Implementation and Analysis of AES Encryption on GPU

Qinjian Li
Center for High Performance Computing
Northwestern Polytechnical University
Xi’an, CHINA

Chengwen Zhong
National Key Laboratory of Science and Technology on Aerodynamics Design and Research
Northwestern Polytechnical University
Xi’an, CHINA
zhongcw@nwpu.edu.cn

Kaiyong Zhao, Xinxin Mei, Xiaowen Chu
Inspur-HKBU Joint Lab of Heterogeneous Computing
Department of Computer Science
Hong Kong Baptist University
Hong Kong, CHINA

Abstract—**GPU is continuing its trend of vastly outperforming CPU while becoming more general purpose. In order to improve the efficiency of AES algorithm, this paper proposed a CUDA implementation of Electronic Codebook (ECB) mode encoding process and Cipher Feedback (CBC) mode decoding process on GPU. In our implementation, the frequently accessed T-boxes were allocated on on-chip shared memory and the granularity that one thread handles a 16 Bytes AES block was adopted. Finally, we achieved the highest performance of around 60 Gbps throughput on NVIDIA Tesla C2050 GPU, which runs up to 50 times faster than a sequential implementation based on Intel Core i7-920 2.66GHz CPU. In addition, we discussed the optimization under some practical application scenarios such as overlapping GPU processing and data transfer.**

Keywords-CUDA; GPU; AES; Electronic Codebook; Cipher Feedback; parellel computing

I. INTRODUCTION

As Internet becomes more and more important in commercial, government, finance and other key areas, there is an increasing demand of efficient data encryption solution. Advanced Encryption Standard (AES) [1] is a specification of symmetric cryptographic algorithm announced in 2001 by the National Institute of Standards and Technology (NIST) to replace the Data Encryption Standard (DES). It has been adopted by the U.S. government and is now used worldwide. AES has several advantages in security, flexibility, parallel potential and so on.

Over the years, the computer Graphics Processing Unit (GPU) is developing with a speed far exceeding Moore’s predictions and recent advancement in CPU performance. The high computing efficiency, outstanding parallelism as well as programmability developed recently, of GPU, provide a good platform of general purpose scientific and engineering computing other than graphic processing [2]. Nowadays, The GPU has been widely used in PCs, workstations, and data center servers. The various parallel products on the market promote high performance computing gradually into desktop or laptop areas. Since GPU has brought out as a general purpose applications’ accelerator, it has attracted attention from various research fields [3] [4] [5].

It has become increasingly clear that the speed of GPU-based implementation for symmetric block cipher surpasses that of FPGA implementation [6], for which the speedup depends heavily on programmers’ tuning techniques. GPU could act as a good co-processer in block cryptography, so the effort in developing novel approaches in implementation of AES encryption on GPU is of great significance in practice.

II. CUDA

Compute Unified Device Architecture (CUDA) is a parallel computing platform and programming model invented by NVIDIA. Making use of NVIDIA GPU’s parallel computing engine, CUDA is more efficient at solving many complex computational tasks than CPUs.

For CUDA users, GPU behaves like a single instruction multiple thread (SIMT) computing device consisting of a group of streaming multi-processors (SM). A Thread is the basic implementation unit of CUDA program model. Each thread executes the same code, which is called as CUDA kernel. While similar to threads in operating systems, CUDA threads are extremely light weight threads. Due to the efficient hardware support, CUDA programmers can ignore the cost of generating and scheduling these threads. CUDA threads are organized hierarchically. A certain number of threads can be composed of a thread block (BLOCK) up to three dimensions. Certain thread blocks can be composed of one-dimensional or two-dimensional grid (GRID). Warp is the basic unit of SM managing, scheduling and executing threads. Threads in the same warp start at the same time at the same program address. And each time, only one thread is executed.

CUDA memory hierarchy includes registers, shared memory, constant memory, texture memory and global memory. CUDA provides explicit methods to organize memory architecture. Using this storage hierarchy effectively is of great significance to avoid GPU's memory bandwidth bottleneck as well as to improve the overall performance of the applications.

