Architectural Analysis of Cryptographic Applications for Network Processors

Haiyong Xie, Li Zhou, and Laxmi Bhuyan
Department of Computer Science & Engineering
University of California, Riverside
Riverside, CA 92521
yong@cs.ucr.edu

Abstract

Network processors are application specific programmable processors and will become critical components of next-generation networking equipment. As Internet expands exponentially, the need for secure communication increases very quickly. The performance of communication applications including packet processing and cryptographic applications on network processors thus becomes an important issue of network processor system design.

In this paper, we compare and analyze the architectural characteristics of many widespread cryptographic algorithms and their implementations through simulation on a MIPS-like architecture. We find that the instruction mix is different from SPEC95 programs; and that the average size of basic blocks is 2~3 times larger than that of common applications. Only 7% of instructions are conditional branches and most of them are taken. Most of the cryptographic applications have an ILP of 8. Most of the applications have small kernels of less than 16KB. Memory system has much less important effect on the overall performance. We find that only a small direct-map instruction cache and data cache are needed to achieve comparable performance. Cache replacement strategy is not important to the overall performance. The results in this paper are helpful to the design of network processors.

1. Introduction

As Internet expands exponentially, the need for secure communication and electronic commerce increases very quickly. The Internet has been used as a trusted medium, which made cryptography a crucial component of modern information infrastructure. A collection of cryptography applications such as secure IP (IPSEC) and virtual private networks (VPNs) has been widely deployed in both routers and end systems. This trend will further emphasize the importance of cryptographic applications among all types of communication. Security related applications are all computational intensive applications that can consume as much as 95 percent of an application server's processing capacity [10]. As the demands for and deployment of secure communication grow, cryptographic processing may become a bottleneck to the system performance.

On the other hand, the emerging Network Processors (NPs), which are application specific programmable processors, will become fundamental building blocks of next generation networking equipments. Network processors can provide high and flexible packet processing and have been targeted for diverse application domains. Network processors are mainly designed and improved for high and flexible packet processing such as packet forwarding based on routing tables at wire speed. However, they are targeted not only for packet processing applications. As demands for communication security grow, cryptographic processing becomes another type of application domain. To make network processors flexible for diverse application domains, we need study the architectural requirements of each domain, especially cryptographic application domain.

The bandwidth of Internet links and the packet processing power of network processors have been increasing very quickly in the past few years. To meet the increasing demands for secure communication, the network processors have to performance cryptographic functions at the full speed to achieve comparable performance of security processing. There have been a lot of commercial products, the so-called security processors, for example, HIPP 8154 security processor by Hifn, which is claimed to support up to a full duplex OC-48 link [7], BCM 5840 security processor by Broadcom [2], and NSP-series security processors by NetOctave [12], etc. The fastest security processor can sustain the wirespeed of multi-gigabit per second. However, the impact of security

related functions performed on the network processors is still not clearly known to us.

Compared to studies on architectures and applications of packet processing power provided by network processors, little research has been conducted on the architectural requirements of cryptographic applications for processor designs. Our motivation to study the cryptographic applications is that we believe that these computationally intensive applications have great differences in the architectural properties from other applications such as packet processing; therefore, the design of cryptographic application specific processors should be different from that of packet processing application specific processors.

In this paper, we consider three types of cryptographic applications: hash algorithms, and two types of private-key ciphers, namely, block ciphers and stream ciphers. The other form of cryptography, public-key ciphers, is not studied in this paper. Although the advantage of public-key algorithms is being able to establish a secure communication channel without an unsafe exchange of keys, its execution is quite slow which is about 1000 times slower than that of comparable secret-key algorithms. To secure the maximum security and high speed [11], public-key ciphers are mainly used at the start of a secure session to authenticate communicating parties and to securely establish and manage a shared private key, while using secret-key ciphers as the cryptography algorithm for the remaining session. The performance of public-key ciphers is critical for very short sessions, while that of private-key ciphers is critical for longer sessions. We focus our effort on performance of secret-key ciphers, not on that of the public-key ciphers.

Through detailed timing simulation and profiling, we find that cryptographic applications demonstrate quite different architectural properties as expected. The architectural properties we studied include instruction set characteristics, instruction level parallelism (ILP), and cache performance. We find that the instruction mix of these applications consists of much higher percentage of arithmetic instructions (68% in average), lower percentage of memory reference instructions (23% in average), and much lower percentage of unconditional and conditional branch instructions (2% and 7% in average respectively), compared to SPECint95 [19] and CommBench [20] benchmarks. Most of the conditional branches are taken. That facts that the percentage of conditional branch instructions is very low and that most of the branches are taken mean that complicated and high precision branch prediction mechanism will not do much good to the performance, as is proved by the simulation results. The average size of basic blocks is 2~3 times larger than that of SPECint benchmark programs, which means it is possible to take advantage of instruction level parallelism much better. We find that most of the cryptographic applications have an ILP of 8. Compared to instruction level parallelism, cache architectures have much less important effect on the overall performance. We find that all the applications have small kernels of no more than 16 KB, and that a small direct mapped instruction cache and a similar data cache are enough for most of the applications to achieve comparable performance. Cache replacement strategy is not important to the overall performance.

The above results are helpful to the design of network processors. Based on the above architectural properties of the widely used cryptographic applications we have chosen to study, it seems a good idea to use a standalone cryptographic application specific chip attached to the network processor to effectively meet the high throughput demands in secure communication environments.

The rest of this paper is structured as follows. Section 2 details the selection of cryptographic algorithms. Section 3 describes the simulation environment and methodology. Section 4 presents the instruction set characteristics and instruction mix profile of these applications. Section 5 describes the computational complexity of cryptographic programs measured by the number of cycles spent in processing one byte data. Section 6 presents the instruction level parallelism properties. Section 7 shows the branch prediction properties. Section 8 deals with the cache behavior. Section 9 summarizes the contributions of this work and concludes this paper.

