Architectural Analysis of OpenSSL Crypto Algorithms for Network Processor

Piyush Ranjan Satapathy
Department of Computer Science & Engineering
University of California, Riverside
Riverside, CA 92521
piyush@cs.ucr.edu

Abstract

The wide spread adoption of the internet as a trusted medium of communication and commerce has made cryptography an essential component of modern information systems. So the performance of cryptographic communication applications on network processor has become an important topic of network processor system design. In this paper I compare and analyze the architectural characteristics of some wide spread cryptographic algorithms and their implementations through simulation on Simple Scalar, a MIPS like architecture. I have compared the instruction mix of the OpenSSL crypto algorithms with that of the SPEC95, CommBench and Average kind of algorithms. Also I have given a clear comparison of the average computational complexity per byte between the OpenSSL crypto algorithms and the others. Then considering 7 of these crypto algorithms (1 stream cipher type,3 block cipher type and 2 hash type) i have analyzed the impact of various cache sizes, different kind of branch predictions, different number of ALUs, and different Instruction fetch queue sizes. I find that memory system has a significant effect on overall performance. An ILP of 8,128 KB Instruction cache size and 32 KB Data cache size and direct mapping give better cryptographic operation for OpenSSL crypto algorithms. Cache replacement strategy doesn't have an importance in overall performance.

1. Introduction

In an increasingly connected world, cryptography has become an essential component of modern information systems. Cryptography provides the mechanisms necessary to provide accountability, accuracy and confidentiality in inherently public communication medium such as the internet. Today cryptography processing is primarily reserved for electronic commerce transactions and secure e-mail, however the adoption of Virtual private Networks (VPNs) [7], secure IP (IPSEC) [8], Transport Layer security (TLS) [6] and Security Socket Layer (SSL) [5] will subject more of all communication to cryptographic processing. As secure communication bandwidth demands continue to grow, so too will the importance of efficient cryptographic processing. In this paper I focus on SSL protocol based on the open source called OpenSSL [2].

SSL protocol has been widely used in highly secure applications like e-commerce and banking systems. Li Zhao et. al [3] analyzed the SSL performance in secure web transactions. By presenting a detailed description of the anatomy of SSL processing they contributed architectural characteristics of crypto operations. They analyzed the CPI, path length and frequently used instruction in these crypto operations. They also presented an ISA/hardware support to improve the SSL processing. However the detailed architectural analysis of the SSL crypto operations has not been done.

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 processor 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 processor 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. 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 for cryptographic applications for processor designs. Haiyong Xie et. al. [4] have analyzed the architectural characteristics of crypto operations of Average algorithm and have proposed an acceptable architecture for cryptographic application specific uses which is quite different from the architecture of packet processing application specific processor. However keeping the view of SSL protocol and openly available SSL crypto algorithms, i intended to study the same architecture characteristic analysis and to study if there is a need for a different architecture for SSL type of crypto algorithms.

In this paper I considered 2 types of cryptographic applications; namely block ciphers and stream ciphers. The other forms of cryptography such as hash algorithm and public-key ciphers have not been studied yet. Through detailed timing simulation and profiling, I find that the cryptographic applications demonstrate different architectural properties than [4] .The architectural properties I studied include Instruction set characteristics, instruction level parallelism (ILP), and cache performance. I find that the instruction mix of the SSL cryptography operation consists of 60% arithmetic instructions and around 40% of memory reference instructions, and much lower percentage of branch predictions compared to SPECint95 [9], CommBench[10] and Average Crypto [4] .So the high precision branch prediction mechanism is not a need here. However as the average size of the blocks is larger than SPECint95, CommBench and Average Crypto, it is possible to take the advantage of instruction level parallelism much better. I see that an ILP of 8 holds good for most of the time as same to [4]. Compared to ILP, cache architecture have much significant effect on the overall performance as opposed to [4].I see that an instruction cache of 128 KB and a data cache of 32 KB are enough for most of the application not as the same as 16 KB in [4].Cache replacement strategy doesn't provide much improvement on the overall performance.

The above result will guide towards designing a special network processor for SSL crypto processing which is quite different from other crypto operations. The rest of the paper is structured as follows. Section 2 describes the selection of cryptographic algorithms. Section 3 describes the simulation environment and methodology. Section 4 represents 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 1 byte data. Section 6 represents 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 the paper.

2. Selection of SSL Cryptographic Applications

As my main aim is to run the OpenSSL crypto algorithms and analyze the architecture, I have chosen the algorithms according to the availability in the open source of OpenSSL version -0.9.7e. Also, 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 third 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, i choose 7 algorithms and their implementations for analysis. Four of them are block ciphers; One of them is data stream processing and other two are 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. I have chosen 4 block cipher types such as; 1.AES, 2.DES, 3.3DES, 4.IDEA

2.1.1 AES

AES (Advanced Encryption Standard), which is also named as Rijndael [11], is the standard of AES [12]. 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.

AES is a substitution-linear transformation network with 10, 12 or 14 rounds, depending on the key size. A data block to be encrypted by AES is split into an array of bytes, and each encryption operation is byte-oriented. AES's round function consists of four layers. In the first layer, an 8x8 S-box is applied to each byte. The second and third layers are linear mixing layers, in which the rows of the array are shifted, and the columns are mixed. In the fourth layer, sub key bytes are XORed into each byte of the array. In the last round, the column mixing is omitted. So the algorithm consists of 4 main steps: a substitution step, a shift row step, a mix column step and a sub key addition step. The substitution step consists of Sboxes. The shift row step consists of a cyclic-shifting of the bytes within the rows.The key addition is straight forward XOR operations between the data and the key.

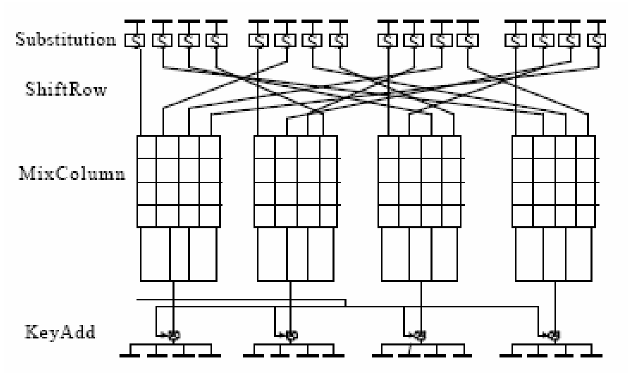


Fig1. (Architecture of Advanced Encryption Standard Data path)

Here I have chosen AES algorithm of key length of 128 bits and of cipher block chaining (cbc) encryption.

