2009 International Conference on Computer Engineering and Applications
IPCSIT vol.2 (2011) © (2011) IACSIT Press, Singapore

The Implementation and Analysis of Important Symmetric Ciphers on Stream Processor

Ping Yao [1], Mu Xu [1], Gu Liu [1], Guang Xu[1], Hong An[1 2 +], Wenting Han[1]

[1]Department of Computer Science and Technology, University of Science and Technology of China, Hefei, 230027, China

[2] Key Laboratory of Computer System and Architecture, Chinese Academy of Sciences, Beijing, 100080, China

Abstract. Imagine is a stream processor that employs a two-level register hierarchy with 9.7 Kbytes of local register file capacity and 128 Kbytes of stream register file (SRF) capacity to capture producer-consumer locality in stream applications. Parallelism is exploited using an array of 48 floating-point arithmetic units organized as eight SIMD clusters with a 6-wide VLIW per cluster. It achieves good performance in multimedia applications, signal processing, and scientific computing, but how it works in information security area remains unknown. This paper implements Base64 and several important symmetric ciphers on Imagine, including Blowfish, Rijndael and RC5. The result shows a speed up of 3x, 2.4x, 2.5x, and 5x over traditional implementations respectively. Imagine shows good potential in information security area. By comparing the performances achieved by Imagine and general purpose processor and analyzing the characteristics of applications, we also propose a checking model for choosing applications which can get better performance on Imagine. By implying this model, we can find whether an application is suitable for Imagine, rather than implementing it on Imagine and then check the performance, which would waste much time.

Keywords: Stream Processing; Stream Programming Model; Symmetric Cipher; Imagine

1. Introduction

Stanford's Imagine is the first stream processor prototype [1] which implements the stream model. Aiming at data level parallel applications, Imagine can fully use the massive on-chip computing elements and cover memory accessing latency. The programming model of Imagine is Stream Processing, which is described in [2]. Imagine can satisfy the real-time demands and high computing intensity of multimedia applications and digital signal processing, which are becoming the most heavily used workload recently. For example, MPEG-2 encoder on Imagine can get 15.35GOPS, 287fps for 320x288 24-bit images [3]. But how it works in information security area remains unknown.

Symmetric ciphers are used widely in information security area. In this paper, we implement and analyze Blowfish, Rijndael and RC5 on Imagine. In these ciphers, one block is encrypted or decrypted each time. Cipher mode decides the relationship between adjacent blocks. In ECB mode, every block has nothing to do with others, which shows perfect data level parallelism. We can expect good performance when we use ECB mode to implement these ciphers.

The most common characteristic of symmetric ciphers is random accessing to a fix-size array. The efficiency of random accessing determines the run time of the whole application. We'll research how to efficiently implement random accessing to arrays on Imagine. Meanwhile, we use Base64, which has no random access to arrays, to further illustrate the importance of high efficient implementation.

+ Corresponding author. Tel.: +86-0551-3603583; fax: +86-0551-3603583.
E-mail address: han@ustc.edu.cn.

Our test platforms are: general purpose processor (GPP) (AMD Athlon64 X2 Dual Core Processor 3800+ stepping 01, MEM = 2G), Isim (Imagine's simulator, 500MHz). The experiment result shows that Blowfish, Rijndael, RC5 and Base64 on Imagine gets speed up 2.4x, 2.5x, 5x and 3x over those on GPP respectively. By analyzing the experiment result and characteristics of them, we propose a checking model for choosing applications which can get better performance on Imagine: 1) have massive available data level parallelism; 2) no random array access or only access an array whose size is smaller than the size of scratch pad of Imagine. We can decide whether to implement an application on Imagine by using this model rather than implementing it on Imagine and then checking the performance, which would waste much time.

This paper is organized as follows: in part2, we introduce Imagine architecture and programming model briefly; then in part 3, we show how we implement Blowfish, Rijdael, RC5, and Base64 on Imagine and the analysis of the experimental results; in part4, we conclude our work.

