Performance Measurement of AES Crypto Algorithm in Microcode Environment of IXP2400 Platform

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

Abstract

In this report first i highlight the mathematical properties of 128-bit AES encryption and then keeping these properties in view i port the algorithm in microcode environment of the IXP2400 platform. Then I do some performance measurement based on the memory optimization and thread optimization. I apply a pipelined and best parallelized approach of implementing the AES algorithm. I also studied the crypto unit of IXP2850 and measured the performance of 3DES algorithm performed on this crypto unit. Finally I try to compare the performance results of the algorithm with the version written in MicroC by authors from Tsinghua University [1].

1. Introduction

Data Security is going to have increased importance as the Internet continues to gain popularity in ecommerce activities. Therefore, Security is given its own category in EEMBC Benchmarks. The Security category includes several common algorithms for data encryption, decryption and hashing. One algorithm, *rijndael*, is the new Advanced Encryption Standard (AES). AES is computationally intensive and furthermore, networks must apply it to every packet crossing a secure link. To address this problem and add security functions to network processors, a straightforward approach—one that achieves comparable performance—is to implement them in hardware. Unfortunately, many security chips or coprocessors can only handle a few algorithms, while most Internet security standards allow flexibility in algorithm selection. In addition, cryptographic hardware is not cheap or readily exportable. To compensate for these drawbacks, vendors often build security functionality directly into the same silicon as the network processor. But this method is still inflexible in that it cannot implement multiple algorithms (the Intel IXP2850 network processor, for example, has only two block ciphers and one hash algorithm). Besides, data must traverse shared memory and buses at least four times. So the resource contention problem actually prevents those inline cryptographic units from reaching their claimed performance. Hence, the implementation of cryptographic applications on network processors via software is still necessary. Clearly, the most challenging work for software implementations is to provide performance guarantees, for instance, covering the handling packets at high speed. On general-purpose processors, traditional optimization techniques emphasize improving instruction level parallelism.

This report studies the architectural properties for AES cryptographic algorithm on an actual Intel IXP2400 network processor. The rest of the report has been organized as below. Section 2 presents the over view of the 128-bit AES algorithm. Section3 describes some current proposition of the parallel and pipelined implementation of AES algorithm. Section 4 describes the Intel Ixp2400 architecture. Next section describes the experiments and some results. And section 6 concludes the paper.

2. AES Algorithm

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.

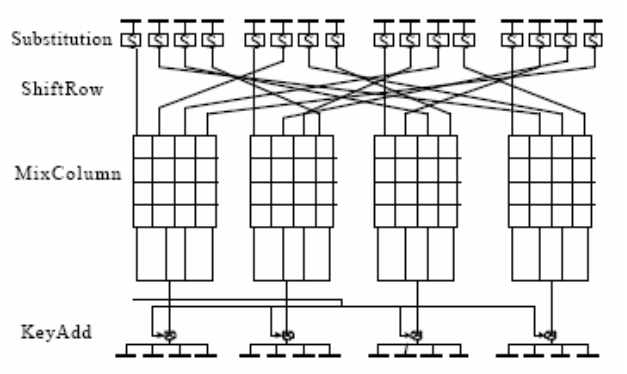

Fig1. (Architecture of Advanced Encryption Standard Data path)

Here I have chosen AES algorithm of key length of 128 bits and of cipher block chaining (cbc) encryption. The Rijndael algorithm is selected by National Institute of Standards and technology (NIST) as a new Advanced Encryption Standards (AES). The Rijndael algorithm is based on arithmetic in a finite Galois field, *GF* (2^8). We consider only operations using a 128-bit cipher key and 128-bit data blocks, although the algorithm scales to accommodate different key and data block sizes. The algorithm requires 11 rounds. Each round operates on the *state*, a 4 x 4 matrix of 8-bit values. Each round involves up to four basic transformations:

1. Byte Substitution (ByteSub) – Each *state* byte is replaced with an affine transformation of it's multiplicative inverse in *GF* (2^8).

2. ShiftRow – Each row of the *state* is cyclically shifted by *i* bytes, where *i* is the row number.

3. MixColumn – Each column of the *state* is treated as a polynomial over *GF* (2^8), and multiplied by the fixed polynomial, c(x) = '03'x^3 + '01'x^2 + '01'x + '02', modulo x^4+1.

4. AddRoundKey – A bitwise XOR of the *state* and the *round key*. The *round key* is an extension of the cipher key unique to each round. The algorithm operates in eleven rounds. The first round performs only the AddRoundKey transformation, while the middle 9 rounds perform all four transformations. The final round performs the ByteSub, ShiftRow, and AddRoundKey transformations, omitting the MixColumn operation.