2.1.2 DES

The Data Encryption Standard (DES) cryptographic algorithm is based on a 128-bit block algorithm developed in the 1960s by IBM. It was designed to use a 64-bit key to encrypt and decrypt 64-bit blocks of data using a cycle of permutations, swaps, and substitutions. Encryption and decryption use the same key. A block to be encrypted is subjected to an initial permutation, then to a complex key-dependent computation, and then to a final permutation. The initial and final permutations take the 64-bit block and change the position of each bit in a pre-determined manner. The final permutation is the reverse of the initial permutation. A DES key consists of 64 binary digits of which 56 bits are randomly generated and used directly by the algorithm. The other 8 bits, which are not used by the algorithm, are used for error detection. The 8 error detecting bits are set to make the parity of each 8-bit byte of the key odd, i.e., there is an odd number of "1"s in each 8-bit byte. DES can operate in different modes like ECB (electronic Code Book), CBC (Cipher Block Chaining), CFB (Cipher Feedback), and OFB (Output Feedback). Here I have chosen the CBC kind of encoding using the 128 bits of key length.

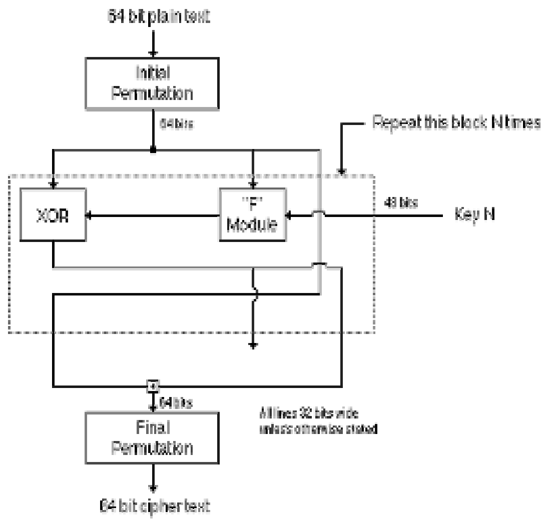


Fig2. (DES Block Diagram)

2.1.3 3DES

3DES [13] 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.

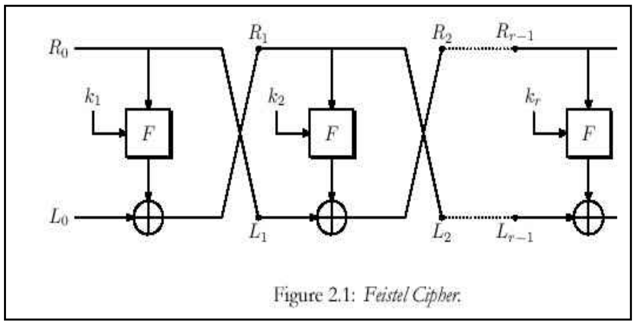
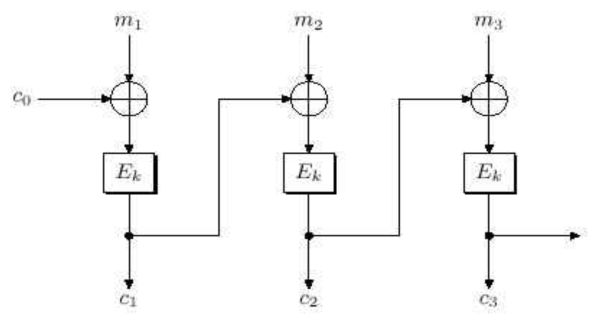


Fig3. (Basic 3DES algorithm Block Diagram)

The block of plaintext is split into two halves (L_0,R_0). each of which is 32 bits long. Also DES uses the original 56 bit key to generate 16 keys of 48 bits each (k_i). These sub keys are used in the 16 rounds. In each round, the function F is applied to one half using a sub key k_i and the result is XORed with the other half. The two halves are then swapped and the process is repeated. All the rounds follow the same pattern except the last one, where there is no swap. The final result is the cipher text (L_r,R_r). Hence the plaintext (L_0,R_0) is transformed to (L_r,R_r). In 3DES, we apply 3 stages of DES with a separate key for each stage. So the key length in 3DES is **168 bits**. I have chosen the 3DES algorithm with CBC kind of chaining and with 168 bits. Because the CBC is the most common mode of using DES/3DES. The CBC mode is represented below.



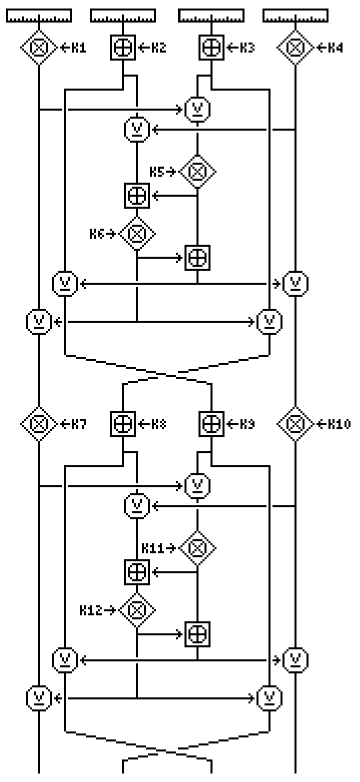
$$c_i = E_k(c_{i-1} \oplus m_i) \qquad m_i = c_{i-1} \oplus D_k(c_i)$$

Fig4. (CBC Mode of Operation of DES/3DES)

2.1.4 IDEA

IDEA [14] 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 selecting IDEA is that it is, on average, much faster than many other ciphers.

