Energy-Efficient and High Performance CGRA-based Multi-Core Architecture

Yoonjin Kim and Heesun Kim

Abstract—**Coarse-grained reconfigurable architecture (CGRA)-based multi-core architecture aims at achieving high performance by kernel level parallelism (KLP). However, the existing CGRA-based multi-core architectures suffer from much energy and performance bottleneck when trying to exploit the KLP because of poor resource utilization caused by insufficient flexibility. In this work, we propose a new ring-based sharing fabric (RSF) to boost their flexibility level for the efficient resource utilization focusing on the kernel-stream type of the KLP. In addition, based on the RSF, we introduce a novel inter-CGRA reconfiguration technique for the efficient pipelining of kernel-stream on CGRA-based multi-core architectures. Experimental results show that the proposed approaches improve performance by up to 50.62 times and reduce energy by up to 50.16% when compared with the conventional CGRA-based multi-core architectures.**

Index Terms—**Embedded systems, coarse-grained reconfigurable architecture (CGRA), multi-core, kernel level parallelism (KLP)**

I. INTRODUCTION

The flexibility of a system is very important to accommodate the short time-to-market requirements for embedded systems. On the other hand, application-specific optimization of embedded system becomes inevitable to satisfy the market demand for designers to meet tighter constraints on cost, performance and power. To compromise these incompatible demands, coarse-grained reconfigurable architecture (CGRA) has emerged as a suitable solution for embedded systems [1]. It can boost the performance by adopting multiple processing elements while it can be reconfigured to adapt to evolving characteristics of the embedded applications like audio, video and graphics processing. However, there is a limit when a CGRA is expected to improve the performance of an entire application. This is because single CGRA is sequentially optimized for the parallelized computations in a kernel at a time whereas the overall speedup of the entire application can be achieved by kernel level parallelism (KLP) that several kernels concurrently run. Therefore, such a limitation of single CGRA has resulted in the appearance of CGRA-based multi-core architecture which allows for the multi-CGRA to support diverse KLPs – running separate kernels or inter-dependent kernels (kernel-stream) in parallel.

However, the existing CGRA-based multi-core architectures suffer from much energy and performance bottleneck when trying to achieve the KLP. This is because the existing multi-CGRA structures are not flexible enough to adaptively support various cases of the KLP. It means that the resources in the multi-CGRAs cannot be efficiently utilized under monotonous aggregation of several CGRAs. Therefore, boosting their flexibility level for the efficient resource utilization is considered as a serious concern. For improving their flexibility, this paper provides a new multi-CGRA fabric with a novel reconfiguration technique focusing on the kernel-stream type of the KLP and its hardware

Manuscript received Nov. 24, 2013; accepted Apr. 7, 2014
Dept. of Computer Science, Sookmyung Women's University, Korea
E-mail : {ykim, khs9101}@sookmyung.ac.kr

implementation.

The paper has following contributions:

- A new ring-based sharing fabric (RSF) has been proposed for raising the flexibility level of the CGRA-based multi-core architectures.
- A novel inter-CGRA reconfiguration technique on the RSF has been introduced for efficient pipelining of kernel-stream on CGRA-based multi-core architectures.
- RT-level design and synthesis have been carried out with varying the number of CGRAs to demonstrate the cost-effectiveness of the proposed approaches in reducing energy while enhancing its performance compared with the existing architecture model.

This paper is organized as follows. After the related work in Section II, we briefly describe CGRA-based multi-core architecture and its representation as preliminaries in Section III. In Section IV, we present the motivation of our approaches. Then we propose the ring-based sharing fabric (RSF) and the inter-CGRA reconfiguration technique in section V. Section VI illustrates the inter-CGRA control mechanism and the experimental results are given in Section VII. Finally we conclude the paper in the Section VIII.

