

Parallel AES Encryption Engines for Many-Core Processor Arrays

Bin Liu, *Student Member, IEEE*, and Bevan M. Baas, *Senior Member, IEEE*

Abstract—By exploring different granularities of data-level and task-level parallelism, we map 16 implementations of an Advanced Encryption Standard (AES) cipher with both online and offline key expansion on a fine-grained many-core system. The smallest design utilizes only six cores for offline key expansion and eight cores for online key expansion, while the largest requires 107 and 137 cores, respectively. In comparison with published AES cipher implementations on general purpose processors, our design has 3.5-15.6 times higher throughput per unit of chip area and 8.2-18.1 times higher energy efficiency. Moreover, the design shows 2.0 times higher throughput than the TI DSP C6201, and 3.3 times higher throughput per unit of chip area and 2.9 times higher energy efficiency than the GeForce 8800 GTX.

Index Terms—Advanced encryption standard (AES), ASAP, fine-grained, many-core, parallel processor, software, synchronous dataflow

1 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.

To satisfy different applications' requirements, numerous hardware implementations of AES have been reported. Verbaauwhede et al. described the first AES implementation on silicon, which can provide a 2.29 Gbps throughput with a nonpipeline architecture [3]. Mukhopadhyay and Roy-Chowdhury improved their AES system to 8 Gbps with pipelining [4], which is a common technique used to enhance the performance of a system [5]. The first AES implementation with a throughput over 10 Gbps was proposed by applying T-box [6], which is a combination of the *SubBytes*, *ShiftRows*, and *MixColumns* phases in the AES algorithm [7]. Furthermore, the area-throughput tradeoffs of fully pipelined AES processors with throughputs between 30 and 70 Gbps have been presented [8]. Recently, Mathew et al. implemented a 53 Gbps AES accelerator in 45 nm CMOS technology [9]. Besides application specific integrated circuit (ASIC) designs, configurable hardware is another choice for AES implementations. For example, there are several FPGA implementations that achieve a throughput approximately

20 to 30 Gbps [10], [11], [12] by applying loop unrolling and pipelining. Recently, Qu et al. demonstrated a 73.7 Gbps AES system on a Xilinx XC5VLX85 chip running at 570 MHz [13].

Although hardware implementations generally offer higher throughput and better energy efficiency than software designs, they are difficult to upgrade and adapt for future possible protocol changes. Moreover, ASIC designs are very time consuming and costly. For example, it takes generally 18 to 24 months for a full custom ASIC product and costs approximately 50 Million USD to design [14]. One advantage of the Rijndael algorithm is that it is not only fit for hardware implementations, but also suitable for efficient software designs. Matsui and Nakajima proposed a bitslice AES implementation on Intel Core 2, which achieves a 9.2 clock cycles per byte throughput for a data chunk longer than 2,048 bytes, equaling 1.85 Gbps when the core is running at its maximum frequency of 2.13 GHz [15]. The bitslice technique was first proposed by Biham for fast DES implementation on a software platform with a word size longer than 16 bits [16]. Bernstein and Schwabe investigated the opportunities of reducing instruction count and cycles by combining different instructions together for various architectures [17]. Both bitslice and specific sets of instructions from Supplemental Streaming SIMD Extensions 3 (SSSE3 [18]) are utilized to enhance the performance of Intel Core i7 920 as high as 6.92 clock cycles per byte [19]. Besides pure general software AES implementations, the Intel AES-NI utilizes specialized hardware to support six AES instructions, and achieves a throughput of 1.28 clock cycles per byte [20]. There is also a trend to use Graphic Processing Units (GPUs) and DSP processors to implement the AES algorithm. Wollinger et al. compared different encryption algorithms on a TMS320C6X processor and achieved a 14.25 clock cycles per byte [21]. Manavski presented an AES implementation with a peak throughput of 8.28 Gbps on a GeForce 8,800 GTX chip when the input data block is longer than 8 MB [22].

- The authors are with the Department of Electrical and Computer Engineering, University of California, Davis, CA 95616. E-mail: {binliu, bbaas}@ucdavis.edu.

Manuscript received 15 Aug. 2011; revised 18 Nov. 2011; accepted 6 Dec. 2011; published online 22 Dec. 2011.

Recommended for acceptance by J. Plusquellic.

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-08-0546. Digital Object Identifier no. 10.1109/TC.2011.251.

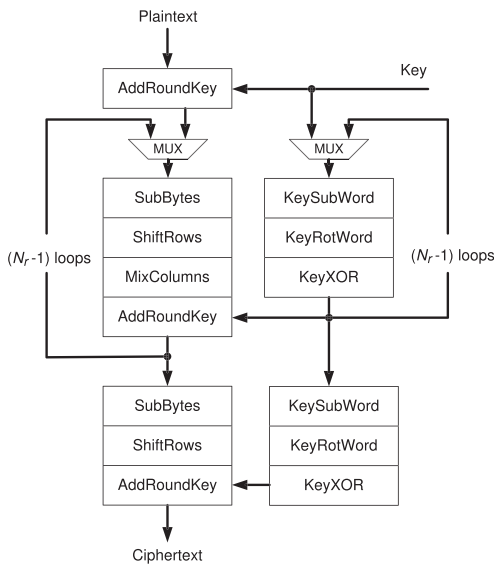


Fig. 1. Block diagram of AES encryption.

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. 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 analyzed by synchronous dataflow (SDF) models, mapped and measured on the targeted platform. Section 5 presents the area optimization methodology and compares the area efficiency among different implementations. Section 6 compares the energy efficiency. Section 7 compares our work with other software designs. Finally, Section 8 concludes the paper.

2 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 ciphertext. 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. *ShiftRows*: In the *ShiftRows* 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. *MixColumns*: During the *MixColumns* 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

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}, \quad (1)$$

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 four-byte 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 Nk positions earlier, $w[i - Nk]$. Nk 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.

3 TARGETED MANY-CORE ARCHITECTURE

3.1 Fine-Grained Many-Core Architecture

According to Pollack's Rule, the performance increase of an architecture is roughly proportional to the square root of its increase in complexity [23]. 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 $2\times$ 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 [23].

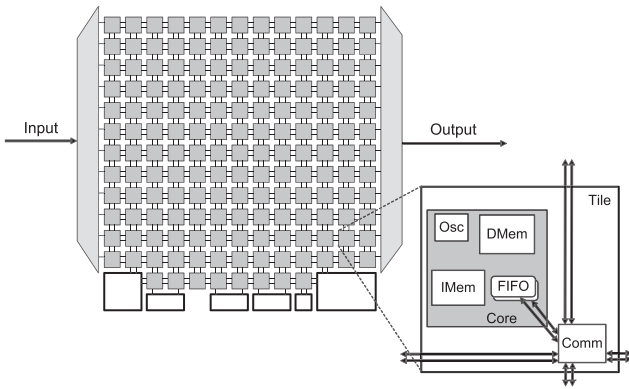


Fig. 2. Block diagram of the 167-processor computational platform [26].

3.2 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 [25]. Each tile on the platform can be statically configured to take input data from two links, while sending its output to other processors via dynamic configuration.

