Heterogeneous Parallel Computing for a Security Application

[1]Nhat-Phuong Tran, *[2]Myungho Lee, [3]Sugwon Hong, [4]Dong Hoon Choi
[1, *2 Corresponding Author, 3]*Dept of Computer Science and Engineering, Myongji University*
38-2 San Namdong, Cheo-In Gu, Yong In, Kyung Ki Do, Korea 449-728
myunghol@mju.ac.kr
[4] *Korea Institute of Science and Technology Information (KISTI)*
245 Dae Hak Ro, Yu Seong Gu, Daejeon, Korea 305-806

Abstract

Heterogeneous High Performance Computing (HPC) platforms consisting of multicore microprocessors and multicore GPUs are becoming increasingly popular these days. Many computationally demanding applications are parallelized on these platforms to deliver more than 100-times performance improvements compared with a sequential execution on single core CPU, thus opened up opportunities for the heterogeneous HPC architectures. Computer and network securities such as data encryption/decryption, among others are interesting application areas where the heterogeneous architectures can be adopted. In data encryption, for example, processing a heavy traffic load is an intimidating task due to the high computational requirements. In order to keep pace with the high input data rate, real-time processing of data encryption is essential. In this paper, we propose a new approach to parallelize a data encryption algorithm on a heterogeneous HPC platform by exploiting a task-level coarse-grain parallelism by bundling multiple unit blocks where the data encryption is applied. The proposed approach leads to significant performance improvements compared with previous approaches.

Keywords*: Heterogeneous HPC Architecture; Multicore; GPU; Parallel Algorithm*

1. Introduction

Latest High Performance Computing (HPC) platforms are built with multicore microprocessors and multicore GPUs (Graphic Processing units). They are commonly called as Heterogeneous High Performance Computing (HPC) platforms because the main composing chips have different architectures and characteristics. A good example is the Tianhe-1A built in China which was ranked number one in the top 500 list by delivering the world best LINPACK benchmark performance in November 2010.

Various applications are being developed on heterogeneous HPC platforms deploying their excellent floating-point performance. They include graphics applications, HPC applications such as linear solvers, and non-HPC applications such as pattern matching for bioinformatics code. For those applications of which the characteristics match well with the heterogeneous HPC architectures, huge performance improvements, usually greater than 100-times speedups, have been reported.

Computer and network security applications are interesting ones where the heterogeneous HPC architectures can be also adopted. Traditionally, for security functions, processing a heavy traffic load has been an intimidating task. To overcome the limitation of pure software implementation, some dedicated hardware solutions have been developed and used for security functions. Hardware solutions can meet the real-time processing requirements imposed on these functions, however it makes the cost of network devices expensive. Efficiently deploying the GPUs on heterogeneous architectures can be an attractive alternative solution for both performance and cost.

In this paper, we parallelize a data encryption algorithm as an example case and a crucial piece of network and computer securities on heterogeneous HPC platforms. For the efficient parallelization, we propose a new technique to exploit task-level coarse-grain parallelism for the data encryption application by bundling multiple unit blocks where the data encryption is originally applied. This approach reduces the overheads associated with the parallel execution of the data encryption application. By implementing the proposed parallelization technique on a heterogeneous HPC platform, we've observed a significant performance improvement compared with previous approaches.

The rest of the paper is organized as follows: Sections 2 shows the architecture of the heterogeneous HPC platforms including their composing chips (general-purpose multicore processors

and multicore GPUs) and their programming models. Section 3 gives a brief overview of the data encryption algorithm. Section 4 explains our parallelization technique compared with the previous approaches. Section 5 shows the experimental results of the new approach compared with the previous one. Section 6 concludes the paper.

2. Overview of Heterogeneous Architecture

In this section, we first describe the architectures and programming models for a general-purpose multicore microprocessor and a multicore Graphic Processing Unit (GPU). Then we describe the heterogeneous HPC architecture based on these chips and its programming model.

