inglés) de bits que se emplea como clave. El cifrado se realiza combinando la secuencia cifrante con el texto claro. Ejemplo: RC4.

Los sistemas simétricos tienen las siguientes ventajas:

- Gran velocidad de cifrado y descifrado.
- No aumenta el tamaño del mensaje
- Tecnología muy conocida y difundida.

Pero presentan las siguientes desventajas:

- La seguridad depende de un secreto compartido entre el emisor y el receptor.
- La administración de las claves no es "escalable".
- La distribución de claves debe hacerse a través de algún medio seguro.

1.3 Sistemas de cifrado asimétricos

Los sistemas asimétricos utilizan dos claves, una privada y una publica (siendo una la inversa de la otra). Ambas pueden ser usadas para encriptar y desencriptar información. Dichas claves están matemáticamente relacionadas entre sí:

- La clave pública está disponible para todos.
- La clave privada es conocida solo por el individuo.

Existen varios algoritmos muy utilizados por ejemplo Diffie-Hellman , RSA, DSA.

Este sistema tiene además dos modos de cifrado:

- Encripción: el mensaje es encriptado usando la clave pública del receptor, el mensaje encriptado es enviado al destinatario, el mensaje recibido se desencripta usando la clave privada del receptor, garantizando así la confidencialidad del mensaje.

- Autenticación: el mensaje es encriptado usando la clave privada del emisor, el mensaje encriptado se envía a uno o más receptores, el mensaje se desencripta usando la clave pública del emisor. Esto garantiza la autenticidad del emisor y la integridad del mensaje.

Este sistema de cifrado tiene las siguientes ventajas:

- No se intercambian claves.
- Es una tecnología muy difundida.
- Sus modos cubren los requisitos de seguridad de la información.

Pero presentan las siguientes desventajas:

- Requiere potencia de cómputo.
- El tamaño del mensaje cifrado es mayor al del original.

Capítulo 2

Algoritmo AES

2.1 Historia del algoritmo AES

Es un algoritmo de cifrado simétrico desarrollado por los estudiantes Vincent Rijmen y Joan Daemen de la Katholieke Universiteit Leuven en Bélgica, bajo el nombre "Rijndael" fue presentado en 1997 al concurso organizado por el Instituto Nacional de Normas y Tecnologías (NIST) para elegir el mejor algoritmo de cifrado; el algoritmo gano el concurso transformándose en un estándar en el año 2002, con algunos cambios fue posteriormente renombrado AES (Advanced Encryption Standard) y se convirtió en uno de los algoritmos más utilizados en la actualidad.

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. Hasta el momento no se conocen ataques eficientes, los únicos conocidos son los denominados ataques de canal auxiliar[1].

2.2 Mención sobre los fundamentos matemáticos

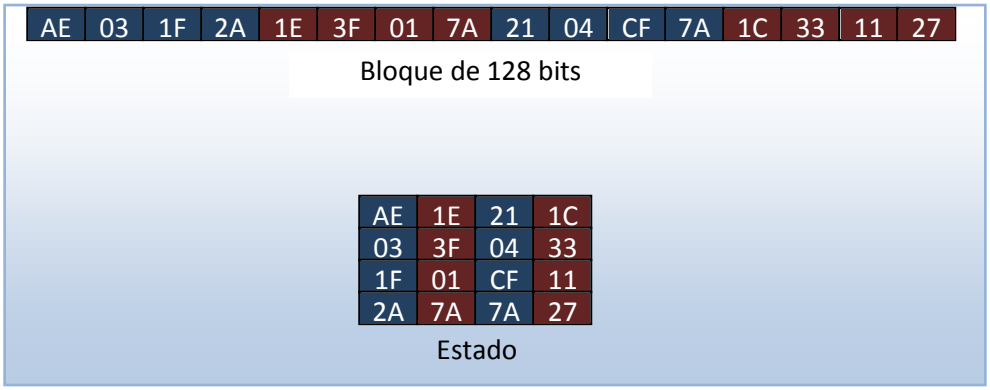
[1] Un ataque de canal auxiliar no ataca al algoritmo de cifrado sino que aprovecha vulnerabilidades de las implementaciones que pueden revelar datos a medida que se realiza el cifrado.[12] [13] [14]

AES toma como elemento básico al byte (8 bits) y ve a los bytes como elementos del campo finito de Galois o GF(2^8), toda operación del algoritmo está basada en operaciones sobre este campo finito, rotaciones de bytes y operaciones de suma módulo 2.

No es objetivo de este trabajo explicar en detalle los extensos fundamentos matemáticos en los que se basa el algoritmo AES.

2.3 Bloques AES

AES es un algoritmo de cifrado por bloques, inicialmente fue diseñado para tener longitud de bloque variable pero el estándar define un tamaño de bloque de 128 bits, por lo tanto los datos a ser encriptados se dividen en segmentos de 16 bytes (128 bits) y cada segmento se lo puede ver como un bloque o matriz de 4x4 bytes al que se lo llama estado, este se organiza de la siguiente forma:

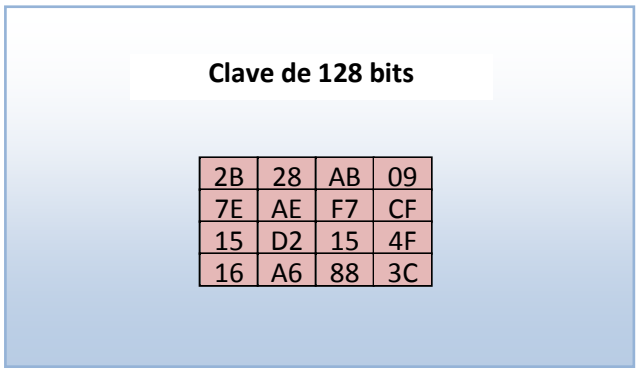


2.4 Claves

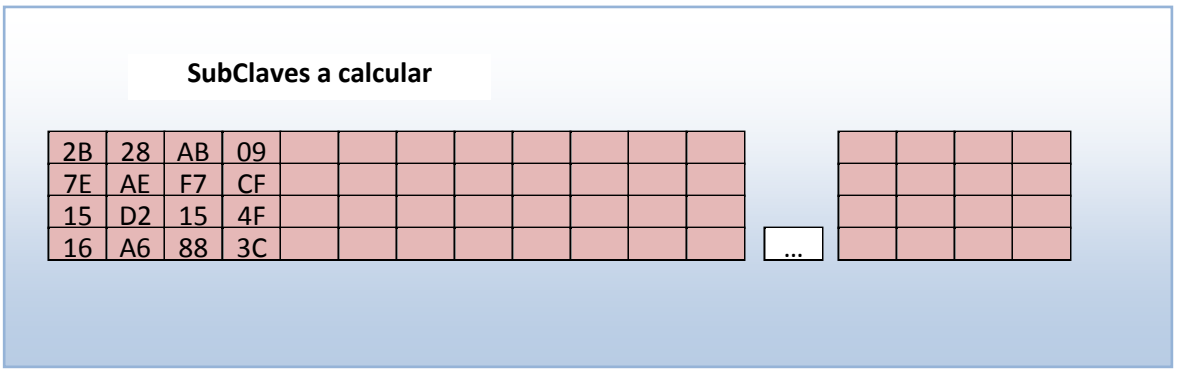
Por ser simétrico, se utiliza la misma clave para encriptar como para desencriptar, la longitud de la clave puede ser de 128, 192 o 256 bits según especifica el estándar, esto permite tres implementaciones conocidas como AES-128, AES-192 y AES-256, el presente trabajo está basado en AES-128.

