Tiempo disponible: 15 min
Tiempo target: 12 min 30 seg
Tiempo mínimo: 10 min

1 minuto

0 – PORTADA
Buenos días, mi nombre es Jesús y voy a presentar mi trabajo sobre una implementación de alto rendimiento de un cifrador moderno y ampliamente usado como es el AES:

1 – MOTIVACIONES
- La motivación principal es dar solución a los cuello de botella en sistemas especializados por el que el cifrador no puede responder adecuadamente al alto ancho de banda que le llega.

- Otra motivación es que el usuario medio hoy en dia tiene acceso fácil a GPUs potentes, se podrían aprovechar para tratar ficheros grandes. Se evitaría así de saturar la CPU, dejarla libre para otras tareas y además terminar la operación incluso en menos tiempo.

1 minuto

2 – OBJETIVOS
- El objetivo ha sido implementar desde cero distintas versiones GPU del cifrado AES para ver que aspectos afectan mas al rendimiento. De esta forma, se ha querido descubrir si sería viable una implementación GPU en lugar de una CPU en sistemas especializados.

- En particular, nosotros lo teníamos pensado utilizar junto con un sistema de distribución de claves cuánticas de alto rendimiento.

- También decir, El único código que he reutilizado ha sido para el algoritmo de expansión de clave que es estrictamente secuencial y no ha podido paralelizarse y los valores de unas tablas que explicaré más adelante.

1 minuto

3.1 – CIFRADO EN GPU (1)
- Como en cifradores de bloque, como el AES, el mensaje se trocea en partes que pueden ser tratadas independientemente. La idea básica es que la GPU puede dar cabida a muchos mas hilos que la CPU, se podría obtener un mayor rendimiento al cifrar cada bloque con un hilo.

3.1 - CIFRADO EN GPU (2)
- Aunque hay que decir que esto solo funciona cuando estamos trabajando con grandes conjuntos de datos, con pocos datos es muy difícil aprovechar el hardware y el delay de enviar y recibir los bloques puede ser mayor que calcularlo en la CPU.

1 minuto

3.2 - MODOS DE OPERACIÓN (1)

- Y también tenemos que tener en cuenta que si ciframos tal como hemos explicado, modo ECB que es el más básico, tendríamos problemas de seguridad porque al cifrar varios bloques de entrada iguales vamos a obtener bloques de salida iguales por lo que es posible detectar patrones.

3.2 - MODOS DE OPERACIÓN (2)

- El problema es que no todos los modos de operación son paralelizables, el CBC por ejemplo, muy usado por que fue el primero que solucionó el problema del ECB para cifrar cada bloque necesita haber cifrado previamente el anterior. Lo bueno es que el descifrado si es posible paralelizarlo.

3.2 - MODOS DE OPERACIÓN (3)

- Como es tan popular se ha implementado el modo CBC en descifrado, también el CFB que era muy similar al CBC, y el modo contador que si que soporta paralelización tanto como para cifrado como para descifrado.

1 minuto

4 – IMPLEMENTACIÓN

A continuación voy a presentar **algunos** de los distintos aspectos tenidos en cuenta en la implementación como la organización de los hilos, en nivel de paralelismo o las transferencias host-GPU.

4.1 – ORGANIZACIÓN DE LOS THREADS

Los hilos se organizan para maximizar la ocupancia, es decir para que se utilicen todos los hilos y bloques de hilos que soporte cada multiprocesador en los que dividida de la GPU. Por ejemplo esta seria una división valida para una GPU que utiliza el máximo de 4 hilos y 2 bloques de hilos por multiprocesador

1 minuto

4.2 – NIVEL DE PARALELISMO (1)

Además se han realizado implementaciones adicionales en la que dos, cuatro o dieciséis hilos colaboran para cifrar un mismo bloque de datos de dieciséis bytes. Hay un inconveniente, y es que hace falta introducir mecanismos de sincronización y el versión de dieciséis threads en la que cada thread se encarga de un único byte no se aprovechan los registros de 32 bits de la GPU.

4.2 – NIVEL DE PARALELISMO (2 y 3)

Por ello, esperábamos encontrar mas diferencias de rendimiento que las que hemos encontrado… Aunque la versión de 1 thread por bloque de datos si que va mas rápido al no necesitar de mecanismos de sincronización posiblemente haya un cuello de botella en el disco que impida ver mayores diferencias conforme el tamaño de fichero aumenta

1 minuto

4.3 – TABLAS DE BÚSQUEDA (1)

Bueno, ahora… ¿Cual es el trabajo que hace cada thread? Pues tal como se recomienda en el propio estándar del AES para procesadores de 32 bits en lugar de realizar directamente las operaciones de permutación y sustitución de bytes del algoritmo, el bloque de datos se actualiza combinando unos valores precomputados en unas tablas. Esto reduce significativamente el numero de operaciones necesarias, esto es especialmente importante en la GPU ya que las operaciones con enteros son caras por que para lo que La GPU esta realmente preparada es para hacer operaciones en coma flotante.

