NPCryptBench: A Cryptographic Benchmark Suite for Network Processors

Yao Yue and Chuang Lin
Department of Computer Science and Technology
Tsinghua University, China
{yueyao, clin}@csnet1.cs.tsinghua.edu.cn

Zhangxi Tan
Department of Electrical Engineering
and Computer Science
University of California, Berkeley, USA
xtan@cs.berkeley.edu

***Abstract*— Network processors (NPs), a type of multicore, multithread system that exploits system-on-chip techniques, usually adopt a distributed, shared memory hierarchy to scale up network processing. Network applications running on NP often display distinct characteristics on memory subsystem compared with traditional processors with multilevel cache. As security, especially cryptography, has become indispensable in today's networks, we present a benchmark suite called NPCryptBench. It is the first benchmark specially designed to evaluate cryptographic performance on NPs, particularly that of processor and memory subsystem. By running NPCryptBench on Intel IXP network processors, we provide quantitative evidence that performance bottlenecks under such a highly parallel environment reside in memory hierarchy and prolonged access latency, instead of limited computing resources described in recent literature. To alleviate this memory-wall effect, we suggest several software optimizations, which averagely boost the benchmark throughput by 145% on Intel IXP2800. Our analysis and optimizations are also suitable for other NPs with similar architectures.**

***Index Terms*— NPCryptBench, Performance Evaluation, Cryptographic algorithm, Network processor**

I. Introduction

Network processors (NPs) have become building blocks in nodal devices by providing a balance between performance and flexibility. Currently, link bandwidth explosion puts increasing demands on multicore System-on-Chip (SoC) NP architectures, which incorporate simple processing engines (PEs) equipped with little/no cache and application-specific instruction set. The memory subsystem in NP is more designed to store and forward packet streams than processor centric. These architectural features make NPs extremely vulnerable to the widening processor-memory gap, which continues to increase with each semiconductor generation. Besides, the additional memory latency typical of such multicore systems only exacerbates the problem. To alleviate the memory-wall effect, NPs often introduce distributed, shared memory hierarchy as well as multithread support by PE.

Because of these architectural traits, benchmarking NP turns out quite difficult. Many researchers suggest developing task-oriented benchmarks, which should combine NP architecture and application specialties [1]. Among the numerous network applications, there is growing interest in efficient cryptographic processing with the prevalence of secure network services, e.g. secure IP (IPSec) and virtual private networks (VPNs). Current hardware solutions like security coprocessors often fall short of flexibility requirements, and they also have scalability problems with shared resources. Therefore, it is still necessary to implement cryptographic processing in a software-based manner.

However, cryptographic processing on NP has not been well addressed yet. On one hand, there is little application-oriented tools and statistics. Cryptographic processing only takes a small portion in known NP benchmarks [2]–[5]. On the other hand, not all experience can be borrowed from traditional processors. Most previous studies assume a symmetric multiprocessor (SMP) or superscalar architecture with multilevel cache, and measure results using simulators for general-purpose processor (GPP). These studies ignore the distinct memory hierarchy of NP, and identify performance bottlenecks as the processor computing power and/or instruction set [6].

Here we present NPCryptBench, an NP-specific cryptographic benchmark suite. Contributions of our work are generally threefold. First, we introduce the first NP benchmark known so far that focuses on cryptography, and present its design methodology. Second, two ports of NPCryptBench have been utilized to evaluate cryptographic performance on two generations of Intel IXP network processors [7]. Quantitative results clearly show the memory-wall effect and its exacerbation in light of a widening processor-memory gap. Finally, we propose several optimizations to tune the benchmark. Their effectiveness has been demonstrated through cycle-accurate simulations.

Up to now, three versions of NPCryptBench have been developed, version 1.0/1.01 are for Intel IXP1200 and IXP2X00 respectively, and a third internal release is composed of hand-optimized codes adopted in the experiments. NPCryptBench can be freely used under an academic license (accessible from University of Wisconsin's WWW Computer Architecture Page[1]). A number of academic institutes have signed the license, and report research with our benchmark on simulators (NePSim [8]), compilers and cryptographic applications on network processors.

The rest of the paper is organized as follows. Section 2 provides a brief view of related work. Section 3 introduces the benchmark design methodology, from its target hardware

[1] `http://www.cs.wisc.edu/arch/www/`

architecture, algorithm selection to implementation considerations. Section 4 and section 5 present compile-time and run-time characteristics of NPCryptBench on Intel IXP network processors and analyze the performance bottleneck. Section 6 discusses optimizations and evaluates their effect on IXP2800. Section 7 concludes the paper.

II. Related Work

Several benchmark suites for NP have already been published. CommBench [2] classifies header and payload applications. NetBench [4] defines three different packet-processing granularities. A later benchmark suite, NPBench [3] mainly focuses on control plane applications. NP's prosperity also draws attention of embedded processor researchers. For example, the Embedded Microprocessor Benchmark Consortium (EEMBC) [9] has released header application benchmarks for NP, and allows source-level modification when necessary. MiBench [5] provides 9 free benchmarks related to security or network for embedded processors. Unlike NPCryptBench, most products tested in EEMBC together with all processor models in other benchmarks can be categorized as GPP-based. Furthermore, NPCryptBench's coverage of occurrences of cryptographic algorithms in popular security applications (listed in TABLE I) turns out much higher than that of other benchmarks. Hence it is more desirable when evaluating security network applications on NPs.

III. Benchmark Design Methodology

To design NPCryptBench, we extract common themes shared by heterogeneous NPs, select representative cryptographic algorithms and describe rules that we follow when implementing the benchmark on a proposed target platform.