2. Selection of Cryptographic Applications

The most important criteria of selection of cryptographic algorithms and their implementations for architectural analysis is the representativeness of a wider application class in the domain of interest. There are two such application domains: hash algorithms and private-key ciphers, the latter of which includes block ciphers and stream ciphers. Cryptographic applications in these domains are all widely used in Internet applications. The only exception is AES because it was taken as the advanced encryption standard not a long time ago. AES will take place of DES in the future. The second criteria is the popularity and availability of the algorithms. Widely used algorithms favor over less used ones. Most of these cryptographic algorithms are employed in popular protocol suites such as SSL and applications such as PGP. With these in mind, we choose 9 algorithms and their implementations for analysis. Five of them are block ciphers; two of them are data stream processing; another two implements hash algorithms.

2.1 Block Ciphers

The majority of the encryption algorithms in use today are block ciphers. They take blocks of data (typically 64

bits or 128 bits) as input and only encrypt the blocks separately. The summaries of selected block ciphers are shown in Table 1.

• AES, which is also named as Rijndael [5], is the standard of AES [1]. It has a variable key size of 128, 192 or 256 bits. The symmetric and parallel structure of this algorithm gives implementers a lot of flexibility, and has not allowed effective cryptanalytic attacks. AES can be well adapted to a wide range of modern processors such as Pentium, RISC and parallel processors. AES has been put into wide use up to now. One of the examples is DMSEnvoy developed by Distributed Management System Ltd.

• Blowfish [4] is one of the most secure algorithms available. It combines a key-dependent S-Boxes, a Feistel network and a non-invertible F function to generate cipher text that none existing attacks ever break it. Another advantage of this algorithm is its simplicity to implement.

• 3DES [6] achieves a high level of security by encrypting the data three times using DES with three different, unrelated keys. Therefore, 3DES use a larger size of key to encrypt than that of DES. The larger the key, the harder the cipher can be broken.

• IDEA [8] is generally regarded as one of the best and the most secure block ciphers available to the public today. It uses 128-bit keys and operates on 64-bit data blocks. Another reason for us to select IDEA is that it is, on average, much faster than many other ciphers.

• RC6 [15] is an AES candidate designed by Ronald Rivest. It operates on 128-bit blocks and can accept keys of variable length. RC6 is a secure, compact and simple block cipher. It offers good performance and considerable flexibility.

Table 1. Selection of block ciphers

	Designer	Key Length (bits)	Block Size (bytes)	Application Example
AES	J. Daemen, V. Rijmen	128, 192 or 256	16	DMSEnvoy
Blowfish	B. Schneier	<=448	8	Nortron Utilities
3DES	D. Coppersmith	168	8	SSL, SSH
IDEA	X. Lai, J. Massey	128	8	PGP, SSH, SSL
RC6	R. Rivest, M. Robshaw, *et al*	128,192, or 256	16	AES candidate

2.2 Stream Ciphers

Compared with block ciphers, stream ciphers take data of variable length as operation objects. They use random numbers as the keys, which are combined with the plaintext to generate the cipher text. The better the keys are randomly generated, the more secure the stream cipher is. The summaries of selected stream ciphers are presented in Table 2.

• RC4 [18] is a variable key-size (up to 2048 bits) stream cipher developed by Ron Rivest for RSA Data Security, Inc. The algorithm is very fast. Its security is unknown, but breaking it does not seem trivial either. Because of its speed, it may have uses in certain applications such as Lotus Notes and Oracle Secure SQL.

• SEAL [17] is probably the fastest secure cipher available. Although it uses SHA1 in the key setup process, requiring several kilobytes of space and very intensive computation, it only needs five operations per byte to generate the key stream. The main application for SEAL is disk encryption and similar applications where data must be read from the middle of a cipher text stream.

Table 2. Selection of stream ciphers

	Designer	Key Length (bits)	Application Example
RC4	R. Rivest	Minimum 8, maximum 2048, multiple of 8 bits; default 128 bits	SSL
SEAL	P. Rogaway	Variable, default 160	Disk encryption

2.3 Hash Algorithms

Table 3 gives the summaries of selected hash algorithms.

• MD5 [16] is an accepted standard for message digest. It generates an output of 128-bit message digest of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest. The MD5 algorithm is commonly used for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private key under a public-key cryptosystem. MD5 is much more reliable than checksum and many other commonly used methods.

• SHA1 [9] is specified within the Secure Hash Standard (SHS) for using with Digital Signature Standard (DSS). It has a greater hash size than MD5, so it is more secure. It generates 160-bit digest, which is large enough to protect against “birthday” attacks.

Table 3. Selection of hash functions

	Designer	Block Size (bits)	Digest Size (bits)	Output Size (bits)	Application Example
MD5	R. Rivest	512	128	128	digital signature
SHA1	U.S. National Security Agency	512	160	160	digital signature

3. Methodology

What we focus on in this paper is the generalized architectural properties of the selected cryptography applications that are applicable to all network processor architectures. To study the architectural characteristics of these applications, we port and run them in the execution driven simulator, SimpleScalar [3]. The SimpleScalar tool set is a suite of publicly available simulation tools that provides fast, flexible, and accurate simulation of modern processors that implement the SimpleScalar architecture, which is a close derivative of the MIPS architecture [14].

The reasons we choose SimpleScalar tool set, version 2.0, as the simulation environment include the problems with measurement and unavailability of simulators designed for network processors specifically. We can only get very limited information from measurement because we are unable to verify the architectures to see the corresponding performance. We need such a simulation environment as SimpleScalar in which we can verify various architectural parameters such as the number of function units, cache sizes, and branch prediction mechanisms, etc. On the other hand, the unavailability of simulators designed for network processors specifically make us unavoidably resort to such simulators as designed for general-purpose processor architectures. However, we are able to obtain most of the architectural features for the applications even with SimpleScalar which simulates a close derivative of MIPS architecture, although MIPS architecture is quite different from most of network processors.

