IEICE TRANS. INF. & SYST., VOL.E103–D, NO.3 MARCH 2020

PAPER

Daisy-Chained Systolic Array and Reconfigurable Memory Space for Narrow Memory Bandwidth

Jun IWAMOTO[†a)], **Yuma KIKUTANI**[†b)], **Renyuan ZHANG**[†c)], ***Members***, ***and*** **Yasuhiko NAKASHIMA**[†d)], ***Fellow***

SUMMARY A paradigm shift toward edge computing infrastructures that prioritize small footprint and scalable/easy-to-estimate performance is increasing. In this paper, we propose the following to improve the footprint and the scalability of systolic arrays: (1) column multithreading for reducing the number of physical units and maintaining the performance even for back-to-back floating-point accumulations; (2) a cascaded peer-to-peer AXI bus for a scalable multichip structure and an intra-chip parallel local memory bus for low latency; (3) multilevel loop control in any unit for reducing the startup overhead and adaptive operation shifting for efficient reuse of local memories. We designed a systolic array with a single column × 64 row configuration with Verilog HDL, evaluated the frequency and the performance on an FPGA attached to a ZYNQ system as an AXI slave device, and evaluated the area with a TSMC 28nm library and memory generator and identified the following: (1) the execution speed of a matrix multiplication/a convolution operation/a light-field depth extraction, whose size larger than the capacity of the local memory, is 6.3× / 9.2× / 6.6× compared with a similar systolic array (EMAX); (2) the estimated speed with a 4-chip configuration is 19.6× / 16.0× / 8.5×; (3) the size of a single-chip is 8.4 mm^2 (0.31× of EMAX) and the basic performance per area is 2.4×.

key words: *systolic array, reconfigurable memory space, memory bandwidth*

1. Introduction

Computing infrastructures equipped with flexible programmability and massively parallel processing mechanisms such as GPUs have greatly contributed to the exploration of algorithms [1], [2] and the accuracy improvement [3], [4] of modern artificial intelligence. In particular, horizontally parallel processing (SIMT/SIMD architecture) is widely used for speeding up von Neumann computing [5]. For efficient SIMT/SIMD operation, the input data must be retrieved for each operation from the main memory (DDR). Therefore, in addition to guaranteeing sufficient DDR bandwidth, a coalescing mechanism must be used for aligning memory references to consecutive addresses. To hide the DDR latency over 1000 cycles, a multithreading mechanism supported by a huge register file should also be incorporated. A huge GPU power consumption seems inevitable for preserving conventional programmability.

On the other hand, the miniaturization of semiconductors is slowing down and becoming less beneficial for low power consumption and cost reduction. Further performance improvements of the von Neumann computing infrastructure face a huge obstacle. In this scenario, a GPU is used for developing an algorithm, and Domain Specific Accelerators (DSAs) are also used for implementing the algorithms on FPGA or ASIC. In fact, a huge amount of accelerators specialized for CNN [6]–[8] and systolic array architectures have been reported that efficiently execute CNN [9], [10]. Concerning these features, even if the CNN inference lowers the operation accuracy, the recognition error rate only slightly decreases [11], but the amount of weight information is greatly reduced to improve the ALU performance and the area efficiency. These are promising techniques for engineers who agree that small footprint and easy-to-estimate performances are important for social implementation regardless of the limited programmability. However, FPGA has suffered drawbacks in price and operating frequency, and ASIC has limitations in the flexibility of the target algorithm and in development cost, including a custom high speed I/O. We continue to be plagued by universal platforms that have high operating frequency, more flexibility, and simple scalability such that the performance and the cost can be tuned just by changing the number of chips.

In this paper, we propose the following: a systolic array with (1) column multithreading for reducing the number of physical units while maintaining the performance for such back-to-back floating-point accumulations as C=C+A×B; (2) a cascaded peer-to-peer AXI bus for scalable multichip structure and an intra-chip parallel local memory bus for low latency; (3) multilevel loop control in any unit for reducing the startup overhead and adaptive operation shifting for efficient reuse of local memories. As real-world examples, we demonstrated three applications: matrix multiplication, convolution operations, and light-field depth extraction. From our experiment results, the processing speed and area-efficiency are both superior to the state-of-art work EMAX. The speed is further improved by employing our proposed 4-chip configuration. The rest of this paper is organized as follows. Section 2 summarizes the problems in conventional systolic arrays and describes ideas for improvements. Section 3 shows detailed implementation by analyzing the optimal mapping of three different types of applications. Section 4 compares the performance and area between the baseline and proposed architecture. A conclusion is described in Sect. 6.

Manuscript received May 27, 2019.
Manuscript revised September 27, 2019.
Manuscript publicized December 6, 2019.

[†]The authors are with Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan.
a) E-mail: iwamoto.jun.iz0@is.naist.jp
b) E-mail: kikutani.yuma.kw3@is.naist.jp
c) E-mail: rzhang@is.naist.jp
d) E-mail: nakashim@is.naist.jp

DOI: 10.1587/transinf.2019EDP7144

2. Problems and Proposals

2.1 Problems with Conventional and Modern Systolic Arrays

In this section, we summarize the problems with conventional and modern systolic arrays. In conventional systolic arrays, ALUs are organized in a two-dimensional structure, and a set of memory is attached next to the ALU arrays. Therefore, the ALUs are re-configurable, but the memory bandwidth is not scalable. Many accelerators have improved the performance in some specific tasks [12]–[15]. Even some accelerators, which were implemented by such ALU grids as MuCCRA [16] and CMA [17], lack local memory along with each ALU. On the other hand, different real-world applications usually demand various specifications of local memory. In addition, the external memory is power-hungry in contrast to the operation and data movement in the core circuit [18]. To alleviate the unbalanced scalability, modern systolic arrays have been reported with a local memory module (LMM) attached to each unit that resembles the L1 cache [19], [20]. In GPUs, for calculating D = A×B+C, every A, B, and C should be kept in the L1 cache. Multithreading and coalescing mechanisms were introduced to hide the latency and reduce the traffic between the L1 cache and the external memory. However, unpredictable cache misses are inevitable due to the contentions of the cache lines. In the modern systolic arrays with LMM, cache miss is eliminated because each LMM keeps a single stream of input data (e.g., C) for the operation mapped on each unit. Although systolic arrays are less programmable than von Neumann computing platforms, the configurability is promising for arranging many data streams among ALUs, LMMs, and external memory in parallel. The utilization ratio of the ALUs and the traffic between external memory and LMM are comparable with state-of-the-art DSAs [21].

However, the first problem is unavoidable: wide memory bandwidth. Since the DPU [22] proposed by Wave Computing has multiple memory interfaces, it is unsuitable for edge computing. Similar to GPUs, the memory controller in such systolic arrays has to transmit the data between LMM and external memory, and so a wide memory bus is required for reducing the latency. The second problem is LMM's area redundancy. Modern algorithms such as deep convolution require many read ports, and the complicated access patterns in light-field image processing and inverse matrix calculation requires many irregular read/write and read-modify-write operations on LMM. For such requirements, multiple copies should be employed to increase the number of read ports, which sacrifices area efficiency. The third problem is the bubbles in floating-point pipelines. Such applications as inverse matrix calculation with floating-point accumulation degrade the performance of systolic arrays due to such bubbles in the pipeline. The fourth problem is the overhead related to the excessive invocation of the burst execution and DMA. Systolic arrays

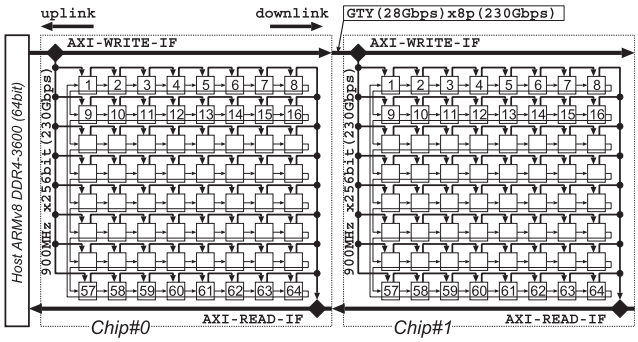

Fig. 1 Cascaded peer-to-peer AXI bus for scalable multichip systolic array

easily accelerate the innermost loops. However, the length of the burst execution depends on the length of the data. For full use of the LMM capacity and reducing the startup overhead, multilevel loops should be executed by a single invocation. Ichikura's work also employs a multilevel loop controller without combining the improvement on the footprint. For irregular applications, we must dynamically determine whether LMM's contents can be reused.