A. Target NP Architecture

The design of benchmark rests on understanding the target platform. NPCryptBench assumes a multicore SoC NP architecture illustrated in Fig. 1:

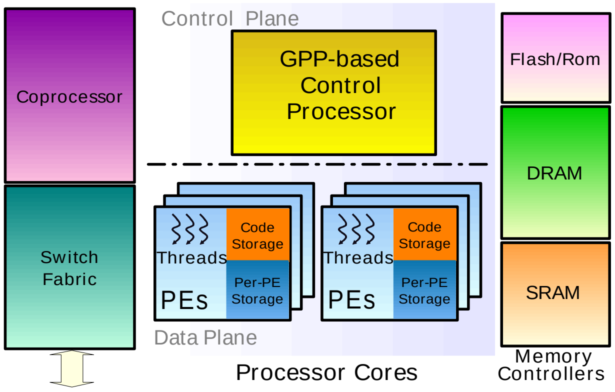

Fig. 1. Target NP Architecture

1) Layered Processor Model: According to requirements of network applications, processor cores within a network processor can be classified into two layers:

- *Control Plane Processor:* Most NP vendors build one GPP-based core on chip, (e.g. XScale core in Intel IXP2X00). It is typically used to perform non-realtime control plane tasks, like updating route tables and resource initialization.
- *Data Plane Processor:* This type of processors is usually based on RISC technologies and carry out realtime data plane tasks like package forwarding and data encryption. They are also called Processing Engines (PEs) in most literature [3] [10]. It is common to build multiple PEs within one NP and organize them as a pipeline (Cisco's PXF [11]) or in clusters (Intel IXP2X00).

2) PE Architecture: Compared to control plane processor, PEs are rather simple. They employ a short pipeline and small, separate local storage for data and instructions other than plenty of register resources. PEs are not built with advanced architectures like branch prediction, superscalar, out-of-order execution and multilevel cache. Instead, there are special function units and instruction set optimized for packet forwarding. Besides, some PEs introduce hardware multithreading to hide I/O and memory access latency.

3) Distributed, Shared Memory Hierarchy: To maximize performance at reasonable cost, designers adopt memories with various speeds and sizes. These memories can have either different or same address space and are shared by all processor cores on the chip. While such hierarchy offers flexibility of freely assigning data to different memories, it also introduces difficulty in efficient management.

B. Algorithm Selection

Security primitives used in practice generally fall into three categories: unkeyed, symmetric-key and asymmetric-key. NPCryptBench focuses on symmetric-key ciphers and unkeyed hash functions, taking ten algorithms based on their representativeness, popularity and availability. TABLE I samples popular security applications in both wired and wireless networks. Currently, we do not address asymmetric-key ciphers because certain implementation shows they are 3 magnitude slower than symmetric-key ciphers [12], too computationally expensive for data plane processors.

TABLE I
Popular Security Applications and NPCryptBench's Coverage of Cryptographic Algorithms Used in Theses Applications

Application	Introduction	Algorithm Coverage
openPGP	Privacy and authentication, (RFC 2440)	5/8
S/MIME	Similar to openPGP, (RFC 3370)	3/3
SSH	Encrypted login and communication	6/7
SSL/TLS	Transport Layer Security, (RFC 2246)	6/7
IPSec[a]	IP-level security, (RFC 2401-2412)	3/3
WEP	Data encryption in IEEE 802.11	2/2
WPA2	Implements the majority of IEEE 802.11i	2/2
WTLS	Wireless Transport Layer security	5/5
Overall Coverage		10/15

[a] IPSec supports customized ciphers

NPCryptBench contains two hash functions, MD5 [13] and SHA-1 [14]. Hash functions produce an output of constant length (called digest) from arbitrary length of input. The bulk of the package belongs to symmetric-key ciphers, which can be further classified into two categories. RC4 [15] and SEAL [16] represent stream ciphers, which randomly generate key streams to ”XOR” with plaintext of variable length. AES [17], DES [18], RC5 [19], RC6 [20], Blowfish [21] and IDEA [22] represent block ciphers, which transform a fixed-length block of plaintext into same-length ciphertext.

Little data storage is needed for the unkeyed hash functions. In contrast, ciphers usually require more memory for keys and lookup tables (S/P-boxes). Some ciphers (RC5, RC6 and IDEA) obviate large tables. However, they introduce operations like modular multiplication (RC6, IDEA) or variable bit rotation (RC5, RC6), which are not directly supported by all NP models. As a summary, TABLE II lists characteristics of these algorithms.

TABLE II

ALGORITHM CHARACTERISTICS OF NPCRYPTBENCH

Type	Name	Block Size (bits)	Round	Key Size (bytes)	Table Size (bytes)
Block Cipher	AES	128	10	16,24,32	4096
	DES	64	16	7	512
	RC5 [†]	32,**64**,128	0-255,**16**	0-255,**16**	0
	RC6 [†‡]	128	20	**16**,24,32	0
	Blowfish	64	16	4-56	4096
	IDEA [‡]	64	9	16	0
Stream Cipher	RC4	-	-	1-256,**8**	256
	SEAL	-	-	20	<4096[a]
Hash Function	MD5	512	64	0	0
	SHA-1	512	80	0	0

NPCryptBench adopts recommended values (in bold) for variable parameters

[†] require 32-bit variable bit rotation

[‡] require multiplication

[a] here lists the upper bound

C. Implementation Considerations

The main considerations during implementing NPCryptBench are listed as follows:

1) Algorithm Kernels and Coding Convention: NPCryptBench only implements the critical inner loops of cryptographic algorithms, where the processing majority lies. Other parts like initialization are carried out by control plane processors. To ease programming, many NPs can translate applications written in high-level language with their own compilers, which support some general-purpose language and extra hardware-related intrinsic functions. On these platforms, the C style codes of NPCryptBench can be easily ported with limited modification.

