Parallel Software Encryption of AES Algorithm by Using CAM-Based Massive-Parallel SIMD Matrix Core for Mobile Accelerator

Kyosuke Kageyama [1,*], Sota Arai [2], Hajime Hamano [2], Xiangbo Kong [2], Takeshi Kumaki [2], and Tetsushi Koide [3]

[1] Department of Electrical, Electronic and Communication Engineering, Kindai University, Osaka, Japan
[2] Department of Electronic and Computer Engineering, Ritsumeikan University, Shiga, Japan;
Email: {ri0077er @ed, ri0093fr@ed, kong@fc, kumaki@fc}.ritsumei.ac.jp (S.A., H.H., X.K., T.K.)
[3] Research Institute for Nanodevices (RIND), Hiroshima University, Hiroshima, Japan;
Email: koide@hiroshima-u.ac.jp (T.K.)
*Correspondence: kageyama@ele.kindai.ac.jp (K.K.)

***Abstract*—Recently, it has become possible to execute various digital multimedia applications, such as image compression, video compression, and audio processing, on mobile devices — as long as the processing core in the mobile device has the required high levels of performance, versatility, and programmability. Generally speaking, multimedia applications operate by performing repeated arithmetic and table-lookup coding operations. Therefore, to make it easier to achieve those required high levels of performance, versatility, and programmability, we propose an accelerator for mobile Central Processing Units (CPUs) known as a Content Addressable Memory-based massive-parallel Single Instruction Multiple Data (SIMD) Matrix Core (CAMX) that improves the processing speeds of both arithmetic and table-lookup coding operations. Our proposed CAMX, which is equipped with two CAM modules, has highly parallel processing capabilities that facilitate fast table-lookup coding operations. In fact, the results of Advanced Encryption Standard (AES) encryption simulations conducted in this study show that its AES encryption total clock cycles are 1,362,699. Additionally, a detailed breakdown of the number of clock cycles shows 1,312,160 for SubBytes, a combined total of 17,161 for ShiftRows and MixColumns, and 2519 for AddRoundKey. This paper also confirmed that CAMX could process AES encryptions at a rate of 83.17 clock cycles/byte. Also, the performance of CAMX, related works, and existing mobile processors are compared. The related works do not have a dedicated circuit for AES processing. From the comparison results, CAMX provides a performance improvement of approximately 4.4- and 3569.1-times over the related works. The existing mobile processors are Texas Instruments (TI) DM3730 and a TI OMAP3530. From the comparison results, CAMX provides a performance improvement of approximately 2.1 times over TI DM3730 and TI OMAP3530.**

***Keywords*—CAMX, CAM, parallel processing, Single Instruction Multiple Data (SIMD), Advanced Encryption Standard (AES)**

I. Introduction

Recently, spread by rapid advancements in semiconductor technology performance, smartphones that can quickly process sounds and images have entered widespread use in our daily lives. In addition, applications hosted on Internet websites are often downloaded onto such smartphones where they are expected to perform functions that include executing secure encrypted communications [1, 2]. This means current smartphones must provide increasingly high levels of performance, versatility, and programmability within the constraints of small size and low power consumption [3, 4]. Therefore, it is becoming increasingly important for smartphones, as well as other mobile devices, to be capable of processing multimedia applications on a single core. This concept is called "digital convergence" [5]. Such multimedia applications need the ability to process both repeated arithmetic operations and table-lookup coding operations at high speed. Note that in this study, repeated arithmetic operations are defined as AND, OR, XOR, addition, and multiplication operations, while table-lookup coding operations are used for input data to output data conversions via search operation. Furthermore, while parallel processing capabilities of both repeated arithmetic operations and table-lookup coding are needed for high-performance mobile devices, table-lookup coding operations are particularly difficult to process in parallel [6–8].

Accordingly, to make it easier to achieve those required performance levels, we propose an accelerator for mobile CPUs known as a Content Addressable Memory-based massive-parallel Single Instruction Multiple Data (SIMD) matrix core (hereafter, CAMX). Our proposed CAMX core readily handles basic instructions, which include AND, OR, XOR, addition, search, etc., and can process repeated arithmetic and table-lookup coding operations in parallel because it

Manuscript received July 4, 2022; revised September 7, 2022; accepted October 13, 2022; published April 17, 2023.

features two CAM modules specialized for use in search operations. CAMX can use simple instructions to execute various processing at high levels of performance via a concept referred to as "cellular automaton on content addressable memory" (CAM^2) and MX-1 [9–12]. Our previous researches confirmed the repeated arithmetic operation mainly, on the other hand, this paper confirms the table-lookup coding operation. Therefore, this paper focuses on simulations involving the Advanced Encryption Standard (AES), which is a well-known process for performing basic table-lookup coding operations. This paper also explains the CAMX architecture and compares CAMX with existing technologies.

The rest of this paper is organized as follows: Section II surveys two related works that do not have dedicated circuits of AES processing, like CAMX. Section III explains the CAMX architecture. Section IV shows AES encryption processing. Section V explains the processing flows of AES encryption using CAMX. Section VI shows AES encryption results using CAMX. Section VII compares the results of Section VI and other techniques, and Section VIII shows our conclusions.