2.2 Efficient Use of Narrow Memory Bandwidth

The first proposal is a cascaded peer-to-peer AXI bus and a reconfigurable LMM space for a scalable multichip systolic array. The integration of the systolic array and the reconfigurable space is key for reducing the traffic between the main memory (DDR) and LMMs. In low-cost edge computing infrastructures, the standard I/O interface is AXI whose number is significantly limited. However, by configuring the uplink/downlink as an AXI slave/master and connecting it in a cascaded manner, all the chips can be transparently connected to the host as a single slave AXI device. Figure 1 illustrates a 2-chip configuration where each chip has 64 sets of units, including an ALU and an LMM. The bandwidth assumptions in this figure are ideal and too detailed to illustrate the system's overview. However, providing a big picture of performance here is helpful for understanding further description of our proposed ideas. In each chip, 64 execution units (1-64) are connected as a one-dimensional ring for easy programming, and the interface between the uplink and the 64 LMMs is connected as a 8×8 two-dimensional array for reducing the access latency. The AXI-WRITE-IF in each chip receives a read/write request from the uplink and distributes it to the top (bandwidth is 900 MHz × 256 bits) of eight groups of units and to the next chip (28G bps × 8 sets, which is comparable to the intra-chip bandwidth). In the same way, the AXI-READ-IF in each chip merges the read data from the bottom of the eight groups of units and the next chip. In this structure, the DMA engine attached to the host controls the data transmission between the DDR and all the LMMs mapped on a single memory space. By setting the registers (mapped on the host memory space and accessed by programmed input/output (PIO)) in each unit

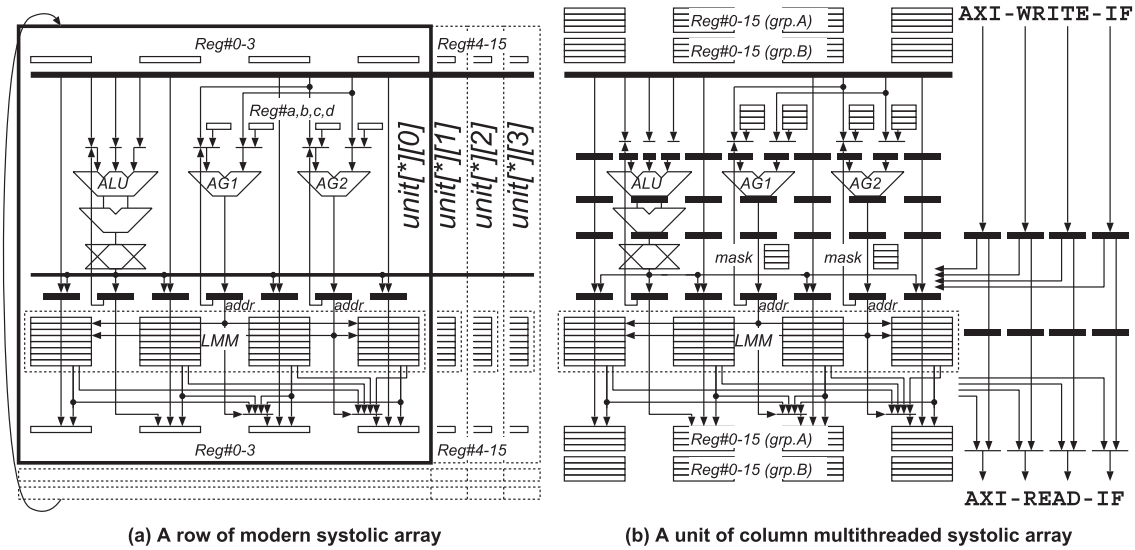

Fig. 2 Column multithreading for small footprint and bubble-free execution

that keeps a set of top and bottom addresses covered by each LMM in advance, the host can broadcast/gather data from the DDR/LMMs to the LMMs/DDR with a single DMA request.

2.3 Reducing the Redundancy of LMM and the Bubbles in Floating-Point Pipeline

The second proposal is the column multithreading of systolic arrays for alleviating the second and third problems. Although the frequencies of the pipelined floating-point units and the LMM with moderate capacity are comparable, modern systolic arrays basically cannot completely exploit pipelined floating-point units for back-to-back accumulation. Moreover, non-pipelined floating-point units are too slow compared to LMM's moderate capacity. Column multithreading can make complete use of floating-point units and the LMM without decreasing the frequency. Figure 2 (a) shows a row of a modern systolic array with a dual-port LMM [20]. Each unit has an ALU and two address generators (AG1 and AG2) in the first half and an LMM in the second half. The unit operates in a two-stage pipeline isolated by the address latches and the store registers for succeeding LMMs. The ALU output and the data read from LMM are stored in Reg#0-15 and sent to the ALU, AG1, and AG2 in the succeeding units through the 16-to-1 selectors expressed as a horizontal bold line at the top. The Reg#0-15 are also used for propagating the data to the succeeding units through pipeline registers. If an application requires eight discrete reads every cycle, four dual-port LMMs in the same row (unit[*][0-3] in the figure) keep the same data and simultaneously update the eight registers at the bottom. For executing back-to-back floating-point accumulations, the floating-point unit cannot be pipelined or have many bubbles in the floating-point pipeline. Figure 2 (b) illustrates the proposed structure of a unit with four-column multithreading that provides the same function as Fig. 2 (a) with four cycles. The first pipeline stage selects the ALU, AG1, and AG2 input from Reg#0-15, and the remaining three stages perform the calculation. The total number of Reg#a-d is the same, but double buffering of Reg#0-15 is required to keep the results that were produced by the preceding unit every four cycles (grp.A and grp.B). Consequently, back-to-back floating-point accumulations with no bubbles become available. The latency of the unit in (b) becomes eight cycles, and eight ports are provided with 4-cycle operation of dual-port LMMs. Partitioning should be provided to manage a maximum of four different spaces, if four logical units require different address ranges.

2.4 Reducing the Excessive Invocation of Burst Execution and DMA

In general, before starting a burst execution, we should do the following: map the operations, initialize the registers (Fig. 2 (a)), and fill the LMMs. The third proposal is multilevel loop control and adaptive operation shifting for reducing such overhead.