2) Modification: Modification helps adjust benchmarks to heterogeneous NP architectures. NPCryptBench allows the following two types of modification:

- Assembly: Codes can incorporate assembly, when the operation is included in the instruction set but not efficiently represented with high-level language, e.g. bit rotation.
- Intrinsic Function: Hardware-oriented or vendor-defined functions, such as I/O and memory access, can be used in accordance with specific platforms.

3) Optimization: In NP world, benchmarking often helps develop optimizing techniques and test their effectiveness. Our previous works [23], [24] discuss optimizations of cryptographic processing on certain platforms, which enable us to introduce four levels of hand-optimizations that could be applied to our benchmark.

- Level-0: Raw implementation without optimizations.
- Level-1: Generic, NP-independent optimizations also available on traditional processors.
- Level-2: NP-dependent optimizations together with those of level-1.
- Level-3: Optimizations for a massive parallel environment plus those of level-2.

A full description of these optimizations is provided in section VI-B and section VI-D. Depending on these hand-optimizations, codes of NPCryptBench are classified into 4 levels correspondingly.

4) Parallelism: With network bandwidth well outpacing silicon speed and memory lagging behind processors, many NPs rely on parallel techniques like multi-PE and multithread to meet performance requirements. NPCryptBench supports benchmarking multi-PE and multithread performance by including thread/PE control instructions into the source.

IV. COMPILE-TIME CHARACTERISTICS OF NPCRYPTBENCH

A. Target Platform and Experiment Environment

As a particular implementation, NPCryptBench is applied to two generations of Intel IXP network processors, IXP1200 and IXP2400/2800. Their processing engines are called micro-engines (MEs) by Intel. MEs are scaled-down, cacheless RISC cores which support hardware multithreading. TABLE III lists processor/memory parameters of IXP1200/2X00. All three models incorporate two types of external memory, DRAM and SRAM. DRAM is large in size and used as mass packet buffers. SRAM is for faster access of small data structures (e.g. IP header). In addition, there exists an on-chip Scratch SRAM shared among all MEs for low latency access. Compared with IXP1200, IXP2X00 is equipped with an extended instruction set and additional on chip memories. IXP2400 and IXP2800 are architecturally similar except the latter has more MEs, higher working frequency and faster memory modules.

TABLE III

MEMORY AND PROCESSOR CHARACTERISTICS OF INTEL IXP NETWORK PROCESSORS

	Type	Tran. Size	Memory Access Latency		
memory	DRAM	16 bytes	48/36 100MHz	119/53 300MHz	291/54 400MHz
	SRAM	4 bytes	22/19 100MHz	90/48 200MHz	100/54 200MHz
	Scratch	4 bytes	20/20	59/48	56/40
	LM	4 bytes	-	1/1	1/1
processor	number of ME		6	8	16
	thread number per ME		4	8	8
	ME frequency		200MHz	600MHz	1400MHz
Network Processor model			IXP1200	IXP2400	IXP2800

Read/Write memory access latencies are measured in ME cycles, with SRAM and DRAM bus frequencies listed below.

As shown in TABLE III, the evolution of Intel IXP network processors leads to a quickly widening processor-memory performance gap. An over 50% growth of relative latency is caught between the two generations. In particular, DRAM read latencies (in ME cycles) of IXP2400 and 2800 are 248% and 606% that of IXP1200. To compensate for the processor-memory gap, IXP2X00 integrates fast internal storage called *local memory (LM)* into each ME, which can be accessed at almost the same speed as per-ME registers.

In the following experiments, ME, SRAM and DRAM are configured with parameters in TABLE III. NPCryptBench 1.00/1.01 (level-0 codes) are used on Intel IXP1200/2X00 respectively, and all codes are written in Intel Microengine C. In our experiments, 4 versions of Intel IXA SDK (integrating Intel Microengine C compiler and cycle-accurate simulator) are utilized. Among them, SDK 2.01 is for IXP1200, while SDK 3.1, 3.5 and 4.1 are for IXP2X00. Three compile optimization options, -O2 (optimized for speed), -O1 (optimized for code size) and -O0 (for debug, no optimization), are available. Without explicit mention, codes in this paper are generated by SDK 2.01 for IXP1200 and SDK 4.1 for IXP2X00 with -O2 compiler option.

B. Data Storage

TABLE IV

MEMORY USAGE OF NPCRYPTBENCH ON INTEL IXP NETWORK PROCESSORS (IN BYTES)

Algorithms	IXP1200		IXP2X00	
	Scratch	SRAM	local memory	Scratch
AES	176	5120	176	5120
DES	2176	0	2176	0
RC5	136	0	136	0
RC6	176	0	176	0
Blowfish	72	4096	72	4096
IDEA	208	0	208	0
RC4	0	1024	1024	0
SEAL[a]	0	<4096	4	<4096
MD5	0	0	0	0
SHA-1	0	0	0	0
Memory Capacity	4096	8M	2560	16384

1 Plaintext/Ciphertext are stored in DRAM.
a SEAL uses SRAM to store generated keys

Ciphers need to store their control data, i.e. subkeys and tables (if any). TABLE IV shows the memory occupation of each benchmark algorithm. For the majority of ciphers in NPCryptBench, control data are kept in the fastest memory provided. For example, when local memory becomes available on IXP2X00, 5 ciphers fully localize their control data and merely visit DRAM to fetch plaintext and write back ciphertext. However, the capacity ceilings force 4 algorithms on IXP1200 and 3 on IXP2X00 to arrange tables in second fastest memory.