By means of simulation, we obtain such characteristics as instruction level parallelism, instruction mix profile, and cache performance for the applications as a function of various architectural parameters. The C compiler used is *gcc 2.6.3* (optimization level O2) coming with SimpleScalar. The O2 optimization level is selected for the resason that the compiler only performs optimizations that are independent of the target processors and does not exploit particular architectural features.

The programs are executed with a relatively large text file of 260 KB as the input. A key of 128 bits is used with all the block and stream ciphers except 3DES and SEAL, which are executed with a key of 168 bits and 160 bits respectively.

The default configuration of the simulated processor architecture has a L1 instruction cache and a L1 data cache, a unified L2 cache, an ILP of 4, and bimod as the branch prediction algorithm. The L1 caches have 4-way set associative, 32-byte line size, LRU replacement strategy, and 16KB in size. The unified L2 cache has 4-way set associative, LRU replacement strategy, 64-byte line size, and 512KB in size. This L1 and L2 cache configuration are the same as that of PentiumII microprocessors.

4. Instruction Set Characteristics

The instruction set characteristics give an indication on the types of instructions executed and their frequencies in the programs. Figure 1 presents the instruction mix profile and frequencies for the implementations of all the selected algorithms, averages for these algorithms, and SPECint95 programs.

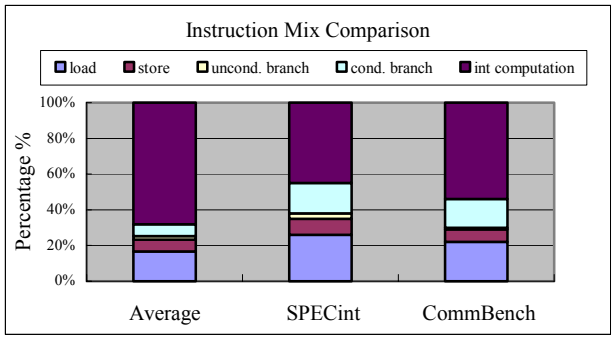
The average instruction mix of these cryptographic programs shows great differences from that of both SPECint and CommBench programs. Compared to the CommBench, the proposed benchmark for network processors, the selected cryptographic programs demonstrate quite different instruction mix properties; The differences between these selected programs and SPECint programs are even greater, as shown in Figure 1(a).

Figure 1(a) depicts the averages of block ciphers, stream ciphers, hash algorithms. The following points out the differences:

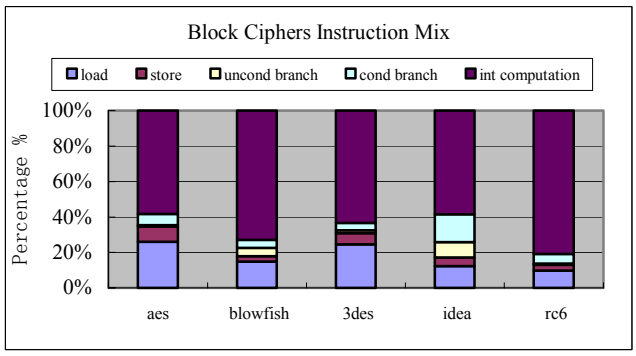
(1). The cryptographic programs have much higher percentage of arithmetic instructions, which is 68% in average compared to 45% of SPECint95 [13]. They have 23% percentage more arithmetic instructions than general programs. This fact proves from another aspect that these cryptographic applications are in nature computational intensive applications. Cryptographic applications are computation oriented; therefore, they may consume most of network processor's computation power.

(2). The cryptographic programs have much less percentage of memory reference instructions, which is 23% compared to 35% of SPECint95 programs. This observation is very interesting. It means that cryptographic applications are not memory reference bounded and that we may not need a complicated memory system with very high hit rate. It may also means that high memory bandwidth is not necessary for good performance. By studying the cache behaviors of these programs, we find that the cache system need not be very complicated with high hit rate for cryptographic applications. Simple caches are enough for most of these programs to achieve comparable good performance, as is described in detail in the section dealing with cache behaviors.

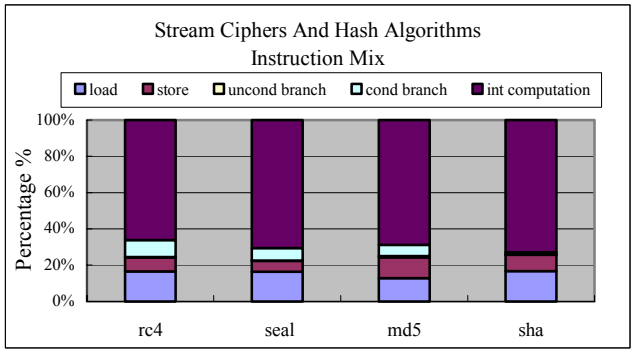
(3). The cryptographic programs have much lower percentage of branch instructions, which is 9% in average compared to 20% of SPECint95. 2% of these 9% branch instructions are unconditional instructions compared to a similar 3% in SPECint95, and the other 7% are conditional instructions compared to 17% in SPECint95. The sharp difference in the conditional branch instruction frequencies makes it unnecessary to employ complicated branch prediction mechanisms. In the following sections we also study the branch prediction requirements for these cryptographic applications.



(a) Comparisons of average instruction mix



(b) Block ciphers instruction mix



(c) Stream and hash ciphers instruction mix

Figure 1. Instruction mix characteristics

Figure 1 (b) and (c) show the instruction mix profile for block ciphers, stream ciphers, and hash algorithms respectively. The instruction mix profiles of the latter two kinds of applications are combined into one figure for the sake of brevity. The following observations are important:
(1). Among all the selected block ciphers, only AES and 3DES have similar percentage of memory reference instructions (35% and 31% respectively) compared to SPECint95 programs (35% in average). Thus these two applications have higher requirements on the L1 data cache architectures, as is proved by the studies of cache behaviors.
(2). Among all the selected block ciphers, IDEA has similar percentage of conditional branch instructions (16%) compared to SPECint95 programs (17% in average). This potentially means that IDEA implementation may need better branch prediction mechanisms with higher hit rate to achieve good performance. However, in later section dealing with branch prediction properties, it is learned that it is still not necessary to employ such mechanisms for IDEA applications.
(3). All the selected stream ciphers and hash algorithms have quite different instruction mix properties from SPECint95. They all have much lower percentage of both memory reference and branch instructions, and higher percentage of arithmetic instructions.
(4). Stream ciphers are more similar to hash programs in terms of instruction mix. Block ciphers are quite different from both stream ciphers and hash programs. This observation is proved by later studies in this paper as well.
The above observations are very important to our further studies in this paper. The following sections deal with the points mentioned above respectively.

