International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 2, Number 2, pages 251–268, July 2012

High-Performance Symmetric Block Ciphers on Multicore CPU and GPUs

Naoki Nishikawa, Keisuke Iwai, and Takakazu Kurokawa
Dept. of Computer Science, National Defense Academy of Japan
1-10-20 Hashirimizu Yokosuka Kanagawa 239-8686, Japan

Received: January 30, 2012
Revised: May 16, 2012
Accepted: June 19, 2012
Communicated by Sayaka Kamei

Abstract

As the data protection with encryption becomes important day by day, the encryption processing using General Purpose computation on a Graphic Processing Unit (GPGPU) has been noticed as one of the methods to realize high-speed data protection technology. GPUs have evolved in recent years into powerful parallel computing devices, with a high cost-performance ratio. However, many factors affect GPU performance. In earlier work to gain higher AES performance using GPGPU in various ways, we obtained the following two technical viewpoints: (1) 16 Bytes/Thread is the best granularity (2) Extended key and substitution table stored in shared memory and plaintext stored in register are the best memory allocation style.

However, AES is not the only cipher algorithm widely used in the real world. For this reason, this study was undertaken to test the hypothesis that these two findings are applicable to implementation of other symmetric block ciphers on two generation of GPU. In this study, we targeted five 128-bit symmetric block ciphers, AES, Camellia, CIPHERUNICORN-A, Hierocrypt-3, and SC2000, from an e-government recommended ciphers list by the CRYPTography Research and Evaluation Committees (CRYPTREC) in Japan. We evaluated the performance of these five symmetric block ciphers on the machine including a 4-core CPU and each GPU using three method: (A) throughput without data transfer, (B) throughput with data transfer and overlapping encryption processing on GPU, (C) throughput with data transfer and non-overlapping encryption processing on GPU. Results demonstrate that the throughput of implementation of SC2000 in method (A) on Tesla C2050 achieved extremely high 73.4 Gbps. Additionally, the throughput obtained using methods (B) and (C) deteriorated to 33.4 Gbps and 18.3 Gbps, respectively. Method (B) showed effective throughput with an approximately 4.7 times higher speed compared to that obtained when using 8 threads on a 4-core CPU.

Keywords: GPU, CUDA, Multicore CPU, Symmetric Block Cipher

1 Introduction

As the data protection with encryption becomes important day by day, the encryption processing using GPGPU has been noticed as one of the methods to realize high-speed data protection technology. GPUs have evolved in recent years into powerful parallel computing devices, with a high cost-performance ratio. However, many factors should be undertaken to improve the performance on GPUs, for example the granularity, memory allocation style, the number of threads, and the number of thread blocks. This fact makes it difficult to detect what parameters can extract the GPU's performance. In our previous work to gain higher AES performance using GPGPU in various ways,

we obtained the following two technical viewpoints: (1) 16 Bytes/Thread is the best granularity (2) extended key and a substitution table stored in shared memory is the best memory allocation style[10].

However, AES is not the only cipher algorithm widely used in the real world. For example Camellia is already adopted as one of the cipher algorithms embedded in Firefox internet browser. For this reason, this study was undertaken to test the hypothesis that these two findings are applicable to the implementation of other symmetric block ciphers on two generations of GPU. The algorithm of modern symmetric block ciphers resembles that of AES: integer operations, logical operations, and several table substitutions. Therefore, herein, we targeted five 128-bit symmetric block ciphers, AES[12], Camellia[2], CIPHERUNICORN-A[18], Hierocyrpt-3[14], and SC2000[17], from an e-government recommended ciphers list by the CRYPTography research and evaluation committees (CRYPTREC) in Japan[1].

2 CUDA

CUDA is a GPGPU development environment released by Nvidia Corp. In the CUDA environment, the GPU is hidden as a parallel computing device. As shown in Figure 1, the GPU has N multiprocessors (MP) and a global memory. Each MP has M scalar processors (SP), a shared memory, several 32-bit registers, and a shared instruction unit. The shared memory is multi-bank type of multiport memory. In addition, the global memory has a special area called constant memory. When reading common data among themselves such as constant, threads get cache effect. We chose an Nvidia Tesla C2050 and a Geforce GTX 285 from the CUDA GPU family. Tesla C2050 incorporates the latest Fermi architecture, including 14 MPs; one MP incorporates 32 SPs. In Tesla C2050, the capacity of the shared memory is 48 KB. By contrast, Geforce GTX 285 has conventional GT200 architecture including 30 MPs, and one MP incorporates 8 SPs. In Geforce GTX 285, the capacity of the shared memory is 16 KB.

Parallel processing on CUDA is attributed to many-thread parallelism, and the resources in MPs are assigned evenly among many active threads. Threads can be processed effectively provided that the number of threads is in multiples of 32. Especially, the SP pipeline can be kept filled if more than 192 threads are activated, which engenders a considerable performance increase [19]. In addition, GPU has a DMA controller. Therefore, we can divide data into an arbitrary number of segments. Then we overlap the data processing with the data copy.

Moreover, in the architecture of CUDA GPUs a thread block is switched to other one in round-robin fashion to hide the memory access latency, unlike traditional CPU architectures. More specifically, the instructions by the latter thread are executed under the background of the memory access instruction issued by the former one. This is a unique mechanism of hiding the memory access latency, based on a feature of the GPU architecture specialized for many-thread execution.

3 Algorithms of targeted symmetric block ciphers

3.1 Overview

The CRYPTREC in Japan publishes an e–government recommended ciphers list, including several excellent block ciphers, for example AES[12], Camellia[2], CIPHERUNICORN-A[18], Hierocrypt-3[14], and SC2000[17].

For speeding up of encryption processing, a strategy that optimizes repetitive parts in each cipher algorithm is valid. In general, the method to take advantage of precomputing substitution table is available for the optimization of symmetric block cipher on software. This method have also advantage in the optimization on CUDA, because a GPU has relatively large amount of memories to store substitution tables. In this section, we show the original algorithm of each cipher first; then illustrate its optimized algorithm.

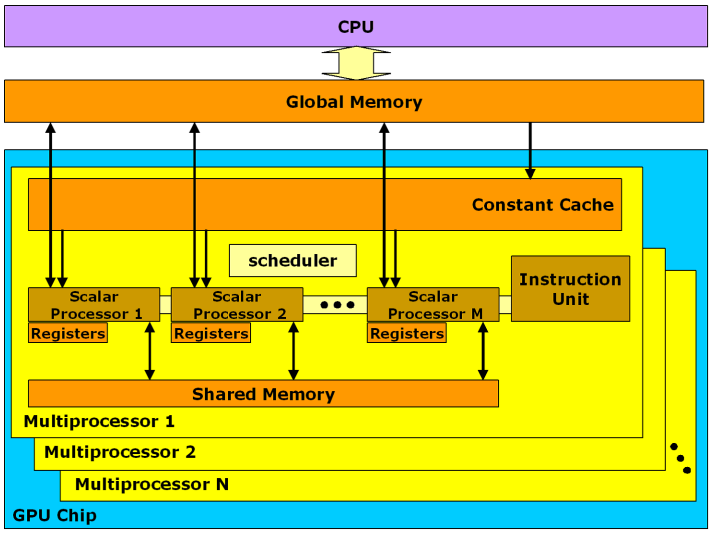


Figure 1: CUDA Architecture.