IDEA uses 52 sub keys, each 16 bits long. Two are used during each round proper, and four are used before every round and after the last round. It has eight rounds. The plaintext block in IDEA is divided into four quarters, each 16 bits long. Three operations are used in IDEA to combine two 16 bit values to produce a 16 bit result, addition, XOR, and multiplication. Addition is normal addition with carries, modulo 65,536. Multiplication, as used in IDEA, requires some explanation. Multiplication by zero always produces zero, and is not invertible. Multiplication modulo n is also not invertible whenever it is by a number which is not relatively prime to n. The way multiplication is used in IDEA, it is necessary that it be always invertible. This is true of multiplication IDEA style. Let the four quarters of the plaintext be called A, B, C, and D, and the 52 sub keys called K(1) through K(52). Before round 1, or as the first part of it, the following is done: Multiply A by K(1). Add K(2) to B. Add K(3) to C. Multiply D by K(4).Round 1 proper consists of the following: Calculate A xor C (call it E) and B xor D (call it F). Multiply E by K(5). Add the new value of E to F. Multiply the new value of F by K(6). Add the result, which is also the new value of F, to E. Change both A and C by XORing the current value of F with each of them; change both B and D by XORing the current value of E with each of them. Swap B and C. Repeat all of this eight times, or seven more times, using K(7) through K(12) the second time, up to K(43) through K(48) the eighth time. Note that the swap of B and C is *not* performed after round 8.Then multiply A by K(49). Add K(50) to B. Add K(51) to C. Multiply D by K(52). The intricacies of IDEA encryption may be made somewhat clearer by examining the following diagrams:



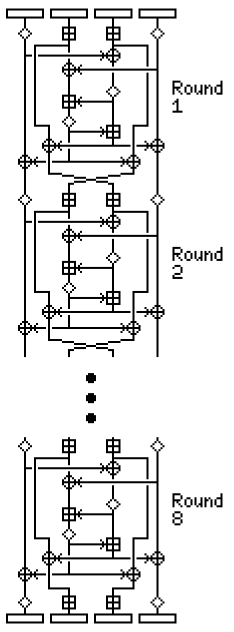


Fig 5a. *(Details)* Fig 5b. *(Overview)*

I have chosen the IDEA algorithm with 128 bits of key length and of CBC kind of operation.

Table 1: Selection of Block Ciphers

Type	Designer	Key Length(Bits)	Block Size (Bits)
AES	Rijmen	128,192,256	16
DES	Coppersmith	128	8
3DES	IBM	168	8
IDEA	Massey	128	8

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 plain text 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. I have chosen only 1 stream cipher i.e. RC4.

2.2.1 RC4

RC4 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.

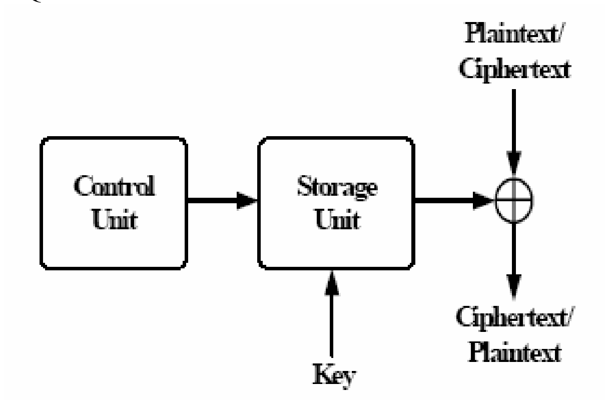


Fig 6. *(Block Diagram of RC4 Stream Cipher)*

RC4 uses a variable length key from 1 to 256 bytes to initialize a 256-byte array. The array is used for subsequent generation of pseudo-random bytes and then generates a pseudorandom stream, which is XORed with the plaintext/ciphertext to give the ciphertext/plaintext It works in Output Feedback (OFB) mode [15] of operation. There are two 256-byte arrays, S-Box and K-Box. The S-array is filled linearly, such as S0=0, S1=1, S2=2, ..., S255=255. The K-array consists of the key, repeating as necessary times, in order to fill the array. The RC4 stream cipher works in two phases. The key setup phase and the pseudorandom key stream generator phase. Both phases must be performed for every new key. Here I have chosen the OpenSSL RC4 algorithm with a key length of 128 bits and in OFB mode.

Table 2: Selection of Stream Ciphers

Type	Designer	Key Length(Bits)	Application
RC4	Rivest	8 to 2048 multiple of 8 bits; default 128 bits	SSL

2.3 Hash Algorithms

Hash algorithms are fundamentals to many cryptographic applications. Although widely associated with digital signature technology, the hash algorithm has a range of other uses. SHA-1 and MD5 are amongst the most widely known, trusted and used As OpenSSL has these two algorithms I have chosen both of them.

2.3.1 MD5

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.

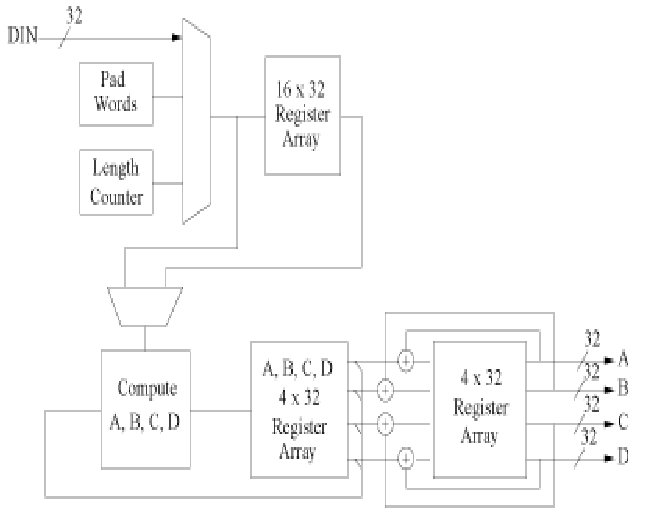


Fig7. (Block Diagram for the MD5 Megafunction)

This megafunction is a fully compliant hardware implementation of the MD5 Message-Digest Algorithm, suitable for a variety of applications. It computes a 120-bit message digest for messages of up to (264 – 1) bits. MD5 algorithm operates on message blocks of 512 bits for which a 128-bit (4 by 32-bit word) digest is produced. Corresponding 32-bit words of the digest from consecutive message blocks are added to each other to form the message of the whole message. Here I have chosen digest size of 128 bits and block size of 512 bits.

2.3.2 SHA1

SHA1 [17] 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.

