High Performance CUDA AES Implementation: A Quantitative Performance Analysis Approach

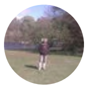
Conference Paper · July 2017

DOI: 10.1109/SAI.2017.8252225

4 authors:

Ahmed Awadallah
Military Technical College
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Mohamed M. Fouad
Military Technical College
28 PUBLICATIONS **80** CITATIONS
SEE PROFILE


Hisham Dahshan
Military Technical College
18 PUBLICATIONS **83** CITATIONS
SEE PROFILE

Ahmed Magdy Mousa
The American University in Cairo
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Computing Conference 2017
18-20 July 2017 | London, UK

High Performance CUDA AES Implementation: A Quantitative Performance Analysis Approach

Ahmed A. Abdelrahman[1], Mohamed M. Fouad[1]
and Hisham Dahshan[2]
Dept. of Computer Engineering[1],
Dept. of Communication Engineering[2]
Military Technical College
Cairo, Egypt
Email: {ahmedsoliman, mmafoad, hdahshan}@mtc.edu.eg

Ahmed M. Mousa
Dept. of Computer Science
American University in Cairo
Cairo, Egypt
Email: ahmed.helal@aucegypt.edu

***Abstract*—The importance of cryptography on ensuring security or integrity of the electronic data transaction had become higher during the past few years. Multiple security protocols are currently using various block ciphers. One of the most widely used block ciphers is the Advanced Encryption Standard (AES) which is chosen as a standard for its higher efficiency and stronger security than its competitors. Unfortunately, the encryption and decryption processes of AES takes a considerable amount of time for large data size. The GPU is an attractive platform for accelerating block ciphers and other cryptography algorithms due to its massively parallel processing power. In this work, an implementation of the AES-128 ECB Encryption on three different GPU architectures (Kepler, Maxwell and Pascal) has been presented. The results show that encryption speeds with 207 Gbps on the NVIDIA GTX TITAN X (Maxwell) and 280 Gbps on the NVIDIA GTX 1080 (Pascal) have been achieved by performing new optimization techniques using 32bytes/thread granularity.**

Keywords*—*AES; ECB; CUDA; GPU; Rijndael; Throughput; Granularity; Performance optimization; Encryption

I. Introduction

Today's web communication and cloud computing requires more secure data communications than ever before and as a result, block ciphers cryptography becomes more important day by day. One of the most widely used block ciphers is the Advanced Encryption Standard (AES) [1] which is chosen as a standard for higher efficiency and stronger security than its competitors. Since AES algorithm consists of a large amount of homogeneous computations, the use of hardware accelerators such as GPUs has been concluded to be one of the most important methods to achieve high-speed data encryption technology.

Over the past few years, the GPU has evolved into powerful parallel computing devices with a high cost-performance ratio. Although it was originally developed only to accelerate graphic applications, the GPUs has been researched as an efficient hardware accelerator for non-graphical general computations. Thus, GPU has been addressed since then as GPGPUs (General-Purpose Computation on Graphics Processing Unit).

In this paper, multiple optimization techniques has been exploited to achieve higher throughput and speedup for the AES algorithm on three different GPU architectures: Kepler (Nvidia GTX 780), Maxwell (Nvidia GTX TITAN X) and Pascal (Nvidia GTX 1080).

GPUs performance is affected severely by memory accesses especially in block ciphers because of the high load of accessing and writing data, thus it was essential to exhaust many possible approaches of key storage and the way its accessed. New computational and parallel granularities have also been experimented and placed in comparison with the latest previous work. Also, different sizes of input (random plaintext) has been tested and analyzed for their effect on CUDA AES-128 ECB encryption performance has been introduced.

The rest of the paper is organized as follows. Section II presents a detailed description of the implemented AES algorithm. Different implementation techniques of AES using GPU is presented in Section III . Related work is introduced in Section IV. Section V explains our proposed work. The experimental results on different platforms are presented and analyzed in Section VI. Finally, Section VII concludes this work and discusses future research directions.

II. Overview of AES Algorithm

AES is a block cipher established by National Institute of Standards (NIST) based on the Rijndeal cipher [1] and adopted for replacing the Data Encryption Standard (DES). AES can currently encrypt blocks of 16 bytes at a time. If the number of bytes being encrypted is larger than the specified block length, then AES is executed concurrently. Interestingly, AES executes all its computations on bytes rather than bits. Hence, AES 128 bit block of plain-text is treated as 16 bytes, which are arranged in four columns and four rows for processing as a matrix called "state". In AES a state defines the current condition of the block. The operations of AES depends on a variable called number of rounds (Nr) which is fixed for each key size. AES Encryption consists of 10 rounds for 128-bit key, 12 rounds for 192-bit key and 14 rounds for 256-bit key. Each of these rounds uses a different 128-bit round key, Which is calculated from the original AES key as shown in Fig. 1.

A. *Operation of AES*

The AES-128 algorithm basically consists of two phases: Key expansion and Transformation rounds. The next sections will explain them briefly.

