AES-128 ECB

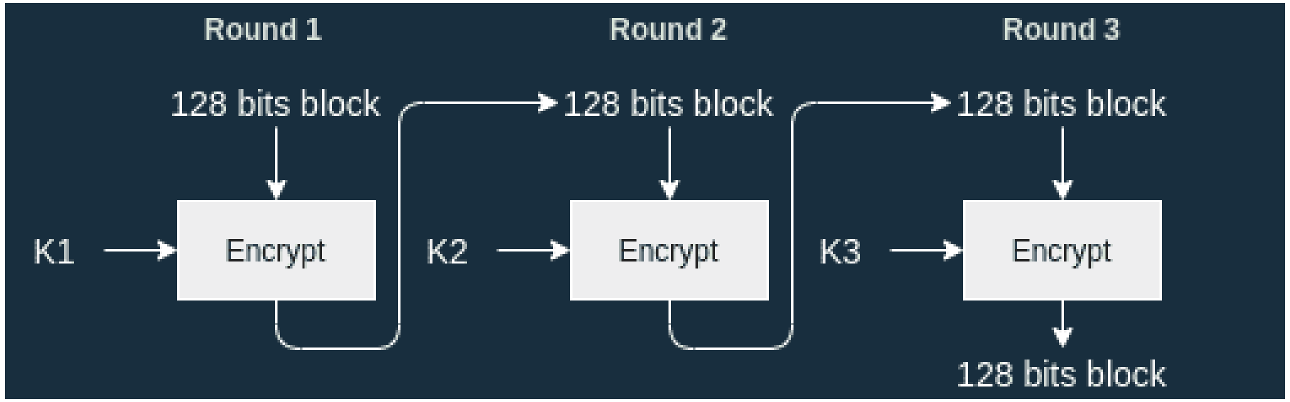
Comparison between sequential, OpenMP and CUDA implementations