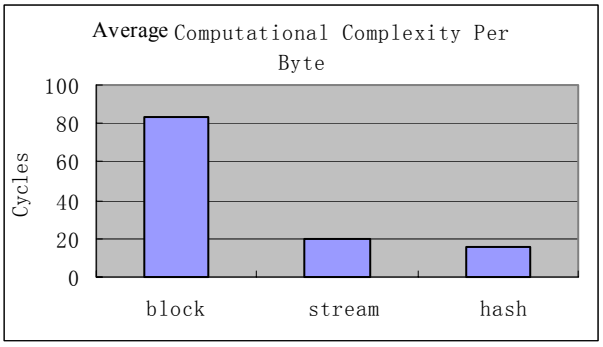
5. Computational Complexity

As deduced in the above section, the cryptographic programs are all computationally intensive programs. This section shows the computational complexity measured in terms of the number of cycles spent per byte of the input data for each of the selected programs.

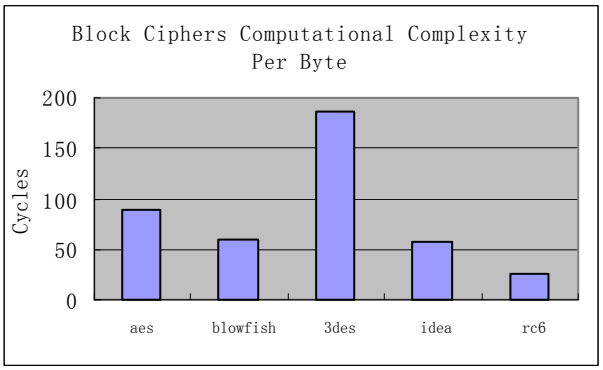
Figure 2(a) shows the average computational complexity per byte for each of the three types of applications. Block ciphers spend 84 cycles in processing each byte of input data in average, while stream ciphers and hash programs spend 20 and 15 cycles respectively. Stream ciphers and hash programs have more similarities. Block ciphers are more computationally intensive than both stream cipher and hash programs.

Figure 2(b) depicts the computational complexity for each of the block ciphers. We can see that 3DES spends much more cycles (187 cycles) than all other ciphers in processing one byte data. This is because 3DES applies the same data manipulation process three times with three different keys. The computational complexity is thus tripled. AES has a relatively high computational complexity compared to the other 3 ciphers.

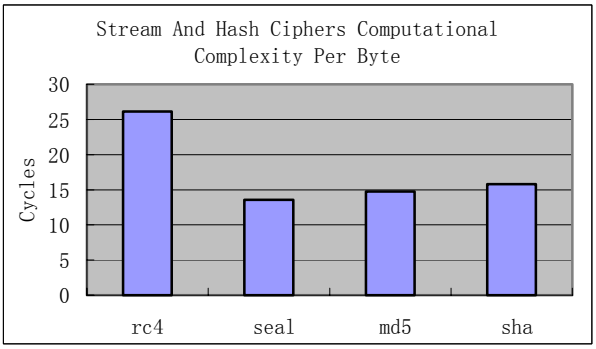
Compared to block ciphers, stream ciphers and hash programs need much less cycles to process one byte data, as shown in Figure 2(c). Only RC4 spends more than 25 cycles in processing one byte data, while other ciphers spend around 15 cycles. From this aspect, stream ciphers are more like hash algorithms rather than block ciphers.



(a) Average computational complexity



(b) Block ciphers computational complexity



(c) Stream and hash ciphers

Figure 2. Computational complexity

From the computational complexity measured by cycles per byte shown in Figure 2, we are able to derive the maximal security processing power of this simulated MIPS architectures. The average number of cycles needed to process one byte for block cipher, stream cipher, and hash functions are approximately 80, 20, and 15 respectively. If this simulated MIPS-like CPU runs at 100MHz, then the average security processing power measured in terms of throughput is 10Mbps, 40Mbps, and 53Mbps respectively. Even if the CPU runs at 1GHz the processor still cannot sustain a full duplex OC-12 link. Security related functions requires much more processing power from the processors than packet processing functions, therefore, they will become the performance bottleneck of network processors. The processing power of a general-purpose processor is not sufficient for a network processor to sustain the high bandwidth links. Novel architectures optimized for security functions are needed in order for the network processors to have sufficient security processing power.

6. Instruction Level Parallelisms

Instruction level parallelism is a crucial issue for consideration. We can exploit instruction level parallelisms to achieve high performance for these computationally intensive cryptographic applications.

To study the ILP properties of these applications, we varify the number of ALUs, the size of instruction fetch queue, and both, to see the separate and combined contributions of each component. We did not consider the decoder and issue units and we simply set the number of these resources to their maximum value so that they do not have any impact on the performance.

Figure 3 shows the impact of changing the number of ALUs. The size of instruction fetch queue is set to its maximum number available to eliminate the affect. It is observed that the number of instructions that can be executed in one cycle increases by 64%, 56%, and 85% for all the three kinds of applications respectively when the number of ALUs increases from 1 to 2, and by 23%, 18%, and 26% when the number of ALUs increases from 2 to 4. However, with more than 4 ALUs, the number of instructions executed in one cycle increases only less than 1%.