2.1. Multicore Microprocessor Architecture

Microprocessor designers have long been considering many design choices to efficiently utilize the ever increasing effective silicon area with the increase of transistor density. The noticeable trend since mid-2000 is the multicore design. Instead of employing a complicated processor pipeline on a chip with an emphasis on improving single thread's performance, incorporating multiple processor cores on a single chip has become a main stream microprocessor design trend. As a Chip Multi-Processor (CMP), it can execute multiple software threads on a single chip at the same time. Thus a multicore processor provides a larger capacity of computations performed per chip for a given time interval (or throughput) [16].

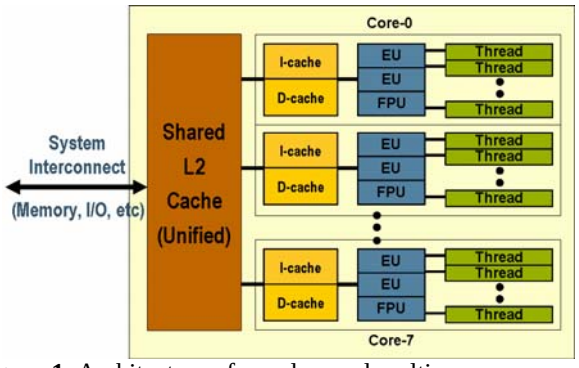


Figure 1. Architecture of an advanced multicore processor

In addition to the CMP based multi-core design, some designs go one step further to incorporate Simultaneous Multi-Threading (SMT) or similar technologies such as Intel Hyperthreading on a processor core. Examples are Intel Nehalem and UltraSPARC T2/T3 microprocessor from Oracle/Sun. Figure 1 shows the architecture of an advanced multicore processor. On each processor chip, there are *N*-processor cores, with each core having its own level-1 on-chip cache. The *N*-cores share a larger level-2 cache on or off the processor chip. Each core also has *M* hardware threads performing SMT or similar features. Thus it supports two levels of parallelism.

2.2. Multicore GPU Architecture and Programming

The Graphic Processing Unit (GPU) was introduced in the late 1990s as a co-processor for accelerating the simulation and visualization of 3D images commonly used in applications such as game programs. Since then GPU has become widespread and these days it is incorporated in almost all desktop and palmtop computers as well as HPC platforms. In the architecture of earlier GPU's, there were separate processing units for Shader, Vertex, Pixel. In the latest GPU's, those units are incorporated into multiple programmable processing units or thread processors. This design trend reflects the fact that application programs dealing with 3D graphic images are suitable for SIMD (Single Instruction Multiple Data) processing. Furthermore recent GPU architectures adopt multicore design. As can be seen in Figure 2 [11], multiple thread blocks or cores are incorporated on a chip. Each core includes multiple thread processors.

In order to utilize the flexible hardware design, more user friendly programming environments are recently developed by the GPU vendors. CUDA from Nvidia, OpenCL from AMD/ATI are good examples of such software environments [12, 13]. Using those environments, programmers can have more direct control over the GPU pipeline and the memory hierarchy, whereas in the old GPU's they relied on specific graphics API's. The flexible GPU hardware and user friendly software have recently led to a number of innovative performance improvements in many application areas and more improvements are still to come [14, 18].

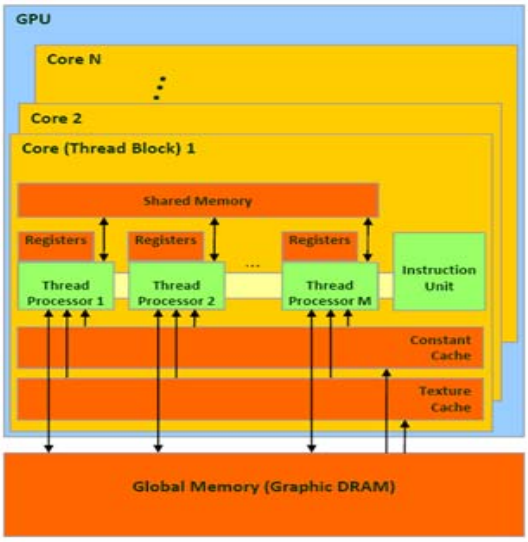


Figure 2. General architecture of a latest GPU

For executing CUDA programs, for example, a hierarchy of memories is used on the Nvidia's GPU. They are registers and local memories belonging to each thread, a shared memory used in a thread block, and Global memory accessed from the thread block arrays [12, 13]:

- Global memory is an off-chip GDDR through which GPU can communicate with the host CPU.
- Shared memory sits within each thread block. The access time closely matches with the register access time, thus it is a very fast memory.
- Registers are used for temporarily storing data used for GPU computations, similar to CPU registers.