3.2 Encryption mode

Several modes can be selected at the time of encryption of multiple plaintexts. ECB (Electonic Code Book), CTR (CountTeR), or XTS (Xor-encrypt-xor Tweakable code book mode with cipher text Strealing)[9] among them are known as parallelizable modes. ECB is the mode that a single key is applied to all plaintexts. CTR is the mode that a key stream generated from a secret key is combined to plaintexts. In this mode, the generation of the key stream is conducted in the same manner as ECB. XTS is the mode for the disk encryption, specified in IEEE 1619-2007 standard. In this mode, plaintexts are encrypted with sector number of disk and two 128-bit secret keys using two ECB modes. For these reason, the performance increase of symmetric block ciphers on GPUs can be comprehensively evaluated by means of ECB mode. Therefore, in this paper, we will discuss ECB mode as parallel computing.

3.3 AES

AES[12] is a 128-bit block cipher, of which the structure is an SPN network. Although 128-bit, 192-bit, or 256-bit key size can be selected, we discussed only 128-bit in this paper. The length of the expanded key generated from 128-bit key is $10 \times$ 128-bit.

Its algorithm of the 128-bit key defines 10-round processes. Each round includes four transformations: SubBytes, ShiftRows, MixColumns, and AddRoundKey. The final round differs slightly from the other rounds: it does not include MixColumns.

In AES, a round process can be combined into a transformation simply using a lookup table called T-box and XOR operation[8]. Letting a be the round input, which is divided into four inputs a_0, a_1, a_2, a_3, each of which consists of 32 bits, the round output e is represented as

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

where T_0, T_1, T_2, and T_3 are lookup tables and k_j is the j-th column of a roundkey. This algorithm requires only four lookup table transformations and four XOR operations.

3.4 Camellia

Camellia[2] is a 128-bit block cipher, of which the structure is a Feistel network. Although 128-bit, 192-bit, or 256-bit key size can be selected, we discussed only 128-bit in this paper. The length of

expanded key size generated from 128-bit key is 26 × 64-bit. In Camellia, the algorithm of 128-bit key defines 18 round processes. Each round includes an F-function, in which 64-bit plaintext and 64-bit extended key become the input. After the input, the plaintext is combined with the extended key by XOR; then replaced by S-function consisting of an S-table. After the S-function, the plaintext is stirred by P-function consisted of 8-bit XOR; then the intermediate of 64-bit plaintext becomes the output of F-function. The FL and FL^{-1} functions, consisting of 32-bit AND, OR, XOR, and cyclic left shift, are inserted per six rounds.

The optimized architecture of Camellia is presented in Figure 2. In Camellia, the F-function occupies the most parts of the data randomization procedure. The permutation network in the F-function can be combined with the S-function, which combined S-table is called SP-table. Then, the F-function can be deformed to the algorithm including eight SP-tables with 8-bit input and 32-bit output.

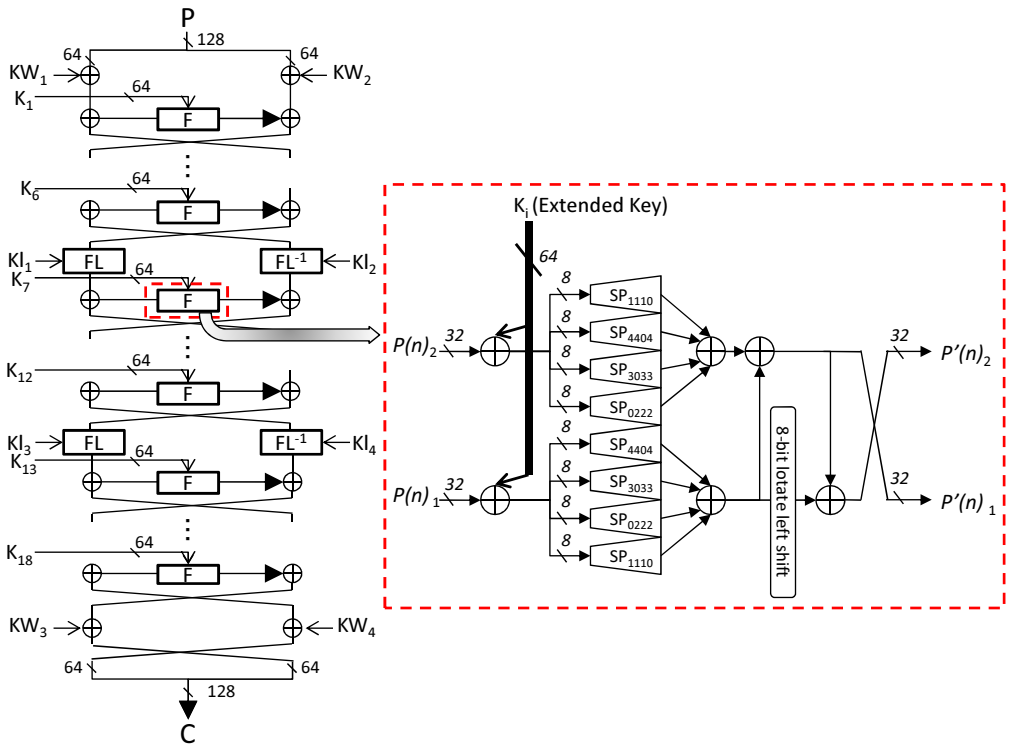


Figure 2: Optimized Camellia (Data randomization procedure).

3.5 CIPHERUNICORN-A

CIPHERUNICORN-A[18] is a 128-bit block cipher with a Feistel network structure, as is Camellia's. In CIPHERUNICORN-A, 128-bit, 192-bit, or 256-bit key size can be selected. However, the length of the extended key is constant 18 × 128 bit, irrespective of the length of the secret key. The data randomization procedure also defines constant 16 processes, irrespective of the length of the secret key. In CIPHERUNICORN-A, each round includes an F-function, in which 64-bit plaintext and 64-bit extended key become the input. After the input, the plaintext is added with a half of the extended key, i.e. 2 × 32 bit. After the addition, the plaintext is stirred by XOR, rotate shift, constant multiplication, and Tn-function. The other half of extended key, 2 × 32 bit, is stirred by table substitution, constant multiplication, etc. Finally in F-function, the stirred 2 × 32 bit plaintexts are combined with the 2 × 32 bit extended key by XOR; it then becomes the output as the intermediate of the plaintext. Additionally, the Tn-function is a function, in which 8-bit out of

32-bit of plaintext specifies the location of four different tables; then four different 8-bit become the output. Finally, the four different 8-bit are combined into 32-bit using shift operation.

The optimized architecture of CIPHERUNICORN-A is shown in Figure 3. In CIPHERUNICORN-A, Tn-function occupies the most part of the F-function. In the Tn-function, the substitution table with 8-bit I/O can be transformed into a new substitution table with 8-bit input and 32-bit output. Using this new table, the accesses to substitution table are optimized from four accesses with shift of operations to one access without a shift operation.

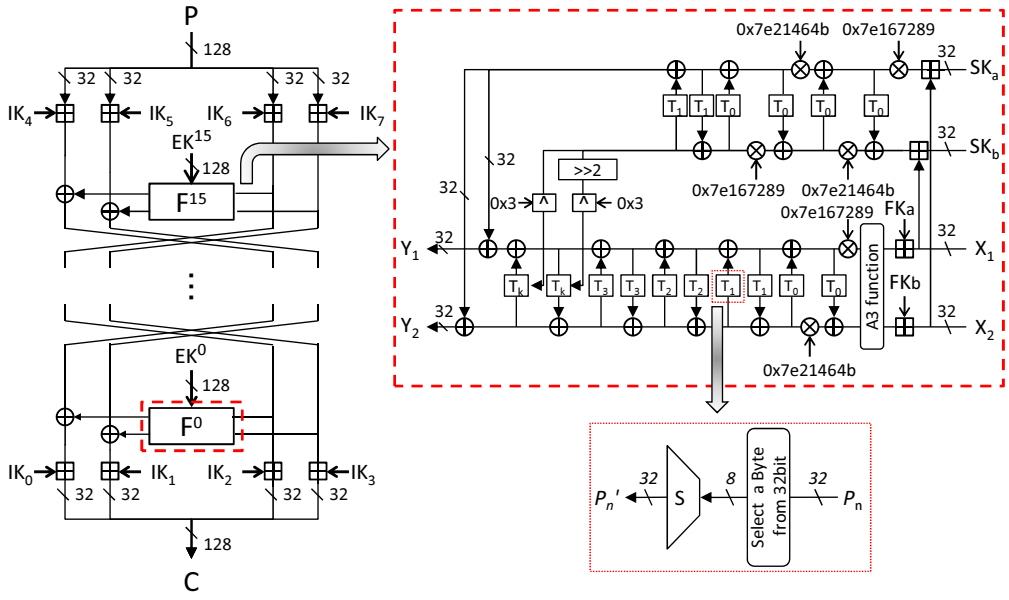


Figure 3: Optimized CIPHERUNICORN-A (Data randomization procedure).