Partiendo de una clave inicial de 16 bytes (128 bits), que también se la puede ver como un bloque o matriz de 4x4 bytes, se generan 10 claves, estas claves resultantes junto con la *clave inicial* son denominadas *subclaves.*

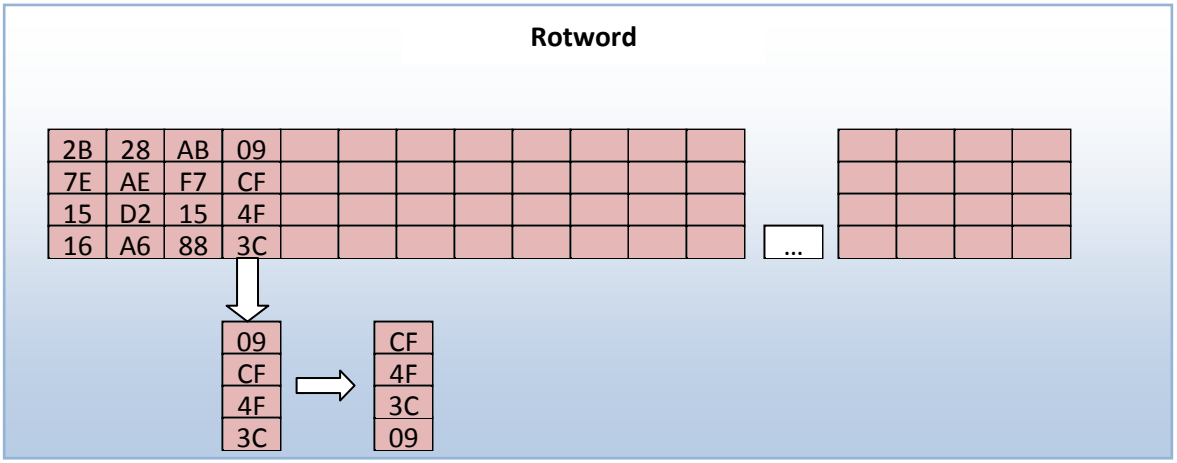
El proceso de generación de subclaves parte de la clave inicial vista como una matriz de 4x4 bytes:



Para mostrar claramente como se calculan las subclaves, el conjunto de subclaves puede verse como una matriz de 4 filas x 44 columnas, o sea una subclave a continuación de otra:

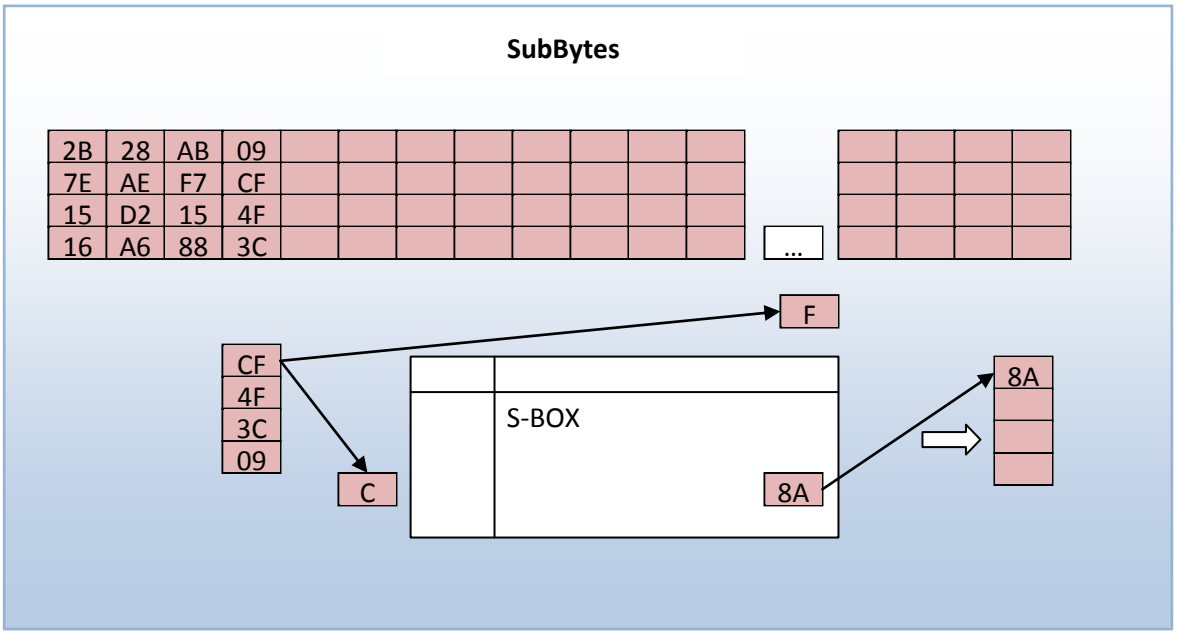


Para calcular la primera columna de la siguiente subclave se toma la última columna de la subclave anterior (en este caso la clave inicial) y se aplica una operación llamada Rotword que consiste en realizar una rotación del primer byte hacia el último lugar en la columna:

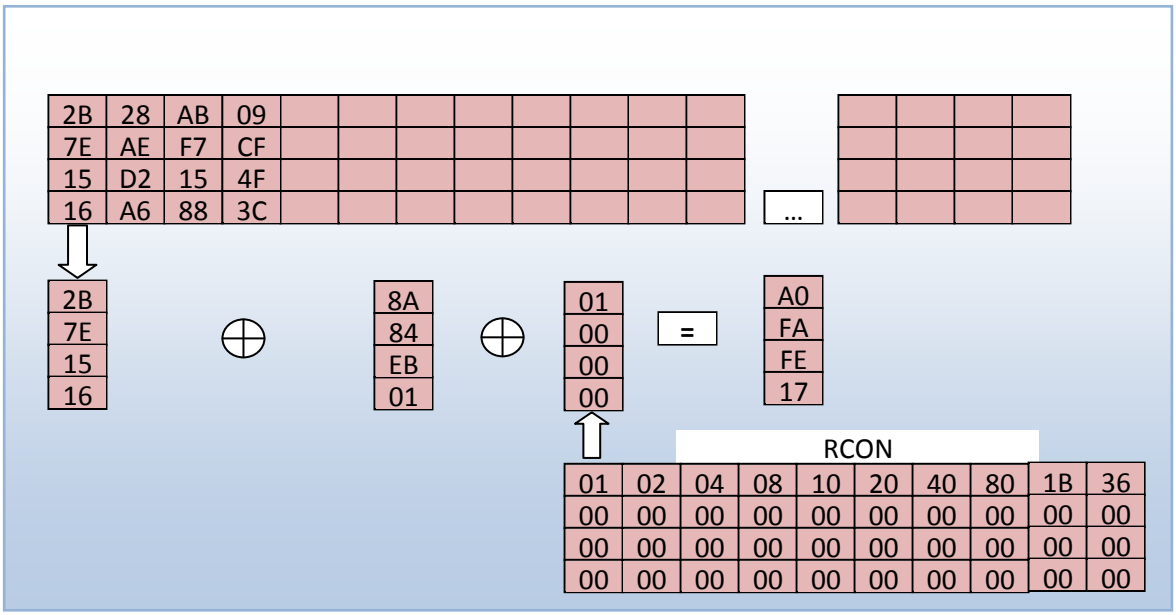


Luego, a la columna resultante, se aplica una operación llamada SubBytes que consiste en