Readers, who are interested, please find the details of CUDA program model and its software and hardware structure in [7].

III. Overview of AES

A. *The Encryption Scheme*

The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plaintext into the final output of ciphertext. Each round consists of several processing steps: *AddRoundKey*, *SubBytes*, *ShiftRows* 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. The number of transformation rounds, however, is decided by the size of encryption key. The ciphertext can be transformed back into the original plaintext by applying a set of reverse rounds using the same encryption key. Although 128-bit, 192-bit, or 256-bit key size can be selected, we discuss only 128-bit in this paper. Our work can be easily extended to different key sizes.

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: T_0 T_1, T_2 and T_3. 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:

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j+1}] \oplus T_2[a_{2,j+2}] \oplus T_3[a_{3,j+3}] \oplus k_j \qquad (1)$$

where a 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.

According to the compact solution above, it only takes 4 exclusive-OR operations and 4 table lookups per column per round. Although the transformation order of AES decryption and encryption are different, an equivalent version of decryption algorithm and encryption algorithm has the same structure. The details of AES algorithm can be found in [8].

B. *The work modes of AES block cypher*

In cryptography, block cipher modes of operation allow encrypting more than one plaintext blocks with the same key and ensure its security. In 2001, NIST released five recommended modes of AES operation [9]. These modes have different characteristics of parallelization caused by their algorithm difference. A simple comparison of these modes is listed in Table I.

The input to the encryption processes of the CBC, CFB, and OFB modes includes a data block called the initialization vector (IV) in addition to the plaintext. An initialization vector must be generated for each execution of the encryption operation, and the same vector is necessary for the corresponding execution of the decryption operation.

TABLE I. COMPARISON OF FIVE RECOMMENDED MODES

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.	Suitable for 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

It is notable that in CBC encryption, the input block to each forward cipher operation (except the first) depends on the result of the previous forward cipher operation, so the forward cipher operations cannot be performed in parallel. In CBC decryption, however, the input blocks for the inverse cipher function, i.e., the ciphertext blocks, are immediately available, so that multiple inverse cipher operations can be performed in parallel.

IV. Related Work

Harrison et al. presented the first CPU competitive implementation of AES on GPU [10]. The authors achieved AES encryption with 870 Mbps throughput on Geforce 7900GT using Direct X9.

Manavski implemented CUDA-AES and achieved 8.28 Gbps throughput using Geforce 8800GTX in [11]. Authors of this paper took the table lookup approach and stored pre-computed T-boxes in GPU constant memory.

Harrison et al. [12] presented an AES implementation of CTR mode of operation on an NVIDIA G80 GPU. It is notable that the authors used all available types of on-chip memory (shared memory, constant memory and texture memory) to store lookup tables and found that using shared memory lead to best performance.

Di Biagio et al. [13] also proposed an implementation of CTR mode of operation using Geforce 8800GT. Authors of this paper indicated that using on-chip shared memory rather than constant memory to store T-boxes would bring a considerable performance improvement. In addition, they carried out a tentative discuss on parallel processing granularity. Also Chonglei Mei et al. [14] implemented AES encryption on Geforce 9200M in a similar way.

Liu et al. [15] implemented AES on a Geforce 9800 GPU to investigate the program behavioral characteristics and their impacts on the performance. They found the following four observations: (1) The number of threads can affect the overall performance. (2) Larger shared memory capacity is necessary to hold the lookup tables. (3) Data stored in global memory are organized to generate burst access to global memory. (4) Communication between CPU and GPU might be a program bottleneck.

Nishikawa et al. conducted a study on granularity optimization for GPU-based AES encryption in [16]. They defined 16Bytes/thread as the granularity of one thread to process one plaintext block and other similar granularities such as 4Bytes/thread and 1Byte/thread.

Keisuke Iwai et al. [17] studied the influence of memory usage scheme as well as the parallel processing granularity on the GPU-based AES encryption efficiency. They affirmed the 16Bytes/thread granularity and storing T-boxes in shared memory lead to best encryption throughput. Most recently, Nishikawa et al. [18] evaluated AES and other three ciphers based on previously reported insights using NVIDIA Tesla C2050 GPU.

