Refinements in Data Manipulation Method for Coarse Grained Reconfigurable Architectures

Takuya Kojima and Hideharu Amano
Dept. of Information and Computer Science, Keio University, Yokohama, Japan
Email: wasmii@am.ics.keio.ac.jp

Abstract—**Coarse-grained reconfigurable architectures (CGRAs) is one of the suitable devices for IoT (Internet of Things) and edge-computing because of their high energy efficiency and programmability. The CGRAs process a compute-intensive part of an application program (especially a loop part) more efficiently than general purpose processors. CMA (Cool Mega Array) is an energy-conscious CGRA with a task-level reconfiguration instead of a cycle-level one. However, the CMA faces some limitations related to data management because of the aggressive pursuit of power saving. In this paper, we introduce a new CMA architecture *VPCMA2* to relax the constraints and to improve energy efficiency. Then, we implement it with a 65-nm process technology to evaluate a hardware overhead due to the improvement. According to the evaluation results, the new design does not influence its maximum operating frequency. Although new functionalities brought about 17% power overhead and 10% area overhead, a remarkable improvement of application mappability and data handling was achieved.**

I. Introduction

Recently, IoT (Internet of Things) and edge-computing have grown significantly, and consequently, highly energy-efficient and flexible devices are needed. Coarse-grained reconfigurable architectures (CGRAs) are one of the hardware platforms which satisfy these demands. CGRAs provide word-level reconfigurability instead of a bit-level one such as FPGAs (Field-Programmable Gate Arrays). Therefore, hardware overhead for the reconfigurability is small, and then they can achieve high energy efficiency close to that of ASICs (Application Specific Integrated Circuits).

CMA (Cool Mega Array) has been proposed as a low power CGRA [1]. Similar to other CGRAs, it consists of an array of PEs (Processing Elements) as illustrated in Fig. 1. The PE array performs a task-level reconfiguration. In other words, the configuration does not change during the execution of a single application kernel. This reconfiguration strategy reduces the dynamic power consumption dramatically, however, brings less flexibility than the CGRAs with cycle-by-cycle reconfiguration. To tackle this shortage, CMA has a dedicated micro-controller, which is a 16-bit tiny RISC processor and provides flexible data transfer between the PE array and data memory according to microinstruction codes.

In general, multi-bank data memory is connected to the PE array with a high bandwidth switch like a crossbar in order to avoid stall due to memory access conflicts. Likewise, previously proposed CMAs[2], [3] used “data manipulator” for the interconnection. The data manipulator is a $N \times N$ network and is composed of N multiplexers (MUXs). The number of input ports and output ports N is the same as that of memory banks. Thanks to the MUXs, input data to a port can be transferred to arbitrary output port according to a transfer table. There exist two data manipulators. One is used when loading data to the PE array from the data memory, and the other is for storing data to the data memory from the PE array.

In the CMA, accessed data for each bank is limited to be aligned sequentially in its address space. For instance, assuming 12 memory banks, the PE array can simultaneously load data in the address range of 0x0-0xB. Although such an access constraint causes inefficiency of data handling, it is sufficient for simple application kernels as used in [2], [3]. Besides, the constraint makes memory architecture simple so that power consumption related to the memory access can be reduced. Nevertheless, considering more practical applications, the access constraint is obviously unacceptable.

In this work, we propose a data access method removing the constraint. In addition to the memory access, a limitation of constant registers in the PE array is relaxed to enhance the mappability of complex application kernels. However, such improvement of functionality will increase area and power consumption. This paper analyzes trade-offs between rich data handling capability and hardware overheads for the task-level reconfiguration CGRAs such as CMA.

The rest of the paper is organized as follows. Overview of the latest CMA architecture and related work are introduced in Section II. The enhanced CMA architecture is proposed in Section III Then, evaluation results are shown in Section IV. Finally, the conclusion of this paper is summarized in Section V.

II. Background and related work

Some CGRAs support a cycle-level reconfiguration which changes the configuration of the PE array every clock cycle. However, this reconfiguration strategy requires a large amount of dynamic power consumption. Therefore, another type of CGRAs employ a task-by-task reconfiguration in order to save the dynamic power consumption. In this paper, we call those simple CGRAs “Straight Forward CGRAs” (SF-CGRAs).

The SF-CGRAs is designed to form a straightforward dataflow on their PE array as shown in Fig. 2. If input data are just forwarded through the PE array, computation results are finally outputted since the SF-CGRAs use the same

This work is partially supported by JSPS KAKENHI B Grant Number 18H03215 and JSPS KAKENHI Grant Number 19J21493.