Daniele Bianchini

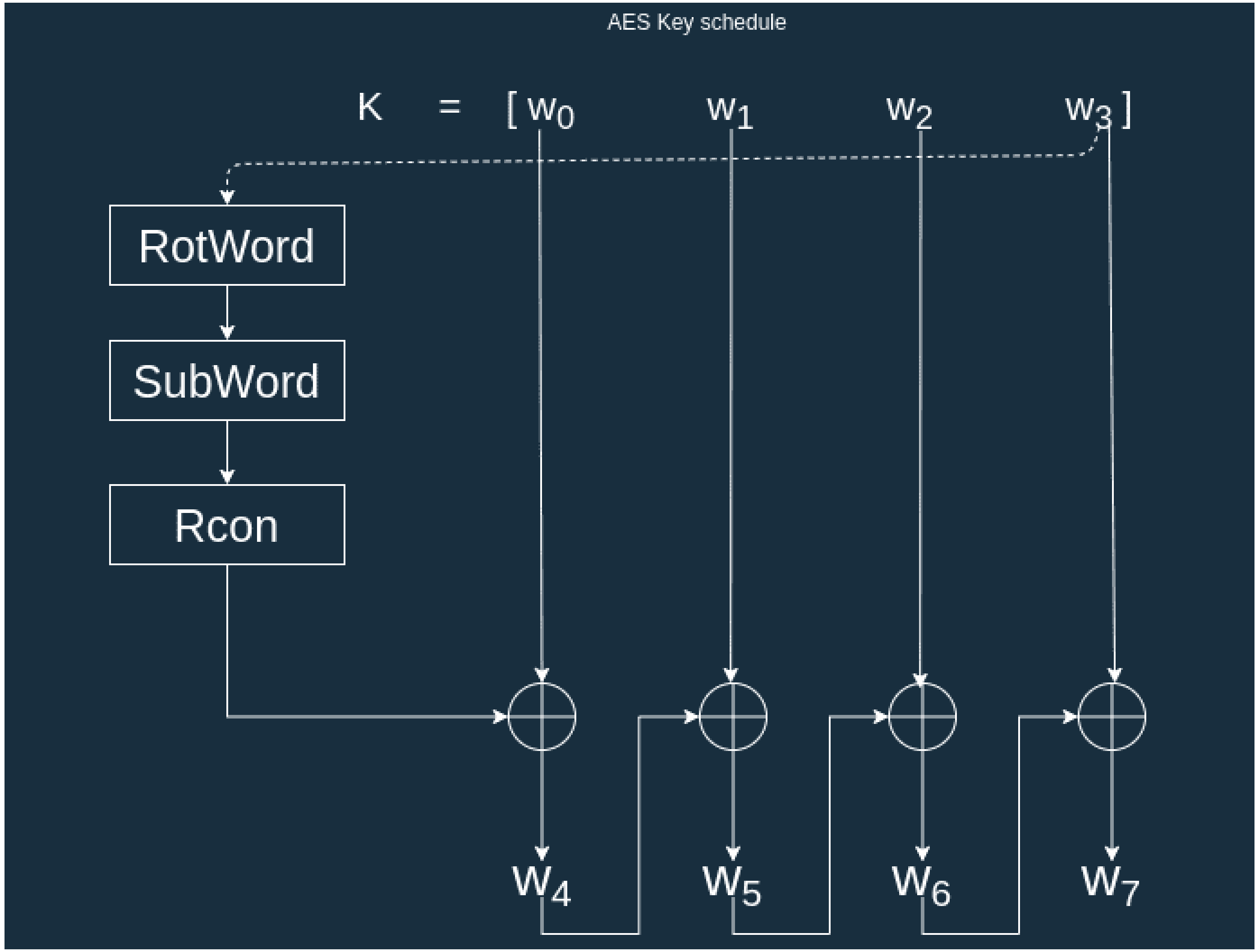
HPC Assignment

AES-128

- Plaintext divided in 16 bytes blocks
- 16 byte long secret key (128 bits)
- 10 round keys derived from secret key -> Key scheduling
- Electronic codebook mode of operation: each block is ciphered indipendently
- Each block undergoes 11 rounds with each roundkey
- S-Box: constant matrix used to permutate bytes



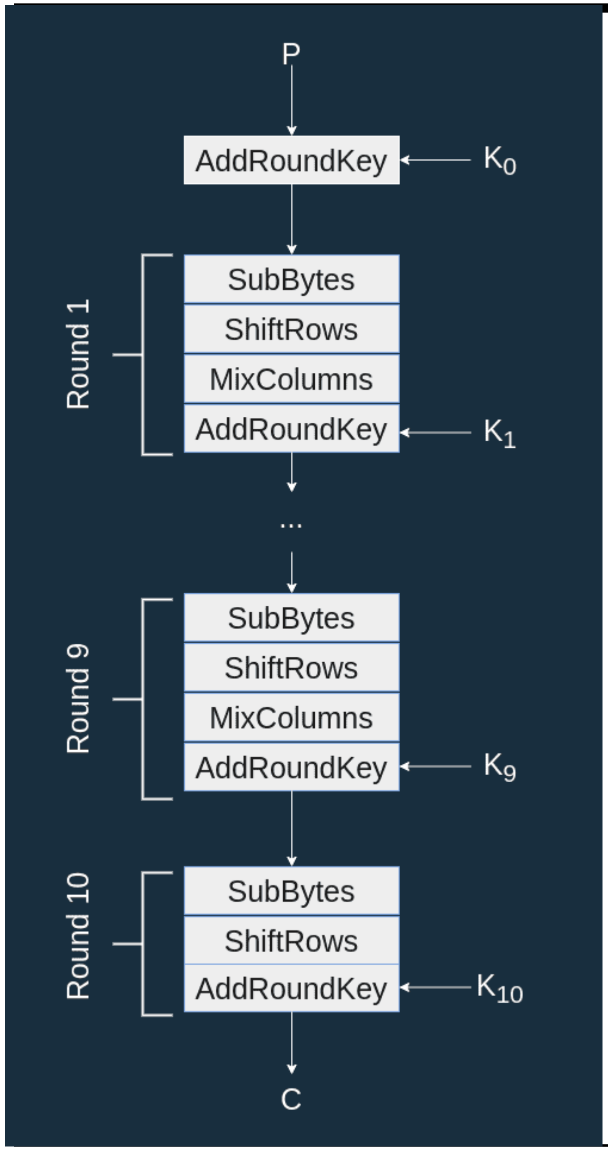
Key Scheduling



- RotWord: Left-rotate a 4 byte word
  - (w0 w1 w2 w3) -> (w1 w2 w3 w0)
- SubWord: Apply S-Box permutation to each byte of a word
- Rcon: xor first byte of a word with a constant round r dependant -> 2^r-1 mod 2^8