Table II summarizes the peak performances and implementation environments of all previous works. Since the CBC mode AES is wildly used (such as the encryption/decryption algorithm of RAR files) but its parallel implementation of decryption has not received any discussion, this paper is motivated to discuss the GPU-based implementation of CBC mode AES decryption in addition to the basic ECB mode AES encryption.

TABLE II. PERFORMANCE OF PREVIOUS WORKS

Reference	Compute Device	Mode	Throughput
Harrison et. al.[10]	Geforce 7900GT	ECB	870Mbps
Manavski et.al.[11]	Geforce 8800GTX	Unknown	8.28Gbps
Harrison et. al.[12]	NVIDIA G80	CTR	6.91Gbps
Di Biagio et.al.[13]	Geforce 8800GT	CTR	12.5Gbps
Chonglei Mei et.al.[14]	Geforce 9200M	Unknown	6.4Gbps
Nishikawa et.al.[16]	Geforce GTX285	ECB	6.25Gbps
Keisuke Iwai et.al.[17]	Geforce GTX285	ECB	35Gbps
Nishikawa et.al.[18]	Tesla C2050	ECB	50.6Gbps

V. OUR GPU-BASED AES APPROACH

A. *Parallel granularity*

Our implementation also adopts the 16 bytes per thread parallel granularity, which means that each thread is mapped to a 16 bytes AES block and the blocks are processed concurrently. Since there is no need of parallel execution for each thread computation, or synchronization and sharing data among executed threads, the threads can complete processing on their own registers.

B. *Memory usage scheme*

Both the Round Keys values and T-boxes are read-only data, and shared by all threads, in line with constant memory features. However, the T-boxes access is random, which is not the same as the Round Keys. Considering that fast T-boxes access is of great significance in AES algorithm, and the T-box size is only 4KB, we store T-boxes in on-chip shared memory to ensure fast memory access.

C. *ECB mode encryption*

In ECB mode encryption, each thread has to load four continuous 32-bit words (16 bytes block) into the global memory for processing. Due to the global memory access of one 32-bit word by threads within a half-warp is separated and non-sequential, it would lead to low global memory bandwidth. To solve this problem, we can reorder the plaintext in host memory before they are loaded onto device memory. The reorder process is as following: divide each plaintext block into four 32-bit words, store them column-majored in a two dimensional memory space, so that different 32-bit words of different AES blocks are consistent in host linear memory space. Then all 16 threads in a half-warp could read the consistent 32-bit words in global memory, thus the memory access is coalesced. The reorder process could be done efficiently and naturally when loading plaintext from disk files. There is another approach make access to global memory coalesced, for example, Liu et al. used a CUDA built-in vector data type to combine 16 linear bytes into a 128-bit 'int4' variable in [15].

D. *CBC mode AES decryption*

The CBC mode decryption implementation is almost the same with the ECB mode encryption mentioned above except that the decrypted blocks need to exclusive-OR the previous ciphertext blocks to recover the exact plaintext. In order to avoid reloading the ciphertext blocks from the low bandwidth global memory for the exclusive-OR operations, we can store the ciphertext blocks, which had already been loaded in the registers by each thread, into share memory. Thus, when the inverse cipher function has been applied to the corresponding cipher block, the resulting block can exclusive-OR with its previous ciphertext block in share memory to recover the exact plaintext with high efficiency.

VI. EXPERIMENTAL RESULTS AND ANALYSIS

A. *Environment*

TABLE III. SPECIFICATION OF EXPERIMENT PLATFORM

Platform	Inspur NF5588
CPU	Intel(R) Xeon(R) E5620@2.4GHz
Memory	24GB
OS	Windows 7(64-bit)
Compiler	Visual C++9.0(option –O2)
GPU Accelerator	NVIDIA Tesla C2050
GPU Memory	3GB
PCI Bus	PCI-E 2.0×16
CUDA Compiler	Nvcc Ver4.0

Table III shows the configuration of our experiment platform. Our early results show that each block contents 512 threads can get the 100% occupancy rate of the SM and the highest computational speed. Therefore, we use 512 threads in a block as our default setting. Additionally, we set the capacity of shared memory and L1 cache in Tesla C2050 to 48 KB and 16 KB and enable the "ECC" check by default.