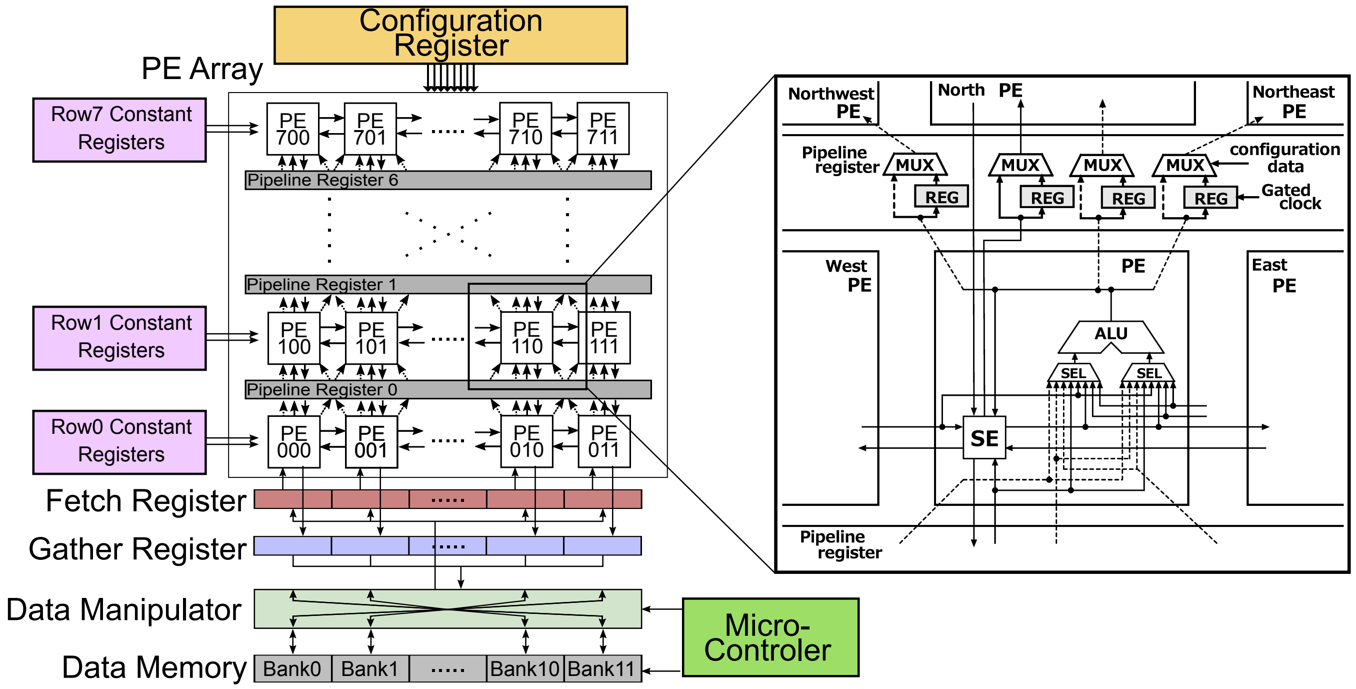


Fig. 1: Diagram of VPCMA architecture

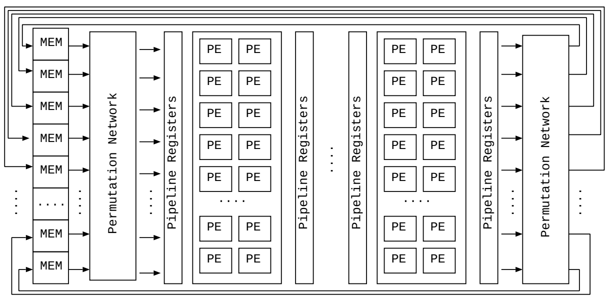


Fig. 2: An overview of SF-CGRA

configuration while handling a task of an application. Registers in the PE array can be regarded as pipeline registers. Permutation networks between the data memory and the input/output of the PE array provide flexible data transfer. For example, Piperench [4], Kilo-core [5], S5 engine [6], and EGRA [7] are classified into the SF-CGRAs. Some application mapping methods[8], [9], [10], [11] consider statical mapping during the task so that they can be applied to the SF-CGRAs.

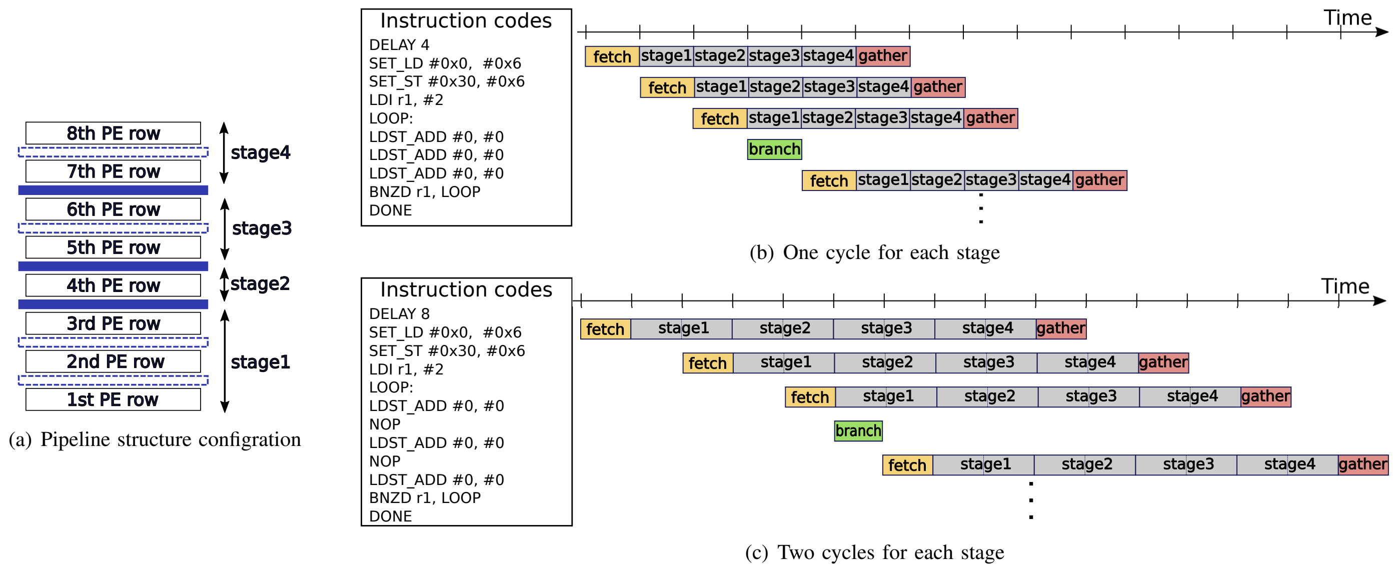
CMA architecture is also categorized in the SF-CGRA [1]. A PE of the CMA does not have a register file to hold intermediate results. Therefore, the PE array is composed of a large combinational circuit, and can save the dynamic power consumption since clock single for the PE array is not necessary. Although it suffers from a long critical path delay, the PE array is designed to be multi-cycle paths. Hence, it does not degrade a system clock frequency. The number of execution cycles is programmable depending on a mapped application. However, there still exists a performance bottleneck due to the low throughput of the multi-cycle PE array.