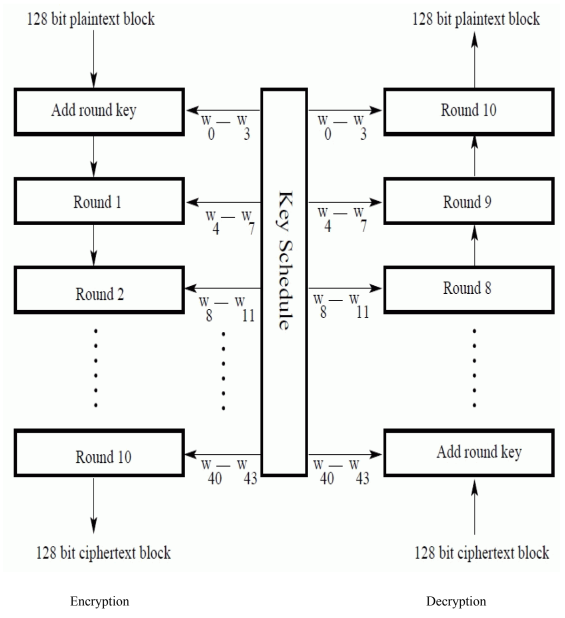


Fig. 1: General AES Architecture [1]

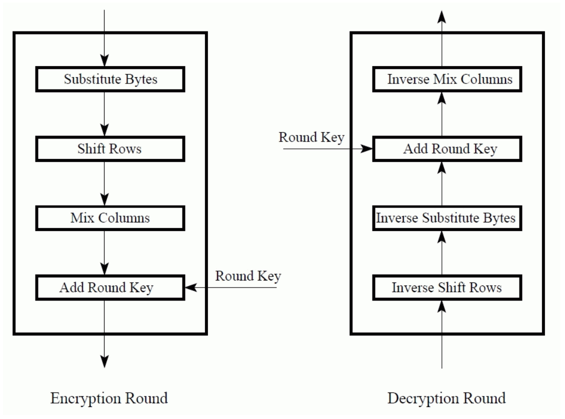


Fig. 2: AES Round Structure [1]

1) Key expansion: The AES algorithm takes the Cipher Key and performs a key scheduling algorithm to generate round keys [1]. It allows for the full key expansion to precede the round transformations, which is better if multiple blocks are encrypted using the same key.

2) Transformation Rounds: Input plaintext data to be divided into 16 Byte Blocks arranged in a 4x4 column-majored array state. This also means that AES has to encrypt a minimum of 16 bytes. If the plain text is smaller than 16 bytes then it must be padded. Each round consists of four processing steps: AddRoundKey, SubBytes, ShiftRowsm and MixColumns. Note that the initial round only includes AddRoundKey step that depends on the encryption key, while the last round does not include MixColumns step as shown in Fig.2.

2.1 SubBytes: The 16 input bytes are substituted by looking up a fixed 2D 15x15 Substitution Box (S-box) which is pre-given. The result is a matrix of four rows and four columns. The first four bits of the byte are used to index the row, while the second four bits used to index columns of the S-Box.

TABLE I: AES Modes of Operation [2]

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 into the input blocks of the forward cipher to generate output blocks that are exclusive-ORed with the plaintext to produce the 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

2.2 ShiftRows: Each row goes through cyclic shifts by differing offsets. First row is not shifted, Second row is shifted one byte position to the left, third row is shifted two positions to the left, fourth row is shifted three positions to the left. The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

2.3 MixColumns: Each column of four bytes is now transformed using a special linear transformation represented by the polynomial c(x) given by:[1]:

$$C(X) = \{03\}X^3 + \{01\}X^2 + \{01\}X + \{02\}. \qquad (1)$$

This function takes as input the four bytes of one column and outputs four bytes, that replace the original column. The result is a new matrix consisting of 16 new bytes.

2.4 AddRoundKey: The 16 bytes of the input matrix (128 bits input) are XOR-ed with the 128 bits of the round key which is done in each round in the Algorithm.

Each transformation has an inverse from which decryption follows in a straightforward way by reversing the steps in each round: AddRoundKey (inverse of itself), InvMixColumns, InvShiftRows, and InvSubBytes.

B. Fast AES approach

In Fast AES, A lookup table solution could be substituted for the four steps in an AES transformation round, to enable a more compact and efficient implementation on 32-bit or more bits processors. In this method, four lookup tables are defined as: T0, T1, T2 and T3. Each table (or T-box) accepts one byte of input, and comes out with a 32-bit column vector. The operations of each transformation round can be defined as follows:

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{0,j}] \oplus T_2[a_{0,j}] \oplus T_3[a_{0,j}] \oplus K_j \quad (2)$$

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 $a_{0,j}$. According to the compact solution above, it only takes 4 exclusive-OR operations and 4 table lookups per column per round, and hence parallelizable which reduce the computational complexity. Although the transformation order of AES decryption and encryption are different, an equivalent version of decryption algorithm and encryption algorithm has the same structure [1]. This algorithm requires only four lookup table transformations and four XOR operations.

C. *AES Modes of operation*

In Table I six modes of AES encryption is recommended by NIST [1] providing different levels of security and parallelism (Electronic Code book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), Counter (CTR) and XTS (Xor-encrypt-xor Tweaked codebook mode with cipher text Stealing)). The experiments was based on a CUDA AES implementation in ECB mode which is exploitable for parallel acceleration.

III. Implementation Techniques on GPU Platform

All previous work of AES on GPGPU are divided into four major implementation techniques [3]:

- Memory usage optimization
- Parallel granularity
- GPU platform specific optimization
- CPU-GPU data transfer optimization

A. *Memory optimization*

In this subsection, this work focuses on three main parameters: (1) Input plaintext, (2) Lookup tables, and (3) Key expansion.

1) Input plain-text: In order to achieve maximum performance, encrypted plain-text are stored into global memory, which require the access pattern to be coalesced. This can be done by using the built in SIMD data types (int4 for 128-bit), so that its possible to combine the 16 linear bytes for a coalesced memory access [3].

2) Lookup tables: Many approaches for accessing AES lookup tables using GPU have been done to obtain max performance. The best choice to store (T-box) is in the shared memory in GPU as the access speed is faster and the requirement for optimized usage is less stringent [3].