Each simple processor has a 6-stage pipeline, which issues one instruction per clock cycle. Moreover, no application-specific instructions are implemented. Each processor has a 128×32 -bit instruction memory and a 128×16 -bit data memory. Each processor occupies 0.17 mm^2 and has a maximum clock frequency of 1.2 GHz. The 167-processor chip was fabricated in 65 nm CMOS technology. [26].

3.3 Programming Methodology on AsAP

Programming the AsAP array follows three basic steps [27]:

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 [28].
3. After compiled by our C compiler and assembler, the programs of tasks are mapped to the 2D mesh AsAP array.

4 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 *MixColumns*

TABLE 1
Execution Delays of Processors on AsAP2

Processors	Execution Delays on AsAP2	IMEM Usage
SubBytes-1	10 cycles/byte	9%
SubBytes-4	40 cycles/four bytes	9%
SubBytes-16	132 cycles/block	10%
ShiftRows	38 cycles/block	18%
MixColumns-16	266 cycles/block	63%
MixColumns-4	70 cycles/column	31%
AddRoundKey	22 cycles/block	18%
KeySubWord	56 cycles/block	13%
KeySchedule	60 cycles/block	22%

Each data block is a 4-by-4 byte array.

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 substitute 1 byte. In our proposed implementations, the key expansion process is divided into two processing units, *KeySubWord* and *KeySchedule*. Each *KeySchedule* processor contains two steps of the key expansion process, *KeyRotWord* and *KeyXOR*.

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 is discussed in detail in Section 4.9.

4.1 One-Task One-Processor (OTOP)

The most straightforward 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

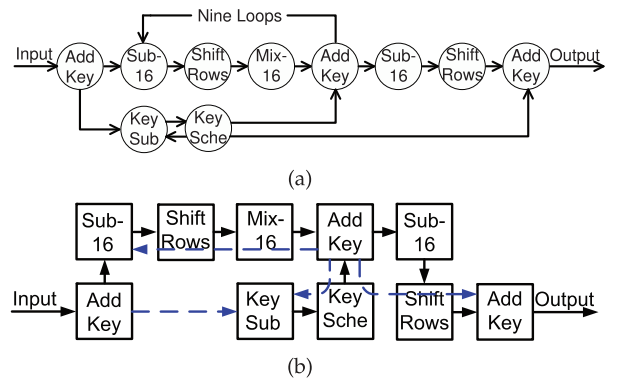


Fig. 3. One-task One-processor (a) dataflow diagram and (b) 10 cores AsAP mapping.

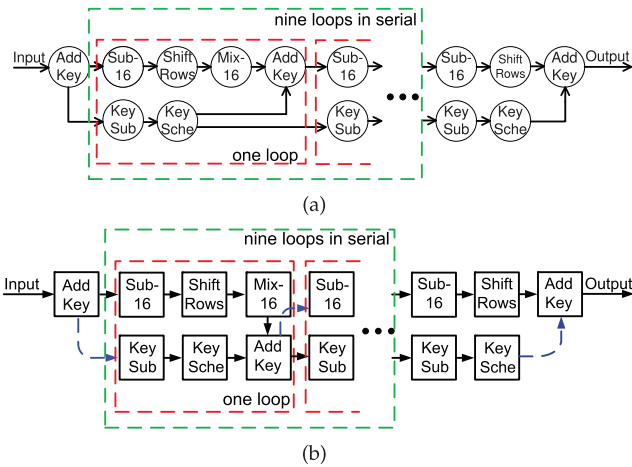


Fig. 4. Loop-unrolled Nine Times (a) dataflow diagram and (b) 60 cores AsAP mapping.

shown in Fig. 3b. The throughput of the OTOP implementation is 3,582 clock cycles per data block, equaling 223.875 clock cycles per byte.

4.2 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 $N_r - 1$ and N_r times, which equals 9 and 10, respectively. After loop unrolling, the throughput of the AES implementation is increased to 266 cycles per data block, equaling 16.625 cycles per byte. The mapping of the Loop-unrolled Nine Times model is shown in Fig. 4b, which requires 60 cores.

4.3 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 $N_r - 1$), instead of $N_r - 1$ 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.

4.4 Parallel-MixColumns

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

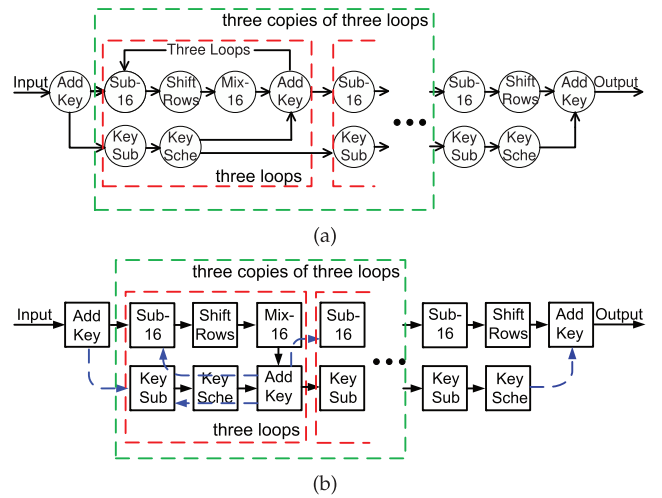


Fig. 5. Loop-unrolled Three Times (a) dataflow diagram and (b) 24 cores AsAP mapping.

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-MixColumns implementation is increased to 2,180 cycles per block, equaling 136.25 cycles per byte. The dataflow diagram and mapping of the Parallel-MixColumns 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 *MergeCore*, are used to merge the four data streams from *MixColumns-4s* into one stream for *AddRoundKey*.

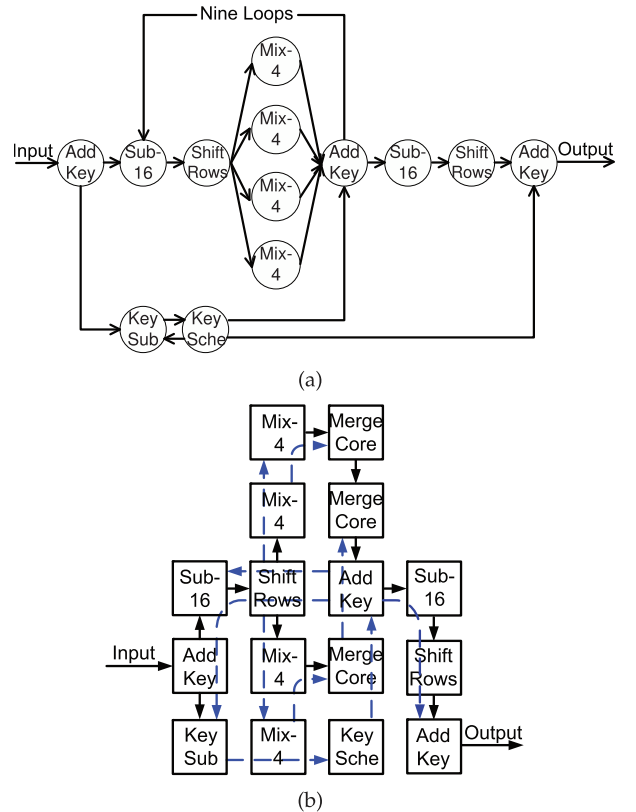


Fig. 6. Parallel-MixColumns (a) dataflow diagram and (b) 16 cores AsAP mapping.