A. *VPCMA Architecture*

The latest version of the CMA called as VPCMA (Variable Pipelined CMA) has been proposed in order to overcome the bottleneck[3]. Unlike the original CMA, the PE array of VPCMA has a limited number of pipeline registers as shown in Fig. 1. They are placed between every row. VPCMA has the 8×12 PE array so that there are seven pipeline registers. It includes a micro-controller, a data manipulator and a 12-bank memory as well as previous CMAs[2].

As illustrated in Fig. 1, each PE is composed of an arithmetic logic unit (ALU), input selectors, and a switching element (SE). PE does not have register file so that an output of the ALU is transferred to the adjacent PEs through the link by the SE (the solid line in the PE) or direct-link (the dashed line in the PE). Three direct-links go to the north, northeast, and northwest directions. Constant registers provide two constant values for each PE row. If a PE row needs more than two constant values, unused constant registers in lower PE rows are utilized through the interconnection network. The pipeline register composition is also described in Fig. 1. Each pipeline register is activated by the configuration data and works as a standard register. On the other hand, deactivated pipeline registers are clock-gated, and input data are bypassed to upper PEs by the multiplexers. In this way, it can form various pipeline structures so as to fit a mapped application and to minimize the overhead of the pipelining. Please note that no register is placed on the south direction path from the north PE because it just returns computational results.

The micro-controller is a customized tiny RISC processor with the 16-bit length instruction set. It manages data transfer between the PE array and the banked data memory according to the instruction codes. “Fetch register” and “Gather register”



Instruction codes (b):

```
DELAY 4
SET_LD #0x0,  #0x6
SET_ST #0x30, #0x6
LDI r1, #2
LOOP:
LDST_ADD #0, #0
LDST_ADD #0, #0
LDST_ADD #0, #0
BNZD r1, LOOP
DONE
```

Instruction codes (c):

```
DELAY 8
SET_LD #0x0,  #0x6
SET_ST #0x30, #0x6
LDI r1, #2
LOOP:
LDST_ADD #0, #0
NOP
LDST_ADD #0, #0
NOP
LDST_ADD #0, #0
BNZD r1, LOOP
DONE
```

(a) Pipeline structure configration

(b) One cycle for each stage

(c) Two cycles for each stage

Fig. 3: Examples of pipeline execution of the PE array

are respectively connected to inputs and outputs of the PEs in the 1st row. The micro-controller writes data to the fetch register, and then, the PE array runs automatically. After a few cycles, the output data will be written back to the gather register. In this paper, we call the former operation is "fetch" and the latter one is "gather".

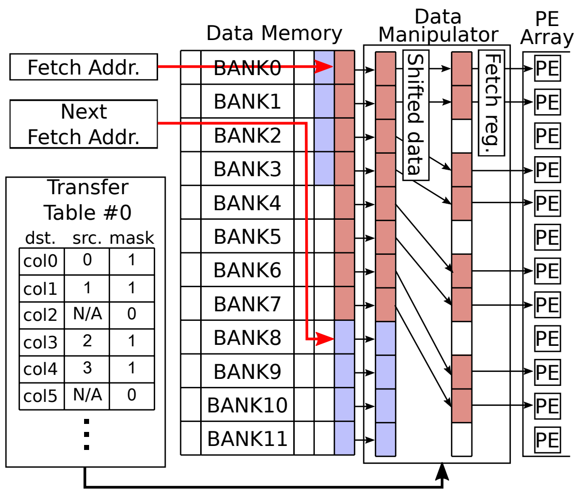
Here are execution examples of the VPCMA in Fig. 3. We consider two cases: 1) taking one cycle per stage (Fig.3(b)) and 2) taking two cycles per stage (Fig. 3(c)). In both cases, we assume a 4-stage pipeline structure as illustrated in Fig.3(a). "DELAY" operation is a dedicated instruction to specify the cycle number. For the one-cycle execution, four-cycle delay is designated whereas eight-cycle one is applied for the two-cycle execution. "SET_LD" and "SET_ST" respectively set configurations for fetch and gather operations. They require two arguments: the first means initial memory address for fetch/gather and the second is an incremental number of the address. The fetch and gather operations fuse together into an instruction "LDST_ADD". The fetch operation is executed as soon as the "LDST_ADD" is issued. In contrast, execution of the gather operation is delayed like Fig. 3(b). This instruction also needs two arguments to specify transfer tables for the data manipulator. Details of the table are explained in subsection. III-A. The fetch and gather address are automatically incremented by the specified number at the end of each execution. For the two-cycle execution, "NOP" instruction has to be inserted every "LDST_ADD"s. The micro-controller supports some branch instructions such as "BNZD" (Branch Not equal to Zero with Decrement). In both case, "BNZD" examines whether the value of register *r0* is not equal to zero, and then, branches and decrements the register value if true. These branch instructions take one cycle so that it delays the "LDST_ADD" in the next iteration in case of the one-cycle execution. On the contrary, for the two-cycle execution, it substitutes the "NOP" instruction. Besides, the micro-controller executes integer arithmetic and logical operations between general purpose registers. Finally, "DONE" instruction is executed, and then, completion of the task is notified to a host processor.

