GPU ACCELERATED AES

IMPLEMENTACIÓN EN GPU DEL CIFRADOR AES

ETSIINF DE LA UNIVERSIDAD POLITÉCNICA DE MADRID

RESEARCH CENTER FOR COMPUTATIONAL SIMULATION

RESEARCH GROUP ON QUANTUM INFORMATION AND COMPUTATION

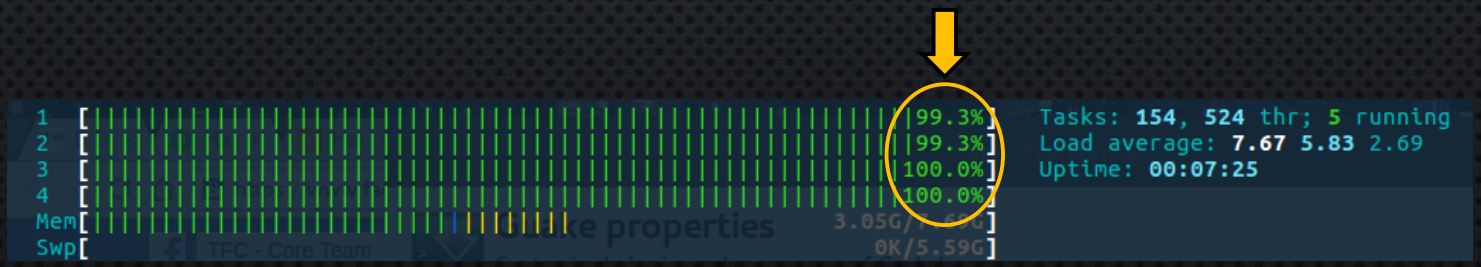
AUTOR: JESÚS MARTÍN BERLANGA

TUTOR: JESÚS MARTÍNEZ MATEO

1. MOTIVACIONES

¿Por qué usar una GPU para cifrar/descrifrar?

- Uso en Sistemas Especializados
  - Cuellos de botella en el cifrador
  - Necesidad de nuevas soluciones que puedan sostener altas transferencias de datos
- Además, uso en computación "doméstica":
  - Aprovechar la GPUs potentes a cada vez precios mas asequibles para el usuario medio para
    - Evitar saturar la CPU al cifrar grandes ficheros
    - Poder seguir usando la CPU para otras tareas
    - Operación en menor tiempo



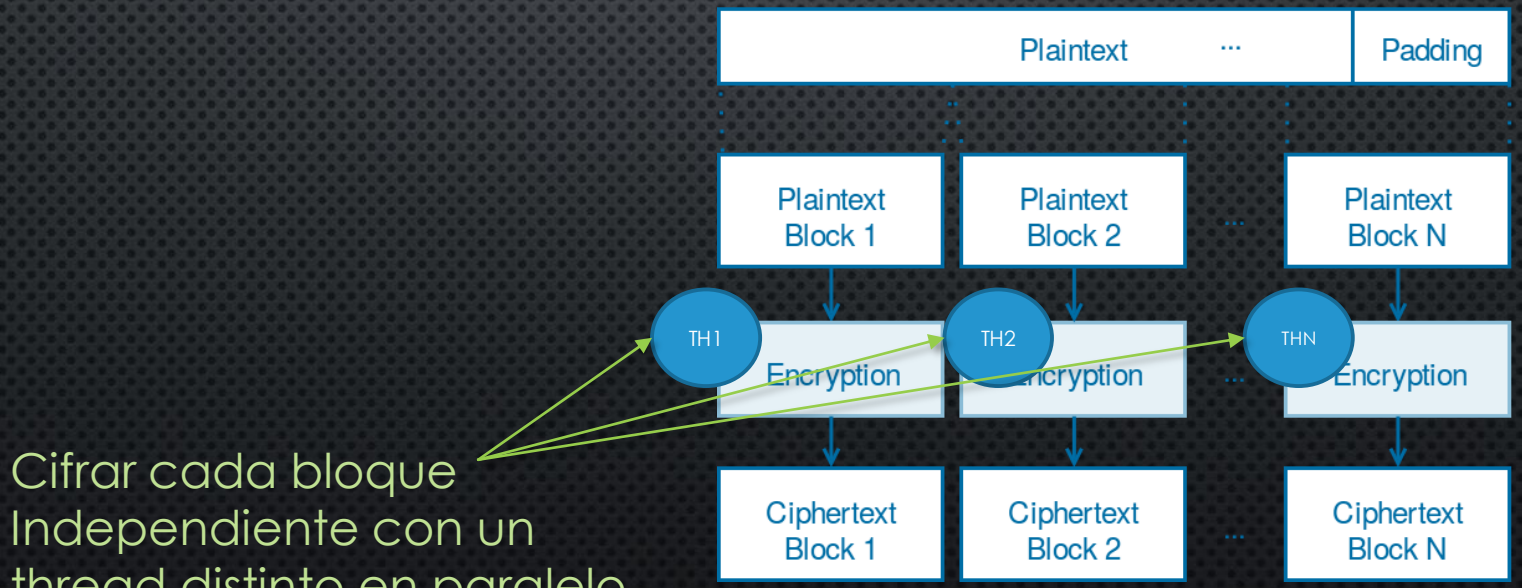
2. OBJETIVOS

- Realizar implementación GPU (CUDA) del cifrador AES y revisando distintos factores de la implementación que pueden afectar al rendimiento
    - La implementación se ha desarrollado partiendo de cero
        - Solo se ha reutilizado el código para el algoritmo de expansión de clave, que es estrictamente secuencial, y los valores de unas tablas, de la librería OpenSSL
- Descubrir limitaciones de la versión GPU en contraste con una CPU para explorar su viabilidad en Sistemas Especializados
    - Pensado usarse con distribución de claves cuánticas de alto rendimiento

3.1 FUNDAMENTOS

Cifrado en GPU

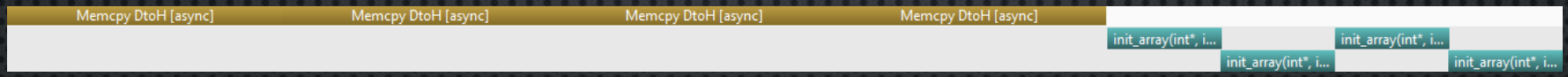
- Como cada bloque se puede cifrar de forma independiente a los otros, la idea es que al cifrarlos con 2304 Cores de la GPU en lugar de solo 4 de la CPU podremos obtener un gran beneficio.



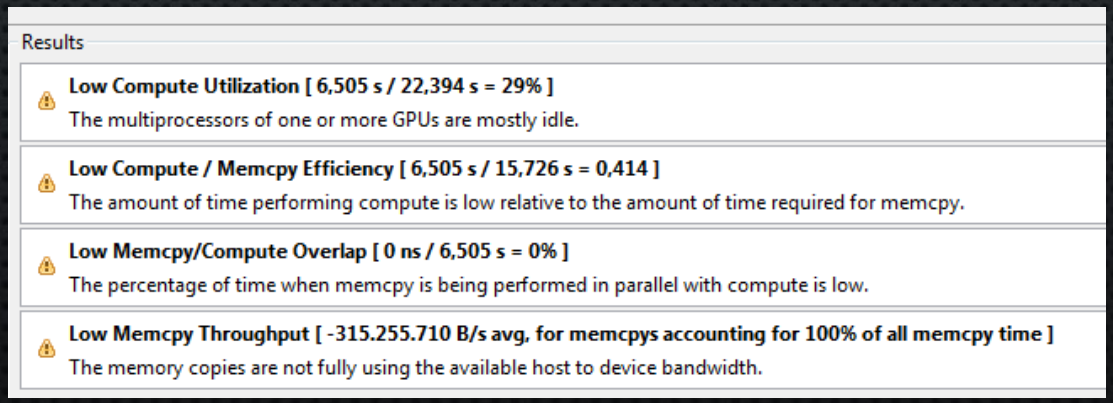
3.1 FUNDAMENTOS

Cifrado en GPU

- A tener en cuenta:
  - Un Core en la GPU es menos potente y mucho mas simple que un Core de la CPU
  - Las transferencias de memoria son caras - con pocos datos el tiempo de transferencia puede ser mayor que el tiempo de computo en la CPU



