Adding Security to Network via Network Processors

Hao Yin[1], Zhangxi Tan[1], Chuang Lin[1], Guangxi Zhu[2]

[1] Department of Computer Science and Technology, Tsinghua University, China,
{hyin, xtan, clin}@csnet1.cs.tsinghua.edu.cn
[2] Department of Electronic & Information Engineering, Huazhong University of Sci.&Tech, China,
{gxzhu}@mail.hust.edu.cn

Abstract. With the increasing need of security, cryptographic processing becomes a crucial issue for network devices. Traditionally security functions are implemented with Application Specific Integrated Circuit (ASIC) or General-Purposed Processors (GPPs). Network processors (NPs) are emerging as a programmable alternative to the conventional solutions to deliver high performance and flexibility at moderate cost. This work compares and analyzes architectural characteristics of many widespread cryptographic algorithms on Intel IXP2800 network processor. In addition, we investigate several implementation and optimization principles that can improve the cryptographic performance on NPs. Most of the results reported here should be applicable to other network processors since they have similar components and architectures.

1 Introduction

Information security is an indispensable concern owing to the growing demands for trusted communication and electronic commerce. For example, a collection of applications such as secure IP (IPSEC) and virtual private networks (VPNs) has been widely deployed in nodal processing. However, cryptographic algorithms are all computational intensive [1]. To address this problem and add security functions to network equipment, such as secure gateway, a straightforward approach to achieve comparable performance is to implement them in hardware. Unfortunately, many security chips or coprocessors are only designed for a few algorithms, while most Internet security standards are written to allow flexible in algorithm selection. In addition, cryptographic hardware is not cheap or readily exportable.

On the other hand, Network processors (NPs) are an emerging class of programmable processors used as a building block for implementing packet processing applications such as switches and routers [2]. They are highly optimized for packet processing and I/O operations. As demands for communication security grow, cryptographic processing becomes another type of application domain. Currently, there are two approaches to add security into NPs: 1) Security functionality is directly built into the same silicon as the Network Process. Nevertheless, this method is still inflexible in implementation of multiple algorithms. 2) Implement cryptographic ap-

plications on NPs using software, which provides a good trade-off between performance and flexibility. Compared to other similar approaches, such as software implementation over general-purposed processors (GPPs), this approach has the following advantages: a) NPs utilize system-on-chip (SoC) technology and have better performance-price ratio than GPPs. b) NPs often involve multi-thread and multi-core architecture, thus various parallelism can be exploited to boost performance. So, in this paper, we focus on software implementation over NPs.

Clearly, the most challenging work for software implementations is to provide some performance guarantees. Most of recent studies [2][3] related with cryptographic issues on NPs assume a symmetric multiprocessor (SMP) or super scalar architecture with multi-level caches, which are more similar with GPPs and ignore many characteristics like hardware multi-thread, asynchronous I/O in real-life NPs. This work aims to conduct studies of architectural properties of several widespread cryptographic algorithms on an actual platform - Intel IXP2800 network processor. Their implementation and optimization principles have been proposed. The rest of this article is organized as follows. In section 2 we briefly review the architecture of IXP2800. Then, we detail the selection of cryptographic algorithms and their characteristics in section 3. Next, we propose several optimization principles and illustrate the results through benchmarks in section 4. Finally, we summarize this work and offer some suggestions for network processor designs.

2 Architecture of Intel IXP2800

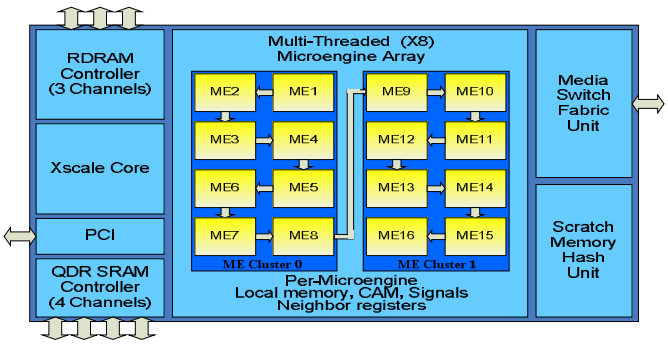

Fig. 1. The hardware architecture of Intel IXP2800

Closely examining the hardware architecture of IXP2800 shown in Fig. 1 helps to elucidate our implementation and optimization. IXP2800 is a 32-bit RISC based multi-core system that exploits system-on-chip (SOC) technique for deep packet inspection, traffic management and forwarding at high speed. The 700 Mhz XScale core is a general purpose processor used for control plane tasks (slow-path processing). The sixteen 1.4 Ghz microengines (MEs) are data plane PEs, which are connected in two clusters. IXP2800 has distributed, shared memory hierarchy which supports two types of external memory: RDRAM and QDR SRAM. In addition, the