4.3 – TABLAS DE BÚSQUEDA (2)

Hay distintas opciones para almacenar las tablas y clave expandida en la GPU: memoria global, memoria constante, y memoria compartida. Hemos probado las tres versiones…

4.3 – TABLAS DE BÚSQUEDA (3 y 4)

...Aunque como se puede ver en las siguientes gráficas las diferencias tampoco han sido significativas…

1 minuto

4.4 – TRANSFERENCIAS HOST-GPU (1)

La implementación se ha realizado de tal forma que las transferencias los bloques de entrada con los que la GPU va a trabajar y las transferencias vuelta los resultados se solapen con el computo. Para ello se hace uso de las colas independientes que tiene la GPU para trocear el problema y lanzar varias funciones de cifrado en paralelo.

4.4 – TRANSFERENCIAS HOST-GPU (2)

De esta forma se ha podido obtener una mejora de hasta el 7% como se puede ver en la siguiente tabla.

4.4 – TRANSFERENCIAS HOST-GPU (3)

Esta es una captura del profiler en la que se puede ver que con 4 colas [señalar] se consigue que las transferencias se (señaladas en marrón) [señalar] se solapen con el computo (en azul) [señalar]

1 minuto

4.4 – TRANSFERENCIAS HOST-GPU (4)

Las operaciones de lectura y escritura se realizan de manera asíncrona: Se utilizan dos threads adicionales – uno para leer trozos del fichero de entrada introduciendolos en una en cola para ir que el hilo principal pueda ir lanzando funciones de cifrado en paralelo y otro para escribir los trozos que ya han sido procesados en un fichero de salida.

Y abajo se puede ver una captura del profiler en la que las operaciones de entrada y salida están marcadas en verde.

4.4 – TRANSFERENCIAS HOST-GPU (5)

El buffer donde se guardan los trozos de fichero leídos se ubica directamente en RAM física, ahorrándonos el coste paginación y permitiendo mayor ancho de banda

1 minuto

4.4 – TRANSFERENCIAS HOST-GPU (7)

El tamaño de este buffer también tiene un impacto en el rendimiento. Tanto uno muy pequeño como uno muy grande tienen un efecto negativo. Como se puede ver en la tabla, se han obtenido los mejores resultados reservando 8MB.

4.5 – ACCESO ALEATORIO

En la implementación se da soporte a acceso aleatorio, que permite dado un fichero cifrado obtener parte del mensaje en claro sin necesidad de descifrar por completo el mismo. Es especialmente util en ficheros muy grandes.

En el ejemplo de la derecha se puede ver como solo es necesario procesar los bloques dentro de los cuales cae el rango deseado que queremos descifrar. En este caso de los cuatro bloques del fichero de entrada solo es necesario procesar el segundo y tercero.

1 minuto

5 – COMPARACIÓN CON OPENSSL (1)

Finalmente, aquí podéis ver las diferencias de rendimiento de la implementación GPU en contraste la CPU de OpenSSL, ampliamente usada.

Como esperábamos el rendimiento de la GPU es pobre con ficheros pequeños, pero a partir de ficheros de 63 MB se pueden obtener beneficios del 55% de media para el cifrado.

5 – COMPARACIÓN CON OPENSSL (2)

Para el descifrado, los beneficios son ligeramente inferiores. En este caso a partir de ficheros de 125MB se obtiene una mejora del 50% de media.

1 minuto

Como conclusión decir que a pesar de que se han obtenido beneficios con respecto a una versión CPU, los beneficios han sido bastante bajos comparado con lo que debería ofrecer una versión GPU.

Esto puede ser por un cuello de botella en el disco de la máquina en la que se han realizado las pruebas porque el disco tienen una velocidad de lectura de secuencial de 208 MB/s mientras que la capacidad del bus de la GPU es de 8GB/s por lo que se esta desaprovechando un gran potencial.

Entonces es necesario verificar que realmente se pueden obtener mejores resultados al aumentar las prestaciones de disco antes de descartar soluciones más baratas de aceleración hardware AES-NI que incorporan algunas CPUs y que permiten obtener beneficios similares.

30 seg

Finalmente decir que el código desarrollado se puede compilar en una herramienta para la linea de comandos o como una librería compartida.

Se ha verificado que la implementación es correcta con una suite de test unitarios en la que se han contrastado con los vectores de prueba del estándar como con el cifrado y descifrado de mensajes aleatorios en los que se ha comprobado que se obtiene la misma entrada original.

Eso es todo, gracias.