Analysis on the AES Implementation with Various Granularities on Different GPU Architectures

Ahmed Awadalla ABDELRAHMAN[1], *Mohamed Mahmoud FOUAD*[1], *Hisham Mohamed DAHSHAN*[2]

[1] Department of Computer Engineering, Military Technical College, Al-Khalifa Al-Maamoon Street Kobry Elkobbah, Cairo, Egypt
[2]Department of Communications, Military Technical College, Al-Khalifa Al-Maamoon Street Kobry Elkobbah, Cairo, Egypt

ahmedsoliman@mtc.edu.eg, mmafoad@mtc.edu.eg, hdahshan@mtc.edu.eg

DOI: 10.15598/aeee.v15i3.2324

Abstract. *The Advanced Encryption Standard (AES) is One of the most popular symmetric block cipher because it has better efficiency and security. The AES is computation intensive algorithm especially for massive transactions. The Graphics Processing Unit (GPU) is an amazing platform for accelerating AES. it has good parallel processing power. Traditional approaches for implementing AES using GPU use 16 byte per thread as a default granularity. In this paper, the AES-128 algorithm (ECB mode) is implemented on three different GPU architectures with different values of granularities (32,64 and 128 bytes/thread). Our results show that the throughput factor reaches 277 Gbps, 201 Gbps and 78 Gbps using the NVIDIA GTX 1080 (Pascal), the NVIDIA GTX TITAN X (Maxwell) and the GTX 780 (Kepler) GPU architectures.*

Keywords

AES, compute unified device architecture (CUDA), GPU, granularity.

1. Introduction

Nowadays, The demand for constructing fast and secure communication networks is very important. As the size of data sets increases, the speed of encryption process also increases. One of the most widely used block ciphers is the AES [1] that consists of many intensive computations [2], [3], [4] and [5]. Usch computations can be performed on GPUs which are originally developed to deal with accelerating graphical and video applications. GPUs are also designed to deal with non-graphical computations (i.e., General-Purpose Graphics Processing Unit (GPGPU)).

In this paper, we implement the AES algorithm using GPU taking into consideration different granularity values of 32, 64, and 128 Bytes/thread seeking to increase the AES performance (i.e., throughput). The implementations of the AES are performed on three different GPU architectures: Kepler (Nvidia GTX 780), Maxwell (Nvidia GTX TITAN X), and Pascal (Nvidia GTX 1080) using various input block sizes with random plain text.

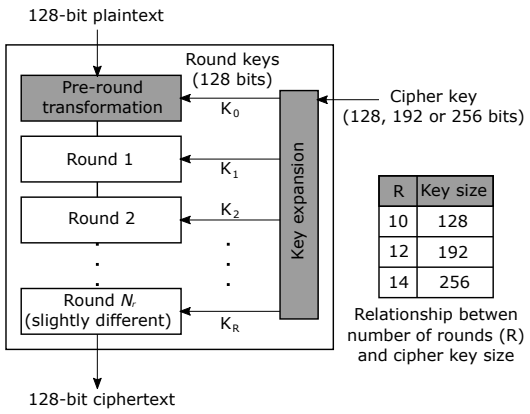

Fig. 1: The overall structure of the AES algorithm [1].

2. The AES Algorithm Mechanism

In this section, the overall structure of the AES encryption algorithm is overviewed with three cipher im-

plementations: AES-128, AES-192, and AES-256 as shown in Fig. 1. All of those implementations use a 128-bit block input, however, with a key size of 128 bits, 192 bits, and 256 bits, respectively. The AES encryption algorithm consists of 10 rounds using the 128-bit key, 12 rounds using the 192-bit key, and 14 rounds using the 256-bit key. Each of these rounds uses a different 128-bit round key, which is evaluated from the original AES key using the key expansion technique. The AES algorithm basically consists of two steps: key expansion and round transformations [1], as presented in Subsec. 2.1. and Subsec. 2.2. , respectively.

2.1. Key Expansion Step

The AES key expansion step is used to expand the cipher key from 128, 192, or 256 bits to 1280 bits (10 round-keys), 1472 bits (12 round-keys) or 1664 bits (14 round-keys), respectively. The round keys are used in the AddRoundKey transformation [1]. It consists of three tasks:

- **Substitute Word (SubWord)**: takes a 32-bit word a four-Byte as input and makes a substitution based on the S-Box for each of the four Bytes to produce an output word.
- **Rotate Word (RotWord)**: takes a 32-bit word and makes one cyclic left rotation to produce an output word.
- **Round constant (Rcon)**: XOR 32-bit word with the round constant (Rcon).

2.2. Round Transformations Step

The input plain text data are divided into 16-Byte Blocks arranged in a 4×4 column-majored array called state. The AES algorithm has four different kinds of layers.

- **Add round key layer**: is a bitwise XOR operation between each column in the state and the corresponding round key from the key expansion.
- **Bytes substitution layer (using S-box)**: is a non-linear substitution, where each Byte in the state is independently substituted according to a given substitution table (i.e., S-Box). The first four bits of the Byte are used to index the S-Box rows, while the second four bits are used to index the S-Box columns.
- **ShiftRows layer**: rotates the last three rows in the state to the left. The result is a new matrix consisting of the same 16 Bytes, but shifted with respect to each other.
- **MixColumns layer**: operates on each column in the state by using Galois Field (GF) math. This is done by treating every column as a four-term polynomial over GF $(2 \cdot 8)$. The result is a new matrix consisting of new 16 Bytes.