TABLE II: Occupancy in GPU SM 5.2 [3]

thread blocksize (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%

3) Key expansion: Many approaches for accessing AES key using GPU have been done to obtain max performance. The best approach is to pre-compute the encryption keys in CPU, and they are only generated once for entire encryption process. The expanded keys are then copied to GPU global memory and when the GPU kernel is launched, each thread in a warp copies key value(s) from global memory and stored it in one or two registers, which is known as warp shuffle operation. This approach has been appeared in modern GPU architectures [3]. There are 32 threads in a warp, so up to 64 keys can be kept in total. By using this strategy, the encryption keys can be kept entirely in registers for high access speed. This work bench marks the warp shuffling approach and compares it with the shared memory approach.

B. *Parallel granularity*

There are two main approaches in this section previously:

- Multiple threads can cooperatively encrypt one block (4 bytes/thread, 8 bytes/thread).
- Each thread can encrypt one whole block (16 bytes/thread).

According to previous work, the second approach (16 byte/thread granularity) should be the best one [4] [7] [10] as the entire encryption of a block can be done within a thread, no synchronization and shared data between threads are needed.

In this work a new approach is proposed such that one thread will be encrypting multiple blocks (32 bytes/thread, 64 bytes/thread, etc..).

C. *GPU platform specific optimization*

Each GPU architecture usually have its own hardware layout which is slightly different in design from previous architectures, thus optimizations may differ from architecture to another. The GPU occupancy is a measure of thread parallelism in a CUDA program. The higher the occupancy, the more opportunities an Stream Multiprocessor (SM) has to put, compute and load/store units to work each cycle which leads to higher throughput. The implementation thread block size will affect the GPU occupancy which depends on GPU architecture. As an example for Maxwell GPU SM 5.2, the thread block size which satisfy 100% Occupancy are (128, 256, 512, 1024) TBS as shown in Table II. The CUDA occupancy tool provided by the Nvidia SDK calculates such values [11]

In this work, benchmarks and experiments are executed on the latest three generations of Nvidia GPU architectures. Basically, one of the main optimizations that this work has

TABLE III: Summary of Previous Implementation Techniques and Performance on AES-128 ECB

References	AES (Gbps)	Mode	GPU device	Architecture	Memory Optimizations	Computation granularity	Year
Mei et al.[4]	51.2	Unknown	GS9200M	Tesla	T-box and key in shared memory	Unknown	2010
Bos et al.[5]	59.6	Unknown	GTX295	Tesla	T-box and key in shared memory	Unknown	2010
Nishikawa et al.[6]	48.6	ECB	Tesla C2050	Fermi	T-box and key in shared memory	16B/thread	2012
Li et al.[7]	60.0	ECB	Tesla C2050	Fermi	T-box and key in shared memory	16B/thread	2012
Gilger et al.[8]	47.0	ECB	GeForce GTX 295	Fermi	T-box and key in shared memory	16B/thread	2012
Nishikawa et al.[9]	68.6	ECB	GTX680	Kepler	T-box and key in shared memory	16B/thread	2014

TABLE IV: Comparison of Eight Practical Parallel AES Schemes

AES schemes	Shared Memory Approach	Warp Shuffle Approach
1024 TPB - 16 B/thread	1024 byte per thread 16 Byte Granularity	1024 byte per thread 16 Byte Granularity
512 TPB - 16 B/thread	512 byte per thread 16 Byte Granularity	512 byte per thread 16 Byte Granularity
256 TPB - 16 B/thread	256 byte per thread 16 Byte Granularity	256 byte per thread 16 Byte Granularity
128 TPB - 16 B/thread	128 byte per thread 16 Byte Granularity	128 byte per thread 16 Byte Granularity
1024 TPB - 32 B/thread	1024 byte per thread 32 Byte Granularity	1024 byte per thread 32 Byte Granularity
512 TPB - 32 B/thread	1024 byte per thread 32 Byte Granularity	1024 byte per thread 32 Byte Granularity
256 TPB - 32 B/thread	1024 byte per thread 32 Byte Granularity	1024 byte per thread 32 Byte Granularity
128 TPB - 32 B/thread	1024 byte per thread 32 Byte Granularity	1024 byte per thread 32 Byte Granularity

performed is about both the CUDA blocks and grids dimensions that forms the granularity of the GPU parallelism; These dimensions affects the multiprocessors inside the GPU and the distribution of the work loads over them. We do believe that this work will have a considerable impact on performance especially on the latest architectures.

D. CPU-GPU data transfer optimization

Generally in GPGPU, the data transfer overhead is substantial. However, to exploit the effective performance, it is necessary to consider the overhead caused by data transfer between CPU and GPU. Generally, to hide this overhead, Nvidia GPUs provide the function of overlapping data transfer (memory copy) and processing using stream programming model [11].

IV. RELATED WORK

There are several research work focusing on implementation techniques used for optimizing AES using GPU that are mentioned above. Table III summarizes previous implementations techniques of CUDA based AES-128 ECB on GPU since 2010.

Di Biagio et al. implemented AES-CTR using GT8800 in [12]. They explored the fine grain (4 bytes/thread) and coarse grain (16 bytes/thread) parallel implementation and found that 16 Bytes/thread show better performance because no synchronization between threads is needed.Di Biagio et al. proposed to store T-box in shared memory instead of constant memory, and discussed about shared memory partitioning to avoid bank conflicts.

Same approach was adopted by Mei et al. in [4]. The authors proposed an efficient approach to parallelize AES and fine-tune the memory utilization in GeForce 9200M GS. They also used 16 bytes/thread granularity, and stored both the T-box and key in shared memory. They achieved 48 Gbps maximum throughput speed.

Bos et al. discussed three strategies to generate the key: on the fly, in texture and in shared memory in [24]. They also run multiple streams to overlap memory copy and kernel execution. They performed implementation using GTX295.