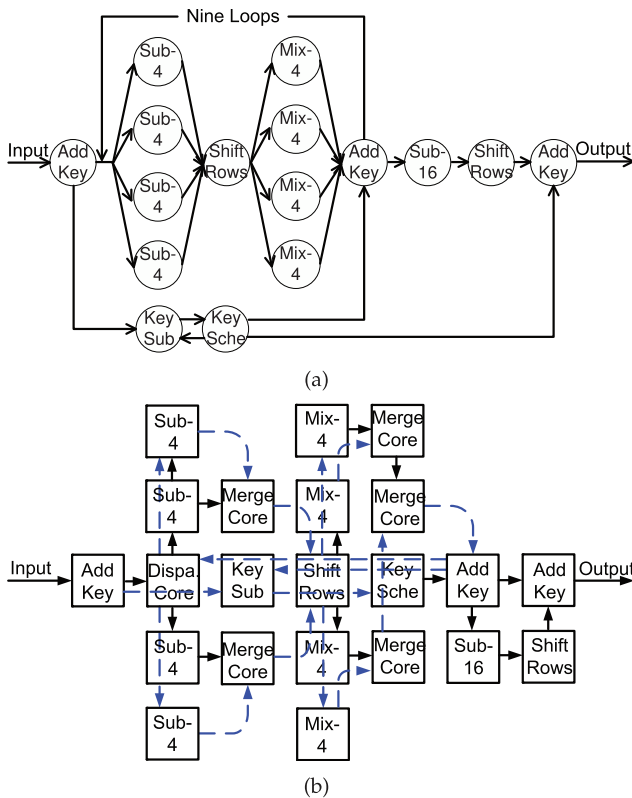


Fig. 7. Parallel-SubBytes-MixColumns (a) dataflow diagram and (b) 22 cores AsAP mapping.

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 *MixColumns* cores and four extra *MergeCores*). Therefore, further parallelization on the *MixColumns* process would impair the area and energy efficiency of the entire system without significant performance improvement.

4.5 Parallel-SubBytes-MixColumns

In the Parallel-MixColumns 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 *SubBytes* process is decreased to 40 cycles per block, only around one fourth as before. Therefore, the throughput of the Parallel-SubBytes-MixColumns model is increased to 1,350 cycles per block, equaling 84.375 cycles per byte. The mapping graph of the Parallel-SubBytes-MixColumns 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

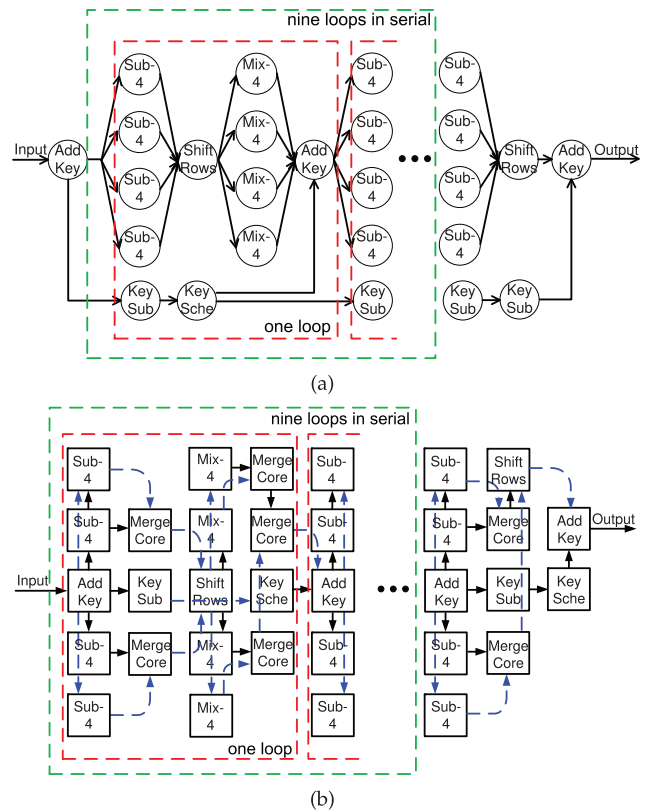


Fig. 8. Full-parallelism (a) dataflow diagram and (b) 164 cores AsAP mapping.

67.5 cycles per byte. However, it requires seven additional cores dedicated to communication (four *MergeCores* and three *DispaCores*), which impair the area and energy efficiency of the implementation.

4.6 Full-Parallelism

The Full-parallelism AES implementation combines the Parallel-SubBytes-MixColumns 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, equaling 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.

4.7 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.

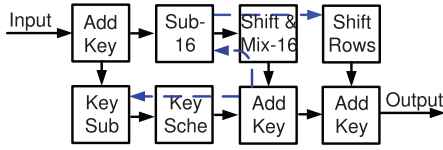


Fig. 9. Eight cores AsAP mapping of the Small implementation.

4.8 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 *MixColumns* 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, equaling 9.5 cycles per byte.

4.9 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, N_r , and reprogramming the key expansion related cores. The throughputs of these designs are decreased due to the increased number of N_r 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.

5 AREA EFFICIENCY ANALYSIS

Area is a significant metric in system design. Less area means less silicon, therefore less cost. From a many-core processor perspective, area is represented by the number of cores required to implement applications. Smaller area translates into fewer used cores and leaves more opportunities for dealing with other applications on the same platform simultaneously. To evaluate the area efficiency between various AES implementations, a metric called

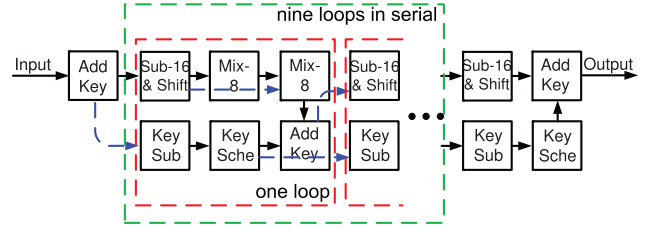


Fig. 10. Fifty Nine cores AsAP mapping of the No-merge-parallelism implementation.

ThroughputPerCore is defined as the ratio between the throughput of each design to the number of cores used to implement it,

$$\text{ThroughputPerCore} = \frac{\text{Throughput}}{\text{Number of Cores}}. \quad (2)$$

5.1 Area Optimization Methodology

Before comparing area efficiency among different AES implementations, area optimization is applied to all of the models without impairing performance. In this section, the area optimization methodology is illustrated through a detailed example of minimizing the number of cores used by the Full-parallelism model. As shown in Fig. 8b, there are 17 cores in one loop of the Full-parallelism mapping, including five communication-dedicated cores, which are used for routing only. And the final round operation requires 11 cores. Therefore, the number of cores utilized for the unoptimized Full-parallelism model is $(N_r - 1) \times N_{\text{one-loop}} + N_{\text{last-round}} = 9 \times 17 + 11 = 164$.