2.3. AES Modes of Operation

In Tab. 1, six modes of the AES encryption algorithm are presented providing different levels of security and parallelism Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), Counter (CTR), and XEX-based tweaked-codebook mode with ciphertext stealing (XTS) [1]. In this paper, we implemented a CUDA-based AES algorithm using the ECB mode as it could be implemented in a parallel manner. Given the ECB mode, the message is divided into blocks, then each block can be encrypted separately.

Tab. 1: AES Modes of operation [6].

Mode	Description	Parallelization potential
Electronic Codebook (ECB)	For a given key, the forward cipher function is applied directly and independently to each block of the plaintext.	Suitablefor parallelization
Cipher Block Chaining (CBC)	Each successive plaintext block is exclusive-ORed with the previous output/ciphertext block to produce the new input block. The forward cipher function is applied to each input block to produce the ciphertext block.	Decryption is suitable for parallelization
Cipher Feedback (CFB)	The feedback of successive ciphertext segments is input to the forward cipher blocks to generate output blocks that are exclusive-ORed with the plaintext to produce a new ciphertext and vice versa.	Not suitable for parallelization
Output Feedback (OFB)	The iteration of the forward cipher on an IV to generate a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa.	Not suitable for parallelization
Counter (CTR)	The application of the forward cipher to a set of input blocks, called counters, to produce a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa.	Suitable for parallelization
XEX-based tweaked-codebook mode with ciphertext stealing (XTS)	IEEE standard, IEEE Std 1619-2007, which a method of encryption for data stored in sector-based devices	Suitable for parallelization

2.4. The Fast AES Implementation Algorithm

In the fast AES algorithm, the lookup table solution [7] and [8] could be replaced with four versions of the AES round transformation step. In this fast algorithm, the four lookup tables are defined as: T_0, T_1, T_2, and T_3. Each table accepts one Byte of input, and comes out with a 32-bit column vector. The operations of each

round transformation can be defined as follows:

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{0,j}] \oplus T_2[a_{0,j}] \oplus T_3[a_{0,j}] \oplus K_j, \quad (1)$$

where $a_{0,j}$ represents the round input, K_j is one column of the stage key, and e_j denotes one column of the round output in terms of Bytes of $a_{0,j}$. T_0, T_1, T_2, and T_3 refer to the lookup tables which have 256 of 32-bit word entries each. Each table needs 1 KBytes of storage space.

3. Implementation Techniques on GPU Platform

This section presents the traditional implementations of the AES algorithm on GPU platforms. The GPU-based implementations of the AES algorithm can be divided into three major categories: memory optimization, parallel granularity, and GPU platform specific optimization [9], [10] and [11].

3.1. Memory Optimization

In this subsection, we will deal with two parameters: (Lookup Tables) lookup tables and (Encryption Keys Storage) Encryption keys storage.

- **Lookup Tables**: need 4 KB of storage space. The best choice is to store (T-box) in the shared memory as it has high access speed. It is allocated per thread block, so all threads in the block have access to the same shared memory [9].
- **Encryption Keys Storage**: The AES performance can be also further enhanced if the encryption keys are first computed inside the CPU. Then, those keys are stored in the GPU global memory. When the GPU kernel is launched, each thread in a warp copies key value(s) from the global memory and stores it in two registers. This process is known as warp shuffle [9]. There are 32 threads in a warp, so only 64 keys can be hold in total. Using this strategy, all encryption keys can be saved in registers with higher access speed.

3.2. Parallel Granularity

In the parallel processing concept, many threads can be assigned to perform one process for speeding up the functionality, thus reducing the execution time. In the AES algorithm, the default number of blocks needed to be encrypted (i.e., granularity) per one thread is one block (i.e., 16 Bytes) as shown in Tab. 3 [9], [12], [13], [14], [15], [16] and [17]. Having the parallel processing concept been embedded into the AES algorithm, the number of data blocks encrypted using one thread can be further increased to 2, 4, or 8 blocks. This is called parallel granularity of the AES algorithm. In this paper, we examine new parallel granularities that have never been used in the literature, such as 32 Bytes/thread (i.e., two data blocks), 64 Bytes/thread (i.e., four data blocks), and 128 Bytes/thread (i.e., eight data blocks).

3.3. GPU Platform Specific Optimization

Each new GPU architecture has its own hardware specifications that are different in design from previous architectures in order to enhance the overall performance. The GPU occupancy is a measure of thread parallelism in a CUDA program. The higher the occupancy, the higher the performance (i.e., throughput). The field thread block size affects the GPU occupancy that depends on the GPU architecture. For example, given the Maxwell GPU SM 5.2, the thread block size that satisfies 100 % occupancy is set to 128, 256, 512, and 1024 as shown in Tab. 2. However, in this paper, the thread block size is set to 128 threads per block.

Tab. 2: Occupancy in GPU SM 5.2 [9].