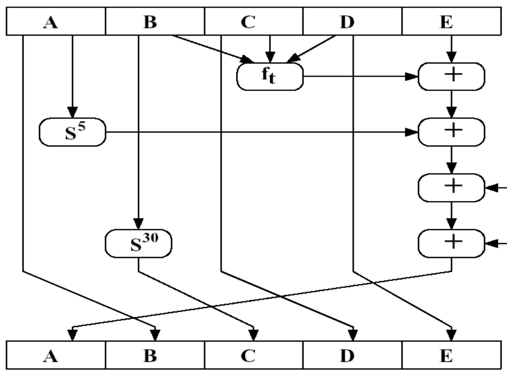


Figure 9.6 Elementary SHA Operation (single step)

Fig 8. (Elementary SHA operation: single Step)

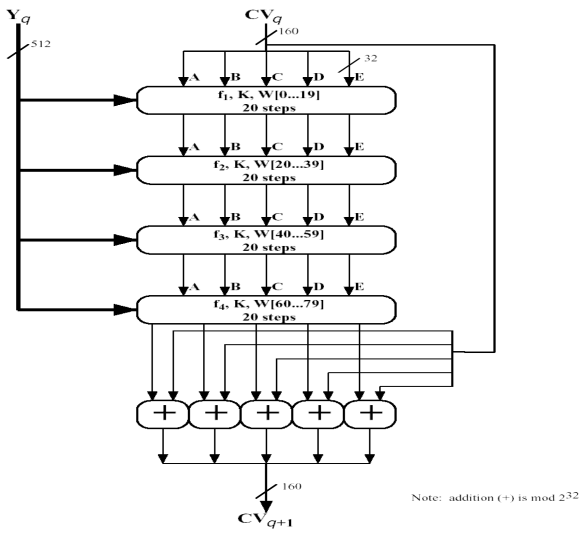

Y_q
CV_q
512
160
32
f_1, K, W[0...19] 20 steps
f_2, K, W[20...39] 20 steps
f_3, K, W[40...59] 20 steps
f_4, K, W[60...79] 20 steps
160
Note: addition (+) is mod 2^{32}
CV_{q+1}

Figure 9.5 SHA-1 Processing of a Single 512-bit Block (SHA-1 Compression Function)

Fig9. (SHA1 processing of a Single 512-bit Block)

I have chosen the SHA1 algorithm from OpenSSL source with 512 bits of block size, and 160 bits of digest size.

Table 3: Selection of Hash Algorithms

Type	Designer	Block Size(Bits)	Digest Size(Bits)
MD5	Rivest	512	128
SHA1	USA Security Agency	512	160

3. Simulation Environment

In this paper I focus the architectural properties of above described cryptography applications. I collected the above said algorithms from OpenSSL-0.9.7e [2]. After creating the library i made the separate codes for self execution for each of the 7 algorithms. Then i ported and ran them in the execution driven simulator, SimpleScalar version3.0 [1]. 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. The C compiler used is *gcc 2.7.2.3* (optimization level O2) coming with SimpleScalar. The O2 optimization level is selected for the reason that the compiler only performs optimizations that are independent of the target processors and does not exploit particular architectural features.

The algorithms are executed with a relatively large text file of 260 KB as well as a small file of 1byte. A key of 128 bits is used with all the block and stream ciphers except 3DES, which is 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 bimodal 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.

Table 4: Simulator Parameters

Type	Default values	Variable parameters
Processor Speed	2.4 GHz	NIL
Fetch Width	8 Instructions	1,2,4,8,16,32
Pipeline	11	NIL

Depth		
Functional Units	6IntALU,6IntMult, 2FpALU,2FpMult	IntALU:1,2,4,8 FpALU:1,2,4,8
Issue Width	8Int, 4Fp	NIL
Issue Queue Size	64 Int, 32Fp	NIL
Load/Store Queue Size	64 LQ, 64 SQ	NIl
Branch Predictor	Bimodal	Not Taken, taken, 2level, bimodal, combinational
Branch target buffer	1K Entry, 4way	NIl
Branch mispredict penalty	9	NIL
L1 Instruction Cache	64KB, 2way, 32byte Block line, 1cycle latency, l replacement policy	→ change the cache size:4,8,16,32,64,128,256 KB → Change the Block Size: 8,16,32,64 bytes → Set Associativity: 1,2,4,8,16 → Replacement Policy: l, f, r
L1 Data Cache	64KB, 2 way, 64byte Block line, l replacement policy	→ Cache size: 4,8,16,32,64,128,256 KB → Block Size: 8,16,32,64 bytes → Set Associativity: 1,2,4,8,16 →Replacement Policy:l, f, r
L2 Cache	512 KB, 2Way, 64B Line,	NIl
L3 Cache	4Mb, 4 Way, 64B line	NIl
UL2 unified Cache	1024 KB, 4 Way, l replacement,64 Byte line size ILP of 4.	→ Cache Size: 4,8,16,32, 64, 128, 256, 512, 1024, 2048 →Replacement Policy: l, f, r

4. Instruction set characteristics

The instruction set characteristics give an indication on the types of instructions executed and their frequencies in the programs. Figure 10 presents the instruction mix profile and frequencies for the implementations of all the selected algorithms, SSLcrypto algorithms, averages algorithms [4], SPECint95 programs and CommBench programs. The average instruction mix of these cryptographic programs shows great differences from that of both SPECint and CommBench programs.