3.6 Hierocrypt-3

Hierocrypt-3[14] is a 128-bit block cipher, of which the structure is a nested SPN network. Although 128-bit, 192-bit, or 256-bit key size can be selected, we discussed only 128-bit in this paper. The length of the expanded key generated from 128-bit key is 7×256 bit. In Hierocrypt-3, the data randomization procedure in 128-bit key defines seven round processes. Each round includes a ρ-function, in which S-table substitution, XOR with extended key, mds_L-function, and mds_H-function are conducted. The mds_L-function and the mds_H-function consist of matrix operations. However, the sixth and the seventh rounds differ slightly from the other rounds: The sixth round does not include mds_H-function, and the seventh round consists of only XOR with the extended key.

The optimized architecture of Hierocrypt-3 is portrayed in Figure 4. The mds_L and mds_H-function are the bottleneck of the ρ-function in Hierocrypt-3. Each matrix operation can be transformed to table substitution, and then combined with an S-function. The ρ-function can be deformed eventually to the algorithm including mds_L_ST and mds_H_ST-table substitution.

3.7 SC2000

SC2000[17] is a 128-bit block cipher, of which the structure is a combined Feistel and SPN network. The data randomization procedure in 128-bit includes seven round processes. Each round executes five transformations in sequence: I, B, I, R, and R-function. In the I-function, the input of 128-bit plaintext is combined with the extended key by XOR. In the B-function, 4×32-bit are transformed into 32×4-bit and are replaced by S_4-tables. Then it is inversely transformed into 4×32-bit. The

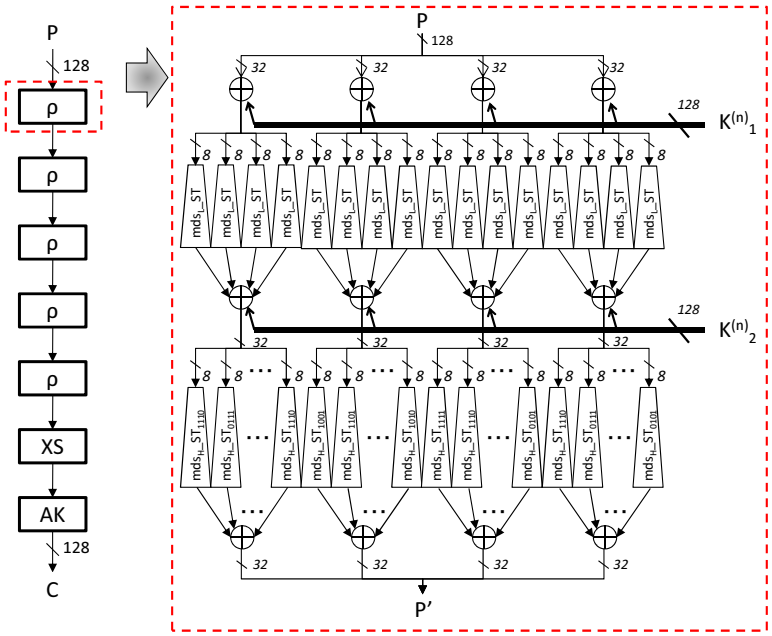


Figure 4: Optimized Hierocrypt-3 (Data randomization procedure).

R-function consists of an S-function, M-function, and L-function. In the S-function, the 32-bit input is separated into several fragments of arbitrary size, for example (6-bit, 5-bit, 5-bit, 5-bit, 5-bit, 6-bit) or (11-bit, 10-bit, 11-bit). Then they are replaced by S_6 and S_5-table or S_{11} and S_{10}-table, etc. In the M-function, the input is multiplied with M-matrix of 32-bit $\times$ 32-bit over GF(2). In the L-function, the input of 2 $\times$ 32-bit is stirred by AND with a constant value and XOR operation. The final round differs slightly from the other rounds: it does not include two R-functions

The optimized architecture of SC2000 is portrayed in Figure 5. In SC2000, B-function, and S-function and M-function can be optimized. In the B-function, 32 accesses to S_4-table can be replaced to logical operation, for whose operation B-function with logical operation can be processed faster than that with S_4-table. In addition, M-function can be transformed to table substitution, and then combined with an S-table in S-function. As a result, according to [17], multiple combinations, (6-bit, 5-bit, 5-bit, 5-bit, 5-bit, 6-bit), (6-bit, 10-bit, 10-bit, 6-bit), (11-bit, 10-bit, 11-bit), or (16-bit, 16-bit), can be selected as the 32-bit input in the combined S-function. The coarser the combination of the 32-bit input in the combined S-function, the less frequent of table access becomes; at the same time the larger the table size becomes. For Nvidia GPUs, the capacity of shared memory varies among their architectures. Therefore, the feature of arbitrarily changeable table size is valid to exhaustively use the capacity of shared memory and to attract GPU performance.

4 Related works

In this section, we sum up several related works of implementation for symmetric block cipher using GPGPU. Regarding our investigation, no related work for CIPHERUNICORN-A, Hierycrypt-3, and SC2000 using GPGPU is described in the literature.

4.1 Intel AES-NI instruction set

AES-NI[7] is a extended instruction set introduced from Intel Xeon 5600 processor. The instructions are designed to implement some of the complex and performance intensive steps of the AES algorithm using hardware and thus accelerating the execution of the AES algorithms. The instruction set is

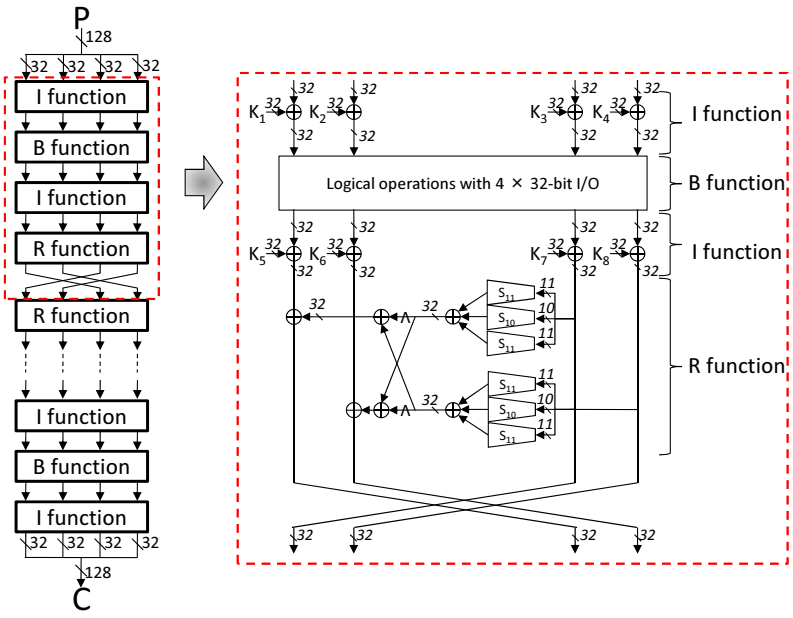


Figure 5: Optimized SC2000 (Data randomization procedure).

comprised of six instructions that perform several compute intensive parts of the AES algorithm. Four of the new instructions are for accelerating the encryption or decryption of a round and two instructions are for round key generation. Except of AES, to date acceleration mechanism of other cipher algorithms are not embedded in a CPU.