Thread Block Size (TBS)	Total blocks (2048/TBS)	Occupancy (active blocks/ total blocks)·100
64	32	75 %
128	16	100 %
256	8	100 %
512	4	100 %
1024	2	100 %

4. Related Work

There are several researches showing implementing the AES algorithm using the GPU architectures as mentioned above with CUDA language. Table 3 summarizes recent implementations of CUDA-based AES-128 ECB on GPU. Mei et al. in [12] proposed an efficient approach to parallelize AES and fine-tune the memory utilization in GeForce 9200M GS. Their best performance is 51.2 Gbps throughput rate using 16 Bytes/thread granularity.

N. Nishikawa et al. in [14] implemented AES on CUDA and studied the following computation granularities: 16 Bytes/thread, 8 Bytes/thread, 4 Byte/thread and 1 Byte/thread. Their best performance is 35.2 Gbps throughput rate using 16 Bytes/thread granularity. Zola et al. [16] proposed a speculative AES-CTR scheme. The proposed scheme

Tab. 3: Summary of AES implementations on different GPU architectures.

References	AES (Gbps)	Mode	GPU device	Architecture	Round keys storage	Parallel granularity	Year
Mei et al. [12]	51.2	Unknown	GS9200M	Tesla	Shared memory	Unknown	2010
Bos et al. [18]	59.6	Unknown	GTX295	Tesla	Shared memory	Unknown	2010
Zola et al. [16]	72.0	CTR	GTX260	Tesla	Shared memory	16B/thread	2012
Nishikawa et al. [19]	48.6	ECB	Tesla C2050	Fermi	Shared memory	16B/thread	2012
Li et al. [13]	60.0	ECB	Tesla C2050	Fermi	Shared memory	16B/thread	2012
Nishikawa et al. [15]	68.6	ECB	GTX680	Kepler	Shared memory	16B/thread	2014
Lee et al. [9]	149.5	CTR	GTX980	Maxwell	warp shuffle approach	16B/thread	2016

Tab. 4: 32, 64 and 128 granularities effect compared with 16 granularity.

Granularities	Explaination	Total number of threads run	Processing functionality per thread
16 Bytes/thread	One thread will encrypt one AES block	x	y
32 Bytes/thread	One thread will encrypt two AES blocks	x / 2	y * 2
64 Bytes/thread	One thread will encrypt four AES blocks	x / 4	y * 4
128 Bytes/thread	One thread will encrypt eight AES blocks	x / 8	y * 8

encrypts the counter blocks processes in GPU plus CPU with throughput of 72.0 Gbps.

Li et al. in [13] stored T-boxes on on-chip shared memory. Moreover, the granularity where one thread handles a 16 Bytes AES block was adopted, yielding throughput of 60 Gbps on NVIDIA Tesla C2050 GPU. Nishikawa et al. in [15] presented implementation of block ciphers in NVIDIA and AMD GPU based on Geforce GTX 680 architecture with throughput of 68.6 Gbps.

Wai-Kong Lee et al. in [9] presented implementation of AES in NVIDIA GTX 980 with Maxwell architecture and utilized the advanced features warp shuffle approach to further accelerate the performance. Although, all aforementioned approaches have used different GPU architectures to speedup the execution time, the granularity is still limited to 16 Bytes/thread (i.e., one block per thread). In this paper, we examine all available granularities on various GPU architectures to achieve the optimized settings for each architecture.

5. Proposed AES Algorithm

We implemented the AES algorithm with two techniques of optimization as mentioned in Sec. 3. The proposed parallel granularities used and the encrypted key storage are explained in Subsec. 5.1. and Subsec. 5.2. , respectively.

5.1. Parallel Granularity

There is a trade-off between the number of data blocks needed to encrypted by one thread (i.e., parallel granularity), and the processing functionality of each thread that negatively affects the AES performance according to the specifications of the GPU used. We propose larger granularities (32, 64, and 128 Bytes/thread) in order to encrypt more than one data block instead of only one as shown in Tab. 4 compared to conventional granularity of 16 Bytes/thread.

5.2. Encrypted Keys Storage

We used three different techniques for storing the AES encryption keys inside GPU in order to show its effects using different parallel granularities.

- Shared memory storage: the encryption keys are stored in the shared memory inside the GPU.
- Warp shuffle storage: the encryption keys are stored in the CPU registers and can be accessed using the warp shuffle feature.
- Global memory storage: the encryption keys are stored in the global memory inside the GPU.

6. Implementation Setup and Performance Evaluation

- **GPU platforms**: We use three different platforms supporting three different GPU architec-

tures (Kepler, Maxwell, and Pascal) as shown in Tab. 5.

- **Implementation**: All AES implementations, used in our experiments, are performed using the CUDA Toolkit 7.5 on Linux (i.e., Ubuntu 14.04). The proposed approach is performed on the Pascal GPU (i.e., NVIDIA GTX 1080).
- **Competing approaches**: For fair comparison, the proposed approach is compared to the approach shown by Nishikawa et al. [15], on the Kepler GPU (i.e., NVIDIA GTX 780), and that shown by Lee [9] on the Maxwell GPU (i.e., NVIDIA GTX TITAN X).
- **Parallel granularity values**: The granularity is set to different values (i.e., 16, 32, 64, and 128) in all competing approaches using three different GPU architectures as shown in Tab. 5 for a thorough analysis.
- **Performance evaluation**: the performance of all competing approaches is evaluated by the throughput metric, in Giga bits per second (Gbps), on the basis of the higher the better.