Two optimization steps are applied to the Full-parallelism model. First, since the *ShiftRows* process is only byte-rotation, alternating the sequence of the *SubBytes* and the *ShiftRows* stages would not affect encryption results. However, this alternation reduces two *MergeCores* for each loop. As a result, 18 cores are reduced from the Full-parallelism model. Second, the throughput of the Full-parallelism model is 70 cycles per block, which is determined by the operation delay of *MixColumns-4s*. Any actors with less execution delay would not impair the performance of the system. Therefore, a processor fusion of the *ShiftRows* in the N th loop and the *AddRoundKey* in the $(N - 1)$ th loop can reduce one more core for each loop, while keeping the same throughput since these new combination processors take only 60 cycles to process one data block. The dataflow diagram and mapping of the optimized Full-parallelism model are shown in Figs. 11a and 11b, respectively.

In summary, without losing any performance, the number of cores required by the online Full-parallelism model is decreased by approximately 16 percent to 137.

5.2 Area Efficiency Comparison

Based on the optimization methods discussed above, the number of cores utilized for each implementation is optimized as follows:

1. Small: Optimization methods are not applicable.
2. OTOP: The *SubBytes* and *ShiftRows* processors in the last round are fused into one processor, saving one processor.

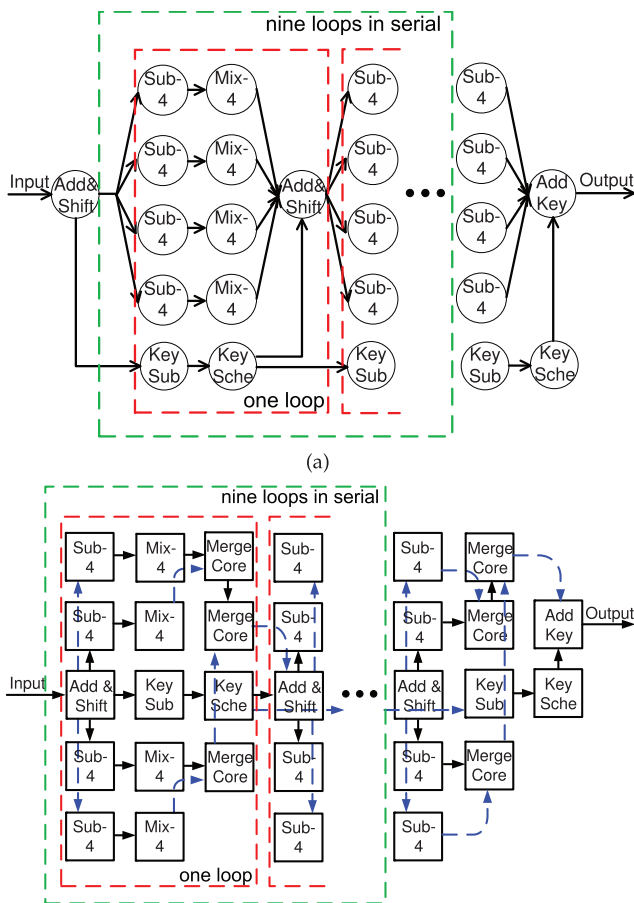


Fig. 11. Optimized Full-parallelism (a) dataflow diagram and (b) 137 cores ASAP mapping.

3. Parallel-MixColumns: The *SubBytes* and *ShiftRows* processors in the last round are fused into one processor, saving one processor.
4. Parallel-SubBytes-MixColumns: The sequence of the *SubBytes* and the *ShiftRows* stages is alternated, which saves three processors. The *SubBytes* and *ShiftRows* processors in the last round are fused into one processor, saving one more processor.
5. Loop-unrolled Three Times: The *SubBytes* and *ShiftRows* processors in the last round are fused into one processor, saving one processor.
6. Loop-unrolled Nine Times: The *SubBytes* and *ShiftRows* processors in the same round are fused into one processor, which saves 10 processors.

7. No-merge-parallelism: Optimization methods are not applicable.
8. Full-parallelism: The optimization has been discussed in detail in Section 5.1

5.2.1 Implementations with Online Key Expansion

The number of cores used for each optimized implementation is shown in Column 3 of Table 2. As expected, the Small implementation uses the fewest cores due to its simplicity. On the other hand, the Full-parallelism model occupies 137 cores, exploiting the greatest range of types of data parallelism. As a result, the Full-parallelism implementation requires $17\times$ as many cores as the Small model, while it also gains a $40\times$ throughput increase.

As defined in (2), *ThroughputPerCore* is used to compare the area efficiency between different models. The higher the throughput, the better the performance. The fewer the cores used, the smaller the area. As a result, a larger *ThroughputPerCore* ratio shows a higher area efficiency. In Table 2, Column 5 shows the *ThroughputPerCore* numbers of various implementations normalized to the Parallel-MixColumns model with online key expansion. The No-merge-parallelism implementation has the highest throughput per core rate, since it avoids any dedicated communication cores and exploits as much parallelism as possible simultaneously. The Full-parallelism and the Loop-unrolled models also offer high throughput per unit of chip area. Although the Small model has a relatively low throughput, it still offers a good area efficiency due to its extremely small area.

5.2.2 Implementations with Offline Key Expansion

Besides the online key expansion AES algorithm, the detailed results of AES with offline key expansion are also shown in Columns 6, 7, and 8 of Table 2. The processors used for key expansion process can be eliminated for the AES implementations with offline key expansion, which results in 29 percent improvement in average throughput per area compared to the implementations with online key expansion.

The throughput versus the number of cores of the eight offline implementations is shown in Fig. 12. The throughput is obtained when all processors are running at 1.2 GHz. Besides the basic implementations discussed above, we duplicate each implementation two and four times to scale the throughput and area. On the targeted platform, for any scaled implementation with a $4\times$ duplication, two mergecores are required to gather the outputs for the subsequent processor by assuming each processor could take only two inputs.

TABLE 2
Throughput and the Number of Cores Required by Different Implementations

Implementation	1/Throughput (cycles/byte)	Online Key Expansion			Offline Key Expansion		
		Total Cores	Comm. Cores	Normalized Throughput/Core	Total Cores	Comm. Cores	Normalized Throughput/Core
Small	167.375	8	0	1.53	6	0	2.04
One-task one-processor	223.875	9	0	1.01	7	0	1.30
Parallel-Mixcolumns	136.250	15	3	1	12	2	1.25
Parallel-SubBytes-Mixcolumns	84.375	18	3	1.35	15	2	1.61
Loop-unrolled Three Times	68.625	23	0	1.29	15	0	1.99
Loop-unrolled Nine Times	16.625	50	0	2.46	30	0	4.10
No-merge-parallelism	9.500	59	0	3.65	39	0	5.52
Full-parallelism	4.375	137	30	3.41	107	20	4.37

Communication cores are used for routing only, including MergeCores and DispatchCores. All of the throughput per core numbers are normalized to the Parallel-Mixcolumns model with online key expansion.