B. Data manipulator

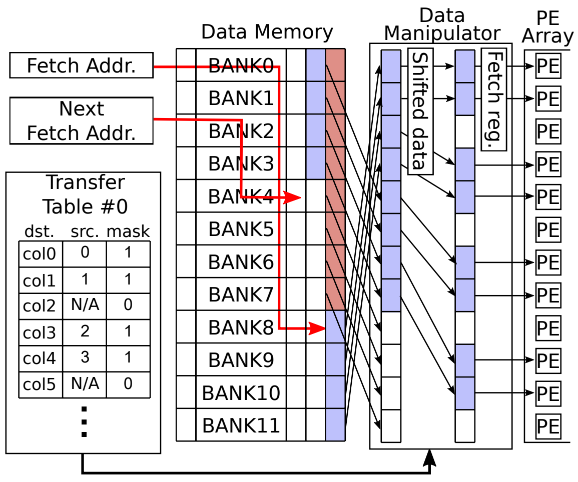
The data manipulator takes a role as the permutation network which provides flexible data transfer among the banked memory, fetch register, and gather register. Fig. 4 shows examples of fetch operation with the data manipulator. In the examples, the fetch address and the incremental number are respectively initialized to 0x0 and 8 by the "SET_LD" instruction. When the fetch operation is executed, twelve consecutive words starting at the fetch address are loaded from the 12-way interleaved bank memory. Then, they are shifted as a bank data at the fetch address is placed on the first input if necessary. In case of the first fetching (Fig. 4(a)), it is not necessary whereas the second fetching (Fig. 4(b)) needs the shift operation because the 8th bank contains the head data. The data manipulator is composed of twelve multiplexers so that it can transfer any input of the shifted data to any position of the fetch register depending on a transfer table. The transfer table includes a mask pattern as well. VPCMA has 16 tables for the fetch and gather operation respectively. Therefore, if an application task contains different memory access patterns, the micro-controller just changes transfer tables without reconfiguration.

C. Related work on data management

Regardless of reconfiguration policy, memory access by the PE is usually limited to some extent. For most of CGRAs supporting the cycle-level reconfiguration, PEs in the same row shares links to the data memory so that they can not access to the data memory simultaneously. Hence, it is important to take care of memory allocation because inefficient data placement brings about significant performance degradation. Some researches propose optimization methods for the data



(a) 1st fetch



(b) 2nd fetch

Fig. 4: Data transfer by the data manipulator

placement [12], [13], [14]. MEMMap[15] is one of sophisticated mapping technique to exploit the data memory as routing resource between PEs. It also considers memory allocation since the efficiency of the memory access impacts on the mapping quality significantly. However, they assume the cycle-level reconfiguration so that it is not clear whether these methods are appropriate for the SF-CGRAs.

A powerful memory architecture is also employed as another approach to improve data management. CPM[16] and SPIRA[17] present enhanced memory architectures for data sharing between the PE array and controller. Although they can reduce data transfer time and control overhead, they do not contribute to an improvement in the mappability of applications.

III. Proposed Architecture

The VPCMA architecture explained in the previous section has the following three limitations in data handling. In this work, we propose a new architecture VPCMA2 relaxing these constraints.

- Sequential interleaving access to the banked memory
- Only two constant values available for each PE row
- No general-purpose data bus for the micro-controller

A. Enhancement of data manipulator

In order to overcome the bank access constraint, VPCMA2 improves its data manipulator. Assuming a sample loop code in Fig. 5(a), the previous data manipulator cannot handle it naturally because an element of the array a is at most 64 words apart from that of array b. The distance is too far for the previous data manipulator which can access 12 consecutive words. Therefore, we have to rearrange the elements of array a and array b alternately like $a[0], b[0], a[1], b[1], ...$ in the data memory. However, the rearrangement takes additional data transfer time. In addition, array b have to be duplicated three times for $a[16 : 31], a[32 : 47], anda[48 : 63]$ so that it wastes more memory space.

Then, we add address offset as new content of the transfer table as described in Fig. 5(c). The transfer table includes the same contents illustrated in Fig. 4(a), and they are omitted because of limited space. Fig. 5(c) and Fig. 5(d) show examples of improved bank access while executing the loop code. In this example, four iterations are executed on the PE array simultaneously like a SIMD processor. In the beginning, the address for each bank is calculated based on the fetch address according to the algorithm in Fig. 5(b). Although the algorithm itself is not different from the previous data manipulator, the proposed one adds each offset to each calculated address. In the 1st iteration, all calculated addresses are 0, and, 5th-8th bank's offsets are set to 5 to fetch $a[0 : 3]$ and $b[0 : 3]$ simultaneously. Then, each bank address is updated in the 2nd iteration since the fetch address is incremented by 4. Similar to the data shifting as explained in the previous section, the address offsets are shifted. Hence, we can use the same transfer table. Even though the access pattern is different, we just switch the transfer tables. In this example, we have to change it between the 4th iteration (for $a[12 : 15]$) and the 5th iteration (for $a[16 : 19]$) because offset values to the array b is also changed. Besides, array allocation should be optimized to avoid a bank conflict.

B. Improved connectivity to the constant registers

The constant registers of VPCMA provide 16 constant values with the PE array. Nevertheless, the PEs cannot use them freely. A PE row is connected to a constant register containing two constant values. If PEs in the row need more constant values, they have to borrow constant registers from different rows through the interconnection network on the PE array. However, it consumes more routing resources and often results in application mapping failure due to the insufficient resource.

