An Efficient Block Cipher Implementation on Many-Core Graphics Processing Units

Sangpil Lee*, Deokho Kim*, Jaeyoung Yi** and Won Woo Ro*

Abstract—This paper presents a study on a high-performance design for a block cipher algorithm implemented on modern many-core graphics processing units (GPUs). The recent emergence of VLSI technology makes it feasible to fabricate multiple processing cores on a single chip and enables general-purpose computation on a GPU (GPGPU). The GPU strategy offers significant performance improvements for all-purpose computation and can be used to support a broad variety of applications, including cryptography. We have proposed an efficient implementation of the encryption/decryption operations of a block cipher algorithm, SEED, on off-the-shelf NVIDIA many-core graphics processors. In a thorough experiment, we achieved high performance that is capable of supporting a high network speed of up to 9.5 Gbps on an NVIDIA GTX285 system (which has 240 processing cores). Our implementation provides up to 4.75 times higher performance in terms of encoding and decoding throughput as compared to the Intel 8-core system.

Keywords—General-Purpose Computation on a Graphics Processing Unit, SEED Block Cipher, Parallelism, Multi-Core Processors

1. INTRODUCTION

Today's computer networks and Internet computing requires more secure data communications than ever before. Security and cryptography must guarantee that personal information does not leak out to the public or unauthorized users. To provide sufficiently secure data communication, various data encryption methods have been proposed, such as the Data Encryption Standard (DES), Advanced Encryption Standard (AES), SEED [1], and the Fast Data Encipherment Algorithm (FEAL). Wireless networks for ubiquitous computing also have security issues; hence various studies have been conducted for improving their reliability [2, 3].

However, because of high network transmission rates, the process of encryption and decryption causes major bottlenecks in present network systems [4]. In particular, when the network speed reaches 10-100 Gbps, the encryption and decryption speed becomes a severe performance bottleneck. To address this problem, FPGA-based platforms have been proposed in the previous literature [5-9]. Since many cryptographic algorithms consist of a large amount of homogenous

※ This work was supported by the Basic Science Research Program through the National Research Foundation of Korea, which is funded by the Ministry of Education, Science and Technology [2009-0070364].
Manuscript received September 2, 2011; first revision January 4, 2012; accepted January 31, 2012.
Corresponding Author: Won Woo Ro
* School of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea ({madfish; nautes87; wro} @yonsei.ac.kr)
** System IC Center, LG Eletronics, Korea (jaeyoung.yi@lge.com)

computations, using FPGAs we can achieve enhanced computational performance and high data throughput. However, because of its design complexity, an FPGA runs on low frequency and has limited flexibility as compared to that of the software approach.

Recently, the Graphics Processing Unit (GPU), which was originally developed as a co-processor to accelerate 2D/3D graphic applications, has been researched as programmable devices for non-graphical general computations. This new approach and paradigm is called general-purpose computation on a GPU (GPGPU) [10-12]. The GPGPU concept is believed to provide significant computing power as a future parallel processing platform. In fact, GPU can improve the performance of many general applications by fully utilizing its large number of processing cores [10]. These applications include scientific computations such as fluid simulation, Monte Carlo simulation, molecular dynamics, and multimedia encoding/decoding. The goal of this paper is to provide an efficient parallel implementation and performance evaluation of a cryptography application (a block cipher algorithm) on GPU architecture.

We believe that greater performance enhancement of block cipher algorithms can be achieved with the recently developed GPUs. As GPUs become standard consumer items, they will become an attractive option for use in cryptosystem implementation. Our contribution in this paper is the implementation of the SEED block cipher algorithm, which is the Korean standard block cipher algorithm, on commercial NVIDIA GPU architectures and performance comparisons with the legacy implementations on general purpose processors and an FPGA chip. To the best of our knowledge, this is the first study to develop a parallel implementation of the SEED block cipher algorithm on NVIDIA GPUs. Using this new hardware platform, we can achieve a high level of parallelism and increased computational performance.

We claim that the GPU can be used as a flexible cryptographic coprocessor because of its immense processing power. In particular, it can be deployed as an alternative and solution to multi-core processors, by offering a better price/performance advantage. Our work shows how GPUs can significantly improve the throughput of encryption and decryption protocols. The designs are fully functional and we have achieved a throughput of up to 9.5 Gbps. Our implementation provides up to 4.75 times higher performance in terms of throughput as compared to the baseline Intel 8-core system. The performance is improved by 48% as compared to that of the high-performance FPGA hardware accelerator.

The rest of this paper is organized as follows: In Section 2, we describe the structure of the 128-bit SEED block cipher algorithm and overview of the GPU architecture. In Section 3, we present a detailed description of our design for using SEED on the GPU. Experimental results and an analysis are shown in Section 4. Finally, prior related work is introduced in Section 5 and we conclude the paper and discuss future works in Section 6.

2. BACKGROUND

In this section, we present the background needed to understand the basic algorithm of the SEED block cipher application and we provide a detailed description of the GPU architecture.

2.1 Structure of the SEED Algorithm

A block cipher algorithm is a cryptography application and it is based on a symmetric key cipher operating on a block (a fixed-length group of bits). The SEED block cipher algorithm was

developed by Korea Information Security Agency (KISA) in 1998, because of the government's concern regarding the importance of cipher systems [1]. It has become a national standard since 2000, and has been adapted by most of the security systems in Korea [1]. In particular, the The SEED algorithm was designed as an important protection tool against differential cryptanalysis and linear cryptanalysis.

The SEED block cipher algorithm processes a block of 128-bit plaintext using a 128-bit cipher key, and outputs a 128-bit ciphertext. The SEED algorithm is a private key algorithm (it uses the same key for encryption and decryption) and has a Feistel structure with 16 rounds for high security. A Feistel structure receives two t-bit blocks, L0 and R0, and repeats r rounds ($r \geq 1$) of encryption. A Feistel structure is generally used in more than three rounds, in an even number of rounds.