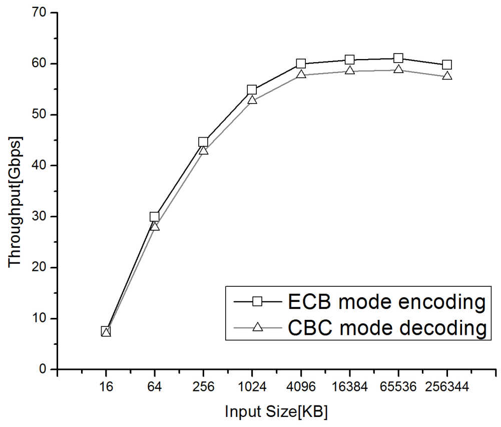

Figure 1. AES throughput of different input size

B. *Throughput*

Fig.1 gives the throughput of ECB mode encryption and CBC mode decryption of different input size.

In CBC mode AES decryption, each thread needs 8KB (512x16Byte) shared memory to store ciphertext blocks for exclusive-OR operations. In NVIDIA Tesla C2050, each SM can offer 48KB shared memory, except the 4KB for T-boxes and other space for few parameters, each SM could still afford 3 thread blocks working concurrently. The CBC mode decryption could make full use of SMs as same as ECB mode encryption. The consistent trend of two curves in Fig.1 shows that.

Our experiment achieves coalesced global memory access and high speed grouping on low latency registers. Compared with the theoretical 1.03Tflops compute performance and 144GB/s memory bandwidth, however, the highest throughput we have reached is 60GB/s. To analyze the execution process, we can find that the compute task in AES is not that much. The main reason of low utilization rate may caused by frequent random access to T-boxes in shared memory, which brings lots of bank conflicts [7].

Finally, we get a maximum of 50 times speedup compared with Core i7-920 2.66GHz CPU implementation which achieved 1.2Gbps throughput [17].

C. *Overlapping data transfer and processing*

In order to study the parallel computing capacity of GPU, we do not take data transfers time into account when calculating the throughput. However, it is still necessary to consider the data transfers cost to evaluate actual promotion of general GPU computing.

When GPU is working, the data should transfer from host memory to device memory through PCI-E bus. Although PCI-E 2.0×16 can provide 8GB/s throughput up and down in theory, there is only about 3GB/s in our real experiments. In our experiment, the time of data transfer from host to device is more than the time of computing (approximately 2 fold). Including data transfers, the maximum throughput is only 11.3Gbps. Therefore, we try to use stream mechanism provided by CUDA to overlap data transfer and kernel execution in decryption process.

TABLE IV. TOTAL COST USING STREAMS

Data Scales(KB)	**Time spent with different stream numbers(ms)**					**Pined memory allocation time(ms)**
	0	***2***	***4***	***8***	***16***	
256	0.44	0.20	0.19	0.23	0.29	0.3
1024	0.96	0.45	0.43	0.60	0.67	0.31
4096	3.15	2.23	2.10	2.06	2.09	0.93
16384	12.58	6.69	6.12	5.88	7.94	3.36
65536	48.06	35.12	32.83	23.45	31.23	13.12
262144	189.5	140.2	97.03	92.47	124.36	58.5

Table IV shows the time spent with different stream numbers and the time of pined memory application in decryption process. Technically speaking, using streams could hide the small parts of data transfer and kernel execution, and the total time cost would decrease with more streams. However, as table IV shows, the total computing time is minimized with 4 or 8 streams, while increases with stream number grow. The reason is that besides the inherent cost by streams, the transfer speed through PCI-E bus of overlapping data as well as kernel execution decreases heavily. This is proved by Visual Profiler analysis [19].

Using stream asynchronous transfer data must use the page-lock memory. Note that the page-locked memory allocation takes time can't be ignored. Consider this case, the decryption of 256MB cipher can get the highest actual throughput is only 14.22Gbps. And, decrypt the small file (e.g. 256KB) with streams will reduce the actual throughput, so we must decide whether to use overlapping technique or not, according to the file size.