The main loop of the Rijndael AES algorithm implementation is implemented with a series of table lookups and functionally reduced to operations that is be of the form:

```
C or Java code:
    a[i] = (T1[(t[ i      ] >> 24) & 0xFF] ^
           T2[(t[(i + s1)] >> 16) & 0xFF] ^
           T3[(t[(i + s2)] >>  8) & 0xFF] ^
           T4[ t[(i + s3)]        & 0xFF] ) ^ Key[r][i]
```

Fig2. (C or Java Code of the AES algorithm)

This code can be implemented as combinations of the following basic operations:

1. BYTE SELECT => t[i+...] is a byte select
2. TABLE LOOK UP => T1[xxx] (the byte selected is the index for the table)
3. SHIFT/MASK => >> 24,16,8 & 0xFF
4. XOR => ^ (bit-wise logical Exclusive OR)

Where, Key [][] is the extended encryption key.

3. Parallel and Pipelined Approach to AES

The paper [2] presents a single-chip parallel architecture for advanced encryption standard (AES). The proposed architecture uses the thread approach, which integrates fully pipelined parallel units, that process 128 bits/cycle and quadruples the data throughput. The threads architecture allows the reduction of the clock rate by a factor of four, while maintaining the data throughput, and consumes lesser power. The prototype runs at data rate of 7.68 Gbps on a Xilinx xc2V1500 Virtex-II FPGA. The data rate shows that the proposed thread approach produces one of the fastest single-chip FPGA implementation currently available. In addition, the proposed architecture is scalable to 192, 256 and higher bits.

The data pipeline is designed noting that the complex middle round transformations can be reduced to a series of look-up-table operations and bitwise XOR operations, using the T-table methods outlined in [6]. Since the look-up-tables can be mapped into the dedicated Virtex-II SelectRAM block RAM (BRAM) resources, and bitwise XORs are easily implemented in Virtex-II configurable logic blocks (CLBs), the T-table optimizations are ideal for our implementation. One middle round of the pipeline is illustrated below (Figure 3). The individual round is repeated 11 times to form the entire pipeline (the specifics of the first

and last rounds are not explained here, though they deviate from the middle rounds).

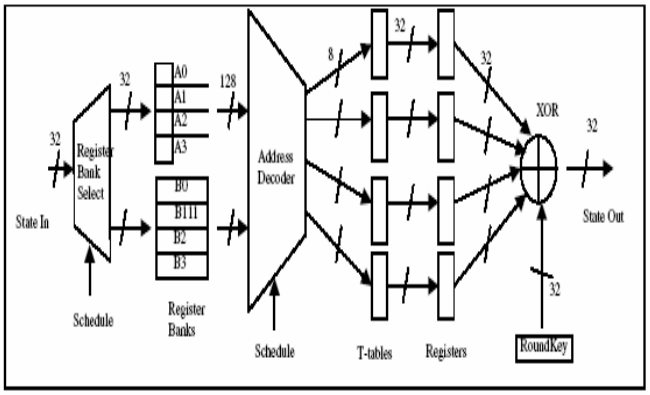

Fig3. Rijndael Middle Round

For each round, every cycle we take in 32-bits and output 32-bits (a column of the input and output state matrices, respectively). Therefore, four cycles are required per 128-bit data block. Each 32-bit output is the result of XORing four 32-bit data values (one from each of the four T-tables) with a 32-bit word of the key round. The correct T-table values are determined from the state. Since all 128-bits of the state are required for T-table addressing, and the design is pipelined, an entire 128-bit state matrix is registered (taking 4 cycles) in one of two register banks (A, comprised of 32-bit registers A0, A1, A2, and A3, or B, comprised of 32-bit registers

