

# New Approach for Advanced Encryption Standard (AES) Engines for Many-Core Processor Arrays

Krishnakumar .A<sup>1</sup> J. Vijaya<sup>2</sup> Dr. T. K. Shanthi<sup>3</sup>

<sup>1</sup>PG Scholar <sup>2,3</sup>Associate Professor

<sup>1, 2,3</sup>Department of Electronics and Communication Engineering  
<sup>1, 2, 3</sup>TPGIT, Vellore.

**Abstract**---The Advanced Encryption Standard (AES) has been lately accepted by NIST as the symmetric key standard for encryption and decryption of blocks of data. In encryption, the AES accepts a plaintext input, which is limited to 128 bits, and a key that can be specified to be 128 bits to generate the Cipher text. By exploring different granularities of AES as a onetime one Processor (OTOP), Small Encryption, Parallel Mixed Column and Full Parallelism we map these implementations on a field programmable gate array system. In comparison with published AES cipher implementations on general purpose processors. The proposed design has occupied less area and small delay.

## I. INTRODUCTION

With the development of information technology, protecting sensitive information via encryption is becoming more and more important to daily life. In 2001, the National Institute of Standards and Technology (NIST) selected the Rijndael algorithm as the Advanced Encryption Standard (AES) [1], which replaced the Data Encryption Standard (DES) [2]. Since then, AES has been widely used in a variety of applications, such as secure communication systems, high-performance database servers, digital video/audio recorders, RFID tags, and smart cards.

This paper presents various software implementations of the AES algorithm with different data and task parallelism granularity, and shows that AES implementations on a fine grained many-core system can achieve high performance, throughput per unit of chip area and energy efficiency compared to other software platforms.

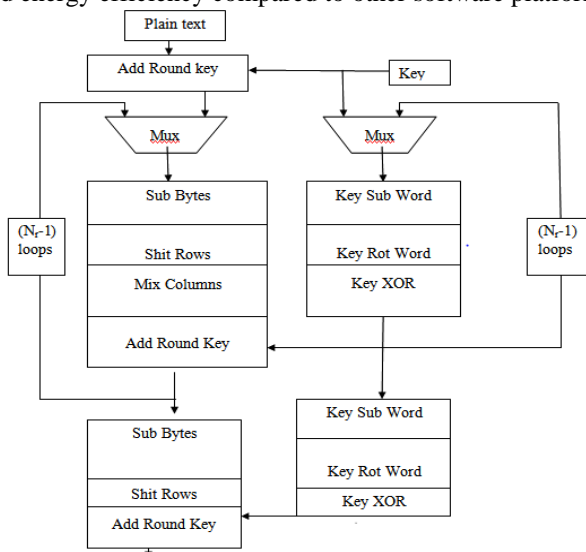


Fig. 1: Block diagram of AES encryption.

Both the online and offline key expansion process for each implementation model are discussed. The remainder of this paper is organized as follows: Section 2 introduces the AES algorithm. Section 3 briefly describes the features of the targeted fine-grained many-core system. In Section 4, various implementations are analysed by synchronous dataflow (SDF) models, mapped and measured on the targeted platform. Section 5 presents the optimization methodology. Finally, Section 6 concludes the paper.

## II. ADVANCED ENCRYPTION STANDARD

AES is a symmetric encryption algorithm, and it takes a 128-bit data block as input and performs several rounds of transformations to generate output cipher text. Each 128-bit data block is processed in a 4-by-4 array of bytes, called the state. The round key size can be 128, 192 or 256 bits. The number of rounds repeated in the AES,  $N_r$ , is defined by the length of the round key, which is 10, 12 or 14 for key lengths of 128, 192 or 256 bits, respectively. Fig. 1 shows the AES encryption steps with the key expansion process. For encryption, there are four basic transformations applied as follows:

1. SubBytes: The SubBytes operation is a nonlinear byte substitution. Each byte from the input state is replaced by another byte according to the substitution box (called the S-box). The S-box is computed

Based on a multiplicative inverse in the finite field  $GF(2^8)$  and a bitwise affine transformation.