As shown in table IV, decrypting 256MB cipher text achieves the highest 23.2Gbps throughput with 8 streams, corresponding to the result of [17]. Because of the limitation of PCI-I bandwidth, the actual throughput of the GPU-based AES algorithm hardly increases with better compute capacity GPUs.

VII. CONCLUSION AND FUTURE WORK

As described in this paper, 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.

The maximum 60Gbps throughput of AES-CBC decryption on Tesla C2050 indicates that GPU-based AES can obtain one order magnitude speedup to CPU one. The brute force of lowest 128-bit AES encryption algorithm on GPU is still not practical, but a wide range of GPU in PC can be used as general-purpose accelerator to improve the security of network applications.

Lastly, many cipher algorithms exist in the real world in addition to the algorithms implemented in this paper. Future research might investigate other common encryption algorithms.

ACKNOWLEDGMENT

This project was supported by the graduate starting seed fond of Northwestern Polytechnical University (Grant No. Z2012126).

Appendix A: Kernel function encrypt_Kernel

Algorithm 1 ECB mode AES encryption

```
__global__ static void encrypt_Kernel( u32_t *dev_input,
u32_t* dev_output, size_t pitch_a,  size_t pitch_b, u32_t* dev_sm_te1,
u32_t* dev_sm_te2,  u32_t* dev_sm_te3,  u32_t* dev_sm_te4  )
{
    // local thread index and global index
    int tid = threadIdx.x;
    int index = THREAD_NUM*(BLOCK_NUM*blockIdx.y+blockIdx.x)
              + threadIdx.x;
    u32_t w1,w2,w3,w4,s1,s2,s3,s4;

    // store the T-boxes and sbox in share memory.
    __shared__ u32_t sm_te1[256], sm_te2[256], …
    __shared__ u8_t sm_sbox[256];

    if (tid<256){
          //load dev_sm_te1, dev_sm_te2, dev_sm_te3, dev_sm_te4 and
          // const_sm_sbox to share memory variables sm_te1, sm_te2,
          //sm_te3, sm_te4 and sm_sbox;
          ...
    }
    //load the cipher blocks, all the global memory transactions are
    //coalesced. The original plain text load from files, due to the read
    //procedure reverse the byte order of the 32-bit words, So a reverse
    //process was necessary.
    w1 = dev_input[index];
    w1= ((w1>>24)&0xFF)|(( w1>>8)&0xFF00)|( (w1<<8)&0xFF0000)|
        ( (w1<<24)&0xFF00 0000;
    ...
    // first round AddRoundKey: ex-or with round key
    w1 ^= const_m_ke[0];    ...
    // round transformation: a set of table lookups operations.
  #pragma unroll
  for (int i = 1; i < 10; i++) {
          s1 = (sm_te1[w1 >> 24] ^ sm_te2[(w2 >> 16) & 0xFF] ^
               sm_te3
               [(w3 >> 8) & 0xFF] ^ sm_te4[w4 & 0xFF]) ^
               const_m_ke[i*4];
          w1 = s1;
          ...
   }    //The final round doesn't include the MixColumns
   w1  = (u32_t)(sm_sbox[ s1 >> 24 ]) << 24;     //SubBytes and ShiftRows
   w1 |= (u32_t)(sm_sbox[(s2 >> 16)& 0xFF]) << 16;
   w1 |= (u32_t)(sm_sbox[(s3 >> 8)& 0xFF]) << 8 ;
   w1 |= (u32_t)(sm_sbox[s4 & 0xFF]);
   w1 ^= const_m_ke[ROUNDS*4];                //AddRoundKey
   w1=( (w1>>24)&0xFF)|(( w1>>8)&0xFF00)|( (w1<<8)&0xFF0000)
       |( (w1<<24)&0xFF000000;
   dev_output[index] = w1 ;                   //store the cipher text
   ...
}
```