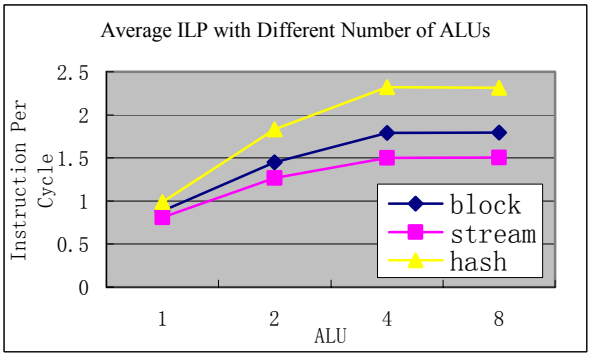


Figure 3. Impact of ALU on average ILP

Figure 4 shows the separate contribution of instruction fetch queue to the overall performance measured by instructions executed per cycle. The number of ALUs is set to its maximum value in order to eliminate the affect of ALUs. From the figure it is observed that an instruction fetch queue of size 4 is enough for all cryptographic applications. The number of instructions executed per cycle increase by 25%, 20%, and 26% after the size of the instruction fetch queue changes from 2 to 4. After that, the performance improvement is less than 3% when we double the size of the queue.

Figure 5 depicts the overall impact of functional unit resources available, that is, the number of ALUs and the size of instruction fetch queue. We use ILP to represent the changes of functional unit resources, for example, an ILP of 8 means that the processor has an instruction fetch queue of size 8 and 8 ALUs available. It has been seen that the performance measured by the number of instructions executed per cycles increases linearly as ILP increases from 1 to 2 to 4. The performance improvement is 6% when ILP increases from 4 to 8. From these figures we conclude that an ILP of 4 is enough and the most cost-effective for achieving the best performance for the selected cryptographic applications.

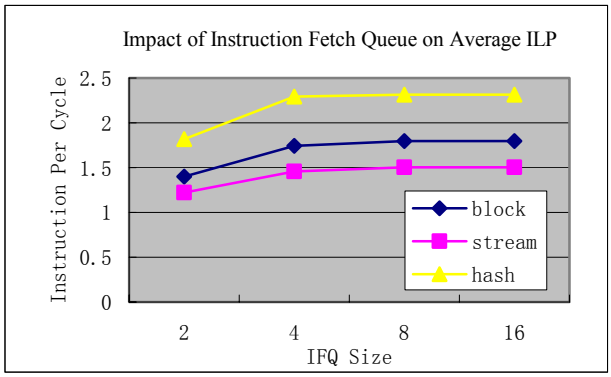


Figure 4. Impact of IFQ on average ILP

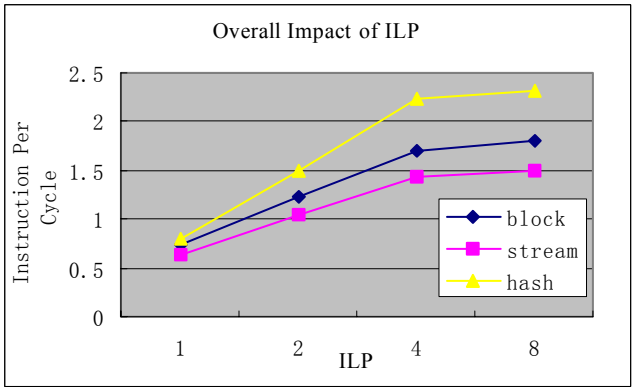


Figure 5. Overall impact of ILP

Another observation is that there exist high data dependences in the instruction sequences of these programs.

Figure 6 shows the average size of basic blocks for these three kinds of applications. Both block and stream ciphers have large basic blocks of 12 instructions, which is nearly 2 times larger than that of SPECint95 in average. Hash programs have very large basic blocks of 60 instructions. Since there are only a small amount of branch instructions in cryptographic programs, there are less control dependences in the instruction sequences. Generally speaking, programs with larger basic blocks have more instruction level parallelism provided that the data dependences are not too high. However, this is not the case with these cryptographic applications. All the programs have high data dependences in the instruction sequences that make the best exploitable ILP 4.

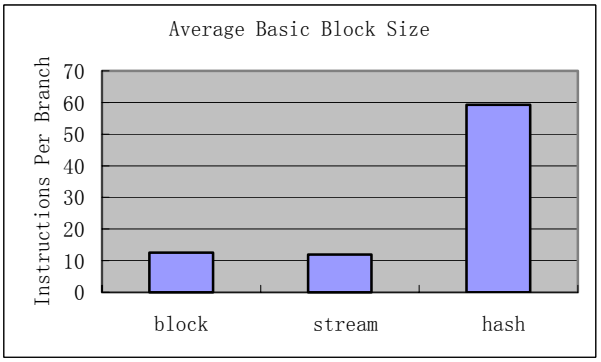


Figure 6. Average basic block size

7. Branch Prediction

Branch prediction does not seem to be so important as instruction level parallelisms. One of the reasons has been stated in Section 4: the percentage of conditional branch instructions (7% in average) is so low in cryptographic applications. Even if the miss prediction rate is high, the effect of this miss prediction is still not so serious to the overall performance.

Another consideration is what kind of branch prediction mechanism is best suitable to cryptographic programs. Figure 7 presents the prediction hit rate of each prediction mechanism available in the simulator for the three types of applications.

It can be observed from Figure 7 that most of the conditional branches are taken, namely, 80%, 98%, and 90% of the branches are taken for block ciphers, stream ciphers, and hash programs respectively. Therefore, sophisticated branch prediction mechanisms are not cost-effective for cryptographic programs. A simple static branch prediction mechanism with the branches being predicted always taken is sufficient.