The first key idea is how to represent multilevel loops in C code and mapping on multithreaded units with less hardware resource. Figure 3 (a) shows the beginning of a sample C code for our proposed systolic array, and Fig. 3 (b) shows the mapping of multilevel loop control on four logical columns where four operations are simultaneously executed in one cycle. The 1st line indicates the number of chips (N) to be activated. Each line from the 2nd to the 5th is mapped on each logical column. Loops 1 and 0 are the outer and innermost loop counters. Loop 0 is initialized to M (e.g., the number of columns in a matrix) and decremented every cycle. Loop 1 is initialized to GRP (e.g., the number of rows processed in a burst execution) and is not changed while loop 0 is not zero. When loop 0 reaches zero, the operand of "add loop 1" is switched from 0 to −1, and loop 1 is decremented. The magic description "init0?col:col" in the 4th

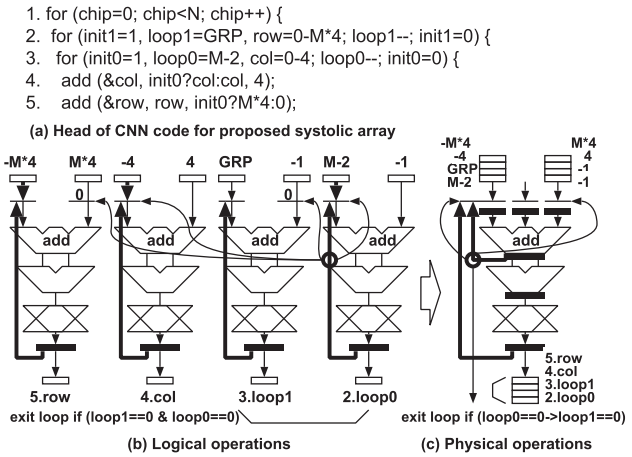

Fig. 3 Multilevel loop control

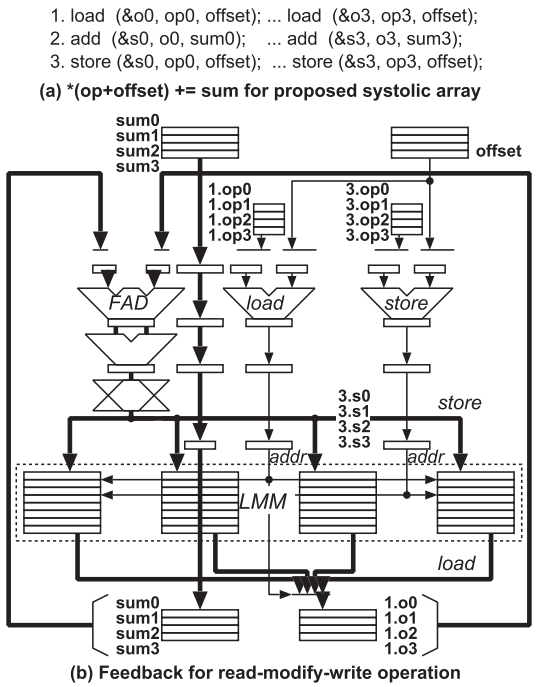

Fig. 4 Read-modify-write operation

line has no effect as a C statement, but it specifies reloading the initial value (−4) to "col" when loop 0 is restarted with "init0=1". Otherwise, "col" is incremented by 4. In the same manner, "row" is initialized to - M×4 and incremented by M*4 (number of bytes) when "init0=1". When both loop 0 and 1 become zero, the stop signal is asserted in the unit and propagates to the successors. Figure 3 (c) illustrates the mapping on a multithreaded unit. Four operations were sequentially executed with four cycles. The self-loop of the first stage in the pipeline can manage the inter-column data dependence.

Multilevel loop control works well with LMM's read-modify-write where multilevel accumulation is mapped. By accumulating the results of each innermost loop into LMM across the outer loop, LMM's spilling out/in can be eliminated. Figure 4 illustrates the mapping of the read-modify-write function to a multithreaded unit. With column multithreading, at most four sets of read-modify-write can be mapped on a physical unit. The partial sums (sums 0-3) provided by the preceding stages are first sent to the register at the bottom of the unit and returned to the ALU input to adjust the timing with the results (o0-3) of the load operations. After accumulation in the ALU, the results (s0-3) are stored in the same LMM.

The second key idea is a software technique. For regular applications, the fixed pattern of the DMAs generated by a compiler can reuse the LMMs just-in-time. However, if the fixed pattern does not follow the actual transition of the memory access, excessive DMAs are invoked. If the host checks whether the neighbor LMMs have valid data and slides the mapped operations among units, such extra DMAs can be eliminated.

3. Exploration for Mapping Three Applications

In this section, we choose a matrix multiplication (MM) for full-connect layers in DNN, a convolution operation (CNN) for feature extraction, and light-field depth extraction (LF) for background elimination as representative edge-side applications with various memory access patterns. The algorithms and mapping schemes on the proposed systolic array are explored to find the best multichip employment where the LMMs can share addresses so that the external memory can both broadcast and reuse as much data as possible.

3.1 MM: Matrix Multiplication

Figure 5 (a) shows the MM data structure and (b) shows a simple implementation in C language. M, N, H, W, and GRP are respectively the size of the matrix, the number of chips to be activated, the height of the systolic array (H≤M), the logical width of the systolic array (W=4 ≤M), and the number of rows in a group processed in a single burst execution. For efficient use of the H×W fused multiply-and-add ALUs, operations A[row][*]×B[*][col] corresponding to output C[row][col] should be divided by H, mapped on sequential locations in a column [* mod H][col mod W], and executed in a pipelined manner. Figure 5 (c) respectively shows a snapshot of the systolic array when A00×B0c, A01×B18, A02×B24, and A03×B30 are performed simultaneously in the left most column for partial C00, C04, C08, and C0c. With H-stages, partial sums of C00, C04, .., C0((H-1)×W) are produced every cycle and accumulated into a final LMM. Consequently, by employing W columns and repeating the burst execution M/H/GRP times, we obtain complete C00, C01, .., C0(M-1).

In Fig. 5 (d), the corresponding implementation for the N-chips is shown. A quintuple nested loop in the rectangle corresponds to a burst execution with N-chips. Each iteration of the chip-loop corresponds to the matrix multiplication of the GRP rows of array A and the entire B. Array A is divided into N-chips, and each LMM keeps M×GRP

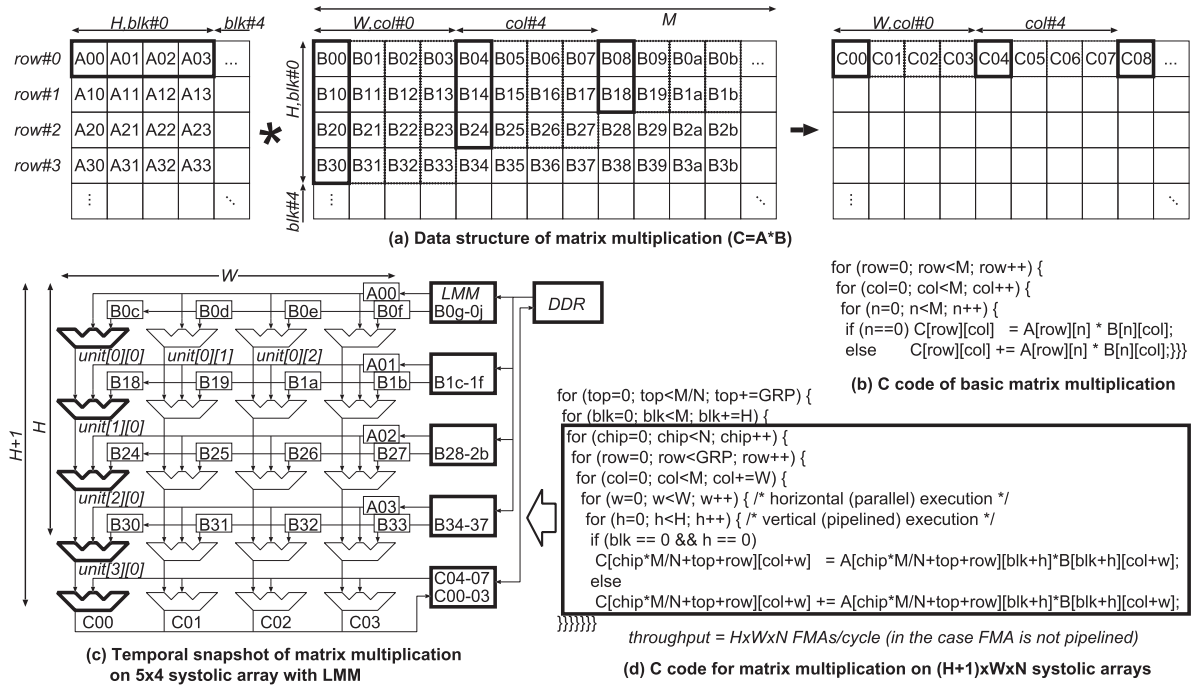

Fig. 5 Matrix multiplication on systolic array

elements of A and a row of B. While the outer blk-loop increases from 0 to M-1, the LMMs in each chip hold the GRP rows of array A, and the last stages hold the GRP rows of C×0, C×1, .., C×(M-1). The H rows of array B can be sent to the N-chips by broadcasting when the blk-loop moves to the next set of H rows. Theoretically, the above calculations require $M^3/(H\times W\times N)$ cycles, excluding the delay for the startup (H+1) cycles and the overhead for the LMM replacement. In addition, the theoretical amount of DDR-LMM transmission with enough LMM capacity where A, B, and C are kept is $M^2\times 3$. The amount of transmission with a moderate capacity of LMM where the M×GRP of A and C, and the M×H of B are kept is optimally $M^2\times 2$ for A and C, and $M^2\times M/GRP$ for B (M^2 when M=GRP).