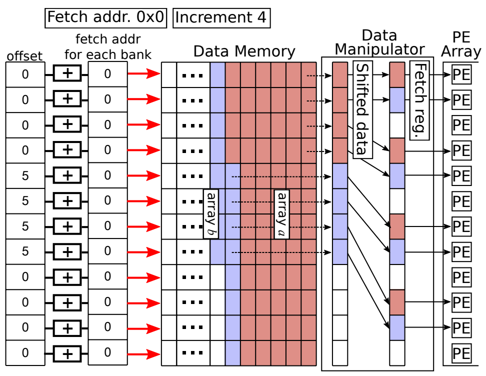
To eliminate the limitation, we enrich the connectivity between the PEs and the constant registers. The number of constant values in the VPCMA2 remains sixteen, yet PEs can use any of these values. Instead, SEs cut interconnection links for the constant value routing. It causes an increase in

```
for $i \leftarrow 0$ to $N = 64$ do
    $c[i] \leftarrow a[i] \times b[i \bmod 16]$
end for
```

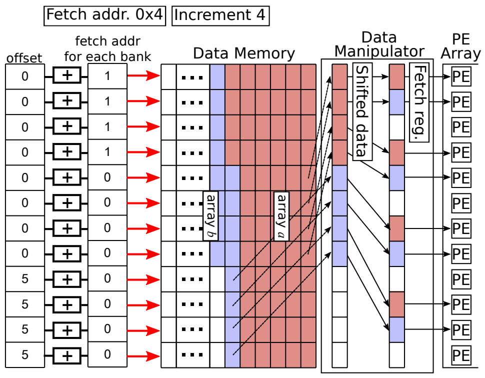
(a) Loop sample

```
for $i \leftarrow 0$ to 11 do
    $fetch\_addr_i \leftarrow fetch\_addr$ div 12
    if $i < fetch\_addr$ mod 12 then
        $fetch\_addr_i \leftarrow fetch\_addr_i + 1$
    end if
end for
```

(b) Address calculation for each bank



(c) 1st iteration



(d) 2nd iteration

Fig. 5: Improvement in the interleaving memory access

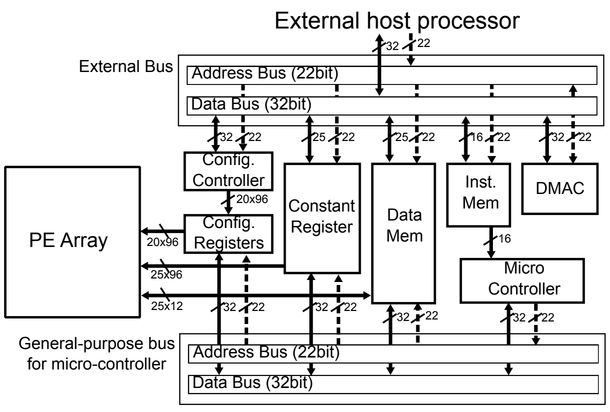


Fig. 6: Extended data bus of VPCMA2

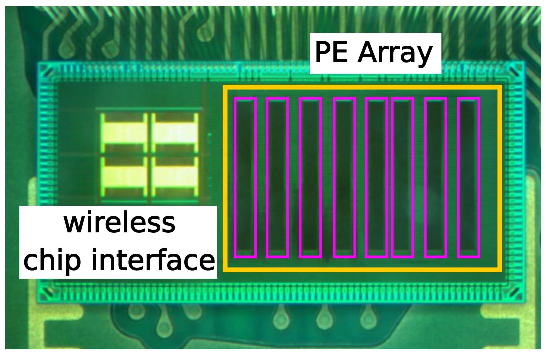


Fig. 7: Chip photograph of the VPCMA

configuration data, and its impact is evaluated in the next section.

C. Extended instruction set of the micro-controller

VPCMA has only an external bus to communicate with a host processor. The host processor reconfigures the PE array and transfers all of the data including instruction codes, processing data, and constant values through the bus. In other words, the VPCMA needs control by the host processor to change some data even though the changed part is quite small. Moreover, the micro-controller cannot load data in the data memory to its general purpose registers. Therefore, it cannot handle an application task whose iteration count depends on its own processing result.

To address the issue, we add a new general-purpose bus to the new micro-controller as shown in Fig. 6, and then, extends its instruction set to use the bus. Each component such as data memory and instruction memory is placed in 22-bit address space. The address space for the bus is common to the external bus. However, the word length of the micro-controller is 16-bit whereas most of the other components are designed with 25-bit or 32-bit word length. Therefore, two registers are combined to store a double-word number. Double-word instructions including “ADD.D” are accordingly implemented. Thanks to the bus, the VPCMA2 can do self-reconfiguration efficiently as far as the difference in the configuration is relatively small. A bus arbiter prioritizes the bus over the external bus while the micro-controller is working.

IV. Evaluation

A. Evaluation Setup

In order to evaluate hardware overhead due to relaxing the limitation of data manipulation, we have designed the proposed VPCMA2 architecture with Verilog HDL. Then, the design has been synthesized with Renesas SOTB 65-nm technology. The previous version VPCMA have also been implemented as a real chip with the same process [11]. Fig. 7 is a chip photograph of the real chip. These specifications are shown in Table I. For the SOTB process, two options were available: LP (Low Power) and LSTP (Low STanby Power). The LSTP version consumes lower leak power than the LP one, while it consumes larger dynamic power because of the higher standard supply voltage. According to preliminary evaluation, an RTL design can be synthesized at about 25% faster timing constraints with LSTP version than LP one. The VPCMA employs the LP version whereas the VPCMA2 utilizes the LSTP version since only the LSTP option is currently available.