2. Shift Rows: In the Shift Rows transformation, the first row of the state array remains unchanged. The bytes in the second, third, and fourth rows are cyclically shifted by one, two, and three bytes to the left, respectively.

3. Mix Columns: During the Mix Columns process, each column of the state array is considered as a polynomial over  $GF(2^8)$ . After multiplying modulo  $x^4 + 1$  with a fixed polynomial  $a(x)$ , given by  $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$ , (1) The result is the corresponding column of the output state.

4. AddRoundKey: A round key is added to the state array using a bitwise exclusive-or (XOR) operation. Round keys are calculated in the key expansion process. If Round keys are calculated on the fly for each data block, it is called AES with online key expansion. On the other hand, for most applications, the encryption keys do not change as frequently as data. As a result, round keys can be calculated before the encryption process, and kept constant for a period of time in local memory or registers. This is called AES with offline key expansion. In this paper, both the online and offline key expansion AES algorithms are examined.

Similarly, there are three steps in each key expansion round.

1. KeySubWord: The KeySubWord operation takes a fourbyte input word and produce an output word by substituting each byte in the input to another byte according to the S-box.
2. KeyRotWord: The function KeyRotWord takes a word  $[a_3, a_2, a_1, a_0]$ , performs a cyclic permutation, and returns the word  $[a_2, a_1, a_0, a_3]$  as output.
3. KeyXOR: Every word  $w[i]$ , is equal to the XOR of the previous word,  $w[i-1]$  and the word  $N_k$  positions earlier,  $w[i-N_k]$ .  $N_k$  equals 4, 6 or 8 for the key lengths of 128, 192 or 256 bits, respectively.

The decryption algorithm applies the inverse transformations in the same manner as the encipherment. As a result, we only consider the encryption algorithm in this work for simplicity, since the decipherment yields very similar results.

### III. TARGETED MANY-CORE ARCHITECTURE

#### A. Fine-Grained Many-Core Architecture

According to Pollack's Rule, the performance increase of architecture is roughly proportional to the square root of its increase in complexity. The rule implies that if we double the logic area in a processor, the performance of the core speeds up around 40 percent. On the other hand, a many core architecture has the potential to provide near linear performance improvement with complexity. For instance, instead of building a complicated core twice as large as before, a processor containing two cores (each is identical to the other) could achieve a possible 2\*performance improvement if the application can be fully parallelized. Therefore, if the target application has enough inherent parallelism, an architecture with thousands of small cores would offer a better performance than one with a few large cores within the same die area.

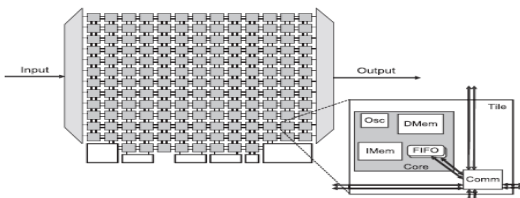


Fig.2: Block diagram of the 167-processor computational platform.

#### B. Asynchronous Array of Simple Processors (ASAP)

The targeted Asynchronous Array of Simple Processors architecture is an example of a fine-grained many-core computation platform, supporting globally-asynchronous locally synchronous (GALS) on-chip network and dynamic voltage and frequency scaling (DVFS) [24].

Fig.2 shows the block diagram of ASAP. The computational platform is composed of 164 small identical processors, three hardware accelerators and three 16 KB shared memories. All processors and shared memories are clocked by local fully independent oscillators and are connected by a reconfigurable 2D-mesh network that supports both nearby and long-distance communication.