3.2 CNN: Dense Stencil Computation

Figure 6 (a) shows the memory access pattern of CNN. M, K, IC, OC, N, W, and GRP are respectively the size of the matrix (each input and output channel), the kernel size (e.g., 3), the number of input channels (e.g., the number of input images), the number of output channels, the number of chips, the logical width of systolic array (W=4), and the number of rows in a group processed in single burst execution. The total number of words in "in", kernels, and "out" are $M^2\times IC$, $K^2\times IC\times OC$, and $M^2\times OC$. In general, when is K used for CNN, it is an odd number, but hardware parameter W is an even number. For the efficient use of hardware, K×K-chains of the sum-of-products should be mapped on unit[*][oc mod W] in a systolic array column. Furthermore, by employing W columns, $K^2\times W$ units become active. Figure 6 (b) illustrates a data flow graph (not a snapshot) of a systolic array employing $(K^2+1)\times W$ units (W≤OC). CNN calculates the sum-of-products between a K×K kernel and a part of "in", accumulates all the input channels, and stores one result in "out". In this case, the K-LMMs should keep the same row of "in". However, the host can broadcast the data into the K-LMMs by a single DMA. Figure 6 (c) shows the temporal overview of memory access, where "t" shows the direction of the time. The arrows from the input layer to the output layer indicate the direction of the convolution operation.

3.3 LF: Sparse Stencil Computation

Figure 7 (a) shows the data structure of LF. IM, OM, R, N, and W are respectively the size of the input image, the size of the output image, the distance of the micro images, the number of chips, and the logical width of systolic array (W=4). The input image has 7500×7500 pixels in which the micro images (75×75 pixels) are located in a 100×100 2D-mesh form. The output image has 1600×1600 pixels generated by finding a minimum sum of absolute difference (SAD) of discrete stencil patterns while changing the distance (75+depth pixels) among the six 3×3 micro images. The SAD is obtained from the sum of the 36 differences among the six pixels in the center (p0) and the surrounding six discrete locations (p1). Figure 7 (b) illustrates a data flow graph of the systolic array employing 12×4 units. Each unit has a SAD function in ALU. The top line of 3×3 pixels (i#-1) at the y#0 position is mapped on the first stage (unit[0][*]), and six discrete pixels including p0 and p1 are loaded from the top LMM that holds a line of the input image (emphasized by dotted rectangles). The second stage

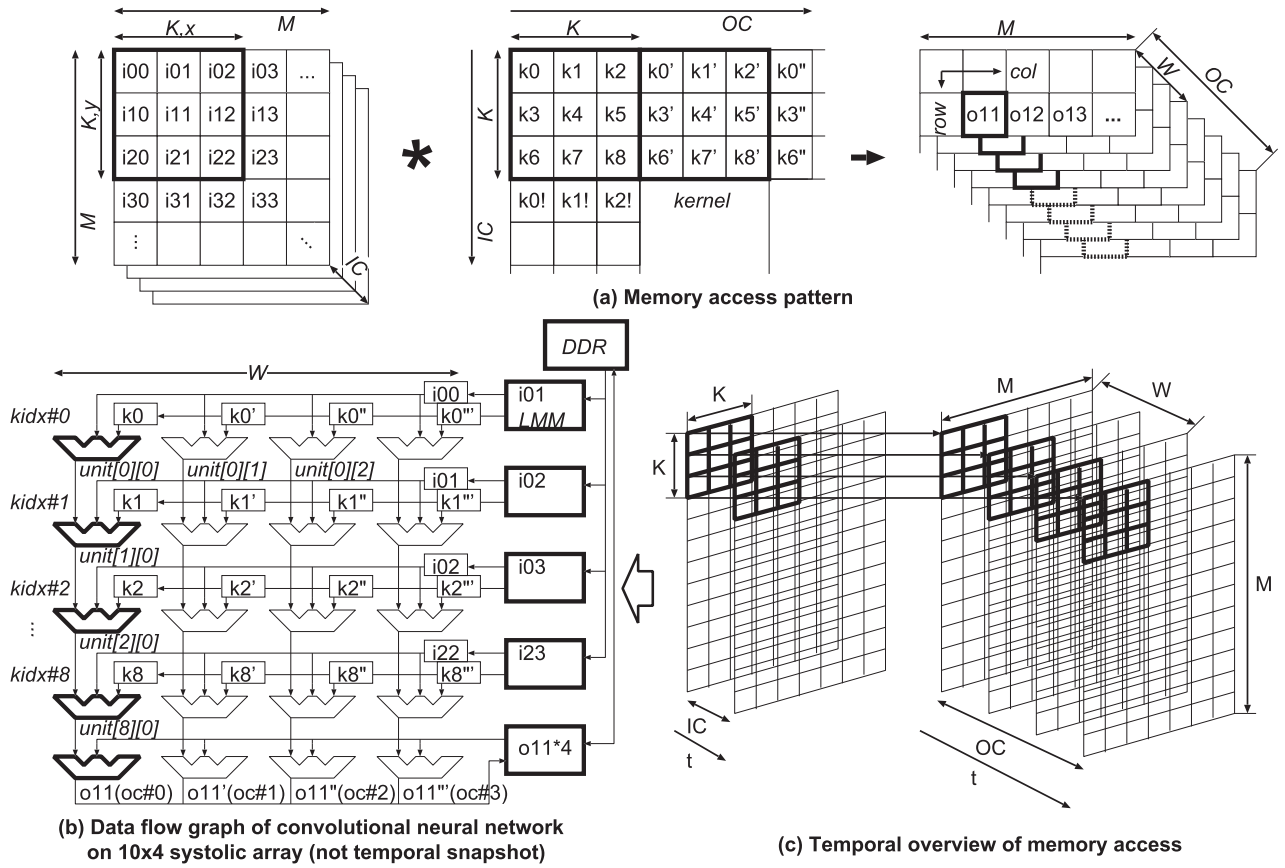

Fig. 6 CNN on systolic array

(unit[1][*]) and the third stage (unit[2][*]) are loaded from the y#-1 and y#1 positions (p1) and compared with the p0 obtained in the first stage. After passing through nine stages in the systolic array, four SADs from four columns are accumulated into a SAD through two additional stages. In an outermost loop omitted in Fig. 7 (c), "depth" is incremented. The minimum value of SAD and the corresponding "depth" for each output pixel are stored and updated with conditional storing when a smaller SAD is obtained with an updated "depth". In this way, 11 stages are used to obtain the SADs of 36 pixels, and 40 out of 44 units become active. The total number of stages is 18, including 12 in Fig. 7 (b) and the preceding six stages for the address calculation of "c". Figure 7 (c) shows the temporal overview of memory access, where "t" shows the direction of the time. The depth increases with time, the SAD of each window is calculated, and a pair of the smallest SAD and corresponding "depth" is stored as final result.