II. Related Works

In this section, the related works are introduced. The CAMX architecture can execute various processing to provide versatility as an accelerator for mobile CPUs. Therefore, it does not have a dedicated circuit for AES processing. For this reason, the related works without dedicated circuits for AES processing are surveyed.

Muri and Fortier explain a Processor-in-Memory (PIM) computer architecture, and PIM is implemented in AES processing [13]. PIM can execute various processing types in memory. Therefore, PIM can execute processing between memory performance and processor performance without delays. In the results of [13], PIM execute AES encryption at 5797 clock cycles.

Sideris and Sanida *et al.* explain the AES processing implementation using a NIOS-II processor [14]. This implementation is compared using two techniques, without custom instruction and with floating point 2. NIOS-II processors are used in embedded devices, and floating point 2, which executes basic arithmetic operations at a low clock cycle, has custom instruction implementations for additional floating point operations. In the results of [14], the clock cycle of the technique without custom instruction is 6,120,265, and the clock cycle of the technique with floating point 2 is 4,749,510.

III. Content Addressable Memory-Based Massive-Parallel SIMD Matrix Core Architecture

In this section, the architecture and operational concepts for our proposed CAMX are shown. Fig. 1 shows CAMX architecture, which consists of two content addressable memories (left and right CAM modules), *n* Processing Elements (PEs), an interface module, and a controller. A large number of small 1-bit PEs are located between the left and right CAM modules. CAMX adopts bit-serial and word-parallel processing concepts to achieve highly parallel processing and uses PE processes for *x*-bit stored data, which means pipeline processing can be executed because the two CAM modules are controlled alternately. In addition, the CAM module size (*x*-bit) and the number of PEs (*n* entries) can be changed adaptably.

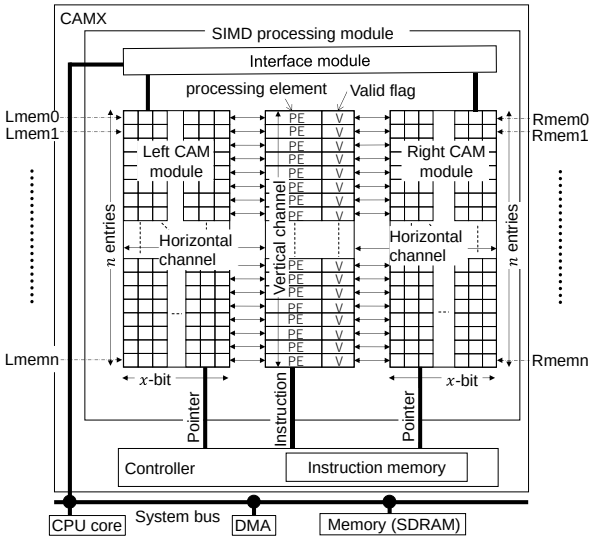

Figure 1. CAMX architecture.

A. Bit-Serial and Word-Parallel Processing

CAMX can process a large volume of stored data in parallel as bit-serial and word-parallel. Generally speaking, most conventional architectures operate with bit-parallel and word-serial operations. Therefore, multimedia data are processed sequentially in pipeline processing. On the other hand, our proposed CAMX supports *n*-way bit-serial and word-parallel operations. Thus, all PEs can be synchronized with a single command, and all stored data can be processed simultaneously. The internal processing of PE also executes pipeline processing. For example, when the left CAM module sends 1-bit data to PEs at some clock cycle, the right CAM module sends 1-bit data to the PEs at next clock cycle. These processing can be executed repeatedly to calculate the required bit width.

B. SIMD Processing Operation

The CAMX core can execute multiple processing with one instruction command. Hence, when the controller outputs an instruction to all PEs, it is executed by all PEs in parallel. Furthermore, because CAMX is equipped with two CAM modules, it can select whether to use a search operation to execute PEs for each entry. CAM module search operations are explained in Fig. 2. Note that both CAM modules can search the matched data from a contents table by comparing data [15].

When stored data in either the left or right CAM modules are matched with comparison data, the CAM module outputs the match signal results. In addition, since each CAM module can use mask data to determine which search-bit position to use when either the left or

right CAM module uses match data to output match signals in the valid flag (V in Fig. 2), only the matched entries become active and can be used when performing calculations in the processing element (PE in Fig. 2). This makes it possible for CAMX to simultaneously execute any entries data in parallel.

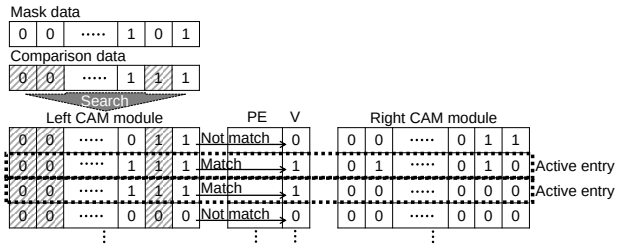

Figure 2. Block diagram and function of CAM module.

