Proceedings of the Second International Conference on Inventive Systems and Control (ICISC 2018)
IEEE Xplore Compliant - Part Number:CFP18J06-ART, ISBN:978-1-5386-0807-4; DVD Part Number:CFP18J06DVD, ISBN:978-1-5386-0806-7

Parallel Implementation of Cryptographic Algorithm: AES Using OpenCL on GPUs

Govardhana Rao Inampudi
Department of Computer Science and engineering
Osmania University, Hyderabade
igrao10@gmail.com

Prof. Shyamala K
Department of Computer Science and engineering
Osmania University, Hyderabade
prkshyamala@gmail.com

Prof.S.Ramachandram
Department of Computer Science and engineering
Osmania University, Hyderabade
Schandram@gmail.com

***Abstract*—**The importance of protecting the information has increased rapidly during the last decades. This motivates the need for cryptographic algorithms. The Acceleration of the symmetric key cryptography algorithm is enhanced using parallel implementation on GPGPUs with Open Computing Language (OpenCL). The General Purpose Graphics Processing Units (GPGPU) enables high level of parallelism with Compute Unified Device Architecture (CUDA)/Open Computing Language (OpenCL) programming environments using (Single Instruction Multiple Data) SIMD architecture. In this paper, the parallel implementation of Advanced Encryption Standard (AES) Algorithm using OpenCL is presented. The experimental results show that, the parallel implementation of encryption algorithm tested on GPUs accelerates the speed when compared to sequential implementation of encryption algorithm. The experimental result shows that, the percentage 99.8% is improved compared to sequential implementation of Encryption algorithm.

Keywords: Advanced Encryption Standard (AES), Graphics Processing Unit (GPU), Image Restoration, OpenCL, SIMD

I. INTRODUCTION

The Graphics Processing Unit (GPU) [1] plays a vital role in various types of image and video processing applications. The invention of these many core GPUs provides the scope to accelerate speed in case of massive parallel applications. The GPU ported applications improves the performance by offloading the compute intensive part onto GPU and remaining code onto Central Processing Unit (CPU). The multi core GPUs provide high performance for data parallel tasks using SIMD architectures [2] [3]. In [4], the author shows that, the usage of GPGPUs accelerates the cryptographic solution to crack the UNIX password cipher in 100 MHz.

The focus of this paper is to accelerate the implementation of AES algorithm. The proposed work is implemented using OpenCL and tested on Nvidia GPUs. The experimental results are compared with sequential implementation on different set of inputs.

The rest of the paper is organized as follows: section II discusses the existing parallel formulation of AES algorithm and introduction to GPU computing. Section III describes the GPU implementation of the AES algorithm. Experimental results followed by conclusions are presented in section IV and V respectively.

II. LITERATURE SURVEY

Details of AES algorithm [5], hardware and software employed are briefly discussed in this section. The software constructs and terms used in the implementation are also elucidated. The program flow of OpenCL [2] is explained in detail.

A. Advanced Encryption Standard Algorithm

The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the National Institute of Standards (NIST) [3] in 2001 based on Rijndael cipher, where input is a plain text and encryption produces cipher text.

The algorithm has four rounds based on Rijndael cipher [6] as shown in Fig 2:

1. **Key Expansion:** Round keys are derived from the cipher key using Rijndael's key schedule. 128-bit round key block for each round is required for AES.

2. **Initial Round: E**ach byte of the state is combined with a block of the round key using bitwise XOR.

3. **Rounds:**
   i) Sub Bytes: A non-linear substitution step where each byte is replaced with another based on lookup table.
   ii) Shift Rows: In this step, the last three rows of the states are shifted cyclically.
   iii) Mix Columns: In this step, the four bytes in each columns are combined.
   iv) AddRoundKey: In this step, the sub key is added by combining each byte of the state with the corresponding byte of the sub key using bit wise XOR.
4. **Last Round:** In this round, Sub Bytes, Shift Rows and an AddRoundKey operation takes place.

B. Introduction to Graphics Processing Unit

The Graphics Processing Unit (GPU) is a specialized electronic circuit designed to accelerate the creation of images in a frame buffer intended for output to display. A typical GPU is characterized by the presence of hundreds of cores compared to any conventional CPU, which has limited number of cores, such as 2 to 16 cores. GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles. Modern GPUs are very efficient at manipulating computer graphics and image processing, and their highly parallel structure makes them more effective than general-purpose CPUs for algorithms where, the large blocks of data is processed in parallel.