In the LF, we observed another opportunity for reducing the DDR-LMM transmission. The input image stored in the 1-4-7th, the 2-5-8th, and the 3-6-9th LMMs are adjacent (Fig. 7 (b)). When the base position (p0) moves to the next line, p0 and p1 can reuse the next lines stored in the lower LMMs and only new lines can be provided from DDR to LMM. By shifting the ALU operations downward, the DDR-LMM transmission can be minimized. However, unlike MM and CNN, the reuse interval is irregular. The above adaptive operation shifting is helpful to change the shift amount based on the address calculation results.

4. Evaluation Setup

4.1 Setup of LMM and Application Characteristics

Table 1 summarizes the MM, CNN, and LF mapping. From the requirements shown in (a), (b), (c), and (d), we obtained the minimum LMM configurations. MM requires the input of (1+W) words from 2-ports, and Fig. 8 (a) has two sets of single port LMMs and two address generators for reading a word from A and four contiguous words from B. In the same way, CNN requires the input of (1+W) words from (1+W) ports, and Fig. 8 (b) has a single port LMM for reading a word from "in", and two dual port LMMs for reading four discrete kernel values. Furthermore, LF requires 6-words from 6-ports, and Fig. 8 (c) has three dual port LMMs for reading six discrete pixels from "in". Consequently, a symmetric configuration with a 1R+1RW dual port LMM that provides four consecutive reads or two discrete reads shown in Fig. 8 (d) seems adequate for covering all three applications.

For further discussion of performance, Table 1 (e)~(i) summarizes the number of operations and the amount of

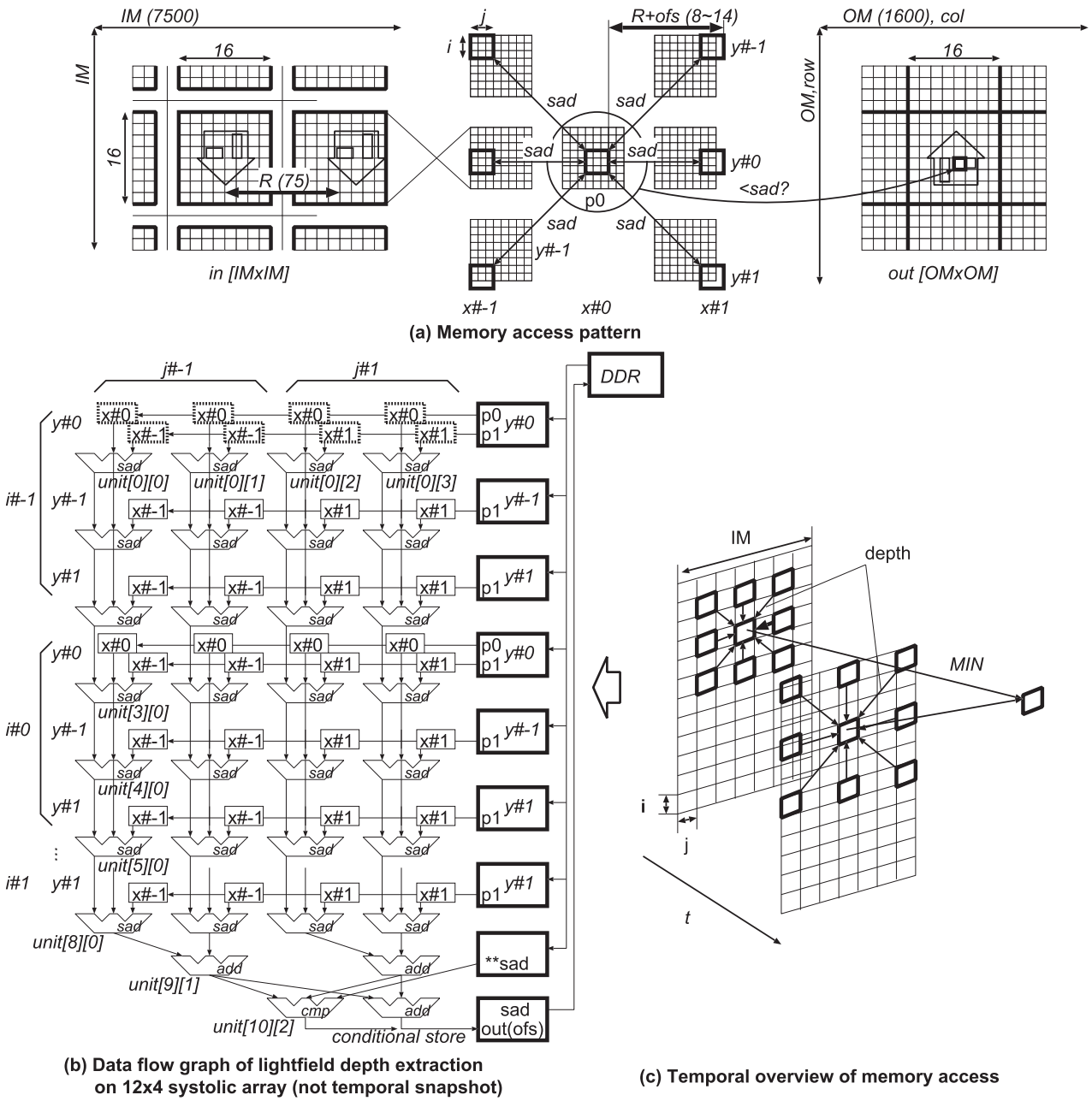

Fig. 7 Light-field depth extraction on systolic array

data that are transmitted. The parameters of proposed hardware are number of logical columns, number of stages, and capacity of each LMM. For hiding the latency of floating-point accumulation, four is optimum as the number of logical columns. The number of stages are limited by the capacity of FPGA, and the capacity of each LMM is limited by the operating frequency of ALU. As described later, the actual capacity of LMM is 32 KB. H and W of each application are automatically decided by the number of stages and logical columns, and other parameters are decided based on the capacity of LMM. Basically, evaluation with small data that can fit in the LMM is unfair because overestimation is useless for the real applications. Therefore, massive data should be used to evaluate the performance with sufficient traffic between LMM and DDR. However, extremely massive data that has no opportunity for systematic reuse of LMM makes the effect of proposed ideas invisible.

In MM, the specific parameters are M and GRP. If M is larger than 32 KB/4 bytes (8 K words), multilevel loop control is useless due to enough length of burst execution, and lack of LMM for keeping multiple rows of the matrix. For emphasizing the effect of multilevel loop control (M $\times$ GRP elements are processed at once), the number of elements in a rows of A and B are set to 480 where 480 elements $\times$ 4 bytes $\times$ (GRP=8) of A and B fit to 32 KB.

In CNN, the specific parameters are M, GRP, K, IC,

Table 1 Summary of mapping of three applications

	MM	CNN	LF
Size of systolic array	(H+1)×W	K^2 ×REP×W	(14×REP+3)×W
(a) max read words/cycle from LMM	1 for A W (consecutive)	1 for in W (discrete) for k	6 (discrete) for in
(b) max read ports/LMM	1+1 (consecutive)	1+W (discrete)	6 (discrete)
(c) max write words/cycle to LMM	W (consecutive)	W (discrete) for out	1 for sad 1 for out
(d) max write ports/LMM	1 (consecutive)	W (discrete)	1+1
(e) total FMA/SAD/etc. ops	M^3	K^2M^2 ×IC×OC	40×OM^2
(f) algorithmic DDR read words	A:M^2 B:M^2	in:M^2 ×IC ker:K^2 ×IC×OC	in:9×OM^2 sad:OM^2
(g) ideal DDR read words	A:same B:M^2 ×M/GRP	in,ker:same out:M^2 ×OC×IC/REP	in:<9×OM^2 by reuse sad,out:OM^2 ×2
(h) algorithmic DDR write words	C:M^2	out:M^2 ×OC	sad,out:OM^2 ×2
(i) ideal DDR write words	C:same	out:M^2 ×OC×IC/REP	sad,out:same

