AES Cryptosystem Acceleration Using Graphics Processing Units

•••

Ethan Willoner
Supervisors: Dr. Ramon Lawrence, Scott Fazackerley

Overview

- Introduction
- Compute Unified Device Architecture (CUDA)
  - Hardware Capabilities
  - Threads
  - Memory Models
- Advanced Encryption Standard (AES)
  - Rijndael Algorithm
  - Hardware Acceleration and AES-NI
- CUDA Implementation of AES
- Applications to Databases
- Results
- Conclusion

Introduction

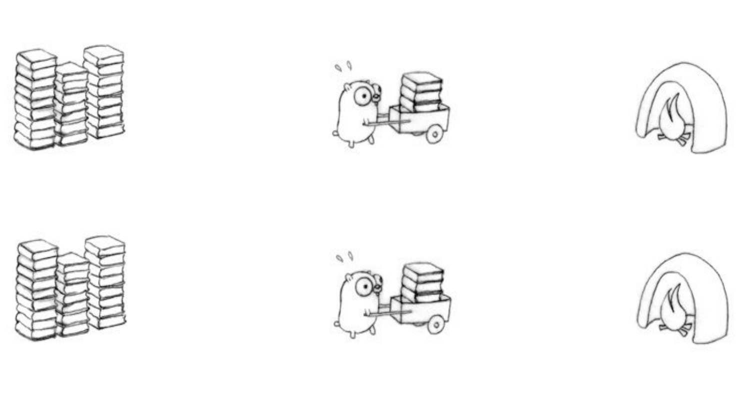
- Digital security is important
  - Big data and large volumes of data means a lot of data to secure
- Graphics Processing Units are a worthwhile solution for processing large data sets
- Research has shown that utilizing Graphics Processing Units to encrypt and decrypt data is faster than traditional Central Processing Units
  - But previous analysis has been lacking, they don’t compare to real world CPU implementations
- Database applications
- We need to consider Parallelism vs Concurrency

Visual Example

Gophers destroy old computer manuals

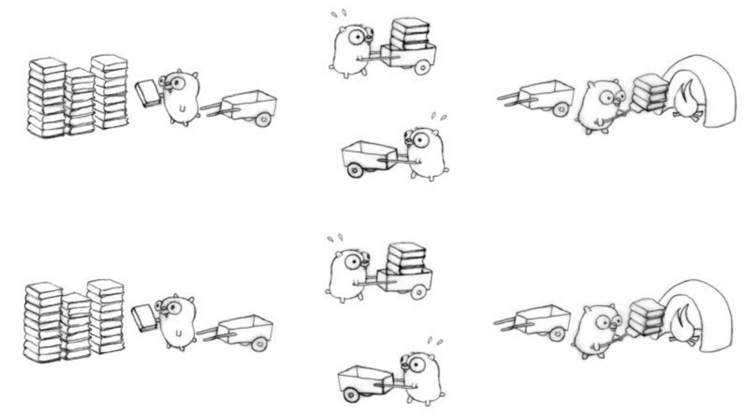
Parallelism!

- Each gopher independently performs the same task



Concurrency!

- Gophers performing one of the tasks A, B, C or D

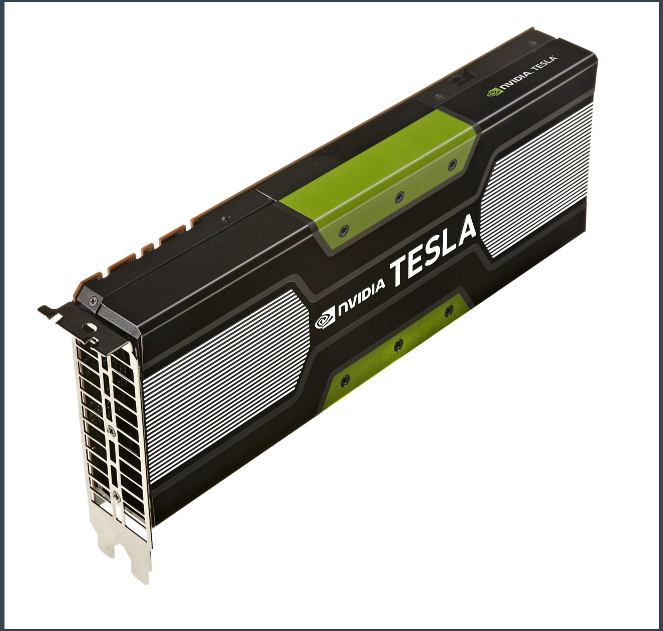


Source: Rob Pike, https://talks.golang.org/2012/waza.slide

Compute Unified Device Architecture (CUDA)

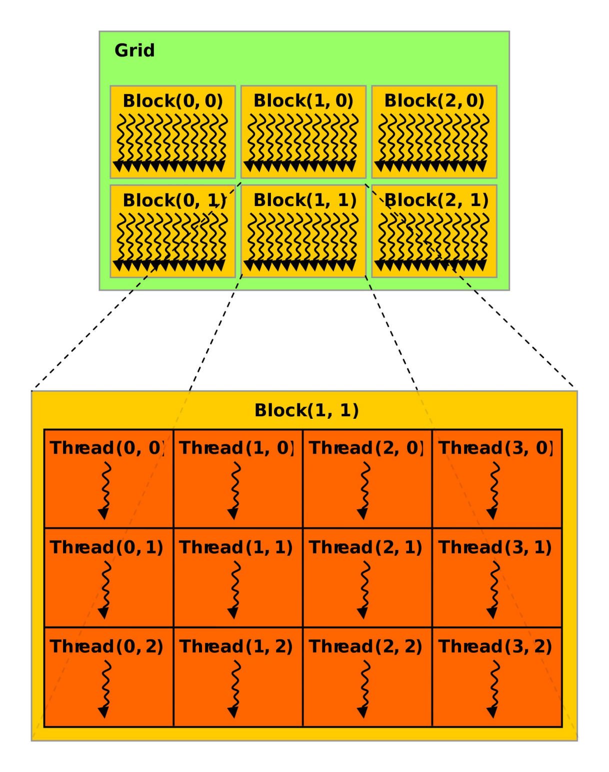
- Invented by NVIDIA and was released in 2006
- Parallel computing runtime and programming model which leverages Graphics Processing Units (GPUs) for massively parallel data processing
- Researchers were drawn to them because of excellent floating point performance in a parallel manner (Lots of Gophers! 1000’s!)
- CUDA has become incredibly important for CAD, Deep Learning, Medical Research and Imaging, and many other applications