Until we obtain 10 keys. Dependencies prevents parallelization

Cipher algorithm



- AddRoundKey: xor block of plaintext with round key
- ShiftRows: performs a cyclical rotation of the block by rows of bytes (matrix of the state)

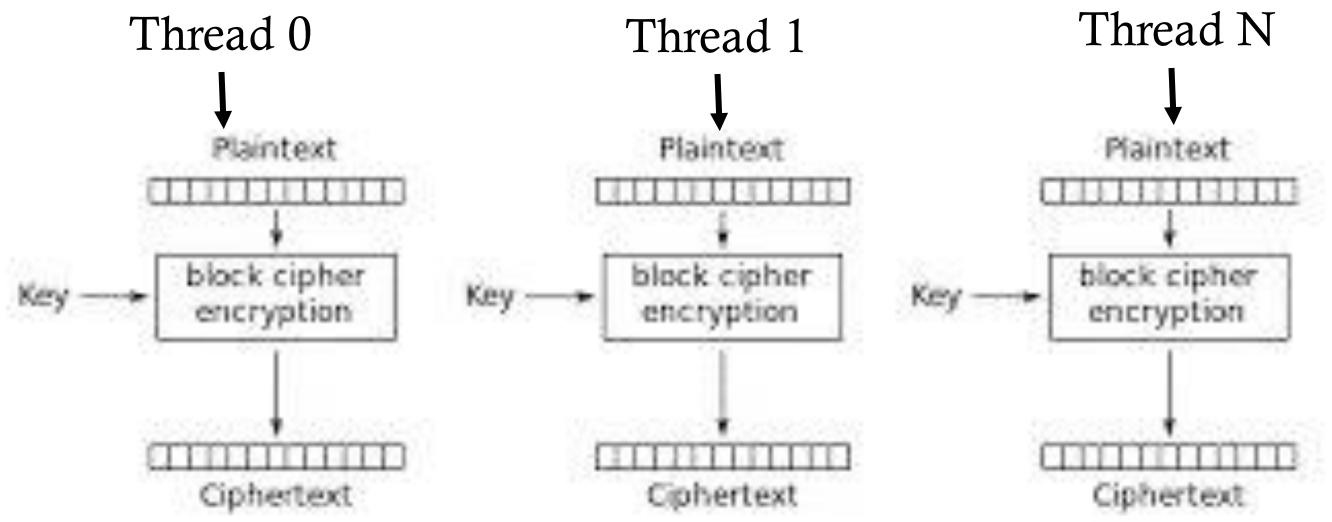
$$\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \rightarrow \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{bmatrix}$$

- MixColumns: Matrix constant multiplication in GF 2^8

$$\begin{bmatrix} t_{0,0} \\ t_{1,0} \\ t_{2,0} \\ t_{3,0} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} \\ s_{1,0} \\ s_{2,0} \\ s_{3,0} \end{bmatrix}$$

Improving performance

- Bigger plaintext means more blocks to cipher, longer time of execution
- Key scheduling is relatively fast and not parallelizable
- Cipher function treats each block indipendently
    - synchronization ok
    - Static workload
- Cipher function treats contiguous addresses of blocks
- Cipher function can be unrolled very well



Electronic Codebook (ECB) mode encryption

OpenMP

- Max improvement with 16 threads
- For cycle iterating on each block
- Static and balanced workload