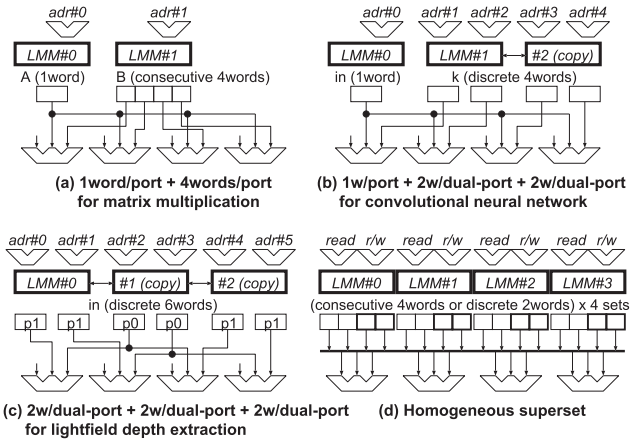

Fig. 8 LMM setup

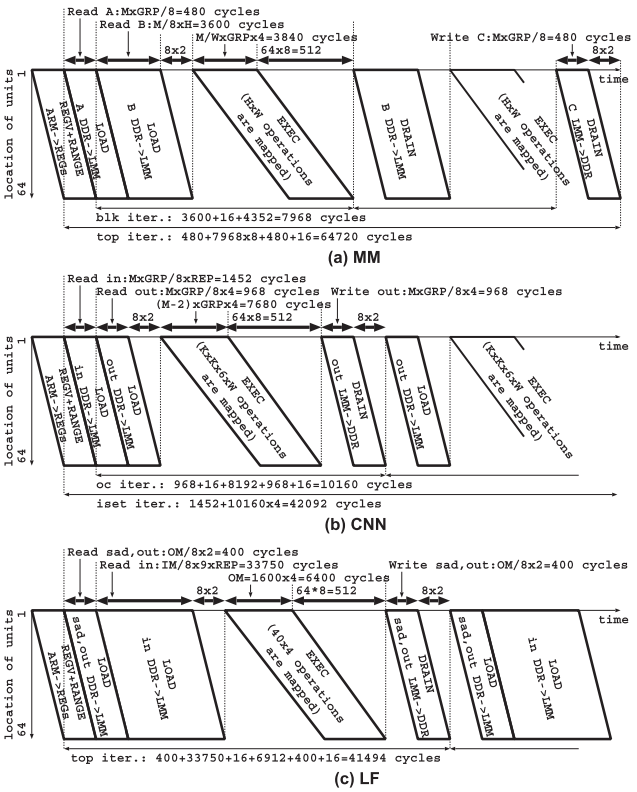

Fig. 9 Timing diagram of top-level iteration

OC and REP. M and GRP are set to 242 and 8 for the same reason as MM. M=242 is similar size used in the first layer of VGG16, and K=3 is the typical size of kernel in VGG16. The number of input channel (IC) is set to 18 so that the amount of input data exceeds the capacity of total LMM of 64 stages (REP=6 sets of input channel fit to 64 stages), and sufficient traffic between LMM and DDR is observed. The number of output channel (OC) is set to 16 in the same way so that the amount of output data exceeds the capacity of last LMM (4 sets of output channel fit to the last stage). The execution time is propotional to IC and OC.

Whereas, in LF, the size of input image (IM×IM) and output image (OM×OM) are large enough to keep burst execution. The number of stages for calculating one output pixel is around 16. Consequently, REP=4 sets of operations can be mapped on 64 stages so that the output image is divided into four areas and four pixels are produced in every cycle.

When M=480, GRP=8, H=60, W=4 for MM, M=242, GRP=8, K=3, IC=18, OC=16, W=4, REP=6 for CNN, and IM=7500, OM=1600, W=4, and REP=4 for LF, the total number of operations shown in Table 1 (e) are 111 Mops, 149 Mops, and 102 Mops, and the algorithmic total number of input words read from DDR in (f) are 461 K words, 1.06 M words, and 25.6 M words. Consequently, the computation strength based on DDR calculated by (e)/(f) varies from 241 to 3.98. However, the ideal baseline of the input words read from DDR due to spilling out/in of LMM are 14.1 M words, 3.87 M words and around 12.8 M words, as calculated from (g). The variation of strengths ((e)/(g)=7.87, 38.5 and 7.97) becomes smaller. In the same way, (h) and (i) are the algorithmic/ideal baselines of the output words to DDR. The ideal peak is helpful to calculate the efficiency later.

4.2 Calculating the Ideal Peak Performance

Figure 9 shows the ideal timing charts based on the above

Table 2 Summary of ideal peak performance

	MM	CNN	LF
Total number of cycles	3.88 M	3.79 M	16.6 M
FMA/SAD hardware peak	64-FMA/SADs /cycle		
Total number of ops	111 Mop	149 Mops	102Mops
Utilization ratio of ALUs	/64/3.88 M =44%	/64/3.79 M =61%	/64/16.6 M =9.6%
DMA hardware peak	8-words /cycle		
Ideal DDR read words	14.1 M words	3.87 M words	12.8 M words
Ideal DDR write words	230 K words	2.81 M words	5.12 M words
Utilization ratio of DMA	/8/3.88 M	/8/3.79 M	/8/16.6 M
	=46%	=22%	=13%

discussion (horizontal axis is time, and vertical axis is location of units) when single chip is employed. Before starting the burst execution, the mapping of operations, the initialization of the registers, and filling the LMMs should be done. However, while the same burst execution is repeated, the mapped operations are reused and the overhead for mapping (denoted as CONF below) is ignored.

In (a) MM, the number of cycles for a burst execution observed at the first unit is 3840 (960×4 based on column multithreading), and the number of cycles for completion is 4352 (3840 + 64×8 cycles for passing through 64 units). The number of cycles for one blk-loop is 7968 (4352 + 3600 for DDR-LMM transmission + 8×2 for passing through a lane with eight units). The cycles for a top-loop iteration are 64720 (7968×M/H + 480 for loading A + 496 for draining C) excluding the cycles for the register initialization (REGV) and the address range initialization (RANGE) performed by PIO. Consequently, the total number of cycles for the whole MM is 3.88M (64720×M/GRP) when N=1.

In (b) CNN, the number of cycles for a burst execution at the first unit is 7680 (1920×4 based on column multithreading), and the number of cycles for completion is 8192 (7680 + 64×8 cycles for passing through 64 units). Furthermore, the number of cycles for one oc-loop is 10160 (8192 + "968+16" for loading out + "968+16" for draining out). The cycles for an iset-loop iteration are 42092 (10160×OC/W + 1452 for loading "in") excluding REGV and RANGE. Consequently, the total number of cycles for the whole CNN is 3.79 M (42092×(M-2)/GRP×IC/REP) when N = 1.

In (c) LF, the number of cycles for a burst execution at the first unit is 6400 (1600×4 based on column multithreading), and the number of cycles for completion is 6912 (6400 + 64×8 cycles for passing through 64 units). The number of cycles for a top-loop iteration is 41494 (6912 + "400+400+16" for loading and draining sad and out + "33750+16" for loading "in"). Consequently, the total number of cycles for the whole LF is 16.6 M (41494×OM/REP) when N=1.

Table 2 summarizes the ideal peak performance calculated by combining the cycles and the discussion in Sect. 4.1. The actual performances are compared with these peak performances below.

4.3 Metrics and Comparison

The metrics are the operating frequency, the MM, CNN, and LF performances, and the chip's area. We mainly compared the baseline (original EMAX) [20] and the proposed systolic arrays with some combinations of ideas and assumptions.