4.2 GPGPU implementation of AES

Biagio et al. implemented counter mode AES (AES-CTR) on Nvidia Geforce 8800 GT using CUDA[4]. Authors of this paper achieved 12.5 Gbps with an input size of 128 MB considering about processing granularity. They defined as fine-grained design a solution exposing the internal parallelism of each AES round. They proposed that four 32-bit words blocks were dispatched to four SPs, each to four SPs, with each thread as a fine-grain processing. Moreover, the coarse-grained design was design as exploiting higher level parallelism, which exists between independent plaintext blocks. In this granularity, each thread processes each 128-bit plaintext block.

Iwai et al. implemented AES on CUDA with different granularity and various memory allocation styles[10]. For the AES encryption module, they contrived what bytes should be mapped to each thread, granularity, as one factor to increase its performance. They defined the following four ways of granularity: 16 Bytes/Thread and the other granularities. In 16 Bytes/Thread, a thread is in charge of encryption of a plaintext block. However, in other granularities, multiple threads are in charge of encryption of a plaintext block. According to the experimentally obtained results described in their paper, 16 Bytes/Thread granularity showed a tendency to have higher throughput than the other granularities because, in other granularities, the processing resulting in a round should be stored in the shared memory to the next round for synchronization. By contrast, 16 Bytes/Thread do not require such synchronization. Therefore, it was possible for 16 Bytes/Thread to continue to hold processing results not in the shared memory but the registers. Consequently, 16 Bytes/Thread tended to show higher throughput than the other granularities

4.3 GPGPU implementation of Camellia

Oikawa et al. implemented a plain Camellia encryption module at about 23.0 Gbps without data transfer using OpenCL on a Nvidia GTX 260 GPU, which has 216 SPs[15]. Moreover, they tried to implement a Camellia encryption module with bitslice style, which was introduced by [5] and viewed a N-bit processor as a N-bit SIMD computer. Unfortunately, however, the throughput was deteriorated to about 5.0 Gbps. They speculated that the reason was that too many registers were required by a thread. An SP in CUDA GPU is 32-bit processor. Consequently, if 128-bit cipher Camellia is implemented on CUDA GPU with bitslice style, then the thread consumes 32 parallelism$\times$ 128-bit = 512 Byte. Unfortunately in the GTX 260, more than 32 threads can not be activated because each 512 Byte is assigned to a thread as temporary variables. To keep the SP pipeline filled, more than 192 threads per MP are required, as shown in section 2. Consequently, the experimentally obtained result is probably acceptable.

4.4 Analysis for encryption performance increase on GPU

Liu et al. implemented AES on an Nvidia Geforce 9800 GTX to investigate the performance increase factors on GPUs[11]. They insisted on four points: (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) DMA transfer function is helpful to hide the overhead derived from the communication between CPU and GPU.

5 Implementation

5.1 Outline

Based on knowledge from the previous section, we implemented 128-bit block ciphers AES, Camellia, CIPHERUNICORN-A, Hierocrypt-3, and SC2000 on CUDA. For implementation, AES and Camellia are based on well optimized OpenSSL code[16].All of CIPHERUNICORN-A, Hierocrypt-3, and SC2000 are based on the evaluative documents by CRYPTREC. For their optimization, we consulted their specifications[12][2][18][14][17] and [3].

Additionally, we generated an extended key on CPU; then sent it to GPU along with substitution tables and plaintexts.

5.2 Granularity

We adopted 16 Bytes/Thread as granularity for the reasons presented in section 4.2. For the 16 Bytess/Thread granularity, threads process plaintext blocks independently, as shown in Figure 6. Moreover, we did not adopt bitslice implementation because it was obvious for the performance of 128-bit block cipher to become lower than the plain implementation using table substitution, as shown in section 4.3.

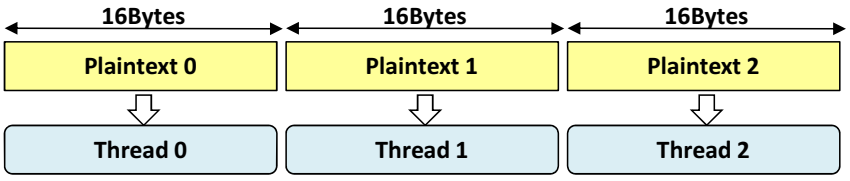


Figure 6: 16 Bytes/Thread granularity.

5.3 Memory allocation

An earlier study[10] revealed that the AES implementation on CUDA can produce higher throughput when the extended key and the substitution tables are stored in the shared memory compared with a case in which these data are stored in constant memory. Therefore, for our implementation, we also stored substitution tables in shared memory. In general CUDA applications, data cannot be sent from the CPU directly to the shared memory. The global memory is so-called storage for data copy from CPU to the shared memory in GPU. For that reason, the access to global memory came about necessarily at the time of the data copy. For faster access to the substitution tables and the extended key, we decided to allocate them in the constant memory first; then to copy them to the shared memory. In 16 Bytes/Thread granularity, all threads load the same extended key and the same substitution tables from the constant memory. Therefore, this access method benefits from a cache effect.

Moreover, as shown in section 4.4, the processing result is expected to be held in registers for the highest performance in AES. Unlike the extended key and substitution table, plaintexts are usually random numbers. No cache effect of holding them in the constant memory exists. For this reason, we first sent plaintexts to global memory. Then we loaded them to registers. The code corresponding to memory allocations of AES on CUDA is shown in Figure 7.

```
/* s_Te0, s_Te1, s_Te2, and s_Te3 are T-Box on shared memory. */
/* c_Te0, c_Te1, c_Te2, and c_Te3 are T-Box on constant memory. */
/* s_key is round key on shared memory. */
/* c_key is round key on constant memory. */
/* s0, s1, s2, and s3 are registers for plaintext. */
s_Te0[tidx]=c_Te0[tidx]; s_Te1[tidx]=c_Te1[tidx];
s_Te2[tidx]=c_Te2[tidx]; s_Te3[tidx]=c_Te3[tidx];
s_key[tidx]=c_key[tidx];

/* start AES encryption */
typedef unsigned int u32;
*(u32*)s0=GETU32(in+   tid*16)^s_key[0];
*(u32*)s1=GETU32(in+ 4+tid*16)^s_key[1];
*(u32*)s2=GETU32(in+ 8+tid*16)^s_key[2];
*(u32*)s3=GETU32(in+12+tid*16)^s_key[3];

t[0]=s_Te0[s0[0]]^s_Te0[s1[0]]^s_Te0[s2[0]]^s_Te3[s3[0]]^s_key[4];
t[1]=s_Te0[s1[1]]^s_Te0[s2[1]]^s_Te1[s3[1]]^s_Te0[s0[1]]^s_key[5];
t[2]=s_Te0[s2[2]]^s_Te0[s3[2]]^s_Te2[s0[2]]^s_Te1[s1[2]]^s_key[6];
t[3] s_Te0[s3[3]]^s_Te0[s0[3]]^s_Te3[s1[3]]^s_Te2[s2[3]]^s_key[7];
```

Figure 7: Code corresponding to memory allocations of AES on CUDA.