II. Related Works

Until now, there have been a few multi-core architecture projects based on CGRAs for kernel-level parallelism [2-5]. However, most of them are monotonous aggregation of several CGRAs. For example, Samsung reconfigurable processor (SRP)-based multiprocessors have been presented in [6, 7]. In [6], the multiprocessor consists of sixteen reconfigurable processors through networks-on-chip (NoC) interconnection with mesh type topology for exploiting data parallelism of volume rendering. However, the experimental result shows modest performance improvement compared with CPU/GPU improvement. It is because there is performance limitation with general NoC-based multi-core architecture. Another SRP-based multi-core architecture is shown in [7] - it is composed of ARM9 processor, two CGRAs, and AHB bus which couples them. Even though they have demonstrated the software implementation of DVB-T2 on dual CGRAs running at 400 MHz, this work also shows performance limitations because of inefficient resource utilization in the multi-core architecture. In addition, power/energy evaluation of the multiprocessors are not shown in both cases [6, 7].

The Xentium tiles [8] is another example of CGRA-based multi-core architecture. The Xentium is a programmable digital signal processing tile and the different tile processors are connected to a router on NoC. It turns out that the hyperspectral image compression algorithm can indeed be efficiently mapped on this multi-tiled architecture and adding more tiles give a close to linear speedup. However, it is unclear whether other applications may be also successfully mapped on to the architecture already specialized for the compression algorithm. In addition, adding more tiles means increase of power consumption as well as speedup but such a power issue has not been dealt with in [8].

Multi-core architecture with dynamically reconfigurable array processors [10] is more flexible than [6-9] because the shared data-memory banks are connected to all processing cores through crossbar switches unlike communication among the CGRAs is only restricted by NoC or on-chip bus in [6-9]. However the centralized shared data-memory banks may cause performance bottleneck with much power consumption when the number of cores increases. In addition, there are no quantitative evaluation and analysis about power, area, and timing with increasing the number of cores in [10].

III. Preliminaries

1. CGRA-Based Multi-Core Architecture

Typically, a coarse-grained reconfigurable architecture (CGRA)-based multi-core architecture includes general purpose processors (GPP), multi-CGRA, and their interface. Fig. 1 shows such an example of the CGRA-based multi-core architecture – it is composed of a GPP, a DMA, four CGRAs, and on-chip communication architecture like networks-on-chip (NoC) or on-chip bus which couples them. The GPP executes control intensive, irregular code segments and the multi-CGRA performs data-intensive kernel code segments – in this paper, we make use of the multi-CGRA in Fig. 1 as base architecture for comparison with proposed architecture.

Each CGRA consists of PE array (PA), data buffer

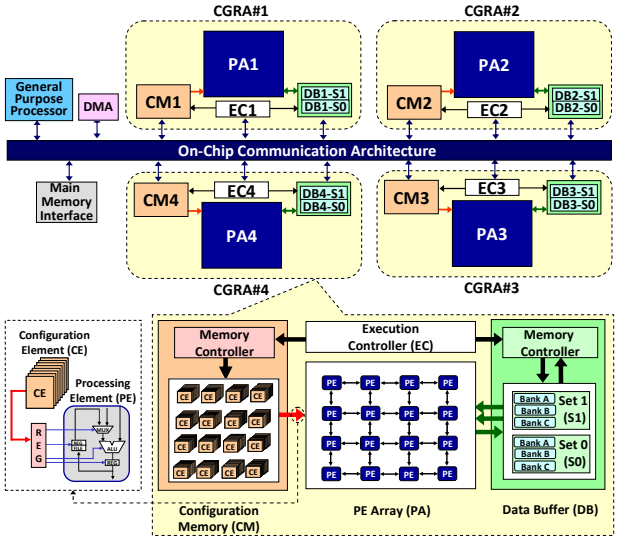


Fig. 1. CGRA-based multi-core architecture.