2.3. Heterogeneous HPC Architecture and Programming

Many heterogeneous HPC platforms are recently built using multicore microprocessors and multicore GPU's. They are based on unit nodes where these heterogeneous chips are employed on the same system board. Those unit nodes are interconnected using a high-speed interconnect such as Infiniband. A good example system is the Tianhe-1A announced in Nov 2010 which had delivered the world record LINPACK performance. Figure 3 above shows the unit node architecture of the heterogeneous HPC platforms.

Various programming models have been proposed and deployed for heterogeneous HPC platforms. Earlier models used OpenMP or MPI for multicore processors and proprietary solutions for GPU's such as CUDA. A more recent model is the OpenCL standard which is being developed by the Khronos Group with a consortium consisting of a few dozen companies including major microprocessor vendors (Intel, Fujitsu, ARM), GPU vendors (Nvidia, AMD), DSP companies (Texas Instrument), and other chip suppliers (IBM). An OpenCL program can be run on multicore processors, GPU, DSP's, IBM CellBE, etc., thus it is platform independent and targeted as the industry standard for heterogeneous parallel programming. Although OpenCL is expected to evolve as the industry standard, its wide acceptance in the user community is still in question. Main reason is that OpenCL reveals a lot of low level architecture details in the program which burdens application developers significantly.

There emerges another stream of thoughts to leverage existing parallel programming APIs such as OpenMP for programming heterogeneous HPC platforms. Another similar approach, called OpenACC, is currently being proposed by a group of four companies--Nvidia, Cray, Portland Group, CAPS—from OpenMP Architecture Review Board (ARB). The goal of OpenACC is to develop a high-level API for accelerator chips such as GPUs. The draft version is currently available for public comments before version 1.0 is officially announced in the 1Q of 2012. The next step for OpenACC is to incorporate it in the OepnMP framework by 4Q of 2012, as OpenMP 4.0.

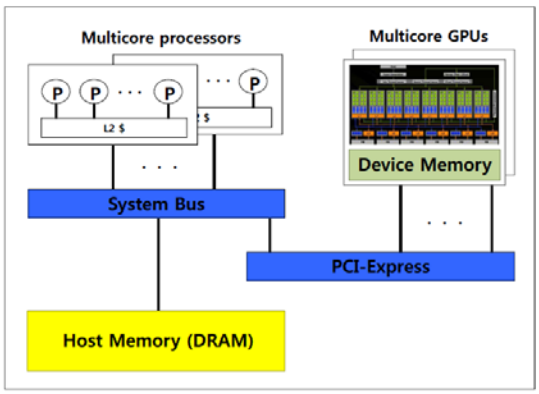


Figure 3. Unit node architecture of heterogeneous HPC platform

3. Overview of Data Encryption Application

As our target security application, we use a data encryption application AES. The Advanced Encryption Standard (AES) is a symmetric cryptographic algorithm published by NIST [10] which had replaced the previous Data Encryption (DES) standard. AES is the most widely used block cipher in recent years because of its high security and low cost. AES algorithm is carried out by applying a number of repetitions of transformation rounds that converts the input plain text into the final cipher text. AES has a fixed block size of 128 bits with three key lengths 128 bits, 192 bits and 256 bits, thus comprises three block ciphers AES-128, AES-192, and AES-256. Depending on the key and the block lengths, the number of rounds of AES varies: 10 for 128 bits, 12 for 192 bits and 14 for 256 bits. Each round includes several steps. The output of each round is the input of the next round. Each round consists of the same steps, except for the first round where an extra addition of a round key is added and for the last round where the last step is skipped [2, 3, 5].