C. Basic Processing Flow

The PEs can calculate all stored data in either the left and right CAM modules, and, as shown in Fig. 1, each CAM module has *n* entries *x*-bit data. Since CPU core and CAMX are connected on the system bus, CAMX works as an accelerator for the embedded processor. The CPU core processes sequential operations and CAMX executes parallel operations in all programming codes. The basic processing procedure is shown in the items below and Fig. 1.

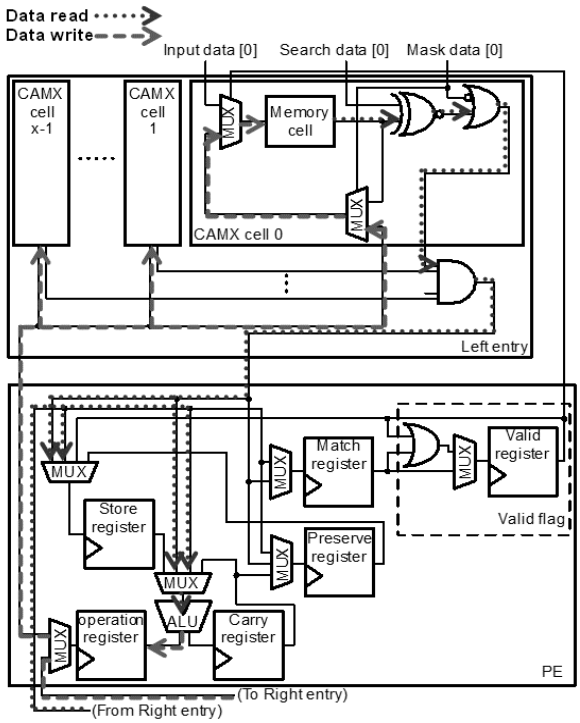

Figure 3. Detailed block diagram of PE and CAMX cell.

(1) The CPU reads target data from a synchronous dynamic random-access memory (SDRAM) and sends the data to CAMX for processing via the interface module. The CPU can also access stored data with read and write addresses in CAMX. Note that all read-out data are stored vertically in both the left and right CAM modules.

(2) These data are sent to PEs, where they are calculated simultaneously in arithmetic logic units (ALUs). These operations continue sequentially until the least significant bit (LSB) of the *x*-bit stored data is obtained. To facilitate understanding of the operation flow, Fig. 3 shows the structure of the PE and CAMX cells. When 1-bit data in a CAM cell are sent to PEs, the 1-bit data are deposited in the store registers. In the next clock cycle, stored 1-bit data from the other CAM module are sent to the PEs and calculated in ALUs. The calculated data are then stored in the operation register. Since these operations are handled as pipeline processing, CAMX calculates *n* entries simultaneously and can execute them as bit-serial word-parallel processing.

(3) After processing the 1-bit operation, the calculated data are stored in the left and right CAM modules in parallel.

D. CAMX Instruction Command Specifications

Here, CAMX instruction command specifications are explained. CAMX has basic and search instructions because it requires versatility and programmability as the mobile device accelerator. These instructions are sent to CAMX from the CPU, and the controller of CAMX executes processing at all left and right CAM modules entries and PEs. Fig. 4 shows the CAMX instruction command specifications. CAMXLIB means the CAMX instruction command, and the instruction details are specified by A, B, C, D, and E. A specifies the left or right CAM module for storing the processing result data by LEFT_WING or RIGHT_WING. B specifies the instructions. For example, the XOR instruction is CAMX_DATA_XOR, the AND instruction is CAMX_DATA_AND, etc. C specifies the processing data bit width. In this paper, because the CAM modules have 1,024 entries and 256 bits, C uses 8 bits with binary (8'bxxxx_xxxx). D and E specify the processing data bit positions of the left and right CAM modules. CAMX processes the bit width of C from the positions of D and E. D and E use 8'bxxxx_xxxx like C.

The basic instructions of Fig. 4(a) are logical and arithmetic operations (XOR, AND, OR, addition, etc.). As an example, the XOR instruction is explained. In this example XOR instruction, because D and E is 8'b0000_0000, this instruction processes the stored data from the least significant bit positions of the left and right CAM modules. In addition, because C is 8'b0000_0011, the stored data are executed using the XOR instruction for the 4-bit width. The XOR instruction result data are stored in the left CAM module because A is LEFT_WING. In Fig. 5, this processing image is explained. CAMX reads 4-bit data from the least significant bit position of the left and right CAM modules, and the data are sent to the PEs. The calculated data are stored in the left CAM module. CAMX executes all entries in parallel.

```
<CAMX instruction command specifications>
    CAMXLIB = {A, B, C, D, E};
        A : Specify left or right CAM module for the stored data
            (LEFT_WING stores the data in left CAM module. RIGHT_WING stores the data in right CAM module)
        B : Specify instruction (CAMX_DATA_XOR is XOR instruction, CAMX_DATA_AND is AND instruction, etc.)
        C : Specify the processing data of CAM module entry width with 8'bxxx_xxxx
        D, E : Specify the processing data bit position of CAM module entry with 8'bxxx_xxxx

(a) Basic instruction (XOR instruction example)
    CAMXLIB = {LEFT_WING, CAMX_DATA_XOR, 8'b0000_0011, 8'b0000_0000, 8'b0000_0000};

(b) Search instruction
    SEARCH_DIN = 256'h0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0007;
    MASK_DIN   = 256'h0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0005;
    CAMXLIB = {LEFT_WING, CAMX_MASK_SEARCH, 8'b0000_0000, 8'b0000_0000, 8'b0000_0000};

(c) Rewrite instruction
    CAMXLIB = {LEFT_WING, CAMX_ALL_WRITE, 8'b0000_0011, 8'b0000_0010, 8'b0000_0000};
```