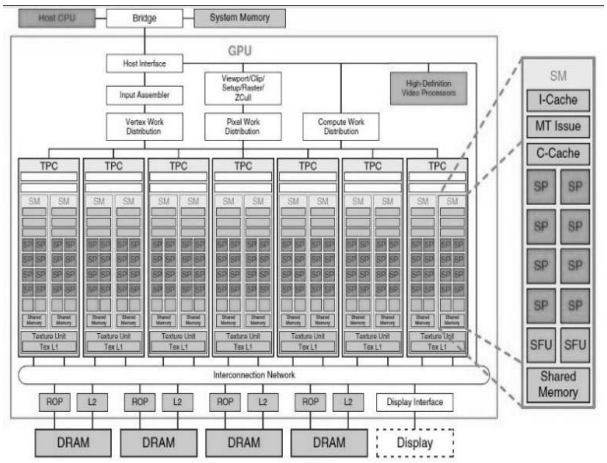

Figure 1 Basic Unified GPU Architecture

C. Implementation of AES using GPUs

This section presents the details of sequential implementation of AES algorithm and GPU implementation of AES algorithm. The various steps in AES algorithm is shown in Fig 2. The time taken to execute all the rounds increases linearly as the input size increases.

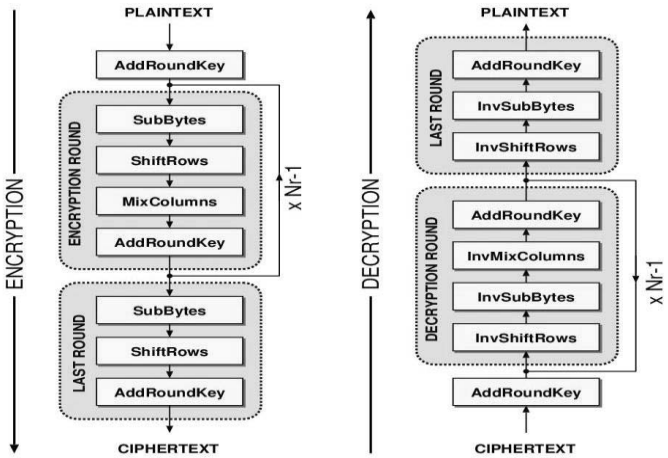

Fig. 2. Steps in AES Algorithm

D. Introduction to OpenCL

Open CL is a framework for parallel programming of heterogeneous systems. OpenCL provides an effective way to program for heterogeneous systems, homogeneous, multi-core processors. The OpenCL execution model comprises of two components, such as kernels and host programs. kernels are the basic unit of executable code that runs on one or more OpenCL devices. Similar to C functions, kernels can be invoked using data or task parallelism. The host program executes on the host system called as device context, and queues kernel execution instances using command queues. Kernels are queued in in-order, but can be executed in either in-order or out-of-order. OpenCL allows the kernel to access Global memory, Constant memory, Local memory and private memory.

A profiler is used to analyze the various constraints, and aspects of a program. AMD APP Profiler is a performance analysis tool that gathers data from the OpenCL run-time and AMD Radeon GPUs during the execution of an OpenCL application. Similarly, CodeXL also gives comparative analysis of kernel executions. In this paper, these profilers are used to analyze the kernels execution for AMD Radeon 8550M GPU.

In [8], the author proposed two schemes for parallel AES encryption implementation with off-line key expansion on shared-memory multi core architecture. The tasks are equally partitioned and grouped into cluster. High efficient inter-core communication is obtained based on the shared memory. This kind of implementation reduces the latency for a single AES encryption, compared with pipelining schemes.

The author [6] describes both traditional style approaches based on the OpenGL graphics API and presents an efficient implementation of the AES algorithm in the CUDA platform by NVIDIA Graphics card. The performance of the new fastest GPU solution is compared with those of the reference sequential implementations running on an Intel Pentium IV 3.0 GHz CPU.

III. PARALLEL IMPLEMENTATION OF AES ALGORITHM

In this section, the parallel implementation of AES algorithm is discussed in detail. The various steps involved in parallel Implementation are as follows:

1. The kernel functions are programmed based on data decomposition technique, data input is divided into 256 work-items which is maximum possible for the GPU hardware used in this paper.
2. Data Parallelism is achieved by utilizing the maximum available work-items, which in turn assigns these elements to multiple threads and simultaneously to GPU cores for execution. For the AMD 8550M and AMD 8570M GPU's, a maximum of 16777216 elements can be invoked at a time.

3. OpenCL constructs, such as BARRIER construct allows results to be copied back to the host program only after complete execution of a work-group
4. The AES algorithm employed is of 14 rounds (i.e, 256-AES).
5. The size of each work item is 24 bits, which represent the RGB pixels in hexadecimal.