2. Architecture and Programming Model of Imagine

2.1. Imagine Architecture

Imagine is a programmable stream processor and is a hardware implementation of the stream model. A block diagram of the architecture is shown in Fig.1. Imagine is designed to be a stream coprocessor for a general purpose processor that acts as the host, as seen in the figure. The stream controller sequences stream commands from the host processor and issues them to the various modules on the chip. All data stream transfers are routed through a 32 KW stream register file (SRF). The streaming memory system transfers entire streams between the SRF and off-chip SDRAM. Kernel programs consist of a sequence of VLIW instructions and are stored in a 2K x 576-bit RAM in the microcontroller. The microcontroller issues kernel instructions to eight arithmetic clusters in a SIMD manner. Each cluster consists of six ALUs (plus a few utility functional units) and 304 registers in several local register files (LRFs). The network interface module routes streams between the SRF of its node and the external network.

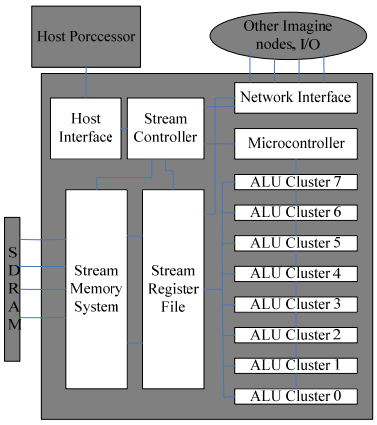

Fig. 1: Imagine architecture block diagram.

2.2. Programming Model

The programming model is divided into two level: stream level (using StreamC) and kernel level (using KernelC). In stream level, programmers define streams and kernels. The kernel size and stream schedule are their most concerned objects. In kernel level, programmers concern about how to fully use hardware resource to achieve good performance. Imagine supports ILP by compile kernel into VLIW, DLP by using one kernel instruction to control multiple computing clusters, i.e. SIMD, and TLP by the overlap of kernel computing and stream loads and stores between off-chip SDRAM and SRF. Further discussion can be found in [5].

3. Experiments and Analysis

3.1. Brief Introduction to Symmetric Ciphers

In this paper, we implement Blowfish, Rijndael [4], RC5 [6], and Base64 [7] on Imagine. We first introduce these four ciphers briefly in table 1. While the former three ciphers all need to randomly access a fix-size array, Base64 doesn't. We use it as a contrast to the others.

Tab. 1: several important symmetric ciphers

Symmetric cipher	Brief description
Blowfish	high-efficient algorithm , widely used in personal information security area.
Rijndael	Advanced Encryption Standard. The block and key length can be 16, 24 or 32 bytes. We use 16-byte block and key. The process of encryption has 10 rounds. In round 1-9, it uses four modules: subbyte, rotcolum, colshuffle, and addkey. In round 10, use subbyte, rotcolum, and addkey.
RC5-w/r/b	Simple, quick but safety cipher, very small memory consumption, can be used in smart card or other memory-limited device. W, r, b means block length, iteration numbers and key length respectively. In this paper, w = 32, r = 12, b = 16.
Base64	Used to encrypt emails or webs. Encrypts plain text using cipher table rather key. No need to randomly access arrays.

3.2. Implementations on Imagine

As we can see, Blowfish, Rijndael, and RC5 all need to access a fix-size array. The most important part of the implementations of these symmetric ciphers is how to efficiently do random accesses to an array on Imagine. We use Blowfish as an example to illustrate our method to solve this problem.

By analyzing the Blowfish encryption algorithm, we can see that the pre-processing to produce the P-box and S-box is a chain process, which can't implements on Imagine, so we only implement the encrypt function on Imagine.

In Blowfish, we need to randomly access P-box and S-box frequently. So how we implement these accesses matters the performance. In order to randomly access the two boxes, we should put them in the scratch pad (SP) of Imagine in style of array. The size of P-box is 18 words, S-box is 256 words, but the size of SP is only 256 words, so we can't hold both of them in SP. In our implementation, we put a whole copy of P-box and part of S-box in each SP: k-th of S-box stores in sbox_c[k/8] of cluster k%8, as in Fig 2.