#### C. Programming Methodology On ASAP

Programming the ASAP array follows three basic steps:

1. Each task of the application is mapped to one or few processors on the array. Each processor is programmed using either C or assembly language.
2. The inputs and outputs of different tasks are interconnected using a configuration file or a GUI mapping tool.
3. After compiled by our C compiler and assembler, the programs of tasks are mapped to the 2D mesh ASAP array.

### IV. AES IMPLEMENTATIONS ON ASAP

In this section, we present 16 different complete and fully-functional AES ciphers. The throughput of each design is measured from simulations on a cycle-accurate Verilog RTL model of the actual silicon chip.

Table 1 shows the execution delays of different processors. For example, MixColumns-16 executes the Mix Columns process on a whole 16-byte data block, while MixColumns-4 performs on a single 4-byte column. The execution time of MixColumns-4 is more than one fourth of the delay of MixColumns-16 due to programming overhead on ASAP. Similarly, SubBytes-16 requires 132 clock cycles to process a 16-byte data block, and it takes 10 clock cycles for SubBytes-1 to substitute1 byte. In our proposed implementations, the key expansion process is divided into two processing units, Key Sub Word and Key Schedule. Each Key Schedule processor contains two Steps of the key expansion process, KeyRot-Word and KeyXOR.

Processors	Execution delays on ASAP	IMEM usage
SubBytes-1	10 cycles/byte	9%
SubBytes-4	40 cycles/four bytes	9%
SubBytes-16	132 cycles/block	10%
Shift Rows	38 cycles/block	18%
Mix-Columns-16	266 cycles/block	63%
Mix-Columns-4	70 cycles/column	31%
Add Round Key	22 cycles/block	18%
Key Sub Word	56 cycles/block	13%
Key Schedule	60 cycles/block	22%

Table. 1:Execution Delays of Processors on ASAP2

Each data block is a 4-by-4 byte array. In the following sections, we present the eight AES implementations with online key expansion in detail, since the offline implementations can be derived by removing the cores used for key expansion from the online designs. For simplicity, we focus on the situation with a 128-bit key and  $N_r=10$  in this paper and the impact of different key lengths to our designs are discussed in detail inSection 4.9.

#### A. One-Task One-Processor (Otop)

The most straight forward implementation of an AES cipher is to apply each step in the algorithm as a task in the dataflow diagram as shown in Fig. 3a. Then, each task in the dataflow diagram can be mapped on one processor on the targeted many-core platform. We call this implementation one-task one-processor. For simplicity, all of the execution delay (shown in Table 1), input rates, and output rates in the following dataflow diagrams are omitted. Since the key expansion is processing in parallel with the main algorithm, the throughput of the OTOP implementation is determined by the nine ( $N_r - 1=9$ ) loops in the algorithm. The OTOP implementation requires 10 cores on ASAP as shown in Fig.

3b. The throughput of the OTOP implementation is 3,582 clock cycles per data block, equalling 223.875 clock cycles per byte.

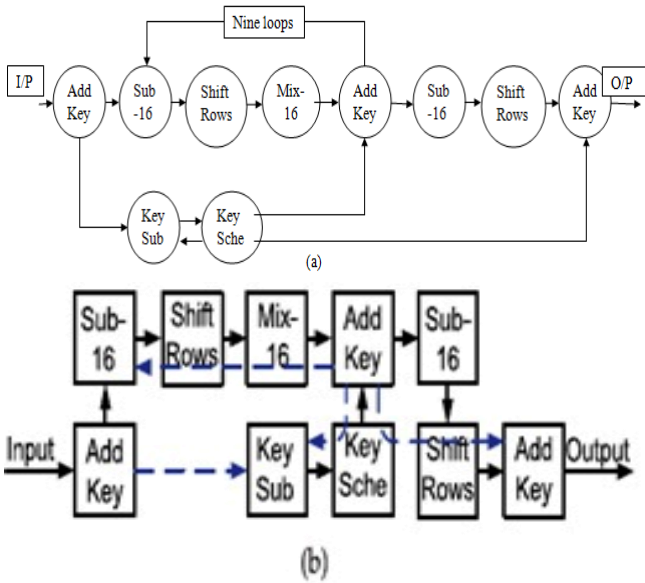


Fig. 3: One-task One-processor (a) Data flow diagram and (b) 10 cores ASAP mapping.