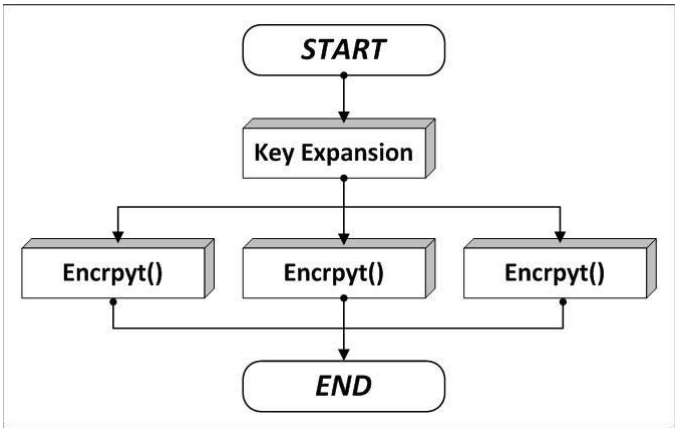

Fig 3: Flow diagram of Parallel implementation of AES algorithm

As shown in Fig 3, the Encrypt() function executed concurrently, where as key expansion is executed sequentially. Based on the prefix computation technique, the encrypt function is executed concurrently by utilizing the supported GPUs. The parallel formulation of the shiftrow() function is illustrated with the example is as follows:

Code Snippet for Sequential Execution of shift row function:

```
/ / Rotate first row 1 columns t o l e f t
Temp= s t a t e [ 1 ] [ 0 ] ;
s t a t e [ 1 ] [ 0 ] = s t a t e [ 1 ] [ 1 ] ;
s t a t e [ 1 ] [ 1 ] = s t a t e [ 1 ] [ 2 ] ;
s t a t e [ 1 ] [ 2 ] = s t a t e [ 1 ] [ 3 ] ;
s t a t e [ 1 ] [ 3 ] = temp ;
/ / Rotate second row 2 columns t o l e f t
Temp= s t a t e [ 2 ] [ 0 ] ;
s t a t e [ 2 ] [ 0 ] = s t a t e [ 2 ] [ 2 ] ;
s t a t e [ 2 ] [ 2 ] = temp ;
Temp= s t a t e [ 2 ] [ 1 ] ;
s t a t e [ 2 ] [ 1 ] = s t a t e [ 2 ] [ 3 ] ;
s t a t e [ 2 ] [ 3 ] = temp ;
/ / Rotate third row 3 columns t o l e f t
Temp= s t a t e [ 3 ] [ 0 ] ;
s t a t e [ 3 ] [ 0 ] = s t a t e [ 3 ] [ 3 ] ;
s t a t e [ 3 ] [ 3 ] = s t a t e [ 3 ] [ 2 ] ;
s t a t e [ 3 ] [ 2 ] = s t a t e [ 3 ] [ 1 ] ;
s t a t e [ 3 ] [ 1 ] = temp ;
```

Code Snippet for Parallel Formulation of shift row function:

```
// Rotate first row 1 columns to right

if ( k==4)
{
emp=P [ i + 3 ] ;
P [ i +3]=P [ i + 2 ] ;
P [ i +2]=P [ i + 1 ] ;
P [ i +1]=P [ i ] ;
P [ i ]= temp ;
}
// Rotate second row 2 columns to right

i f ( k==8)
{
emp=P [ i ] ;
P [ i ]=P [ i + 2 ] ;
P [ i +2]= temp ;
temp=P [ i + 1 ] ;
P [ i +1]=P [ i + 3 ] ;
P [ i +3]= temp ;
}
//  Rotate third row 3 columns to right
i f ( k==12)
{
emp=P [ i ] ;
P [ i ]=P [ i + 1 ] ;
P [ i +1]=P [ i + 2 ] ;
P [ i +2]=P [ i + 3 ] ;
P [ i +3]= temp ; }
```

IV. EXPERIMENTAL RESULTS

The parallel implementation of the proposed work is implemented using OpenCL and tested on GPUs. The GPU implementation of AES algorithm is tested on AMD Radeon 8550M GPU and 8570G GPU. AMD APP Profiler is used for performance analysis, to evaluate the proposed work from the OpenCL run-time and AMD Radeon GPUs during the execution of an Open CL application. The figure 4 shows the screen-shots of GPU implementation of the proposed work. Various parameters like the GPU platform being executed currently, the global and local item sizes processed by the GPU, kernel occupancy of OpenCL application are considered to evaluate the proposed work.

Table I shows the execution time in milliseconds for the encryption and decryption functions. As shown in the Table I, First column represents the number of work items, column 2 and 3 lists the time taken for parallel implementation of the proposed work and column 4 and 5 represents the time taken for sequential implementation of AES algorithm. As the

number of work items increases from 256 to 1024, the execution time for GPU implementation of AES algorithm is rapidly decreasing when compared to sequential implementation. The best speedup achieved for GPU implementation is 98.8% when compared to sequential implementation for 1024 work items. When the number of work items are more than 1024, the execution time is increasing due to the communication and task interactions between the nodes. Due the communication latency and task interactions, the execution time for GPU implementation is increasing , but when compared to sequential implementation for 10240000 work items, the percentage of speedup is 99.6%.