Figure 4. CAMX instruction command specifications.

The search instruction is executed as shown in Fig. 4(b). CAMXLIB means the CAMX instruction. B specifies the CAMX_MASK_SEARCH used to process the search instructions. In addition, the search instruction needs additional instruction commands, which are SEARCH_DIN and MASK_DIN. SEARCH_DIN is the comparison data in Fig. 2, and MASK_DIN is the mask data. These data are sent to CAMX from the CPU. In this paper, CAMX has 1,024-entry 256-bit CAM modules. Thus, SEARCH_DIN and MASK_DIN specify search bit positions in 256 bits. Here, Fig. 4(b) shows the search instructions used for Fig. 2. In Fig. 2, the comparison data and the mask data are "111" and "101" at the lower 3-bit position. Therefore, the SEARCH_DIN and MASK_DIN of the instruction command are specified with 256 bits, as shown in Fig. 4(b). In addition, because the left CAM module is searched, A and B are specified as LEFT_WING and CAMX_MASK_SEARCH in CAMXLIB. Also, because the search instruction does not need the bit width and position, C, D, and E are specified as 8'b0000_0000. The search instruction results are stored in the valid flag. When the valid flag is 1, the entry is active and can execute the next instruction (Fig. 2).

Fig. 4(c) shows the rewrite instruction for the CAM module data. B specifies CAMX_ALL_WRITE in CAMXLIB. In this example, A is specified as LEFT_WING. Also, C is 8'b0000_0011, D is 8'b0000_0010, and E is 8'b0000_0000. C specifies the rewritten bit width. D specifies the rewritten data. E specifies the bit position. Therefore, the left CAM module is rewritten as 4-bit data to 0010 from the least significant bit. This instruction image is shown in Fig. 6.

From the above, we see that CAMX can execute processing by combining these instruction commands. When CAMX converts the data, it executes the search instruction. Thus, CAMX searches the match data in the CAM module, and the entries with the match data are stored as 1 in the valid flag. The matched entries activate, and CAMX executes the rewrite instruction in the next clock cycles. Thus, just the matched data are converted in parallel.

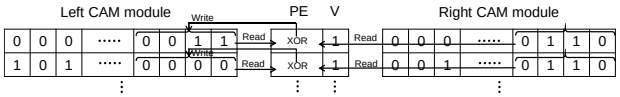

Figure 5. Basic instruction flow.

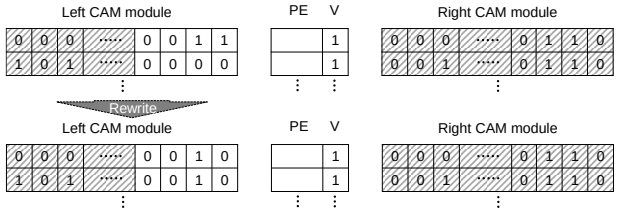

Figure 6. Rewrite instruction flow.

IV. AES Encryption

AES is a common key cryptosystem system that uses block cipher [16]. In this study, the AES uses a 128-bit cipher key and performs processing in four steps (SubBytes, ShiftRows, MixColumns, and AddRoundKey), which are repeated every 10 rounds.

1. SubBytes
   Data are converted in 8-bit increments via S-box, which is a non-linear table. Since SubBytes is a table-lookup coding operation, it is generally difficult to process it in parallel.
2. ShiftRows
   Since data are processed via cyclic shifts to the left, ShiftRows is generally easy to process in parallel.
3. MixColumns
   In this process, Galois theory is used to calculate data via matrix operations. Parallel processing for MixColumns is generally easy because it employs repeated arithmetic operations.
4. AddRoundKey
   XOR operations data for AddRoundKey are processed by cipher key. Additionally, since AddRoundKey also consists of repeated arithmetic operations, it is generally easy to process in parallel.

V. AES Encryption of CAMX

This section explains the AES encryption flow of CAMX. Since CAMX PEs comprise simple logical and adder circuits, the CAMX processes AES encryption via basic, OR, XOR, addition, and search instructions. Fig. 7 shows the 1-round AES encryption flow. Incidentally, the CAMX processes AES encryption against the data of all *n* entries in parallel. Fig. 7 shows one entry out of *n* entries.