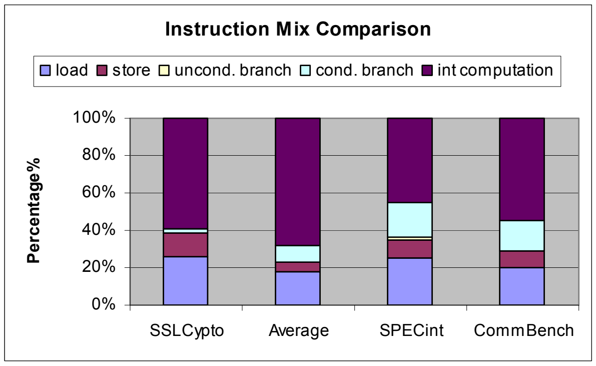


Fig10(a).Comparisons of instruction mix

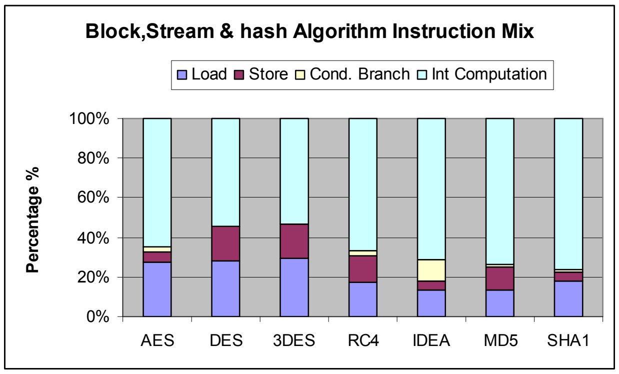


Fig10(b).Block and Stream Ciphers Instruction mix

Figure 10(a) depicts the averages of block ciphers, stream ciphers, hash algorithms of OpenSSL named as SSLCrypto. The "Average" shows that of [4]. The following points out the differences: 1).The SSLCrypto has higher percentage of arithmetic instructions than CommBench and SPECInt and lower than the Average. It is clear from the above graph that the SSLCrypto applications are more computational intensive than SPECInt and CommBench but less than Average. This computation may consume most of the network processor's computation power. (2).The SSLCrypto has significant amount of memory reference compared to Average algorithms. As opposed to [4], here It means that SSLCrypto applications are more or less memory reference bounded and that we may need a less complicated memory system with significant amount of hit rate. (3).The SSLCrypto programs have much lower percentage of branch instructions, which is 3.42% in average compared to 20% of 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.

Figure 10(b) shows the instruction mix profile for block ciphers, stream ciphers, and hash algorithms. The following observations are important: (1). Among all the selected block ciphers, only DES and 3DES have similar and high percentage of memory reference instructions (45% and 47% 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 (11%) 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).The selected stream ciphers and hash algorithms have quite different instruction mix properties from SPECint95. They all have significant percentage of both memory reference (up to 25% each) 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.

5. Computational Complexity

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 11 depicts the computational complexity for each of the block ciphers, stream ciphers and hash algorithms. We can see that 3DES spends much more 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 the figure. Stream ciphers exhibit more like hash algorithms rather than block ciphers in perspectives of cycles spent per byte operation.

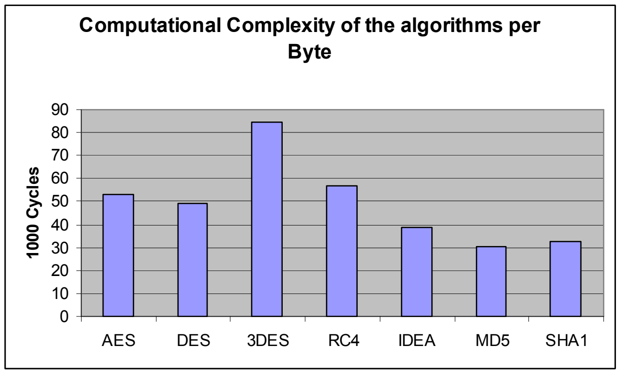


Fig11. Computational Complexity of the selected Algorithms

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 verify 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 12(a) shows the impact of changing the number of integer ALUs keeping the default value of FP 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 26%, 37%, and 40% for Block, stream and hash kind of algorithms respectively when the number of ALUs increases from 1 to 2, and by 6%, 10%, and 5% 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%.

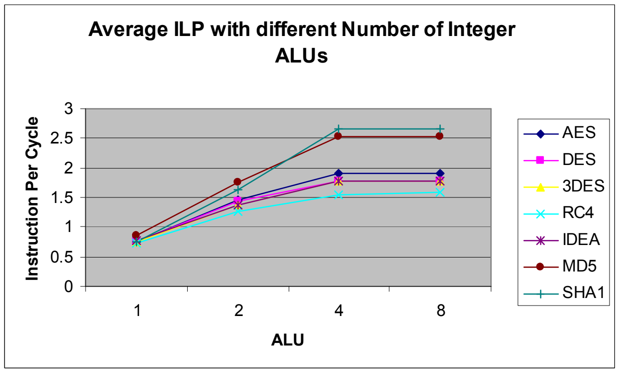


Fig12(a). Impact of ALU on Average ILP

Figure 12(b) 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 26%, 37%, and 40% for block, stream and hash kind of algorithms respectively after the size of the instruction fetch queue changes from 1 to 2. And the same increases as 6%, 10% and 5% respectively if the instruction fetch queue changes from 2 to4. After that, the performance improvement is less than 2% when we double the size of the queue.

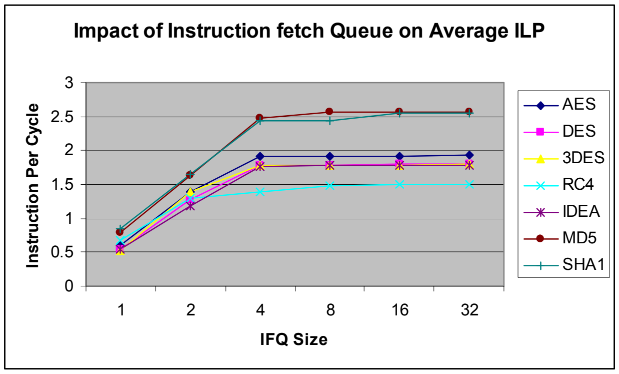


Fig12(b). Impact of IFQ on average ILP

Figure 12(c) 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 integer ALUs and 8 FP 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 less than 1% 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.