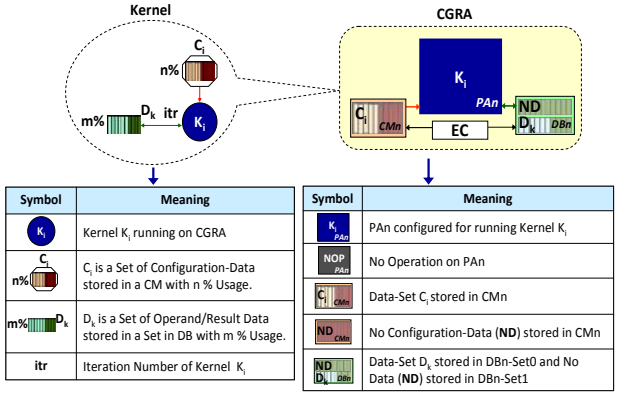


Fig. 2. Symbolic representation of kernel running on CGRA (a) Symbol for kernel, (b) Symbol for CGRA.

(DB), configuration memory (CM), and execution controller (EC). The PA has identical processing elements (PEs) containing functional units and a few storage units. The PA has reconfigurable interconnections between PEs for efficient data-transfer. The DB provides operand data to PA through a high bandwidth data bus. The CM is composed of configuration elements (CEs) and each CE provides context word to configure each PE. The EC has control data that contains execution cycles, read/write mode and addresses of the DB and the CE for correct operations of the PA.

2. Symbolic Representation

In this paper, we bring up the problems of resource utilization in the conventional multi-CGRA and propose new approaches to overcome such issues. Therefore, panoptic illustration of resource utilization in multi-CGRA is necessary for intelligible explanation of our approaches. In this section, we define an efficient way expressed in symbols to show such a utilization status as Fig. 2 It shows the symbolic representation of the resource utilization with CM/DB usage when kernel K_i run on a CGRA. The meaning of the symbols for kernel and CGRA are defined in Figs. 2(a) and (b) respectively.

IV. Motivation

In this section, we present the motivation of our approaches. The main motivation comes from the resource utilization problems when trying to exploit kernel level parallelism (KLP) on multi-CGRA. Even though various cases of the KLP can be considered, we focus on the pipelining of kernel-stream that is the most complex and ever-changing case.

1. Pipelining of Kernel-Stream

Pipelining of kernel-stream is a type of the KLP and means that interdependent kernels (kernel-stream) iteratively run on multi-CGRA in the manner of pipelining. In this case, each kernel may be mapped on each CGRA without any problems of resource utilization if each kernel requires CM/DB usage less than CM/DB capacity of each CGRA. However, there may be more cases causing poor utilization of resources as Example#1 in Fig. 3 - four interdependent kernels (kernel-stream) iteratively run on the base multi-CGRA. First of all, Example#1 shows lack of DB resources when mapping kernel K_A on CGRA#1 as Fig. 4 - K_A requires 400% DB usage for 40 iterations whereas DB capacity of a CGRA is 200%. In this case, DMA-transfer 'to DB1 (200%)' and CGRA-computation 'Pipelining for 21~40 iterations' should be sequentially performed because of insufficient DB capacity. Such a sequential operation causes performance bottleneck. However, if a DB has sufficient capacity (400%), it allows overlap of the DMA-transfer with the CGRA-computation without performance bottleneck as the bottom of Fig. 4.

In addition, pipelining of kernel-stream on the base multi-CGRA causes another case of the performance