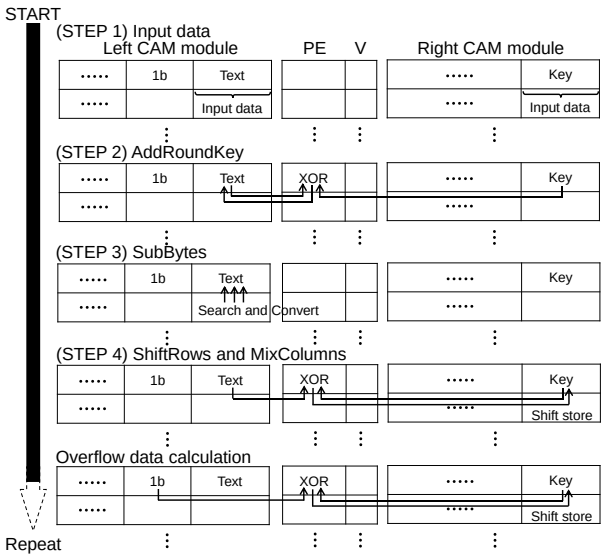

Figure 7. AES encryption flow of CAMX.

Step 1: The CAMX data are stored in plain text and Key. The Plain text data are stored in the left CAM module, and the Key data are stored in the right CAM module. In addition, the left CAM module is stored "1b" in hexadecimal to calculate overflow data.

Step 2: The Plain text and Key data stored in the left and right CAM modules are processed via XOR operations as AddRoundKey. The left and right CAM modules data are read to PE, and PE calculates XOR operations. The calculated data is stored in left CAM module.

Step 3: The AddRoundKey results are converted to SubBytes. The upper and lower data of the left CAM module are searched in four-bit increments, and the match data are converted to an S-box table. The CAMX can search the CAM modules data by comparison and mask data. Here, the comparison and mask data are sent for search operation from the CPU each clock cycles, and these data are the same as S-box table data. The only match entries of the left CAM module are rewritten to the S-box table data.

Step 4: The SubBytes result is processed as ShiftRows and MixColumns, which the CAMX can execute simultaneously. The left CAM module data are doubled and tripled, then calculated via XOR and stored in the right CAM module. At next cycle, overflow data are calculated XOR operation via "1b" in hexadecimal of left CAM module. When the data are stored in the right CAM module, they are shifted.

Step 5: These processing steps (Steps 1–4) are repeated 10 times.

VI. Experimental Results

A. Simulation Waveform Result

To verify the effectiveness of our proposed CAMX architecture, this paper performed AES encryption simulation experiments using the Xilinx Vivado v2019.2.1 development tool. In these experiments, the two CAM modules installed in the CAMX provided 1024 (= *n*) entries vertically and 256 bits (= *x*) horizontally. Experimental instructions were executed simultaneously for 1024 entries, and the results were output as a waveform. Fig. 8 waveform shows one of 1024 entries as an example.

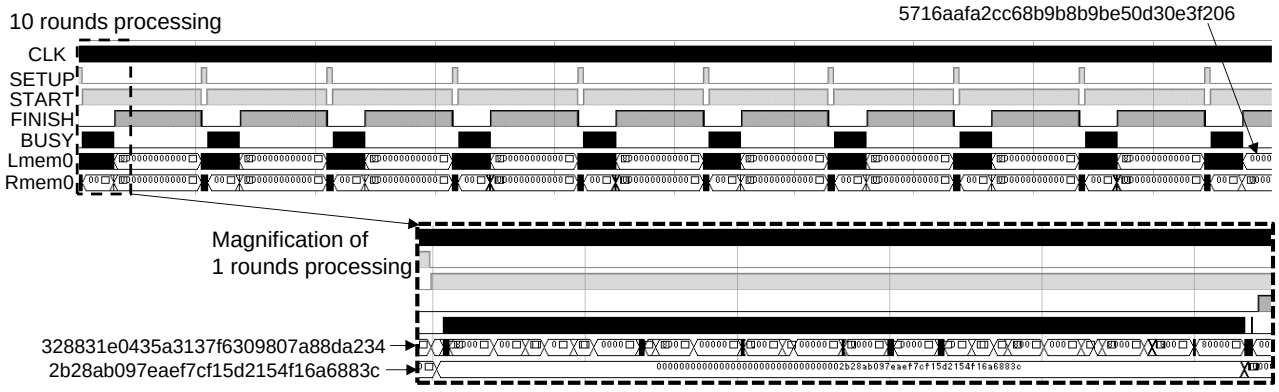

Figure 8. Simulation waveform result.

Here, *CLK* is the system clock, and *SETUP* is the preparation process. *START* is the start of the operation, and *FINISH* is the end of the operation. *BUSY* is the instruction time signal, while *Lmem* and *Rmem* are the stored data in the left and right CAM modules, respectively. This waveform is processed 10 times because AES requires 10 rounds of processing. Also, a magnification of 1 round processing is shown in Fig. 8. In

the obtained results, the left CAM module stores "328831e0435a3137f6309807a88da234" in hexadecimal as plain text, while right CAM module stores "2b28ab097eaef7cf15d2154f16a6883c" in hexadecimal as the Key. After these data are processed in the CAMX PEs, "5716aafa2cc68b9b8b9be50d30e3f206" is stored in left CAM module as the encrypted text, thus confirming the ability of CAMX to accurately process AES with 1,024 entries in parallel.