Computations for AES-128 consist of the following steps [3]:

- The input which is a block of 128 bits plain text is represented by a 4x4 byte matrix called “State”.
- KeyExpansion step is used to generate the RoundKeys from the original key for rounds.
- The four round steps are AddRoundKey (XOR each column of the State with a word from key schedule), SubBytes (process the State with non-linear byte substitution table (S-box) that operates on each of the State bytes independently.), ShiftRows (cyclically shifts the last three rows in the State by different offsets), MixColumns (takes all of the columns of the State and mixes their data to produce new columns)
- For the initial round, perform only the operation AddRoundKey.
- For the next N–1 rounds, perform four operations SubBytes, ShiftRows, MisColumn and AddRoundKey.
- For the last round, perform the same operation the previous N–1 rounds except without the MixColumns operation.

4. Heterogeneous Parallel Processing of Data Encryption Application

In a typical network-based application with security, data encryption and decryption are intensively performed with respect to large amounts of data continuously received from and forwarded to other senders/receivers. In such an environment, the demand for parallel execution

of AES is high. As described in the previous section, data encryption in AES goes through a number of rounds with respect to a unit sized block. Within each round, four computation steps involving XOR, byte substitution, shift, etc., are performed with respect to each block. In order to encrypt and decrypt more than one block, modes of operation have been developed. In this paper we use CTR mode because it is parallel in nature and secure by using different keys in blocks. A straightforward approach parallelizes the execution of four computation steps in AES-CTR. In other words, it parallelizes the encryption of each unit-sized block. This approach, however, incurs large parallelization overheads such as synchronization and multiple concurrent accesses to the same cache block which can lead to a false-sharing. Thus this approach is not a desirable one.

Contrast to the fine-grain approach, a coarse-grain approach attempts to parallelize the AES-CTR algorithm at the level of unit-sized block. Since a large amount of data is typically received at a computing node in a network-based application, there are multiple blocks in the received data. Assume that N-blocks are received in total and P-threads are used for the encryption. P-blocks are encrypted at the same time, and the execution is repeated N/P-times. This approach, thus, does not attempt to speed-up a single block encryption, but attempt to divide the total number of blocks amongst a number of threads. This approach may lead to a longer run time to encrypt a single block, but incurs significantly low parallelization overheads such as synchronization compared with the fine-grain approach. Furthermore, it significantly reduces the risk of false-sharing. Therefore this approach usually gives better performance than fine-grain approach in encrypting N-blocks [1].

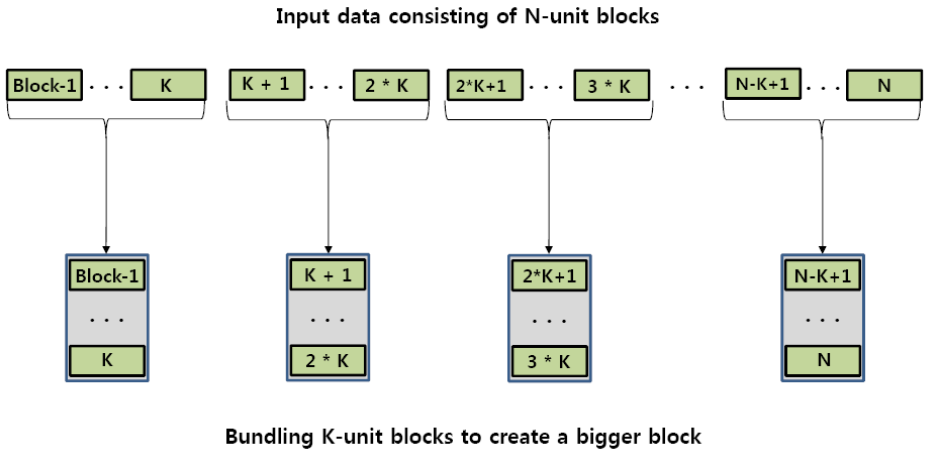


Figure 4. Data encryption using the new approach

Given N-blocks, the coarse-grain approach distributes N/P-blocks to each core where P is the number of cores available for parallel execution of AES. Each processor core, thus, encrypt the assigned blocks sequentially N/P-times. The N/P-times repetition involves parallelization overheads. In our approach, we intend to significantly reduce these overheads by increasing the level of parallelism by bundling multiple unit blocks into a big block and assign the enlarged block to each thread. For instance, creating a big block (e.g., 256-Bytes) by bundling 16-Bytes unit blocks 16-times (see Figure 4). With the bigger block size, we reduce the cost of parallelization. Thus, each thread can be used more effectively because the portion of the computation in the overall parallel execution time increases. This approach, however, needs to enlarge the key size also to the length of the resulting big block size so that the encryption of the resulting big block can be performed at one time.