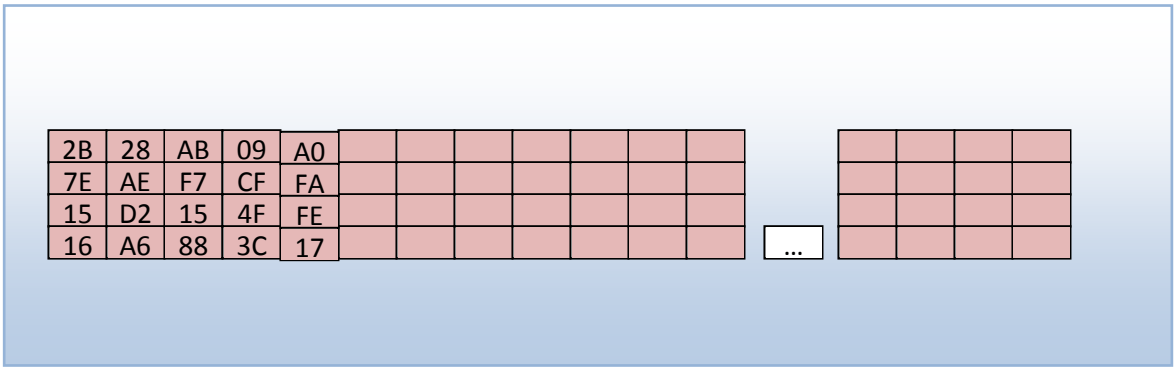
reemplazar cada byte de la columna ya rotada por un byte almacenado en una tabla llamada S-Box, esta tabla contiene pre calculados el resultado de aplicarle a cada byte la inversión en el campo GF y una transformación afín, la dimensión de la tabla es de 16x16 bytes donde los índices tanto de las columnas como de las filas van de 0 a F, para obtener la transformación S-Box de un byte se toman los primeros 4 bits como el índice de la fila de la tabla y los segundos 4 como índice de la columna de la tabla:



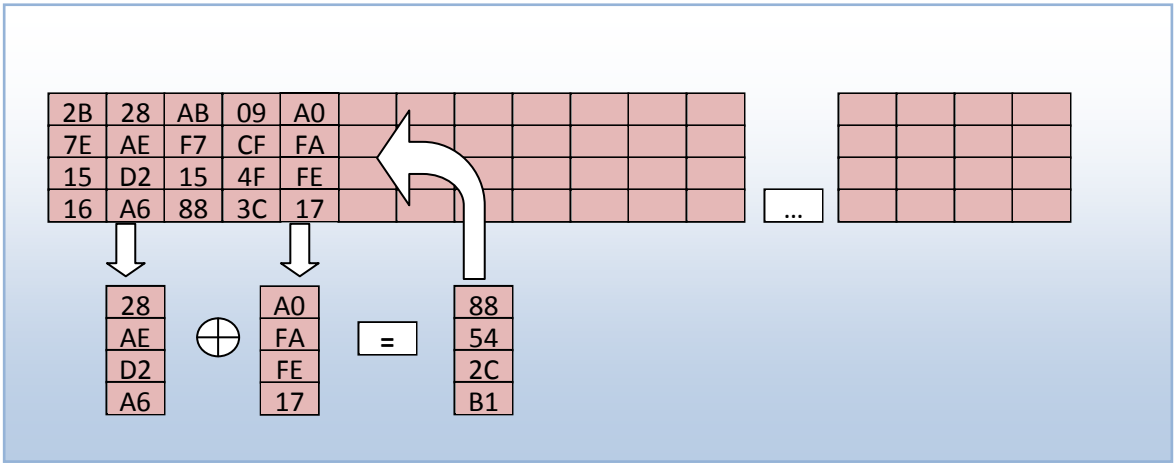
Luego al resultado se le aplica un XOR byte a byte con la columna 4 posiciones atrás (en este caso la primer columna de la clave inicial) y un XOR byte a byte con una columna de una tabla llamada RCON que mantiene en la primer fila constantes 2^i en el campo GF y en las restantes filas 0, por ser la primer subclave la que estamos calculando se toma para el cálculo la primer columna de la tabla RCON, para las siguientes subclaves se toma la próxima columna no utilizada de esta tabla:



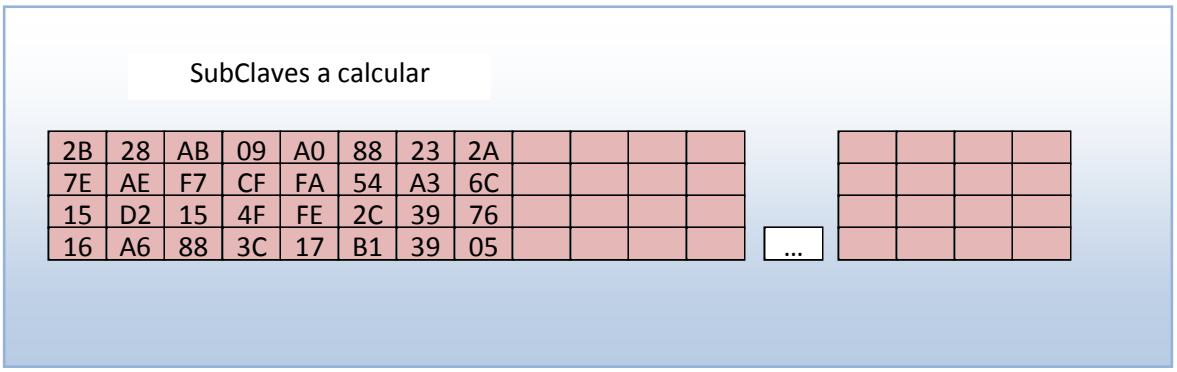
El resultado de esta última operación será la primera columna de la subclave calculada (en este caso la segunda subclave siguiente a la inicial):



Para calcular las tres columnas siguientes se hace un XOR entre la columna anterior y la columna de cuatro posiciones atrás:



Una vez aplicadas estas operaciones se tiene una nueva subclave:



Y se procede de la misma forma para calcular las siguientes subclaves.

Al finalizar se tendrán 11 subclaves, cada una de estas subclaves se aplica en una de las rondas de operaciones que se explican en detalle a continuación.