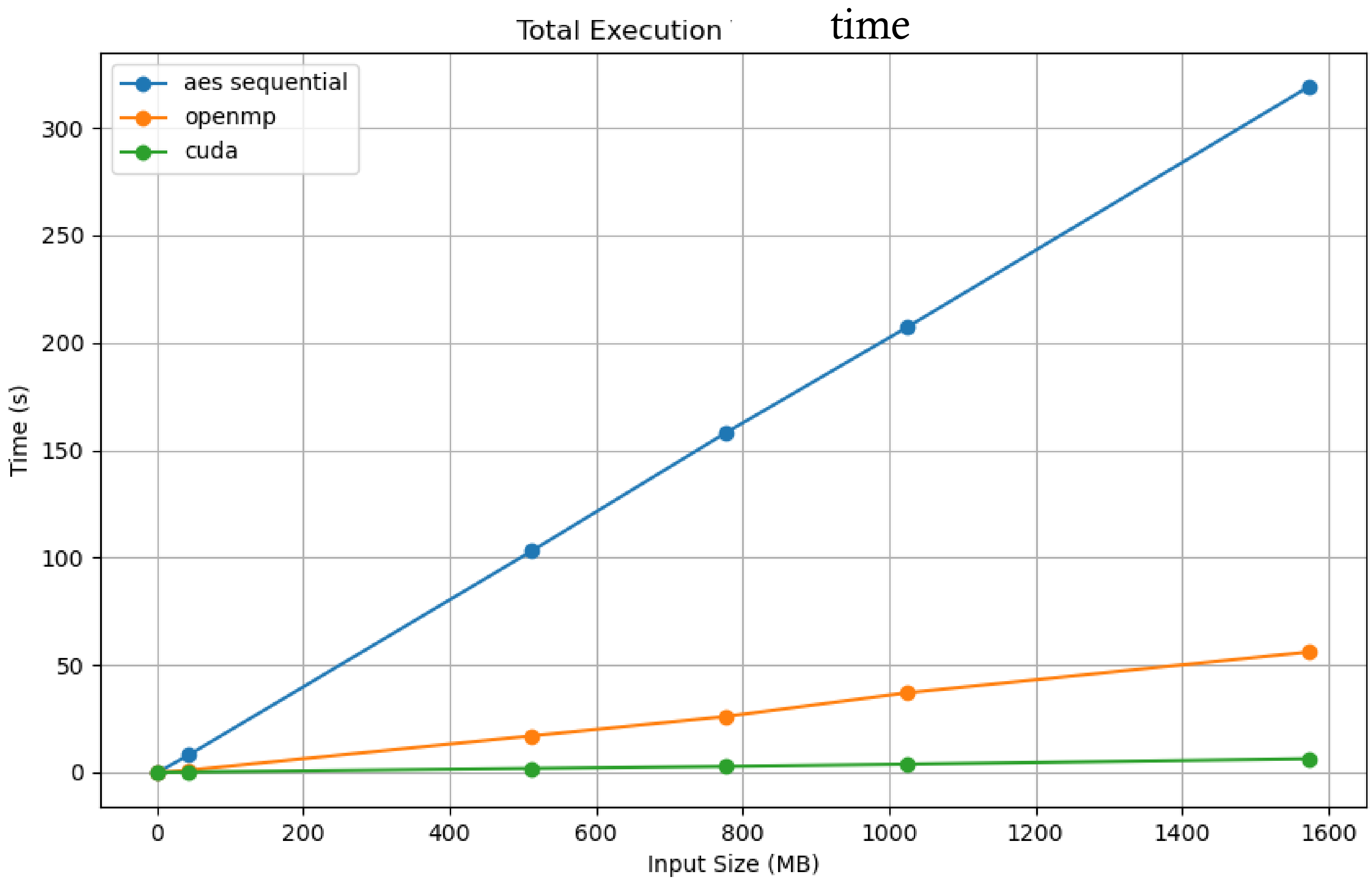
```
omp_set_num_threads(16);

printf("Encrypting...\n");
[redacted] #pragma omp parallel for
for(int i = 0; i < blocks_to_cipher; i++){
    encrypt_ECB(plaintext + (i*BLOCKSIZE),KEY);
}
```