5. Experimental Results

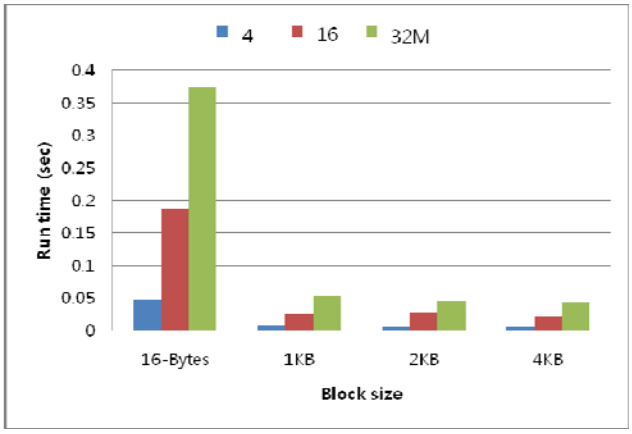
We've conducted experiments to compare the performance of the coarse-grain parallelization approach and our new parallelization approach using the big block size. The experiments were

conducted on a heterogeneous HPC platform consisting of general-purpose multicore processor and a GPU. We present the results in the following subsections.

5.1. Results on GPU

We've parallelized the AES-CTR code for both the coarse-grain approach and the new approach on our experimental platform (2.2Ghz, 4-core Intel Core 2 Duo processor with 2GB DRAM, running Centos 5.5 OS, Nvidia GT9500 GPU). We used CUDA to parallelize the AES-CTR code. The run times of the two approaches were measured. Figure 5 shows the experimental results of both approaches:

- The new approach (block sizes 1KB, 2KB, 4KB) shows dramatic performance improvements on a GPU. The speedup reaches up to 8.53 (16MB data with 4KB block). Considering that the number of cores on GT9500 is small (32), the performance improvement is impressive.
- Larger block sizes generally give more speedups than small blocks. This was quite expected when we designed the new parallelization approach using a big block size, as the bigger blocks reduce the number of iterations in the major computation routines of the data encryption application which reduces the parallelization overheads.



(16-Bytes: coarse-grain approach; 1/2/4KB: new approach)

Figure 5. Performance results on Nvidia GT9500

5.2. Performance Considerations

In the AES-CTR algorithm, the time to access lookup tables takes a significant portion of the runtime. To reduce the latency of lookup table accesses, we store the tables in the shared memory of the GPU which is shared by all threads in a block (see Figure 6). In our parallel algorithm 4 lookup tables are stored in shared memory, whereas plain text is stored in the global memory. When threads need to bring data for encryption, it is fetched from the global memory to the shared memory. The time to fetch data from the global memory to the shared memory affects the run time of the program, because global memory is a DRAM of which the typical access time takes hundreds of clock cycles. Furthermore, the bandwidth is also limited. Having many threads available for execution can theoretically tolerate the long global memory access latencies, it can easily lead to a situation where traffic congestion in the global memory access paths prevents all but very few threads from making progress. This will increase the idling cycles of the multiple Streaming Multiprocessors in the GPU. If there are too many threads in a thread block, corresponding access times for fetching data will increase significantly. Thus we need to find an optimal number of threads and the data size for the threads to handle.