N. Nishikawa in [10] implemented AES on CUDA and studied the following computation granularities: (1) 16 bytes/thread: Meaning that each thread is mapped to a standard AES plain-text block having a size of of 16 bytes. (2) 8 bytes/thread and 4 bytes/thread: With sizes less than 16 byte, it means that more than one thread will be needed to complete the work on an AES plaintext block. (3) 1 byte/thread: Where 16 threads are required to process a whole plain-text block. Moreover, they stored the T-box table and round keys on shared memory. The best performance they got was 35.2 Gbps throughput rate using 16 bytes/thread granularity.

Li et al. in [7] stored T-boxes on on-chip shared memory. Morever, the granularity where one thread handles a 16 Bytes AES block was adopted. They achieved the highest performance of around 60 Gbps throughput on NVIDIA Tesla C2050 GPU.although the AES encryption and decryption make significant performance advance, the bandwidth of PCI-E bus and page-lock memory allocation cost are vital limitations. It makes the throughput of encryption and decryption greatly reduced. Even overlapping techniques used, this problem can't be solved satisfactorily.

Nishikawa et al. in [9] presented implementation of block ciphers in NVIDIA and AMD GPU based on Kepler and GCN architectures and analyzed energy efficiency of both GPU platforms. They achieved the highest performance of around 68.6 Gbps throughput on Geforce GTX 680 on the Kepler architecture.

A. H. Khan. in [13] we implement the AES-128 ECB Encryption on two of the recent and advanced GPUs (NVIDIA Quadro FX 7000 and Tesla K20c) with different memory usage schemes and varying input plaintext sizes and patterns. We obtained a speedup of up to 87x against an advanced CPU (Intel Xeon X5690) based implementation

Wai-Kong Lee. in [3] presented implementation of block ciphers in NVIDIA GTX 980 with Maxwell architecture. Their implementation focus on four main areas, namely memory optimization, algorithmic optimization, parallel granularity and GPU platform specific optimization. they also proposed a novel

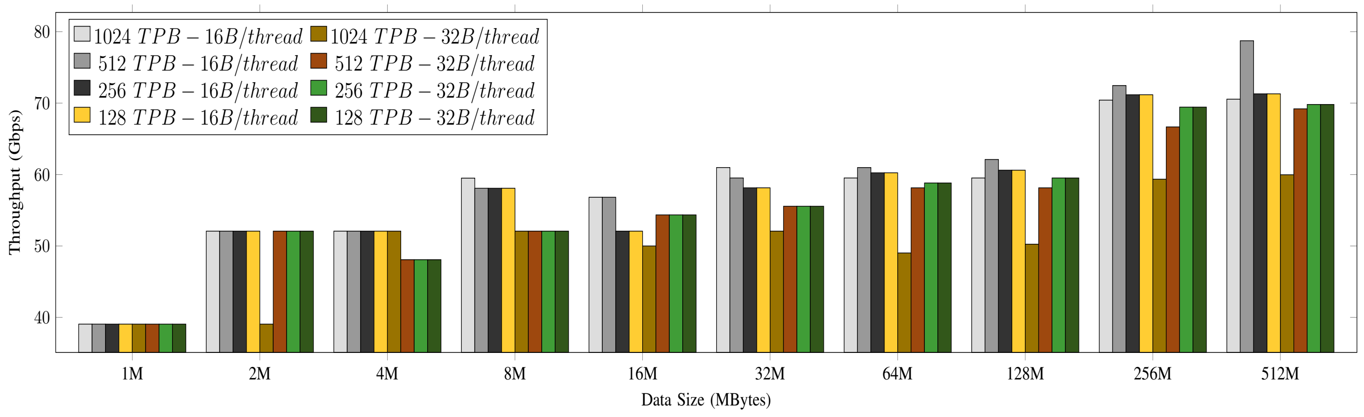


Fig. 3: Warp Shuffle approach for NVIDIA GTX 780

method to store the encryption keys in high speed registers and exchange it across threads in same warp using warp shuffle operation existing in Maxwell architecture to further accelerate the performance. As for granularity they used 16 bytes/thread granularity.

V. Proposed Approach

We implement AES algorithm and evaluate its performance using eight different parallel schemes as shown in Table IV. The experiments are performed on three different platforms whose specifications are introduced in section VI-A. We focused in three techniques of optimization seeking the best performance results.

- Key memory location: We will be storing the round keys in either shared memory or in registers accessing them through warp shuffling technique.
- Computation granularity: We propose trying a new approach for the parallel granularity in this work by increasing the work done per thread, making a single thread handles more than a single AES block. We will be running a 32 bytes/thread benchmarks versus the classical 16 byte/thread approach.
- Thread Block size (TPB): This is a factor that has been neglected through the past research done on AES in the latest GPU architectures. We believe that due to the changes and the increase of parallelism level, this should be a major factor now affecting computation performance. We chose to experiment with thread block sizes (1024 TPB, 512 TPB, 256 TPB, 128 TPB) due to the fact that they satisfy a 100% occupancy generally over the latest GPU architectures.

In this work, the focus mainly is on the total kernel time which exempts the data transfer time as the optimizations done focus on the processing performance. For simplification we divide these implementation schemes into two approaches :

- Shared memory approach: Encryption keys in Shared Memory.
- Warp shuffle approach: Encryption keys in registers and accessing them using warp shuffle feature.

TABLE V: Configurations of Three Experiment 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

VI. Experiments and Performance Evaluation

We present results for the eight approaches of our implementation described earlier which executed on three different GPU architectures: Kepler, Maxwell and Pascal.

A. *Experimental platforms*

In order to exhibit the effects, of the AES implementation schemes experimented in this work, for different block inputs we use three different platforms supporting three different GPU devices with the most recent GPU architectures(Kepler, Maxwell and Pascal) as shown in table V.