processor includes a 16KB on-chip Scratch SRAM shared among all MEs and plenty of registers in conjunction with a small amount of local memory per ME. Memory access latencies have not kept pace with ME processing speed. For instance, the minimum read latency for fastest shared SRAM (Scratch) is 100 ME cycles. To solve this problem, IXP architecture uses 8 "zero thread switching overhead" hardware threads for interleave operation - one thread does computation while others block waiting for memory operations to complete.

3 Selection of Cryptographic Algorithms

There are three such application domains for cryptographic processing: Public-key ciphers, Private-key ciphers and Hash functions. In this article, public-key ciphers are not studied, since many of them are not practically applicable to be implemented on fast-path of NP. First, large code storage is required. Besides, public-key ciphers are usually used for short sessions and private key managements, while private-key ciphers are critical for long session performance. Therefore, we will only focus our effort on private-key ciphers and hash functions. The former can be further classified into block ciphers and stream ciphers. Of the many algorithms, we select a subset of 10 algorithms based on their representativeness, popularity and availability. The summaries and characteristics of these algorithms are presented in Table 1.

Table 1. Selection and characteristics of cryptographic algorithms

Type	Name	Block Size (bits)	Round	Table Size (bytes)	Special Requirements	Description or Applications
Block Cipher	DES [4]	64	16	256		The first commercial -grade modern cipher
	AES [5]	128	10	5120		802.11i
	IDEA [6]	64	9	0	Multiply Unit	PGP, SSH / SSL
	RC5 [7]	64	16	136	32-bit variable rotation engine	Wireless Transport Layer Security in WAP
	RC6 [8]	128	20	176	Multiply Unit, 32-bit variable rotation engine	AES candidate, improved version of RC5
	Blowfish [9]	64	16	4168		Norton Utilities
Stream Cipher	RC4 [10]	-	-	256		SSL / TSL, 802.1x
	SEAL [11]	-	64+2	<4096[a]		Disk encryption
Hash Function	MD5 [12]	512	64	0		Digital Signature
	SHA-1 [13]	512	80	0		

[a] ***The table size of SEAL is variable concernina the output lenath. Here lists the upper bound.***

4 Optimization and Benchmark

4.1 Methodology

To observe architectural characteristics of cryptographic algorithms and utilization of internal re-sources, as well as the performance bottlenecks, we conduct our ex-

periments under Workbench 3.1, which is a cycle-accurate simulator of IXP2800. We configure MEs working at 1.4 Ghz, SRAM at 200 Mhz and RDRAM at 400 Mhz. All the source codes are compiled using Intel Microengine C compiler 3.1 with optimization level -O2 enabled. When encounter operations such as rotation that can not be directly expressed using C operators but supported by instructions of IXP, we implement them with inline assembly codes. Hence, our optimization principles do not focus on specific instructions unique to one target but general features applicable to a wide range of NPs. In addition, to test the scalability of parallel optimization we exploit up to 8 MEs (64 threads in total) in one ME cluster as described earlier.