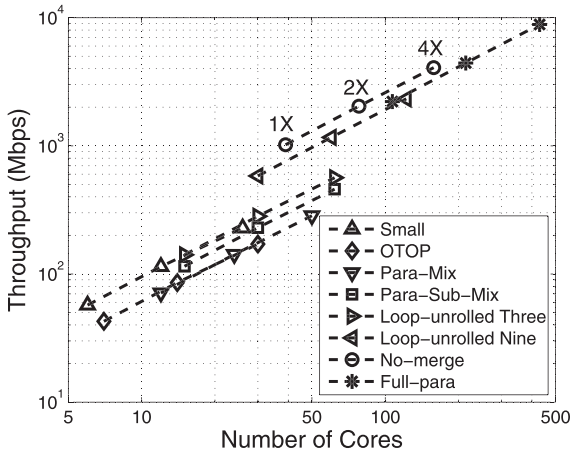


Fig. 12. Throughput versus the number of cores for the AES implementations with offline key expansion. All processors are running at 1.2 GHz. 1 \times , 2 \times and 4 \times represent the throughput when each implementation is duplicated once, twice, and four times, respectively.

6 ENERGY EFFICIENCY ANALYSIS

In this section, the power consumption and energy efficiency of the previously discussed eight implementations are investigated based on chip measurement results.

6.1 Power Numbers from Chip Measurements

Each core on AsAP can operate up to 1.2 GHz at 1.3 V [26]. The maximum frequency and power consumption of cores on AsAP have a near-linear and quadratic dependence on the supply voltage, as shown in Fig. 13. The average power dissipation of one core and communication link at 1.3 V and 1.2 GHz is shown in Table 3. This supply voltage and clock frequency are used in the power estimation and optimization case study in the next section. The table also shows during stalls (i.e., nonoperation while the clock is active), the processors still consume a significant portion, approximately 50 percent, of its active power. The leakage power is decreased to a negligible number when the clock is halted.

6.2 Power Estimation Methodology

On our targeted platform, each processor has four states: *active*, *NOP with active clock*, *stall with active clock*, and

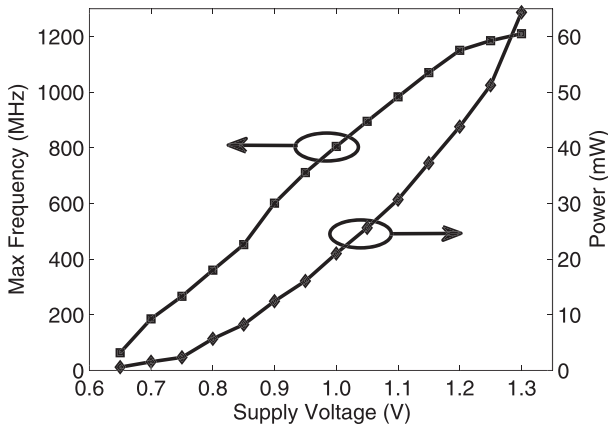


Fig. 13. Maximum operation frequency and 100 percent active power dissipation of one core versus supply voltage.

TABLE 3
Average Power Dissipation Measured at 1.3 V and 1.2 GHz [26]

Operation of	100% Active (mW)	Stall (mW)	Leakage (mW)
Processor	59.5	31.0	0.13
Nearest-neighbor comm.	5.9	NA	~ 0
Long-distance comm. one tile	12.1	NA	~ 0

standby with halted clock. The *active* mode means that the processor is busy with instruction execution, while the *NOP with active clock* represents the NOP operation in programs due to data and control hazards. Either an empty input or a full output FIFO is capable of halting each processor's clock and causing the processor to sleep in the *standby with halted clock* mode. Finally, the *stall with active clock* is the transition state between *active* and *stall with halted clock*. As a result, the overall power dissipated by all of the processors utilized in any implementation can be derived by

$$P_{Total} = \sum P_{Exe.i} + \sum P_{Stall.i} + \sum P_{Leak.i} + \sum P_{Comm.i}, \quad (3)$$

where $P_{Exe.i}$, $P_{Stall.i}$, $P_{Leak.i}$, and $P_{Comm.i}$ are the power consumed by computation execution, stalling (including NOP) with active clock, standby with halted clock (leakage only), and communication activities of the i th processor, respectively, and are estimated as follows:

$$\begin{aligned} P_{Exe.i} &= \alpha_i \cdot P_{ExeAvg} \\ P_{Stall.i} &= \beta_i \cdot P_{StallAvg} \\ P_{Leak.i} &= (1 - \alpha_i - \beta_i) \cdot P_{LeakAvg}, \end{aligned} \quad (4)$$

where P_{ExeAvg} , $P_{StallAvg}$, and $P_{LeakAvg}$ are the average power of processors while performing 100 percent execution, stalling (including NOP with active clock and stall with active clock) or halting (leakage only); α_i , β_i , and $(1 - \alpha_i - \beta_i)$ are the percentages of execution, stall, and standby activities of processor i , respectively. The communication power can be calculated as follows:

$$P_{Comm.i} = \gamma_i \cdot P_{CommNear} + \delta_i \cdot P_{CommLong}, \quad (5)$$

where γ_i and δ_i are the percentages of communication between neighboring and long-distance processors, respectively. $P_{CommNear}$ is the 100 percent active power consumed by a link when it is used for communication between neighboring processors, while $P_{CommLong}$ is for long-distance [29].

The optimized Full-parallelism model with offline key expansion is used as an example to illustrate the power estimation methodology discussed above. For the i th processor, its α_i , β_i and $(1 - \alpha_i - \beta_i)$ are derived from Columns 3, 4, 5, and 6 of Table 4. Furthermore, γ_i and δ_i are obtained from Columns 7 and 8. Note that the throughput of the Full-parallelism implementation is 70 cycles per block.

Additionally, if the implementation works under 1.3 V and 1.2 GHz, the power consumed by execution, stalling, standby, and communication activities of each processor are listed in Columns 9, 10, 11, and 12. In Column 2, the number of processors with the same operation are listed. Therefore,

TABLE 4
Operation Cycles and Power Consumption of Offline Key Expansion Full-Parallelism Implementation at 1.3 V and 1.2 GHz

Processor	Number	Execution NOP with Stall with Stall with			Nearby Comm. (cycles)	Long-dist. Comm. (cycles)	Execution Power (mW)	Stall Power (mW)	Leakage Power (mW)	Comm. Power (mW)	i^{th} Core Power (mW)	
		Time (cycles)	AC ^a (cycles)	AC ^a (cycles)								HC ^b (cycles)
AddKeyShiftRows	10	37	0	18	15	8	8	31.47	7.97	0.03	2.05	40.13
SubByte-1	40	27	12	16	15	4	0	22.97	12.40	0.03	0.34	36.59
MixColumn-4	36	63	7	0	0	2	2	53.59	3.10	0	0.51	57.20
FinalRoundAddKey	1	18	0	40	12	16	0	15.31	17.71	0.02	1.35	21.60
MergeCore	20	10	0	44	16	0	8	8.34	19.49	0.03	1.38	18.63
Total								3.35×10^3	1.09×10^3	2.12	81.4	4.52×10^3

^aAC stands for active clock. ^bHC stands for halted clock.

the total power number can be derived by the following equation and is listed in the last row:

$$P_{Total} = \sum N_i \cdot P_{Total_i}, \quad (6)$$

where N_i is the number of processors with the i th kind of operation, and P_{Total_i} is the total power dissipated by the i th processor. The total power consumption is 4.52 W with a 2.21 Gbps throughput. The communication power consumed by FIFOs and switches is 81.4 mW, which is 1.8 percent of the total power, while the leakage power is only 2.12 mW and 0.05 percent of the total power dissipation.