DESIGNED FOR
NVIDIA.
CUDA™



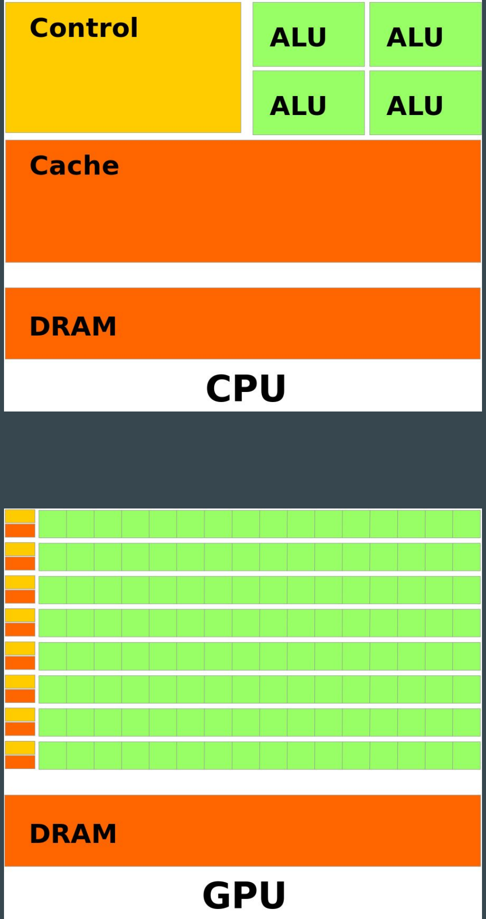
CUDA Threads

- Similar to CPU threads, CUDA threads are the smallest unit of execution which happens on the GPU.
- Groups of Threads run in Thread Blocks, and Thread Blocks run in Grids
- Thread blocks are scheduled across numerous streaming multiprocessors located on the GPU
  - Each processor may have 8, 16, or 32 blocks executing
- CUDA threads are only effective when executing identical operations on a set of data
  - GPUs are parallel but not concurrent



CUDA Memory Models

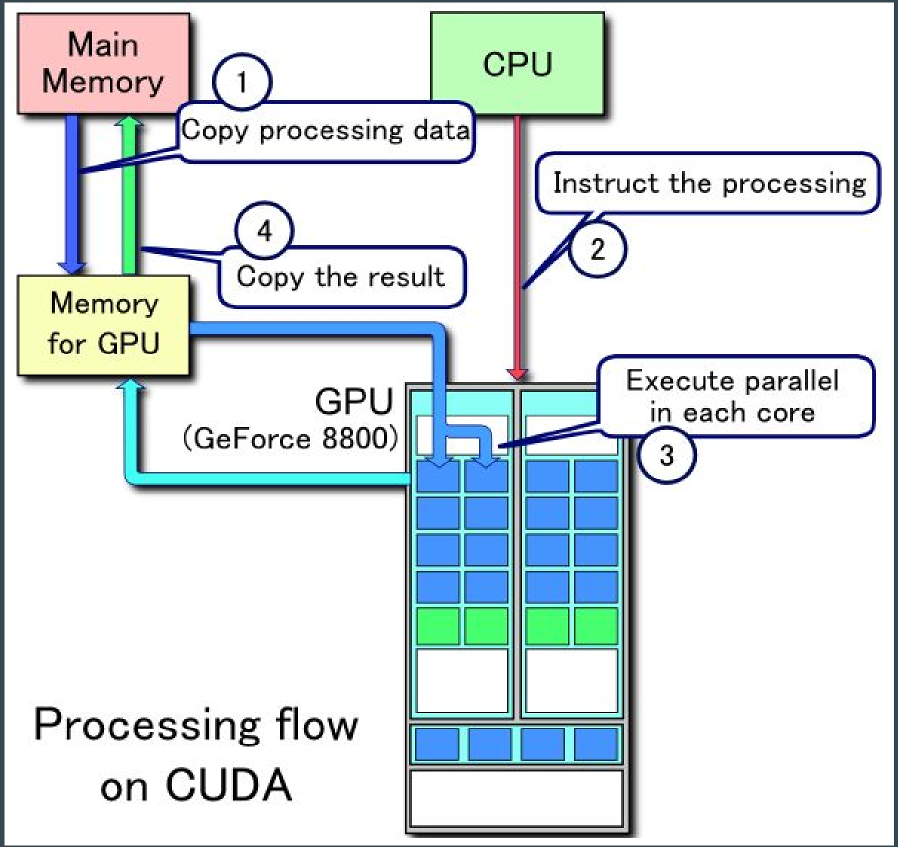
- GPUs utilize their own DRAM on the device itself, separate from the host system’s RAM, for mass data storage, called Global Memory
- Sharing data between the GPU and host can either be managed by the programmer, or managed by CUDA using unified memory
  - Manually managed: explicitly copy memory between CPU DRAM and GPU DRAM
  - Unified Memory: CUDA runtime worries about copying memory as it is needed. Simpler, but slower.