4.2 Instruction Characteristics

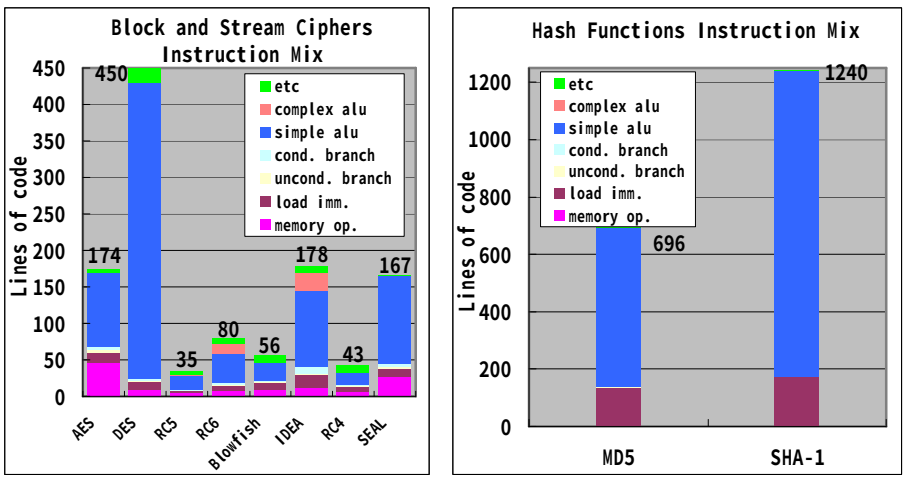

Fig. 2. Raw code sides and instruction mix

In this section we present the experimental statistics on instruction distribution of these algorithms. These metrics are essential information for understanding their dynamic properties and developing implementation and optimization principles. Figure 2 illustrates the instruction mix profile and code size of all selected algorithms. The following gives indications on their instruction patterns and great differences:

- Most block ciphers and stream ciphers need small code storages (less then 200 lines of code). The only exception is DES because it has several complex bit operations. Hash functions usually need more code storage.
- The most frequently used instructions are ALU instructions, especially simple ALU instructions like add, shift and logic. As a whole, ALU instructions occupy a significant share of the total instruction mix, which is 79.9% on average
- Branch instructions are less used in every algorithm. The average percent is 1.5% (0.8% for unconditional branch and 0.7% for conditional branch).
- For memory and load immediate instructions, there are significant differences among all selected algorithms. Stream ciphers and some block ciphers (AES

and Blowfish) tend to have a relative high percentage of memory instructions (exceeding 15%) than Hash functions. The average percent of memory instructions of the 10 algorithms is only 4%.

4.3 Optimization Principles and Benchmarks

We describe our optimization and benchmarks in two subsections. The first one focuses on general implementation and optimization principles for single thread within one ME. The second one considers multi-thread and performs scalability tests with multiple MEs.

Optimizations for single thread

Generic implementation and optimization rules

These rules are NP-independent and most of them are suitable for optimizations on GPPs. Their goal is to minimize the overall computation complexity and downcast expensive operations.

- Take full advantage of rich register resource and distributed memory hierarchy: To minimize access latencies, some frequently used tables should be placed into registers and per-ME local memories as much as possible.
- Avoid using complex instructions: Instructions like multiplication which consume more than one cycle of time should be avoided.
- Pre-calculate part of algorithms: Aside from table initialization and key scheduling mentioned earlier, immediate data used in inner loops can also be preloaded.
- Unroll loops: This can prevent the flush of pipeline and save extra clock cycles. Besides, unrolling loops can reduce calculations concerning iteration variables and make addressing in arrays more efficiently.

NP-dependent memory optimizations

These principles make use of special optimized memory and I/O units on NPs to increase ME utilization rate and stretch the computation capacity to the outmost.

- Align memory operations: Access to data size smaller than those supported by the hardware incurs overhead (i.e. table access in RC4 and DES). Thus, to achieve optimal performance tables should be aligned at hardware boundaries.
- Memory burst read and write: On most NPs, memory burst operations can be directly issued at instruction level. IXP2800 allows 32 bytes Scratch/SRAM or 64 bytes RDRAM burst reference within one instruction. Employing this, memory instructions will be further reduced. In our benchmark, reading plaintext and writing back ciphertext are all burst at their block size.
- Latency hiding and I/O parallelization: This makes use of asynchronous memory operations to ‘hide’ long memory access latencies and improve the ME utilization rate. The core idea is to continue calculations while ‘waiting’ for references to be completed. Further, with the mechanism of complete signals and command queues, multiple memory references can be issued simultaneously.