6.3 Energy Efficiency Comparison

The energy efficiency of a system describes how much energy is consumed for processing a specific workload. This metric influences a critical design parameter, battery lifetime, made more important by the increasing popularity of mobile devices. In our discussion, the energy efficiency is defined as the energy dissipated for processing one bit by

$$\begin{aligned} \text{EnergyPerBit}_{Vdd} &= \text{Power}_{Vdd} / \text{Throughput}_{Vdd} \\ &= (\text{Power}_{Vdd} \times \text{Delay}) / \text{freq}_{Vdd} \times 128, \end{aligned} \quad (7)$$

where Power_{Vdd} and freq_{Vdd} are the power dissipation and frequency for one model at supply voltage Vdd . Delay represents the number of clock cycles required for processing one data block. Since power has a general relationship with

supply voltage and operation frequency as $\text{Power} \propto Vdd^2 \cdot f$, from (7), it is expected that $\text{EnergyPerBit}_{Vdd} \propto Vdd^2$.

As shown in Fig. 14, for the eight offline implementations discussed above, the energy dissipated for processing one bit is nearly quadratically dependent on supply voltage, which is consistent with the theoretical analysis. Furthermore, the no-merge model consumes the least energy to encrypt one bit compared with other implementations, which is from 0.39 to 1.54 nJ/bit depending on the supply voltage and throughput. On the other hand, the Parallel-MixColumns implementation shows the lowest energy efficiency, which consumes approximately $2 \times$ the energy to encrypt a data block as the No-merge-parallelism model.

Fig. 15 shows that the AES implementations with online key expansion consume 35 to 55 percent more energy to process same workload, compared to their counterparts with offline key expansion.

7 RELATED WORK AND COMPARISON

Since the AES ciphers presented in this work are implemented on a programmable platform without any application-specific hardware, we compare our work with other software implementations on programmable processors, and do not compare with implementations that contain or are composed of specialized hardware (e.g., ASICs, ASIPs, FPGAs, etc.). AES hardware implementations have been reported to achieve throughputs per area up to tens and even hundreds of Gbps/mm² [9] and energy efficiencies in the range of several pJ/bit—they are in an entirely different class both in efficiencies achieved and in the cost and effort required to design.

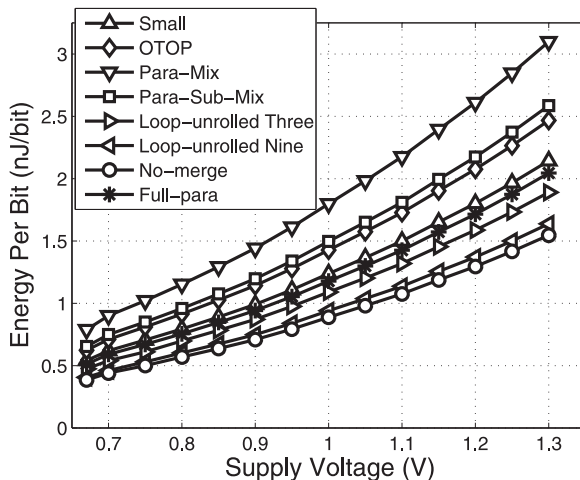


Fig. 14. Energy consumed for processing one bit of data versus supply voltage. All of the implementations shown in this figure are associated with offline key expansion.

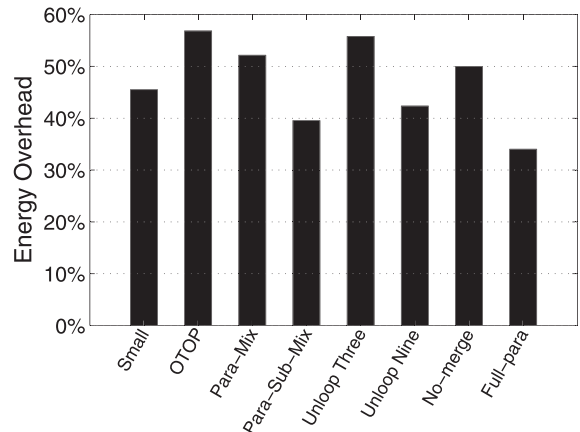


Fig. 15. Energy overhead of the AES implementations with online key expansion compared with the ones with offline key expansion.

TABLE 5
Comparison of AES Cipher Implementations on Different Software Platforms

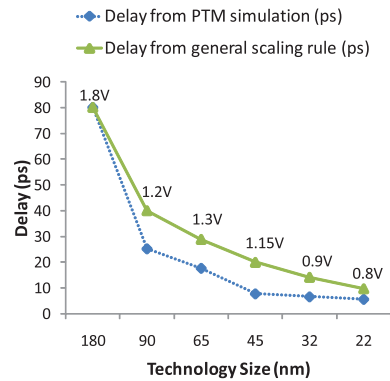
Platform	Method	Tech. (nm)	Area (mm ²)	Vdd (V)	Max Freq. (MHz)	Throughput (cycles/byte)	Power ^a (W)	Scaled Throughput (Mbps)	Scaled Area (mm ²)	Scaled Throughput/Area (Mbps/mm ²)	Scaled Energy/bit (nJ/bit)
Pentium 4 561 [15]		90	112	1.2	3600	16	57.5	2570	58.42	43.99	17.50
Athlon 64 3500 [15]		90	193	1.2	2200	10.6	44.5	2370	101	23.55	14.69
Core2 Duo E6400 [15]	Bitslice	65	111	1.3	2130	9.19	32.5	1854	111	16.70	17.53
Core2 Quad ^b Q6600 (one core)[19]	Bitslice + SSSE3	65	286/2 = 143	1.3	2400	9.32	26.25	2060	143	14.41	12.74
Core2 Quad ^b Q9550 (one core)[19]	Bitslice + SSSE3	45	214/4 = 53.5	1.15	2830	7.59	11.88	1307	112	11.71	21.16
Core i7 ^b 920 (one core)[19]	Bitslice + SSSE3	45	263/4 = 65.75	1.15	2668	6.92	16.25	1351	137	9.84	28.00
TI C6201 [21]		180	NA	1.8	200	14.25	NA	509	NA	NA	NA
GeForce 8800 GTX [22]	T-box	90	484	1.2	575	NA	67.5	11800	252	46.82	4.48
This Work^c AsAP [26]	No-merge offline key expan.	65	6.63	1.3	1210	9.5	1.58	1019	6.63	153.70	1.55

The original data are presented with different CMOS technologies and supply voltages. For comparison, area, performance, and power consumption are scaled to 65 nm technology with a supply voltage of 1.3 V. Area are scaled by $1/(s^2)$. Throughput and power numbers are scaled based on the PTM simulation results shown in Fig. 16. ^aThe typical power is not available, so 50 percent of thermal design power (TDP) is used based on the benchmark data of a general-purpose processor [30]. ^bThe throughput results from [19] are for only one core, so the area and power numbers are scaled proportionally. ^cAll referenced designs do not consider key expansion; therefore, the AES implementations on AsAP associated with offline key expansion are applied for a fair comparison.