2.5 Cifrado AES: Rondas y operaciones

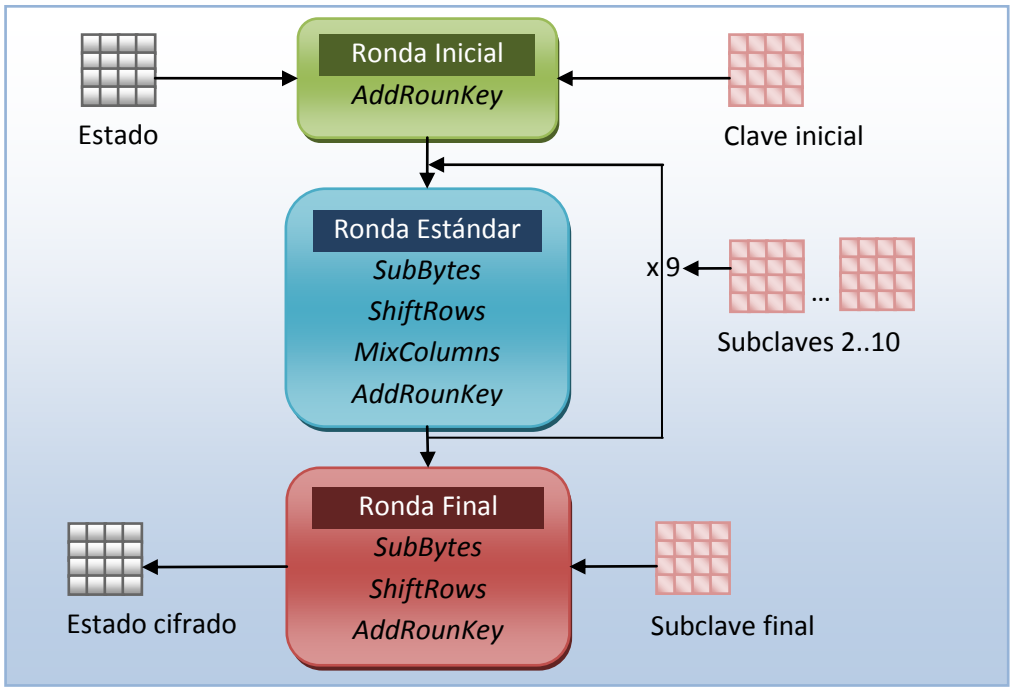
El proceso de cifrado del algoritmo consiste en aplicar a cada estado un conjunto de operaciones agrupadas en lo que se denominan rondas, el algoritmo realiza 11 rondas, donde en cada ronda se aplica una subclave diferente.
Las 11 rondas se pueden clasificar en 3 tipos:

- 1 ronda inicial (se aplica la subclave inicial).
- 9 rondas estándar (se aplican las 9 subclaves siguientes, una en cada ronda).
- 1 ronda final (se aplica la última subclave).

Las operaciones que realiza el algoritmo dentro de las rondas se reducen a 4 operaciones básicas:

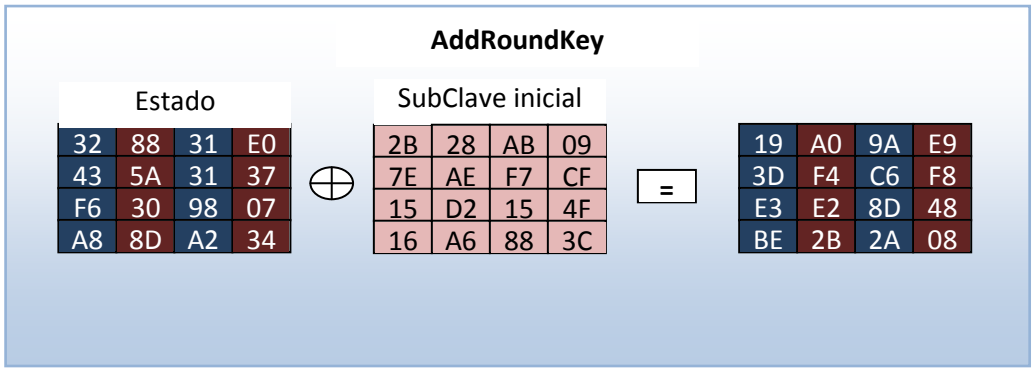
- SubBytes.
- ShiftRows.
- MixColumns.
- AddRoundKey.

A continuación se muestra un diagrama de como se aplican las operaciones y claves en cada una de las rondas:



2.5.1 Ronda inicial

La ronda inicial aplica solamente la operación **AddRoundKey** que no es más que un XOR byte a byte entre el bloque a cifrar y la clave inicial.



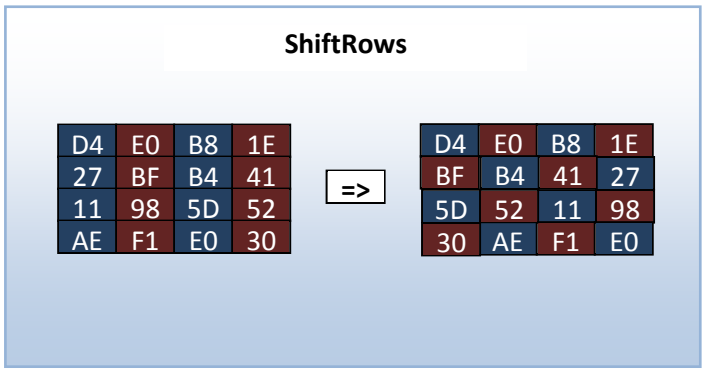
2.5.2 Rondas estándar

Luego se realizan 9 rondas estándar donde cada ronda consiste en las siguientes operaciones:

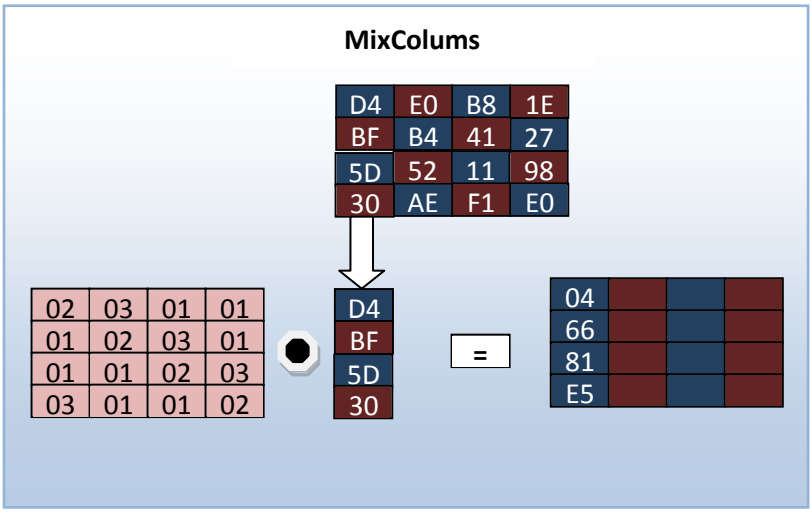
SubBytes: Cada byte del estado se reemplaza por otro valor de acuerdo a la tabla de sustitución de bytes S-Box ya vista en el cálculo de las subclaves.

ShiftRows: En cada fila del estado, a excepción de la primera, se rotan

circularmente hacia la izquierda los bytes, en la segunda fila se rotan una posición, en la tercera dos posiciones y en la cuarta tres posiciones.