By NVIDIA (NVIDIA CUDA Programming Guide version 3.0) [CC BY 3.0 (http://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons

Putting it together

1. Input data is copied to Global Memory
2. Launch the GPU Kernel
3. Kernel processes data in parallel
4. Output data is copied to Main Memory



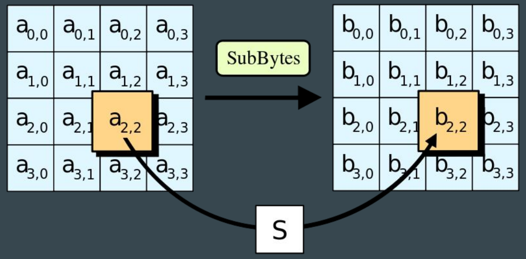
By Tosaka - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=5140417

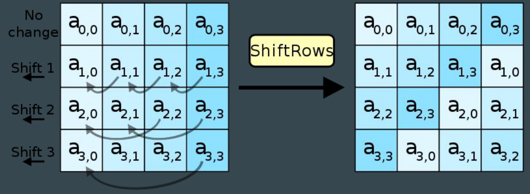
Advanced Encryption Standard (AES)

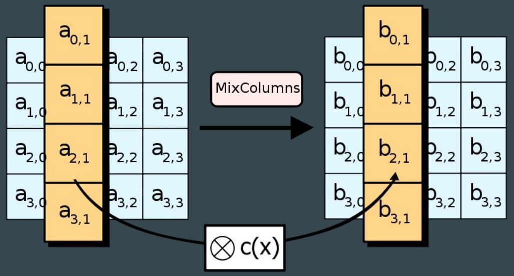
- AES is a symmetric block cipher chosen by the National Institute for Standards and Technology to serve as a standard encryption system for the US government
- The Rijndael algorithm was submitted by Joan Daemen and Vincent Rijmen
- Chosen based on features such as estimated security, performance, and cryptanalysis resistance
    - Supports variable key sizes for security/performance trade-offs
- AES is heavily used in all areas of computing, including secure web browsing (HTTPS), Disk Encryption (Bitlocker, FileVault), Databases, and protects data stored by hospitals, governments, universities, etc.

Rijndael Algorithm

- Every block of 16 bytes is treated as a 4x4 matrix in the Rijndael algorithm
- Encryption:
  - AddRoundKey
    - Combine (XOR) our secret key with the data block
  - SubBytes
    - Substitute every byte with a value from a lookup table
  - ShiftRows
    - Each row of our matrix is cyclically shifted
  - MixColumns
    - Each column of our matrix is transformed with an invertible linear transformation
- For decryption, each operation has an applicable inverse function

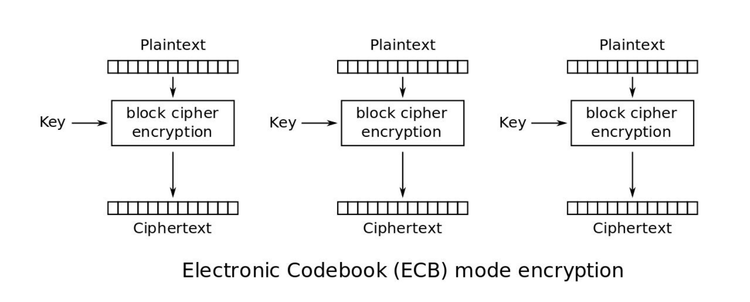

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
S
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$


No change
Shift 1
Shift 2
Shift 3
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

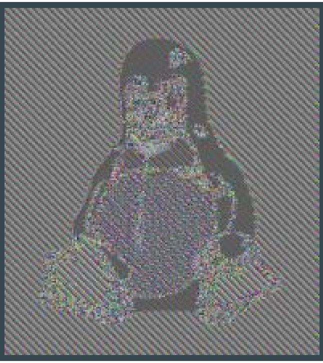

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
MixColumns
$\otimes c(x)$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

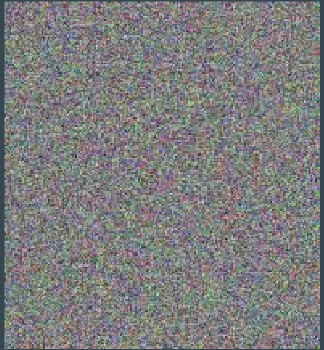
Rijndael Algorithm cont.

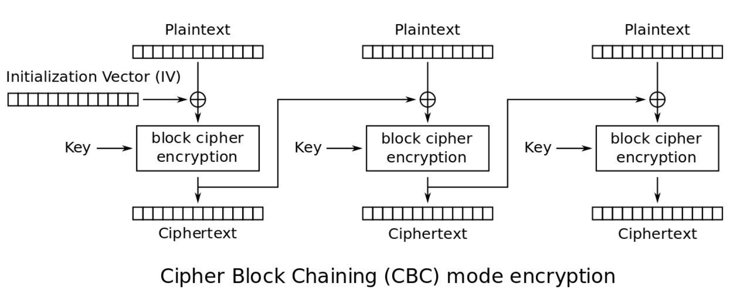
- Rijndael encrypts one block at a time (ECB), which poses some issues for identifying patterns in data
- By utilizing a method such as Cipher Block Chaining (CBC), we can eliminate pattern based analysis of our data



Electronic Codebook (ECB) mode encryption







Cipher Block Chaining (CBC) mode encryption

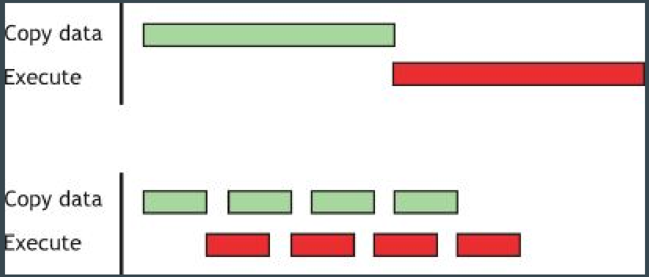
Tux the Penguin, the Linux mascot. Created in 1996 by Larry Ewing with The GIMP.