5.4 Other optimization techniques

5.4.1 Cut down of thread block switching

Usually in CUDA applications, massively parallel processing data are mapped to respective threads. In implementation on cryptographic modules, we were able to map each plaintext to each thread. However, the processing time of one encryption procedure by a thread was so slight that the overhead of the switching thread blocks tended to be larger and were not negligible. For this reason, after threads are finished encrypting the plaintext, these threads return to the starting point and continue to encrypt other plaintext again. This method requires only a small number of threads to encrypt quite a few plaintexts. Thereby, we were able to avoid the overhead of switching thread blocks

in encryption procedures. Consequently, we set the optimum number of thread blocks between 2–4 times number of MPs. Moreover, as for the number of threads, 32 threads is a processing unit in CUDA. For that reason, we set the number of threads in multiples of 32 to extract better performance. In addition, more than 192 threads are guaranteed for our implementations to keep the SP pipeline filled, as presented in section 2.

5.4.2 Overlapping GPU processing and memory copy

Generally in GPGPU, the data transfer overhead is substantial. However, in implementation of cipher algorithms on GPU, transfer of such data as plaintext and ciphertext between CPU and GPU is unavoidable. Therefore, 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. We implemented the encoding process of each cipher algorithm with overlapping transferring plaintext to (and ciphertext from) the off-chip memory of GPU and the GPU's encoding process. Figure 8 presents a diagram of this overlapping in the case of data division into two plaintext blocks. Figure 9 is the code corresponding to the diagram presented in Figure 8. This overlapping process is known as pipeline processing between data transfer and processing. To optimize this pipeline, a tradeoff exists between block size, which is the dividing size of plaintext as the pipeline stage, and the pipelined overhead. This pipeline optimization will achieve good performance when the data transfer time and processing time are balanced.

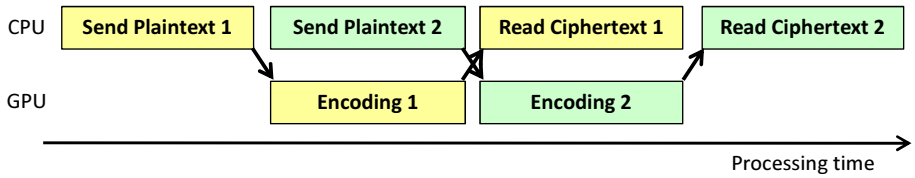


Figure 8: Overlapping data transfer and encryption processing.

```
/* offset is the pointer of plaintext in each stream */
cudaMemcpyAsync(d_plaintext+offset*0, plaintext+offset*0, FILESIZE/2,
        cudaMemcpyDeviceToHost, stream[0]);
cudaMemcpyAsync(d_plaintext+offset*1, plaintext+offset*1, FILESIZE/2,
        cudaMemcpyDeviceToHost, stream[1]);
AES_encrypt<<< grid, block, 0, stream[0]>>>(
        d_plaintext+offset*0, d_ciphertext*offset*0, LOOP/STNUM);
cudaMemcpyAsync(ciphertext+offset*0, d_ciphertext+offset*0,
        FILESIZE/2, cudaMemcpyDeviceToHost, stream[0]);
AES_encrypt<<< grid, block, 0, stream[1]>>>(
        d_plaintext+offset*1, d_ciphertext*offset*1, LOOP/STNUM);
cudaMemcpyAsync(ciphertext+offset*1, d_ciphertext+offset*1,
        FILESIZE/2, cudaMemcpyDeviceToHost, stream[1]);
```

Figure 9: Code corresponding to the diagram presented in Figure 8.

6 Evaluation

6.1 Evaluation environments

Evaluation environments are presented in Table 1. To accurately evaluate the performance increase of the throughputs on CUDA, it is essential to compare them with the throughput on multiple cores in a CPU. Therefore, we adopted Intel Core i7-920 2.66 GHz as a multicore CPU, which has 4 cores. Moreover, we can take advantage of 8 threads attributed to Intel Hyper-Threading Technology[6] embedded in this CPU. Hyper-Threading Technology improves the usability of resources in a CPU as well as hides memory access latency by a thread. Furthermore, we adopted OpenMP API for multicore programming on a CPU in this paper.

A GPU is connected to a PC through the PCI Express bus. At present, Fermi architecture, whose computing power is enhanced from conventional GT200, has been released. To evaluate the performance of these two generations of GPU, we chose Tesla C2050 with Fermi architecture and Geforce GT285 with GT200 architecture.

Table 1: Specification of environments.

	Multicore CPU	GPU 1	GPU 2
CPU	Core i7-2600K 3.40 GHz (4 core and 8 threads available)		
Motherboard	ASUS P8Z68-V		
Memory	32 GB		
OS	CentOS 6.0 (Kernel ver2.6.32-71)		
Compiler	gcc ver4.4.4 (option -O3)		
GPU Accelerator	–	Nvidia Tesla C2050	Nvidia Geforce GTX 285
GPU Memory	–	3 GB	1 GB
CUDA Compiler	–	nvcc ver4.0	

6.2 Evaluation result

6.2.1 Throuthput on a multicore CPU

Evaluation results of AES, Camellia, CIPHERUNICORN-A, Hierocrypt-3, and SC2000 on the multicore CPU are shown in Figure 10. Using 4 threads on 4 cores, the performance increase achieved to nearly multiple of the number of cores. In the case of 8 threads on 4 cores, the throughput increased by about 7 to 34 % compared to the throughput of 4 threads on 4 cores due to the Hyper-Threading technology.

6.2.2 Throughput on GPUs

Evaluation results of AES, Camellia, CIPHERUNICORN-A, Hierocrypt-3, and SC2000 on each GPU are shown respectively in Figures 11 to 15. In each figure, the horizontal and vertical axes respectively show the input size of plaintexts and the throughput. The maximum throughput abstracted from these figures is shown in Figure 16.

Using Tesla C2050, extremely high throughput of SC2000 was achieved 73.4 Gbps without data transfer. In addition, using Geforce GTX 285, 47.0 Gbps throughput of SC2000 was achieved. The reason of the great throughput improvement was that different substitution tables were utilized in accordance with shared memory size in respective GPUs . For the implementation on Tesla C2050 and Geforce GTX 285, we respectively adopted a (11-bit, 10-bit, 11-bit) combination with 20 KB of table size and a (6-bit, 10-bit, 10-bit, 6-bit) combination with 8.5 KB of table size.

As shown in Figure 16, CIPHERUNICORN-A and Hierocrypt-3 can not obtain so much performance gain as other encryption modules. This reason is that the throughput depends on the computational complexity. For example, their throughputs on the CPU were inferior to the other