Appendix B: Kernel function decrypt_Kernel

Algorithm 2 CBC mode AES decryption

```
__global__ static void decrypt_Kernel( u32_t *dev_input,
u32_t* dev_output, size_t pitch_b,  size_t pitch_b, u32_t* dev_sm_td1,
u32_t* dev_sm_td2,  u32_t* dev_sm_td3,  u32_t* dev_sm_td4  )
{
    // local thread index and global index
    int tid = threadIdx.x;
    int index = THREAD_NUM*(BLOCK_NUM*blockIdx.y+blockIdx.x)
              + threadIdx.x;
    u32_t w1,w2,w3,w4,s1,s2,s3,s4;

   // store the T-boxes and sbox in share memory.
   __shared__  u32_t sm_td1[256], sm_td2[256], …
   __shared__  u8_t sm_isbox[256];
   // store the ciphertext blocks for the later ex-or operations.
   __shared__ u32_t iv_1[THREAD_NUM], iv_2[THREAD_NUM], ... ;

   if (tid<256){
          //Load dev_sm_td1, dev_sm_td2, dev_sm_td3, dev_sm_td4 and
          // const_sm_sbox to share memory variables sm_td1, sm_td2,
          //sm_td3, sm_td4 and sm_isbox;
          ...
    }
    // first thread of each block store the cipher blocks that corresponding
    // to the last thread of the previous block to share memory.
    if(tid  = = 0 & index !=0 ){
          w1 = dev_input[index-1];
          iv_1[0] = ( (w1>>24)&0xFF)|(( w1>>8)&0xFF00)|
                 ( (w1<<8)&0xFF0000)|( (w1<<24)&0xFF000000);
          ...
    }
    //the first thread of the whole gird load the initialization vector (IV).
    if(index==0){
          iv_1[0] = dev_iv[0];
          ...
    }
    //Load the blocks and reverses the byte order of the 32-bit words.
    w1 = dev_input[index];
    w1= ((w1>>24)&0xFF)|(( w1>>8)&0xFF00)|( (w1<<8)&0xFF0000)|
        ( (w1<<24)&0xFF00 0000;
    ...
    if(tid<THREAD_NUM-1){
          //store the current cipher block in share memory for later usage.
          iv_1[tid+1] = w1;
          ...
    }
    if(index==THREAD_NUM*BLOCK_NUM-1){
          //the final thread of  a grid store the last cipher block to
          //global memory for next kernel.
          dev_iv[0] = w1;
          ...
    }
    // AddRoundKey: ex-or with round key
    w1 ^= const_m_ke[0];
    ...
    // round transformation: a set of table lookups operations.
#pragma unroll
for (int i = 1; i < 10; i++) {
       s1 = (sm_td1[w1 >> 24] ^ sm_td2[(w2 >> 16) & 0xFF] ^ sm_td3
            [(w3 >> 8) & 0xFF] ^ sm_td4[w4 & 0xFF]) ^
             const_m_ke[i*4];
       w1 = s1;
       ...
}
   w1 = (u32_t)(sm_isbox[ s1 >> 24 ]) << 24;  //SubBytes and ShiftRows
   w1 |= (u32_t)(sm_isbox[(s4 >> 16)& 0xFF]) << 16;
   w1 |= (u32_t)(sm_isbox[(s3 >> 8)& 0xFF]) << 8 ;
   w1 |= (u32_t)(sm_isbox[s2 & 0xFF]);
   //AddRoundKey and ex-or with the corresponding ciphertext to recover
   //the the exact plaintext.
   w1 ^= iv_1[tid]^const_m_kd[ROUNDS*4];
   w1 = ( (w1>>24)&0xFF)|(( w1>>8)&0xFF00)|( (w1<<8)&0xFF0000)
        |( (w1<<24)&0xFF000000;
   dev_output[index] = w1 ;  //store the results
   ...
}
```

REFERENCES

[1] National Institute of Standards and Technology (NIST), "FIPS-197 : Advanced Encryption Standard (AES)", 2001.

http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] WU EH, "State of the art and future challenge on general purpose computation by graphics processing unit," Journal of Software, vol. 15(10), 2004, pp.1493-1504(in Chinese with Englisth abstract).