  - Se necesitan muchos datos (ficheros grandes) para aprovechar el hardware GPU y notar beneficios

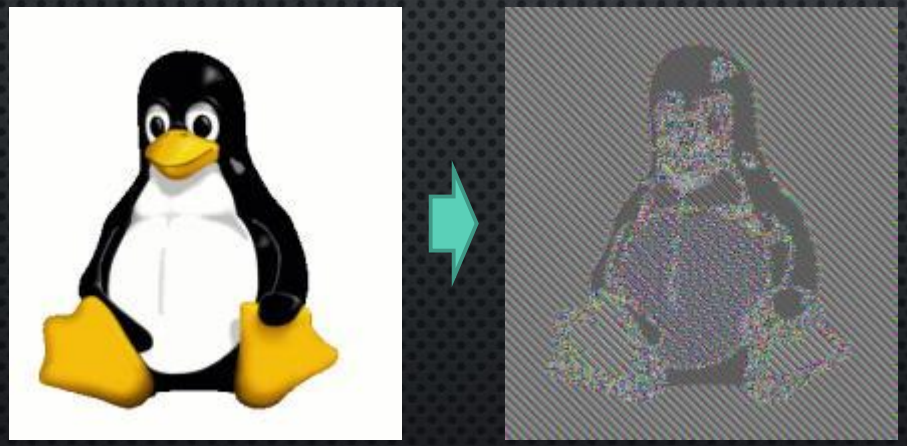


3.2 FUNDAMENTOS

Modos de operación

- Modos de operación necesarios para un cifrado seguro

Si ciframos tal como hemos explicado (modo ECB):

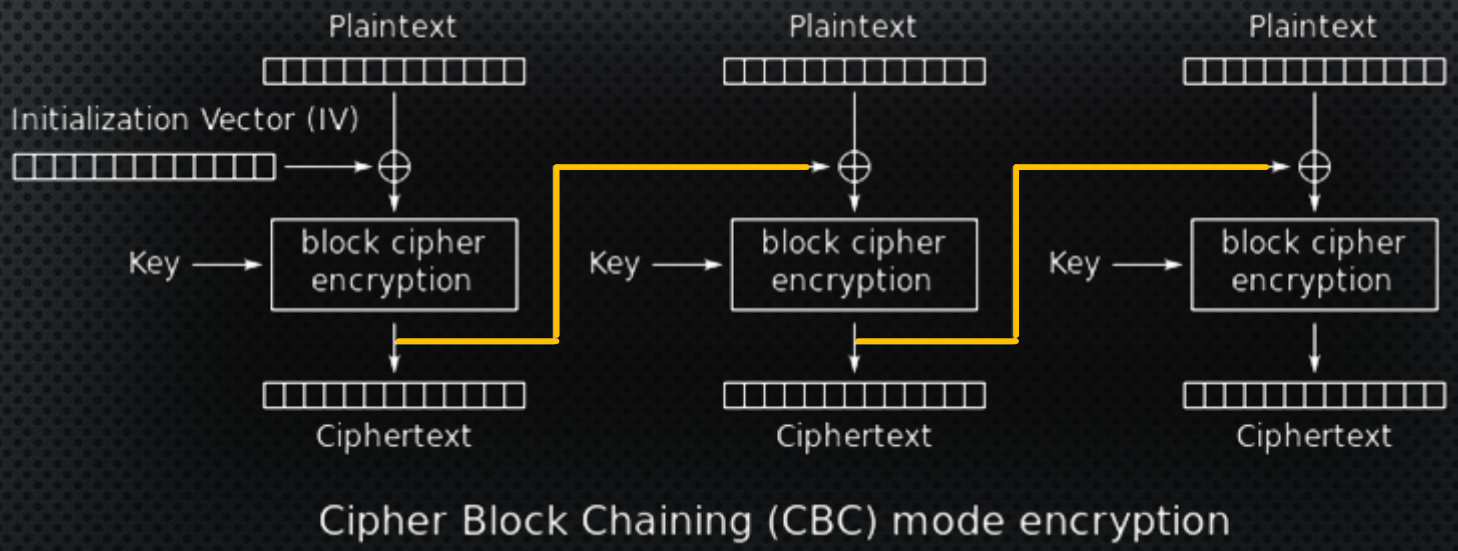


Con otros modos de operación:

3.2 FUNDAMENTOS

Modos de operación

Problema: No todos los modos de operación son paralelizables



Cipher Block Chaining (CBC) mode encryption

Dependencias entre bloques: para cifrar el siguiente
Necesitamos haber cifrado el anterior

3.5 MODOS DE OPERACIÓN

A parte del modo básico ECB no seguro se ha implementado el modo contador (CTR), y los modos CBC y CFB solo para descifrado.

Modo	Cifrado paralelizable	Descifrado paralelizable
ECB	SI	SI
CBC	NO	**SI**
PCBC	NO	NO
CFB	NO	**SI**
OFB	NO	NO
CTR	**SI**	**SI**

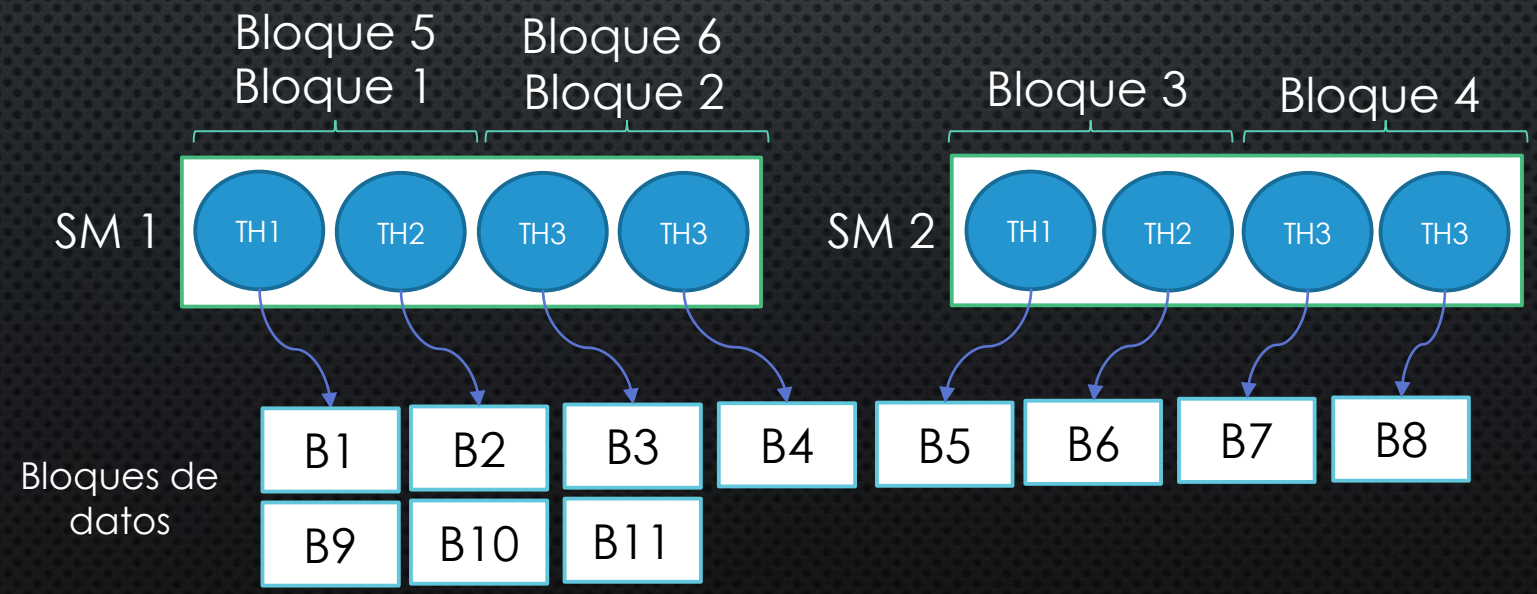
4. IMPLEMENTACIÓN

4.1 Organización de los hilos
4.2 Nivel de paralelismo
4.3 Tablas de búsqueda
4.5 Transferencias host-GPU
4.6 Acceso aleatorio