MixColumns: A cada columna del estado se le aplica una transformación lineal, esto es multiplicarlo por una matriz predeterminada en el campo GF.



AddRoundKey: Se aplica la misma operación que en la ronda inicial pero utilizando otra subclave.

2.5.3 Ronda final

Por último la ronda final consiste en las operaciones de:

SubBytes: igual al de la ronda estándar.

ShiftRows: igual al de la ronda estándar.

AddRoundKey: igual al de la ronda inicial y estándar pero aplicando la última subclave.

2.6 Descifrado AES

El proceso de descifrado aplica las mismas operaciones que el cifrado pero de forma inversa utilizando las mismas subclaves generadas en orden inverso, además se utiliza una matriz distinta en la operación MixColumns de manera de obtener la inversa de la transformación lineal aplicada en el proceso de cifrado.

Capítulo 3

Arquitecturas multicore y herramientas paralelas

3.1 Evolución hacia las arquitecturas multicore

Desde prácticamente sus inicios, las computadoras han sido más rápidas cada año, hoy en día este crecimiento se ve interrumpido debido a problemas térmicos y de consumo en los procesadores, por este motivo los fabricantes decidieron integrar múltiples procesadores en el mismo circuito integrado dando lugar a las arquitecturas multicore existentes en el mercado actual, de esta manera se puede aprovechar el paralelismo que estas arquitecturas proveen para, en la medida que se pueda, acelerar el computo haciendo más rápidas las aplicaciones.

3.1.1 Maquinas multiprocesadores, multicores y clusters

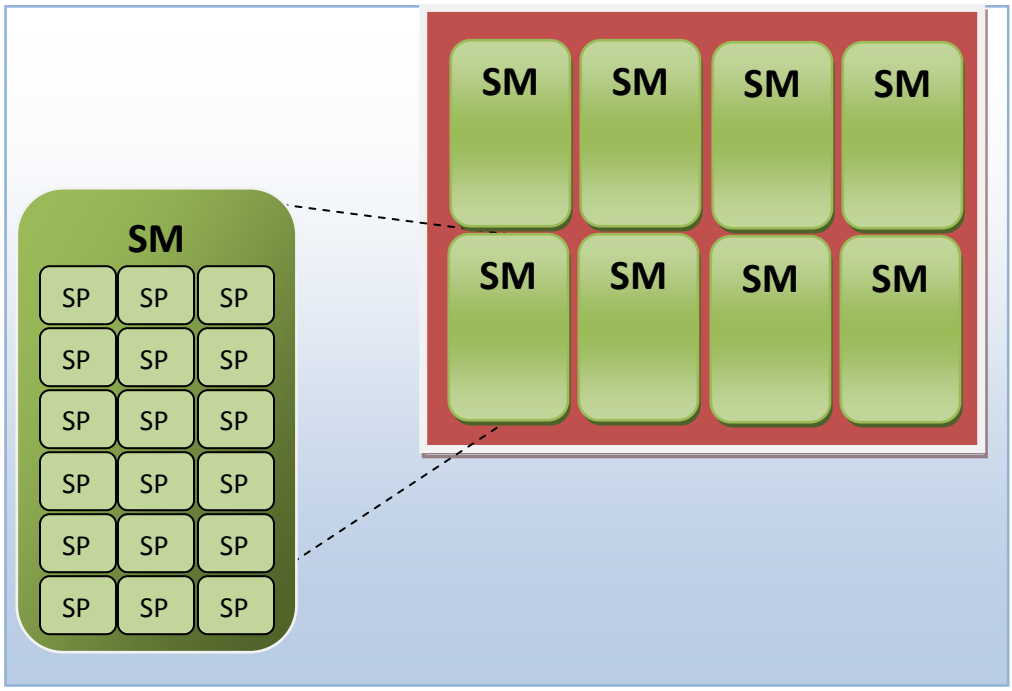
Casi desde el primer momento que se utilizaron procesadores existían maquinas con más de un procesador (ILLIAC IV o CRAY-1), aunque no en el mismo circuito integrado; el costo de tener una máquina de estas características era muy elevado, además del gran tamaño que tenían, el crecimiento de la tecnología en el ámbito de las redes permitió que se pudieran conectar maquinas en red pudiendo verlas como una maquina multiprocesador, a esto de lo llamó clúster. El clúster tiene la ventaja de poder escalar en el número de procesadores y en un principio tenía un costo mucho menor a las maquinas multiprocesador, por lo tanto se convirtió en una de las maquinas paralelas más usadas en el ámbito académico-científico. La evolución de los clusters dio lugar a arquitecturas multicluster (varios clusters conectados en red) y arquitecturas actuales más complejas como son Grid y Cloud.

Hoy en día los costos de tener una maquina con más de un procesador no son tan altos y son accesibles, los procesadores además están dentro de un mismo circuito integrado reduciendo el tamaño que tenían este tipo de máquinas en un principio; si además consideramos tenerlas conectadas en red nos da la posibilidad de tener clusters con una gran cantidad de procesadores.

3.1.2 Graphics Processing Units

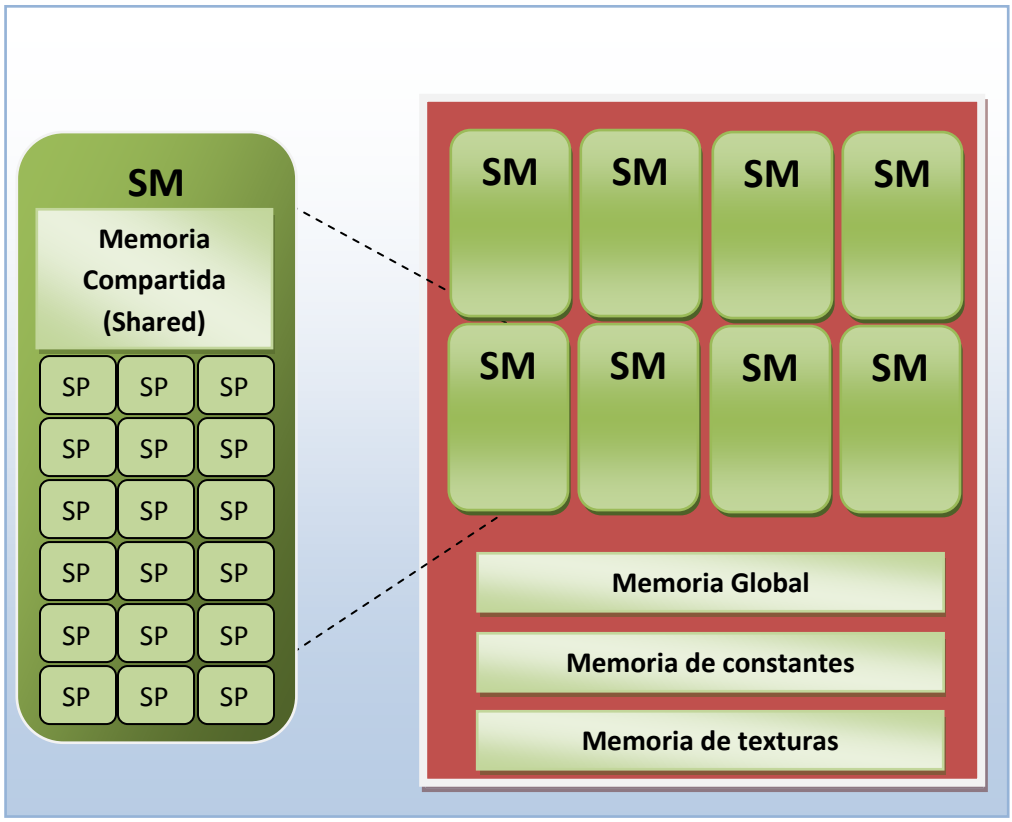
En los últimos años se le dio importancia a las GPU (Graphics Processing Units), estas son una arquitectura multicore (o también llamadas manycores) con una gran cantidad de procesadores simples dedicados a procesamiento gráfico, pero se logró utilizarlas con mucho éxito en aplicaciones de propósito general logrando un muy buen rendimiento, por este motivo también se las conoce como GPGPU (General Purpose GPU).