Tab. 5: Configuration of the three GPU platforms.

Platform	GPU	CPU
1	NVIDIA GTX 780 CUDA Cores =2880 Architecture: Kepler	Intel Xeon E5-2640 v2 Total: 8 Cores
2	NVIDIA GTX TITAN X CUDA Cores =3072 Architecture: Maxwell	Intel Xeon E5-2640 v2 Total: 8 Cores
3	NVIDIA GTX 1080 CUDA Cores =2560 Architecture: Pascal	Intel Xeon E5-2640 v2 Total: 8 Cores

7. Experiments and Results

In our experiments, we focus on computing the kernel time in the GPU that exempts the data transfer time between both CPU and GPU. We repeated the same experiment with a specific setting on a particular GPU for 30 times and took the average throughput. Four different charts are presented to show our results for a specific GPU architecture. In all charts, as mentioned in Tab. 6, we used different granularities with a specific encryption keys storage.

7.1. Experimental Results on Kepler Platform

We evaluated the average throughput for different input block sizes (16 MBytes to 512 MBytes) as follows:

Tab. 6: Four different charts depending on two techniques of optimization.

Chart	Explanation
Shared memory-based	Encryption keys are stored in shared memory
Warp shuffle	Encryption keys are stored using Warp shuffle
Global memory-based	Encryption keys are stored in Global memory
All-in-one	Combine previous charts in one chart to conclude the best throghput

- Shared memory-based chart, in Fig. 2(a), shows that the default granularity (i.e., 16 Bytes/thread) provides a higher average throughput compared to other granularities.
- Warp Shuffle chart, in Fig. 2(b), shows that the default granularity (i.e., 16 Bytes/thread) provides a higher average throughput compared to other granularities.
- Global memory-based chart, in Fig. 2(c), shows that the parallel granularity (i.e., 32 Bytes/thread) provides a higher average throughput compared to other granularities.
- All-in-one chart, in Fig. 3, shows that the default granularity (i.e., 16 Bytes/thread) with the shared memory key storage provides a higher average throughput of 78 Gbps.

7.2. Comparison Between GPU-Based and CPU-Based Implementations

7.3. Experimental Results on Maxwell Platform

We evaluated the average throughput for different input block sizes (16 MBytes to 512 MBytes) as follows:

- Shared memory-based chart, in Fig. 4(a), shows that the parallel granularity (i.e., 32 Bytes/thread) provides a higher average throughput compared to other granularities.
- Warp Shuffle chart, in Fig. 4(b), shows shows that the parallel granularity (i.e., 32 Bytes/thread) provides a higher average throughput compared to other granularities.
- Global memory-based chart, in Fig. 4(c), shows that the parallel granularity (i.e., 64 Bytes/thread) provides a higher average throughput compared to other granularities at most input file sizes. However, in case of setting

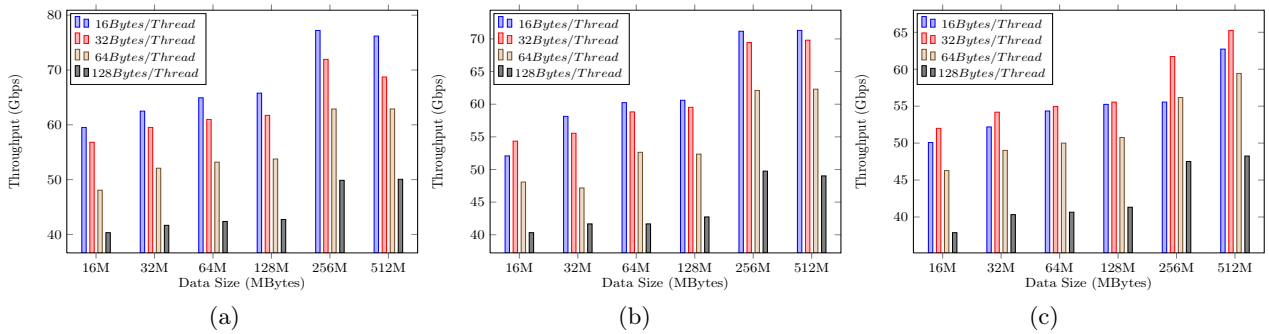

Fig. 2: (a), (b) and (c) are the shared memory-based, warp shuffle-based, and global memory-based charts, respectively, using the NVIDIA GTX 780 (Kepler GPU).