Hardware Acceleration and AES-NI

- Ubiquity of AES led Intel and AMD to implement it with hardware acceleration in many of their CPUs starting in 2010
  - Originally only available for server CPUs
  - Quickly became common in consumer CPUs as well
  - Limited uptake on mobile CPUs
- This hardware acceleration is exposed through an x86 instruction set called AES-NI, or Advanced Encryption Standard New Instructions
  - Intel claims up to 10x faster than software implementations
- Why do we care? Because of OpenSSL
  - OpenSSL of Heartbleed infamy
  - OpenSSL is one of the most common cryptographic libraries
  - OpenSSL is commonly utilized by databases, web servers, and high performance applications
  - OpenSSL supports AES-NI and hardware acceleration

CUDA Implementation of AES

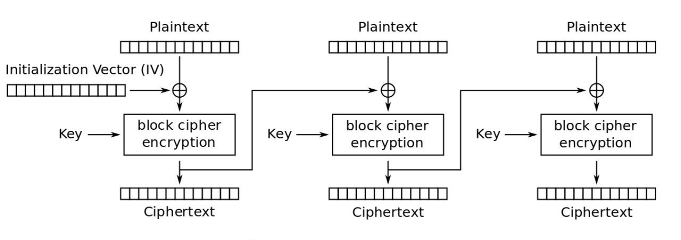
- The implementation is massively parallel, and incredibly high performance
  - Each block of data is assigned to its own thread.
  - This gives us the best parallelism possible as there is no thread divergence
- Shared memory it heavily utilized
  - Our lookup tables and encryption/decryption keys are read often
  - Lookup tables are especially low performance because they are based on the data, so locality cannot be reliably exploited by multiprocessor cache.
  - Shared memory helps lessen these effects
- Utilization of CUDA Streams
  - This overlaps data transfer and kernel execution
  - Copy large data set in smaller pieces
  - Launch shorter running kernels more often
  - We “hide” some of the copy time in kernel execution



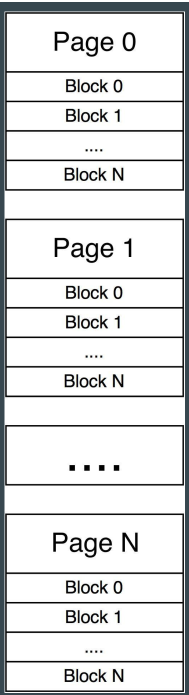
http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/#asynchronous-transfers-and-overlapping-transfers-with-computation

Database Applications

- Recall Cipher Block Chaining mode of AES
    - More secure, resistant to pattern analysis
    - Notice the encryption algorithm, each block relies on the previous block's output
    - This makes parallelization impossible
    - Decryption is fully parallelizable, as there is no dependency on the output of a previous block
- Databases do not store tables in singular large files
    - Chunks of data, called Pages, are encrypted separately
    - We can parallelize the encryption of page sets by assigning each page of data to a CUDA thread