### B. Loop-Unrolled Nine Times

To enhance the AES cipher's throughput, we apply loop unrolling to the OTOP model and obtain the Loop-unrolled Nine Times dataflow diagram as shown in Fig. 4a. The loop unrolling breaks the dependency among different loops and allows the nine loops in the AES algorithm to operate on multiple data blocks simultaneously. To improve the throughput as much as possible, we unroll the loops in both the AES algorithm and the key expansion process by  $Nr - 1$  and  $Nr$  times, which equals 9 and 10, respectively. After loop unrolling, the throughput of the AES implementation is increased to 266 cycles per data block, equalling 16.625 cycles per byte. The mapping of the Loop-unrolled Nine Times model is shown in Fig. 4b, which requires 60 cores.

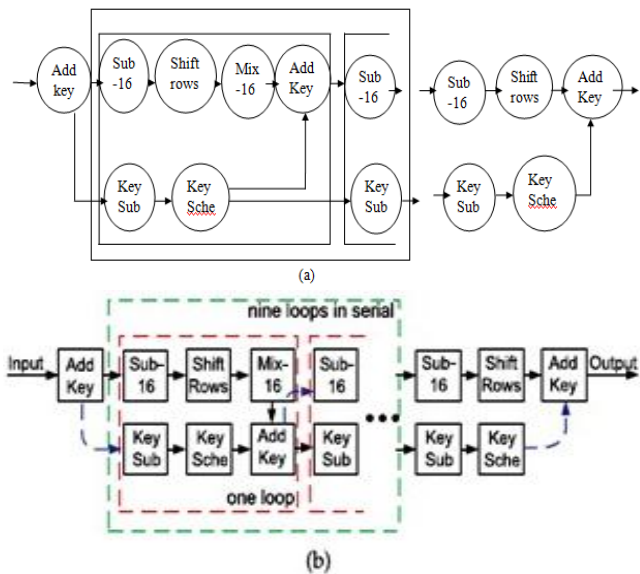


Fig. 4: Loop-unrolled Nine Times (a) Dataflow diagram and (b) 60 cores ASAP mapping.

### C. Loop-Unrolled Three Times

To achieve a moderate throughput with fewer cores, we could unroll the main loops in the AES algorithm by  $S$  times ( $S$  is divisible by  $Nr-1$ ), instead of  $Nr-1$  times. For this example, the nine loops in the AES algorithm could be split into three blocks, and each block loops three times. The dataflow diagram and mapping of the Loop unrolled Three Times implementation are shown in Figs. 5a and 5b, respectively. Compared to the OTOP model, the throughput is improved to 1,098 cycles per data block, which equals 68.625 cycles per byte; while the mapping requires 24 cores, 36 fewer than the Loop unrolled Nine Times implementation.