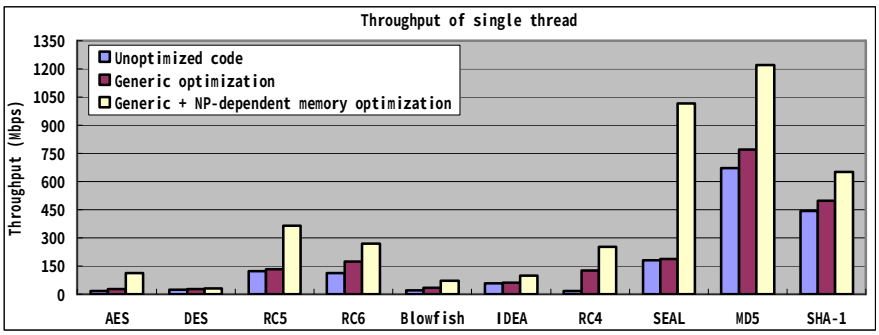

Fig. 3. Single thread performance with different optimization principles

Fig. 3 presents single thread throughputs of selected algorithms applying different optimization principles. Related internal statistics of ME are given in Fig. 4. As is evident from the plot, Hash functions have the best performance (MD5 1219 Mb/s) followed by stream ciphers and block ciphers. DES achieves the lowest throughput (32.2 Mb/s after optimization) because it works at bit level while 32-bit IXP2800 has a weak support on bit instructions.

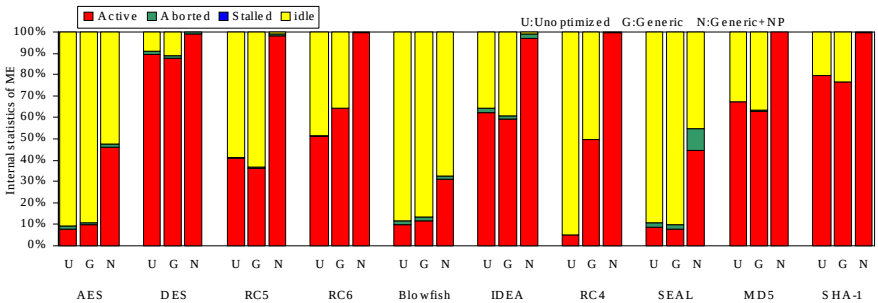

Fig. 4. Internal Statistics of ME

The effect of pipeline optimizations seems quite limited. This is because of the short pipeline architecture of NPs and low percentage of branch instructions (<2%) in cryptographic algorithms. The execution statistics also prove this. Most stream and block ciphers suffer from low ME utilization rate (‘active’ in Fig. 4), but generic optimizations do not take long memory reference latencies into account. On the other hand, NP-dependent memory optimizations effectively ‘hide’ them and increase ME utilization rate significantly, especially for algorithms which have a relative high percentage of memory operations. For instance, SEAL receives 438% performance The effect of pipeline optimizations seems quite limited. This is because of the short pipeline architecture of NPs and low percentage of branch instructions (<2%) in cryptographic algorithms. The execution statistics also prove this. Most stream and block ciphers suffer from low ME utilization rate (‘active’ in Fig. 4), but generic optimizations do not take long memory reference latencies into account. On the other hand, NP-dependent memory optimizations effectively ‘hide’ them and increase ME utilization rate significantly, especially for algorithms which have a relative high

percentage of memory operations. For instance, SEAL receives 438% performance boost after applying memory optimizations. Even though Hash functions have less than 1% memory instructions, memory optimizations still yield more speedup than generic optimizations. From Fig. 4, we also observe that all algorithms except AES, Blowfish and SEAL get a near 100% ME utilization rate after memory optimizations. Hence, ME computing power is still their bottleneck. On the contrary, not the memory bandwidth but long access latency limits the throughput of AES, Blowfish and SEAL. Because, none of tested algorithms has its ME 'stalled' owing to fullness of target memory queues or ME command queues.

Scalability test

An obvious way to improve the cryptographic applications on NPs is to use parallelism. Three types of parallelism can be used: flow-level, block-level and intra-block parallelism. We select flow-level and block-level parallelism to see how well the overall throughput scale using multiple threads and MEs of IXP2800. All block ciphers are implemented in Cipher Block Chaining (CBC) mode. When encrypted with CBC mode, block read/write operations can be paralleled, which are handled by single thread using I/O parallelization. Thus, we assign one hardware thread to one flow and no thread communication is required. Fig. 5 presents the overall throughputs of the selected algorithms with our multi-ME and multi-thread implementation.