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.



Las GPU poseen distintos tipos de memoria:

- Una memoria global que comparten todos los SMs y que su acceso es costoso por lo tanto hay que minimizar la cantidad de accesos.
- Una memoria de constantes de solo lectura y una memoria de texturas también compartidas por todos los SMs.
- Una memoria compartida o shared de acceso rápido ubicada en cada SM y de acceso solo restringido a este.
- Registros internos a cada SM.



3.2 Herramientas paralelas

Para poder sacar provecho de las arquitecturas multicores es necesario contar con herramientas de programación paralela que permitan crear procesos, comunicarlos y sincronizarlos de manera de que estos cooperen en la solución de un problema común, de esta forma y mediante el paralelismo poder lograr mejorar los tiempos de las aplicaciones.

3.2.1 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.

El modelo que sigue es un modelo Fork-Join, se parte de un programa secuencial y mediante directivas se indica que hay partes del código que deben ejecutarse en paralelo, en ese momento el programa crea tantos procesos como sean necesarios y no más de los que provee la arquitectura, cada uno ejecutará paralelamente la parte del código que le corresponda y cuando todos terminan se continua con la ejecución del programa secuencial.

El modelo además es un modelo de memoria compartida, los procesos comparten el mismo espacio de direcciones de memoria por lo tanto se comunican a través de esta.

3.2.2 MPI

MPI (Message Passing Interface) es una especificación de una API para programación con memoria distribuida, es decir, donde los procesos se comunican a través del envío y recepción de mensajes, existen varias librerías que implementan esta API que son usadas en los lenguajes C, C++ y Fortran. Permite ser usada tanto en máquinas multicore como en arquitecturas tipo clúster, así como también en una combinación de ambas de forma de aprovechar todos los cores que estas arquitecturas proveen.

Se parte de un programa MPI, cuando se envía a ejecutar se hace una copia del programa para cada procesador, la ejecución de cada programa dará lugar a procesos que ejecutarán cada uno la parte del código que le corresponda, y si necesitan comunicarse lo harán mediante el envío y recepción de mensajes.

3.2.3 Cuda

A partir del éxito de las GPU en aplicaciones de propósito general , 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.

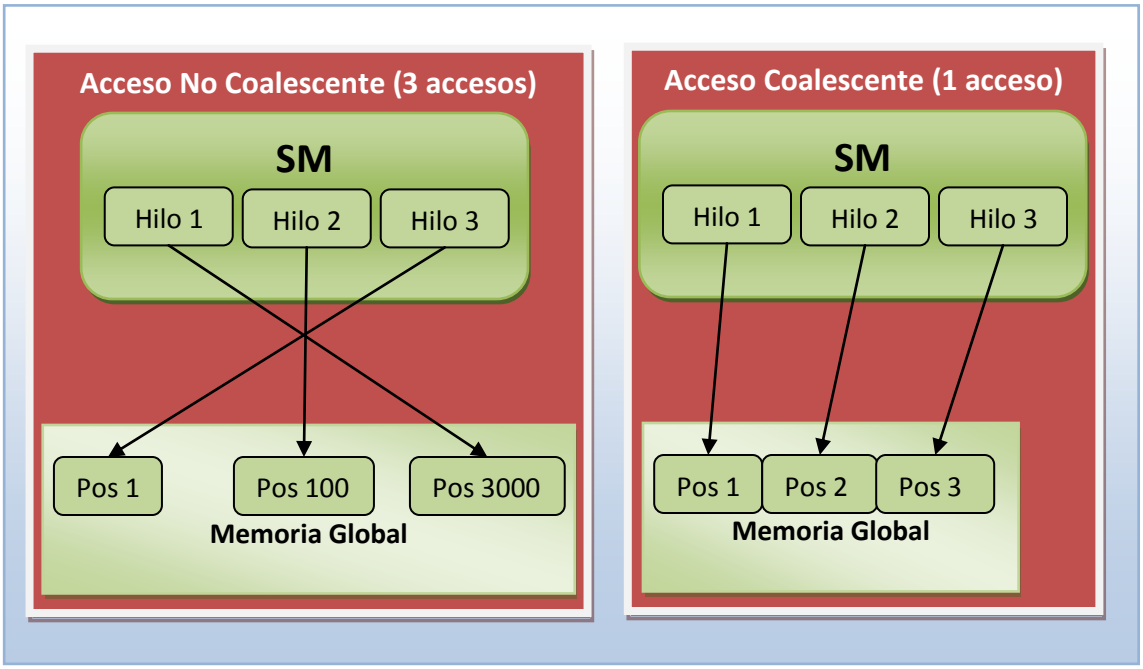
El sistema de cómputo para un programador CUDA está compuesto por una o más GPUs llamadas *devices* y la maquina donde está instalada la GPU que denomina *host*. Un programa CUDA se divide en fases a ejecutar en el host y fases a ejecutar en el device, esto hay que indicarlo explícitamente en el código.

Una ejecución en una GPU generalmente implica, que el host copie los datos a ser procesados por el device desde la memoria RAM del host a la memoria global del device; luego el host invoca el código a ejecutar en el device, este código recibe el nombre de kernel, en la invocación al mismo se debe indicar la cantidad de hilos que se ejecutaran en el device, los hilos se suelen agrupar en lo que se denomina grid, un grid es un conjunto de bloques unidimensionales, bidimensionales o tridimensionales de hilos, cada bloque a su vez puede tener una, dos o tres dimensiones. Los hilos que componen un bloque serán asignados a un SM para su ejecución.

Generalmente, como el acceso a la memoria global es costoso, los hilos de un mismo bloque suelen cooperar para llevar los datos desde la memoria global a la memoria compartida o shared ubicada en el SM para luego trabajar en esta, el acceso deben realizarlo de forma coalescente; la coalescencia es una técnica de implementación para conseguir que hilos consecutivos, cuando solicitan acceso a memoria global, pidan direcciones lógicas contiguas. Esta técnica permite minimizar el número de segmentos de transacción solicitados a la memoria global de esta forma con un solo acceso se pueden traer varios datos.