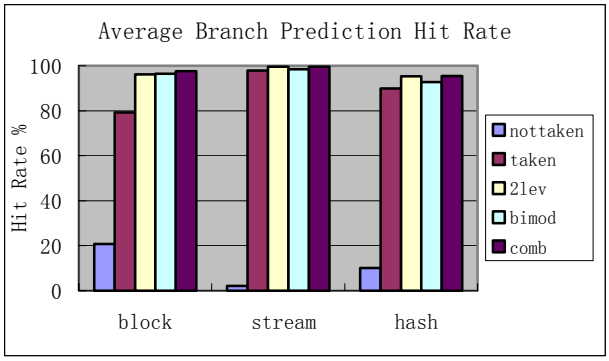


Figure 7. Average branch prediction hit rate

8. Cache Behaviors

Cache behaviors are another important consideration for the design of network processors. However, as found in Section 4, the memory reference instructions account for only 23% of all the instructions executed for cryptographic programs in average. This means that a relatively simple cache architecture is possibly enough for most of the security related applications.

We measure the cache performance for each of selected applications. Separate instruction and data caches ranging from 4KB to 256KB are simulated.

8.1 L1 Instruction Cache Behaviors

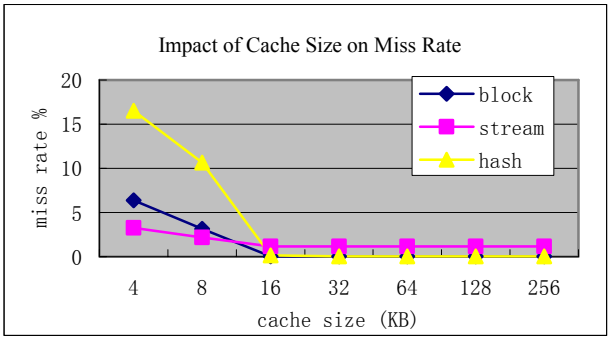
Figure 8 shows the results for instruction cache behaviors by changing such cache parameters as cache size, line size, set associative, and replacement strategy.

Figure 8(a) presents the simulation results of cache performance when the cache size is changed. It can be seen that 16KB instruction cache is enough to achieve miss rates less than 0.1% for both block ciphers and hash programs, and less than 1% for stream ciphers. This figure also shows that the miss rate of stream ciphers cannot be reduced even with much larger cache sizes due to compulsory misses. Compared to benchmark programs of CommBench, cryptographic programs have larger kernels. For CommBench programs, the instruction cache miss rates are lowered to under 0.5% when the instruction cache is increased to 8KB; however, the miss rates are as higher as 2% with the same cache size for the selected programs. Even when the cache size is 16KB, the lowest miss rate of the programs is still approximately 1%, which is 4 times higher than that of CommBench programs. The instruction cache behaviors of cryptographic programs are much more similar to that of SPECint programs, which have an average miss rate of 2.2% and 1% for 8KB and 16KB instruction caches respectively.

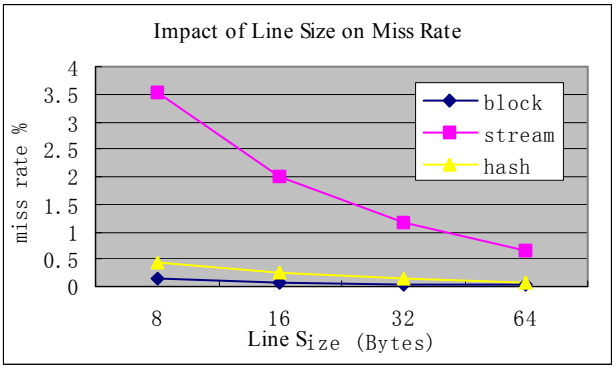
When the line size of instruction cache is increased, the miss rate is lowered for all the applications. The line size increase has greater impact on stream ciphers compared to that on block and hash ciphers. A line size of 8 bytes is enough for block and hash ciphers to achieve miss rates less than 0.5%; on the contrary, a line size of 64 bytes can only lower the miss rate to 1.2% for stream ciphers.

Figure 8 (c) shows that impact of set associative on miss rate. For most of the applications a cache with direct mapping is enough to obtain less than 5% miss rate. The exceptions are AES and RC4. Both of these two exceptions need 4-way set associative cache to obtain the next to the best performance.

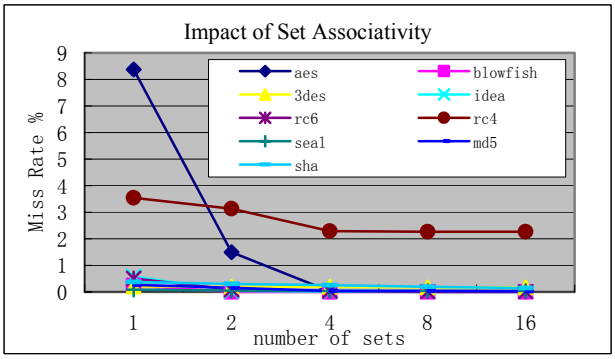
Figure 8 (d) presents the impacts of cache replacement strategies. It is not surprising that FIFO has similar performance to LRU replacement algorithm. This is only true when the size of the cache is large enough to avoid the compulsory misses.



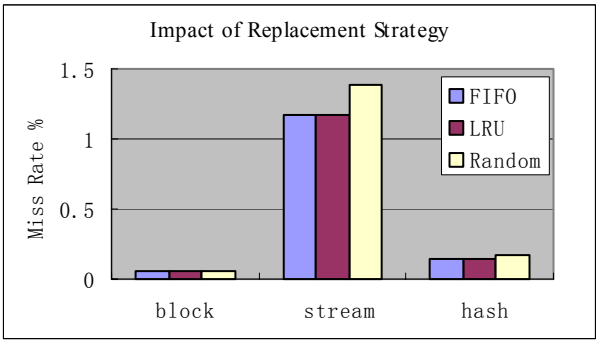
(a) Impact of cache size



(b) Impact of line size



(c) Impact of set associativity



(d) Impact of replacement strategy

Figure 8. Instruction cache behavior

From the four figures we can also find that all the cryptographic programs have rather small kernels which are less than 16KB. This observation is important in that we can easily store the kernels of cryptographic applications in a reserved area in the cache so that instruction cache miss will be reduced.