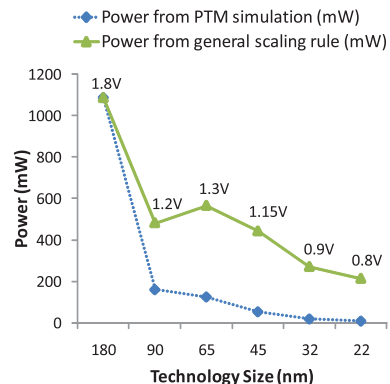
A comprehensive comparison of the state-of-the-art software AES implementations is summarized in Table 5. In order to make a fair comparison, all of the referenced data are scaled to 65 nm CMOS technology with a supply voltage of 1.3 V. The area data are scaled to 65 nm with a $1/(s^2)$ reduction, where s equals the ratio between the minimum feature size of the old technology and 65 nm. The delay and power data are scaled by SPICE simulation results of a fanout-of-4 (FO4) inverter under different technologies and supply voltages with predictive technology model (PTM) [31] as shown in Fig. 16.

As discussed in Section 5.2, we could always map one of our designs for multiple times to get a higher throughput while possibly introducing a small overhead. Therefore, it is less meaningful to compare the throughput solely of each design. In this section, we use the metrics of throughput per chip area (Gbps/mm²) and energy per workload bit (nJ/bit) to compare the area efficiency and energy efficiency of various designs. As shown in Table 5, compared to the highly optimized AES ciphers on CPUs with bitslice [15], the proposed AES cipher on AsAP has 3.5-9.2 times higher throughput per unit of chip area and consumes 9.5-11.3 times less energy to encrypt a fixed amount of data. Besides bitslice, SIMD instructions are applied to improve the throughput and efficiency of AES implementations on CPUs further [19]. Even so, our design on AsAP still has 10.7-15.6 times higher throughput per unit of chip area and 8.2-18.1 times lower energy per bit. The TI DSP C6201 is an 8-way VLIW architecture for high performance DSP applications. The referenced data shows that our design has 2 times higher throughput. The area and power numbers of the TI DSP C6201 are not available, but we believe that AsAP has significantly higher throughput per unit of chip area and energy efficiency due to a much smaller core size.

The AES implementation on GeForce 8,800 GTX achieves the highest throughput in the referenced designs, due to its



(a) Delay



(b) Power

Fig. 16. Delay and power of a FO4 inverter based on SPICE simulation using predictive technology model [32]; the general scaling rule assumes a $1/s$ reduction in delay and a $1/(v^2)$ reduction in power where s is the technology scaling factor and v is the voltage scaling factor [33].

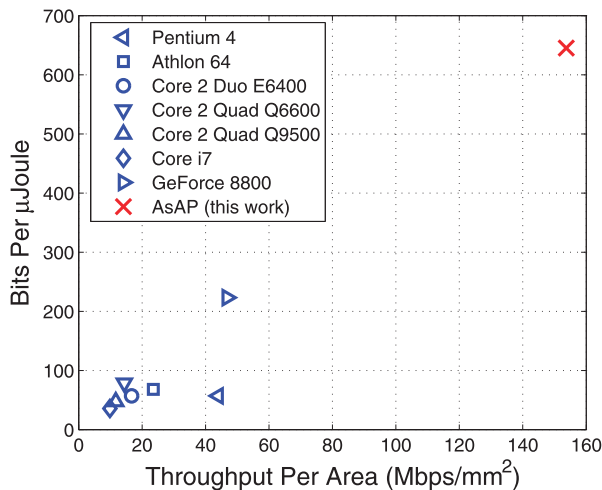


Fig. 17. Comparison of peak performance per area and workload per unit energy of programmable processors. All numbers are scaled based on the PTM simulation results shown in Fig. 16.

large chip area and the utilization of the T-Box method, which works effectively for SIMD architectures with large memory [22]. However, our design still shows a 3.3 times higher throughput per unit of chip area and 2.9 times higher energy efficiency.

Fig. 17 shows that the software AES implementation on AsAP outperforms other software platforms in terms of energy efficiency and performance per area.

8 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, equaling 1.02 mm² 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.

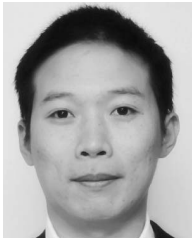
ACKNOWLEDGMENTS

The authors gratefully acknowledge support from US National Science Foundation (NSF) Grants 0430090, 0903549 and CAREER Award 0546907, SRC GRC Grants 1598 and 1971, CSR Grant 1659, UC Micro, ST Microelectronics, Center of Circuit and System Solutions (C2S2), Intel, and Intelliasys. The authors also acknowledge the support of the C2S2 Focus Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity.