CUDA

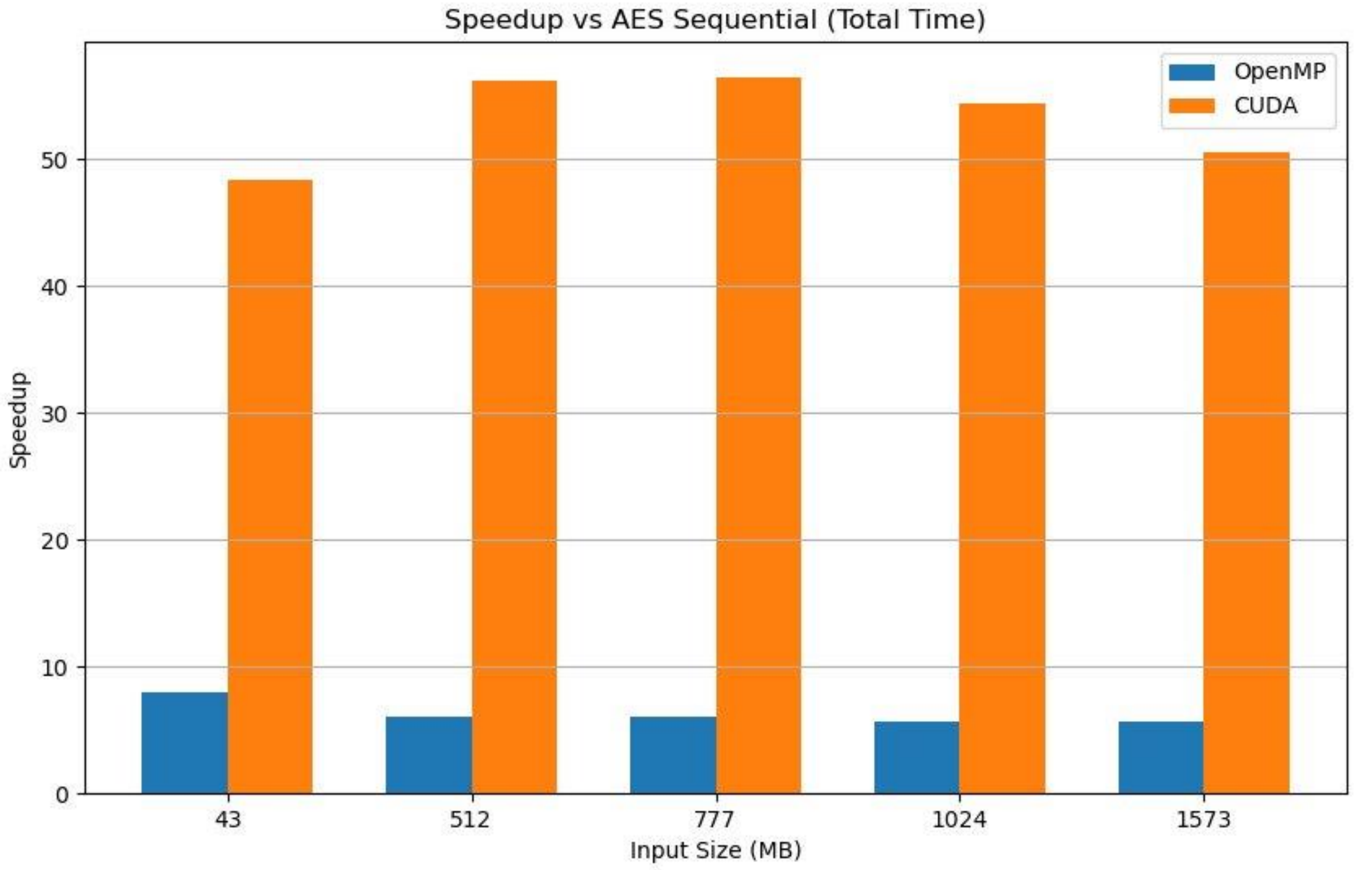
- S-Box and keys in constant memory for fast read-only access
- Optimal number of threads: 256
- Cuda blocks: (blocks_to_cipher + thread_n - 1) / thread_n
- Each thread works on the block corresponding to its ID and all blocks at offset (blockDim.x * gridDim.x)

```
int thread_n = 256;
int blocks_pergrid = (blocks_to_cipher + thread_n - 1) / thread_n;
printf("Blocks: %d\n",blocks_pergrid);
cudaEventRecord(alg_start);
encryptFull_ECB<<<blocks_pergrid,thread_n>>>(plain_d,blocks_to_cipher);
if (err != cudaSuccess) {
    printf("CUDA kernel launch failed: %s\n", cudaGetErrorString(err));
}
cudaDeviceSynchronize();
cudaEventRecord(alg_stop);
```