A general overview of the SEED algorithm is depicted in Fig. 1. The initial input data stream is 128-bit wide. Internally, the 128-bit input text stream is divided into two 64-bit blocks (called L0 and R0 in the diagram). The entire SEED procedure is composed of 16 rounds all together. After the data goes through 16 rounds, the result will be two 64-bit blocks forming the 128-bit cipher-text [1].

The specific details of each round are shown in Fig. 2. In Round 1, there are two inputs, L0 and R0. L0 includes the left 64 bits of the original input stream and R0 contains the remaining right 64 bits. The output of the first round consists of L1 and R1. The first output, R1, is the result of L0 exclusive or F(R0); the F operation will be explained shortly. On the other side, R0 just flows to L1. In other words, Round 1 bypasses R0 to L1, which becomes one of the inputs to Round 2. The SEED design consists of the round-key generator, F-function, G-function, and S-boxes [1].

The F-function of SEED has a Feistel structure, as in Fig. 3. It provides resistance against differential cryptanalysis, linear analysis, and other known attacks. The F-function receives a 64-bit block, Rn, and a 64-bit round key as inputs. The 64-bit block input is divided into two 32-bit

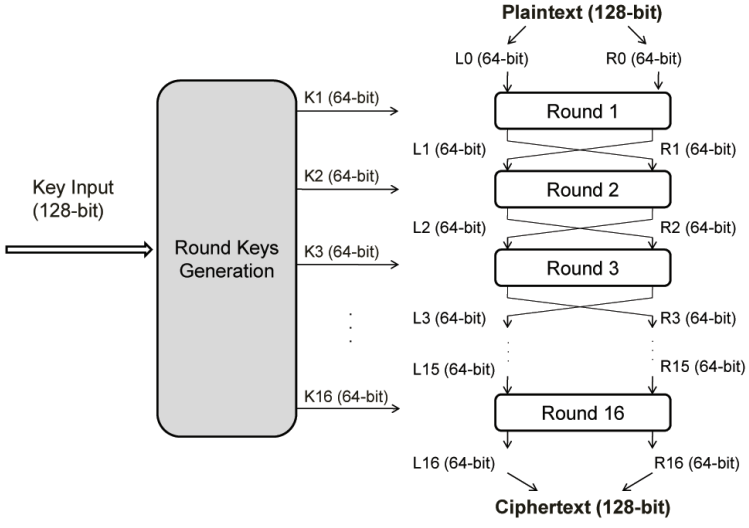


Fig. 1. SEED algorithm structure

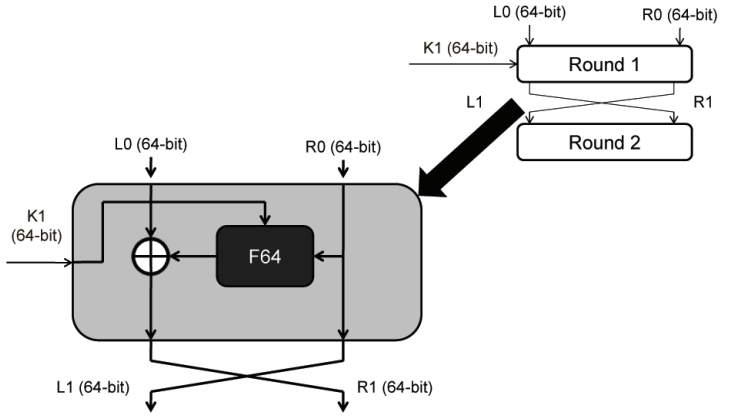


Fig. 2. Each round of SEED

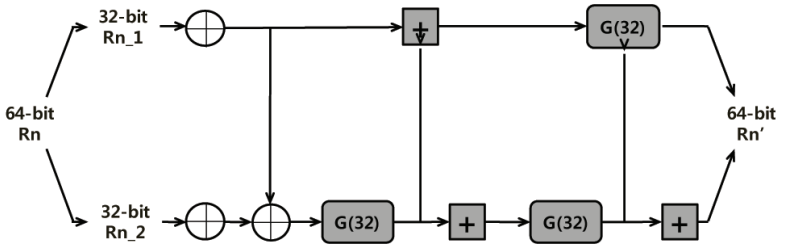


Fig. 3. Block diagram of F-Function

blocks and processed through a mixture of exclusive-ORs, additions (mod 32), and G-functions. The two 32-bit blocks that are generated are concatenated to form the output of the F-function, which is a 64-bit block, Rn'.

The G-function is the main function used in the F-function and also in the round key generation. The formulas that characterize the G-function are laid out in Table 1. A 32-bit input is divided into four 8-bit blocks (x0 ~ x3), which are passed through S-boxes S1 and S2.

The outputs of the S-boxes are four 8-bit blocks called y0, y1, y2, and y3. These blocks are distributed over m0, m1, m2, and m3, and then a bitwise AND operation is performed. Finally, the four 8-bit outputs of the four AND operations are exclusive-ORed to produce z0, z1, z2, and

Table 1. Formulas for G-function

$y_0 = S_1(x_0),\ y_1 = S_2(x_1),\ y_2 = S_1(x_2),\ y_3 = S_2(x_3)$
$z_0 = (y_0 \,\&\, m_0) \oplus (y_1 \,\&\, m_1) \oplus (y_2 \,\&\, m_2) \oplus (y_3 \,\&\, m_3)$
$z_1 = (y_0 \,\&\, m_1) \oplus (y_1 \,\&\, m_2) \oplus (y_2 \,\&\, m_3) \oplus (y_3 \,\&\, m_0)$
$z_2 = (y_0 \,\&\, m_2) \oplus (y_1 \,\&\, m_3) \oplus (y_2 \,\&\, m_0) \oplus (y_3 \,\&\, m_1)$
$z_3 = (y_0 \,\&\, m_3) \oplus (y_1 \,\&\, m_0) \oplus (y_2 \,\&\, m_1) \oplus (y_3 \,\&\, m_2)$
$(m_0 = 0xfc, m_1 = 0xf3, m_2 = 0xcf, m_3 = 0x3f)$