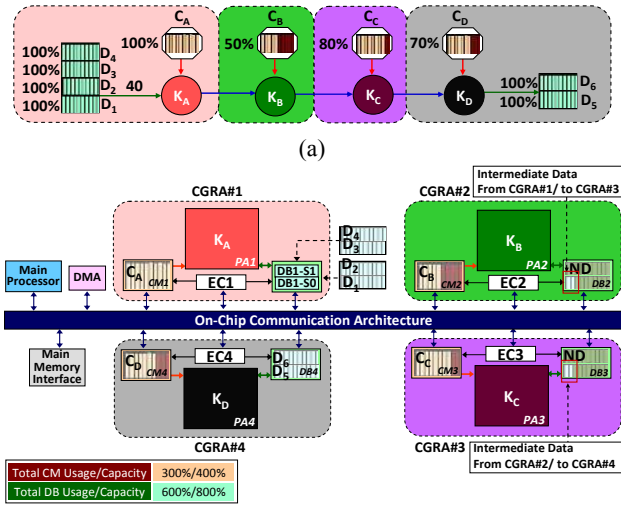


Fig. 3. Example#1 – pipelining of the kernel-stream composed of four interdependent kernels on the base multi-CGRA (a) Kernel-stream with four interdependent kernels, (b) Kernel-stream iteratively runs on the base multi-CGRA.

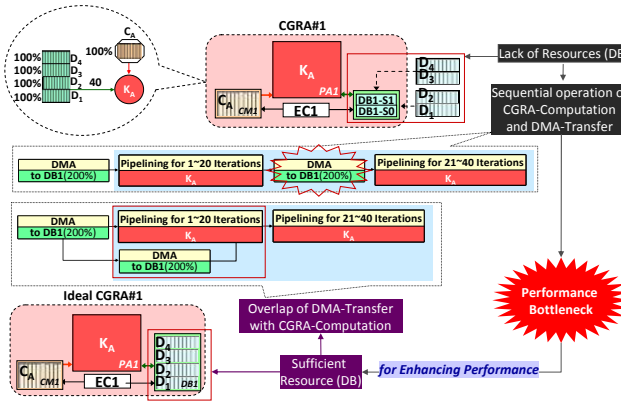


Fig. 4. Lack of resources (DB) in Example#1.

bottleneck and waste of energy. It is because DB read/write operations frequently occur when the result data from the previous CGRA transfer to the next CGRA as input data through on-chip communication architecture. Fig. 5(a) shows such problems in Example#1 in detail. DMA transfer through on-chip communication architecture means frequent DB read/write operation and it leads to the waste of energy and performance bottleneck. However, if adjacent Pas are directly connected together in Fig. 5(b), it allows direct data-transfer without DB read/write operations passing through on-chip communication architecture.

To sum it up, the kernel-stream (K_A~K_D) in Example#1 can be successfully mapped on the ideal multi-CGRA as Fig. 6 with reducing energy and enhancing performance compared with the base multi-CGRA. A point to consider is that the ideal multi-CGRA does not include more CM/DB resources compared with the base multi-CGRA - shown in the bottom of Fig. 6, its total CM capacity (400%) is the same size as the base multi-CGRA and its total DB capacity (600%) is less than the capacity (800%) of the base multi-CGRA. Therefore, the ideal multi-CGRA is only configuration with different number of DBs per a PA with the direct interconnection while keeping within the bounds of the