Figure 4 shows the execution time for GPU implementation for two different GPU devices, such as 8550M and 8570G, where X-axis represents the work item sizes and Y-axis represents the time taken to execute the encryption and decryption function. This shows that, the same implementation has given different results of time based on the execution depending on the configuration of the devices indicating scope for further efficient results. The graph shows the improvement in execution time for two different devices, indicates that, there is a scope for further improvement to enhance speedup.

Table I: Sequential and Parallel Execution Time of AES algorithm.

Work Items	Execution time for Parallel Encryption (ms)	Execution time for Parallel Decryption (ms)	Execution time for Sequential Encryption (ms)	Execution time for Sequential Decryption (ms)
256	0.131	0.098	2	3
1024	0.129	0.104	10	10
2560	0.136	0.108	10	10
102400	2.3	2.3	460	710
25600	5.08	4.884	1246	1685
1024000	18.95	15.94	4488	7066
2560000	46.36	37.85	11589	17351
10240000	183.6	147	`44256	70823

Table II shows the execution time in milliseconds for the encryption and decryption functions on two different GPU devices. As shown in the Table II, First column represents the number of work items, column 2 and 3 lists the time taken for GPU implementation of the proposed work on 8550M device and column 4 and 5 represents the time taken GPU implementation of the proposed work on 8570G device.

Table II: Execution Time of GPU Implementation of AES Algorithm on 8550M and 8570G

Work-items	Encrypt 8550G(ms)	Decrypt 8550G(ms)	Encrypt 8570M(ms)	Encrypt 8570M(m)
256	0.131	0.098	0.093	0.09
1024	0.129	0.104	0.093	0.09
2560	0.136	0.108	0.095	0.089
10240	0.27	0.27	0.122	0.111
25600	0.62	0.65	0.263	0.241
102400	2.3	2.3	0.759	0.721
256000	5.08	4.884	1.74	1.597
1024000	18.95	15.94	4.885	4.532
2560000	46.36	37.85	12.119	11.223
10240000	183.6	147	48.4	44.7

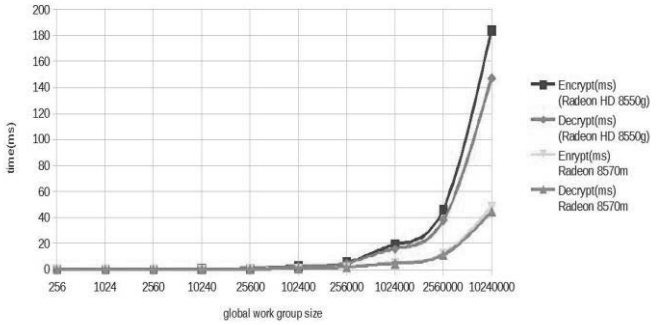

Fig. 4. Parallel Implementation 8550M vs 8570G

V. CONCLUSION

In this paper, the Acceleration of the symmetric key cryptography algorithm is enhanced using parallel implementation on GPGPUs with Open Computing Language (OpenCL). The Advanced Encryption Standard (AES) Algorithm is implemented using OpenCL and tested on GPU devices, such as 8550M and 8570G. The experimental results shows that, the parallel implementation of encryption algorithm tested on GPUs accelerates the speed when compared to sequential implementation of encryption algorithm. The percentage of speedup achieved on 1024 work items is 99.8%. compared to sequential implementation of AES algorithm

VI. REFERENCES

[1] D. Kirk, "Nvidia CUDA software and GPU parallel computing architecture," in ISMM, vol. 7, 2007, pp. 103– 104.

[2] Y. Boykov and V. Kolmogorov, "An experimental comparison of mincut/ max-flow algorithms for energy minimization in vision," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 26, No. 9, pp.1124–1137, 2004.

[3] A. Brunton, C. Shu, and G. Roth, "Belief propagation on the GPU for stereo vision" IEEE Canadian Conference on Computer and Robot Vision, 2006, pp. 76–76.

[4] G. Kedem and Y. Ishihara, "Brute force attack on Unix passwords with simd computer," in Proceedings of the 8th USENIX Security Symposium. Citeseer, 1999.

[5] http://www.nvidia.com/object/cuda opencl.html.

[6] Svetlin A. Manavski, "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography", In Proc. IEEE International Conference on Signal Processing and Communication, ICSPC, pp.65-68, 2007.

[7] J. Daemen, V. Rijmen, "AES Proposal: Rijndael", Original AES Submission to NIST, 1999. AES Processing Standards Publications.

[8] Jielin Wang, Weizhen Wang, Jianw ei Yang, Zhiyi Yu, Jun Han, Xiaoyang Zeng "Parallel Implementation of AES on 2.5D Multicore Platform with Hardware and Software Co-Design", IEEE 11th International Conference on ASIC (ASICON), 2015.