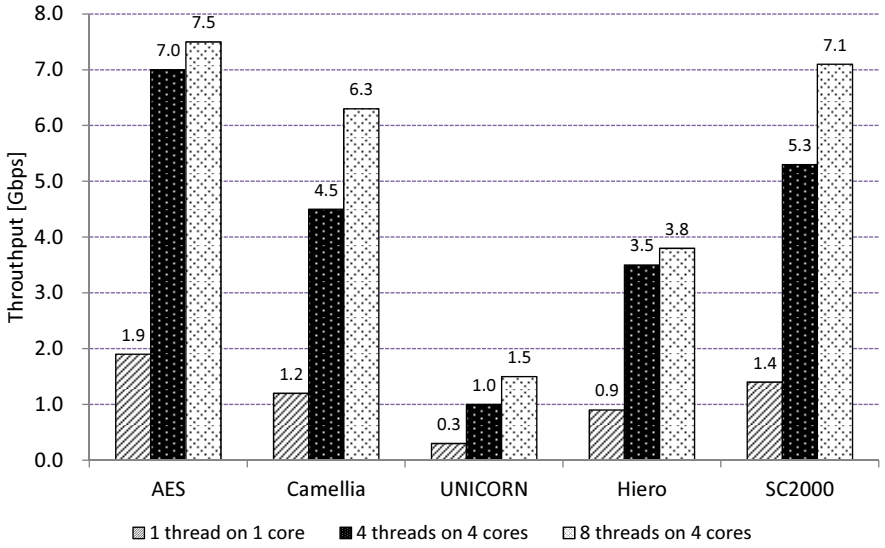


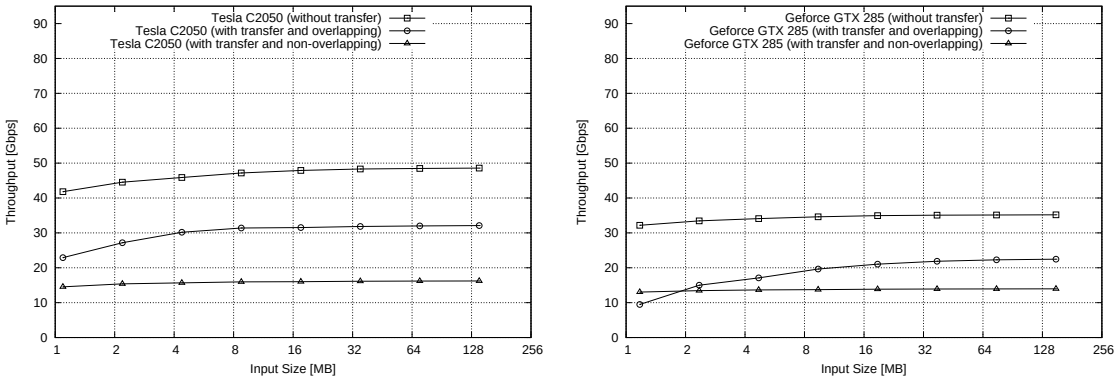
Figure 10: The throughput on Intel Core i7-2600K 3.40 GHz.

modules' on the CPU, as shown in Figure 10. Therefore, the throughput tendency on the two GPUs showed the same result as that on the CPU because our implementations on both CPU and GPU were baed on the evaluative documents, as presented in section 5.1. Herein, to compare the throughput of CIPHERUNICORN-A with that of Hierocrypt-3, the former was one third of the latter in the implementation on the CPU. In contrast, when it comes to on the two GPU, the former became almost the same result as the latter. This reason is that the number of non-memory instruction of CIPHERUNICORN-A is far larger than that of Hierocrypt-3, as is obvious from Figure 3. More specifically integer and logical operations in the randomization procedure of CIPHRUNICORN-A are subject to be hidden under the background of memory access: this encryption module has larger allowance for performance elongation on GPU.

Moreover, as shown in Figure 16 these performance deteriorations including the data transfer show that the data transfer rate holds a dominant position in GPGPU throughput, even if the processing speed of GPU in the environment is extremely fast. For example, although the throughput results without the data transfer of AES, Camellia, and SC2000 show differences in performance, the throughput including data transfer deteriorated to an equal degree. The reasons are the much lower transfer speed of the PCI Express bus than that of inner connection within GPU. Nevertheless, the effective throughput with data transfer produces approximately 4.7 times increase in speed compared with 7.1 Gbps throughput of 8 threads on 4 cores of the Intel Core i7-2600K 3.40 GHz CPU in Figure 10, as in the case of SC2000.

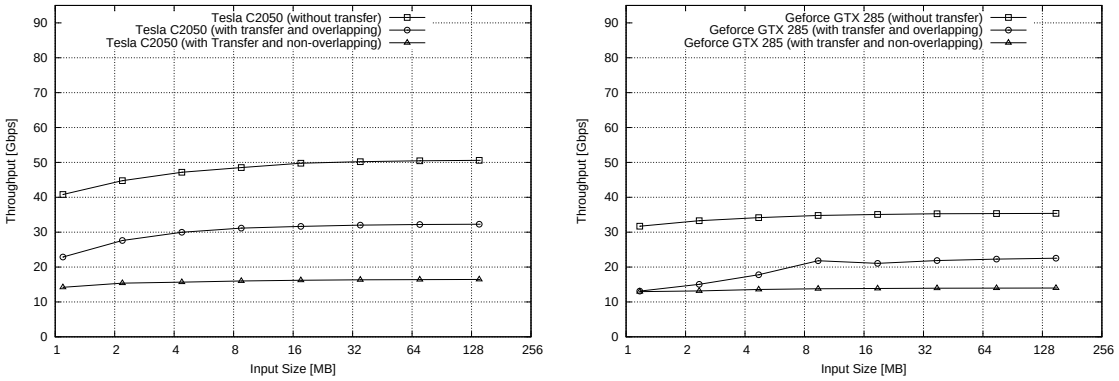
Furthermore, as shown in Figures 11 to 15, irrespective of the cipher algorithms, performance with data transfer and overlapping on Geforce GTX 285 tended to deteriorate for files smaller than 4 MB. In other words, a large file size was required, aimed at a performance increase in Geforce GTX 285. By contrast, the Tesla C2050 achieved nearly maximum throughput, even for files as small as 4 MB. These results indicate that the new generation of Tesla C2050 is more suitable for practical use than Geforce GTX 285 in terms of GPGPU implementation of cryptographic modules.

Incidentally, when the file size was at a point near 2 MB, the performance results obtained with data transfer and overlapping became inverse compared with those obtained with data transfer and non-overlapping; when the file size became greater than 2 MB, the performance was increased by overlapping. The performance deterioration that occurred by overlapping is the overhead of a series of procedure to boot up DMA transfers. Therefore, these results indicate that the use of overlapping is not as simple as in Liu's arguments[11]. In other words, it is necessary to decide whether to use overlapping technique or not, according to the file size.



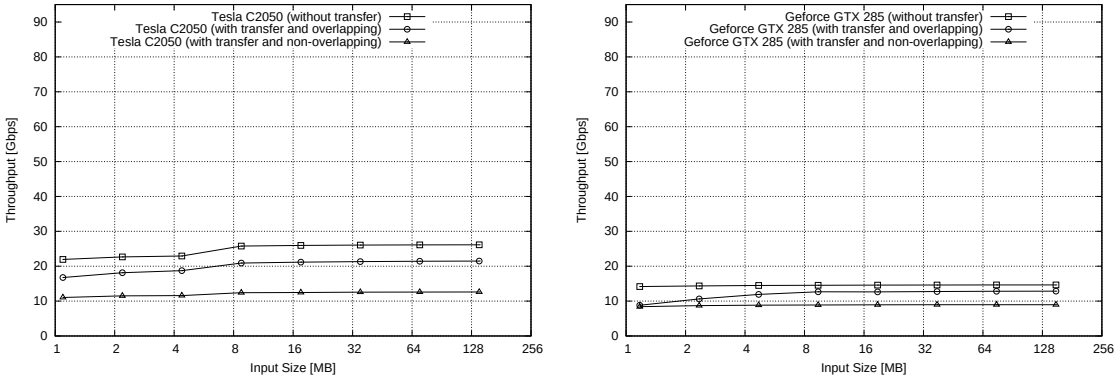
(a) Tesla C2050 (b) Geforce GTX 285

Figure 11: AES throughput on GPUs.