C. Code Storage and Instruction Mix

Ciphers in NPCryptBench need only small code storage, usually less than 250 lines. DES has lengthier codes than other ciphers for it has to perform some complex bit operations. Compared with ciphers, hash functions need more code storage[2]. All algorithms (except Blowfish, see section VI-A) decrease in code size from IXP1200 to IXP2X00, owing to a richer PE instruction set and reduced initialization directives.

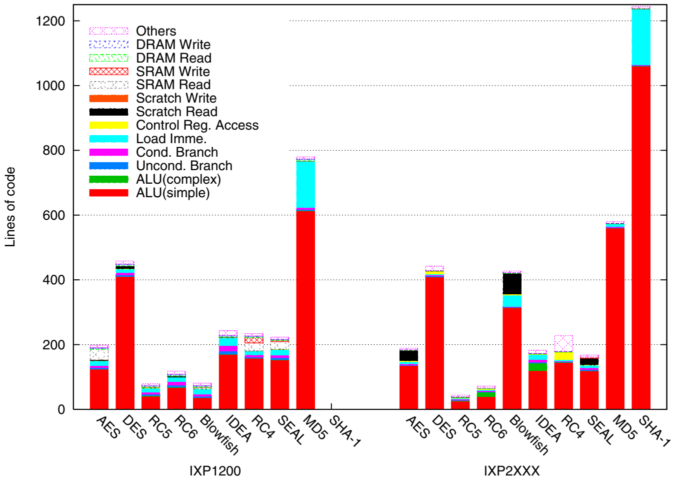

Fig. 2. Static Instruction Mix of NPCryptBench on Intel IXP Network Processors

TABLE V

COMPARISON OF AVERAGE DYNAMIC INSTRUCTION MIX

Instruction Type	**NPCryptBench 1.01**	**CommBench (PPA)** [a]	**CommBench (HPA)** [a]	**NPBench**	**NetBench**
ALU(simple)	78.9	58	41	53.5	60
ALU(complex)	5.9				
Uncond. Branch	0.1	1	1	16.2	3.7
Cond. Branch	1.9	13	18		7.2
Immediate	3.3	1	2	N/A	N/A
Mem. Load	7.0	18	27	19.3	27.7
Mem. Store	0.5	8	6	8.9	
other	2.4	1	5	2.2	1.4
total	100	100	100	100	100
Code Size	359	3430(all) 500(99%time)		N/A	359

[a] CommBench has two groups of applications: Payload Processing Applications (PPA) and Header Processing Applications (HPA)

Fig. 2 illustrates static instruction mix of NPCryptBench on three IXP platforms, and TABLE V compares the dynamic mix of NPCryptBench 1.01 with other popular NP benchmarks. 71.5% (IXP1200), 78.9% (IXP2X00) instructions executed are simple ALU, like add, shift and logic. On IXP2X00, complex ALU instructions like multiplication or variable bit rotation add up to 5.9% in the overall statistics. Another 2.0% is taken by branch instructions. On IXP1200 where multiplication is done by software, these two percentages are 1.2% and 10.1% respectively. In contrast to other benchmarks in TABLE V which report at least 10% branch and at most 60% ALU instructions, our results indicate that cryptographic algorithms are more computationally intensive and sparse in branch.

Another noticed phenomenon is that cryptographic algorithms do not rely heavily on memory and load-immediate instructions. The IXP1200 version contains no more than 12.0% such instructions on average. Later on IXP2X00, the

[2] Because of compiler problems, SHA-1 cannot be compiled on IXP1200.

number is further reduced to 10.8% by taking advantage of the richer register resources and local memory. Consequently, no algorithm reaches the average portion in CommBench (29.5%), NPBench (28.2%) or NetBench (27.7%). Also on IXP2X00, algorithms' portion of memory and load-immediate instructions varies greatly, from 0.2% (MD5) to 24.5% (AES).

V. Run-time Characteristics of NPCryptBench

A. Throughput Test

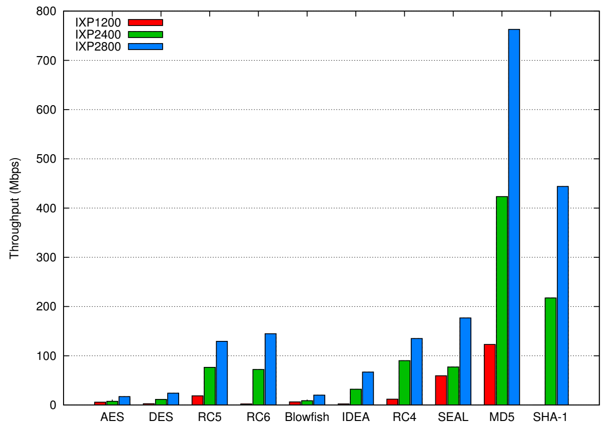

Fig. 3. Benchmark Throughput on Intel IXP Network Processors

To provide basic measure of cryptographic performance on two generations of Intel IXP network processors, throughput of NPCryptBench (Level-0 codes) has been collected under single-thread/single-ME configuration. From Fig. 3 we find that throughput of different algorithms varies greatly, as the fastest MD5 outperforms the slowest RC6 (on IXP1200)/AES (on IXP2X00) by 45 times or more. Unkeyed hash functions run much faster than ciphers, twice at least. Within encrypting algorithms, stream ciphers perform better than block ciphers. On IXP1200, RC6 and IDEA are the slowest two algorithms because they lack hardware support for multiplication. But on IXP2X00, it is cipher storing large tables in external memory that falls behind others, such as AES.