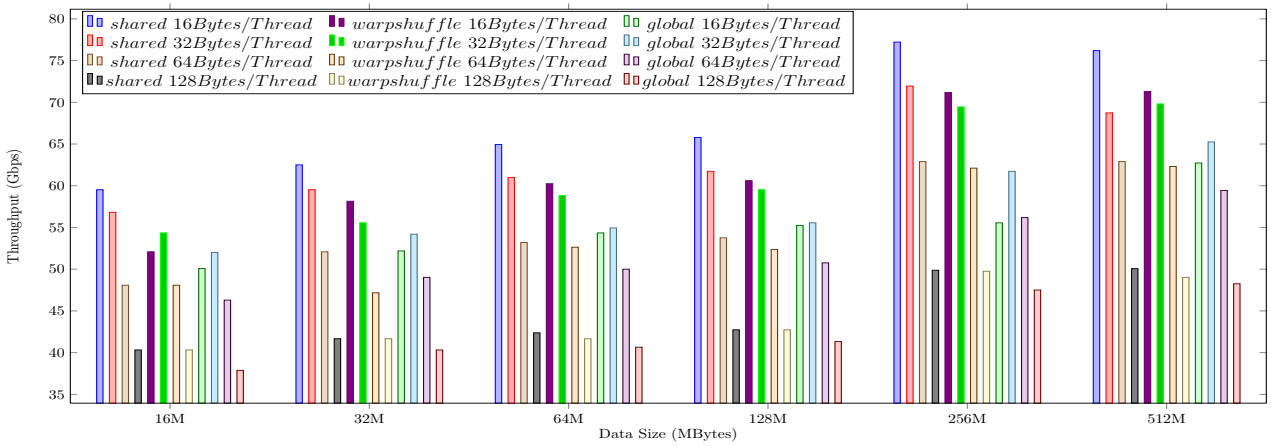

Fig. 3: All-in-one approach for NVIDIA GTX 780 (Kepler).

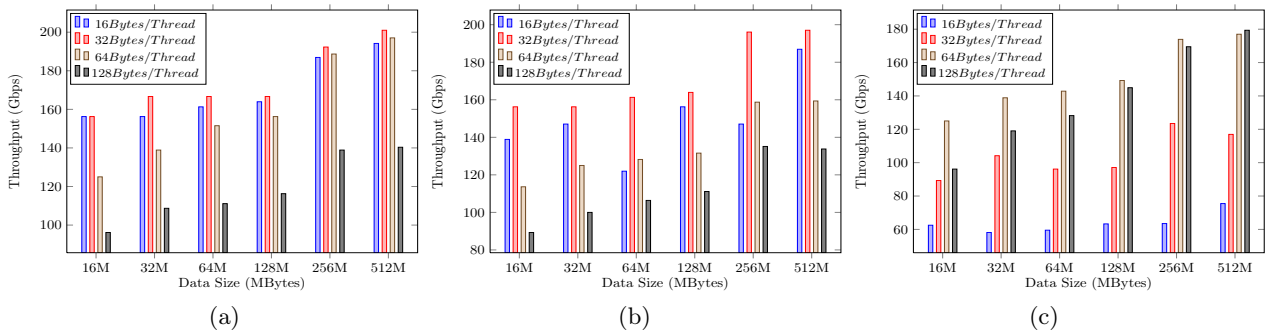

Fig. 4: (a), (b) and (c) are the shared memory-based, warp shuffle-based, and global memory-based charts, respectively, using the NVIDIA GTX TITAN X (Maxwell GPU).

the input file size to 512M, the new parallel granularity (i.e., 128 Bytes/thread) provides a higher average throughput.

- All-in-one chart, in Fig. 5, shows that the parallel granularity (i.e., 32 Bytes/thread) with the shared memory key storage provides a higher average throughput of 201 Gbps. However, in case of setting the input file size to 256M, the parallel granularity (i.e., 32 Bytes/thread) with the warp shuffle memory storage provides a higher average throughput.

7.4. Experimental Results on Pascal Platform

We evaluated the average throughput for different input block sizes (16 MBytes to 512 MBytes) as follows:

- Shared memory-based chart, in Fig. 6(a), shows that the parallel granularity (i.e., 32 Bytes/thread) provides a higher average throughput compared to other granularities.
- Warp shuffle chart, in Fig. 6(b), shows that the parallel granularity (i.e., 32 Bytes/thread) pro-

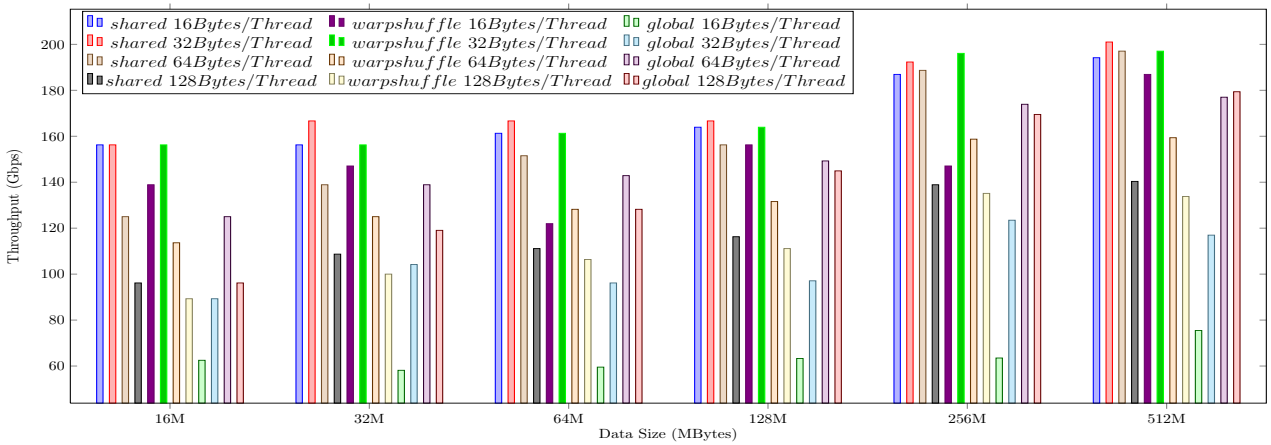

Fig. 5: All-in-one chart for NVIDIA GTX TITAN X (Maxwell).