4.1 ORGANIZACIÓN DE LOS THREADS

- Se establece un tamaño de bloque (de threads) que maximize la ocupancia, es decir, que todos los threads y bloques de trheads disponibles por multiprocesador se utilizen
- Se crean suficientes bloques de *threads* para procesar todos los bloques de datos que se quieren cifrar/descrifrar

- Ejemplo GPU con 2 multiprocesadores (SM):
  máximo de 4 threads y 2 bloques de threads por SM

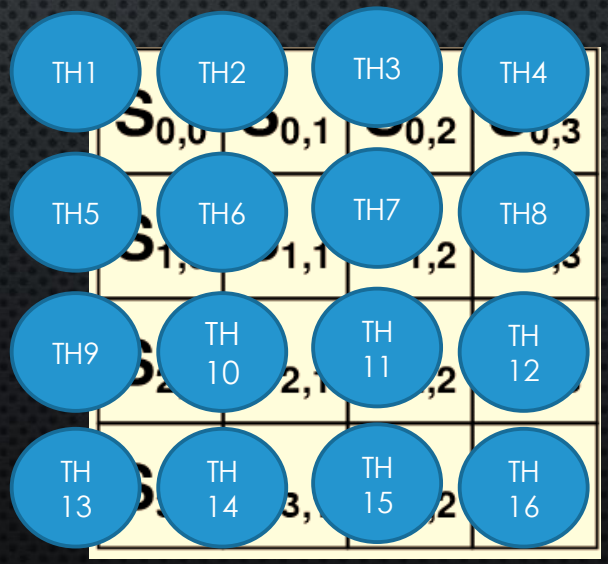


4.2 NIVEL DE PARALELISMO

- Implementaciones de dos, cuatro, y dieciséis hilos por bloque de datos
- Al tener que trabajar varios hilos con el mismo conjunto de datos es necesario incluir mecanismos de sincronización
  → más problemático
- Con 16 threads no se aprovechan los registros de 32 bits de la GPU

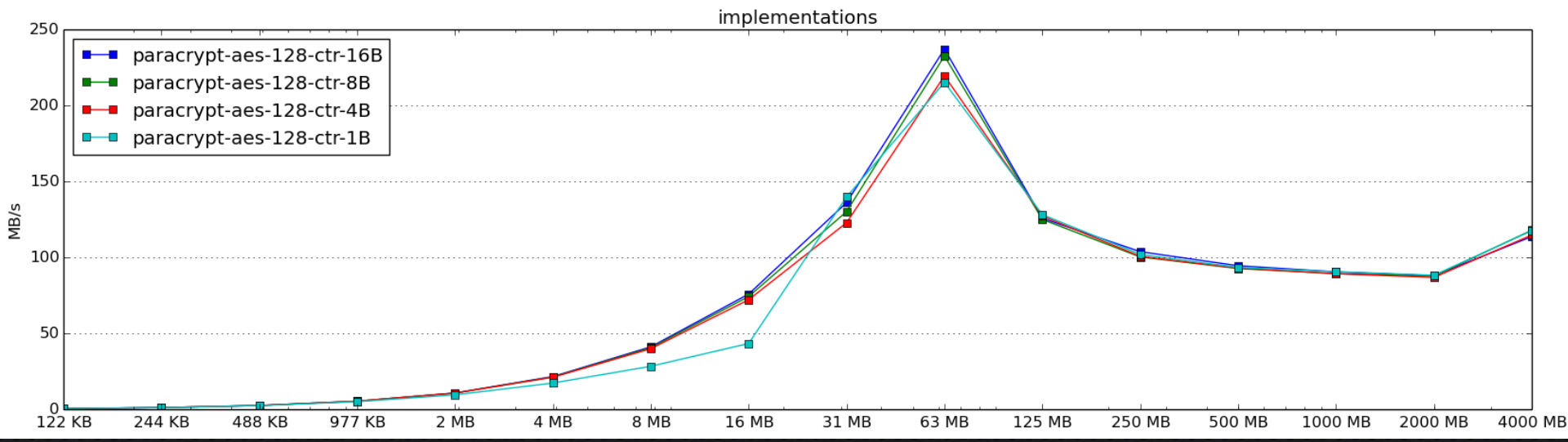
Ver. **4 hilos**

Ver. **16 hilos**



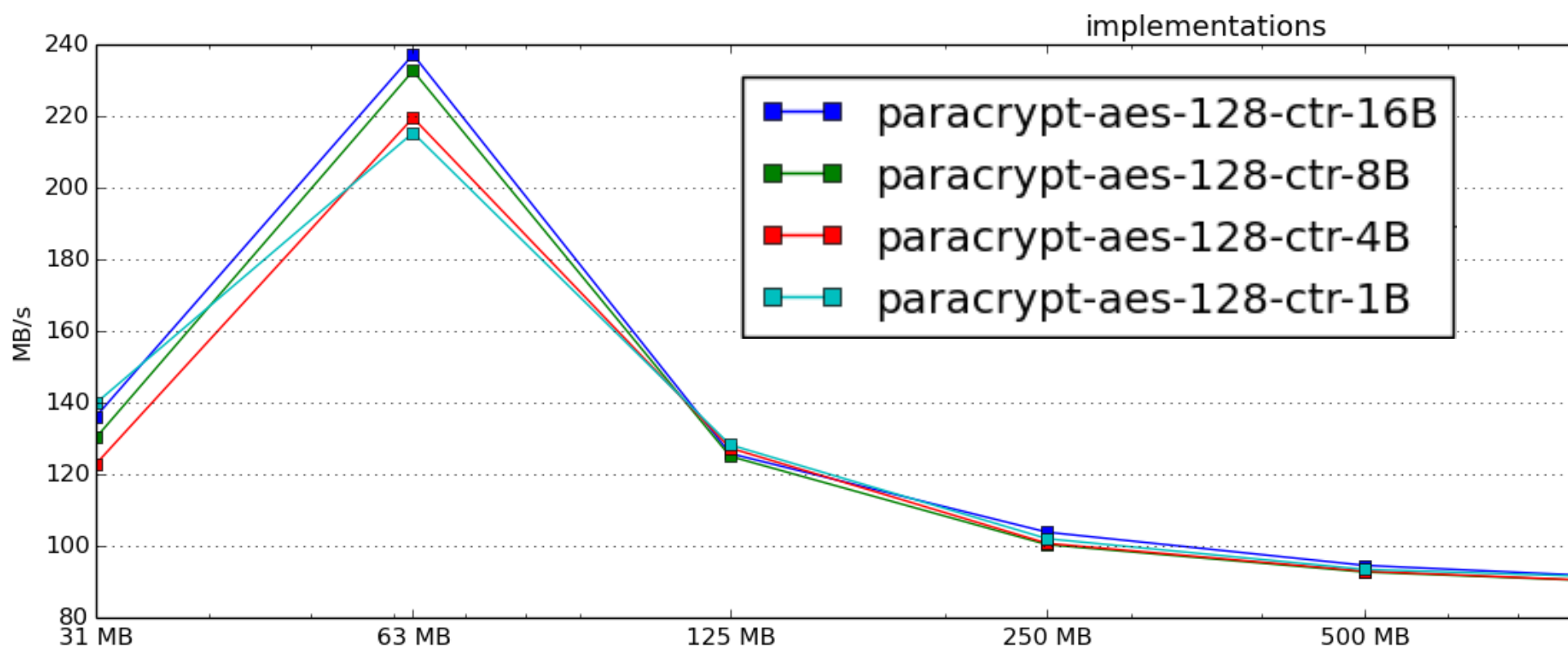
4.2 NIVEL DE PARALELISMO

- Se esperaban mayores diferencias resultados
- Es posible que un cuello de botella en el disco no haya permitido obtener mayores diferencias conforme el tamaño de fichero aumenta