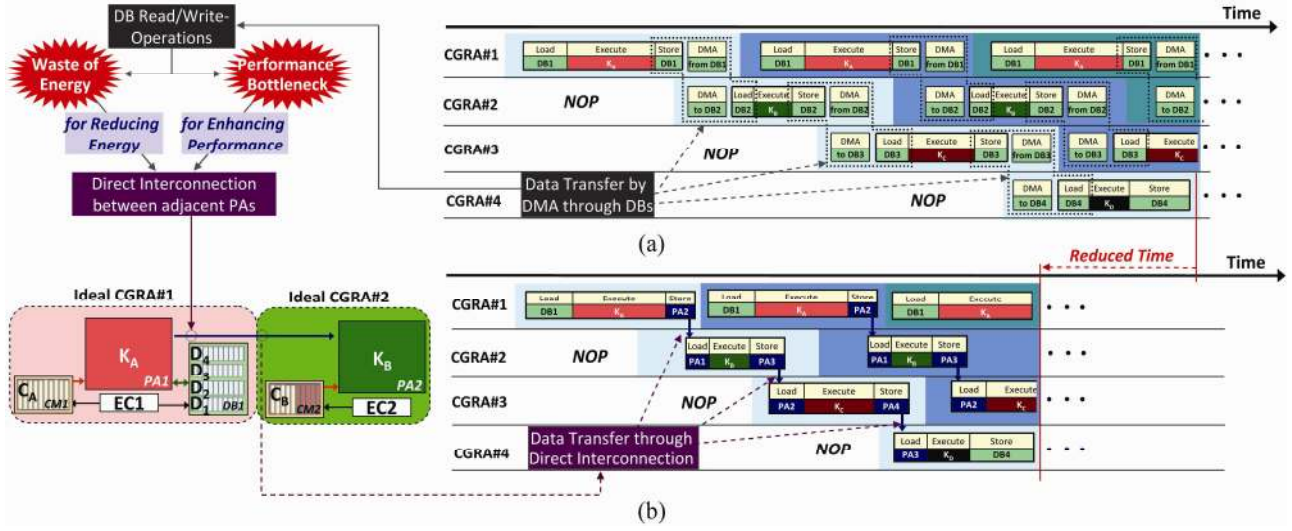


Fig. 5. Comparison between two cases of pipeline-scheduling for Example#1 (a) Pipeline-scheduling on the base multi-CGRA, (b) Pipeline-scheduling on the multi-CGRA with direct interconnections.

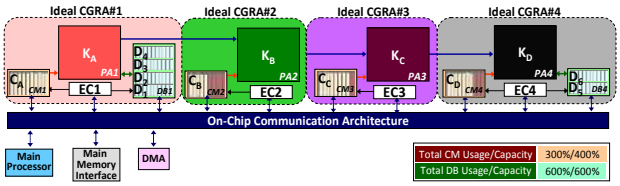
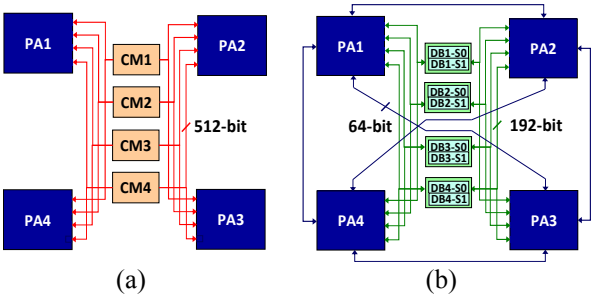


Fig. 6. Ideal multi-CGRA for Example#1.



(4x4 PA, 32-bit context word, and16-bit operand/result data)

Fig. 7. An example of completely connected fabric (CCF) (a) CM-PA Interconnections, (b) PA-PA/DB Interconnections.

total capacity.

2. Necessity of Inter-CGRA Reconfiguration

As mentioned in the previous section, the base multi-CGRA suffers from performance bottleneck and much energy when trying to achieve kernel-stream type of the KLP. This is because such a monotonous aggregation of several CGRAs cannot be flexible to support efficient resource utilization. We hypothesize that a multi-CGRA can support component-level (CM, DB or PA) inter-CGRA reconfiguration that means use of each component is not limited to a CGRA. Then the CGRA-based multi-core architecture can be optimized for its performance and energy because such an inter-CGRA reconfiguration can efficiently utilize component-level resources of the multi-CGRA in various cases of kernel-streams. In the next section, we propose a new multi-CGRA fabric and a novel reconfiguration technique that support the inter-CGRA reconfiguration.