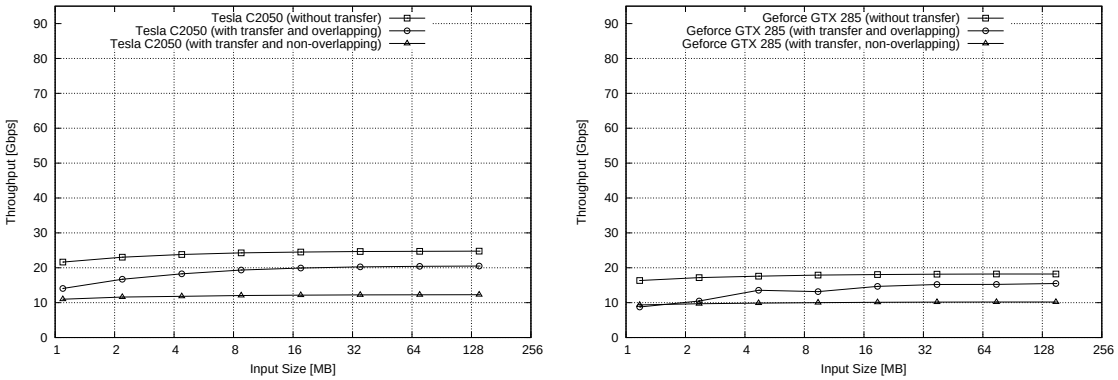
(a) Tesla C2050 (b) Geforce GTX 285

Figure 12: Camellia throughput on GPUs.



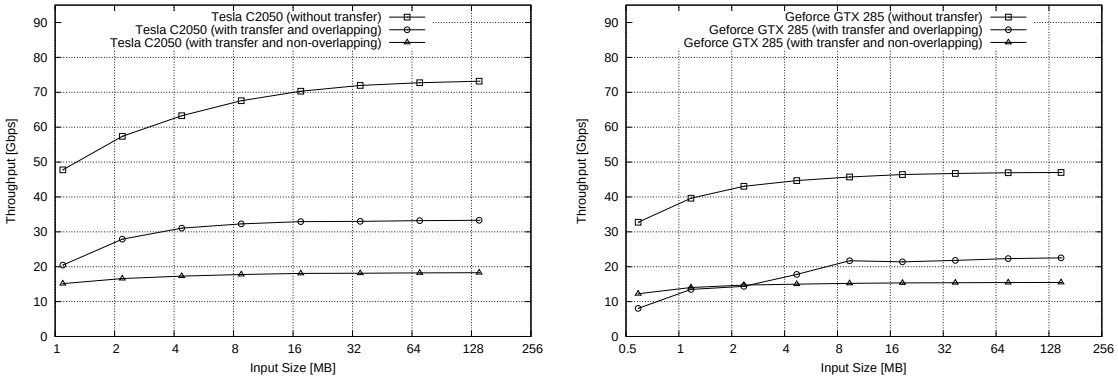
(a) Tesla C2050 (b) Geforce GTX 285

Figure 13: CIPHERUNICORN-A throughput on GPUs.



(a) Tesla C2050 (b) Geforce GTX 285

Figure 14: Hierocrypt-3 throughput on GPUs.



(a) Tesla C2050 (b) Geforce GTX 285

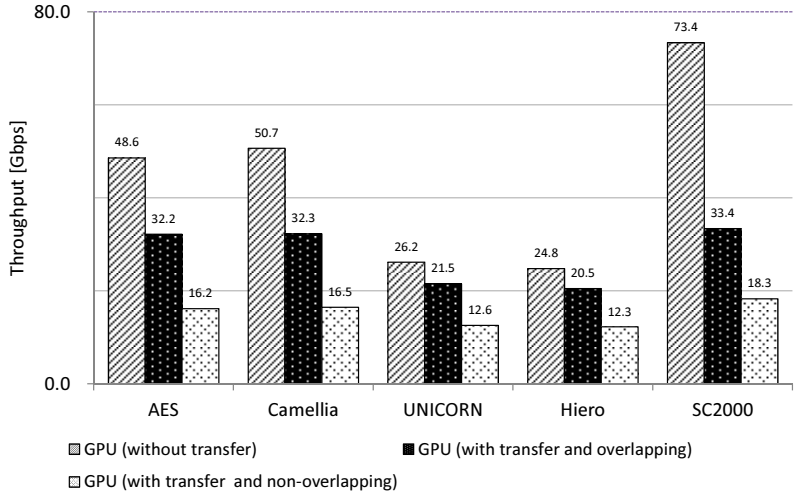
Figure 15: SC2000 throughput on GPUs.

6.2.3 Performance factors

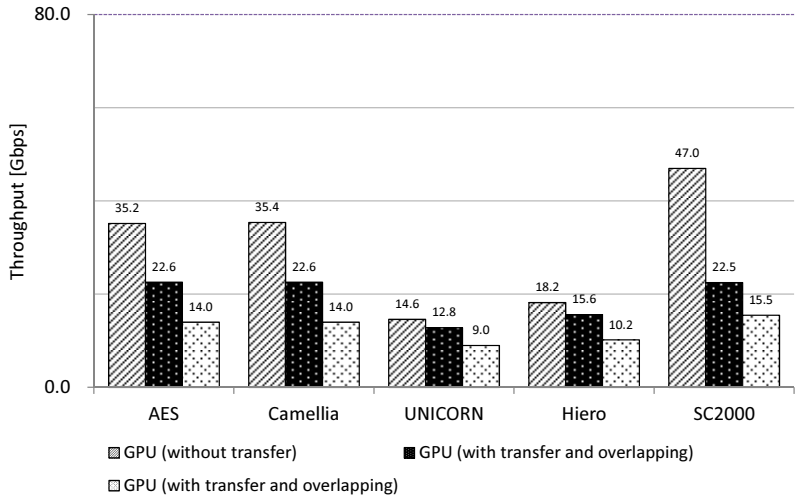
The information such as the number of threads and thread blocks and occupancy, obtained from CUDA profiler[13], are presented in Table 2. For our implementation of all ciphers on CUDA, the optimum number of threads and thread blocks were found in a manual optimization style. Generally, occupancy is essential for optimizing the performance in CUDA programming and then the programs are designed with a goal of making it higher. But in our implementation of block ciphers on CUDA, the occupancies in Table 2 did not necessarily show high values at the maximum throughputs; occupancy levels were not tied to the thoughts directly. In short, implementation techniques presented in section 5, for example granularity and memory allocation, are expected to be more essential than occupancy in order to extract higher throughput of block ciphers on CUDA.

6.2.4 Comparison with the throughput using Intel AES-NI instruction set

For the specification of the evaluation environment presented in Table 1, Intel AES-NI instruction set is available. Here, as the reference to evaluate the AES throughput on CUDA, its comparison with AES-NI is discussed. For the implementation of AES-NI instruction set, we took advantage of OpenSSL code. Comparison between the throughputs of AES-NI and Nvidia Tesla C2050 at 140 MB fixed size is presented in Figure 17. The throughput of AES-NI using 4 threads on 4 cores was surprisingly high 44.2 Gbps, which was approximately 1.4 times higher than that on Tesla



(a) Tesla C2050



(b) Geforce GTX 285

Figure 16: Comparison of the best throughputs of five block ciphers.

Table 2: Summary of performance factors information

(a) GPU 1 (Tesla C2050)

	AES	Camellia	UNICORN	Hiero	SC2000
Thread blocks	28	28	56	28	28
Threads per thread block	512	512	256	512	512
Registers per thread	63	63	30	49	63
Shared memory per thread block	4.25 KB	4.27 KB	1.28 KB	15.28 KB	20.25 KB
Occupancy	0.33	0.33	0.67	0.33	0.33

(b) GPU 2 (Geforce GTX 285)

	AES	Camellia	UNICORN	Hiero	SC2000
Thread blocks	60	60	60	60	60
Threads per thread block	256	256	256	256	256
Registers per thread	15	15	16	30	15
Shared memory per thread block	4.30 KB	4.30 KB	1.33 KB	15.36 KB	8.79 KB
Occupancy	0.75	0.75	1.00	0.25	0.25

C2050 with data transfer and overlapping. Therefore, it is concluded that AES-NI is an extremely valuable option of AES encoding if machine fulfills prescribed requirement to use its instruction set. Nevertheless, encryption processing using GPGPU is an excellent option because it is adaptive to multiple block cipher algorithms and requires no special hardwares.