4.2 NIVEL DE PARALELISMO

- Se esperaban mayores diferencias resultados
- Es posible que un cuello de botella en el disco no haya permitido obtener mayores diferencias conforme el tamaño de fichero aumenta



4.3 TABLAS DE BÚSQUEDA

Ya sabemos como dividir el trabajo en bloques de *threads* pero, ¿Qué realiza exactamente cada *thread* con los bloques de datos que le toca procesar? Se repiten una serie de rondas en las que:

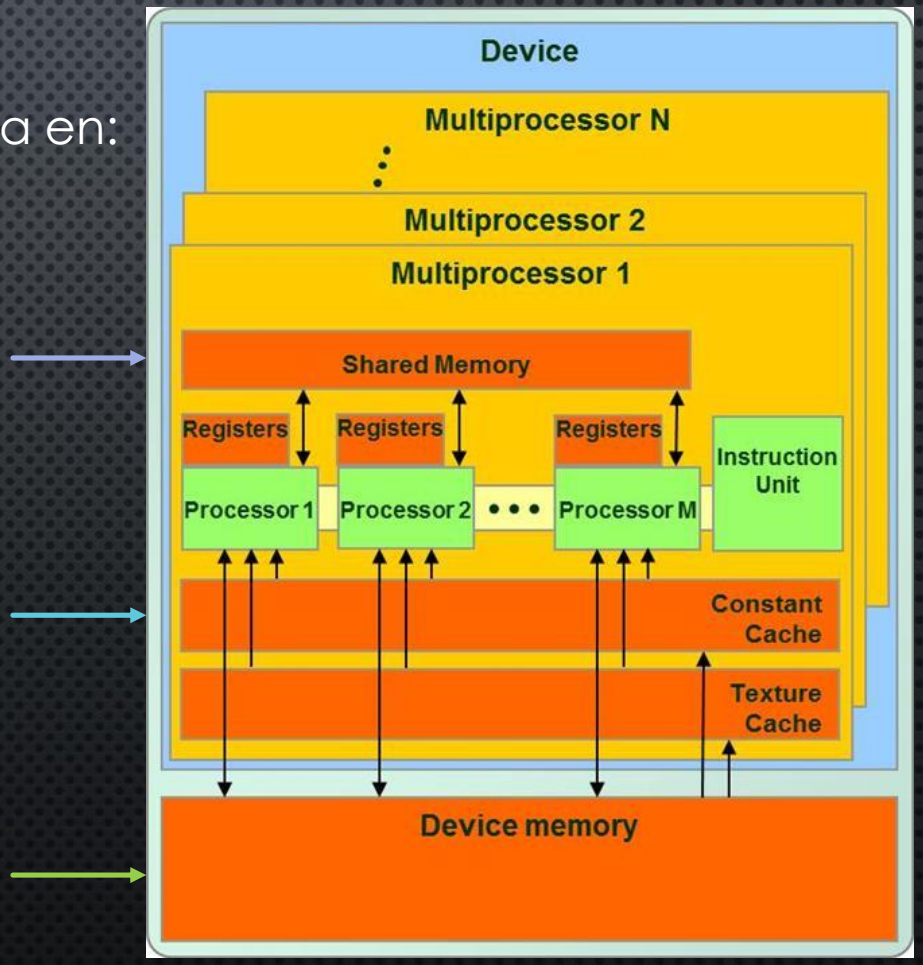
- En lugar de aplicar operaciones de permutación y sustitución de bytes actualizamos el bloque de datos usando los valores pre-computados en unas tablas reduciendo el numero de operaciones necesarias y aumentando el rendimiento

$$\begin{matrix} S_0 & S_1 & S_2 & S_3 \end{matrix}$$
$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$S'_0 = T_0[a_{00}] \oplus T_1[a_{11}] \oplus T_2[a_{22}] \oplus T_3[a_{33}] \oplus k_0$$
$$S'_1 = T_0[a_{01}] \oplus T_1[a_{12}] \oplus T_2[a_{23}] \oplus T_3[a_{30}] \oplus k_1$$
$$S'_2 = T_0[a_{02}] \oplus T_1[a_{13}] \oplus T_2[a_{20}] \oplus T_3[a_{31}] \oplus k_2$$
$$S'_3 = T_0[a_{03}] \oplus T_1[a_{10}] \oplus T_2[a_{21}] \oplus T_3[a_{32}] \oplus k_3$$

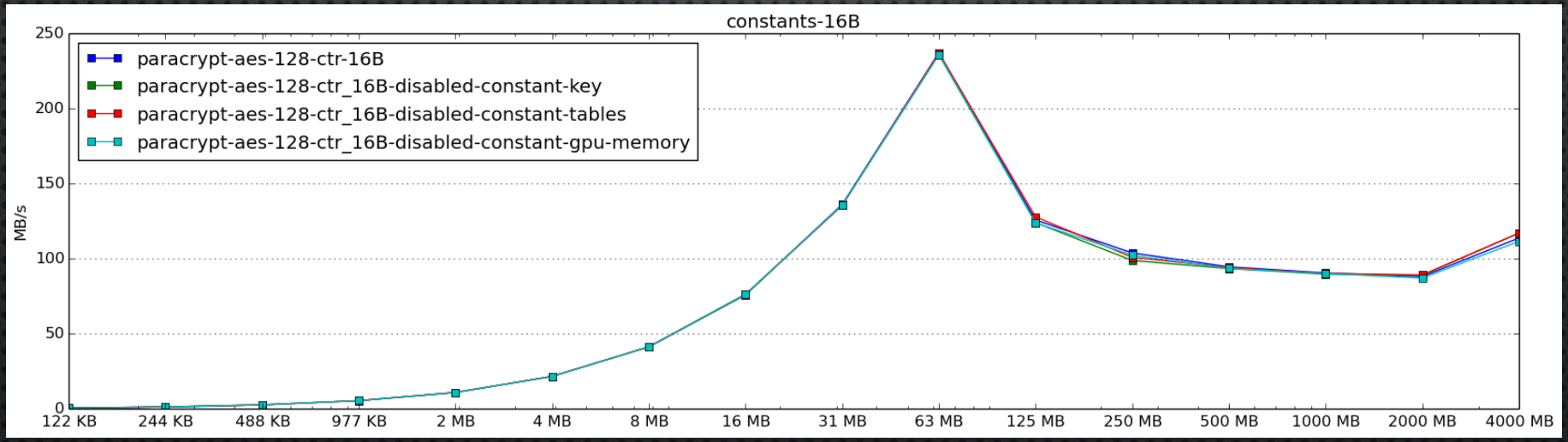
4.3 TABLAS DE BÚSQUEDA

- Almacenamiento de tablas de búsqueda (8KB) y clave expandida en:
  - ó memoria global
  - ó memoria constante
  - ó memoria compartida