V. Inter-CGRA Reconfiguration on Ring-based Sharing Fabric (RSF)

1. Design Objectives

We can easily consider a highly flexible fabric for inter-CGRA reconfiguration as Fig. 7. It shows the completely connected fabric (CCF) based on four CGRAs that seems as a good candidate to facilitate reconfigurable inter-CGRA – the CCF can enable any combination of mapping between all of the CMs (or DBs) and all of the PAs. However, such a full connectivity causes significant area and power overhead with increasing the number of CGRAs – as shown in the Fig. 7, the bit-width of the interconnections between the components is not small. On the other hand, only less connectivity may degrade the reconfigurability of inter-CGRA. Therefore, in Section V.2 through Section V.3, we propose ring-based sharing fabric and intra/inter-CGRA co-reconfiguration coming close to both two design objectives as follows:

- Design objective#1: The multi-CGRA fabric should show minimal interconnection overhead even though the number of CGRAs increases.
- Design objective#2: The multi-CGRA fabric should be as reconfigurable as CCF.

2. Ring-based Sharing Fabric (RSF)

The proposed multi-CGRA fabric based on four CGRAs is shown in Fig. 8 – it is called ring-based sharing fabric (RSF). The RSF connects all of the PAs through single-cycle interconnections and a DB (or a CM) is shared by two adjacent PAs on the RSF. Such connectivity fits in well with design objective#1 because the design overhead is only interconnections and switching logics between two adjacent PAs. Therefore, the overhead is trivial even though the number of CGRAs increases. The next subsection illustrates inter-CGRA reconfiguration on the RSF with the previous examples and suitability of RSF for design objective#2 is evaluated in Section V.2.B.

A. Example of inter-CGRA reconfiguration

Figs. 4 and 5 show that Example#1 causes the waste of energy and the performance bottleneck. However, the proposed RSF can be configured by inter-CGRA reconfiguration as Fig. 9 that is equivalent to the ideal multi-CGRA as Fig. 6. Therefore, the pipelining of the

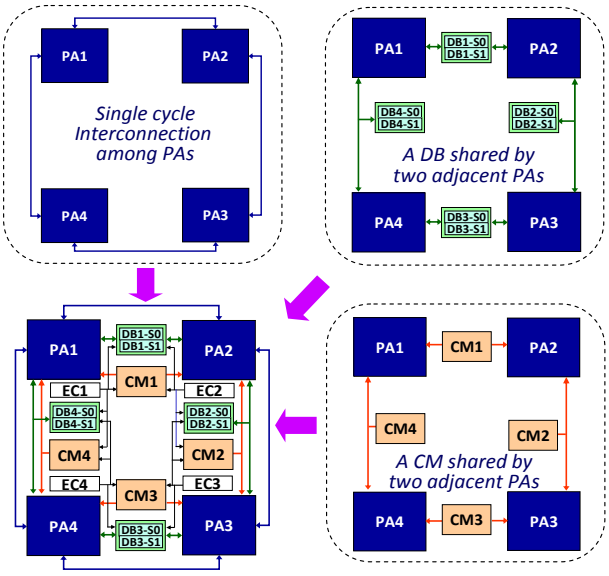


Fig. 8. Ring-based sharing fabric (RSF).

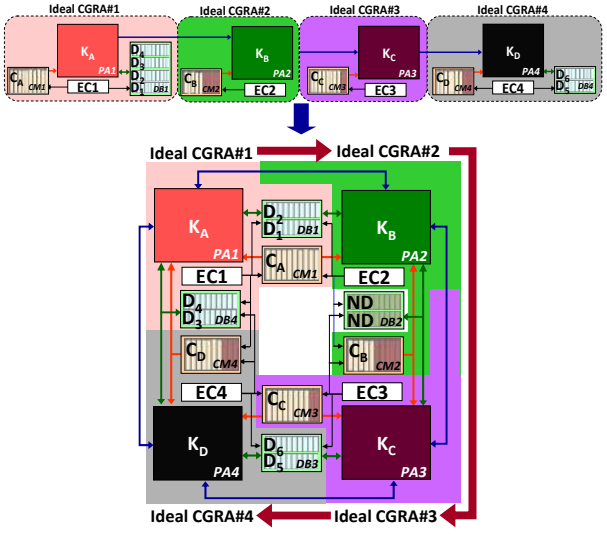


Fig. 9. Inter-CGRA reconfiguration for Example#1.

kernel-stream can be successfully mapped on the RSF with reducing energy and enhancing performance compared with the base multi-CGRA.