($\oplus$ denotes the exclusive-OR, & is the bitwise AND)

Table 2. Pseudo-code for round key generation

1.Divide the 128-bit cipher key into four 32-bit blocks: A, B, C, and D
2.Generate the first round key by

$$K_{1,0} = G(A + C - KC_0),\ K_{1,1} = G(B - D + KC_0)$$

(KC_0: Round 1 coefficient)
3.A || B = (A || B) >> 8
4.Generate the second round key by

$$K_{2,0} = G(A + C - KC_1),\ K_{2,1} = G(B - D + KC_1)$$

(KC_0: Round 2 coefficient)
(Repeat the process to generate the other keys)

z3. The four 8-bit blocks that result are combined to make the 32-bit output of the G-function. Table 1 shows the internal operation of the G-function.

The key schedule generates each round subkey. It uses the G-function, addition, subtraction, and (left/right) circular rotation. A 128-bit input key is divided into four 32-bit blocks (A, B, C, D) and the two 32-bit subkeys of each round are generated as following the pseudo-code in Table 2. Each constant KCi is generated from a part of the golden ratio number.

2.2 GPU Architecture

Modern GPUs have evolved from fixed graphics pipeline hardware, which accelerates specific graphics operations, to programmable unified shader architecture that provides more flexibility. In general, the programmable graphics hardware is composed of an array of computing units. In fact, the hardware is designed to execute logical, arithmetic, and floating-point operations that are similar to those of general-purpose processors. The newly introduced GPGPU concept utilizes the computing units of a graphics device for compute-intensive applications [13].

NVIDIA's Tesla architecture is one of the GPU architectures that were developed on the basis of the GPGPU concept. NVIDIA also provides a Compute Unified Device Architecture (CUDA) programming model [13] in order to enable the programmability of their graphics hardware. (Hereafter, we refer to the graphics hardware as device in accordance with the NVIDIA terminology.) There are several types of graphics devices that are based on the Tesla architecture and in this study we have used the GeForce GTX 285 GPU.

The Tesla architecture consists of a group of several streaming processor arrays that provide a certain level of scalability. The internal architecture of GeForce GTX 285 is shown in Fig. 4. This GPU is composed of a single streaming processor array that contains 10 Texture Processor Clusters (TPCs), each of which includes 3 Streaming Multiprocessors (SMs). Each SM has 8 Scalar Processors (SP), which serve as individual execution units, 2 Super Function Units (SFU), a double precision floating point processing unit, and shared resources. Shared resources include a 16 KB shared memory, 16K 32-bit registers, and other cache memories. The SP processes general arithmetic, logic, and single-precision floating-point operations. The SFU processes transcendental functions. In GeForce GTX 285, there are 240 SPs grouped in 30 SMs.

In the CUDA programming model, GPU threads are grouped into thread blocks that allow data sharing and synchronization among the threads. Each thread block is assigned to an SM. However, there is no data sharing between different thread blocks. For efficient thread processing, 32 GPU instructions in a thread block are grouped into a warp, which is a basic scheduling

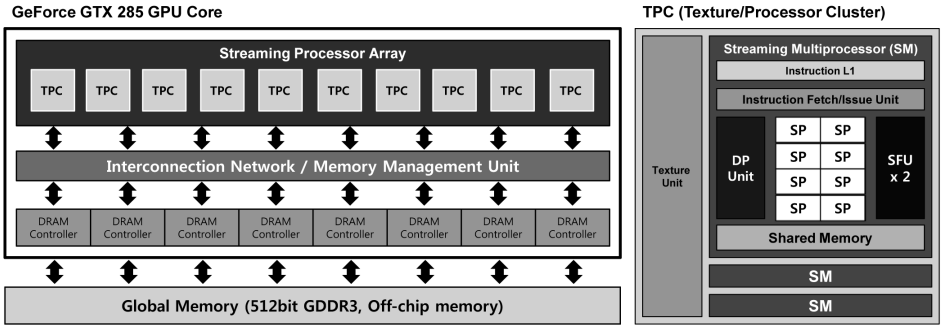


Fig. 4. GeForce GTX 285 (GT200) GPU hardware model

unit of the NVIDIA GPU. A warp is a group of instructions that are executed simultaneously on 32 GPU threads, and instructions in the same warp are executed on SPs during 4 clock cycles. NVIDIA terms this execution model the Single Instruction Multiple Thread (SIMT).

The SIMT unit of an SM selects a warp that is ready for execution and issues the next instructions to the active threads of the warp in an out-of-order fashion. If a warp is stalled for some reason (e.g., processing memory transfer instructions), the SIMT unit immediately switches to another warp that is ready for execution. As a result, hundreds of threads are required to maintain maximum hardware utilization and to compensate for memory access latency.

3. GPU-Accelerated Parallel SEED Block Cipher Algorithm

This section presents our implementation of the GPU-based parallelized SEED block cipher algorithm.

3.1 Efficient CUDA-based Implementation on GPU

It has been stated that the SEED algorithm is based on a Feistel structure. Indeed, the Feistel structure is a major obstacle in parallel implementation owing to the data dependency across each round. In FPGA implementation [7], this structure can be processed efficiently by applying a pipelined model. In contrast, our implementation exploits data parallelism by using the sufficient hardware processing elements provided on the GPU. In this section, we propose a GPU-based parallelized SEED block cipher algorithm that efficiently utilizes the SIMT feature and data decomposition.

The SEED block cipher performs the encryption and decryption operation for 128-bit unit data blocks. This procedure requires 16 sequential round operations. The parallelization on round operations is not possible due to these data dependencies among round operations. However, there are no dependencies among unit data blocks during the encryption/decryption procedure. The plaintext is a set of unit data blocks. Therefore, the plaintext can be encrypted rapidly through the parallel processing of each unit data block.