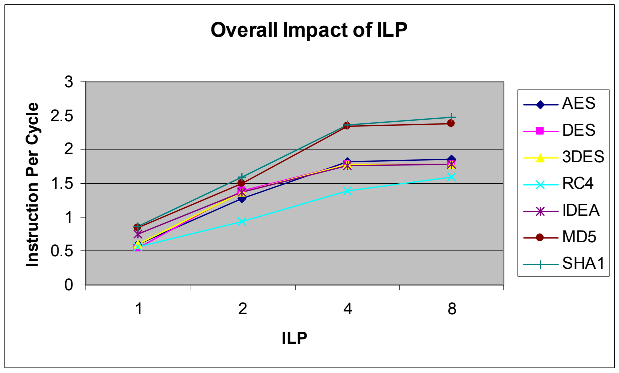


Fig12(C). Overall Impact of ILP

7. Branch Prediction

Branch prediction does not seem to be as important as instruction level parallelisms. One of the reasons has been stated in Section 4: the percentage of conditional branch instructions is 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 13 presents the prediction hit rate of each prediction mechanism available in the simulator for all the crypto applications I have considered. It can be observed from Figure 13 that most of the conditional branches are either bimodal or combinational or 2level.Each kind of these branch prediction gives us a Hit rate more than 60%. Therefore, sophisticated branch prediction mechanisms like Bimodal or Combinational or 2level are some short of requirement for these cryptographic programs. A simple static branch prediction mechanism with the branches being predicted always taken or not taken may not be sufficient.

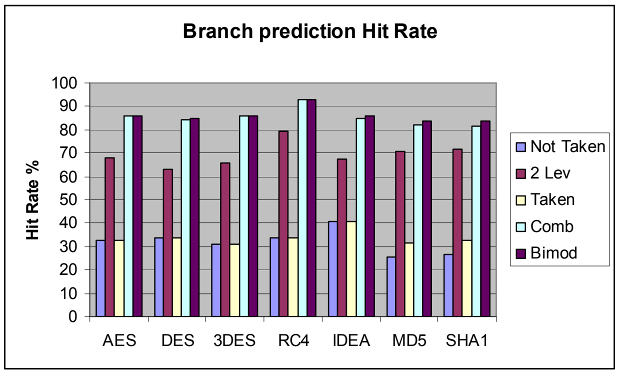


Fig13. Average Branch Prediction Hit Rate

8. Cache Behaviors

Cache behaviors are other important considerations for the design of network processors. From Section 4, it is found that the memory reference instructions account for around 45% of all the instructions executed for cryptographic programs. This means that proper cache architecture is required 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 14 shows the results for instruction cache behaviors by changing such cache parameters as cache size, line size, set associative, and replacement strategy. Figure 14(a) presents the simulation results of cache performance when the cache size is changed. It can be seen that these crypto algorithms require a big L1 cache size for getting a very low miss rate. L1 instruction Cache size of 128 KB is enough to achieve miss rates at the lowest value possible for all the three kind of algorithms. This figure also shows that the miss rate of all the three kind of algorithms 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 different with that of SPECint programs, which have an average miss rate of 2.2% and 1% for 8KB and 16KB instruction caches respectively.

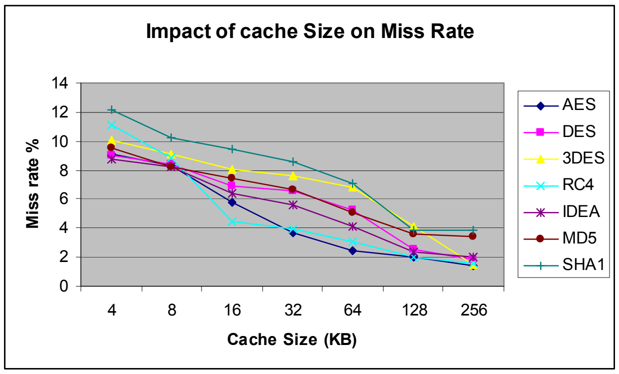


Fig14(a). Impact of Cache Size

In figure 14(b), 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 all kind of crypto algorithms. It is seen that a line size of 32 bytes is enough for block, stream and hash kind of ciphers to achieve miss rates less than 10% at a lower constant level. However it is observed that after 32 bytes if we increase more there is very little effect on the miss rate and it doesn't help much.

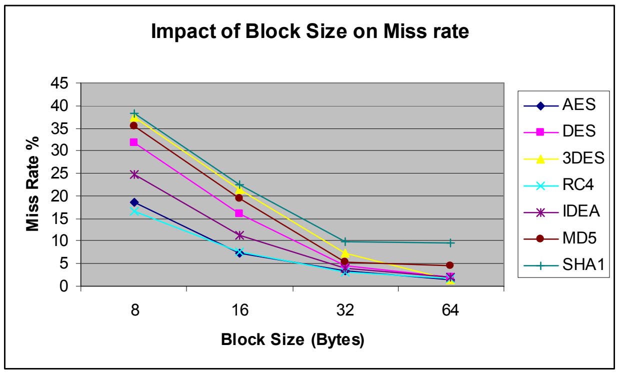


Fig14(b). Impact of Block Size

Figure 14(c) shows the 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. All of these crypto algorithms need a 2-way set associative cache to obtain the best performance.

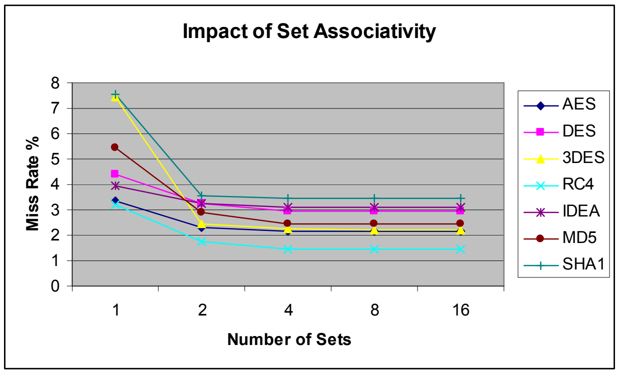


Fig14(c). Impact of Set Associativity

Figure 14(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. Random choosing policy gives the maximum miss rate.

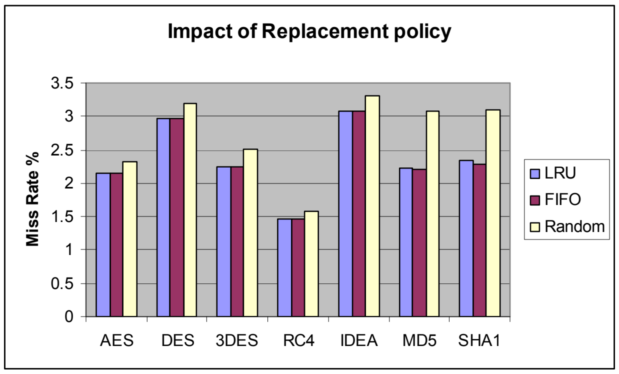


Fig14(d). Impact of replacement strategy

Combining the above four observations into one, we can safely conclude that a direct-mapped 128KB instruction cache with line size as 32 bytes, 2-way set associativity is enough for most of the cryptographic applications to obtain the best performance. LRU or FIFO replacement strategy contributes almost the best performance, 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 15 shows the simulation results by changing cache size, line size, set associative, and replacement strategy.