B. Suitability of RSF for design objective #2

The previous example of inter-CGRA reconfiguration shows very successful mapping cases on the RSF with the ideal resource utilization. This is because the PAs in Example#1 fortunately utilize 1 CM or two DBs at most - the RSF structurally allows that a PA can utilize up to two CMs or two DBs. However, if a PA requires more than three CMs or three DBs, the RSF seems to be far

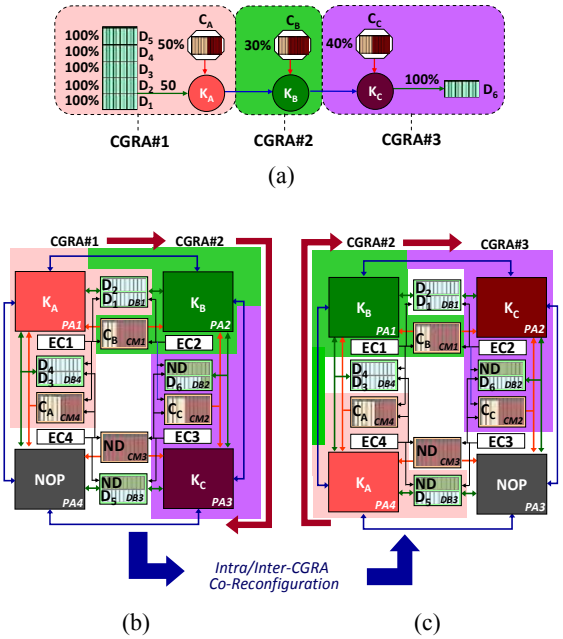


Fig. 10. Mapping the pipelining of the kernel-stream with three DBs on the RSF (a) Pipelining of kernel-stream with three DBs, (b) Configuration#1, (c) Configuration#2.

from design objective#2 – The CCF supports that any PA can utilize all of CMs and DBs on the fabric. Therefore, how to alleviate the structural limitation of the RSF is the key to coming close to design objective#2. In the next section, we propose such a key technique on the RSF for supporting efficient resource utilization.

3. Intra/inter-CGRA Co-Reconfiguration

Fig. 10(a) illustrates an example that the pipelining of kernel-stream requires three DBs (500%) and it iteratively runs on the RSF at 50 times – each data-set (100%) includes operand-data for the iterative running at 10 times. In this example, the lack of DB resources may be exposed on the RSF but we can alleviate the limitation of the RSF by shifting configuration of kernel-stream on multiple CGRAs. Fig. 10 illustrates how to exploit intra/inter-CGRA co-reconfiguration in order to achieve the shifting configuration. Before all, Fig. 10(b) shows initial configuration of the kernel-stream that PA1 utilizes DB1 (D_1 and D_2) and DB4 (D_3 and D_4) for the running of 40 iterations. Then the RSF can be configured as Fig. 10(c) that shows the utilization of one more DB (DB3) for the remaining 10 iterations. The utilization of DB3 (D_5) can be achieved by shifting the configurations