In our implementation, each unit data block is processed by a GPU thread. The parallel processing mechanism for unit data blocks is shown in Fig. 5. Each GPU thread has the complete SEED encryption/decryption routine including 16 round operations to perform encryp-

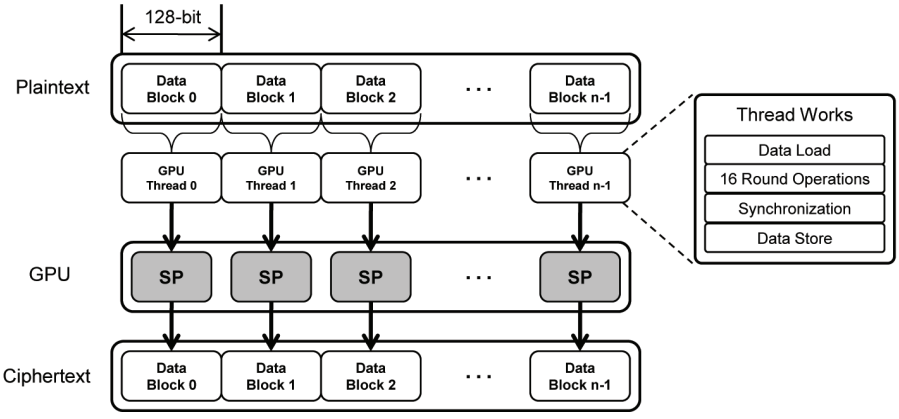


Fig. 5. GPU-based parallelized SEED block cipher algorithm

tion/decryption steps independently. These GPU threads are processed by SPs in the GPU as we mentioned in Section 3.1. Therefore, hundreds of unit data blocks in the plaintext are encrypted or decrypted concurrently. In fact, the block cipher operation is a sort of SIMT method, since each GPU thread performs the same operation but on different unit data blocks. The GPU is a typical SIMT type of hardware, which has hundreds of processing cores. Thus, it is appropriate to process data parallel applications.

During the encryption procedure, the host prepares the round keys for encryption and transfers them, along with the plaintext data, to the global memory. The host system then launches the kernel on the GPU side. The GPU divides the kernel code into multiple threads, processes the threads using its SPs, and then stores the results in the global memory. Finally, the CPU copies the results from the global memory to the host memory.

In fact, the host code is responsible for preparing environmental variables to use the GPU, including CUDA device detection, initialization, host system main memory allocation, device global memory allocation, data transfer, kernel configuration, and launching. The host CPU does not perform the actual computation for encryption/decryption, but is used only for the round key generation.

The actual encryption/decryption operation is performed by the kernel code executed on the GPU. A kernel generates multiple threads on the GPU and all of these threads perform the same instructions but on different data. Each thread uses its own ID, which consists of a block index and a thread index, in order to distinguish its own working area. Since each thread ID points to a 128-bit data block header, each thread can process data independently. As the round keys are generated only once before starting encryption and a large amount of the computation overhead exists in the encryption process, it is efficient to have the CPU generate round keys and the GPU process the encryption routine.

3.2 Optimizations for the GPU-based SEED

In order to maximize the performance, we apply several optimization strategies to our baseline algorithm. As stated in the above explanation of the GPU memory model, memory access time is one of the major performance bottlenecks in a GPGPU application. Therefore, we opti-

mize the memory access pattern, and efficiently utilize the on-chip memories on the GPU side.

The data to be processed has to be loaded from the global memory to the local registers. Performance losses occur if each thread accesses the global memory, which is the off-chip memory, independently. We optimize this memory access latency by moving the data from the global memory to the shared memory, which is the on-chip memory and provides better bandwidth, once. Our memory arrangement can reduce the latency as the GPU memory controller provides the coalesced data access mode.

There are a number of special conditions to enable coalesced access from the global memory to the shared memory. In a GT200, coalesced access is available when 16 threads in warp access a consecutive region of the global memory at the same time. The SEED algorithm uses four 32-bit integer type variables in one encryption sequence. Hence, four 32-bit data variables are loaded into one 128-bit data block for efficient memory loading. This is done by using the __align(16)__ qualifier. This qualifier converts four individual memory transfer instructions on integers into a single 128-bit memory transfer instruction. In our design, this modification significantly improves the memory transfer efficiency.

Another optimization is found in implementing the SBOX and round key data. In the SEED algorithm, since the SBOX, which is implemented as a form of table look-up with a size of 4 KB, and round key data are accessed frequently, these data should be located in a high-speed accessible memory region. The constant cache or shared memory could be used for fast data access.

Theoretically, the shared memory could be a more suitable candidate than the constant cache since it can manage 16 individual memory access requests at one time. However, there is a space restriction on the shared memory, as the total size of the shared memory at each SM is limited to 16 KB. This is especially important to note since each thread uses the shared memory as a buffer to load plaintext data in our implementation. Therefore, using the shared memory for the SBOX and the round key will restrict the number of threads allocated in each SP and will cause performance degradation. For this reason, the constant cache is used for storage for the SBOX and the round key in our implementation.

4. PERFORMANCE EVALUATION

To demonstrate our implementation on the GPU system, we performed thorough experiments with various graphic devices. This section presents the performance results and an analysis of the results.

4.1 Platform Environment

In our implementation, we used six different NVIDIA GPUs, including two of the G80 series (8600GT and 8800GTS), two of the G92 series (9800GT and 9800GTX+), and two of the GT200 series (GTX260 and GTX285). These GPUs have some differences; for example, register file size, memory bus width, and double-precision floating-point operation support. However, these GPUs are based on the same NVIDIA GPU architecture and we can apply a uniform implementation technique without any source code modifications.