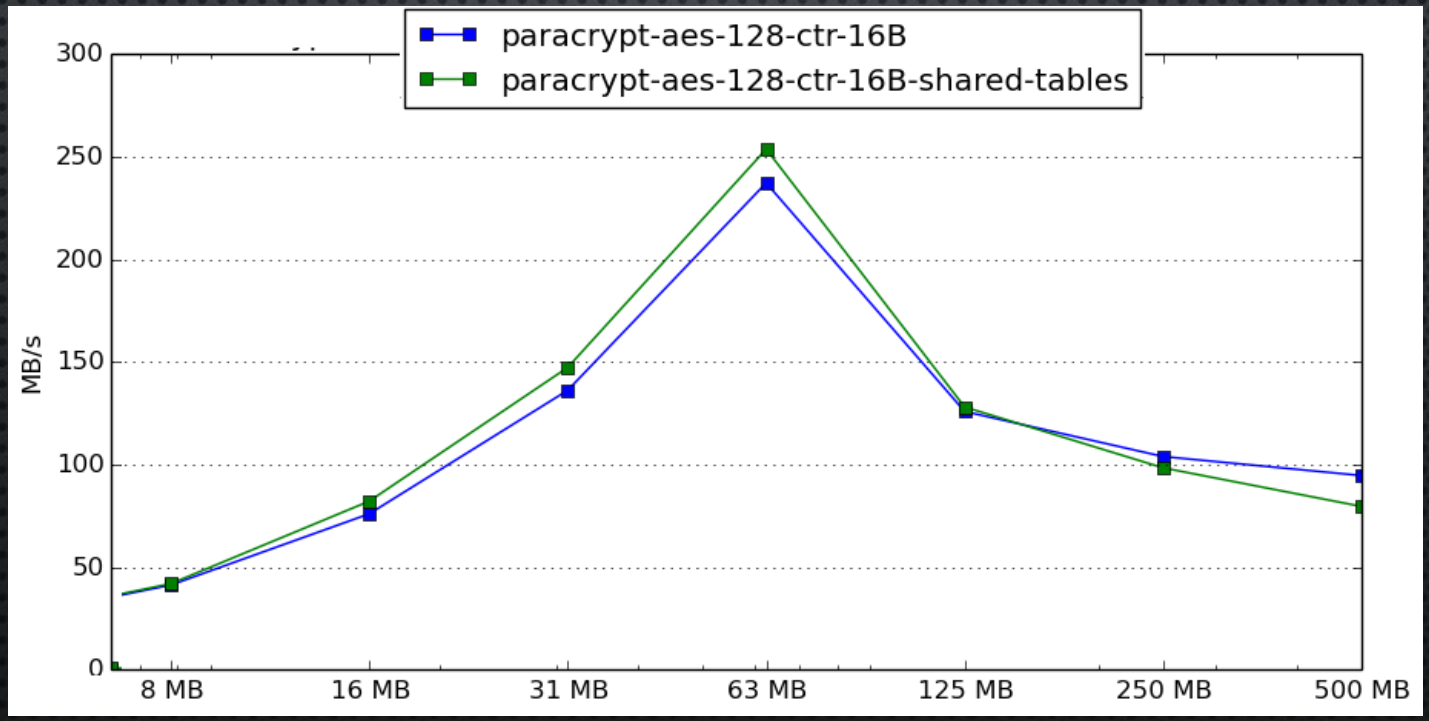
4.3 TABLAS DE BÚSQUEDA

memoria global vs constante



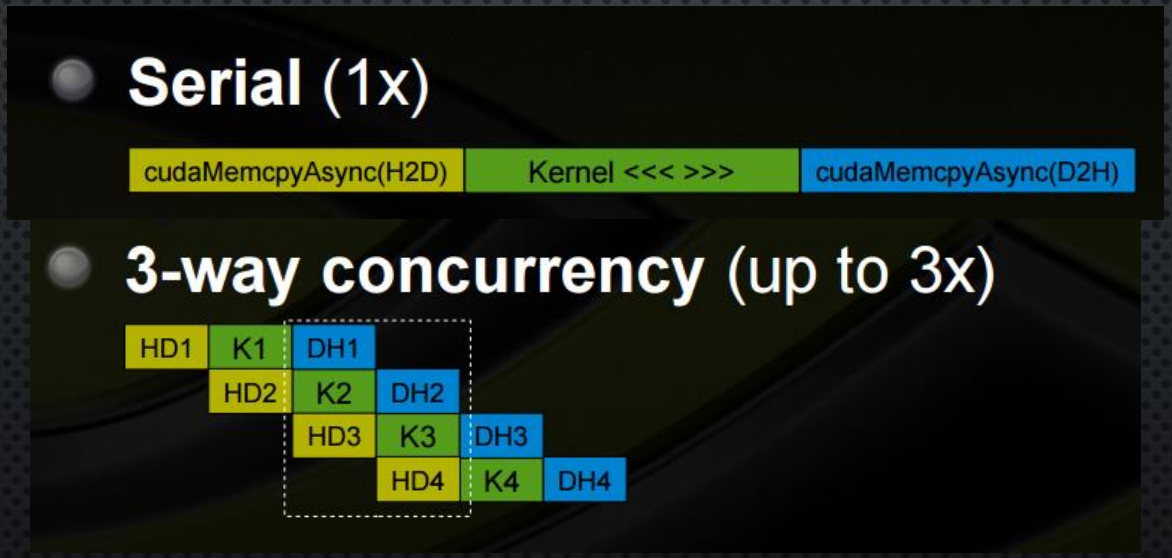
4.3 TABLAS DE BÚSQUEDA

memoria global vs compartida para las tablas



4.4 TRANSFERENCIAS HOST-GPU

- Se ha tenido en cuenta solapamiento de computo y transferencias de datos a la GPU lanzando varios *kernels* (funciones que se ejecutan en la GPU) asíncronos en varios *streams (colas de operaciones independientes)*



4.4 TRANSFERENCIAS HOST-GPU

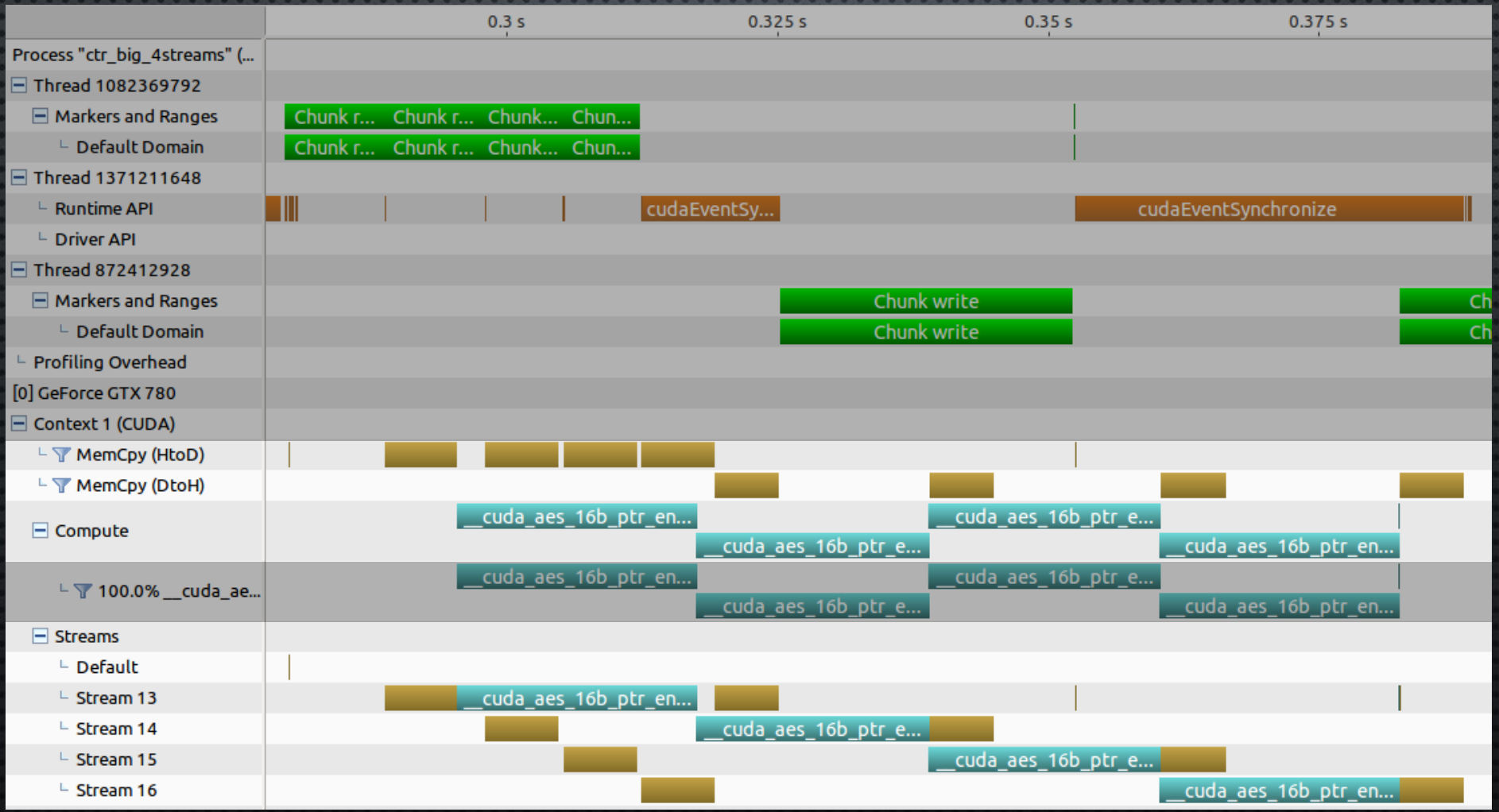
- Solapamiento de computo y transferencias mejora el rendimiento hasta un 7.32%