Combining the above four observations into one, we can safely conclude that a direct-mapped 16KB instruction cache with line size as 8 bytes is enough for most of the cryptographic applications to obtain next to the best performance. The exceptions are AES and RC4, which need a 4-way instruction cache of 16KB and larger line size to achieve comparable performance. FIFO replacement strategy contributes almost the same as LRU; therefore, sophisticated replacement strategy is not necessary.

8.2 L1 Data Cache Behaviors

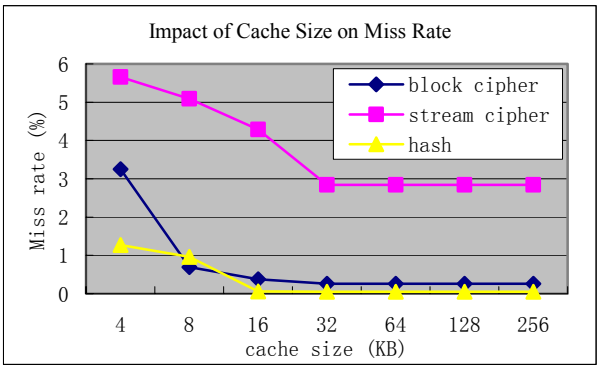
The cryptographic applications have similar data cache behaviors to instruction cache behaviors. Figure 9 shows the simulation results by changing cache size, line size, set associative, and replacement strategy.

Figure 9(a) shows the impact of cache size on the miss rate. A cache of 8KB can reduce miss rates of block and hash ciphers to less than 1%; while stream ciphers need 32KB cache to achieve miss rates less than 3%, which is the best achievable, in average. The miss rate cannot be reduced any more even with larger cache sizes for stream ciphers. This is because there exist too many compulsory misses. Stream ciphers take the input data as continuously streamed data, which leads to high compulsory miss rate.

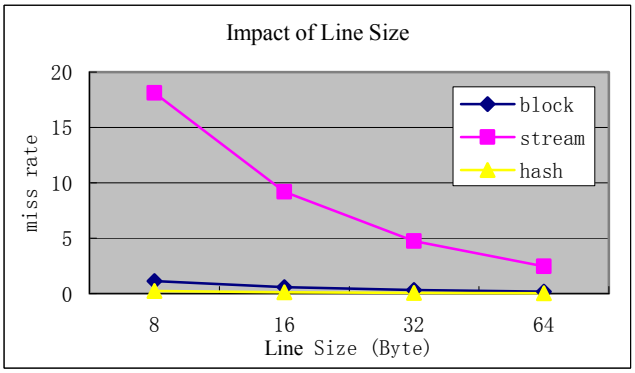
Line sizes have different impacts on miss rates for the three types of applications, as shown in Figure 9(b). A 4-way 16KB data cache with a line size of 8 bytes has less than 1% miss rate for hash and most of block ciphers except 3DES. The larger the line size is, the lower miss rates for stream ciphers and 3DES are.

The data cache behaviors of the selected cryptographic programs are more similar to that of CommBench rather than SPECint benchmark programs. The average data miss rates are lowered to less than 1% with 8KB data cache for selected program except block ciphers, while the miss rates of SPECint and CommBench benchmark programs (except for ZIP, FRAG, and DRR applications) are approximately 4% and below 1%, respectively.

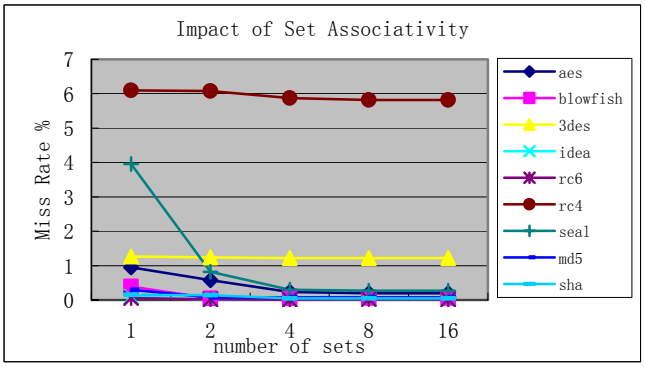
Hash programs and most of the block ciphers do not have high demands for set associativity, as shown in Figure 9(c). A direct-mapped cache of 16KB is enough to achieve miss rate lower than 1% for both hash and block ciphers with 3DES as the only exception. Compulsory miss dominates cache miss rate of RC4 and 3DES. SEAL needs 2-way set associative data cache to obtain miss rate lower than 1%.



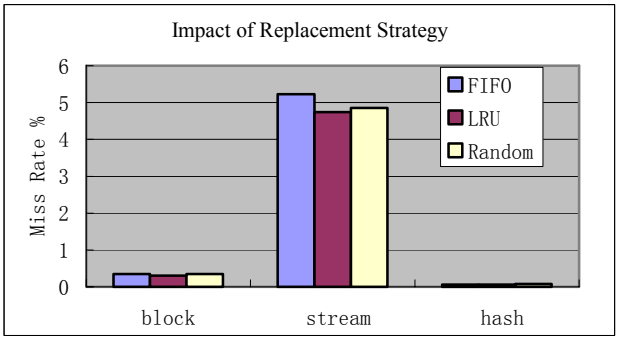
(a) Impact of cache size



(b) Impact of line size



(c) Impact of set associativity



(c) Impact of replacement strategy

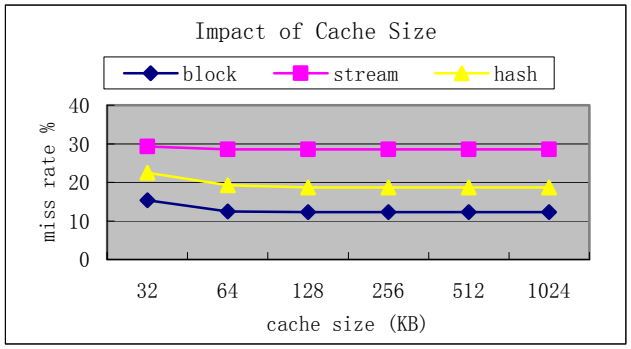
Figure 9. Data cache behavior

Cache replacement strategy does not have greater impact on the cache miss rate. LRU replacement strategy can be replaced with FIFO for block and hash ciphers and Random replacement for stream ciphers without increasing miss rate by more than 3%.