B. *Experimental results on Kepler platform*

We implement the eight AES implementation schemes on Kepler platform using NVIDIA GTX 780. We evaluate their performance (throughput) results using two implementation Approaches:

1) Warp Shuffle Approach: After applying the warp shuffle technique in order to optimize memory access by sharing the key data between the threads of the same warp through their register. Figure. 3, demonstrates the throughput of the eight schemes on Kepler platform for different input block sizes (1M byte to 512M byte). It was observed that when input block sizes increases the throughput of all schemes increases which demonstrates that higher input block sizes get full optimization from GPU. It was also observed that 512 TPB - 16 B/thread

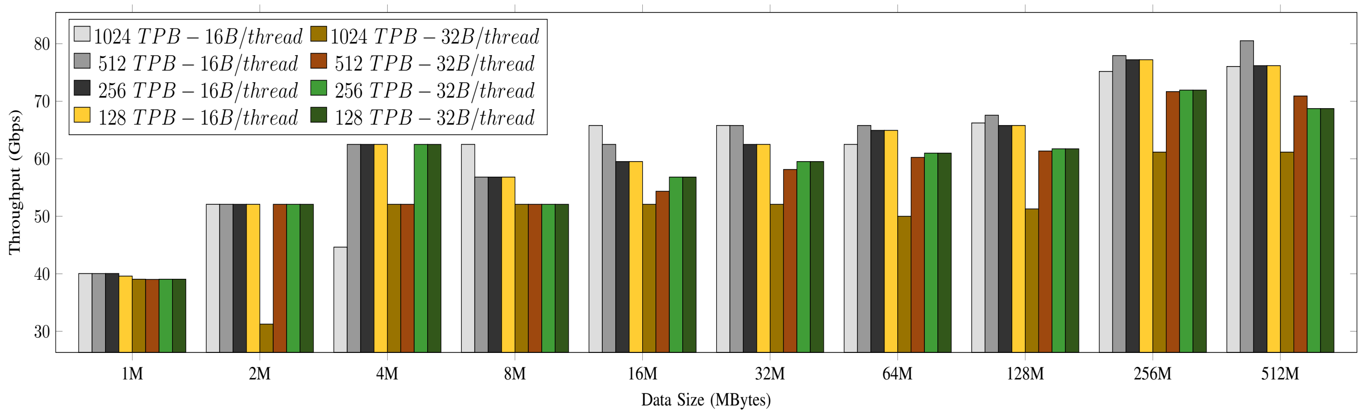


Fig. 4: Shared Memory approach for NVIDIA GTX 780

scheme give the best throughput values during the majority of input block sizes with max throughput equal to 78 Gbps.

2) Shared Memory Approach: By placing the key in Shared Memory, a throughput increase was observed, shown in Figure. 4, over the Warp Shuffle approach. It was also observed that 512 TPB - 32 B/thread scheme give the best throughput value during the majority of input block sizes with max throughput equal to 80 Gbps.

C. Experimental results on Maxwell platform

We implement the eight AES implementation schemes on Maxwell platform using NVIDIA GTX TITAN X which is one of the fastest single-GPU graphics in Maxwell family. From the results we can find obvious speed improvements over Kepler platform. We evaluate their performance (throughput) results using two implementation Approaches:

1) Warp Shuffle Approach: Using Warp Shuffle technique, the same experiments were executed. Figure. 5 demonstrates the throughput of eight schemes on Maxwell platform for different input block size (1M byte to 512M byte) we observe that when input block sizes increases the throughput of all schemes increases which demonstrates that higher input block sizes get full optimization from GPU. We also observe that 256 TPB - 32 B/thread and 128 TPB - 32 B/thread schemes give the best throughput values during the majority of input block sizes with max throughput equal 203 Gbps for 256 TPB - 32 B/thread.

2) Shared Memory Approach: Using Shared Memory key location, the average throughput results on Maxwell platform for Shared Memory Approach are shown in Figure 6. It shows again obvious speed improvements over Warp Shuffle Approach. We also observe that 512 TPB -32 B/thread, 256 TPB - 32 B/thread and 128 TPB - 32 B/thread schemes give the best throughput values during the majority of input block sizes with max throughput equal 207 Gbps for 256 TPB - 32 B/thread.

D. Experimental results on Pascal platform

The eight AES implementation schemes were tested on the new pascal platform to evaluate their performance (throughput) results using two implementation approaches. The results shows obvious speed improvements over Maxwell platform which was expected as the Nvidia GeForce GTX 1080 is the latest GPU in the new Pascal Architecture.

1) Warp Shuffle Approach: After applying the warp shuffle technique in order to optimize memory access by sharing the key data between the threads of the same warp through their register, Figure. 7 it was observed that it increases throughput as input block sizes increases. It was also observed that 512TPB-32B/thread, 256TPB-32B/thread and 128TPB-32B/thread schemes gave the best throughput values during the majority of input block sizes reaching a max throughput of 272 Gbps for 256 TPB-32B thread and 128 TPB-32B/thread.

2) Shared Memory Approach: When key was stored in shared memory the average throughput results on Pascal platform, shown in Figure. 8, showed obvious speed improvements over Warp Shuffle Approach. It was also observed that 512 TPB-32B/thread, 256 TPB-32B/thread, 128 TPB-32B/thread schemes gave the best throughput values during the majority of input block sizes with max throughput equal 280 Gbps for 128 TPB-32B/thread.