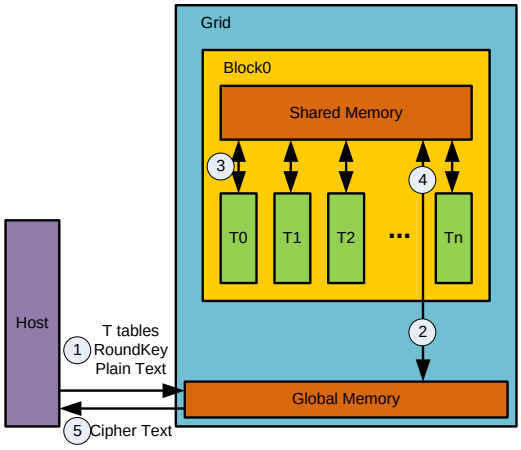


Figure 6. Memory Access scenario for AES-CTR

6. Conclusion

In this paper, we introduced a heterogeneous HPC platform architecture commonly built and used these days for various HPC applications. We then presented a parallelization technique for a data encryption application as an example security application to be executed on such a platform. The proposed approach parallelizes the data encryption by bundling multiple unit blocks which are encrypted at one time in the original algorithm. Therefore this approach reduces the overheads incurred with the parallelization. Experimental results on a heterogeneous system consisting of a 4-core, 2.2Ghz Intel processor with 2GB DRAM, running Centos 5.5 OS, and Nvidia GT9500 GPU shows that the new approach achieves up to 8.53x speedup compared with the original approach run on the same GPU. The parallel implementation was conducted using CUDA. The speedup is larger on the GPU with a larger block size as the parallelization overheads decrease.

7. Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the ministry of Education, Science, and Technology (Grant No: 2010-0012059).

8. References

[1] A.D. Biagio, A. Barenghi, G. Agosta, G. Pelosi, “Design of a Parallel AES for Graphics Hardware using the CUDA framework”, Proceedins of the 2009 IEEE International Symposium on Parallel & Distributed Processing, May 2009.

[2] J. Daemen, V. Rijmen, “The Design of Rijndael: AES The Advanced Encryption Standard”, Springer-Verlag, 2002.

[3] J. Daemen, V. Rijmen, “AES Proposal Rijndael [EB OL]”, http://www.daimi.au.dk/~ivan/rijndael.pdf, Oct 2010.

[4] L. Deri, “nCap: Wire-speed packet capture and transmission,” the IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services 2005.

[5] Morris Dworkin, “Recommendation for Block Cipher Modes of Operation”,

[6] F. Fusco and L. Deri, “High-Speed Network Traffic Analysis with Commodity Multi-Core System,” http://svn.ntop.org/imc2010.pdf.

[7] K. Fatahalian, M. Houston, "A Closer Look at GPUs", Comm. of ACM, Oct 2008.

[8] B. He, N. Govindaraju, Q. Luo, B. Smith, “Efficient Gather and Scatter Operations on Graphics Processors“, Proceedings of the SuperComputing 07, pp. 175-186, Nov 2007.

[9] He Jin, Fang Zhiyi, Ji Liang, Cai Ruicheng, and Chen Lin, "GPU-Based Research of Highly Efficient Ray Tracing", AISS: Advances in Information Sciences and Service Sciences, Vol. 3, No. 10, pp. 207 ~ 215, 2011

[10] National Institute of Standards and Technology (NIST), “FIPS-197: Advanced Encryption Standard.” http://www.itl.nist.gov/fipspubs/ , Nov. 2001.

[11] “Nvidia gtx280", http://kr.nvidia.com/object/geforce_family_kr.html

[12] "Nvidia CUDA", http://developer.nvidia.com/object/cuda.html

[13] Matt Pharr et. al., "GPU Gems 2", Addison Wesley, 2004.

[14] Nvidia CUDA Programming Guide", http://kr.nvidia.com/object/cuda_develop_kr.html

[15] M. Quinn, Parallel Programming in C with MPI and OpenMP, McGraw Hill, 2004.

[16] L. Spracklen and S. Abraham, Chip MultiThreading: Opportunities and Challenges, 11th International Symposium on High-Performance Computer Architecture (HPCA-11), pp 248-252, 2005.

[17] Tao Sun, Jian Sha, Lin Feng, “A GPU-based parallel algorithm for time series pattern mining”, JCIT: Journal of Convergence Information Technology, Vol. 6, No. 12, pp. 163 ~ 170, 2011

[18] V. Volkov and J.W. Demmel,“Benchmarking GPUs to Tune Dense Linear Algebra”, Proceedings of the SuperComputing 08.