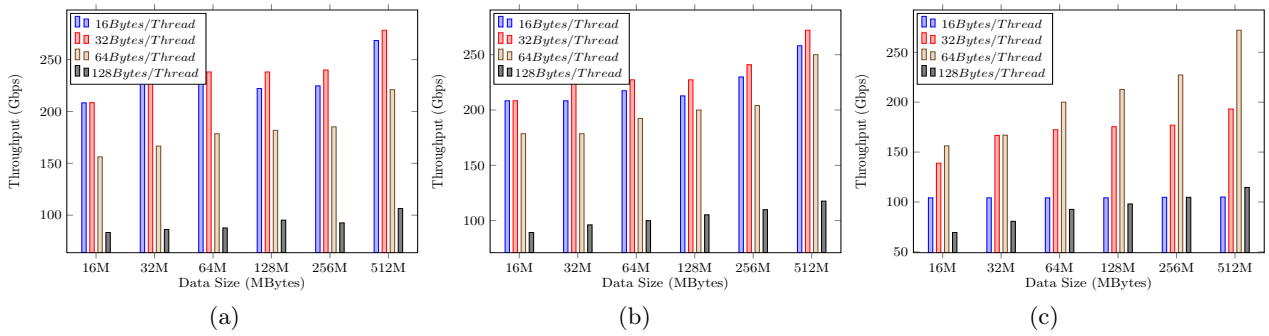

Fig. 6: (a), (b) and (c) are the shared memory-based, warp shuffle-based, and global memory-based charts, respectively, using the NVIDIA GTX 1080 (Pascal GPU).

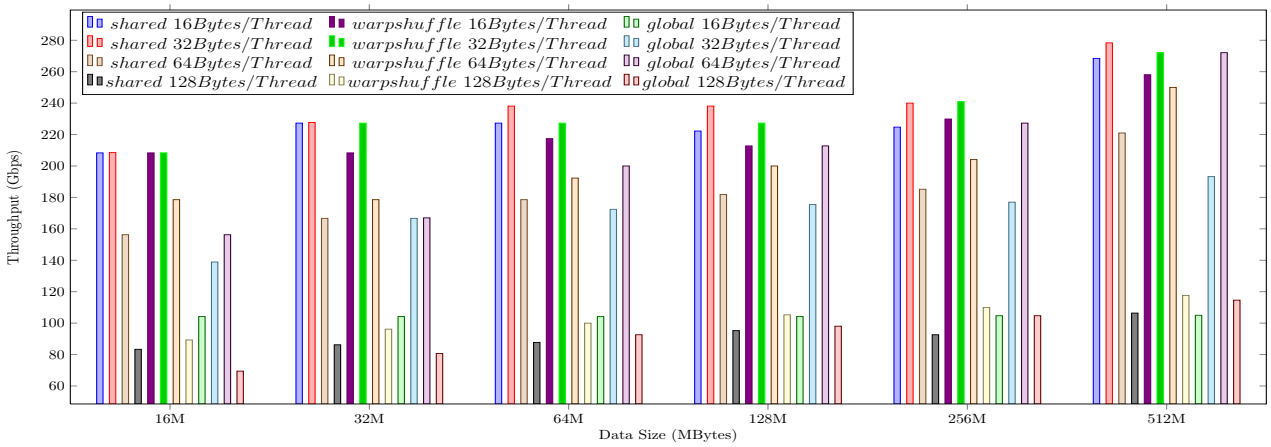

Fig. 7: All-in-one chart for NVIDIA GTX 1080 (pascal).

vides a higher average throughput compared to other granularities.

- Global memory-based chart, in Fig. 6(c), shows that the parallel granularity (i.e., 64 Bytes/thread) provides a higher average throughput compared to other granularities.

- All-in-one chart, in Fig. 7, shows that the parallel granularity (i.e., 32 Bytes/thread) with the shared memory key storage provides a higher average throughput of 276 Gbps.

The speedup-factor of using the GPU architecture over the CPU is needed to be determined. In case of using the CPU, the AES algorithm is implemented on an 8-core dual processor. In case of using the GPU, three different GPU architectures are exploited. The speedup-factor is determined as 40 on the Kepler GPU, 100 on the Maxwell GPU, and 130 on the Pascal GPU

at a specific input file size. Note that the speedup-factor increases as the input block size increases. Table 7 shows the speedup-factor obtained on the three different GPU architectures over the CPU implementation with different input file sizes.

Tab. 7: Speedup-factor for implementing the AES algorithm on the Kepler, Maxwell, and Pascal GPUs compared to that on the CPU.