Cipher Block Chaining (CBC) mode encryption

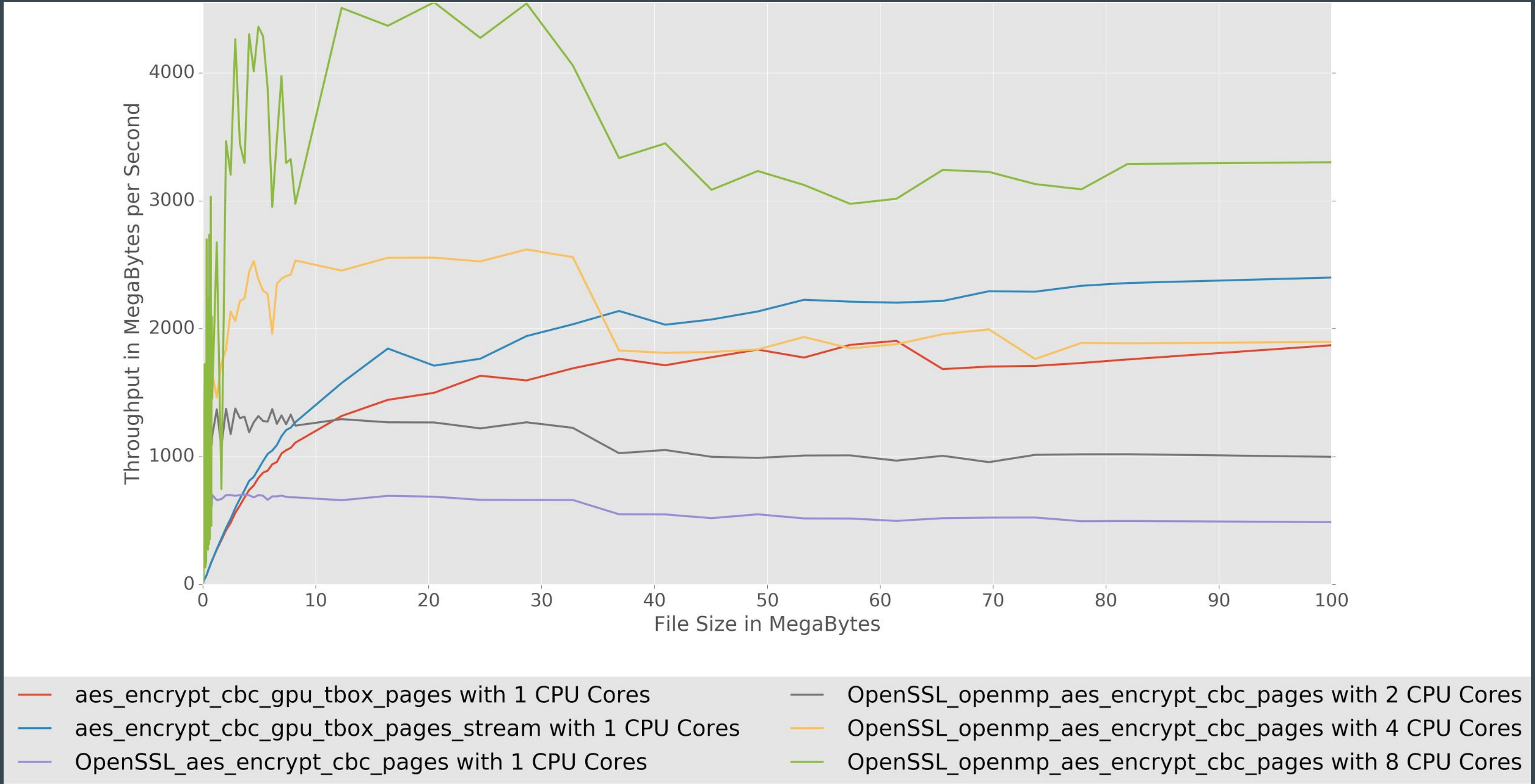


Hardware and Testing

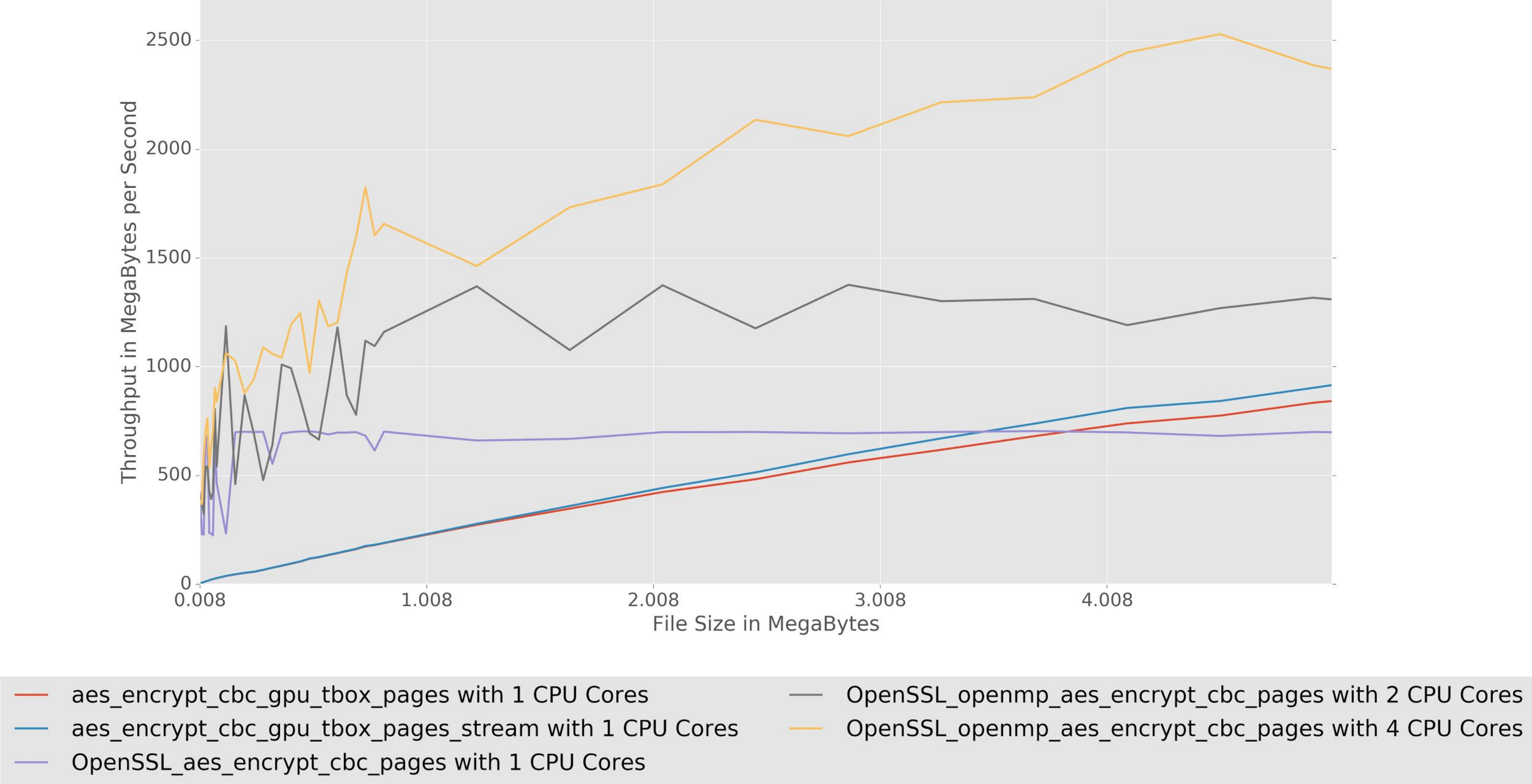
- Test machine:
  - AMD 8350 8 Core CPU
  - 16GB DDR3 RAM
  - Nvidia GTX 780 w/ 3GB VRAM
- We aim to test the following:
  - CPU throughput processing small to large numbers of data pages
  - CPU throughput when parallelized across up to 8 cores using OpenMP
  - GPU throughput both with and without utilizing streams
  - GPU throughput when utilized with a “Copyless” architecture