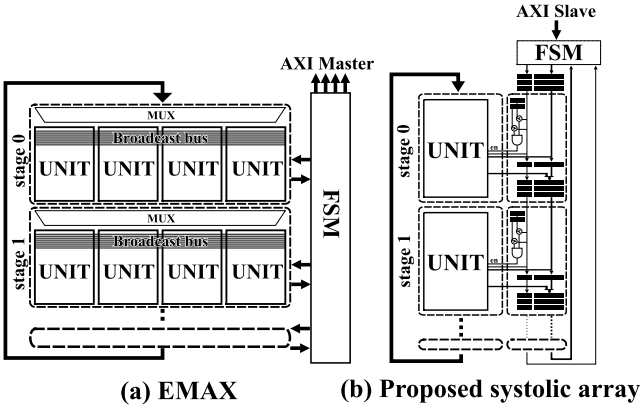

Fig. 10 Structural difference between EMAX and proposed systolic array

Figure 10 (a) shows the outline of EMAX that operates as an AXI master. Each stage has four physical units. Each unit has a single-precision floating-point ALU and two address generators for load/store operations on dual port memory (LMM). Data produced by the previous stage is stored in registers and provided to the next stage through a crossbar switch (MUX). The DMA controller in the finite state machine (FSM) is connected to all stages and manages the transmission between DDR and LMM. Also, FSM can broadcast data to LMMs in the same stage while the calculations are performed. In contrast to EMAX, the proposed systolic array shown in Fig. 10 (b) is designed as an AXI slave for making it easy to increase the number of chips without increasing the number of AXI bus. The FSM is connected only to the top and the bottom of a ring structure, and the data is sent by two cycles per stage pipelining. The combination of AXI slave mode and the address comparator in each stage enables broadcasting to many LMMs across multiple stages and chips with less amount of wires than EMAX. Furthermore, the proposed systolic array improves area efficiency by column multithreading. Each stage has single physical unit that manages the same function as provided by four physical units in a stage of EMAX. The four LMMs are integrated in a LMM by interleaving four sets of memory access. The number of stages (64), the width of registers (64bits), and the supported ALU functions in EMAX used for further comparison are the same as the proposed systolic array.

The host is implemented on Xilinx Zynq UltraScale+ ZCU102, and the systolic arrays designed by Verilog HDL are implemented on a S2C Virtex UltraScale Single VU440 Prodigy Logic Module. Single ARMv8 in Cortex-A53/ 1.2 GHz with DDR4-136 Gbps (host of systolic arrays) and NVIDIA Jetson TX2 (256 cores)/1.3 GHz with DDR4-480 Gbps (tuned with CUDA Version 8.0.84) were also evaluated as references. The target speed of the proposed systolic array on FPGA is 150 MHz, and the comparable

link speed between the host and the systolic arrays is bi-directional 38.4 Gbps (12.8 Gbps-GTH×3). Both are 1/6 of the ideal model shown in Fig. 1. However, the link speed in the actual system is 15 Gbps (5.0 Gbps-GTH×3). The performance with 38.4 Gbps was estimated based on the ratio between the measured speed and 15G bps. For the area, we used a Synopsis Design Compiler (M-2016.12-SP2), a TSMC 28-nm library, and a memory generator. The target speed was 900 MHz (Fig. 1).

4.4 Environment for Generating Code

The applications for the proposed systolic arrays can be written by C code (Fig. 3). Such function calls as load(), exe(), and store() can be compiled by normal C compilers for CPUs, and general tools can be used for debugging the algorithms. After the algorithms are fixed, a special translator gets such information as the ALU operation, the data dependency, and the range of the addresses covered by each LMM from the parameters provided in the function calls and generates another C code including operation mapping (CONF), register initialization (REGV and RANGE), DMA invocation (LOAD and DRAIN), and the trigger of the burst execution (EXEC). Finally, the translated code is compiled by normal C compilers and linked with a library including an interface to the systolic array.

5. Results

The operating frequency of EMAX was 50 MHz and the proposed array with column multithreading and pipelining worked with 150 MHz on the same FPGA. Table 3 shows the resource usage in FPGA where the systolic array is implemented. ZCU102 is the common host just for providing AXI interface. The routing cannot be finished for EMAX due to many wires. The number of block RAM (BRAM) is reduced to 1/4 of EMAX. Figure 11 shows the performance of three programs normalized by the execution time on EMAX. Proposed-1 is a simple column multithreading model, and proposed-2 employs an intra-chip 8×8 parallel local memory bus. Proposed-3 has multilevel loop control and adaptive operation shifting. Proposed-4 is a 4-chip configuration with all the ideas. Finally, proposed-5 and proposed-6 are single-chip and 4-chip configurations with 38.4 Gbps assumption. We estimated the 4-chip performance from the time measured by executing the programs compiled for 4-chip configuration on the single-chip prototype. The 4-chip scalability varies from 3.1 × to 1.3 ×. We conclude that simply changing the performance by adjusting the number of chips is important.

Figure 12 shows the breakdown of the execution time. EMAX is not included because it is an AXI master device and no accurate breakdown can be observed from the host. "ARM," which is the miscellaneous time in ARMv8 for preparing burst executions, is negligible. "CONF" is the time for mapping operations by PIO, and it is also negligible, as mentioned in Sect. 4.2. The other breakdowns were

Table 3 Resource usage in FPGA

Site type	ZCU102 host	EMAX (VU440) 64 stages	Proposed-3 (VU440) 64 stages
LUT	6134 (2.24%)	2421366 (95.59%)	1575371 (62.19%)
LUTRAM	178 (0.12%)	416 (0.09%)	1110 (0.24%)
FF	9481 (1.73%)	298680 (5.91%)	512549 (10.12%)
BRAM	10 (1.10%)	2084 (82.70%)	539 (21.39%)
DSP	0 (0.0%)	2048 (71.11%)	256 (8.89%)

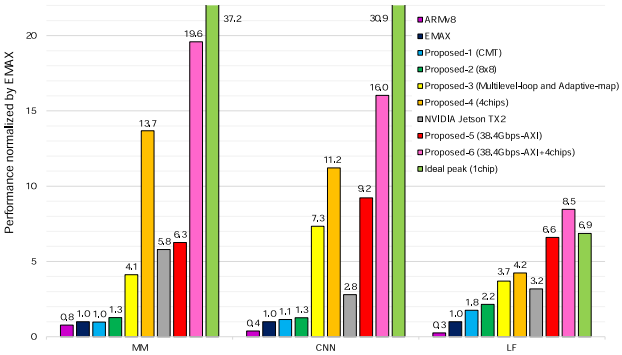

Fig. 11 Performance comparison normalized by EMAX

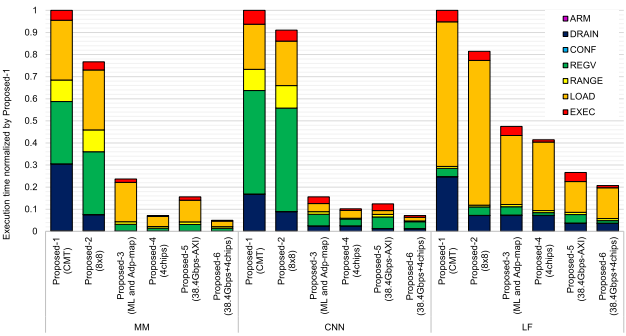

Fig. 12 Breakdown of execution time

confirmed with the ideal timing charts shown in Fig. 9. The result shows that our proposed-2 (intra-chip parallel local memory bus) for reducing the delay time related to AXI-READ-IF successfully reduced the "DRAIN" time. The contributions of proposed-3 and proposed-4 are significant on all three applications. Additionally, assuming 38.4 Gbps, proposed-5 and proposed-6 successfully reduced the time for "LOAD" as expected. Single-chip and 4-chips eventually improved the 6.3 ×/9.2 ×/ 6.6 ×, and 19.6 ×/16.0 ×/8.5 × performance more than EMAX. As for the relationship between the acutual performance of proposed-5 (single chip) and the ideal peak (single chip), large diverence is observed. In MM, actual "LOAD" consumes most of the execution time due to huge overhead of PIO for reading/writing control registers of DMA. In CNN, actual "REGV" consumes most of the execution time due to huge overhead of PIO for writing registers in systolic array through AXI bus. Meanwhile, in LF, proposed-5 is almost the same as the ideal peak irrespective of huge "LOAD". This is because the effect of the adaptive shift is dynamically determined and is

Table 4 Result of hardware synthesis

	EMAX	Proposed-3 N=1
Clock constraints	300 MHz	900 MHz
Area[μm^2]		
Combinational	4,074,460	1,705,263
Combinational (buf/inv)	173,855	278,722
Non-combinational (registers)	773,047	826,694
Memory (8 KB LMM/unit)	7,826,136	1,956,534
Memory (32 KB LMM/unit)	22,374,142	5,593,536
Total cell area (8KB LMM)	12,847,498	4,767,213
(Ratio)	(1.00)	(0.37)
Total cell area (32KB LMM)	27,395,504	8,404,215
(Ratio)	(1.00)	(0.31)

not considered in the ideal peak. The overhead for "LOAD" and the effect of the adaptive shift in proposed-5 are comparable as a result.