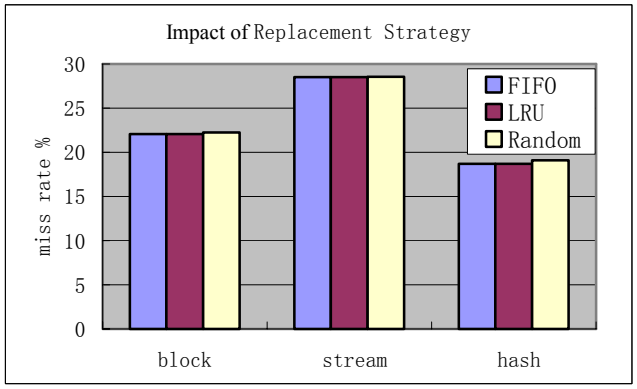
8.3 L2 Unified Cache Behaviors

We have simulated the L2 unified cache behaviors for all the applications as well, the results of which are shown in Figure 10.

Due to most of the memory references are absorbed by L1 caches, the miss rate of L2 cache is high, as shown in Figure 10(a). Figure 10(a) indicates that only a very small L2 cache of 64KB is needed to achieve the lowest miss rate. L2 cache is also used to decrease the latency of memory references. Figure 10(b) shows the impact of different replacement strategies on cache miss rate. All the three replacement strategies, FIFO, LRU, Random, have nearly the identical impacts.



(a) Impact of cache size



(b) Impact of replacement strategy

Figure 10. L2 unified cache behavior

9. Conclusions

The performance of cryptographic processing has become a critical factor of good system design as the Internet expands as the primary medium for secure communication. In this paper we selected nine widely used cryptographic programs and analyzed their architectural demands for network processors. We focus on the generalized architectural properties of the selected cryptography applications that are applicable to all network processor architectures. Impacts of individual network processor architectures are not considered in this paper. We take advantage of the SimpleScalar tool set to simulate the MIPS-like processor architecture to learn the performance of these selected algorithm implementations.

Security related functions requires much more processing power from the processors than packet processing functions. Based on the computational complexity we have observed, we see that the security functions would become the performance bottleneck of network processors with security functions. The processing power of a general-purpose processor embedded in a network processor is not sufficient to sustain the high bandwidth links. Novel architectures optimized for security functions are needed in order for the network processors to have sufficient security processing power.

We studied the architectural properties including instruction set characteristics, instruction level parallelisms, branch prediction, and cache behaviors for these nine programs. We find that the instruction mix of these programs has major differences from that of SPECint95 benchmark programs. Cryptographic algorithm implementations have much higher percentage of arithmetic instruction, much lower percentage of branch instructions and memory reference instructions. Stream ciphers are much more similar to hash ciphers than to block ciphers in terms of computational complexity. We find that the average size of basic blocks is 2~3 times larger than common applications. Most of the branches are taken. A simple branch prediction with all the branches being predicted taken is enough for comparable performance. Most of the cryptographic applications have an ILP of 8. The high data dependence is the main limitation to exploit more ILPs. Memory system has much less important effect on the overall performance. We find that most of the applications have small kernels of less than 16KB. We only need a small direct-map instruction cache and data cache to achieve comparable performance. Cache replacement strategy is not important to the overall performance. The results in this paper are helpful to the design of network processors. It is a good idea to use a standalone cryptographic application specific chip multiprocessor attached to the network processor to effectively meet high throughput demands in secure communication.

References

[1] Advanced Encryption Standard (AES) Development Effort, US Government, http://csrc.nist.gov/encryption/aes/

[2] Broadcom Corporation, http://www.broadcom.com/
[3] D. Burger, T. Austin, The SimpleScalar Tool Set, Version 2.0, Technical Report, Computer Science Department, University of Wisconsin-Madison, June 1997
[4] CryptSoft Technologies, http://www.cryptsoft.com, 2000
[5] J. Daemen, V. Rijmen, AES Proposal: Rijndael, http://csrc.mist.gov/encryption/aes/round2/AESAlgs/Rijndael, 1999
[6] D. Davis, W. Price, *Security for Computer Networks*, Wiley, 1989
[7] Hifn, Inc., http://www.hifn.com/
[8] X. Lai, On the Design and Security of Block Ciphers, Hartung-Gorre Veerlag, 1992
[9] A. Menezes, P. van Oorschot, S. Vanstone, Algorithm 9.53 Secure Hash Algorithm - revised (SHA-1), *Handbook of Applied Cryptography*, CRC Press, 1997
[10] M. Merkow, J. Breithaupt, J. Breithaupt, *The Complete Guide to Internet Security*, AMACOM, 2000
[11] R. Mollin, *An Introduction to Cryptography*, CRC Press, 1999
[12] NetOctave, Inc., http://www.netoctave.com/
[13] D. Patterson, J. hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., 1996
[14] C. Price, MIPS IV Instruction Set, Revision 3.1, MIPS Technologies, Inc., January 1995
[15] R. Rivest, M. Robshaw, R. Sidney, Y. Yin, The RC6 block Cipher, RSA Security, http://csrc.nist.gov/encryption/aes/round2/AESAlgs/RC6.
[16] R. Rivest,The MD5 Message-Digest Algorithm, RFC 1321, April 1992
[17] P. Rogaway, D. Coppersmith, A Software-Optimized Encryption Algorithm, *Journal of Cryptology*, vol. 11, num. 4, pp. 273-287, 1998.
[18] RSA Security, http://www.rsa.com
[19] Standard Performance Evaluation Corporation, SPEC CPU95 Version 1.10, August 21, 1995
[20] T. Wolf, M. Franklin, CommBench – A Telecommunication Benchmark for Network Processors, *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, TX, Apr. 2000