Throughput results on Intel IXP network processors depart from those obtained on GPP. For example, AES outperforms RC6, Blowfish and IDEA on an Alpha 21264 workstation [6], while on IXP2X00 it has the least throughput. Some NP benchmarks like CommBench [2] and NPBench [3] calculate instructions per-byte/packet to estimate performance, but it fails to explain the results in Fig. 3. On IXP2X00, DES has more instructions per byte yet still generates higher throughput than AES and Blowfish.

Although cryptographic processing is believed to be computationally intensive, benchmark throughput cannot always keep pace with the elevation of ME frequency if not for some architectural improvements. ME frequency increases 133% from IXP2400 to IXP2800, but only SEAL produces 1.33 times more throughput. All other algorithms fall behind the speedup of ME. This comparison raises the question that how well ME is utilized during cryptographic processing.

B. Pipeline Statistics

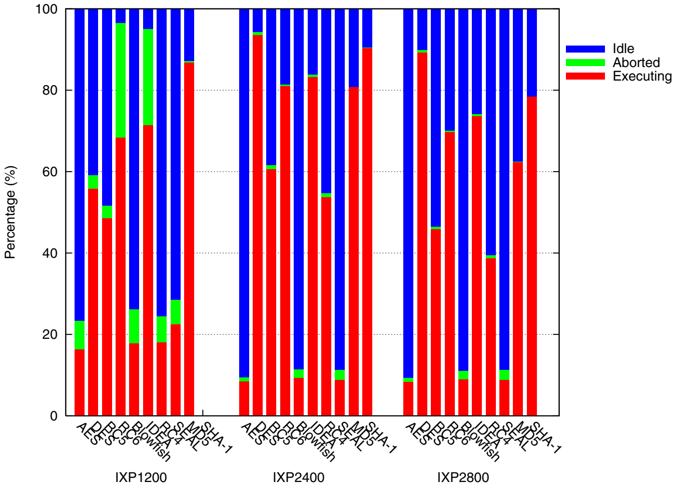

Fig. 4. Pipeline Statistics for Intel IXP Network Processors

Pipeline statistics characterize the utilization of ME. According to Fig. 4, ME's executing percentage on three platforms never exceeds 60%. ME remains underutilized for almost all algorithms as it is often aborted or idling. Two architectural improvements on IXP2X00, a richer ME instruction set and local memory, help reduce aborted state caused by branch penalty or external memory access. As a result, ME pipeline on IXP2X00 is rarely aborted. On the other hand, idle state has always been common on these three models, average portions of the 10 benchmarks are over 42%. When running AES, RC4/5/6, Blowfish and SEAL on IXP2800, ME idles over half the time, waiting for memory operations to complete. From IXP1200 to IXP2X00, 6 algorithms increase ME's idle time. Moreover, from IXP2400 to IXP2800 ME's idle time goes through a universal rise because of the widening processor-memory gap.

C. Execution Cycle Breakdown

Cycle breakdown clarifies time consumed by each type of instructions. With these results we can discern which instructions are creating the bottleneck. Fig. 5 depicts time portion occupied by each type of instructions on IXP1200/2800. RC6 and IDEA on IXP1200 spend around 50% time on branch instructions (to accomplish multiplication in a software manner), while other benchmarks spend at most 4% of total execution time on them. ALU instructions have a considerable time portion, however, they contribute less than memory operations in overall statistics. In addition, load-immediate and other instructions that occupy 10.5% of static code storage use only 2.7% execution time, because they mainly reside outside the loop kernel.

The major difference between static/dynamic instruction mix and execution cycle breakdown is the abrupt rise of memory instructions, which averagely occupy over 50% execution time on either platform. All cryptographic algorithms need memory operations to fetch plaintext and write back digest