Esto además es posible porque el computo sobre una GPU sigue el modelo SIMD (Single instruction multiple data), una misma instrucción es ejecutada por varios hilos al mismo tiempo cada uno sobre distintos datos, cuando un hilo ejecute una instrucción de acceso a memoria también otros hilos van a estar ejecutando esa instrucción en el mismo instante; si estos hilos son

consecutivos y las posiciones de memoria a las que acceden son contiguas, es posible con un solo acceso traer una cantidad importante de datos desde la memoria global a la memoria compartida.



Luego que se cargaron los datos en memoria shared, los hilos trabajan accediendo solo a esta que es mucho más rápida; al terminar la ejecución vuelven a cooperar para llevar los datos ya procesados de nuevo a la memoria global de forma coalescente.

Una vez que se terminó la ejecución en el device, el host debe copiar los resultados de la memoria global del device a su memoria RAM.

Capítulo 4

Algoritmo AES Implementaciones

4.1 Implementación y software utilizado

Se realizaron cuatro implementaciones de este algoritmo, una implementación secuencial y las restantes utilizando distintas herramientas que representan distinto modelos de programación paralela, estas herramientas son OpenMP, MPI y CUDA.

En todos los casos el lenguaje utilizado fue C, como compilador se utilizó GCC sobre plataforma linux.

En el caso de MPI y CUDA utilizan como compilador wrappers que en definitiva terminan llamando al compilador GCC.

A continuación se describen la forma en que se implementó el algoritmo AES en cada caso mencionando solo la forma de cifrado dado que el descifrado tiene las mismas características.

4.2 Implementación secuencial

La implementación secuencial del algoritmo genera las subclaves a partir de la clave inicial. A continuación, de los datos a cifrar va tomando estados de 128 bits y aplica las rondas utilizando las subclaves generadas inicialmente.

4.3 Implementaciones paralelas

Las implementaciones paralelas consideran a los datos de entrada como bloques consecutivos de 128 bits. Además se tiene una cantidad determinada de procesos o hilos, y cada uno se encargará de cifrar un conjunto de bloques. Para simplificar el algoritmo 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.

4.3.1 Implementación en el modelo de memoria compartida usando OpenMP

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.

4.3.2 Implementación en el modelo de memoria distribuida usando MPI

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 a los restantes. Luego distribuye proporcionalmente los bloques de 16 bytes (estados) entre los procesos, incluyéndose a sí mismo. Cada proceso encriptará los bloques que le correspondan y retornará los bloques cifrados.

4.3.3 Implementación en el modelo GPU usando CUDA

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.

Capítulo 5

Análisis de rendimiento

5.1 Hardware utilizado

El algoritmo secuencial fue ejecutado sobre un core de una máquina con arquitectura Intel Xeon E5405 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 clúster de 4 máquinas con la arquitectura anteriormente mencionada conectados a 1Gbit Ethernet utilizando un total de 32 cores.

El algoritmo CUDA fue ejecutado en una tarjeta gráfica Nvidia Geforce GTX 560TI con 1GB de RAM que posee 384 SPs, distribuidos en 8 SMs (48 SPs cada uno), cada uno siendo capaz de ejecutar un máximo de 1536 hilos; se utilizaron bloques de 256 hilos, por lo tanto el máximo número de bloques que podrá ejecutar un SM será 6 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.

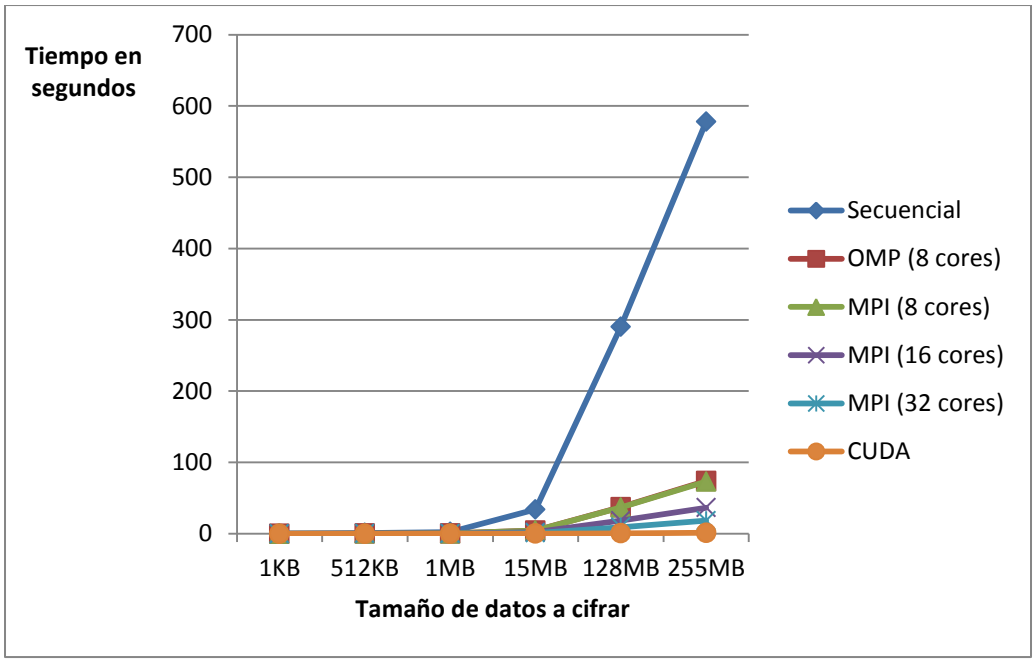
5.2 Tiempos de ejecución

Para los tiempos de ejecución solo se tuvo en cuenta el tiempo de cifrado dado que el tiempo de descifrado presenta resultados similares.

La siguiente tabla muestra los tiempos de ejecución promedio, en segundos, correspondientes al cifrado para las distintas herramientas paralelas sobre varios 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

La siguiente imagen muestra gráficamente los tiempos de ejecución anteriores:



Como se puede apreciar los tiempos de los algoritmos que utilizan OpenMP y MPI con 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, se ve que se reduce el tiempo de computo a medida que los cores aumentan. 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.

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 alcanzado con CUDA, es necesario, en MPI, elevar la cantidad de cores a algo más de 256. Es evidente la relación en el tamaño del hardware para lograr ese rendimiento, un clúster de 256 cores, con el tipo de procesadores que se utilizaron en las pruebas, tiene un tamaño importante en relación a una GPU de unos pocos centímetros.

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 de tiempo. 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, por lo tanto la transferencia de datos tiene poca influencia en los tiempos de ejecución sobre la GPU.