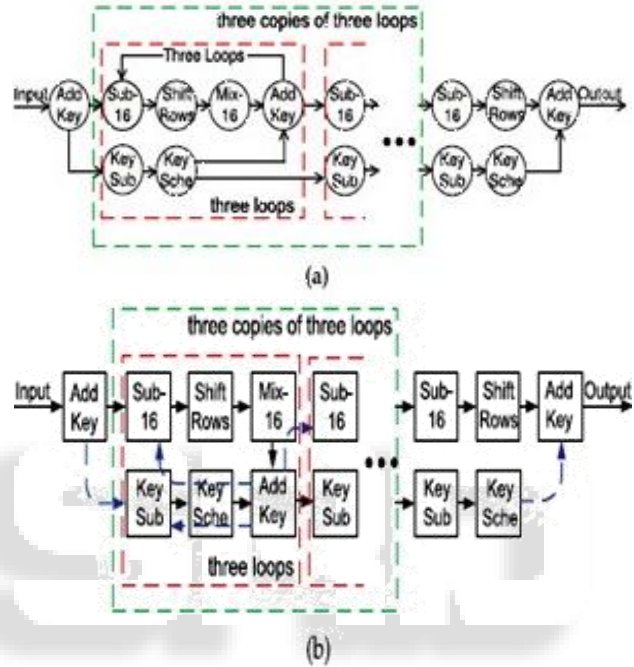


Fig.5: Loop-unrolled Three Times (a) Dataflow diagram and (b) 24 cores ASAP mapping.

### D. Parallel-Mix Columns

Besides loop unrolling, another way to increase the throughput of the OTOP model is to reduce the main loop's latency in the AES algorithm. In a single loop, the execution delay of MixColumns-16 results in 60 percent of the total latency. Each MixColumns-16 operates on a four-column data block, and the operation on each column is independent. Therefore, each MixColumns-16 processor can be replaced by four MixColumns-4s. Each MixColumns-4 actor computes only one column rather than a whole data block. As a result, the throughput of the Parallel Mix Columns implementation is increased to 2,180 cycles per block, equalling 136.25 cycles per byte. The dataflow diagram and mapping of the Parallel- Mix Columns model are shown in Figs. 6a and 6b.

Each core on our targeted computational platform can only support two statically configured input ports. Three cores, each called Merge Core, are used to merge the four data streams from *MixColumns-4s* into one stream for *AddRoundKey*.

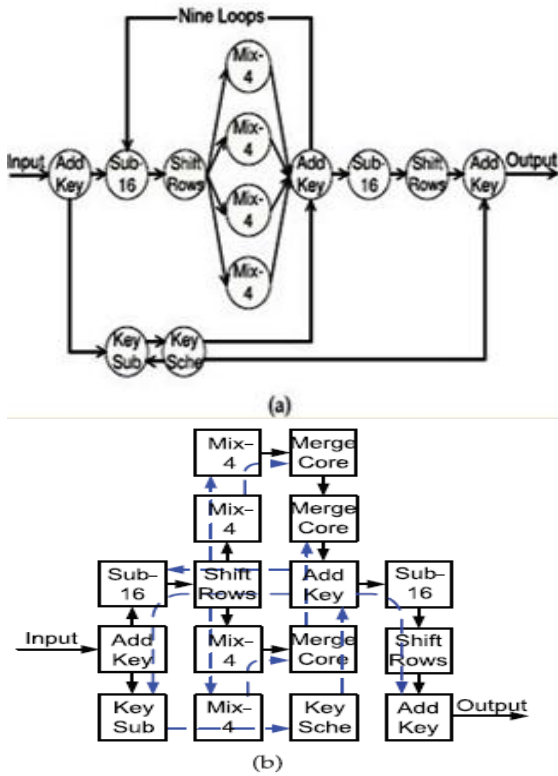


Fig. 6: Parallel-Mix Columns (a) Dataflow diagram and (b) 16 cores AsAP mapping.

The dependence among bytes in one column diminishes the performance improvement for further parallelization. For instance, if we parallelize one MixColumns-4 into two MixColumns-2s, the effective execution delay of the MixColumns process is reduced to 64 cycles from 70 cycles. This saves only 6 cycles while it requires eight more processors (four extra Mix-Columns cores and four extra Merge Cores). Therefore, further parallelization on the Mix-Columns process would impair the area and energy efficiency of the entire system without significant performance improvement.