Media MB/s ficheros entre 16MB y 4GB

			∇ 1-stream
1	stream	111.18	0 %
2	streams	117.10	5.32 %
4	streams	117.79	5.94 %
8	streams	119.07	7.09 %
16	streams	119.33	7.32 %

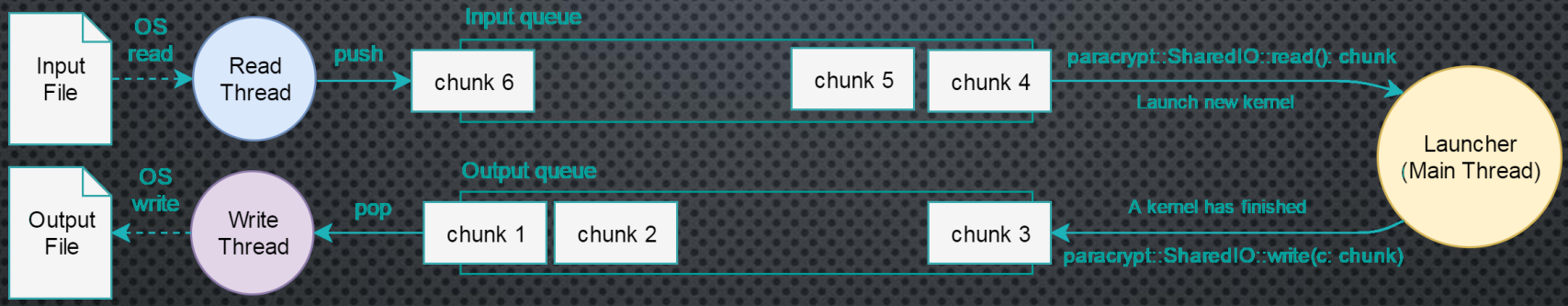
4.4 TRANSFERENCIAS HOST-GPU

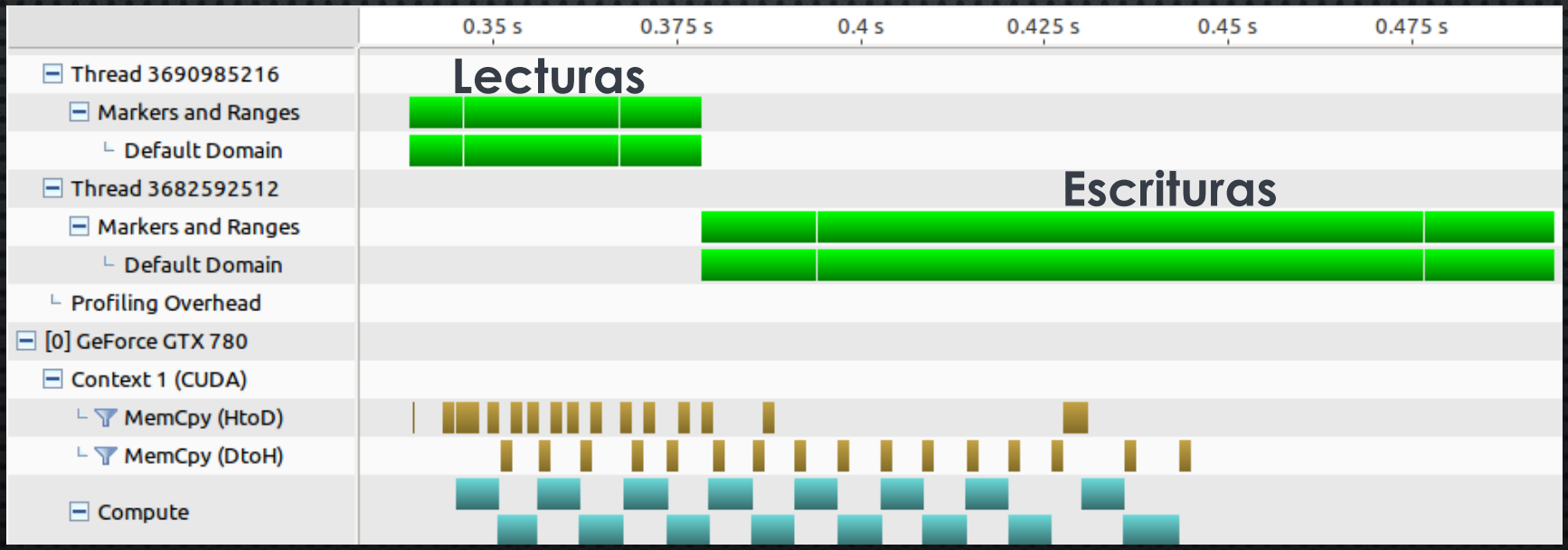
Solapamiento con 4 streams



4.4 TRANSFERENCIAS HOST-GPU

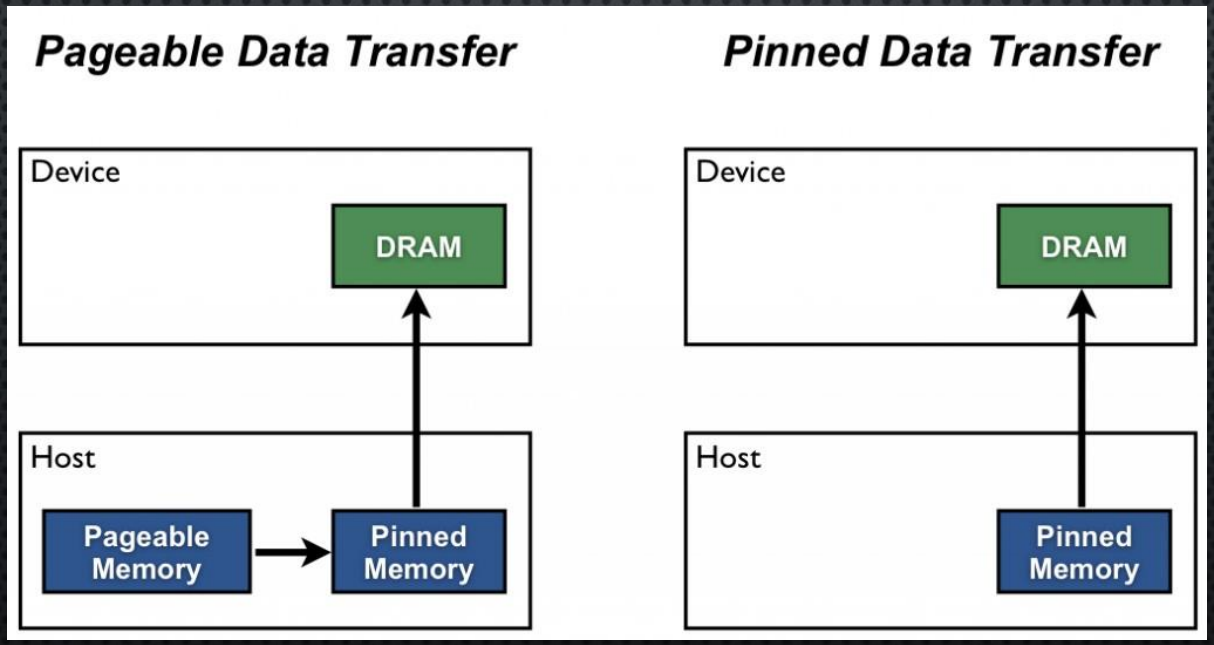
Transferencias host-GPU: Lecturas asíncronas del fichero de entrada