REFERENCES

- [1] NIST, "Advanced Encryption Standard (AES)," <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>, Nov. 2001.
- [2] NIST, "Data Encryption Standard (DES)," <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>, Oct. 1999.
- [3] 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.
- [4] D. Mukhopadhyay and D. RoyChowdhury, "An Efficient end to End Design of Rijndael Cryptosystem in 0.18μm CMOS," *Proc. 18th Int'l Conf. VLSI Design*, pp. 405-410, Jan. 2005.
- [5] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, fourth ed. Morgan Kaufmann, 2007.
- [6] S. Morioka and A. Satoh, "A 10-gbps full-AES Crypto Design with a Twisted BDD s-Box Architecture," *IEEE Trans. Very Large Scale Integration Systems*, vol. 12, no. 7, pp. 686-691, July 2004.
- [7] J. Daemen and V. Rijmen, *The Design of Rijndael*. Springer-Verlag, 2002.
- [8] A. Hodjat and I. Verbauwhe, "Area-Throughput Trade-Offs for Fully Pipelined 30 to 70 Gbits/s AES Processors," *IEEE Trans. Computers*, vol. 55, no. 4, pp. 366-372, Apr. 2006.
- [9] S.K. Mathew, F. Sheikh, M. Kounavis, S. Gueron, A. Agarwal, S.K. Hsu, H. Kaul, M.A. Anders, and R.K. Krishnamurthy, "53 gbps Native GF((2⁴)³) Composite-Field AES-Encrypt/Decrypt Accelerator for Content-Protection in 45 nm High-Performance Microprocessors," *IEEE J. Solid-State Circuits*, vol. 46, no. 4, pp. 767-776, Apr. 2011.
- [10] A. Hodjat and I. Verbauwhe, "A 21.54 gbits/s Fully Pipelined AES Processor on FPGA," *Proc. IEEE 12th Ann. Symp. Field-Programmable Custom Computing Machines*, pp. 308-309, Apr. 2004.
- [11] C.-J. Chang, C.-W. Huang, K.-H. Chang, Y.-C. Chen, and C.-C. Hsieh, "High Throughput 32-Bit AES Implementation in FPGA," *Proc. IEEE Asia Pacific Conf. Circuits and Systems*, pp. 1806-1809, Nov. 2008.
- [12] J. Granado-Criado, M. Vega-Rodriguez, J. Sanchez-Perez, and J. Gomez-Pulido, "A New Methodology to Implement the AES Algorithm Using Partial and Dynamic Reconfiguration," *Integration, the VLSI J.*, vol. 43, no. 1, pp. 72-80, 2010.
- [13] S. Qu, G. Shou, Y. Hu, Z. Guo, and Z. Qian, "High Throughput, Pipelined Implementation of AES on FPGA," *Proc. Int'l Symp. Information Eng. and Electronic Commerce*, pp. 542-545, May 2009.
- [14] "Int'l Technology Roadmap for Semiconductors, Design," http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_Design.pdf, 2009.
- [15] M. Matsui and J. Nakajima, "On the Power of Bitslice Implementation on Intel Core 2 Processor," *Proc. Cryptographic Hardware and Embedded Systems (CHES '07)*, pp. 121-134, 2007.
- [16] E. Biham, "A Fast New DES Implementation in Software," *Proc. Fourth Int'l Workshop Fast Software Encryption*, pp. 260-272, 1997.
- [17] D. Bernstein and P. Schwabe, "New AES Software Speed Records," *Proc. INDOCRYPT '08: Ninth Int'l Conf. Cryptology in India: Progress in Cryptology*, pp. 322-336, 2008.
- [18] "Supplemental Streaming SIMD Extensions 3," <http://en.ikipedia.org/wiki/SSSE3>, 2012.
- [19] E. Kasper and P. Schwabe, "Faster and Timing-Attack Resistant AES-GCM," *Proc. 11th Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '09)*, pp. 1-17, 2009.
- [20] S. Gueron, "Intel Advanced Encryption Standard (AES) Instructions Set," Jan. 2010.
- [21] T. Wollinger, M. Wang, J. Cuajardo, and C. Paar, "How Well are High-end DSPs Suited for the AES Algorithm?," *Proc. Third AES Candidate Conf.*, pp. 94-105, Apr. 2000.
- [22] S.A. Manavski, "CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptography," *Proc. IEEE Int'l Conf. Signal Processing and Comm.*, pp. 65-68, Nov. 2007.
- [23] S. Borkar, "Thousand Core Chips: A Technology Perspective," *Proc. 44th Ann. Design Automation Conf.*, pp. 746-749, 2007.
- [24] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, T. Jacobson, G. Landge, M. Meeuwse, C. Watnik, P. Meija, A. Tran, J. Webb, E. Work, Z. Xiao, and B. Baas, "A 167-Processor 65 nm Computational Platform with Per-Processor Dynamic Supply Voltage and Dynamic Clock Frequency Scaling," *Proc. IEEE Symp. VLSI Circuits*, June 2008.

- [25] A.T. Tran, D.N. Truong, and B.M. Baas, "A Reconfigurable Source-Synchronous On-Chip Network for GALS Many-Core Platforms," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 6, pp. 897-910, June 2010.
- [26] D.N. Truong, W.H. Cheng, T. Mohsenin, Z. Yu, A.T. Jacobson, G. Landge, M.J. Meeuwsen, A.T. Tran, Z. Xiao, E.W. Work, J.W. Webb, P. Mejia, and B.M. Baas, "A 167-Processor Computational Platform in 65 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 44, no. 4, pp. 1130-1144, Apr. 2009.
- [27] B. Baas, Z. Yu, M. Meeuwsen, O. Sattari, R. Apperson, E. Work, J. Webb, M. Lai, T. Mohsenin, D. Truong, and J. Cheung, "AsAP: A Fine-Grained Many-Core Platform for DSP Applications," *IEEE Micro*, vol. 27, no. 2, pp. 34-45, Mar. 2007.
- [28] E.W. Work, "Algorithms and Software Tools for Mapping Arbitrarily Connected Tasks onto an Asynchronous Array of Simple Processors," MS thesis, Univ. of California, Davis, Sept. 2007.
- [29] Z. Yu and B.M. Baas, "A Low-Area Multi-Link Interconnect Architecture for GALS Chip Multiprocessors," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 5, pp. 750-762, May 2010.
- [30] M. Butler, "AMD Bulldozer Core—A New Approach to Multi-threaded Compute Performance for Maximum Efficiency and Throughput," *Proc. IEEE HotChips Symp. High-Performance Chips (HotChips '10)*, Aug. 2010.
- [31] A. Stillmaker, "Exploration of Technology Scaling of CMOS Circuits from 180 nm to 22 nm Using PTM Models in HSPICE," technical report, UC Davis, June 2011.
- [32] W. Zhao and Y. Cao, "New Generation of Predictive Technology Model for Sub-45 nm Early Design Exploration," *IEEE Trans. Electron Devices*, vol. 53, no. 11, pp. 2816-2823, Nov. 2006.
- [33] J.M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, second ed. Prentice-Hall, 2003.



Bin Liu received the BS degree in information engineering from Shanghai Jiao Tong University, China, in 2007, and the MS degree in electrical and computer engineering from the University of California, Davis, in 2010, where he is currently toward the PhD degree. His research interests include high-performance multicore processor architecture, variation-aware dynamic and frequency scaling algorithms and circuits, and parallel encryption engine implementations.

He received the Best Paper nomination at Asilomar 2011. He is a student member of the IEEE.



Bevan M. Baas received the BS degree in electronic engineering from California Polytechnic State University, San Luis Obispo, in 1987, and the MS and PhD degrees in electrical engineering from Stanford University, CA, in 1990 and 1999, respectively. From 1987 to 1989, he was with Hewlett-Packard, Cupertino, CA, where he participated in the development of the processor for a high-end minicomputer. In 1999, he joined Atheros Communications, Santa

Clara, CA, as an early employee and served as a core member of the team which developed the first IEEE 802.11a (54 Mbps, 5 GHz) Wi-Fi wireless LAN solution. In 2003 he joined the Department of Electrical and Computer Engineering at the University of California, Davis, where he is now an associate professor. He leads projects in architecture, hardware, software tools, and applications for VLSI computation with an emphasis on DSP workloads. Recent projects include the 36-processor Asynchronous Array of simple Processors (AsAP) chip, applications, and tools; a second generation 167-processor chip; low density parity check (LDPC) decoders; FFT processors; viterbi decoders; and H.264 video codecs. Dr. Baas was a US National Science Foundation fellow from 1990 to 1993 and a NASA Graduate Student Researcher fellow from 1993 to 1996. He was a recipient of the US National Science Foundation CAREER Award in 2006 and the Most Promising Engineer/Scientist Award by AISES in 2006. He received the Best Paper Award at ICCD 2011, and Best Paper nominations at Asilomar 2011 and BioCAS 2010. During the summer of 2006 he was a visiting professor in Intel's Circuit Research Lab. He is an associate editor for the *IEEE Journal of Solid-State Circuits* and *IEEE Micro* guest editor in 2012. He has been the program committee cochair of HotChips in 2011, and program committee member of Hotchips in 2009-2010, of ICCD in 2004-2005 and 2007-2009, of ASYNC in 2010, and of the ISSCC SRP Forum in 2012. He is a senior member of the IEEE.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.