Table 4 shows the estimation of the chip area based on the synthesis result with the TSMC 28-nm library and the memory generator. The EMAX frequency and area were 300 MHz, 12.8 mm^2 with 8KB-LMM, and 27.4 mm^2 with 32KB-LMM. 32KB-LMM is the upper limit of the capacity that does not affect the operating frequency of the systolic array. 8KB-LMM is also evaluated to explore the sensitively of the capacity of LMM on entire area. We found if we need to reduce the size of the chip to half of the 32KB-LMM version, the capacity of LMM should be reduced to 8KB. The area of the proposed array was estimated with a 900 MHz constraint. The breakdown of the area shows that the combinational logic was 0.42 ×, the register was 1.07 ×, the LMM was 0.25 ×, and the total area with 32KB-LMM was 8.4 mm^2 (0.31 × of EMAX). Consequently, the basic performance per area was 2.4 × (900 MHz/300 MHz×4/0.31). The chip size of the Jetson TX2 (GP10B) is not disclosed and TX2 include ARM core. Therefore, the performance per area of TX2 is estimated from the ratio of the number of cores between GP10B (256 cores) and GP100 (3584 cores, 610 mm^2). The ratio of the area between the proposed-5 (28 nm) and TX2 (16 nm) is 1:5.2. Consequently, as compared with TX2, the performances per area of proposed-5 are 5.6 ×, 17 ×, and 11 × for MM, CNN, and LF. Note that an obvious trade-off exists between the hardware resource and the speed on the clock frequency constraints. There is potential to achieve even better performances per area by increasing the frequency.

6. Conclusion

We proposed the following to improve the footprint and the scalability of systolic arrays: (1) column multithreading; (2) cascaded peer-to-peer AXI and intra-chip parallel local memory buses; (3) multilevel loop control and adaptive operation shifting. We identified the following: (1) the execution speed of a matrix multiplication/a convolution operation/a light-field depth extraction is 6.3×/ 9.2×/ 6.6×, compared with a similar systolic array (EMAX); (2) the estimated speed with 4-chip configuration is 19.6×/ 16.0 ×/8.5 ×; (3) the single-chip is 8.4 mm^2 (0.31 × of EMAX) and the basic performance per area is 2.4 ×.

References

[1] K. Simonyan, et al., "Very deep convolutional networks for large-scale image recognition," CoRR, abs/1409.1556, 2014.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," CVPR2015, 2015.

[3] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," NIPS2012, vol.60, no.6, pp.84–90, 2017.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," CVPR2016, pp.770–778, 2016.

[5] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick, "Stencil Computation Optimization and auto-tuning on State-of-the-Art Multicore Architectures," IEEE Conference on Supercomputing, SC2008, 2008.

[6] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "14.5 Eyeriss: An Energy-efficient Recongurable Accelerator for Deep Convolutional Neural Networks," pp.262–263, ISSCC2016, 2016.

[7] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks," pp.553–564, HPCA2017, 2017.

[8] S. Han, et al., "EIE: Efficient inference engine on compressed deep neural network," ISCA2016, 2016.

[9] T. Nowatzki, V. Gangadhar, K. Sankaralingam, and G. Wright, "Domain Specialization Is Generally Unnecessary for Accelerators," IEEE MICRO, vol.37, no.3, pp.40–50, May/June 2017.

[10] N.P. Jouppi, et al., "In-datacenter performance analysis of a tensor processing unit," ISCA2017, 2017.

[11] Suyog Gupta, et al., "Deep learning with limited numerical precision," ICML2015, 2015.

[12] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," Field Programmable Logic and Application, FPL2003, vol.2778, pp.61–70, 2003.

[13] F. Bouwens, M. Berekovic, B. De Sutter, and G. Gaydadjiev, "Architecture Enhancements for the ADRES Coarse-Grained Reconfigurable Array," HiPEAC2008, vol.4917, pp.66–81, 2008.

[14] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S.W. Keckler, and C. Moore, "Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture," ISCA2003, vol.23, no.6, pp.46–51, 2003.

[15] Y. Kim, et al., "Power-conscious configuration cache structure and core mapping for coarse-grained reconfigurable architecture," NBiS2012, pp.826–831, 2012.

[16] T. Katagiri, K. Hironaka, and H. Amano, "Extension of Memory Controller equipped with MuCCRA-3-DP: Dynamically Reconfigurable Processor Array," NBiS2012, pp.826–831, 2012.

[17] M. Izawa, N. Ozaki, Y. Koizumi, R. Uno, and H. Amano, "A Co-Processor Design of an Energy Efficient Reconfigurable Accelerator CMA," International Journal of Networking and Computing, vol.5, no.1, pp.239–251, 2015.

[18] G. Kestor, R. Gioiosa, D.J. Kerbyson, and A. Hoisie, "Quantifying the Energy Cost of Data Movement in Scientific Applications," IISWC2013, pp.56–65, 2013.

[19] M. Tanomoto, S. Takamaeda-Yamazaki, J. Yao, and Y. Nakashima, "A CGRA-based Approach for Accelerating Convolutional Neural Networks," MCSoC2015, pp.73–80, 2015.

[20] Y. Yuttakonkit and Y. Nakashima, "Performance Comparison of CGRA and Mobile GPU for Light-field Image Processing," CANDAR2016, pp.174–180, 2016.

[21] T. Ichikura, R. Yamano, Y. Kikutani, R. Zhang, and Y. Nakashima, "EMAXVR: A Programmable Accelerator Employing Near ALU

Utilization to DSA," IEEE Symposium on Low-Power and High-Speed Chips, Coolchips2018, 2018.
[22] S. Chaudhuri and A. Hetzel, "SAT-based Compilation to a Non-Von-Neumann Processor," ICCAD2017, pp.675–682, 2017.

Jun Iwamoto received his B.E. degree from Osaka Prefecture University College of Technology in 2018. He has been an master student in Nara Institute of Science and Technology now. His research interests include computer architecture. He is a student member of IEICE.

Yuma Kikutani received his B.E. and M.E. degrees from Osaka Prefecture University College of Technology in 2017, and from Nara Institute of Science and Technology in 2019. He is an engineer in Socionext Inc. since 2019. His research interests include computer architecture.

Renyuan Zhang received his M.E. and Ph.D. degrees from Waseda University and the University of Tokyo in 2008 and 2013, respectively. He has been an Assistant Professor with Nara Institute of Science and Technology since 2017. His research interests include analog-digital-mixed circuits and approximate computing. He is a member of IEEE and IEICE.

Yasuhiko Nakashima received B.E., M.E., and Ph.D. degrees in Computer Engineering from Kyoto University in 1986, 1988 and 1998, respectively. He was a computer architect in the Computer and System Architecture Department, FUJITSU Limited from 1988 to 1999. From 1999 to 2005, he was an associate professor in the Graduate School of Economics, Kyoto University. Since 2006, he has been a professor in the Graduate School of Information Science, Nara Institute of Science and Technology. His research interests include computer architecture, emulation, circuit design, and accelerators. He is a fellow of IEICE, a senior member of IPSJ, a member of IEEE CS and ACM.