Our newly proposed GPU design for SEED was tested thoroughly with diverse test vectors, and we confirmed that it operates successfully on the above six different GPU devices. The CUDA timer function was used to measure the GPU run time in the evaluation of the GPU-

Table 3. GPU specification

Model	GPU Type	Number of Cores	Processor Clock (GHz)
8600GT	G80	32	1.18
8800GTS	G80	96	1.18
9800GT	G92	112	1.5
9800GTX+	G92	128	1.84
GTX260	GT200	216	1.24
GTX285	GT200	240	1.47

based SEED block cipher algorithm. The elapsed time during encryption on the GPU was used as a criterion. The specific details for each GPU are shown in Table 3.

To compare the performance of our implementation on various GPU systems, we tested the SEED algorithm on several different general-purpose processors, including the Intel and AMD processors, Cell BE processor, and MIPS64 architecture based OCTEON processor. Especially, the OCTEON processor platform is a real network device that is used for security and firewall services. We added this platform to confirm the performance of the software implementation of the SEED in the real network device. Additionally, in order to compare our performance to the previously proposed implementation on an FPGA chip [9], we measured the execution time on an advanced Xilinx FPGA chip.

First, to see the performance on the general purpose CPUs, the software implementation was tested on the Intel and AMD multi-core platforms. We used the official distribution of the SEED software implementation by KISA [1]. This was compiled with the level 3 optimization of the GCC compiler tool chain. The machine configurations are given in Table 4, along with the performance results. The throughput obtained from the 8 core Intel Xeon system is used as the baseline performance, which is 2.0 Gbps.

The FPGA implementation was synthesized by targeting the Xilinx Virtex-V XC5LX110T chip. The design is fully verified by executing the code on the chip with the maximum possible clock frequency of 50 MHz [9]. The results were confirmed with a Xilinx ChipScope Pro 10.1.

Additionally, more implementations of the SEED algorithm were tested on the Cell BE and the OCTEON CN5230. In the Cell BE implementation, the encryption components ran in parallel on the six Synergistic Processing Elements (SPEs) available in the programming level. In the design process, the Cell BE's heterogeneous structure was taken into consideration, by assigning the Power Processor Element (PPE) and SPEs different types of jobs. The PPE created the round keys and managed the synchronization signals for the SPEs. It transported data and signals to the SPEs through Direct Memory Access (DMA) commands and mailbox commands. Meanwhile, the SPEs took the job of encrypting the data and executing the 16 rounds of the algorithm containing computations of F-functions and G-functions.

The data was transferred from the main memory to the SPE's local memory, with the synchronization controlled by the PPE. When implementing the SEED algorithm on Cell BE, memory alignment was also taken into consideration. In order to take advantage of the Single Instruction Multiple Data (SIMD) structure in the SPEs, the data was aligned during the process of transferring it from the main memory to the SPE's local memory. When moving the data, instead of receiving the 128-bit input data on one 128-bit register, the data was divided into four parts, which went into four different registers. Three other 128-bit input data blocks were decomposed

Table 4. Experimental results on CPU platforms

CPU processor	Processor Clock GHz)	Throughput
AMD 4-core Phenom 9550	2.20 GHz	1.0 Gbps
Intel 4-core Yorkfield Q9400	2.66 GHz	1.1 Gbps
Intel 8-core Xeon 2-socket E5440	2.83 GHz	2.0 Gbps

in the same way and were distributed into the four registers. In this way, the SIMD units were able to perform a parallel execution of the same instruction on four 32-bit data blocks, which is the maximum that fits on the 128-bit registers in the SPEs. Consequently, the encoding and decoding throughput was increased four-fold.

The OCTEON implementation method was very similar to the implementation on typical commercial desktop CPUs. It contains a quad-core MIPS64 architecture processor that has a dedicated L1 cache and shared L2 cache. Thus, each processor core can encrypt/decrypt plaintext data blocks independently. We used the POSIX thread library to implement the multi-threaded SEED algorithm.

4.2 Performance Results and Analysis

The test system environments of each platform are outlined in Table 5 together with speedup factors. In the cases of the CPU and GPU, only the systems that achieved the fastest performance in each group were chosen. As mentioned previously, the execution times of the multi-core desktop environments were 1.0 Gbps, 1.1 Gbps, and 2.0 Gbps. The highest performance achieved in the Intel 8-core system will be used as the baseline performance in this paper.

Fig. 6 shows the total performance results for all of the platforms. The GPU (GTX285) showed a substantial improvement over the baseline implementation, with a 9.5 Gbps throughput. The FPGA board has the slowest clock speed. Nevertheless, it showed a high throughput of 6.4 Gbps. The Cell BE showed a 10% better encoding performance than the 8-core Intel system.

The FPGA chip achieved a throughput of 6.4 Gbps for both the SEED encoding and decoding processes. On the Cell BE, a throughput of 2.2 Gbps was achieved for both processes. Though this is a lower throughput than that of the 16 round pipelined FPGA implementation, it shows a 10% increase in performance compared to the baseline 8-core desktop CPU platform.