Due to the distribute S-box storage, accessing the S-box becomes complicated. Inter-cluster communication is involved. The S-box item needed by one cluster may be in another cluster, so we need the permutation intrinsic to get the right item from the right cluster for each one. When we need S-box[k], we have to get sbox_c[k/8] from cluster[k%8]. Because the data encrypted by each cluster may be different, k on each cluster may be different too. Meanwhile Imagine uses SIMD manner to control 8 clusters, so we have to process the S-box random access sequentially. When cluster I need to access S-box, it broadcasts its k to all clusters, then all clusters do sdata=sbox_c[k/8] operation. After that, they communicate sdata with each other according to k%8, using permutation intrinsic. But only cluster I replace its own sdata with new sdata eventually.

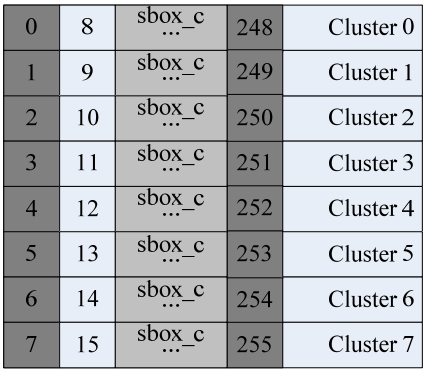

Fig. 2: The distributed S-box in Imagine.

3.3. Experimental Result

As showed in Fig.3, Imagine implementation of Blowfish is much faster than AMD implementation, the speed up is 2.4x. Lots of time is spent on inter-cluster communications to get the right S-box item for every cluster and the performance is not so well. Actually, if we can modify the hardware of Imagine, things will be much simpler. When the size of SP is big enough to hold a whole copy of P-box and S-box, the random accesses to them will be very naïve and efficient. No inter-cluster communication needed. The result shows the speed up becomes 6x.

There are plenty much of producer-consumer locality among the four modules of Rijndael. [8] points out that applications with producer-consumer locality seems to easily achieve good performance on Imagine. But in our implementation, the result is not well enough. By analyzing our implement code, we find the reason is

the platform limits. We have to use 32 bit operations, but only use 8 bit result, wasting 3/4 computing ability. The speed up over AMD implementation is only 2.5x.

Though Base64 is the simplest algorithm to implement on Imagine, its performance is not good enough. The speed up is about 3x over AMD implementation. We have to do some additional operations when implement it on Imagine. About 1/4 computing ability is wasted.

RC5 achieves satisfactory result on Imagine, the speed up is 5x, nearly 2 times more than Blowfish and Rijndael. The array needed to be randomly accessed in RC5 is only 26 words, so it can be put in a SP, which avoids the communications needed to get Sbox. It doesn't waste computing ability like Rijndael and Base64. So it gets the highest speed up.

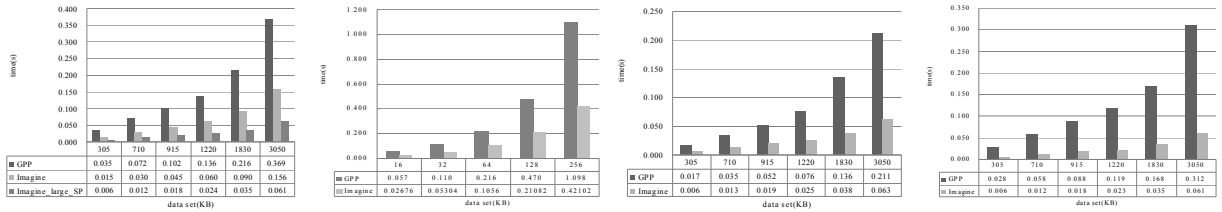

Fig. 3: Performance of Blowfish, Rijndael, Base64, RC5 on Imagine and GPP.

3.4. Result Analysis

The experiment result shows that Blowfish, Rijndael, RC5 and Base64 on Imagine gets speed up 2.4x, 2.5x, 5x and 3x over those on GPP respectively. To understand the performance improvement, we must compare the architecture differences between Imagine and GPPs. We carefully compare the differences in three aspects: 1) arithmetic to memory bandwidth ratio, DLP applications naturally need high arithmetic to memory bandwidth ratio; 2) on-chip memory system, whether locality in applications can be caught by hardware and software system heavily depends on the on-chip memory system; 3) the overlap of computing and memory access, the more overlaps, the less execution time.