B. Hardware overhead

1) Maximum operation frequency: Compared to the VPCMA, the capability of the data manipulator of VPCMA2 is improved. It could bring degradation of maximum operating frequency. To evaluate its effect, we designed two versions

TABLE I: Implementation environment and power consumption

	VPCMA[11]	VPCMA2 (this work)
Design	Verilog HDL	
Process Library	Renesas SOTB 65 nm LP (Low Power)	Renesas SOTB 65 nm LSTP (Low STandby Power)
Standard supply voltage	0.55 V	0.75 V
Synthesis	Synopsys Design Compiler 2016.03-SP4	Synopsys Design Compiler 2017.09-SP1
Place and route	Synopsys IC Compiler 2016.03-SP4	N/A
	Power consumption at 30MHz	
Static (μW)	126.0	25.18
Dynamic (mW)	3.337	4.029
Total (mW)	3.463	4.053

TABLE II: Hardware overhead for each design

	VPCMA	VPCMA2 (1 cycle f/g)	VPCMA2 (2 cycle f/g)
Frequency (MHz)	87.71	95.23	125.0
75% scaled	N/A	71.42	93.75
Cell area (mm^2) except for PE array	10.04	14.55	14.22

of the VPCMA2: 1) one-cycle fetch/gather operation and 2) two-cycle fetch/gather operation. Table. II shows achieved maximum frequency for each design reported by synthesis results. It includes 75% scaled values to consider the process difference. Obviously, two-cycle version achieves higher frequency than one-cycle version. In addition, it enhances the operating frequency by around 6% compared to the VPCMA despite the data manipulator extension. This is because the original data manipulator of the VPCMA contains the critical path while the two-cycle version of the VPCMA makes another part of its critical path. Of course, the frequency of the VPCMA becomes high by applying the two-cycle fetch/gather. However, the increase is expected only to the same extent as the VPCMA2. In contrast, one-cycle design degrades the frequency by about 18% compared to the VPCMA.

2) Area overhead: The improvement of the VPCMA2 requires more chip area than the VPCMA. Then, we analyze the standard cell area of the synthesized netlists at 50MHz timing constraint that is the same condition as the real chip implementation of the VPCMA. Table. II summarizes cell area for each design without the PE array. Although both designs of the VPCMA2 increase about 40% area compared to that of the VPCMA, the PE array occupies most of the cell area for all designs. In case of the two-cycle version, the whole cell area is 50.28mm^2 so that the total area overhead is less than 10%. The two-cycle version of the VPCMA2 is slightly smaller than the one-cycle version. Considering the result of maximum frequency, the two-cycle version is appropriate rather than the one-cycle version. One cycle latency for the bank access does not matter because the computation on the PE array naturally takes several cycles latency. Hereafter, only the two-cycle design is employed.

3) Power overhead: Here, the power consumption of the VPCMA2 is compared with that of the VPCMA. Table. I also shows result of the power consumption. The power of

TABLE III: Application kernels

kernel	Description	Data size	Operations
RGB2YCC	converting color encodings	4 blocks	30
DCT	Discrete Cosine Transform	6 blocks	63
Quantize	Quantization of the DCT coefficient	12 blocks	15

VPCMA is a measurement result of real chip experiment[11] while that of VPCMA2 is simulated with Synopsys PrimeTime and based on switching activities obtained by Cadence NC-Verilog gate-level simulation. To simulate the power under the same condition as the VPCMA experiment, clock frequency is set to 30MHz, and an identical application mapping is utilized. Used application is grayscale of an image. The static power of the VPCMA2 is about 5$\times$ smaller than VPCMA because of the difference of SOTB process version. Nevertheless, in both cases, the dynamic power consumption is dominant. As a result, VPCMA2 increases 17% total power consumption compared to the VPCMA. However, it is partially due to the increase of standard supply voltage from 0.55 V to 0.75 V. Hence, the power overhead attributed to the extended functionality is considered to be small.

C. Increase of configuration data

The proposed scheme increases two types of configuration data: 1) contents of the transfer table and 2) mapping between the constant registers and the PEs. We evaluate them quantitatively.

1) Transfer table: In the previous VPCMA, the transfer table consists of 4-bit $\times$ 12 = 48-bit of multiplexer configuration and 12-bit mask. However, in fact, they are treated as three words (1 word = 25-bit) due to data segmentation. Besides, the transfer table of the VPCMA2 contains the offset values as proposed in the previous section. Memory depth for each bank is 64 words in both the VPCMA and VPCMA2, and thus, at most 6-bit for each bank is needed. They can be packed into three words. Therefore, the size of transfer table of VPCMA2 is twice as large as that of VPCMA.

2) Constant mapping: In case of the VPCMA, PEs in the same row share two constant values. Each PE specifies which constant values to use as a part of ALU configuration. For VPCMA2, additional 4 bits for each PE are needed to choose one from sixteen values. Considering 20-bit configuration for ALU and SE in the PE, 4-bit configuration for the constant mapping causes 20% increase of configuration data.

D. Improvement on application tasks

To demonstrate the improvement of data management in the VPCMA2, we use three application tasks included in JPEG encoding as listed in Table. III. 8$\times$8 chunk of pixels is treated as one block (minimum coded unit), and 4:1:1 of sampling factor is employed. They are based on MiBench source code [18]. At first, the source codes written in C language are compiled by clang 4.0, which is a front-end compiler of LLVM 4.0 [19], with *-O3* option. Clang can generate LLVM-IR (Intermediate Representation) which is independent of any specific architecture. Then, data-flow-graph (DFG) mapped to