Figure 15(a) shows the impact of cache size on the miss rate. A cache of 32KB can reduce miss rates of block and hash and stream ciphers to less than approximately 10%; which is the best achievable, in average. The miss rate cannot be reduced any more even with larger cache sizes for stream ciphers and block 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. However with a cache size of 64KB or even more, we can get a lower miss rate for hash algorithms.

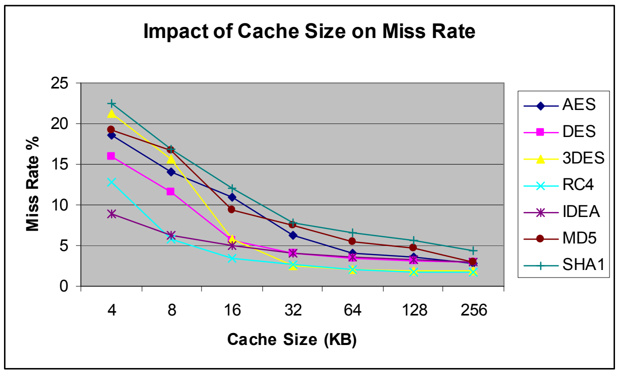


Fig15(a). Impact of Cache Size

Line sizes have different impacts on miss rates for the three types of applications, as shown in Figure 15(b). A 4-way 16KB data cache with a line size of 64 bytes has less than 3% miss rate for hash and most of block ciphers except DES. The larger the line size is, the lower miss rates for all kind of algorithms. 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 3% with 64KB data cache for selected program, while the miss rates of SPECint and CommBench benchmark programs (except for ZIP, FRAG, and DRR applications) are approximately 4% and below 1%, respectively.

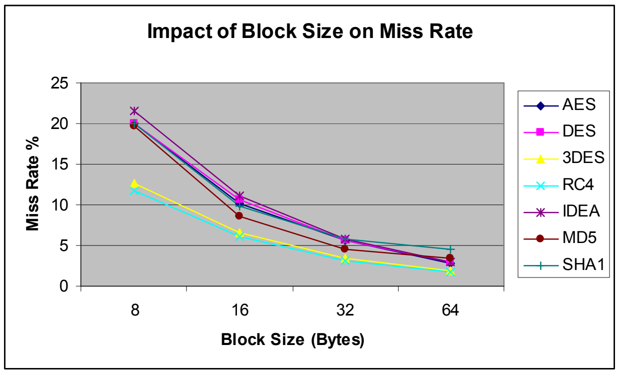


Fig15(b). Impact of Block Size

Hash programs and most of the block ciphers do not have high demands for set associativity, as shown in Figure 15(c). A direct-mapped cache of 32KB is enough to achieve miss rate lower than 10% for all kind of ciphers. Compulsory miss dominates cache miss rate of SHA1 and MD5. AES and DES needs 2-way set associative data cache to obtain miss rate at its best level.

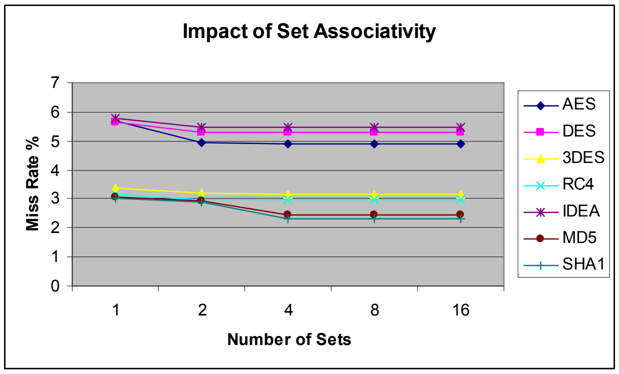


Fig15(c). Impact of Set Associativity

Cache replacement strategy does not have greater impact on the cache miss rate. LRU replacement strategy can be replaced with FIFO or with Random type replacement for all kind of ciphers without increasing miss rate by more than 1%.

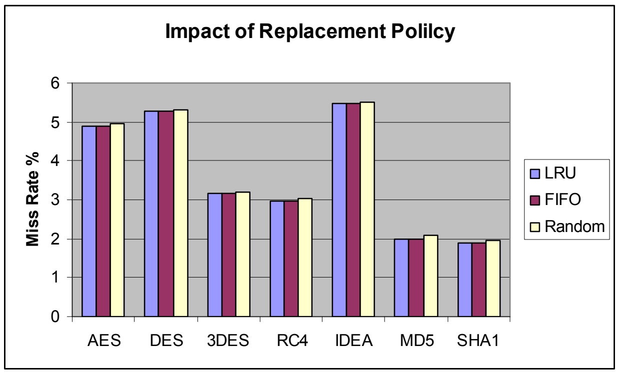


Fig15(d). Impact of Replacement Strategy

8.3 L2 Unified Cache Behaviors

I have simulated the L2 unified cache behaviors for all the applications as well, the results of which are shown in Figure 16. Due to most of the memory references are absorbed by L1 caches, the miss rate of L2 cache is high, as shown in Figure 16(a). Figure 16(a) indicates that only an L2 cache of 512KB is needed to achieve the lowest miss rate. L2 cache is also used to decrease the latency of memory references.

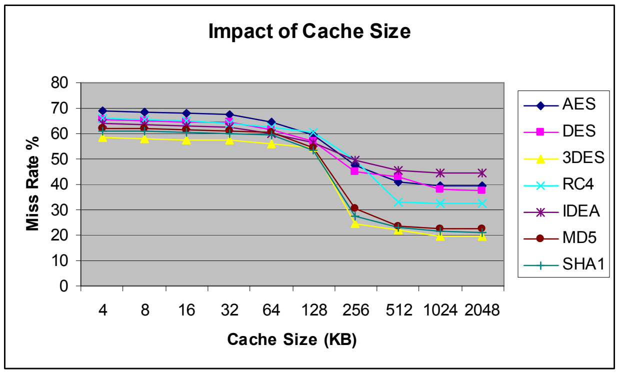


Fig16 (a). Impact of Cache Size

Figure 16(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 on all the three different kinds of crypto ciphering.