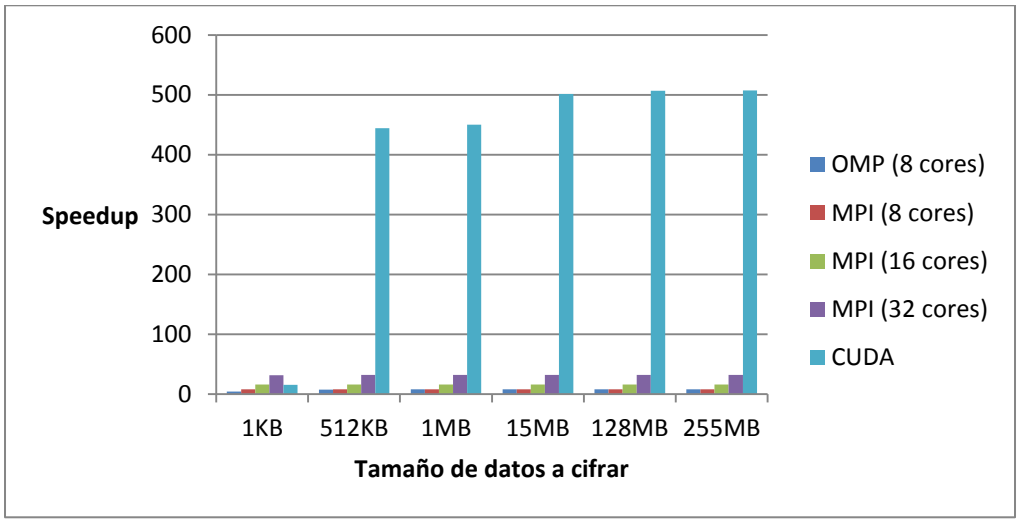
5.3 Aceleración (SpeedUp)

La aceleración (o speedup) mide cuán rápido es el algoritmo paralelo con respecto al algoritmo secuencial, se calcula como Ts/Tp, siendo Ts el tiempo secuencial y Tp el tiempo paralelo.

La siguiente tabla muestra la aceleración alcanzada en cada caso, en el cifrado, para las distintas herramientas paralelas sobre varios tamaños de datos de entrada, en relación al algoritmo secuencial ejecutado sobre la arquitectura Intel Xeon E5405.

	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

La siguiente imagen muestra gráficamente la aceleración presentada en la tabla anterior:



En todos los casos se consigue una buena aceleración, pero como se puede observar la aceleración alcanzada por CUDA sobre la GPU es importante y de un orden de magnitud mucho mayor que la alcanzada al ejecutar el algoritmo con las otras herramientas, algo que ya mostraban los tiempos de ejecución presentados anteriormente.

Capítulo 6

Conclusiones y trabajo a futuro

6.1 Conclusiones

Se presentó el algoritmo de cifrado simétrico AES, se realizó una implementación secuencial en lenguaje C, con clave de 128 bits, adaptando luego esta implementación a herramientas de programación paralela, para poder ejecutarlo aprovechando las arquitecturas multicore actuales.

Las herramientas de programación paralela utilizadas fueron OpenMP para poder ejecutar el algoritmo sobre un ambiente de memoria compartida utilizando un equipo con 8 cores; MPI para poder ejecutar el algoritmo sobre un ambiente de memoria distribuida tipo clúster con 8, 16, y 32 cores; y CUDA para poder ejecutar el algoritmo sobre una GPU.

Se realizó un análisis de los tiempos de ejecución resultantes de las distintas implementaciones para distintas entradas de datos entre 1Kbyte y 255Mbytes, observando la eficiencia de la implementación del algoritmo implementado en CUDA.

En todos los casos se logra una aceleración del tiempo de cómputo importante con respecto al secuencial, pero la aceleración lograda con CUDA sobre la GPU es considerablemente mayor no solo con respecto a la implementación secuencial, sino también a las demás implementaciones paralelas.

Hoy en día existen limitaciones en la velocidad de los procesadores debido a problemas térmicos y de consumo, pero este trabajo mostro que utilizando herramientas de programación paralela se puede aprovechar el paralelismo que proveen las arquitecturas multicores actuales, y de esta forma, poder acelerar el computo de los algoritmos, en particular AES, un algoritmo de cifrado estándar muy utilizado.

Se comprobó que se logra una buena aceleración con las implementaciones paralelas al cifrar distintos tamaños de datos, en particular del algoritmo implementado con CUDA sobre una GPU, de esta forma es posible reducir el costo de cifrado de datos ya sea para almacenarlos o para hacer una transferencia importante de datos sensibles sobre una red pública.

6.2 Trabajo a futuro

Existe en la actualidad una librería de criptografía de propósito general llamada OpenSSL, 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, no solo con tamaño de clave de 128bits sino también con tamaño de clave de 256bits, además de otros algoritmos de cifrado tanto simétricos como asimétricos, estos últimos tienen ventajas con respecto a los simétricos en cuanto a no tener comprometida la clave, pero además son más costosos computacionalmente por lo que tendría sentido aplicar técnicas de paralelizacion para acelerar el computo, la desventaja está en que incrementan el volumen de información generada.

Adaptar los algoritmos OpenSSL no es algo trivial y lleva un análisis del código más profundo, en particular para llevarlos a ejecutar sobre una GPU por el formato que debe tener el código CUDA, donde hay que indicar que es lo que se ejecuta sobre la CPU y que sobre la GPU.

Otras líneas futuras incluyen el estudio de la eficiencia energética, de manera de comparar algoritmos criptográficos en función no solo de su velocidad de cifrado sino del consumo energético en las distintas arquitecturas multicores.

Bibliografía básica:

1. Adrian Pousa, Victoria Sanz, Armando De Giusti, “Análisis de rendimiento de un algoritmo de criptografía simétrica sobre arquitecturas multicore”, Proceedings del XVII Congreso Argentino de Ciencias de la Computación – CACIC 2011.
2. Julián Ortega,, Helmuth Trefftz (Universidad EAFIT, Medellín, Colombia), and Christian Trefftz, (Grand Valley State University, Allendale, Michigan, USA), “Parallelizing AES on multicores and GPUS”, http://www1.eafit.edu.co/rvirtual/Publications/PID1774591.pdf
3. 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.
4. Suresh Siddha, Venkatesh Pallipadi, Asit Mallick. “Process Scheduling Challenges in the Era of Multicore Processors”Intel Technology Journal, Vol. 11, Issue 04, November 2007.
5. Grama A., Gupta A., Karypis G., Kumar V. “Introduction to Parallel Computing”. Second Edition. Addison Wesley, 2003.
6. 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.
7. The OpenMP API specification for parallel programming. http://openmp.org/wp/.
8. MPI Specification http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf.
9. General-Purpose Computation on Graphics Hardware http://gpgpu.org/.
10. FIPS PUB 197: the official AES Standard http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
11. 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.
12. D.J. Bernstein - Cache-timing attacks on AES (2005) http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.
13. 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.
14. A Diagonal Fault Attack on the Advanced Encryption Standard http://eprint.iacr.org/2009/581.pdf.
15. T. Rauber, G. Rünger. Parallel Programming: For Multicore and Cluster Systems. ISBN 364204817X, 9783642048173. Springer, 2010.
16. Buck I. “Gpu computing with nvidia cuda”. ACM SIGGRAPH 2007 courses ACM, 2007. New York, NY, USA.
17. Cuda Home Page http://www.nvidia.com/object/cuda_home_new.html.
18. Cuda best practices guide http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf.
19. Intel Product Specifications http://ark.intel.com/products/33079/Intel-Xeon-Processor-E5405-(12M-Cache-2_00-GHz-1333-MHz-FSB).
20. Nvidia Geforce GTX 560TI Specifications http://www.nvidia.com/object/product-geforce-gtx-560ti-us.html.
21. Computer Architecture: Challenges and Opportunities For The Next Decade - Tilak Agerwala Siddhartha Chatterjee IBM Research 2004. Published by the IEEE Computer Society.