E. Parallel-Sub Bytes-Mix-Columns

In the Parallel- Mix Columns implementation, SubBytes-16 requires 132 cycles to encrypt one data block, which contributes the largest execution delay in one loop. In order to increase the throughput further, we parallelize one SubBytes-16 into four SubBytes-4s, which is shown in Fig. 7a. In this implementation, each SubBytes-4 processes 4 bytes rather than 16 bytes in one data block. The effective execution delay of the Sub Bytes process is decreased to 40 cycles per block, only around one fourth as before. Therefore, the throughput of the Parallel-Sub Bytes- Mix Columns model is increased to 1,350 cycles per block, equalling 84.375 cycles per byte. The mapping graph of the Parallel- Sub Bytes-Mix Columns implementation on AsAP shown in Fig. 7b requires 22 cores.

Instead of parallelizing SubBytes-16 into four SubByte-4s, we can replace it with 16 SubBytes-1s. The effective execution delay of the SubBytes process is reduced to 10 cycles. As a result, the latency of one-loop decreases to 120 cycles. Therefore, the throughput of the cipher is increased to 67.5 cycles per byte. However, it requires seven additional cores dedicated to communication (four

MergeCores and three DispatchCores), which impair the area and energy efficiency of the implementation.

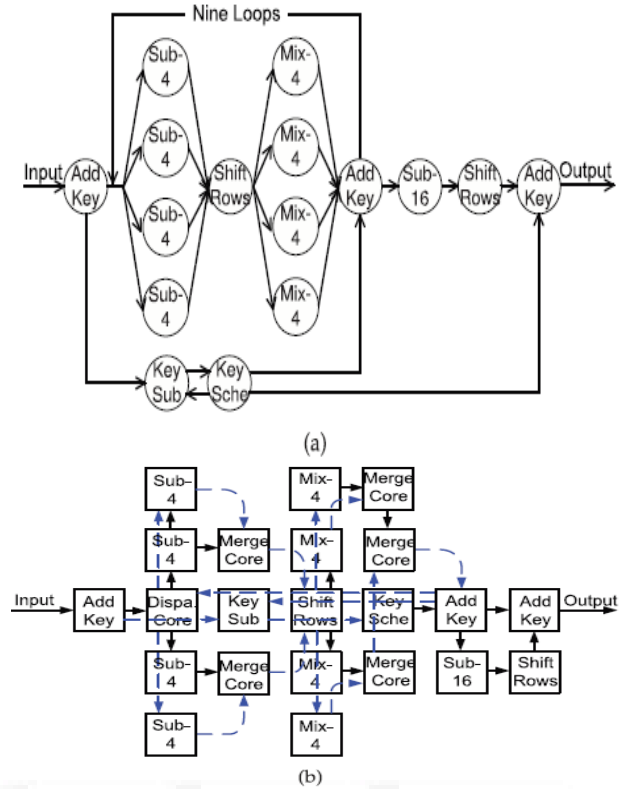
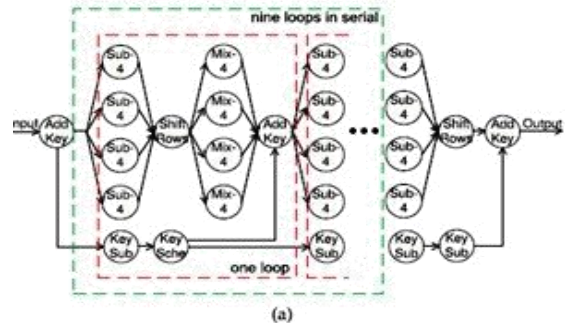


Fig.7: Parallel-Sub Bytes-Mix Columns (a) Dataflow diagram and (b) 22 cores AsAP mapping.

F. Full-Parallelism

The Full-parallelism AES implementation combines the Parallel-Sub Bytes-Mix Columns model and loop unrolling. The dataflow diagram and the mapping of the Full parallelism model are shown in Figs. 8a and 8b. As expected, the throughput of this design is the highest among all of the models introduced in this paper since it employs most data and task parallelism. The throughput of the Full-parallelism model is 70 cycles per block, equalling 4.375 cycles per byte. It also requires 164 cores, which is the largest implementation of all.