Results

Results - Throughput

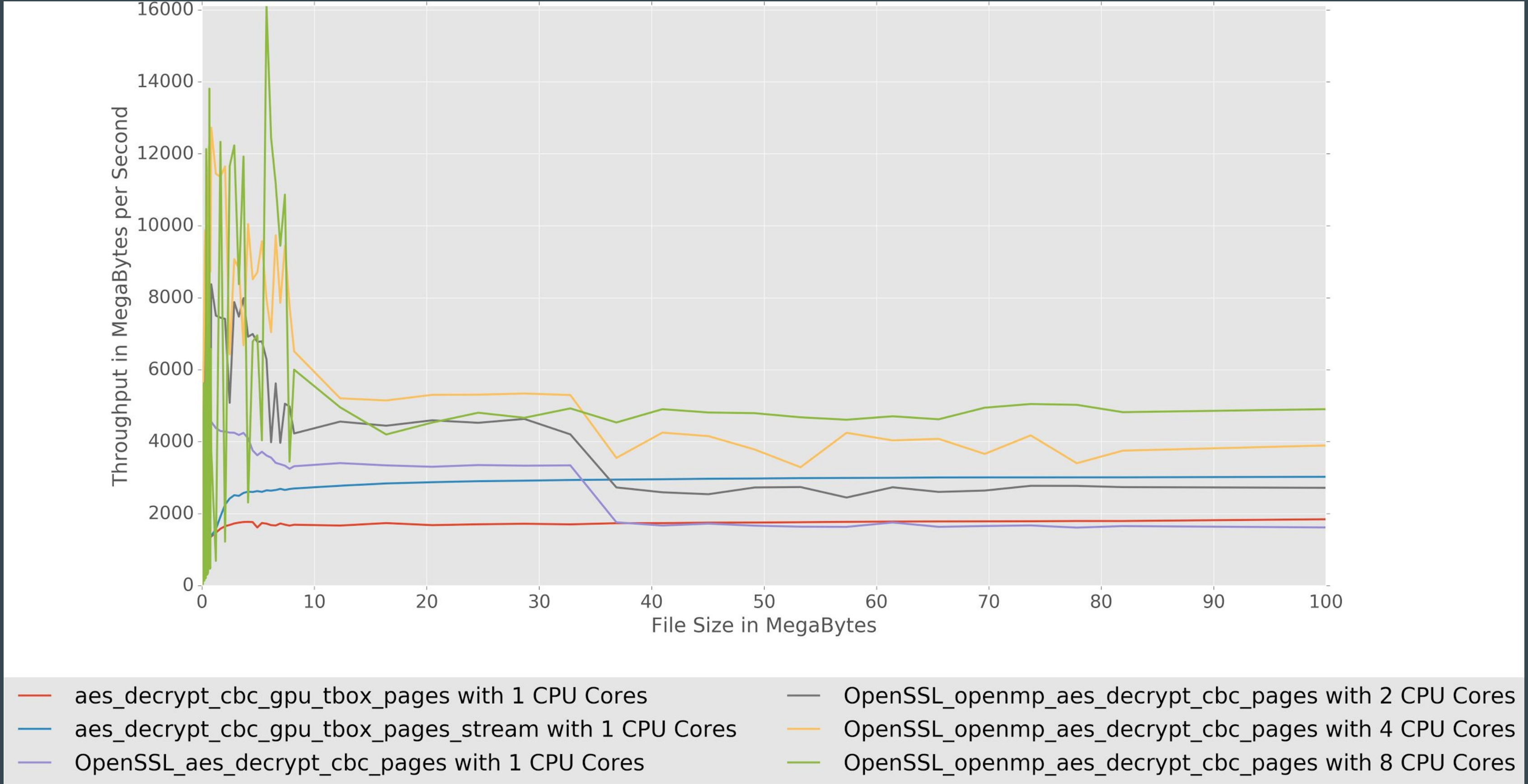


Results

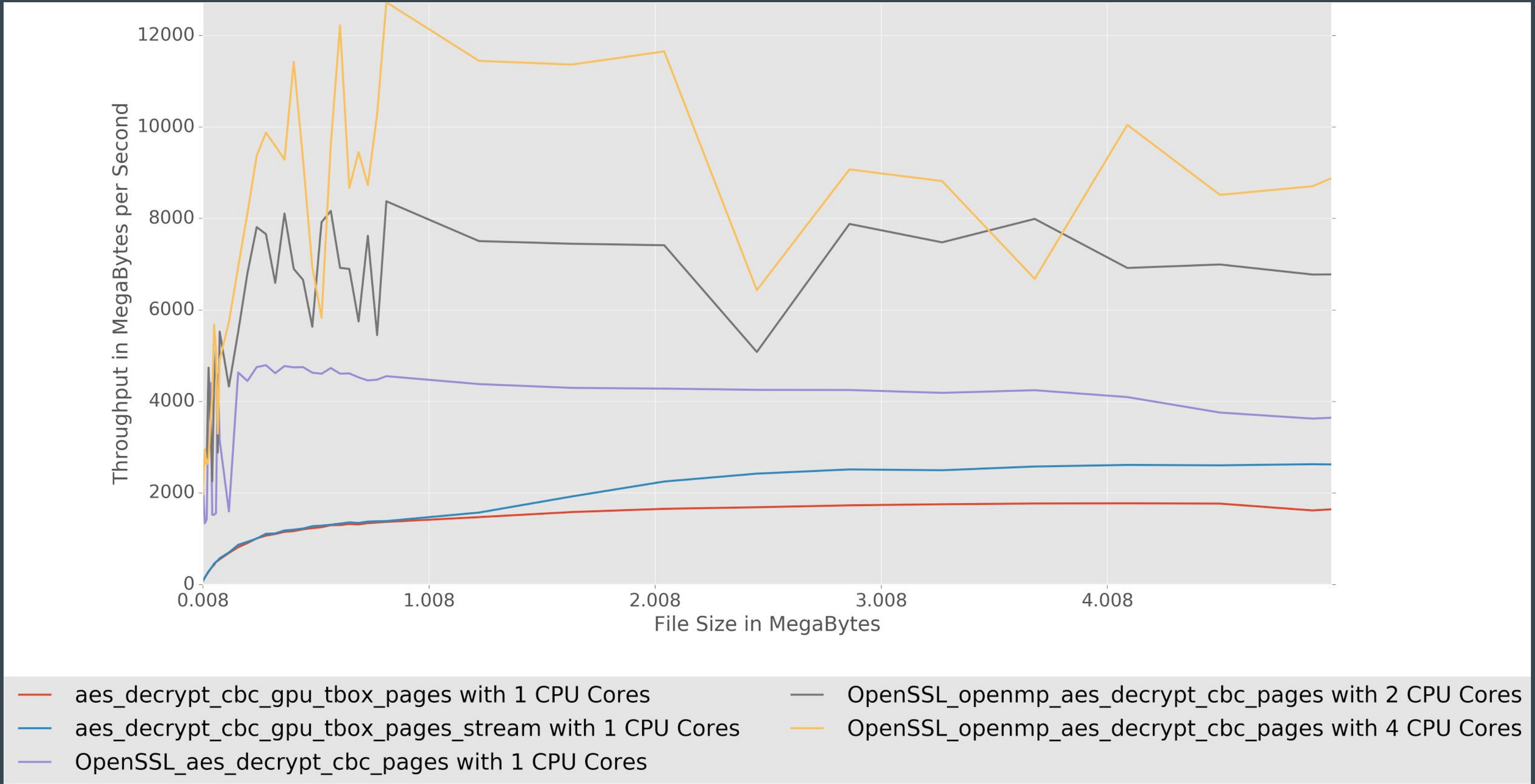


Throughput in MegaBytes per Second
0
500
1000
1500
2000
2500
0.008
1.008
2.008
3.008
4.008
File Size in MegaBytes
aes_encrypt_cbc_gpu_tbox_pages with 1 CPU Cores
aes_encrypt_cbc_gpu_tbox_pages_stream with 1 CPU Cores
OpenSSL_aes_encrypt_cbc_pages with 1 CPU Cores
OpenSSL_openmp_aes_encrypt_cbc_pages with 2 CPU Cores
OpenSSL_openmp_aes_encrypt_cbc_pages with 4 CPU Cores

Results - Throughput