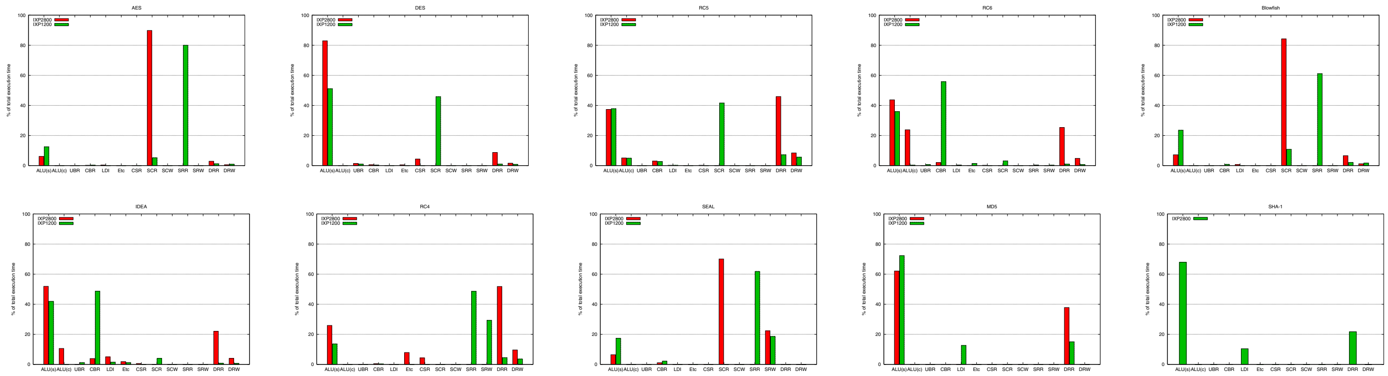

Fig. 5. Execution Cycle Breakdown of NPCryptBench on IXP1200/2800

or ciphertext. What is more, ciphers involve frequent access to subkeys and/or tables. Compared with ALU and branch instructions which cost only one or a few cycles, memory instructions have 1 or 2 magnitude longer latencies (TABLE III). Thus despite their small code storage, memory instructions become the dominant group on the fly. On IXP2800 where ME idles most, even the table-free hash functions spend at least 21.7% of total execution time on memory instructions. On the other hand, AES, Blowfish and SEAL leave over 70% ME computing power unused because of their frequent memory access. This analysis agrees with the results from pipeline statistics in section IV-B.

Study of cryptographic processing on GPP claims processor computing resources as the major bottleneck [6]. This is possible on older generation of NPs. For example, lacking hardware support on multiplication hinders the performance of RC6 and IDEA on IXP1200. However, this problem has been readily tackled by a richer instruction set on newer NPs. From analysis above, the real bottleneck on NP is memory access. With a growing speed disparity among processors and memories, this bottleneck will become more substantial in the future. That means simply increasing ME's computing power cannot produce proportional throughput lift. To better improve cryptographic performance, optimizations mainly attacking this memory-wall effect should be developed.

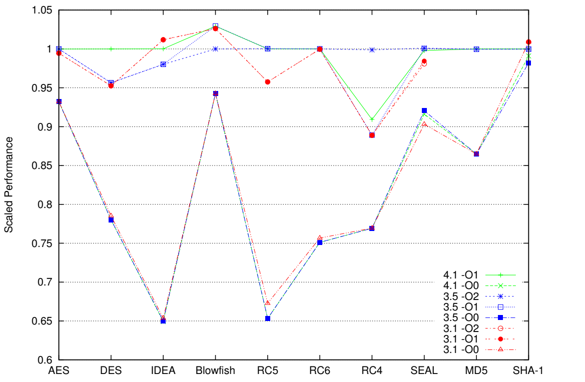

Fig. 6. Comparison of IXP2800 Compilers - Benchmark Throughput

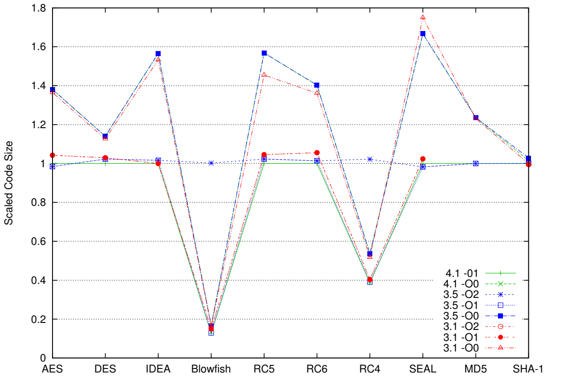

Fig. 7. Comparison of IXP2800 Compilers - Code Size

VI. Optimization and Parallelism

A. *Effect of Compiler Optimization*

In the previous section, the memory-wall effect hampers throughput even for compiler-optimized codes. To find out performance improvements provided by compilers and their ability in addressing the memory-wall effect, we carry out experiments on IXP2800 enumerating 3 compiler versions and 3 optimization options. Scaled throughput and code size are demonstrated in Fig. 6 and Fig. 7 respectively, with results under default settings (section IV-A) as baseline.

Several conclusions can be derived directly from the figures. First, compiler optimization is necessary for cryptographic processing, especially for saving code storage. Codes generated with option -O0 are about 17% slower than -O2 codes and 65% lengthier than -O1 codes. Second, codes generated by different versions of compilers (with option -O1 and -O2) have similar size and performance. Newer compilers introduce generic optimizations like loop-unrolling and lead to much larger code size. However, they are not very effective and even slow down some algorithm, e.g. Blowfish.

By reading compiled codes we find that optimizations performed by compilers are incapable of tasks such as contextaware instruction reordering, table splitting and read/write block size adjusting. In addition, we encountered some compiler failures. Compiler 2.01 fails on SHA-1 for IXP1200, and compiling SEAL for IXP2X00 takes an hour on a Pentium 4 2.4G processor. Current research of NP compilers is still in its infancy, and mainly focuses on correctness rather than performance. Therefore, it is worthwhile to study hand-optimizations that better adapt cryptographic processing to an NP architecture. The development of hand-optimizing strategies in NPCryptBench can also help improve future NP compilers.

B. *Hand-optimizations For Single-thread*

Lacking compiler countermeasures, we put forward hand-optimizations to address the bottleneck discovered earlier. Techniques alleviating the memory-wall effect under single-thread mode are adopted with highest priority among optimizations mentioned in section III-C-3.

Memory-related optimization of level-1 suggests greedy memory allocation of subkey and table storage. This could be achieved through splitting large tables and taking advantage of ”unused” registers. Other level-1 optimizations include loop-unrolling, preloading immediate into registers (precalculation) and replacing multiplication with shift operation.

On level-2, certain features of the target platform enable extra memory optimizations, like reordering memory instructions. On Intel IXP network processors and most other NPs, with the mechanism of complete signals and command queues, the PE can issue multiple memory references simultaneously and keep calculating during the waiting. Level-2 optimizations also incorporate aligning data in memory and using special hardware units when available.