Execution time

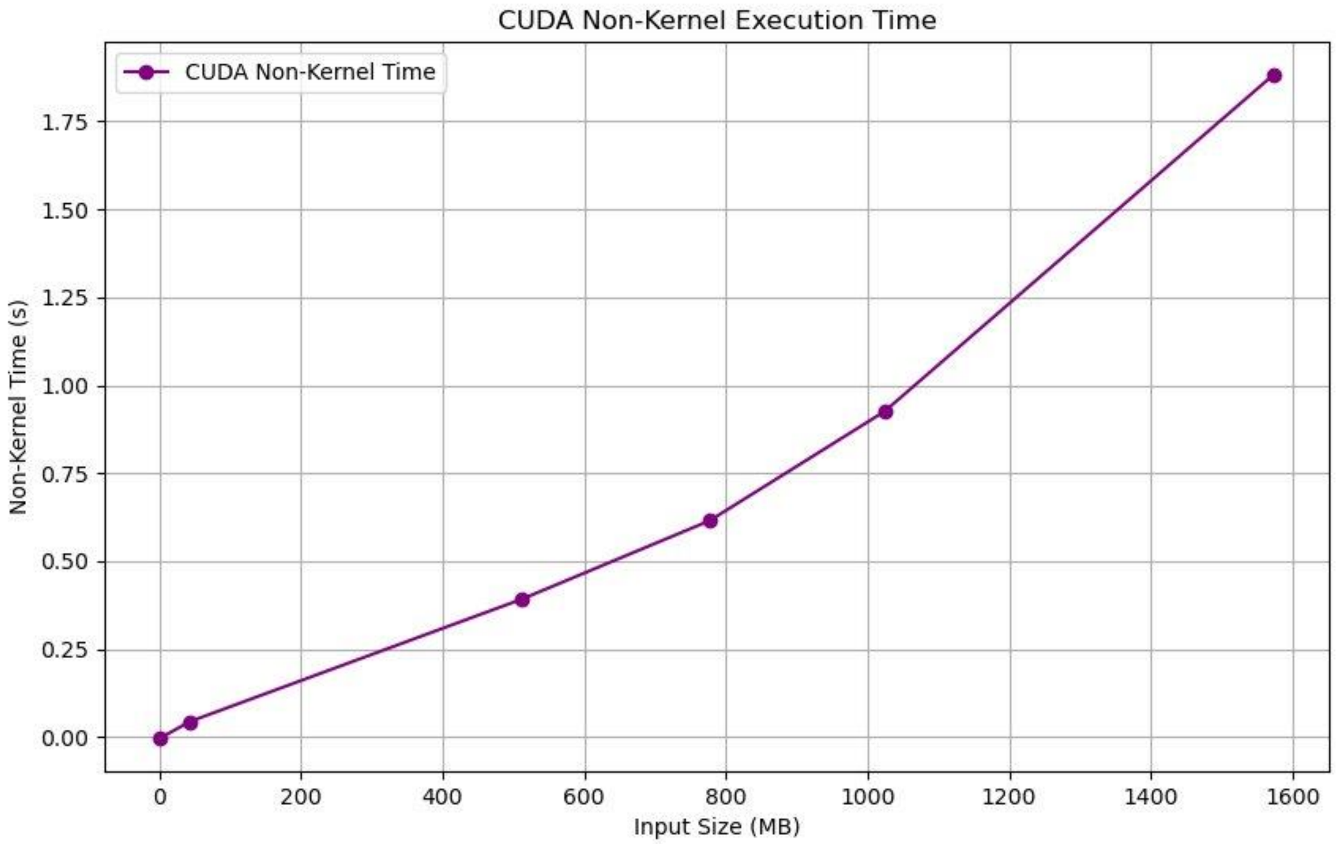


Speedup



- OpenMP speedup is about 5-7x
- Cuda speedup up to >50x
- With bigger filesizes to cipher, speedup lowers for CUDA…

Non-kernel execution time



- Bigger plaintexts means more memory to copy to and from the device
- Non-kernel execution time grows fast
- Speedup compensate more than well

Github Repo

- git clone https://github.com/Lick1Fonzi/AES-128_ECB_Parallel_HPCUnimore.git
- ./xstat.sh -> generates 4 pngs with the result charts