B. AES Encryption Clock Cycles of the CAMX

Next, AES encryption clock cycles were obtained and shown in Table I, where it can be seen that, during the operation time, the total number of clock cycles was 1,362,699. Additionally, a detailed breakdown of the number of clock cycles shows 1,312,160 for SubBytes, a total of 17,161 for ShiftRows and MixColumns, and 2519 for AddRoundKey. Note that since CAMX can execute left and right CAM module entries in parallel, operations can proceed at a constant clock cycle rate regardless of the number of entries.

Table I also shows cycle/byte of AES encryption. In this study, CAMX executes AES encryption against data for all 1024 entries in 1,362,699 clock cycles, and the processed data are output as 16-byte encrypted text. Therefore, CAMX could process AES encryption at 83.17 clock cycles/byte.

TABLE I. THE NUMBER OF CLOCK CYCLES

Clock cycles					The number of AES encryption clock cycles/byte
SubBytes	ShiftRows MixColumns	AddRoundKey	Other (Input, Output)	Total	
1,312,160	17,161	2,519	30,859	1,362,699	83.17

VII. COMPARISON WITH OTHER TECHNIQUES

A. The CAMX and the Related Works

Here, The CAMX and the related works are compared in terms of clock cycles/byte of AES encryption. From Sec. II, the AES encryption clock cycles of [13] and [14] are 5797 and 4,749,510. The processed data of the two techniques are output as 16-byte encrypted text. Therefore, the clock cycles/byte become 362.31 and 296,844.38. On the other hand, the clock cycles/byte of the CAMX are 83.17.

This indicates that CAMX provides a performance improvement of approximately 4.4- and 3569.1-times over [13] and [14], respectively. Fig. 9 shows each performance. These results indicate that CAMX can execute AES encryption with high throughput using high-parallel processing. The reason is that CAMX can achieve 1024 parallel processing.

B. The CAMX and the Existing Mobile Processors

Here, the throughput of CAMX and existing mobile processors are compared. The existing mobile processors used in our study are a Texas Instruments (TI) DM3730 (1.00 GHz) embedded in a BeagleBoard-xM [17], which exploits the Advanced RISC Machines (ARM) Cortex A8 core 32K/32K [18]; and a TI OMAP3530 (720 MHz) embedded in a BeagleBoard [19], which exploits the ARM Cortex A8 core 16K/16K [20]. To begin the comparison, this paper first executed AES encryptions using the three processors and acquired the average number of clock cycles.

Fig. 10 shows the obtained throughputs of these processors. Here, the clock cycles/byte for TI DM3730 and TI OMAP3530 are 171.27 and 175.85, respectively. In contrast, clock cycles/byte for CAMX is 83.17, thus indicating CAMX provides a performance improvement of approximately 2.1 times over TI DM3730 and TI OMAP3530, respectively. These results also indicate that CAMX can execute AES encryption with high throughput using high-parallel processing. The reason is that CAMX can achieve 1,024 parallel processing. From these comparisons, CAMX provides high levels of performance, versatility, and programmability as a mobile device accelerator.

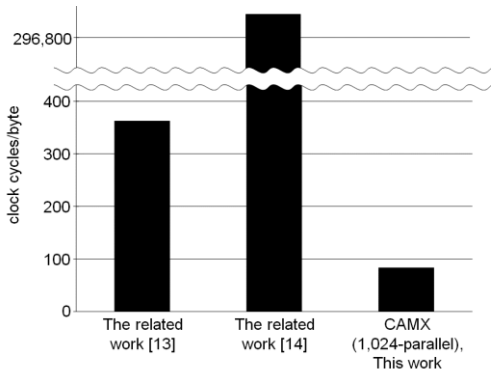

Figure 9. Comparison with the related works.

VIII. CONCLUSION

In this paper, a content addressable memory-based massive-parallel SIMD matrix core (CAMX) processor is proposed as a method for improving the processing of both repeated arithmetic operations and table-lookup coding operations. As explained above, CAMX is designed to serve as an accelerator for mobile CPU cores, is configured with two CAM modules and PEs, and can execute *n* entries in parallel. Additionally, the two CAM modules support 1,024 entries vertically and 256 bits horizontally.

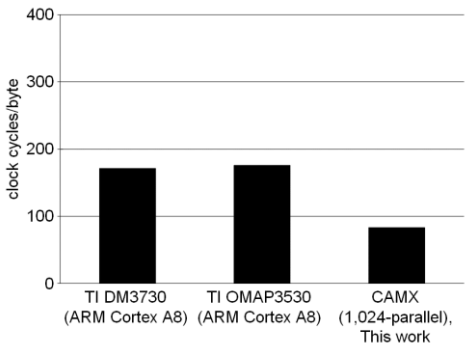

Figure 10. Comparison with the existing mobile processors.