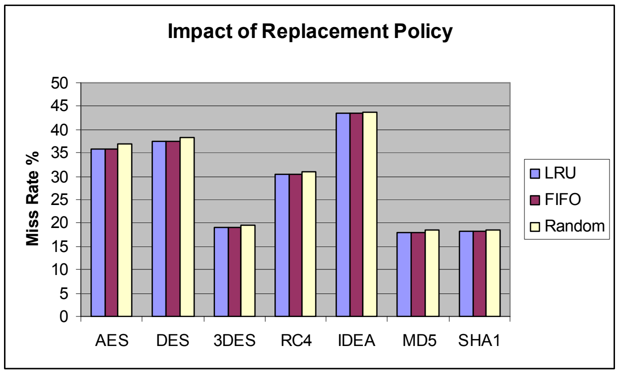


Fig16(b). Impact of Replacement Strategy

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 i selected seven widely used cryptographic programs from OpenSSL open source and analyzed their architectural demands for network processors. I focus on the generalized architectural properties of the selected cryptography applications that are applicable to all network processor architectures.

Taking the advantage of the SimpleScalar tool set to simulate the MIPS-like processor architecture, I learnt the performance of these selected algorithm implementations. Based on the computational complexity I observed, I 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.

I studied the architectural properties including instruction set characteristics, instruction level parallelisms, branch prediction, and cache behaviors for these seven programs. I find that the instruction mix of these programs has major differences from that of SPECint95 benchmark programs and from that of the Average kind of programs as mentioned in [4]. Cryptographic algorithm implementations have much higher percentage of arithmetic instruction, much lower percentage of branch instructions and quite bit of 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 require advanced kind of predictions like bimodal or combinational or 2level. A simple branch prediction with all the branches being predicted taken or not taken is not enough for comparable performance. Most of the cryptographic applications have an ILP of 4. The high data dependence is the main limitation to exploit more ILPs. Memory system has significant amount of important effect on the overall performance. I find that most of the applications need L1 Instruction cache of 128KB and L1 Data cache of 32 KB. Its only needed 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 basically compared to those of the paper [4] to get a clear picture of general average kind of algorithms vs. OpenSSL crypto algorithms which is helpful to the design of network processors considering OpenSSL crypto engines. 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.

Observation:	**Li's Analysis (Widely available crypto Algorithm)**	**My Analysis (OpenSSL Crypto Algorithms)**
Instruction Mix:	23% Memory Reference 60% Arithmetic computations	40-45 % Memory Reference 68% Arithmetic Reference
Cycles per Byte of Computation	Block:80 Stream: 20 Hash: 18	Block: 55 Stream: 55 Hash: 30
ALU Vs IPC IFQ Vs IPC ILP Vs IPC	Best when 4 ALUs Best when IFQ is 4 Best when ILP is 4	Best when 4 ALUs Best When IFQ is 4 Best when ILP is 8
Branch prediction technique	Simple technique (taken or not taken)	Complex technique (Bimodal or Combinational)
L1 Instruction cache parameters	16KB cache size, 8 bytes of line size, 4 way set, 1 replacement	128KB Cache size, 32 bytes line size, 2 way sets, 1 replacement
L1 Data Cache parameters	32KB cache, 8bytes of line size, 2 way sets, 1 replacement	32KB cache Size, 64 bytes line size, 2 way set, 1 replacement
UL2 Unified cache parameters	64 KB cache Size, 1 kind of replacement policy	512 KB cache size, 1 kind of replacement policy

Acknowledgement

I would like to thank ***Li Zhao*** *for her constant help towards making me understand the OpenSSL crypto programs and usage of simplescalar and also for helping me to finish this small project within the small amount of time period.*

References:

1. *SimpleScalr Tool Set* *http://www.simplescalar.com/*
2. OpenSSL 0.9.7e http://www.openssl.org/
3. *Anatomy and Performance of SSL processing by Li Zhao, Ravi Iyer, Srihari Maikeneni, Laxmi Bhuyan.*
4. *Architectural Analysis of Cryptographic applications for Network processors by Haiyong Xie et. al.*
5. A.O. freier, P. karlton, P.C. Kocher, "The SSL protocol, V3.0", IETF Draft, http://wp.netscape.com/eng/ssl3/3-spec.htm
6. *T. Dierks, C. Allen, The TLS protocol version 1.0, http://www.ietf.org/rfc/rfc2246.txt*
7. *P. Ferguson and G. Huston. What is a VPN?http://www.employees.org/ferguson/vpn.pdf, 1998*
8. *R. Atkinson. Security architecture for the internet protocol. IETF Draft Architecture ipsec-arch-sec00, 1996*
9. *Standard performance Evaluation Corporation SPEC CPU95 version 1.10, August 21, 1995*
10. *T. Wolf, M. Franklin, CommBench: A telecommunication benchmark for Network Processors, IEEE International Symposium on Performance Analysis of systems and software, Austin TX, Apr. 2000*
11. *J. Daemen, V. Rijmen, AES Proposal: Rijndael, http://csrc.mist.gov/encryption/aes/round2/AESAlgs/Rijndael, 1999*
12. *Advanced Encryption Standard (AES) Development Effort, US Government, http://csrc.nist.gov/encryption/aes/*
13. D. Davis, W. Price, *Security for Computer Networks*, Wiley,

14. *X. Lai, On the Design and Security of Block Ciphers, Hartung-Gorre Veerlag, 1992*

15. *"Recommendation for Block Cipher Modes of Operation. Methods and Techniques". National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce.* *http://csrc.nist.gov/publications/nistpubs/800-8a/sp800-38a.pdf*

16. *R. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, April 1992*

17. *A. Menezes, P. van Oorschot, S. Vanstone, Algorithm 9.53 Secure Hash Algorithm - revised (SHA-1), Handbook of Applied Cryptography, CRC Press, 1997*