Experiments were also executed on the VForce 2400 UTM network device with an OCTEON CN5230 MIPS64 architecture processor. The result was a 0.51 Gbps throughput. This low performance by the OCTEON platform shows that the software implementation of the SEED on the traditional network devices is meaningless. From this result, we can know that more powerful

Table 5. Performance and speedup of various platforms

Platform	Device	Number of Cores	Processor Speed (GHz)	Performance (Gbps)	Speedup
MIPS64	OCTEON CN5230	4	0.7	0.5	-
CPU	Intel 8-core E5440	8	2.83	2.0	Baseline
FPGA	Xilinx® Virtex-V	Pipelined	0.05	6.4	3.2x
PlayStation 3	IBM Cell BE	7	3.2	2.2	1.1x
GPU	GeForce GTX285	240 (30 SMs)	1.47	9.5	4.75x

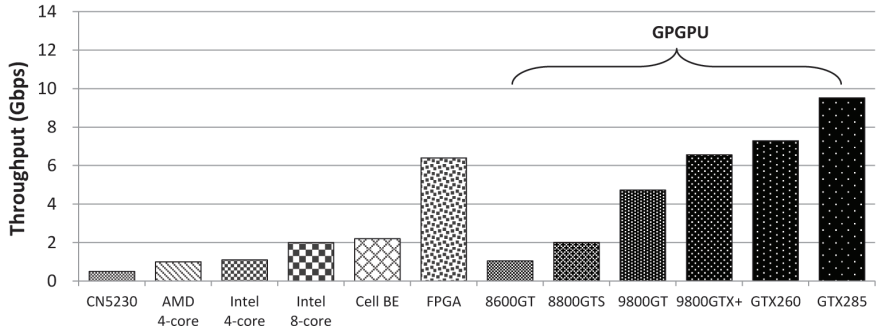


Fig. 6. Performance on various processors

processing elements are required to process the cryptographic algorithm on the real network.

Table 6 shows the scaled throughput on the same clock speed and the same number of cores for various processor architectures. In fact, each SP of NVIDIA GPU is very similar to an execution unit pipeline of typical superscalar processors, an SP could not be considered as a complete processor core. Therefore, we use the throughput of an SM to compare the performance of the "actual core" of the GPU with other processor cores. As shown in the results of scaled throughput for a 1GHz clock speed core, the GPU-based implementation provides higher throughput than other processor architectures. This result can be analyzed by the characteristic of GPU, which uses the warp execution model with SIMT execution methodology. Warp-based out-of-order execution efficiently exploits the execution hardware on GPU and effectively reduces the overhead of memory accesses using multiple sets of warps in an SM.

The detailed performance results on the GPUs with various numbers of threads are shown in Fig. 7. The GPU-based SEED block cipher algorithm had a 4.7 Gbps encryption performance on the 9800GT. This is 2.35 times faster than the baseline software implementation. A higher performance model, the NVIDIA GPU GTX285, had a 9.5 Gbps encryption performance, which is 4.75 times faster than the baseline performance. From this data, it is clear that using the GPU implementation can fulfill the fast encryption speed requirement of high-performance network systems, whereas current desktop/servers can only provide limited performance.

As seen in Fig. 7, the GT200 class GPUs had a maximum performance with 512 GPU threads. In contrast, the G80/G92 class GPUs reached a maximum performance at 272 threads. We believe that the architectural variation in the GPU hardware design makes the difference.

The G80/G92 class GPUs has 8,192 32-bit registers per SM, and the GT200 class GPU has 16,384 32-bit registers per SM. All NVIDIA GPU architecture also has 16KB shared memory

Table 6. Scaled throughput of various processors

Processor	Scaled Throughput (Gbps per 1GHz single core)
Intel 8-core E5440	0.09
OCTEON CN5230	0.18
IBM Cell BE	0.10
GeForce GTX285	0.215 (single SM)

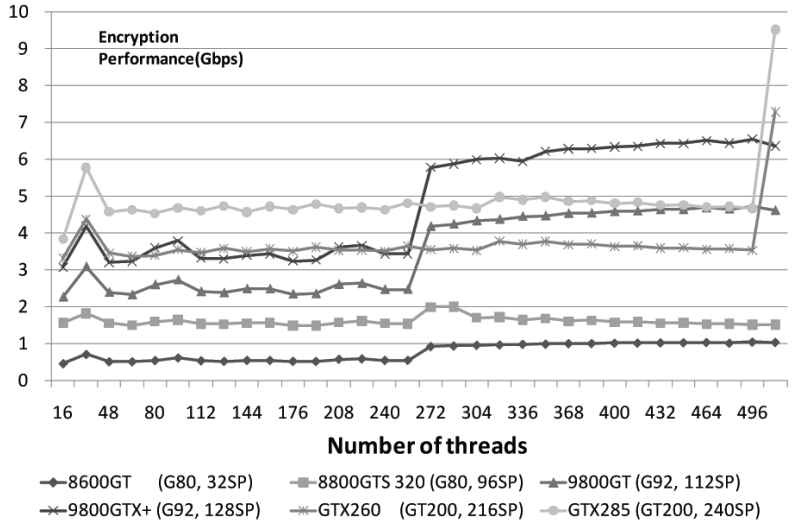


Fig. 7. Encryption performance on various GPUs

space per SM. To achieve maximal performance in typical CUDA applications, the SM should have and process as many thread blocks as possible within the register and the shared memory capacity, since the SM can process thread blocks continuously if one thread block is stalled due to the long latency instructions.

In our implementation, each GPU thread uses 13 registers and 16 Byte shard memory space in the encryption/decryption procedure. Since these registers and the shared memory space are used for a dedicated storage for each GPU thread, the number of GPU threads in a thread block affects the number of thread blocks per SM. If a thread block is composed of 272 GPU threads, 4,352 registers are required to process 1 thread block (272×13 + register allocation unit overhead). In this case, the G80/G92 class GPUs can allocate only one thread block on an SM due to the register limit. The GT200 class GPU can contain more GPU threads in one thread block. However, it only contains 1 thread block if there are 512 GPU threads in one thread block due to the shared memory limit (512×16 Byte + 16Byte shared memory allocation overhead).

Our GPU-based SEED block cipher algorithm shows maximum performance when only one thread block is allocated in an SM (272 threads on G80/G92 and 512 threads on GT200). It is the opposite result that we stated above. The reason is as follows: most instructions in the GPU-based SEED block cipher algorithm perform load operations corresponding to accesses to the SBOX tables on the constant cache. Therefore, the bottleneck of encryption/decryption procedure is owing to the access on the constant cache. The constant cache has very low access latency; however, it can handle only one access request in one clock cycle. Hence, it will serialize thread processing when more than one thread requests for accessing the constant cache in one clock cycle. This is called warp serialization. We can suppose that more than one thread block is allocated in an SM, then it increases warp serializations in encryption/decryption operations, since more and more GPU threads access the constant cache.