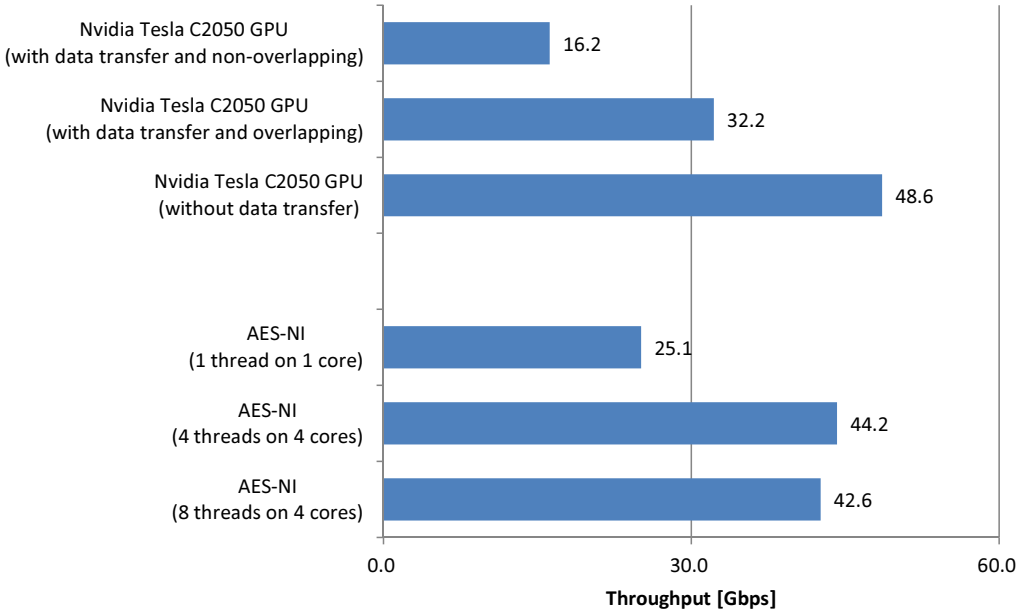


Figure 17: Comparison between throughputs of AES-NI and Nvidia Tesla C2050 GPU.

7 Conclusion

We implemented AES, Camellia, CIPHERUNICORN-A, Hierocrypt-3, and SC2000 from several symmetric block ciphers in an e–government recommended ciphers list by CRYPTREC in Japan. We evaluated these respective performances on an Nvidia Tesla C2050 and Nvidia Geforce GTX 285. The throughput of SC2000 on Tesla C2050 was 73.4 Gbps without data transfer. However, the throughput of SC2000 with data transfer and overlapping was 33.4 Gbps. Considering throughput with data transfer as the effective throughput, the difference between the types of cipher algorithms were absorbed. Nevertheless, the implementation on CUDA achieved several times higher performance than that obtained on 8 threads on 4 cores of Intel Core i7-2600K 3.40 GHz. For implementation of the symmetric block cipher on GPU, data encryption on GPU might suggest some alternative to multicore CPUs. Furthermore, the evaluation results presented in this paper clarified that the upper limit of encryption speed on GPU becomes the transfer rate of the PCI Express bus. For this reason, the diffusion of the next generation of PCI Express bus is anticipated.

Moreover, the throughput of AES-NI was approximately 1.4 times higher than that on Tesla C2050 with data transfer and overlapping. Nevertheless, encryption processing using GPGPU is an excellent option because it is adaptive to multiple block cipher algorithms and require no special hardwares.

Finally, many cipher algorithms exist in the real world in addition to the algorithms implemented in this paper. Future research might include more detailed investigation of what GPU function contribute most to the surprising performance increase, thereby supporting the constitution of a performance predication model for use with symmetric block ciphers.

References

[1] Cryptography Research and Evaluation Committees. http://www.cryptrec.go.jp/english/index.html.

[2] Aoki, Kazumaro and Ichikawa, Tetsuya and Kanda, Masayuki and Matsui, Mitsuru and Moriai, Shiho and Nakajima, Junko and Tokita, Toshio. Specification of Camellia —a 128-bit Block Cipher Version 2.0. *Order A Journal On The Theory Of Ordered Sets And Its Applications*, pages 1–35, 2001.

[3] Kosuke Asahi, Toyoshi Masuda, Eikou Nishikawa, Yuto Yokoyama, Yasutaka Igarashi, and Toshinobu Kaneko. A Fast implementation on the 64 bit environment of the suggested cipher in CRYPTREC (in Japanese). In *Proceedings of The 29th Symposium on Cryptography and Information Security*, Fukuoka, Japan, September 2010.

[4] Andrea Di Biagio, Alessandro Barenghi, Giovanni Agosta, and Gerardo Pelosi. Design of a parallel AES for graphics hardware using the CUDA framework. *Parallel and Distributed Processing Symposium, International*, pages 1–8, 2009.

[5] Eli Biham. A Fast New DES Implementation in Software. pages 260–272. Springer-Verlag, 1997.

[6] Intel Corporation. Intel Hyper-Threading Technology. http://software.intel.com/en-us/articles/performance-insights-to-intel-hyper-threading-technology/.

[7] Intel Corporation. Securing the Enterprise with Intel AES-NI. http://www.intel.com/content/dam/doc/white-paper/enterprise-security-aes-ni-white-paper.pdf, 2010.

[8] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag, Berlin, Heidelberg, New York, 2002.

[9] The IEEE Security in Storage Working Group. XTS block cipher-based mode (XEX-based tweaked-codebook mode with ciphertext stealing). http://siswg.net/.

[10] Keisuke Iwai, Takakazu Kurokawa, and Naoki Nisikawa. Aes encryption implementation on cuda gpu and its analysis. *2010 First International Conference on Networking and Computing*, pages 209–214, 2010.

[11] G. Liu, H. An, W. Han, G. Xu, P. Yao, M. Xu, X. Hao, and Y. Wang. A Program Behavior Study of Block Cryptography Algorithms on GPGPU. In *Frontier of Computer Science and Technology, 2009. FCSTf09. Fourth International Conference on*, pages 33–39. IEEE, 2009.

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

[13] NVIDIA Corporation. *NVIDIA CUDA Occupancy Calcurator 3.0*, 2012.

[14] Kenji Ohkuma, Hirofumi Muratani, Fumihiko Sano, and Shin-ichi Kawamura. The Block Cipher Hierocrypt. In *Proceedings of the 7th Annual International Workshop on Selected Areas in Cryptography*, SAC '00, pages 72–88, London, UK, UK, 2001. Springer-Verlag.

[15] Kazuki Oikawa, Jiajong Wang, Eiichiro Kodama, and Toyoda Takada. Implementation and Evaluation of Cryptography Algorighm on GPGPU (in Japanese). In *Proceedings of The 29th Symposium on Cryptography and Information Security*, Takamatsu, Japan, January 2010.

[16] The OpenSSL Project. OpenSSL: The Open Source toolkit for SSL/TLS. http://www.openssl.org, 2007.

[17] Takeshi Shimoyama, Hitoshi Yanami, Kazuhiro Yokoyama, Masahiko Takenaka, Kouichi Itoh, Jun Yajima, Naoya Torii, and Hidema Tanaka. The Block Cipher SC2000. In *Revised Papers from the 8th International Workshop on Fast Software Encryption*, FSE '01, pages 312–327, London, UK, UK, 2002. Springer-Verlag.

[18] Y Tsunoo. 128bit block cipher - CIPHERUNICORN-A. *Nec Research Development*, 43(3):183–187, 2002.

[19] Vasily Volkov and James W. Demmel. Benchmarking GPUs to tune dense linear algebra. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11, Piscataway, NJ, USA, 2008. IEEE Press.