E. Experiment analysis and discussion

We did micro benchmarks on the three GPU architectures (Kepler, Maxwell and Pascal) as shown in tables VI, VII, and VIII. Below is the analysis for the experimental results shown in these tables:

1) Parallel granularity: Generally, for Kepler GPU 16 bytes/thread granularity gives the best throughput results. However, for the modern architectures Maxwell and Pascal GPUs, its clear that 32 bytes/thread granularity (Two AES blocks per thread) gives better throughput than the classical 16 bytes/thread granularity (One AES block per thread), especially for large input size. We explained this as follows:

1.1 The thread can process two blocks of data in the 32 byte/thread scenario rather than one block done by 16 bytes/thread. By doing this, thread block switching will be reduced and hence increases performance.

1.2 The shared memory access in the 32 byte/thread scenario will be twice shared memory access done by 16

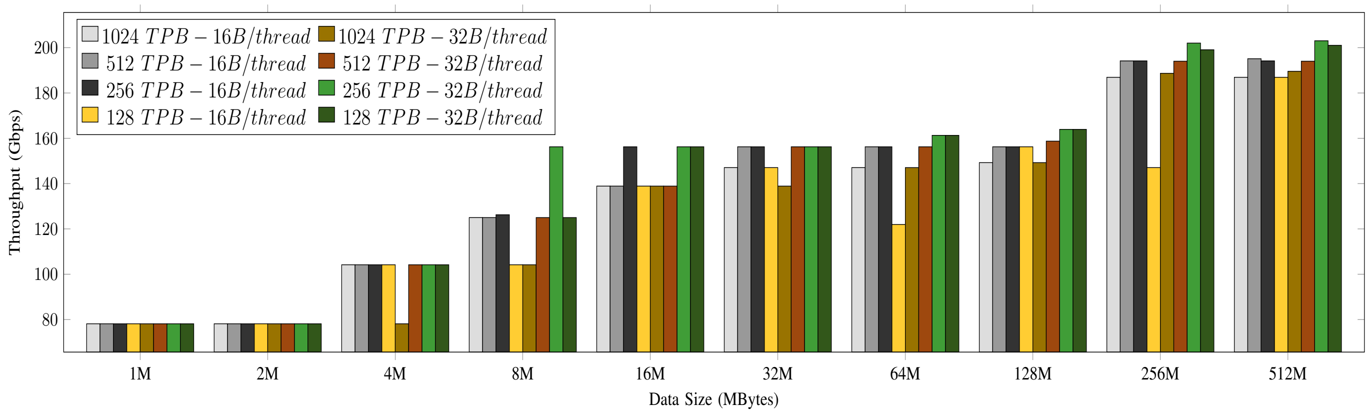


Fig. 5: Warp Shuffle approach for NVIDIA GTX TITAN X

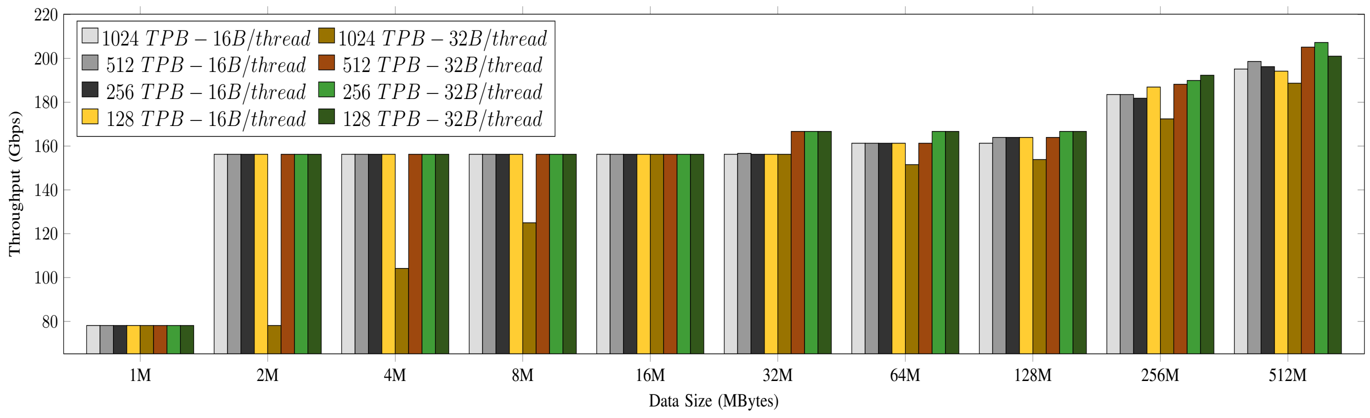


Fig. 6: Shared Memory approach for NVIDIA GTX TITAN X

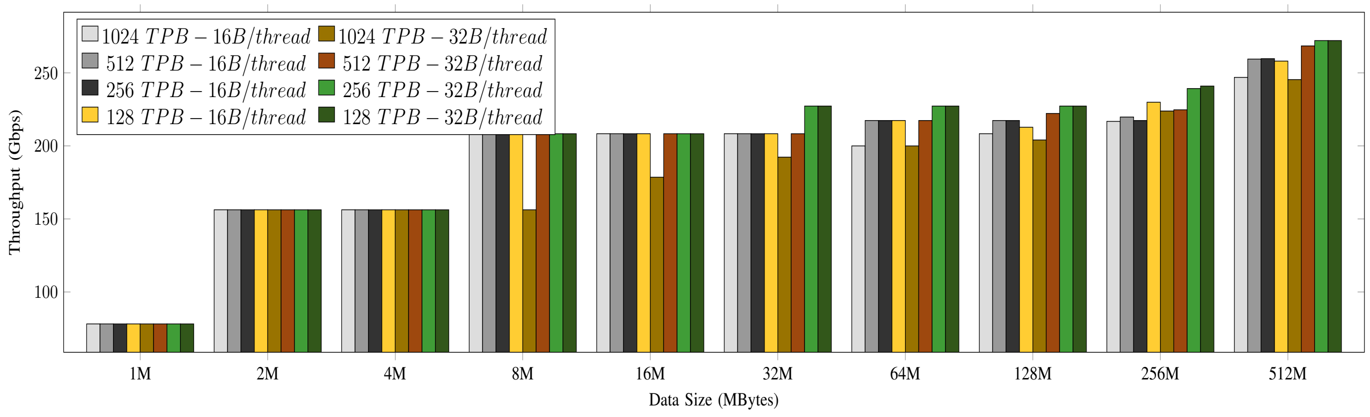


Fig. 7: Warp Shuffle approach for 1080 GPU

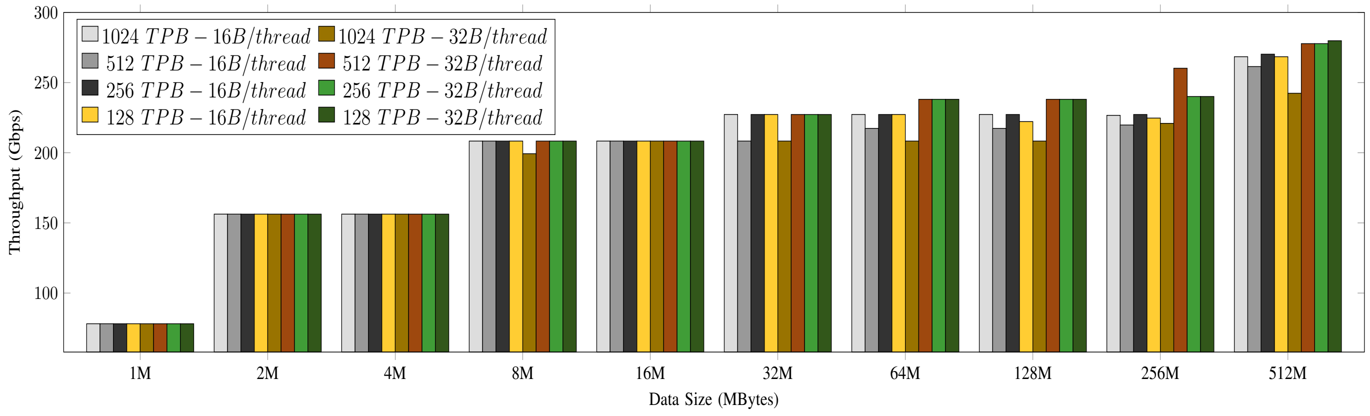


Fig. 8: Shared Memory approach for NVIDIA GeForce GTX 1080 GPU

TABLE VI: Microbenchmarks for NVIDIA GeForce GTX 780 (Kepler) [Gbps]

Data size	**AES Approachs**	**1024 TPB 16 B/thread**	**512 TPB 16 B/thread**	**256 TPB 16 B/thread**	**128 TPB 16 B/thread**	**1024 TPB 32 B/thread$**	**512 TPB 32 B/thread**	**256 TPB 32 B/thread**	**128 TPB 32 B/thread**
512 M	Shared memory approach	76.05	**80.52**	76.19	76.19	61.16	70.92	68.73	68.73
	Warp shuffle approach	70.55	78.73	71.3	71.3	59.97	69.2	69.81	69.81
256 M	Shared memory approach	75.19	77.94	77.22	77.22	61.16	71.68	71.94	71.94
	Warp shuffle approach	70.42	72.46	71.17	71.17	59.35	66.67	69.44	69.44
128 M	Shared memory approach	66.23	67.57	65.79	65.79	51.28	61.35	61.73	61.73
	Warp shuffle approach	59.52	62.11)	60.61	60.61	50.25	58.14	59.52	59.52
64 M	Shared memory approach	62.5	65.79	64.94	64.94	50.0	60.24	60.98	60.98
	Warp shuffle approach	59.52	60.98	60.24	60.24	49.02	58.14	58.82	58.82

bytes/thread. By doing this, kernel execution time will be increased which will minimize performance.