[3] X.-W. Chu, K. Zhao, and M. Wang, "Massively Parallel Network Coding on GPUs," Proc. of IEEE IPCCC'08, Austin, Texas, USA, Dec 2008. pp. 2070-2078.

[4] X.-W. Chu, K. Zhao, and M. Wang, "Practical Random Linear Network Coding on GPUs," Proc. of IFIP Networking'09, Archen, Germany, May 2009.

[5] Y. Li, K. Zhao, X.-W. Chu, and Jiming Liu, "Speeding up K-Means Algorithm by GPUs," Proc. of 2010 IEEE Internactional Conference on Computer and Information Technology(IEEE CIT 2010), July 2010, Bradford, UK. pp.115-122.

[6] H.Qin, T.Sasao, and Y.Iguchi, "An FPGA design of AES encryption circuit with 128-bit keys," Proc. of 15th IEEE/ACM Great Lakes Symposium on VLSI(GLSVLSI'05), 2005, pp.147-152.

[7] NVIDIA, CUDA Programming Guide, Version 4.0, 2011.

http://developer.download.nvidia.com/compute/cuda/4_0/toolkit/docs/CUDA_C_Programming_Guide.pdf

[8] W. Stallings, Cryptography and Network Security: Principles and Practices, 3rd ed., Beijing: Publishing House of Electronics Industry, 2004, pp.103-127.

[9] M. Dworkin, "Recommendation for block cipher modes of operation: methods and techniques," Gaithersburg: U.S.Doc/NIST,2001.

http://csrc.nist.gov/publications/nistpubs/800-38a/sp800- 38a.pdf

[10] Owen. Harrison, J. Waldron, "AES Encryption Implementation and Analysis on Commodity Graphics Processing Units," Proc. of the 9th Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007), Vienna, Austria, September 10-13, 2007. LNCS, Vol. 4727/2007, Pages 209-226.

[11] S.A.Manavski, "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography," Proc. of 2007 IEEE International Conference on Signal Processing and Communication(ICSCP 2007), IEEE press, Nov. 2007, pp.65–68.

[12] Owen Harrison, John Waldron, "Practical Symmetric Key Cryptography on Modern Graphics Hardware," Proc. of the 17th conference on Security symposium. San Jose, CA, 2008, pp. 195-209.

[13] A. D. Biagio, A. Barenghi, G. Agosta, and G. Pelosi, "Design of a parallel AES for graphics hardware using the CUDA framework," Proc. of 2009 IEEE International Parallel and Distributed Processing Symposium, IEEE press, 2009, pp.1–8.

[14] Chonglei Mei, Hai Jiang, Jenness, "CUDA-based AES parallelization with fine-tuned GPU memory utilization," Proc. of 2010 IEEE International Symposium on Parallel Distributed Processing Workshops and PhD Forum(IPDPSW), IEEE press, 2010, pp.1-7.

[15] Gu Liu, Hong An, Wenting Han, Guang Xu, Ping Yao, Mu Xu, Xiurui Hao, Yaobin Wang, "A Program Behavior Study of Block Cryptography Algorithms on GPGPU," Proc. of Frontier of Computer Science and Technology, Fourth International Conference on FCST '09, 2009, pp.33–39.

[16] N.Nishikawa, K. Iwai, and T. Kurokawa, "Granularity optimization method for AES encryption implementation on CUDA(in Janpanese)," IEICE technical report. VLSI Design Technologies (VLD2009-69), Kanagawa, Japan, Jan. 2010, pp.107-112.

[17] K. Iwai, T. Kurokawa, and N. Nishikawa, "AES encryption implementation on CUDA GPU and its analysis," Proc. of 2010 First International Conference on Networking and Computing, 2010, pp.209-214, doi:10.1109/IC-NC.2010.49.

[18] N.Nishikawa, K Iwai, and T.Kurokawa, "High-Performance Sysmmetric Block Ciphers on CUDA," Proc. of 2011 Second International Conference on Networking and Computing(ICNC), 2011, pp.221-227.

[19] NVIDIA, Compute Visual Profiler User Guide, Version 2.0, 2010.

http://developer.nvidia.com/nvidia-gpu-computing-documentation