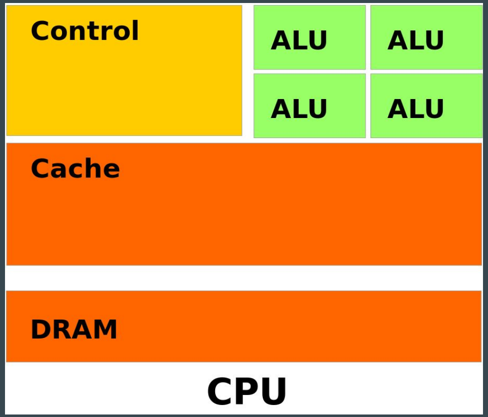
Results

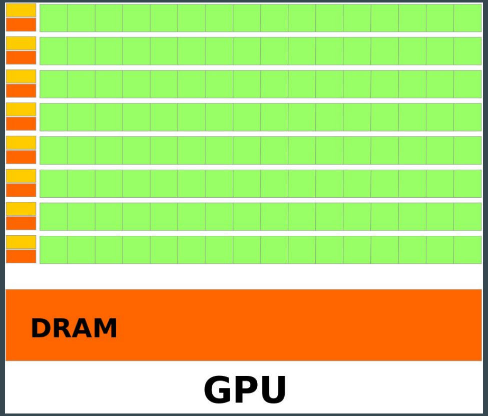


Throughput in MegaBytes per Second
12000
10000
8000
6000
4000
2000
0
0.008
1.008
2.008
3.008
4.008
File Size in MegaBytes
aes_decrypt_cbc_gpu_tbox_pages with 1 CPU Cores
aes_decrypt_cbc_gpu_tbox_pages_stream with 1 CPU Cores
OpenSSL_aes_decrypt_cbc_pages with 1 CPU Cores
OpenSSL_openmp_aes_decrypt_cbc_pages with 2 CPU Cores
OpenSSL_openmp_aes_decrypt_cbc_pages with 4 CPU Cores

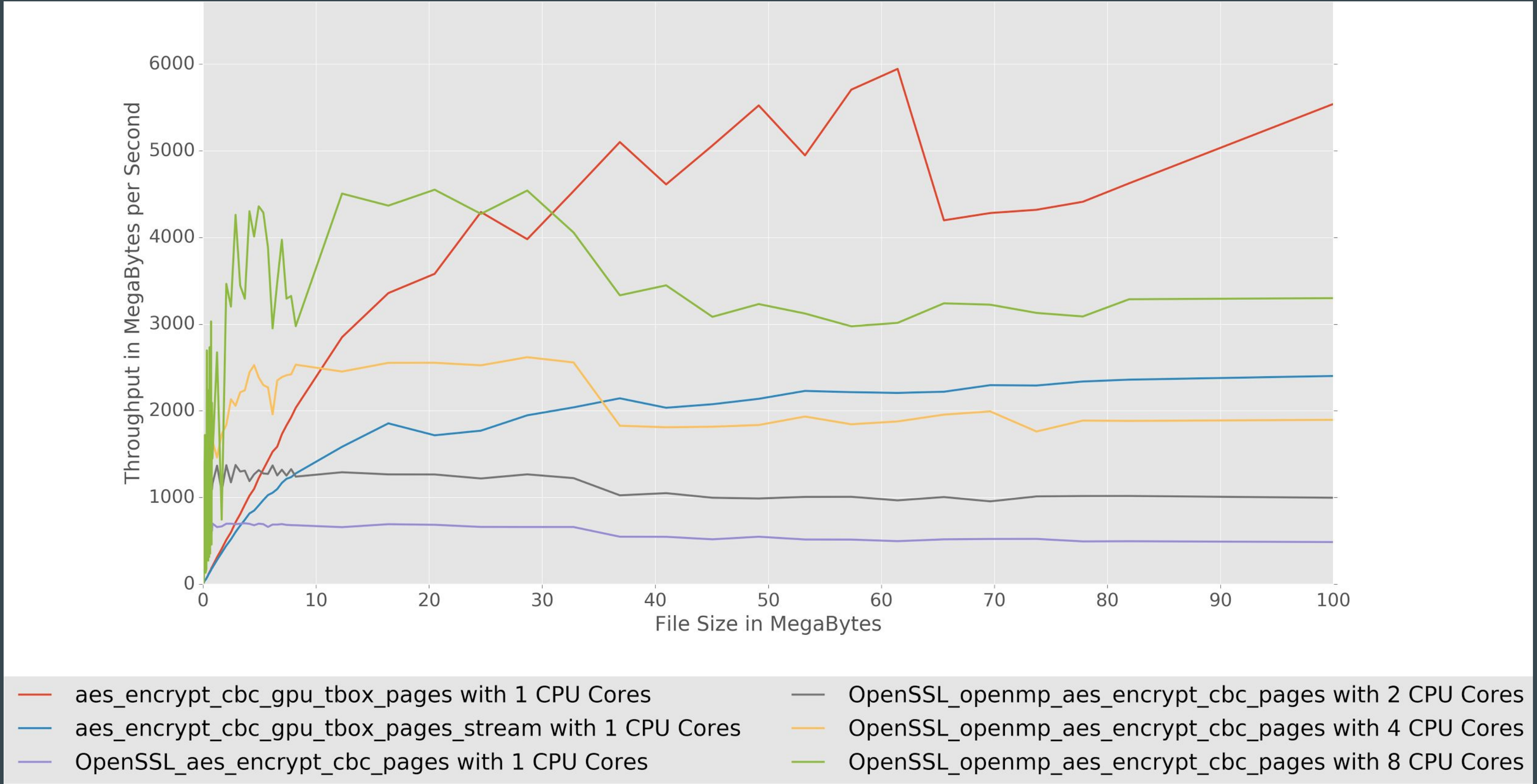
Recall back to our Memory Layout diagram...