C. *Scaling Up Performance with Parallelism*

Besides optimizing single-thread codes, two distinct features of NP can be further used to scale up cryptographic throughput with parallelism. Multi-PE multiplies NP’s computing power, and multithread boosts PE’s utilization rate. Both techniques increase parallelism and the throughput of memory subsystem. Here we evaluate performance of level-2 codes with flow-level and block-level parallelism [23]. Throughput of the benchmark on IXP2800 is gathered on Intel IXA SDK 3.1 (-O2 option), with various numbers of PE and thread.

Fig. 8 presents the throughput attained in the experiment. After applying the hand-optimizations described in section VI-B, performance of single-thread codes remarkably improves for most algorithms. Actually an average of 145% increase is observed, easily surpassing the compiler optimization. However, with multiplied ME/thread number, new bottlenecks occur.

For algorithms that already achieve a near 100 percent ME utilization rate under single-thread mode, multiple threads cannot increase the overall throughput. Algorithms with frequent memory references like AES, Blowfish and SEAL benefit from multithread when ME number is small (one or two), because their memory access latency could be sufficiently hidden. However, throughput stops growing linearly for all algorithms from 4 threads to 8 threads, as ME’s computing power has already saturated under the 4-thread configuration.

When ME number increases from one to two, all algorithms present nearly doubled throughput. But performance growth begins to diverge with more MEs. Only DES, IDEA and SHA-1 maintain a near linear speedup with 8 MEs running concurrently. On the contrary, AES and RC5 hardly gain any performance boost from greater than 4 MEs. In this case, memory requests and internal traffic increase with the growth of ME number, so shared memory or buses become the new bottlenecks.

D. *Optimizing for Parallelism*

On IXP2800, abundant memory requests initiated by many thread/ME make algorithms less scalable under massive parallel environment. To mitigate congestion at memory and shared buses, level-3 optimizations maximize memory read/write burst size therefore reduce overall memory access overhead. It is noticed that the two hash functions already hit the ceiling of burst size on Intel IXP network processors, thus cannot enjoy further increase.

To demonstrate the effectiveness of the level-3 optimizations, we compare it with an increase in memory frequency (16.5% increase for DRAM and 33.5% increase for SRAM).

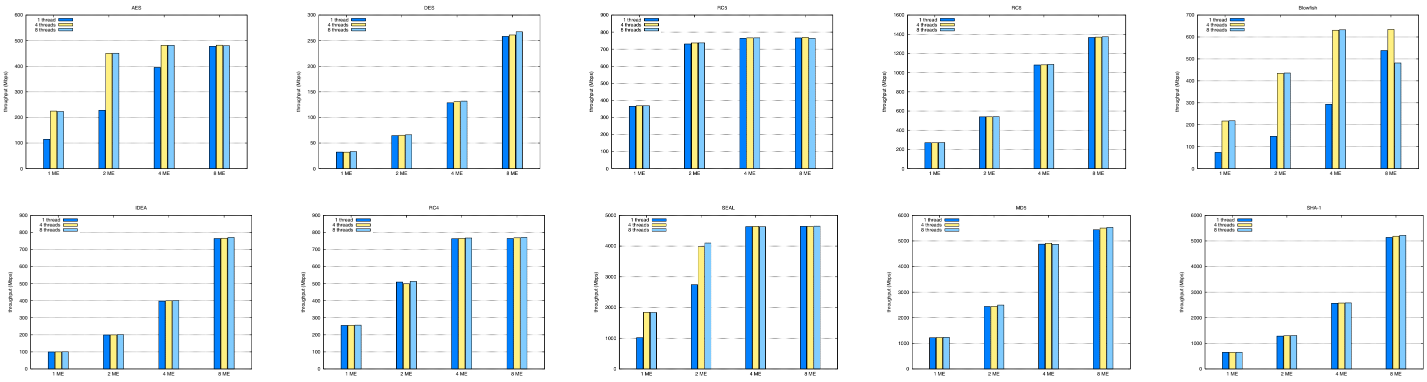

Fig. 8. Throughput of NPCryptBench (Level-2) on IXP2800 with Varying Numbers of Threads and MEs

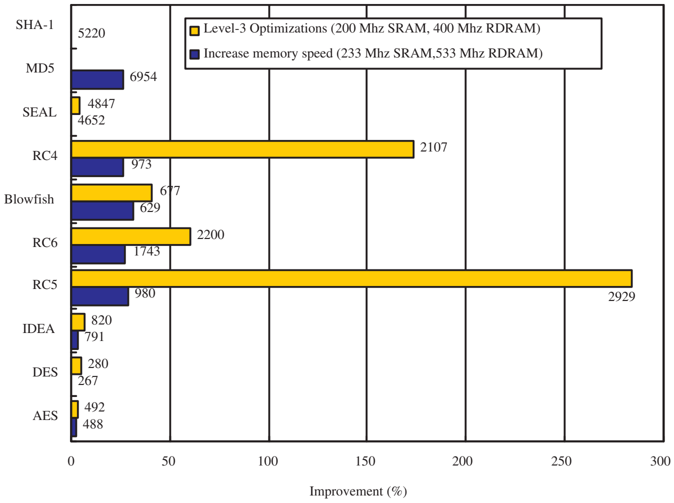

Fig. 9. Throughput and Improvement of Level-3 Optimizations and Increased Memory Frequency on IXP2800 Under 8-ME, 8-thread Configuration

Their improvements over throughput in Fig. 8 are illustrated in Fig. 9. We find Level-2 codes on the modified IXP2800 platform gain an average lift of 14.5%. In contrast, level-3 codes produce a 72.1% elevation. Therefore, hand-optimization of memory access contributes much more than memory bus speedup. Algorithms whose performances rely heavily on DRAM benefit most, such as RC4/5/6. Yet to those whose original bottleneck is ME's computing power (DES and IDEA), memory optimization offers relatively little growth of throughput, usually less than 10%. For these algorithms, hardware puts on a real limit, further lift will rely on faster PE or other architectural improvements.