4.4 TRANSFERENCIAS HOST-GPU

- El buffer de lectura del fichero se almacena en memoria no paginable permitiendo un mayor ancho de banda en las transferencias host-GPU



4.4 TRANSFERENCIAS HOST-GPU

- Un tamaño demasiado pequeño de buffer afecta negativamente al rendimiento al realizar demasiadas operaciones de copia host-GPU
- Un tamaño demasiado grande también afecta negativamente (coste de malloc sobrepasa beneficios)
- La implementación usa un buffer de 8MB por defecto

MB/s vs Tamaño fichero cifrado

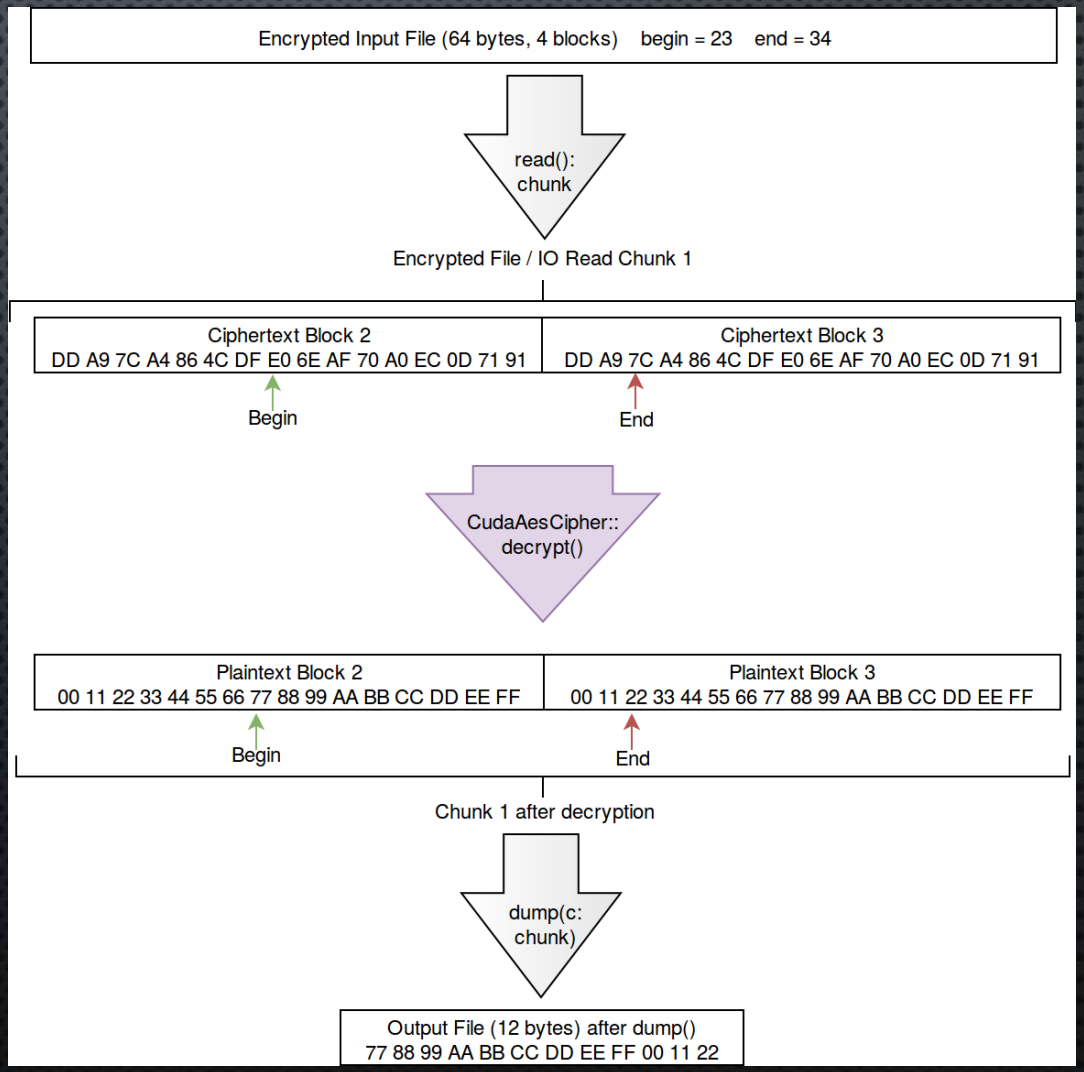
	4 GB	2 GB	1 GB	500 MB	250 MB	125 MB	63 MB	media
1MB	113.34	85.87	85.67	91.19	97.45	130.37	200.12	114.85
2MB	116.38	86.32	88.89	90.58	94.11	116.39	221.26	116.27
8MB	118.28	88.12	90.65	93.84	104.55	124.91	240.05	**122.91**
32MB	117.28	88.24	90.02	93.68	102.07	125.2	234.84	121.61
sin límite*	115.87	86.19	88.83	93.45	103.65	126.79	219.67	119.20

*max. ram disponible

4.5 ACCESO ALEATORIO

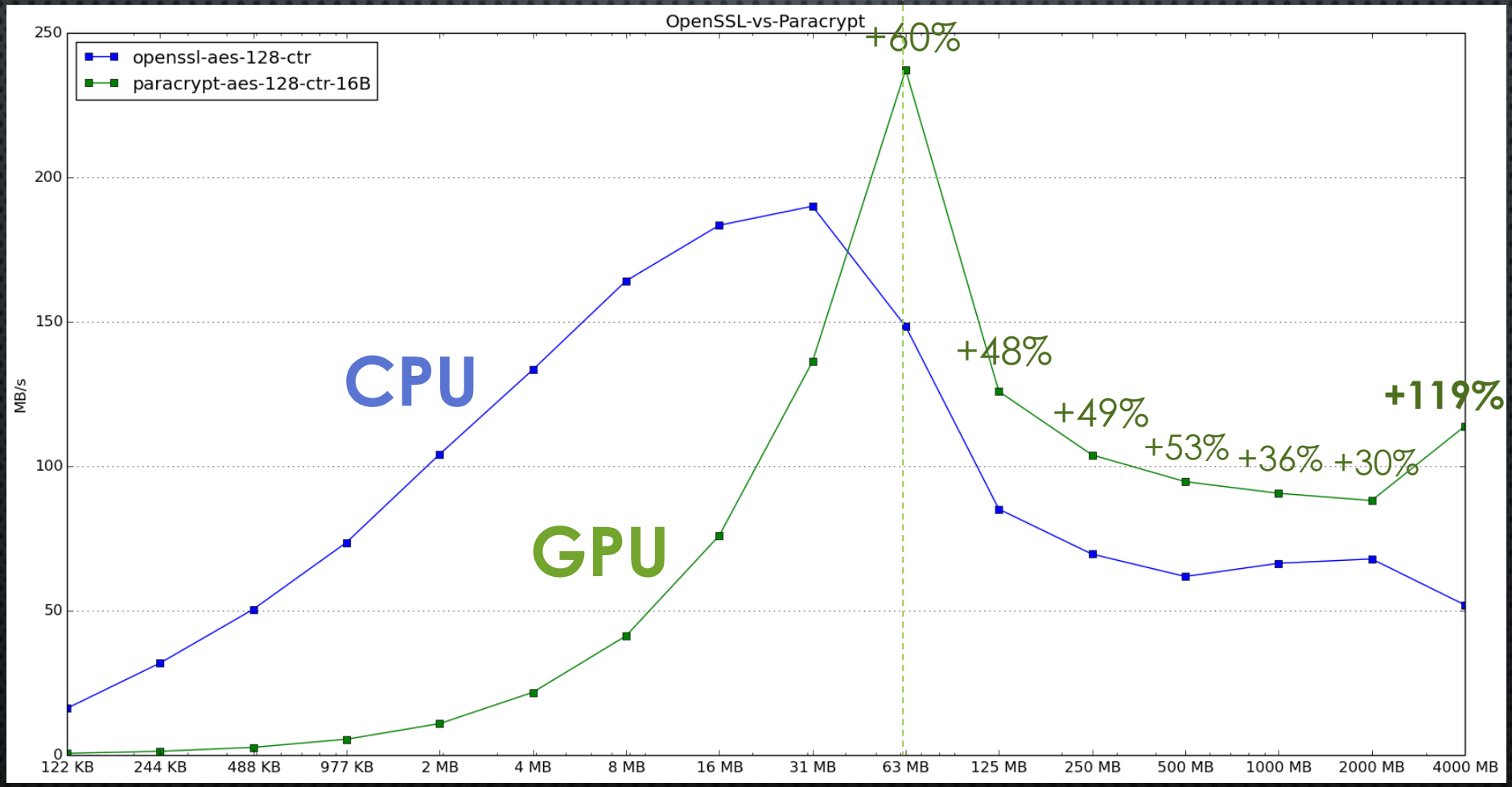
- Se ha implementado soporte para acceso aleatorio:

→ Obtener parte del mensaje en claro sin necesidad de descifrarlo entero



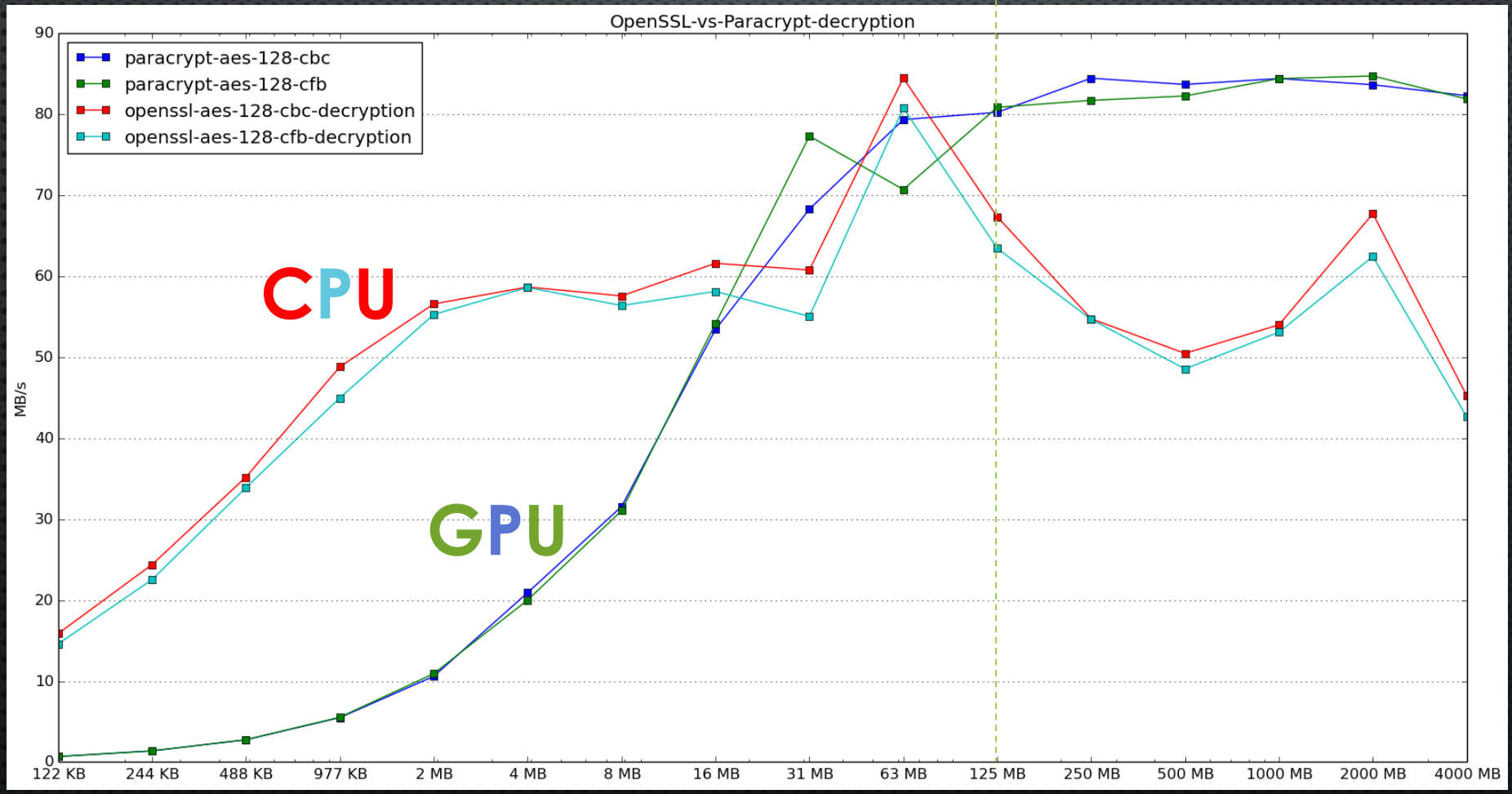
5. COMPARACIÓN CON OPENSSL

Cifrado



5. COMPARACIÓN CON OPENSSL

Descifrado



6. CONCLUSIONES

Máquina de pruebas:

- Velocidad de lectura secuencial en disco: 208 MB/s
- Capacidad bus PCI Express x16 Gen2: 8 GB/s en cada dirección

¡Gran desaprovechamiento potencial de la GPU!

- Hay que verificar que un mayor rendimiento es posible usando discos de mas prestaciones (configuración RAID por ejemplo) y poder cargar más rápidamente datos en la GPU antes de descartar soluciones mas baratas como AES-NI para su uso en sistemas especializados

% beneficio esperado vs CPU:	% beneficio esperado AES-NI:
+55% (+119% max.) cifrado	+38% cifrado
+49% (+87% max.) descifrado	+37% descifrado

- El código desarrollado se puede compilar en Linux en una herramienta para la línea de comandos o como una librería compartida para su uso en C/C++
- Tests Unitarios
  - Pruebas cifrado/descifrado con vectores de prueba del estándar (FIPS 197 del NIST)
  - Cifrado de mensajes aleatorios y comprobaciones de que se obtiene el mismo mensaje tras descifrar

```
> paracrypt --help
GPU accelerated AES

Use: paracrypt -c cipher (-e|-d) -K hexadecimal_key [-iv input_vector] -in input_file -out output_file

Allowed options:
  -h [ --help ]                 produce help message
  --show arg                    show license warranty (w) or conditions (c)
  -q [ --quiet ]                disables logging engine and do not output any
                                messages
  -v [ --verbose ] arg          level of verbosity: warning (default), info,
                                debug, trace
  -c [ --cipher ] arg           selects one of the following ciphers:
                                aes-128-ecb aes-256-ecb aes-192-ecb aes-128-cbc
                                aes-192-cbc aes-256-cbc aes-128-cfb aes-192-cfb
                                aes-256-cfb aes-192-ctr aes-128-ctr aes-256-ctr
  -e [ --encrypt ]              encrypt input
  -d [ --decrypt ]              decrypt input
  -K [ --Key ] arg              hexadecimal key to be used directly by the
```

GPU ACCELERATED AES

IMPLEMENTACIÓN EN GPU DEL CIFRADOR AES

FIN

GRACIAS POR SU ATENCIÓN

POLITÉCNICA

AUTOR: JESÚS MARTÍN BERLANGA

TUTOR: JESÚS MARTÍNEZ MATEO