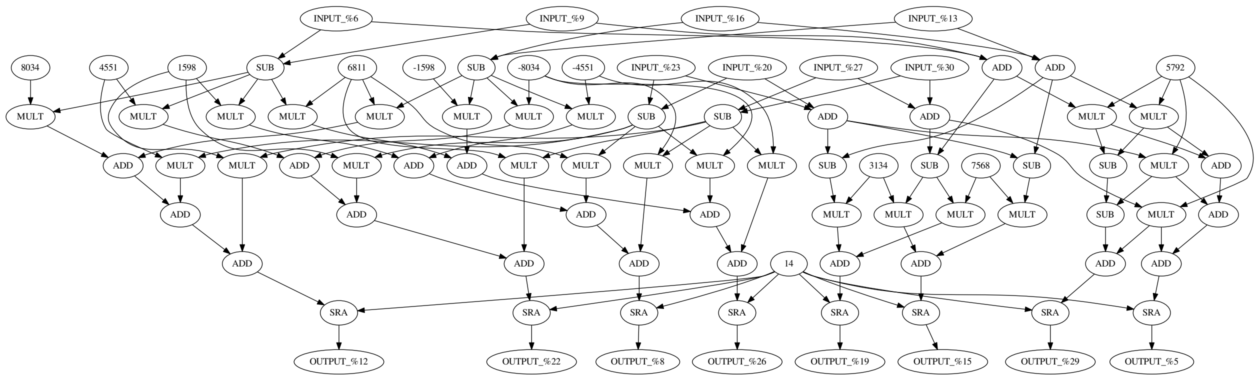


Fig. 8: DFG of DCT kernel

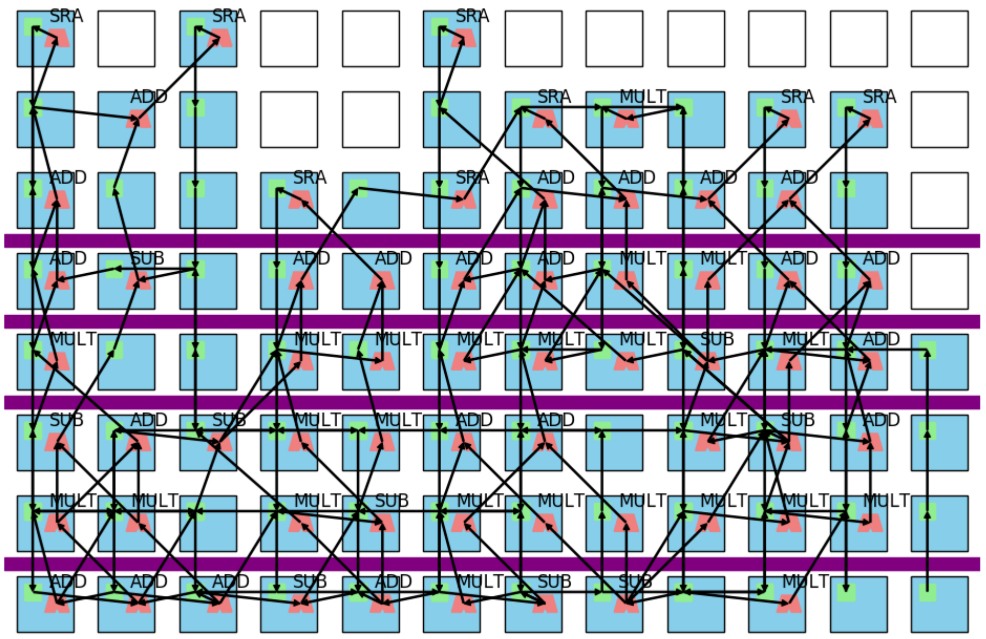


Fig. 9: Application mapping of DCT kernel for VPCMA2

the PE array is obtained from the LLVM-IR. The application mapping is optimized by using a mapping method based on the genetic algorithm [11]. Fig. 8 is an extracted DFG of DCT kernel, and an optimized mapping for the VPCMA2 is shown in Fig. 9. Strings and numbers in the circles of Fig. 8 respectively denote operation types and constant values. Others mean input data or output data. Blue rectangles of Fig. 9 indicate PEs. Pink diagram and green rectangle in the PE respectively denote ALU and SE. Purple bar means an activated pipeline register. Therefore, this mapping makes 5-stage pipeline.

Fig. 10 shows comparison of execution time for each task in both architectures. In this evaluation, we assume two types of data transfer by the host processor. One is single-read/write, and the other is block-read/write. The single-read/write takes two cycles each of which is respectively for address and for data whereas the block-read/write transfers sequential 16-words data in seventy cycles (likewise, one is for address and

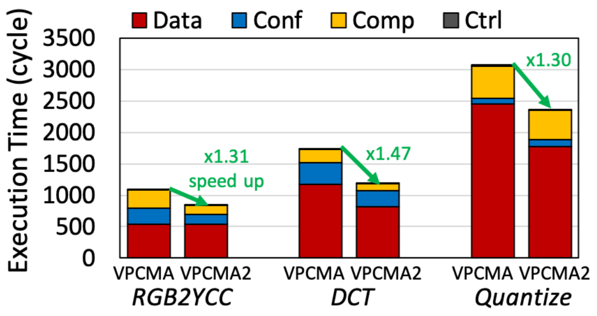


Fig. 10: Execution time for each application

others are for data).

1) RGB2YCC: In case of VPCMA, the DFG of *RGB2YCC* cannot be mapped because of limitation in the constant registers. Then, it is divided into three sub-DFGs: i)RGB-to-Y, ii) RGB-to-Cb, and iii) RGB-to-Cr, and they are executed in turn. On the other hand, PE array of the VPCMA2 can accommodate the whole of DFG so that it performs the computation without reconfiguration. Although configuration data of the VPCMA2 are larger than that of VPCMA as explained in sectionIV-C, the reduction of reconfiguration count results in x1.31 speed-up.

2) DCT: Similar to the *RGB2YCC*, the DFG of *DCT* (Fig. 8) have to be divided for the VPCMA. In this case, the DFG is partitioned into two by Kernighan–Lin algorithm to minimize intermediate results. The second DFG needs the same input data as the first one and also the intermediate result produced by the first one. Some words in the data memory are preserved for the intermediate result every 8-word input data because input data to the VPCMA's PE array must be consecutive. Therefore, using the single-write is faster than the block-write, and takes $64\times2 = 128$ cycles to transfer 64 words (one block). On the contrary, VPCMA2 can perform 8-point DCT without the partitioning. Hence, data transfer time and configuration time are reduced. Consequently, x1.47 speed-up was achieved.