In the Full-parallelism model, the MixColumns-4 processors are the throughput bottlenecks which determine the performance of the cipher. Therefore, parallelizing the SubBytes process with more than four processors would only increase the area and power overhead without any performance improvement.



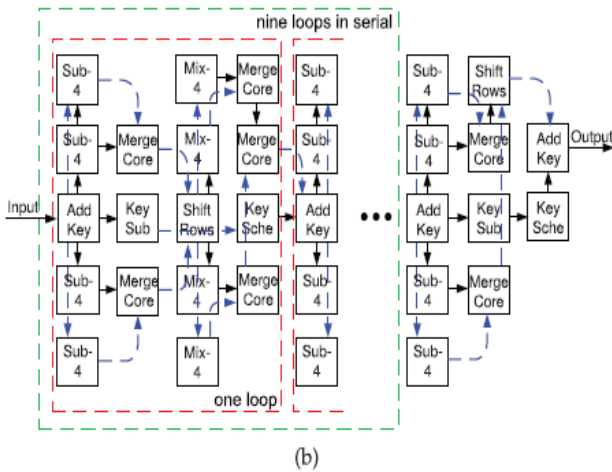


Fig. 8: Full-parallelism (a) Dataflow diagram and (b) 164 cores ASAP mapping.

G. Small

The Small model implements an AES cipher on AsAP with the fewest processors. As shown in Fig. 9, it requires at least eight cores to implement an AES cipher with online key expansion process, since each core on AsAP has only a 128\*32-bit instruction memory and a 128\*16 bit data memory. The throughput of the Small model is 2,678 cycles per data block, which equals 167.375 cycles per byte.

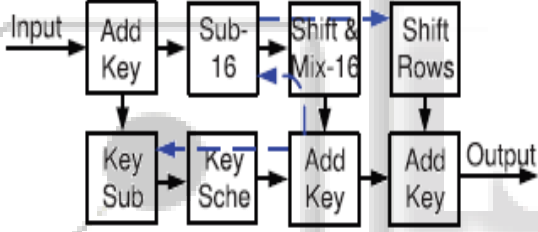


Fig.9: Eight cores ASAP mapping of the Small implementation.

H. No-Merge-Parallelism

In contrast to the Small model, the No-merge-parallelism model exploits as much parallelism as possible without introducing any cores dedicated to communication, including MergeCores and DispatchCores. The mapping graph of the No-merge-parallelism implementation on AsAP is shown in Fig. 10. To speed up the implementation, loop unrolling is applied in this model. Each MixColumns-16 is divided into two MixColumns-8s, which helps reduce the effective delay of the Mix Columns process. In order to eliminate additional communication processors and simplify the routing, we combine the SubBytes and the ShiftRows stages in one core. This implementation requires 59 cores, and has a throughput of 152 cycles per block, equalling 9.5 cycles per byte.

I. Designs with Longer Keys

As introduced in Section 2, besides the 128-bit key, the AES algorithm also supports key lengths of 192 and 256 bits.

Encrypting with longer keys results in two major areas of additional computation. First, the number of loops in the AES algorithm is increased. Second, the key expansion cores require more clock cycles to process round keys.

For the designs without loop-unrolling (Small, OTOP, Parallel-MixColumns, and Parallel-SubBytes-MixColumns), no extra cores are required. These mappings operate with longer keys by increasing the number of round loops, Nr, and reprogramming the key expansion related cores. The throughputs of these designs are decreased due to the increased number of Nr rounds.

For the designs with loop-unrolling, additional cores are added depending on the number of rounds required. For example, 12 and 24 more cores are required for the No-merge-parallelism designs with a 192-bit and 256-bit key, respectively. The throughputs of the Loop-unrolled and the No-merge-parallelism are kept the same as before, which is determined by the MixColumns operation. On the other hand, for the Full-parallelism implementation, the throughput is decreased since the bottlenecks of the system are shifted from the MixColumn-4 processors to the key expansion cores, due to the overhead of processing longer keys.