Input size	Speed up factor = CPU time / GPU time		
	GTX 780 (Kepler)	GTX TITAN X (Maxwell)	GTX 1080 (Pascal)
1 M	20.200	60.590	60.600
2 M	26.989	72.776	87.570
4 M	29.862	73.898	88.705
8 M	28.994	78.488	97.984
16 M	29.796	79.291	99.055
32 M	29.792	79.988	99.955
64 M	30.987	81.510	107.040
128 M	31.824	81.599	109.540
256 M	33.885	96.124	129.843
512 M	37.456	98.517	130.992

8. Analysis and Discussion

We presented four parallel AES charts using the parallel granularity and round keys storage to achieve a higher performance (i.e., throughput). In this section, we can analyze our experimental results mentioned above.

8.1. Parallel Granularity

The new 32, 64, and 128 granularities affect the AES performance depending on the GPU architecture used as follows:

1) The Kepler GPU

- The default granularity(i.e., 16 Bytes/thread) provides a higher throughput of the AES algorithm compared to other granularities with the shared memory-based, warp shuffle-based, and All-in-one charts.
- The 32 Bytes/thread granularity outperforms other granularities with the global memory-based storage. Nevertheless, it didn't achieve as good results for the AES. It can be a good alternative in optimizing another algorithm using the Kepler GPU.

2) Maxwell and Pascal GPUs

- The 32 Bytes/thread granularity provides a higher throughput of the AES algorithm compared to the default granularity (i.e., 16 Bytes/thread) with the shared memory-based, warp shuffle-based, and All-in-one charts.
- The 64 Bytes/thread granularity outperforms other granularities with the global memory-based storage. Nevertheless, it didn't achieve as good results for the AES. It can be also a good alternative in optimizing another algorithm.

As aforementioned in Sec. 5. , there is a trade-off between the number of data blocks to be encrypted by one thread, and the processing functionality of each thread according to the specifications of the GPU used. In recent GPU architectures, such as Maxwell and Pascal, both the number of Streaming Multiprocessors (SMs), and the number of active blocks per multiprocessors are largely increased compared to those of the Kepler GPU. In turn, reducing the number of threads by using higher granularities overcomes the issue of increasing the processing functionality of each thread, thus increasing the AES performance.

8.2. Round Keys Storage

Although the registers are faster than the shared memory inside the GPU, using the latter provides a higher AES throughput compared to the warp shuffle-based storage in all GPU platforms used (i.e., Kepler, Maxwell, and Pascal). This is because storing the round keys in two registers as well as accessing them using 32 threads inside a warp results in higher race conditions that negatively impact the AES performance.

9. Conclusions

We implemented the AES with CUDA language considering the parallel granularity with different round keys storage techniques to eventually increase the AES performance (i.e., throughput). The AES-128 is implemented with different parallel granularities on different GPU architectures compared to the CPU implementation. The proposed approach achieves throughput of 277 Gbps, 201 Gbps, and 78 Gbps on the Pascal GPU, the Maxwell GPU, and the Kepler GPU with granularity values of 32, 32, and 16 Bytes/thread, respectively. In addition, the speedup factor of implementing the AES algorithm using those GPUs are 130.992, 98.517, and 37.456, respectively, at 512 MBytes input file size with the best granularity value mentioned shortly.

References

[1] Federal Information Processing Standards Publication 180-2. *Secure Hash Standard*. Geneva: National Institute of Standards and Technology, 2002.

[2] HARRISON, O. and J. WALDRON. AES encryption implementation and analysis on commodity graphics processing units. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Berlin: Springer, 2007, pp. 209–226. ISBN 978-3-540-74735-2. DOI: 10.1007/978-3-540-74735-2.

[3] CHU, X., K. ZHAO and M. WANG. Massively parallel network coding on GPUs. In: *Performance, Computing and Communications Conference*. Austin: IEEE, 2008, pp. 144–151. ISBN 978-1-4244-3368-1. DOI: 10.1109/PCCC.2008.4745113.

[4] MANAVSKI, S. A. CUDA compatible GPU as an efficient hardware accelerator for AES Cryptography. In: *IEEE International Conference on Signal Processing and Communications*. Dubai: IEEE, 2007, pp. 65–68. ISBN 978-1-4244-1235-8. DOI: 10.1109/ICSPC.2007.4728256.

[5] HARRISON, O. and J. WALDRON. Practical symmetric key cryptography on modern graphics hardware. In: *Proceedings of the 17th conference on Security symposium*. San Jose: USENIX Association Berkeley, 2008, pp. 195–209. ISBN 978-1-931971-03-4.

[6] LI, Q., C. ZHONG, K. ZHAO, X. MEI and X. CHU. Implementation and analysis of AES encryption on GPU. In: *IEEE 14th International Conference on High Performance Computing and Communication and 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*. Washington: IEEE, 2012, pp. 843–848. ISBN 978-0-7695-4749-7. DOI: 10.1109/HPCC.2012.119.

[7] MCLOONE, M. and J. V. MCCANNY. Rijndael fpga implementations utilising look-up tables. *Journal of VLSI signal processing systems for signal, image and video technology*. 2003, vol. 34, no. 3, pp. 261–275. ISSN 0922-5773. DOI: 10.1023/A:1023252403567.

[8] MANAVSKI, S. A. CUDA compatible GPU as an efficient hardware accelerator for AES Cryptography. In: *IEEE International Conference on Signal Processing and Communications ICSPC 2007*. Dubai: IEEE, 2007, pp. 65–68. ISBN 978-1-4244-1235-8. DOI: 10.1109/ICSPC.2007.4728256.