From the results of experiments conducted during this study, the number of AES encryption total clock cycles was 1,362,699. Furthermore, a breakdown of the number of clock cycles shows 1,312,160 for SubBytes, a total of 17,161 for ShiftRows and MixColumns, and 2519 for AddRoundKey. This paper was also shown that CAMX could process AES encryption in 83.17 clock cycles/byte, and it provides an approximately 2.1-times performance improvement over the TI DM3730 and TI OMAP3530 processors. Also, CAMX provides an approximately 4.4- and 3569.1-times performance improvement over the related works.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Kyosuke Kageyama conducted the research and wrote the paper; Sota Arai, Hajime Hamano, Xiangbo Kong, Tetsushi Koide, and Takeshi Kumaki verified CAMX architecture; all authors had approved the final version.

FUNDING

This work has been supported by a Grant-in-Aid for Scientific research (C) (No. 19K04461), Ministry of Education, Culture, Sports, Science and Technology, Japanese government and a Grantin-Aid for JSPS Fellows, 2019. The research has been partly executed in response to support of the Murata Science Foundation. Part of this work was supported by collaborative research of the Research Center for Biomedical Engineering.

REFERENCES

[1] B. Daddala, H. Wang, and A. Y. Javaid, "Design and implementation of a customized encryption algorithm for authentication and secure communication between devices," in *Proc. 2017 IEEE National Aerospace and Electronics Conference (NAECON)*, pp. 258–262, 2017.

[2] A. Dey, S. Nandi, and M. Sarkar, "Security measures in IOT based 5G networks," in *Proc. 2018 3rd International Conference on Inventive Computation Technologies (ICICT)*, pp. 561–566, 2018.

[3] M. Halpern, Y. Zhu, and V. J. Reddi, "Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction," in *Proc. 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 64–76, 2016.

[4] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "A survey on mobile augmented reality with 5G mobile edge computing: Architectures, applications, and technical aspects," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1160–1192, 2021.

[5] K. Uchiyama, "Processor technology in system LSI," *Journal of the Institute of Electronics, Information and Communication Engineers*, vol. 95, no. 7, pp. 582–588, 2012.

[6] D. Knox and S. Panchanathan, "Parallel searching techniques for routing table lookup," in *Proc. IEEE INFOCOM'93 the Conference on Computer Communications*, pp. 1400–1405, 1993.

[7] L. Hung and Y. Chen, "Parallel table lookup for next generation internet," in *Proc. 2008 32nd Annual IEEE International Computer Software and Applications Conference*, pp. 52–59, 2008.

[8] A. M. Fiskiran and R. B. Lee, "Fast parallel table lookups to accelerate symmetric-key cryptography," in *Proc. International Conference on Information Technology: Coding and Computing*, 2005.

[9] M. Nakajima, H. Noda, K. Dosaka, K. Nakata, M. Higashida, O. Yamamoto, K. Mizumoto, H. Kondo, Y. Shimazu, K. Arimoto, K. Saitoh, and T. Shimizu, "A 40GOPS 250mW massively parallel processor based on matrix architecture," in *Proc. 2006 IEEE International Solid State Circuits Conference*, 2006.

[10] K. Kageyama, T. Koide, and T. Kumaki, "Parallel processing of morphological pattern spectrum for a massive-parallel memory-embedded SIMD matrix processor MX-1," *IEEJ Transactions on Electronics, Information and Systems*, vol. 139, no. 3, pp. 237–246, 2019.

[11] K. Kageyama, A. Sekino, K. Watanabe, A. Hamai, T. Koide, and T. Kumaki, "Proposal of content addressable memory-based massive-parallel SIMD matrix core," in *Proc. RISP International workshop on Nonlinear Circuit, computer and Signal Processing (NCSP)*, 2020.

[12] K. Kageyama, K. Watanabe, A. Hamai, T. Koide, and T. Kumaki, "Acceleration of arithmetic processing with CAM-based massive-parallel SIMD matrix core," in *Proc. IEEE International MidWest Symposium on Circuits and Systems (MWSCAS)*, 2020.

[13] R. Muri and P. Fortier, "Embedded processor-in-memory architecture for accelerating arithmetic operations," in *Proc. 2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019.

[14] A. Sideris, T. Sanida, and M. Dasygenis, "Hardware acceleration of the AES algorithm using Nios-II processor," in *Proc. 2019 Panhellenic Conference on Electronics & Telecommunications (PACET)*, 2020.

[15] K. E. Grosspietsch, "Associative processors and memories: A survey," *IEEE Micro*, vol. 12, no. 3, pp. 12–19, 1992.

[16] X. Zhang and K. K. Parhi, "Implementation approaches for the advanced encryption standard algorithm," *IEEE Circuits and Systems Magazine*, vol. 2, pp. 24–46, 2002.