- Imagine's arithmetic to memory bandwidth ratio is much higher than GPP's. The former is about 20:1 [8], while the later is only 4:1 [9]. By using DLP(8-way SIMD) and ILP(6-way VLIW), the 48 computing ALU can keep busy and many memory accesses are reduced.
- Imagine has software controlled SRF. Locality in kernel and producer-consumer locality between kernels can be caught by Programmers and compiler. GPP's cache can't do this job well.
- The overlap of computing and memory access can be well achieved in Imagine. By scheduling stream loads and stores to hide latency, the Imagine memory system can be designed to provide the average bandwidth required by applications without loss of performance. In contrast, GPP are highly sensitive to memory latency and hence providing memory bandwidth well in excess of the average is required to avoid serialization latency on each memory access. Although the memory system of a conventional processor is idle much of the time, reducing its bandwidth would increase memory latency and hence increase execution time.

However, when an application needs to randomly access an array (larger than SP), implementing it on Imagine will be a problem, and the performance is relatively lower. The 8 clusters in Imagine all controlled by the microcontroller, though which they communicate with each other. One cluster can't access other clusters' data directly. Just like Blowfish, when the array's size is larger than SP's, it must be distributed among 8 clusters. This increases programming complexity and decreases performance. Therefore, we can conclude that whether an application is well suit for Imagine depends on:

- Is there data level parallelism in the application? Or can data elements be processed in parallel?
- Do we need to randomly access an array in the kernel? If yes, can we put a copy of it in every cluster?

4. Conclusion

In this paper, we implement Blowfish, Rijndael, RC5, and Base64 on Imagine. We compare the performance of Imagine implementations and GPP implementations. By analyzing the experiment results and characteristics of applications, we propose a checking model for choosing application which can get better performance on Imagine: 1) it has plenty much of available data level parallelism; 2) it doesn't need to randomly access an array, or it needs to, but the array's size is smaller than SP's. For these applications, good performance is easily to achieve.

In information security area, symmetric ciphers are widely used. Imagine achieves good performance in this area while keeps the programmability. But most symmetric ciphers are involved with random accesses to an array. Small SP in Imagine put a limit on the array's size and therefore some applications can't be implemented on Imagine. This limit is eliminated in Merrimac which has a larger SP and supports random access SRF. Evaluating the performance of symmetric ciphers on Merrimac [10] would be our next work.

5. Acknowledgements

This research was supported financially by the National Basic Research Program of China under contract 2005CB321601, the Natural Science Foundation of China grant 60633040 and 60736012, the National Hi-tech Research and Development Program of China under contract 2006AA01A102-5-2 and 2009AA01Z106, the China Ministry of Education & Intel Special Research Foundation for Information Technology under contract MOE-INTEL-08-07.

6. References

[1] Ujval J. Kapasi, William J. Dally, Scott Rixner, et al. The Imagine Stream Processor. In: *Proceedings of the 2002 International Conference on Computer Design,* September. 2002, pp.16-18.

[2] Peter Mattson. A Programming System for the Imagine Media Processor. Thesis, Dept. of Electrical Engineering, Stanford University, 2001

[3] John D.Owens, Scott Rixner, Ujval J.Kapasi. Media Processing Application on the Imagine Stream Processor. In: *Proceedings of the 2002 International Conference on Computer Design*. September 16-18. 2002, Freiburg, Germany, pp. 295-302.

[4] William Stallings. Cryptography and Network Security. Nov. 26. 2005, pp. 102-172.

[5] Peter Mattson, Ujval Kapasi, John Owens. Imagine Programming System Developer's Guide. April 3. 2002.

[6] R. Baldwin, R. Rivest. RFC2040 ——The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms. October 1996.

[7] N. Freed, N.Borenstein. RFC2045——Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. November 1996.

[8] Jung Ho Ahn, William J. Dally, Brucek Khailany, et al. Evaluating the Imagine Stream Architecture. In: *Proceedings Of the 31st Annual International Symposium on computer architecture*. June. 2004, pp. 14- 25.

[9] D. Sager, et al. A 0.18μm CMOS IA32 microprocessor with a 4GHz integer execution unit, In *2001 IEEE International Solid-State Circuits Conference Digest of Technical Papers*. February 2001, pp. 324–325.

[10] William J. Dally, et al. Merrimac: Supercomputing with Streams. In: *Proceeding of the SC 2003*. ACM Press, Nov. 2003, pp.15-21