[9] LEE, W.-K., H.-S. CHEONG, R. C.-W. PHAN and B.-M. GOI. Fast implementation of block ciphers and PRNGs in Maxwell GPU architecture. *Cluster Computing*. 2016, vol. 19, no. 1, pp. 335–347. ISSN 1386-7857. DOI: 10.1007/s10586-016-0536-2.

[10] BIAGIO, A. D., A. BARENGHI, G. AGOSTA and G. PELOSI. Design of a parallel AES for graphics hardware using the CUDA framework. In: *IEEE International Symposium on Parallel and Distributed Processing*. Rome: IEEE, 2009, pp. 1–8. ISBN 978-1-4244-3751-1. DOI: 10.1109/IPDPS.2009.5161242.

[11] TRAN, N.-P., M. LEE, S. HONG and S.-J. LEE. Parallel execution of AES-CTR algorithm using extended block size. In: *14th International Conference on Computational Science and Engineering (CSE)*. Dalian: IEEE, 2011, pp. 191–198. ISBN 978-1-4577-0974-6. DOI: 10.1109/CSE.2011.43.

[12] MEI, C., H. JIANG and J. JENNESS. CUDA-based AES parallelization with fine-tuned GPU memory utilization. In: *IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW)*. Atlanta: IEEE, 2010, pp. 1–7. ISBN 978-1-4244-6533-0. DOI: 10.1109/IPDPSW.2010.5470766.

[13] LI, Q., C. ZHONG, K. ZHAO, X. MEI and X. CHU. Implementation and Analysis of AES Encryption on GPU. In: *14th International Conference on High Performance Computing and Communication and 9th International Conference on Embedded Software and Systems*. Washington: IEEE, 2012, pp. 843–848. ISBN 978-0-7695-4749-7. DOI: 10.1109/HPCC.2012.119.

[14] IWAI, K., T. KUROKAWA and N. NISIKAWA. Aes encryption implementation on cuda gpu and its analysis. In: *First International Conference on Networking and Computing (ICNC)*. Higashi-Hiroshima: IEEE, 2010, pp. 209–214. ISBN 978-1-4244-8918-3. DOI: 10.1109/IC-NC.2010.49.

[15] NISHIKAWA, N., I. KEISUKE, H. TANAKA and T. KUROKAWA. Throughput and power effciency evaluation of block ciphers on Kepler and GCN GPUs using micro-benchmark analysis. *IEICE Transactions on Information and Systems*. 2014, vol. 97, no. 6, pp. 1506–1515. ISSN 0916-8532. DOI: 10.1587/transinf.E97.D.1506.

[16] ZOLA, W. M. N. and L. C. E. DE BONA. Parallel speculative encryption of multiple aes contexts on gpus. In: *Innovative Parallel Computing (InPar)*. San Jose: IEEE, 2012, pp. 1–9. ISBN 978-1-4673-2631-5. DOI: 10.1109/InPar.2012.6339611.

[17] CONTI, V. and S. VITABILE. Design exploration of aes accelerators on fpgas and gpus. *Journal of Telecommunications and Information Technology.* 2017, vol. 2017, no. 1, pp. 1–11. ISSN 1509-4553.

[18] OSVIK, D. A., J. W. BOS, D. STEFAN and D. CANRIGHT. Fast software AES encryption. In: *International Workshop on Fast Software Encryption.* Berlin: Springer, 2010, pp. 75–93. ISBN 978-3-642-13857-7. DOI: 10.1007/978-3-642-13858-4_5.

[19] NISHIKAWA, N., K. IWAI and T. KUROKAWA. High-Performance Symmetric Block Ciphers on Multicore CPU and GPUs. *International Journal of Networking and Computing.* 2012, vol. 2, no. 2, pp. 251–268. ISSN 2185-2839. DOI: 10.15803/ijnc.2.2_251.

About Authors

Ahmed Awadalla ABDELRAHMAN received the B.Sc. in Computer Engineering from the MTC, in 2012. Now, he is pursing his Masters' degree at the Computer Engineering department of the Military Technical College (MTC). His research interests are in implementing highly computational cryptographic algorithms on GPUs, such as the AES, blue-fish and two-fish encryption algorithms.

Mohamed Mahmoud FOUAD received the Bachelor engineering (honors, with great distinction) and Masters' engineering degrees from the Military Technical College (MTC), Cairo, Egypt, in 1996 and 2001, respectively. As well, he received the Ph.D. degree in Electrical and Computer Engineering from Carleton University, Ottawa, Canada in 2010. He is currently an Associate Professor with the Department of Computer Engineering at the Military Technical College (MTC). His research interests are in online handwritten recognition, image processing, multi-view video coding, video compression and implementing GPU-based applications.

Hisham Mohamed DAHSHAN received the Bachelor engineering and Masters' engineering degrees from the Military Technical College (MTC), Cairo, Egypt, in 1994 and 2005, respectively. As well, he received the Ph.D. degree in Electrical Engineering from Strathclyde University, Glasgow, U.K. in 2010. He is currently an Associate Professor with the Department of Communication at the Military Technical College (MTC). His research interests are in wireless network security, Internet of things, software-defined network, and cryptography.