- The GPU is an an inherent disadvantage because we must copy data to it before processing
  - CPU can start working immediately
- High speed PCI links such as storage or network devices allow us to copy the data straight to the GPU memory
  - Bypass the CPU DRAM totally
- This is now a more accurate comparison of systems,
  - CPU is only needed to launch a GPU application, not actually do the data copies
- Data starts off on the GPU, eliminating the data copy expenses
  - Now the GPU is in the same position as the CPU
- Helps to offload parallelizable tasks from the CPU and leave it to perform concurrent tasks

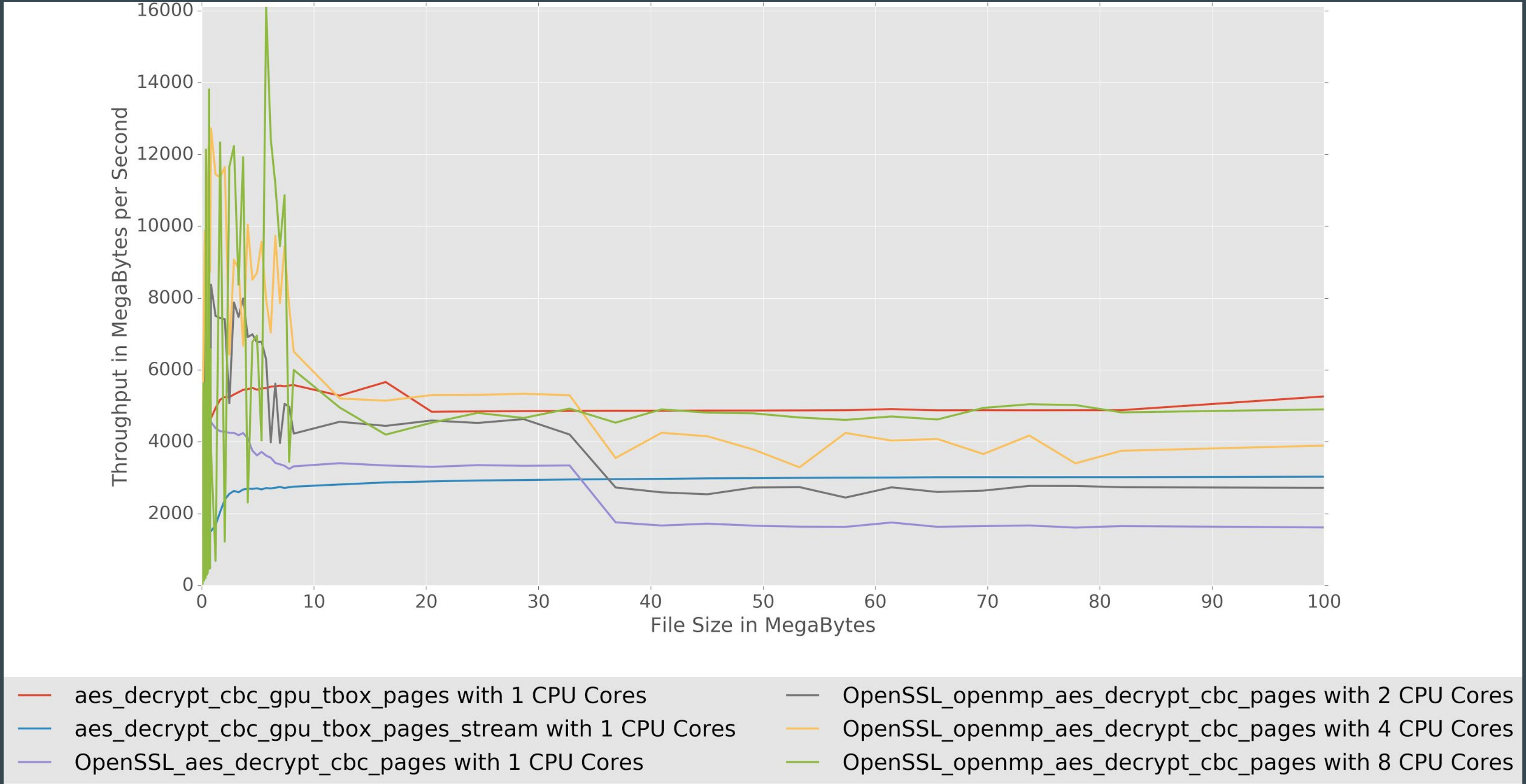




Results - Throughput



Results - Throughput



Conclusion

- Previous researchers have concluded that GPUs are better at AES than CPUs
    - Many of the papers I found ignored AES-NI completely
    - This research has shown that in fact, GPUs are not applicable to many workloads
        - Workloads with “small” data are not as performant as previously claimed
- OpenSSL with hardware support provides competition to CUDA implementations
    - Picking the correct tool for the correct workload is incredibly important
- There is definitely a place for this in environments which deal with large amounts of encrypted data
    - But we need to modify how we think about the problem and aim to remove memory copy costs totally

Questions?