4. Intel IXP2400 Architecture

Closely examining the IXP2400's hardware architecture, shown in Figure 1, helps to elucidate the implementation and optimization. IXP2400 is a member of Intel's second-generation network processor family. Like its predecessor, IXP1200, IXP2400 is also a 32-bit RISC-based multicore system that exploits the system-on-chip (SOC) technique for deep packet inspection, traffic management, and high-speed forwarding. The 600-MHz XScale core is a general-purpose processor used for exception handling, slow-path processing, and other control plane tasks. The eight 600MHz micro engines (MEs) are data plane PEs, connected in two clusters. MEs in the same cluster share a common command bus, which they use to forward memory and I/O requests to other relevant units. Adjacent MEs (referred to as *next neighbors*) connect together in a pipeline with their nearest neighbors to provide one-way communication.
Intel designed the 32-bit media switch fabric and PCI interface to connect to a media access controller and external devices. Unlike general-purpose processors—which rely heavily on a large cache and efficient cache replacement policies to improve performance—the lack of locality in packet processing has forced network processor designers to come up with innovative memory and PE architectures. For example, IXP2400 has a distributed, shared memory hierarchy that supports two types of external memory: RDRAM and quad-data rate SRAM. In addition, the processor includes a 16-Kbyte, on-chip, scratch SRAM (shared among all MEs), plenty of registers, and a small amount of local memory per ME. In Table 1, we list the capacity, transfer size, reference latency, and the typical usage of these registers and memories. As shown in the table, memory access latencies have not kept pace with ME processing speed. For instance, the minimum read latency for fastest shared SRAM (scratch) is 100 ME cycles. To solve this problem, the IXP architecture uses eight *zero-thread-switching-overhead* hardware hard ware threads for interleaved operation—one thread does computation while others block, waiting for memory operations to complete. Thread swapping can be software controlled, and MEs can perform asynchronous memory and I/O operations using multiple signals that indicate the completion of these references. Moreover, each ME supports a single cycle arithmetic logic unit (ALU) with shifter, a multiply unit, and other specially designed I/O instructions.

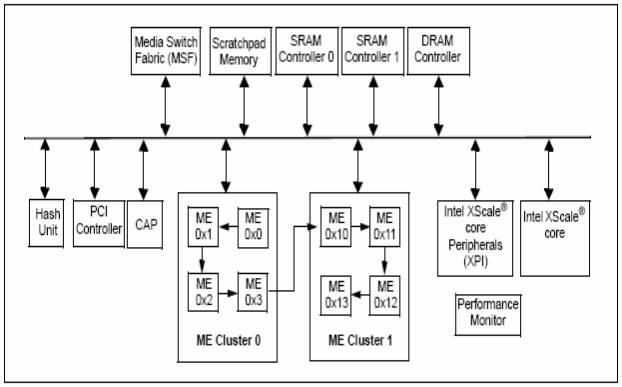

Fig4. (IXP2400 Chassis Concept Block Diagram)

Name	SizeBytes	Transfer Size(Bytes)	Reference latency in cycles
GPR/ME	**256*4 bytes**	**4**	**1**
TR/ME	**512*4**	**4**	**1**
NNR/ME	**128*4**	**4**	**1**
LM/ME	**640*4**	**4**	**3**
Scratch	**16K**	**4**	**60**
SRAM	**64M**	**4**	**90**
DRAM	**1G**	**16**	**120**

Table1. (Characteristics of IXP2400 register and Memories)

5. Experiments and Results

To observe the architectural characteristics of AES cryptographic algorithm and its utilization of internal resources and to detect performance bottlenecks, i conducted experiments under Workbench 4.1, a cycle accurate IXP2400 simulator. My experiments covered 600MHz ME configurations, 200-MHz SRAMs, and 400-MHz RDRAMs. I compiled the source code using the Intel IXP MicroCode Assembler 4.1 instruction sets, which offers the basic instruction and language optimizations. I used up 1 MEs (all 8 threads) in one ME cluster, as described earlier. The algorithm includes several loops that consume the vast majority of all processing time. Other operations, such as key scheduling and able initialization, are pre calculated at set-up time by slow path processors (such as the XScale in the IXP2400). Thus, the work focuses on the algorithm characteristics of the inner loops, while many related statistics include other portions, even those that contribute little to overall performance. Some of the results found from the simulation are plotted as below.

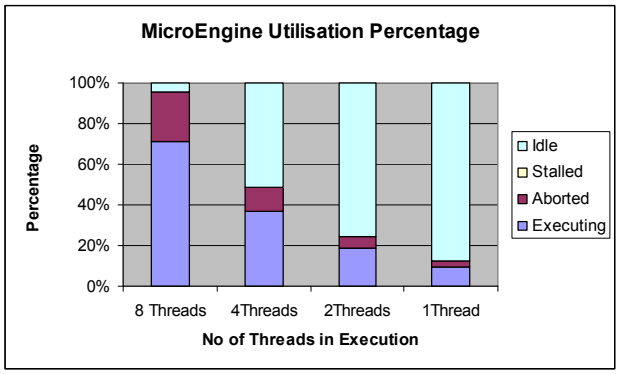

Fig5. MicroEngine utilization Percentage

Fig5 describes the utilization of 1 Micro Engine in percentage keeping threads as a parameter. I change the number of threads in a single micro engine and observe the idle time, stalled time, aborted time and execution time of that particular micro engine. It's obvious from the above graph that as AES is better performed in threaded platform rather than 1 thread. This is because the implementation of the algorithm is having significant memory accesses. And each time a thread accesses a memory it shifts the control to the next thread and thus it enhances the execution percentages of the micro engine.