3) Quantize: Unlike other two kernels, DFG of *Quantize* needs two arrays of input data: first is the DCT coefficient and second is a quantize table. The quantize table is commonly used for each block. Once it is transferred to the data memory, the VPCMA2 can reuse it by using the enhanced data manipulator. In case of the VPCMA, both arrays have to be transferred every time as described in sectionIII-A. It causes longer data transfer time than the VPCMA2. However, the VPCMA2 needs to change offset values of the transfer table in several iterations. The time to modify them is included in the computation time of VPCMA2. The micro-controller undertakes the transfer table modification through the extended data bus. Therefore, slight increase of the computation time is observed. As a result, the VPCMA2 executes the kernel x1.30 faster than the VPCMA. Nevertheless, the more blocks are used, the more reduction of execution time is achieved thanks to the reuse of the quantize table.

V. Conclusion

This work focuses on SF-CGRAs with a task-level reconfiguration. Although these SF-CGRAs usually have some limitations to reduce the hardware overheads, it causes inefficiency to execute complex tasks because of less mappability of application. This work has introduced a relaxed constraint SF-CGRA named *VPCMA2*. As the evaluation result, it can achieve on average x1.35 speed-up with less than 17% power and less than 10% area overhead compared to the original *VPCMA*.

Acknowledgment

This work is supported by VLSI Design and Education Center(VDEC), the University of Tokyo in collaboration with Synopsys, Inc and Cadence Design Systems, Inc.

References

[1] N. Ozaki, Y. Yasuda, M. Izawa, Y. Saito, D. Ikebuchi, H. Amano, H. Nakamura, K. Usami, M. Namiki, and M. Kondo, "Cool Mega-Arrays: Ultralow-Power Reconfigurable Accelerator Chips," *IEEE Micro*, vol. 31, no. 6, pp. 6–18, Nov 2011.

[2] K. Masuyama, Y. Fujita, H. Okuhara, and H. Amano, "A 297mops/0.4mw ultra low power coarse-grained reconfigurable accelerator CMA-SOTB-2," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2015, pp. 1–6.

[3] N. Ando, K. Masuyama, H. Okuhara, and H. Amano, "Variable Pipeline Structure for Coarse Grained Reconfigurable Array CMA," in *2016 International Conference on Field-Programmable Technology*, 2016, pp. 231–238.

[4] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. R. Taylor, "PipeRench: A virtualized programmable datapath in 0.18 micron technology," in *Custom Integrated Circuits Conference, 2002. Proceedings of the IEEE 2002*. IEEE, 2002, pp. 63–66.

[5] B. Levine, "Kilocore: Scalable, High Performance and Power Efficient Coarse Grained Reconfigurable Fabrics," in Proc. of International Symposium on Advanced Reconfigurable Systems, 2005, pp. 129–158.

[6] J. M. Arnold, "S5: The Architecture and Development Flow of a Software Configurable Processor," in *Proc. of the 4th IEEE Int'l Conf. on Field Programmable Technology (ICFPT2005)*, December 2005, pp. 120–128.

[7] G. Ansaloni, P. Bonzini, and L. Pozzi, "EGRA: A coarse grained reconfigurable architectural template," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 6, pp. 1062–1074, 2011.

[8] D. Liu, S. Yin, G. Luo, J. Shang, L. Liu, S. Wei, Y. Feng, and S. Zhou, "Data-Flow Graph Mapping Optimization for CGRA with Deep Reinforcement Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[9] J. W. Yoon, A. Shrivastava, S. Park, M. Ahn, R. Jeyapaul, and Y. Paek, "SPKM: A novel graph drawing based algorithm for application mapping onto coarse-grained reconfigurable architectures," in *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*. IEEE Computer Society Press, 2008, pp. 776–782.

[10] S. Yin, D. Liu, L. Sun, L. Liu, and S. Wei, "DFGNet: Mapping dataflow graph onto CGRA by a deep learning approach," in *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 1–4.

[11] T. Kojima, N. Ando, Y. Matshushita, H. Okuhara, N. A. V. Doan, and H. Amano, "Real chip evaluation of a low power CGRA with optimized application mapping," in *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*. ACM, 2018, p. 13.

[12] Y. Kim, J. Lee, A. Shrivastava, J. W. Yoon, D. Cho, and Y. Paek, "High throughput data mapping for coarse-grained reconfigurable architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 11, pp. 1599–1609, 2011.

[13] Y. Kim, J. Lee, A. Shrivastava, and Y. Paek, "Memory access optimization in compilation for coarse-grained reconfigurable architectures," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 16, no. 4, p. 42, 2011.

[14] Z. Zhao, Y. Liu, W. Sheng, T. Krishna, Q. Wang, and Z. Mao, "Optimizing the data placement and transformation for multi-bank CGRA computing system," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 2018, pp. 1087–1092.

[15] S. Yin, X. Yao, D. Liu, L. Liu, and S. Wei, "Memory-aware loop mapping on coarse-grained reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 5, pp. 1895–1908, 2016.

[16] T. X. Mai and J. Lee, "Software-managed automatic data sharing for Coarse-Grained Reconfigurable coprocessors," in *Field-Programmable Technology (FPT), 2012 International Conference on*. IEEE, 2012, pp. 277–284.

[17] J. Lee, Y. Jeong, and S. Seo, "Fast shared on-chip memory architecture for efficient hybrid computing with CGRAs," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 1575–1578.

[18] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. IEEE, 2001, pp. 3–14.

[19] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*. IEEE Computer Society, 2004, p. 75.