To examine our supposition, the proposed SEED block cipher implementation was analyzed with the CUDA Visual Profiler. The efficiency of a GPGPU application on an NVIDIA GPU

Table 7. Warp serialization analysis

Description	2 Thread Blocks/SM (256 Threads)	1 Thread block/SM (272 Threads)
Warp Serialization	9,130,015	5,162,051
Encryption Performance (Mbps)	2482.0	4187.0
Performance Enhancement	68.7%	

Table 8. Profiling results

Description	2 Thread Blocks/SM (256 Threads)
Registers per Thread	13
Global Memory Load Uncoalesced	0
Global Memory Load Coalesced	5,880
Global Memory Store Uncoalesced	0
Global Memory Store Coalesced	47,040

can be measured using the following factors: warp execution pattern, load/store pattern in the global memory, number of branches, etc. A detailed report is shown in Table 7 and Table 8.

Table 7 shows the difference of the warp serialization when one or more thread blocks are allocated in each SM. A huge number of warp serializations occurs when 2 thread blocks are allocated. In contrast, the warp serialization is significantly reduced and the encryption performance is improved by 68.7% when an SM has only 1 thread block. This result proves that the constant cache access bottleneck is the main cause of the performance imbalance in various numbers of GPU threads.

The warp serialized problem can be solved by using the shared memory for the SBOX and round keys, since the shared memory can accept 16 memory accesses at the same time. However, this method could cause performance degradation as it will restrict the number of threads allocated in each streaming processor.

Other performance evaluation factors are shown in Table 8. There were no non-coalesced global memory loads/stores. Hence, all of the data was transferred efficiently between the global memory and the shared memory. We achieved an additional 12.4% performance improvement by the coalesced global memory access.

5. RELATED WORK

The performance of a cryptosystem should guarantee high computation power, high data throughput, and adaptability to changes in the protocol. In order to accomplish secure and high-speed information transmission, previous studies have focused on implementing existing algorithms on hardware structures such as ASICS and FPGAs. Many researchers have implemented efficient versions of block cipher algorithms in FPGAs, including the New European Schemes for Signatures, Integrity, and Encryption (NESSIE) [5] and the Cryptography Research and Evaluation Committee (CRYPTREC) of Japan [6]. In most cases, the large amount of parallelism of FPGA enables giga-scale data processing in cryptosystems.

Denning et al. implemented a fully pipelined Camellia algorithm on FPGAs [7]. They imple-

mented three different versions for key generation/schedule schemes. Zambreno et al. implemented the AES algorithm on FPGAs. Their work focused on design space explorations for area, data throughput, and latency. Seo et al. implemented the SEED algorithm using FPGAs, focusing on minimizing resource utilization for small FPGA devices [8]. Kim et al. presents compact cryptographic hardware architecture that is suitable for the Mobile Trusted Module (MTM) [14]. A pipelined high-speed FPGA implementation of the SEED block cipher algorithm was proposed in [9]. This pipelined implementation mainly focused on the performance, at the cost of hardware resources and area. In fact, it greatly improved the throughput compared to the software versions of the application.

For cryptography on new hardware architectures, Cook et al. were the first to investigate the use of GPUs for symmetric key encryption [15]. They implemented AES on various previous-generation GPUs using OpenGL. Manavski proposed an efficient implementation of AES on a GPU, which succeeded in achieving an 8.2 Gbps throughput [16]. Canright et al. implemented the AES crypto algorithm on the Cell BE and attained up to a 2.1 Gbps throughput for ECB encryption and a 1.2 Gbps throughput for the CBC encryption mode, both using a 128-bit key size [17]. Bit-slice DES and AES were implemented on a GPU by Yang [18], and Osvik performed research on a bit-slice DES on the Cell BE platform of PlayStation 3 [19]. The symmetric key cryptography and AES encryption implementation on commodity GPUs was also researched by Harrison and Waldron [20], and by Yamanouchi [21].

This paper represents another step forward from [9], maximizing the performance of the SEED block cipher algorithm by taking advantage of, not only multi-core CPUs and FPGAs, but also potentially hundreds of computing cores in commodity off-the-shelf GPUs. Our work in this paper shows how the GPU can significantly increase performance of the encryption and decryption protocols, with high parallelism. The results exceed the performance achieved by the high-performance FPGA hardware accelerator in [9].

6. CONCLUSION

In this paper, we have shown a high-performance parallelized implementation of the SEED block cipher algorithm on GPUs. The proposed design was fully parallelized and tested on various NVIDIA graphic devices. The maximum performance provided a 9.5 Gbps computation throughput. This achieved throughput is 4.75 times faster than a full software version running on an 8-core CPU platform. It is up to 48% faster than that of the implementation executed on the FPGA, the most efficient previous version of the SEED implementation.

Performance of the GPU-based SEED implementation is sufficient to prevent the SEED block cipher from being a bottleneck in high-performance network systems, where the encoding and decoding speed of network security algorithms is crucial. We are confident that our implementations on newly developed hardware architectures would be of great use. GPU-based SEED implementation also provide better flexibility than other solutions using the dedicated hardware (FPGA or ASIC).

As future work, we plan to extend our GPU-based SEED implementation for SEED-256, which is a 256-bit augmented version of SEED, and we also plan to optimize the implementation on various block cipher modes of operation for practical use.

REFERENCES

[1] H. J. Lee, S. J. Lee, J. H. Yoon, D. H. Cheon, J. I. Lee and Korea Information Security Agency, "The SEED Encryption Algorithm," *The Internet Engineering Task Force RFC 4269* [online database], http://www.ietf.org/rfc/rfc4269.txt

[2] M. E. Hoque, F. Rahman, S. I. Ahamed and J. H. Park, "Enhancing Privacy and Security of RFID System with Serverless Authentication and Search Protocols in Pervasive Environments," *Wireless Personal Communications*, Vol.55, No.1, 2010, pp.65-79.