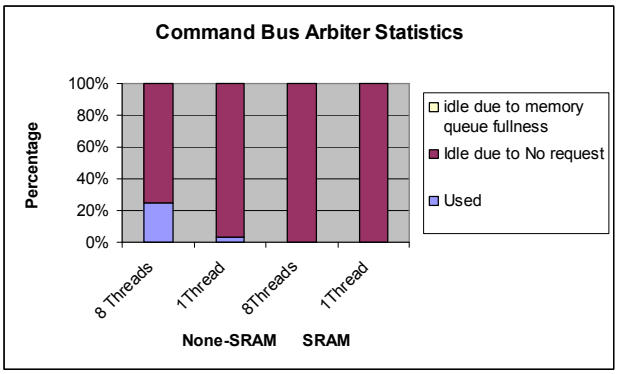

Fig6. Non-SRAM and SRAM Command Bus Arbiter

Fig6 depicts the SRAM command bus arbiter statistics for a single micro engine under different threads. Its observed that the usage of None-SRAM is higher than the usage of SRAM. As I have implemented the most of the memory operation in Scratch pad and DRAM so is the reason to justify the above graph.

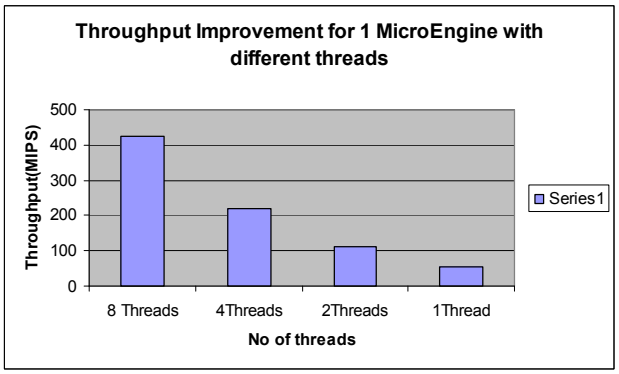

Fig7. Throughput of 1 Micro Engine with different number of threads

I performed the memory throughput across different threads keeping the micro engine number fixed at 1 and plotted the graph as shown in the fig7. It's observed that the throughput increases if we increase the number of the threads.

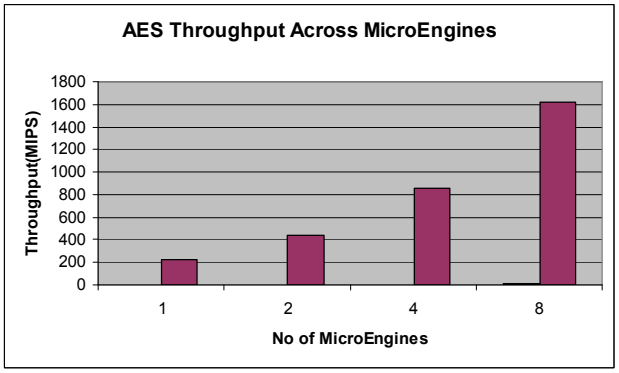

Fig8. Throughput with different number of Micro Engines.

I executed the algorithm in different number of micro engines. As obvious it is, the throughput increases with the increase number of micro engine.

9. Conclusions:

I coded the AES algorithm in assemble language of IXP2400 Intel Network processor and did some performance measurement on the basis of throughput, memory utilization and micro engine utilization. The results clearly demonstrate that crypto algorithms can best be implemented in network processor with the usage of multithreaded architecture. And also its observed that parallelization and pipelining in AES crypto algorithm can be implemented to some extent in IXP2400 Network processor which is my next focused area of work.

Acknowledgement

I would like to thank Yan Luo and Chris Baron for their kind help and support to make me understand the architecture and working style of the Intel IXP Network processor.

References:

1. *"Optimization and Benchmark of Cryptographic Algorithms on Network Processors"by Zhangxi Tan et. al, IEEE Micro, September/October 2004 Vol 24, No5, PP55-69*

2. *"A novel Pipelined Threads Architecture for AES Encryption Algorithm"by Mehboob Alam et al, ASAp'02, IEEE*

3. *"High Throughput, Parallelized 128-bit AES Encryption in a resource-Limited FPGA"by Christopher Caltagirone et. al, SPAA'03, ACM.*

4. *"AES Finalist Algorithm: The Rijndael Block Cipher" by Mel Tsai, University of California, Berkeley.*

5. *"IXP2400_IXP2800_PRM" A reference manual for Assemble code in Intel IXP2400 platform Available with Intel IXP simulator version 4.1.6.*

6. *[4] "The Rijndael Block Cypher: AES Proposal", Joan Daemen, Vincent Rijmen. First AES Candidate Conference (AES1), August 1998.*