VII. Conclusion and Future Work

In this paper, NPCryptBench is presented as the first attempt to evaluate cryptographic performance on NPs. Quantitative evidence proves the principal bottleneck for ciphers and hash functions being the exacerbated memory-wall effect. Since it cannot be tackled by current compilers, we put forward hand-optimizations according to application characteristics. These optimizing strategies can be incorporated in future compilers to gain better performance. Besides, we suggest several hardware improvements to alleviate the bottleneck and enhance cryptographic performance on data plane:

- Increase internal memory size (e.g. to >4K byte) on PEs to lessen the pressure on shared memory.
- Enlarge the memory and command queues to reduce the possibility of PE stalls.
- Adopt a new memory system to shorten memory access latency.

In the future, we consider developing more ports of NPCryptBench other than the Intel IXP family. Asymmetric-key ciphers will be addressed on some advanced NP models. At the same time, we plan to release all levels of NPCryptBench codes to the public, with better encapsulation.

Acknowledgment

This research is supported by Intel IXA University Program (No. 0411A66), the National Natural Science Foundation of China (No.90412012) and the National Grand Fundamental Research 973 Program of China (No. 2003CB314804).

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed, Morgan Kaufmann, 2002.
[2] T. Wolf and M. Franklin, ”CommBenchła telecommunications benchmark for network processors”, *Proc. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'00)*, 2000.
[3] B. K. Lee and L. K. John , ”NpBench: A Benchmark Suite for Control plane and Data plane Applications for Network Processors”, *Proc. International Conference on Computer Design (ICCD'03)*, 2003.
[4] G. Memik and et al., ”NetBench: A Benchmarking Suite for Network Processors”, *Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD'01)*, 2001.
[5] M. R. Guthaus and et al., ”MiBench: A Free, Commercially Representative Embedded Benchmark Suite”, *Proc. IEEE International Workshop on Workload Characterization (WWC-4)*, 2001.
[6] J. Burke and et al., ”Architectural Support for Fast Symmetric-Key Cryptography”, *Proc. 9th International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS'00)*, pp.178-189, 2000.
[7] *Intel® IXP Family of Network Processors*, http://www.intel.com/design/network/products/npfamily/index.htm.
[8] Y. Luo and et al., ”NePSim: A Network Processor Simulator with Power Evaluation Framework”, *IEEE Micro Special Issue on Network Processors for Future High-End Systems and Applications*, 2004.
[9] *EEMBC*, http://www.eembc.org.
[10] M. Peyravian and J. Calvignac, ”Fundamental Architectural Considerations for Nnetwork Processors”, *Computer Networks*, Vol.41, no.5, 2003.
[11] ”Parallel eXpress Forwarding in the Cisco 10000 Edge Service Router”,*Cisco Systems, White Paper*, 2000.
[12] N. Sklavos and O. Koufopavlou, ”Mobile Communications World: Security Implementations Aspects - A State of the Art”, *Computer Science Journal of Moldova*, Vol.11, no.2, 2003.
[13] R. Rivest, *The MD5 Message-Digest Algorithm*, RFC 1321, Apr. 1992; http://www.faqs.org/rfcs/rfc1321.html.
[14] *Secure Hash Standard*, National Institute of Standard and Technology, http://csrc.nist.gov/CryptoToolkit/tkhash.html
[15] B. Schneier, *Applied Cryptography*, 2nd ed., John Wiley & Sons, 1996.
[16] P. Rogaway and D. Coppersmith, ”A Software-Optimized Encryption Algorithm”, *Proc. the Cambridge Security Workshop*, 1994.
[17] *Advanced Encryption Standard (AES)*, National Institute of Standard and Technology, http://www.nist.gov/aes.
[18] *Data Encryption Standard (DES)*, National Institute of Standard and Technology, http://csrc.nist.gov/cryptval/des.htm
[19] R. L. Rivest, ”The RC5 Encryption Algorithm”, *Proc. 2nd International Workshop on Fast Software Encryption*, 1995.
[20] J. Nechvatal and et al., ”Report on the Development of the Advanced Encryption Standard”, National Institute of Standard and Technology, http://csrc.nist.gov/CryptoToolkit/aes/round2/r2report.pdf, 2000.
[21] B. Schneier, ”Description of a New Variable-Length Key, 64-Bit Block Cipher”, *Proc. Cambridge Security Workshop*, 1994.
[22] X. Lai, *On the Design and Security of Block Ciphers*, Hartung-Gorre Veerlag, 1992.
[23] Z. Tan and et al., ”Optimization and Benchmark of Cryptographic Algorithms on Network Processors”, *IEEE Micro, Special Issue on Network Processor For Future High-End Systems and Applications*, pp.55-69, 2004.
[24] Z. Tan and et al., ”Optimization and Benchmark of Cryptographic Algorithms on Network Processors”, *Proc. IEEE International Conference on System, Man and Cybernetics (SMC'03)*, vol.3, pp.2296-301, 2003.