[17] BeagleBoard-xM. [Online]. Available: https://beagleboard.org/beagleboard-xm

[18] DM3730. Digital Media Processor. [Online]. Available: https://www.ti.com/product/DM3730

[19] BeagleBoard. [Online]. Available: https://beagleboard.org/beagleboard

[20] OMAP3530. Applications Processor. [Online]. Available: https://www.ti.com/product/OMAP3530

Kyosuke Kageyama received his B.S. degree from Department of VLSI System Design, and completed the first half of the M.E. program in Department of Electrical and Electronic Engineering from Ritsumeikan University, Shiga, Japan, in 2014 and 2016, respectively. From 2016 to 2017, he was with Scientific Research Institute of Mie Prefectural Police Headquarters. From 2017 to 2019, he was a visiting scholar of Ritsumeikan University, Shiga, Japan. From 2019 to 2021, he was with Kyoto City Fire Department. From 2017 to 2022, he was with Japan Fire and Disaster Management Agency. Since 2022, he has been an assistant professor in the Department of Electrical, Electronic and Communication Engineering, Kindai University, Osaka, Japan, and he has been a visiting scholar of Ritsumeikan University, Shiga, Japan. His research interests include content addressable memory, SIMD processing architecture, visible light communication, image processing, and these applications. He is a member of the Institute of Electrical and Electronics Engineers (IEEE).

Sota Arai received his B.S. degree from Department of Electronic and Computer Engineering from Ritsumeikan University, Shiga, Japan, in 2020. His research interests include content addressable memory, SIMD processing architecture, image processing, and these applications.

Hajime Hamano was born in Osaka, Japan. He received the bachelor degree in electronics and computer engineering from Ritsumeikan University in Shiga, Japan, 2022. Currently he is major in advanced electrical, electronic and computer systems in Graduate School, Ritsumeikan University. He is treasure of IEEE Student Branch at Ritsumeikan University. His research related to apply that parallel processing architecture for embedded devices to accelerate machine learning and multimedia applications.

Xiangbo Kong received B.E. degree from Nankai University in 2012 and he received M.E. and Ph.D degrees from Ritsumeikan University in 2018 and 2020, respectively. In 2020, he joined the College of Science and Engineering, Ritsumeikan University as an assistant professor. His research interests include artificial intelligence, image processing, embedded system, etc. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and the Information Processing Society of Japan (IPSJ).

Tetsushi Koide received the B.E. degree in physical electronics, the M.E. and the Ph.D. degrees in systems engineering from Hiroshima Univ. in 1990, 1992, and 1998, respectively. He was a research associate and an associate professor in the Faculty of Engineering at Hiroshima Univ. in 1992–1999 and 1999, respectively. From 1999, he was with the VLSI Design and Education Center, The Univ. of Tokyo as an associate professor. Since 2001 and 2004, he has been an associate professor in the Research Center for Nanodevices and Systems and Graduate School of Advanced Sciences of Matter, Hiroshima Univ. Since 2008, he has been an associate professor in Research Institute for Nanodevice and Bio Systems. Since 2020, he has also been an associate professor in Graduate School of Advanced Science and Engineering: Division of Advanced Science and Engineering: Quantum Matter Program. Since 2022, he has been an associate professor in the Research Center for Nanodevices (RIND), Hiroshima University, Japan. His research interests include system design and architecture issues for deeplearning application for medical engineering, real-time image processing, memory-based systems, VLSI CAD/DA, genetic algorithms, and combinatorial optimization. Dr. Koide is a member of the Institute of Electrical and Electronics Engineers, the Association for Computing Machinery, the Institute of Electronics, Information and Communication Engineers of Japan, and the Information Processing Society of Japan.

Takeshi Kumaki received his B.S. degree from the Department of Mathematics, Faculty of Science, and completed the first half of the M.E. program in Information Mathematics from the National Defense Academy, Kanagawa, Japan, in 1998 and 2003, respectively. He received the Ph.D. degree in electric engineering from Hiroshima University, Hiroshima, Japan, in 2006. From 2003 to 2004, he was with the Japan Air Self-Defense Force Electric Experimentation Group. From 2005 to 2009, he was with the Research Center for Nanodevices and Systems (RCNS) and the Research Institute for Nanodevice and Bio Systems (RNBS), Hiroshima University, Japan, where he has engaged in system design and architecture research. From 2010 to 2012, he was an assistant professor in the Department of VLSI System Design, Ritsumeikan University (RU), Shiga Japan. From 2013 to 2015, he was a lecturer in the Department of Electronic and Computer Engineering, RU. From 2016 to 2020, he was an associate professor in the Department of Electronic and Computer Engineering, RU. Since 2021, he has been a professor in the Department of Electronic and Computer Engineering, RU. His research interests include content addressable memory, SIMD processing architecture, and these applications. Dr. Kumaki is a member of the Institute of Electrical and Electronics Engineers (IEEE), the Institute of Electronics, Information and Communication Engineers of Japan (IEICE), the Research Institute of Signal Processing (RISP), and the Institute of Electrical Engineers of Japan (IEEJ).