A trade off is shown between the number of blocks and shared memory accessed. In modern GPU architectures like Maxwell and Pascal they have larger number of streaming multiprocessors (SMs) and also larger number of active blocks per multiprocessors compared with Kepler GPU. this will run multiple blocks without block switching which will maximize performance (throughput).

2) Key memory location: Warp shuffle did not seem to be a very effective optimization over the use of shared memory in the experiments we have made.

3) Thread block size: Generally, for Maxwell and Pascal GPUs: 128 TPB and 256 TPB are the best thread block size which gives the best performance result in the majority of the micro benchmarks done. Moreover, for Kepler GPU: 512 TPB is the best thread block size which gives the best throughput in the majority of the presented micro benchmarks.

VII. Conclusion

Eight parallel AES schemes were presented focusing on three aspects of optimization: key memory location, computation granularity and thread block size seeking the best performance (throughput). For simplicity, we classify these schemes in two main categories: 1) Shared memory approach and 2) Warp shuffle approach. These eight schemes are bench marked on the the three latest generations of Nvidia GPU architectures: Kepler, Maxwell and Pascal.

After implementing the new proposed approaches in both multiple parallel granularities and thread block sizes, we managed to explore areas that haven't been experimented thoroughly in previous research exploiting new GPU architectures, in terms of workload distribution over threads and thread blocks, to gain higher performance. In modern architectures this was very useful especially in terms of increasing workload per thread by encrypting two or three AES blocks per CUDA thread. This work has achieved the highest performance up to date to the best of our knowledge reaching a throughput of 279.86 Gbps at encryption speed of AES-128 on the GTX 1080 - Pascal architecture.

However, in the benchmarks we have made, it shows that some techniques varied in performance effect from an architecture to another. Some of the optimizations we have made such as the parallel granularity tweaking did not have much effect on the older platforms. The variation of the effect of the optimizations from platform to another as well as the variation of such effects based on input size shows that the quantitative approach has been useful to reach the higher throughput that this work has presented.