[3] C. S. Jang, D. G. Lee, J. Han, and J. H. Park, "Hybrid security protocol for wireless body area networks", *Wireless Communications and Mobile Computing*, Vol.11, 2011, pp.277-288.

[4] H. Xie, L. Zhou and L. Bhuyan, *"Architectural Analysis of Cryptographic Applications for Network Processors," Proceedings of IEEE First Workshop on Network Processors with HPCA-8*, Boston, February, 2002.

[5] NESSIE, *NESSIE project announces final selection of crypto algorithms*, February, 2003. IST-199-12324.

[6] CRYPTREC, *Report of the Cryptographic technique evaluation*, March, 2003. CRYPTREC Report 2002.

[7] D. Denning, J. Irvine and M. Delvin, *"A key agile 17.4Gbit/sec Camellia implementation," Proceedings of International Conference on Field Programmable Logic and its Applications (FPL 2004),* Vol.3203, 2004, pp.546-554.

[8] Y.H. Seo, I.H. Kim and D.W. Kim, *"Hardware Implementation of 128-bit Symmetric Cipher SEED," Proceedings of the Second IEEE Asia Pacific Conference on AP-SIC 2000*, 2000, pp.183-186.

[9] J. Yi, K. Park, J. Park and W. W. Ro, *"Fully pipelined hardware implementation of 128-bit SEED block cipher algorithm," Proceedings of the Fifth International Workshop on Applied Reconfigurable Computing,* 2009, pp.181-192.

[10] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips, *"GPU Computing," Proceedings of the IEEE*, May 2008, pp.879-899.

[11] Y-T Lin and P-S Chen, "Compiler Support for general-purpose computation on GPUs," *Journal of Supercomputing*, Vol.50, No.1, 2009, pp.78-97.

[12] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn and T. J. Purcell TJ, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*, Vol.26, No.1, 2007, pp.80-113.

[13] NVIDIA, "NVIDIA CUDA Programming Guide 4.0," *NVIDIA GPU Computing Documentation*, http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf

[14] M. Kim, Y. Kim and H. Cho, "Design of Cryptographic Hardware Architecture for Mobile Computing", *Journal of Information Processing Systems*, Vol.5, No.4, 2009, pp.187-196.

[15] D. Cook and A. Keromytis, *CryptoGraphics: Exploiting graphics cards for security (Advances in Information Security),* Springer-Verlag New York, 2006.

[16] S. A. Manavski, *"CUDA compatible GPU as an efficient hardware accelerator for AES cryptography," Proceedings of the IEEE International Conference on Signal Processing and Communications (ICSPC),* 2007, pp.65-68.

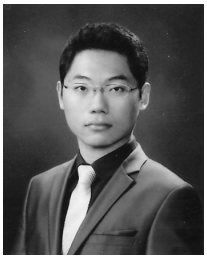
[17] D. Canright, G. Dinolt, S. Garfinkel, J. Herzog and B. Allen, *Implementing AES on the CellBE*, 2009. National Security Agency Technical Report NPS-MA-09-001.

[18] J. Yang and J. Goodman, *"Symmetric key cryptography on modern graphics hardware," Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security*, 2007, pp.249-264.

[19] D.A. Osvik and E. Tromer, "*Cryptologic applications of the PlayStation 3: Cell SPEED*," http://www.hyperelliptic.org/SPEED/slides/Osvik_cell-speed.pdf

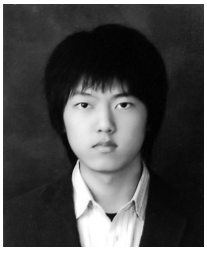
[20] O. Harrison and J. Waldron, *"AES encryption implementation and analysis on commodity graphics processor units," Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems*, 2007, pp.209-226.

[21] T. Yamanouchi, *AES encryption and decryption on the GPU (GPU Gems 3)*, Addison-Wesley Professional, 2007.



Sangpil Lee

He received his B.S. degree in Electrical & Electronic Engineering from Yonsei University, Seoul, Korea, in 2010. He is currently pursuing his Ph.D. degree in the School of Electrical & Electronic Engineering at Yonsei University, and is with the Embedded Systems & Computer Architecture Lab. His current research interests are general-purpose computation using graphics hardware and parallelization.



Deokho Kim

He received his B.S. degree in Electrical & Electronic Engineering from Yonsei University in Seoul, Korea, in 2010. He is currently pursuing his Ph.D. degree in the School of Electrical & Electronic Engineering at Yonsei University, and is with the Embedded Systems & Computer Architecture Lab. His current research interests are parallel computing on the Cell Broadband Engine heterogeneous processor.

Jaeyoung Yi

She received her B.S. degree in Mathematics/Computer Science and her M.S degree in Electrical & Electronic Engineering from Yonsei University in Seoul, Korea, in 2008 and 2010, respectively. Her current research interests include parallelization in multi-core systems and in hardware platforms.

Won Woo Ro

He received his B.S. degree in Electrical Engineering from Yonsei University, Seoul, Korea, in 1996. He received the M.S. and Ph.D. degrees in Electrical Engineering from the University of Southern California in 1999 and 2004, respectively. He worked as a research scientist in the Electrical Engineering and Computer Science Department at the University of California in Irvine. He currently works as an Assistant Professor in the School of Electrical and Electronic Engineering at Yonsei University. Prior to joining Yonsei University, he has worked as an Assistant Professor in the Department of Electrical and Computer Engineering of California State University, Northridge. His industry experience also includes a college internship at Apple Computer Inc. and as a contract software engineer at ARM Inc. His current research interests are high-performance microprocessor design, compiler optimization, and embedded system designs.