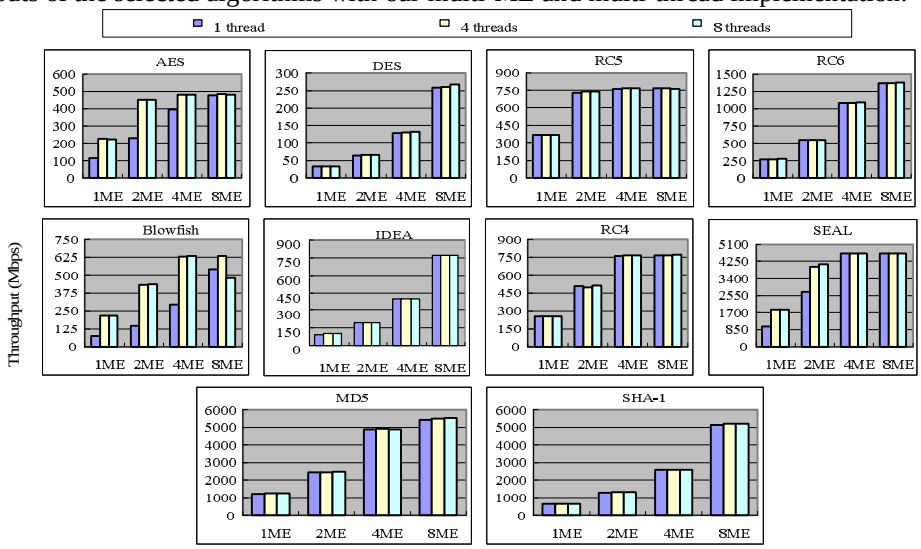

Fig. 5. Throughput of selected algorithms with varying number of threads and MEs

5 Summary

This study selects ten widely used cryptographic algorithms and analyze their instruction architectures on Intel IXP2800 network processor. We suggest several

hardware improvements can be made on current NPs to help 'software' implementations on data path PEs:

- Increase cache size on PEs to hold large tables and lessen the pressure on shared memory and bus.
- Enlarge the size of memory queue and command queue to reduce the 'stalled' possibility of PE.
- Improve communications among different PEs to help intra-block parallelism.
- Adopt a new memory system to shorten the access latency.

We believe that in combination of these improvements the proposed implementation and optimization principles could go a long way to improving cryptographic processing performance on network processors.

Acknowledgment

This research was supported by Intel IXA University Research Plan (No. 9077), the Natural Science Foundation of China (No.90104002, 60173012 and 60372019), the Projects of Development Plan of the State Key Fundamental Research (No.G1999032707) and the Projects of Development Plan of the State High Technology Research (No. 2001AA112080).

References

1. M. Merkow, J. Breithaupt, The Complete Guide to Internet Security, AMACOM, 2000
2. Haiyong Xie, Li Zhou, Laxmi Bhuyan, "Architectural Analysis of Cryptographic Applications for Network Processors", IEEE First Workshop on Network Processors, with HPCA-8, Boston, February 2002.
3. Praveen Dongara and T. N. Vijaykumar, "Accelerating Private-key cryptography via Multithreading on Sym-metric Multiprocessors", Proc. of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 58-69, March 2003.
4. US Government. Data Encryption Standard (DES), Triple DES, and Skipjack Algorithms. http://csrc.nist.gov/cryptval/des.htm.
5. Advanced Encryption Standard (AES) Development Effort, US Government, http://csrc.nist.gov/encryption/aes/
6. X. Lai, On the Design and Security of Block Ciphers, Hartung-Gorre Veerlag, 1992
7. R.L. Rivest, "The RC5 Encryption Algorithm", Proc. of the Second International Workshop on Fast Software Encryption, Springer-Verlag, pp. 86-96, 1995
8. R. Rivest, M. Robshaw, R. Sidney, Y. Yin, The RC6 block cipher, RSA Security, http://csrc.nist.gov/encryption/aes/round2/AESAlgs/RC6.
9. B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher", Proc. of the Cambridge Security Workshop, Springer-Verlag, pp. 191-204, 1994.
10. B. Schneier, Applied Crytography, 2nd Edition, John Wiley & Sons, 1996.
11. P. Rogaway and D. Coppersmith, "A Software-Optimized Encryption Algorithm", Proc. of the Cambridge Security Workshop, Springer-Verlag, pp. 56-63, 1994
12. R. Rivest,The MD5 Message-Digest Algorithm, RFC 1321, April 1992
13. Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. Handbook of Applied Cryptography, CRC Press 1996