TABLE VII: Microbenchmarks for NVIDIA GeForce GTX TITAN X (Maxwell) [Gbps]

Data size	**AES Approachs**	**1024 TPB 16 B/thread**	**512 TPB 16 B/thread**	**256 TPB 16 B/thread**	**128 TPB 16 B/thread**	**1024 TPB 32 B/thread$**	**512 TPB 32 B/thread**	**256 TPB 32 B/thread**	**128 TPB 32 B/thread**
512 M	Shared memory approach	195.13	198.59	196.23	194.17	188.68	205.13	**207.25**	201.01
	Warp shuffle approach	186.92	195.12	194.17	186.92	189.57	194.02	203.05	201.04
256 M	Shared memory approach	183.49	183.49	181.82	186.92	172.41	188.19	189.92	192.31
	Warp shuffle approach	186.92	194.17	194.17	147.06	188.68	194.02	202.02	199.08
128 M	Shared memory approach	161.29	163.93	163.93	163.93	153.85	163.93	166.67	166.67
	Warp shuffle approach	149.25	156.25	156.25	156.25	149.25	158.73	163.93	163.93
64 M	Shared memory approach	161.29	161.29	161.29	161.29	151.52	161.29	166.67	166.67
	Warp shuffle approach	147.06	156.25	156.25	121.95	147.06	156.25	161.29	161.29

TABLE VIII: Microbenchmarks for NVIDIA GeForce GTX 1080 (Pascal) [Gbps]

Data size	**AES Approaches**	**1024 TPB 16 B/thread**	**512 TPB 16 B/thread**	**256 TPB 16 B/thread**	**128 TPB 16 B/thread**	**1024 TPB 32 B/thread$**	**512 TPB 32 B/thread**	**256 TPB 32 B/thread**	**128 TPB 32 B/thread**
512 M	Shared memory approach	268.46	261.44	270.27	268.46	242.42	277.78	277.78	**279.86**
	Warp shuffle approach	246.91	259.44	259.74	258.06	245.4	268.46	272.11	272.11
256 M	Shared memory approach	226.67	219.78	227.27	224.72	220.96	260.25	240.1	240.1
	Warp shuffle approach	216.79	219.78	217.39	229.89	223.9	224.72	239.27	240.96
128 M	Shared memory approach	227.27	217.39	227.27	222.22	208.33	238.1	238.1	238.1
	Warp shuffle approach	208.33	217.39	217.39	212.77	204.08	222.22	227.27	227.27
64 M	Shared memory approach	227.27	217.39	227.27	227.27	208.33	238.1	238.1	238.1
	Warp shuffle approach	200.0	217.39	217.39	217.39	200.0	217.39	227.27	227.27

VIII. Future Work

We believe that due to number of architectures currently active on GPUs, and since it became quite useful as a hardware accelerator, the development of an auto-tuner that select the best configuration parameters based on the GPU architecture would be our next target for future work.

References

[1] S. H. Standard, "National institute of standards and technology (nist), fips publication 180-2, aug 2002."

[2] Q. Li, C. Zhong, K. Zhao, X. Mei, and X. Chu, "Implementation and analysis of aes encryption on gpu," in *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*. IEEE, 2012, pp. 843–848.

[3] W.-K. Lee, H.-S. Cheong, R. C.-W. Phan, and B.-M. Goi, "Fast implementation of block ciphers and prngs in maxwell gpu architecture," *Cluster Computing*, vol. 19, no. 1, pp. 335–347, 2016.

[4] C. Mei, H. Jiang, and J. Jenness, "Cuda-based aes parallelization with fine-tuned gpu memory utilization," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–7.

[5] D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright, "Fast software aes encryption," in *International Workshop on Fast Software Encryption*. Springer, 2010, pp. 75–93.

[6] N. Nishikawa, K. Iwai, and T. Kurokawa, "High-performance symmetric block ciphers on multicore cpu and gpus," *International Journal of Networking and Computing*, vol. 2, no. 2, pp. 251–268, 2012.

[7] Q. Li, C. Zhong, K. Zhao, X. Mei, and X. Chu, "Implementation and analysis of aes encryption on gpu," in *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*. IEEE, 2012, pp. 843–848.

[8] J. Gilger, J. Barnickel, and U. Meyer, "Gpu-acceleration of block ciphers in the openssl cryptographic library," in *International Conference on Information Security*. Springer, 2012, pp. 338–353.

[9] N. Nishikawa, I. Keisuke, H. Tanaka, and T. Kurokawa, "Throughput and power efficiency evaluation of block ciphers on kepler and gcn gpus using micro-benchmark analysis," *IEICE TRANSACTIONS on Information and Systems*, vol. 97, no. 6, pp. 1506–1515, 2014.

[10] K. Iwai, T. Kurokawa, and N. Nisikawa, "Aes encryption implementation on cuda gpu and its analysis," in *Proceedings of the 2010 First International Conference on Networking and Computing*, ser. ICNC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 209–214. [Online]. Available: http://dx.doi.org/10.1109/IC-NC.2010.49

[11] NVIDIA, "CUDA C Programming Guide," no. March, pp. 1–173, 2012. [Online]. Available: http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/

[12] A. Di Biagio, A. Barenghi, G. Agosta, and G. Pelosi, "Design of a parallel aes for graphics hardware using the cuda framework," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–8.

[13] A. Khan, M. Al-Mouhamed, A. Almousa, A. Fatayar, A. Ibrahim, and A. Siddiqui, "Aes-128 ecb encryption on gpus and effects of input plaintext patterns on performance," in *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on*. IEEE, 2014, pp. 1–6.