Due to the significant effort required, 192-bit and 256-bit designs are not implemented in this work.

V. OPTIMIZED FULL PARALLELISMS

In this proposed System presents various software implementations of the AES algorithm with different data and task parallelism granularity, and shows that AES implementations on a fine grained many-core system can achieve high performance, throughput per unit of chip area and energy efficiency compared to other software platforms.

In contrast to the Small model, the No-merge-parallelism model exploits as much parallelism as possible without introducing any cores dedicated to communication, including Merge Cores and Dispatch Cores. The mapping graph of the No-merge-parallelism implementation on AsAP is shown in Fig.10. To speed up the implementation, loop unrolling is applied in this model. Each MixColumns-16 is divided into two MixColumns-8s, which helps reduce the effective delay of the Mix Columns process. In order to eliminate additional communication processors and simplify the routing, we combine the Sub Bytes and the Shift Rows stages in one core.

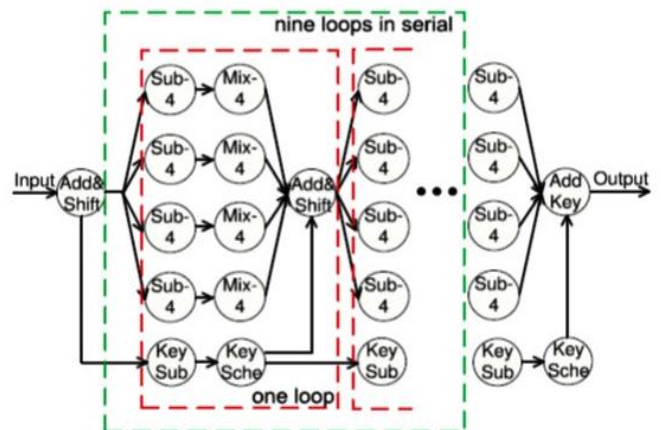


Fig . 10: Optimized Full-Parallelisms

VI. CONCLUSION

We have presented 16 different AES cipher implementations with both online and offline key expansion on a fine-grained many-core system. Each implementation exploits different

levels of data and task parallelism. The smallest design requires only six processors, equalling 1:02 mm<sup>2</sup> in a 65 nm fine-grained many-core system. The fastest design achieves a throughput of 4.375 cycles per byte, which is 2.21 Gbps when the processors are running at a frequency of 1.2 GHz. We also optimize the area of each implementation by examining the workload of each processor, which reduces the number of cores used as much as 18 percent. The design on the fine-grained many core system achieves energy efficiencies approximately 2.9-18.1 times higher than other software platforms, and performance per area on the order of 3.3-15.6 times higher. Overall, the fine-grained many-core system has been demonstrated to be a very promising platform for software AES implementations.

#### REFERENCES

- [1] Bin Liu, Student Member, IEEE, and Bevan M. Baas, Senior Member, IEEE "Parallel AES Encryption Engines For Many-Core Processor Arrays"
- [2] NIST, "Advanced Encryption Standard (AES)," <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>, Nov. 2001.
- [3] NIST, "Data Encryption Standard (DES)," <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>, Oct. 1999.
- [4] I. Verbauwhe, P. Schaumont, and H. Kuo, "Design and Performance Testing of a 2.29 gb/s Rijndael Processor," IEEE J. Solid-State Circuits, vol. 38, no. 3, pp. 569-572, Mar. 2003.
- [5] D. Mukhopadhyay and D. RoyChowdhury, "An Efficient end to End Design of Rijndael Cryptosystem in 0:18\_m CMOS," Proc. 18th Int'l Conf. VLSI Design, pp. 405-410, Jan. 2005.