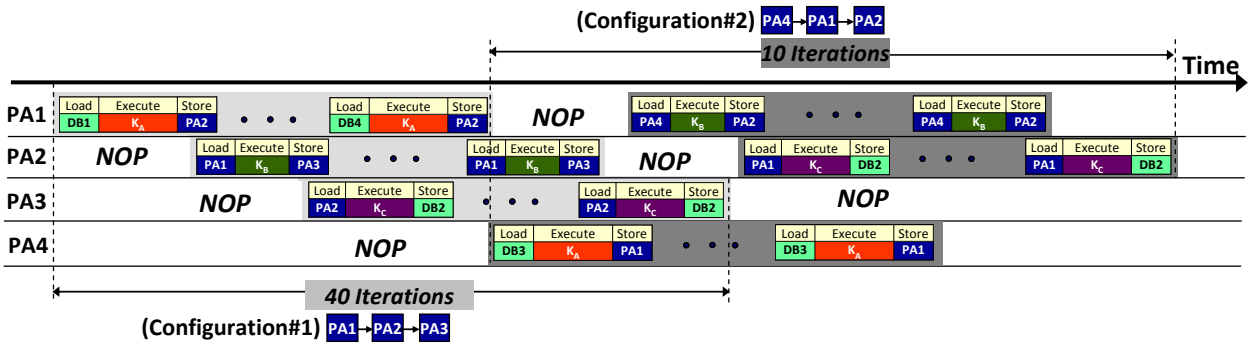


Fig. 11. Pipeline-scheduling on the RSF according to two cases of configurations.

of PAs from 'PA1->PA2->PA3' to 'PA4->PA1->PA2'. Therefore, the RSF operates as if a PA is connected with three DBs. In this case, the intra-CGRA reconfiguration means that PA1 and PA2 are reconfigured twice in order to perform K_A/K_B and K_B/K_C. On the other hand, the inter-CGRA reconfiguration enables that three CGRAs are configured with different number of CMs/DBs and connected through the direct interconnections. Such a co-reconfiguration can start immediately because each CM is shared by two adjacent PAs that are dynamically reconfigurable. It means that the pipelining of the kernel-stream continually runs on the RSF without stall as shown in Fig. 11.

Fig. 12(a) shows another example that the pipelining of kernel-stream requires four DBs (700%) and it iteratively runs on the RSF at 70 times. The almost identical way would apply here as in the previous mapping with three DBs (Fig. 11) but more elaborated co-reconfiguration is needed because shifting configuration of the kernel-stream should be sequentially performed twice in order to utilize two more DBs (DB2 and DB3).

Therefore, first of all, the CMs should be initialized like Fig. 12(b).Unlike the previous example, the CM4 and the CM2 include both data-set C_A and C_C. In addition, C_B is also stored in the CM3 as well as the CM1 whereas the CM3 is not used on the previous RSF. Then the Configuration #1 as Fig. 12(c) and the Configuration#2 as Fig. 12(d) work for the running of 40 iterations and 20 iterations in the same manner of the previous case with three DBs. Furthermore, the RSF is lastly configured as Fig. 12(e) that is made possible by the initialization of CMs as Fig. 12(b). The configuration#3 shows the utilization of DB2 (D_7) for the remaining 10 iterations. The utilization of DB2 can be achieved by the second shifting the configurations of PAs from 'PA4->PA1->PA2' to 'PA3->PA4->PA1'. Therefore, the RSF operates as if a PA is connected with four DBs. In addition, PA1 are reconfigured three times in order to perform $K_A/K_B/K_C$ but the capacity of two CMs (CM_1 and CM_4) is enough to support three different configurations as if PA1 is connected with three CMs. Finally, Fig. 12(f) shows the pipeline-scheduling of the kernel-stream with four DBs that continually runs on the RSF without performance degradation. In this way, intra/inter-CGRA co-reconfiguration can be exploited to map this example with up to three CMs and four DBs on the RSF.

VI. Inter-CGRA Control Mechanism on Ring-based Sharing Fabric (RSF)

The synchronization between adjacent PAs is essential for the pipelining of kernel-stream on the RSF. In addition, efficient DB/CM sharing-structure is necessary to support inter-CGRA reconfiguration. Therefore, in this section, we describe implementation details of EC, DB, and CM to show how to control the inter-CGRA operations on the RSF.

1. Synchronization between Adjacent PAs

As shown in Fig. 13, neighbor ECs are connected to each other as well as two adjacent DBs/CMs. Such connectivity enables data-transfer including timing information and control signal between the ECs - it's necessary for the synchronization between adjacent Pas for the pipelining of kernel-stream. Therefore, in the two following subsections, we show two cases of kernel-

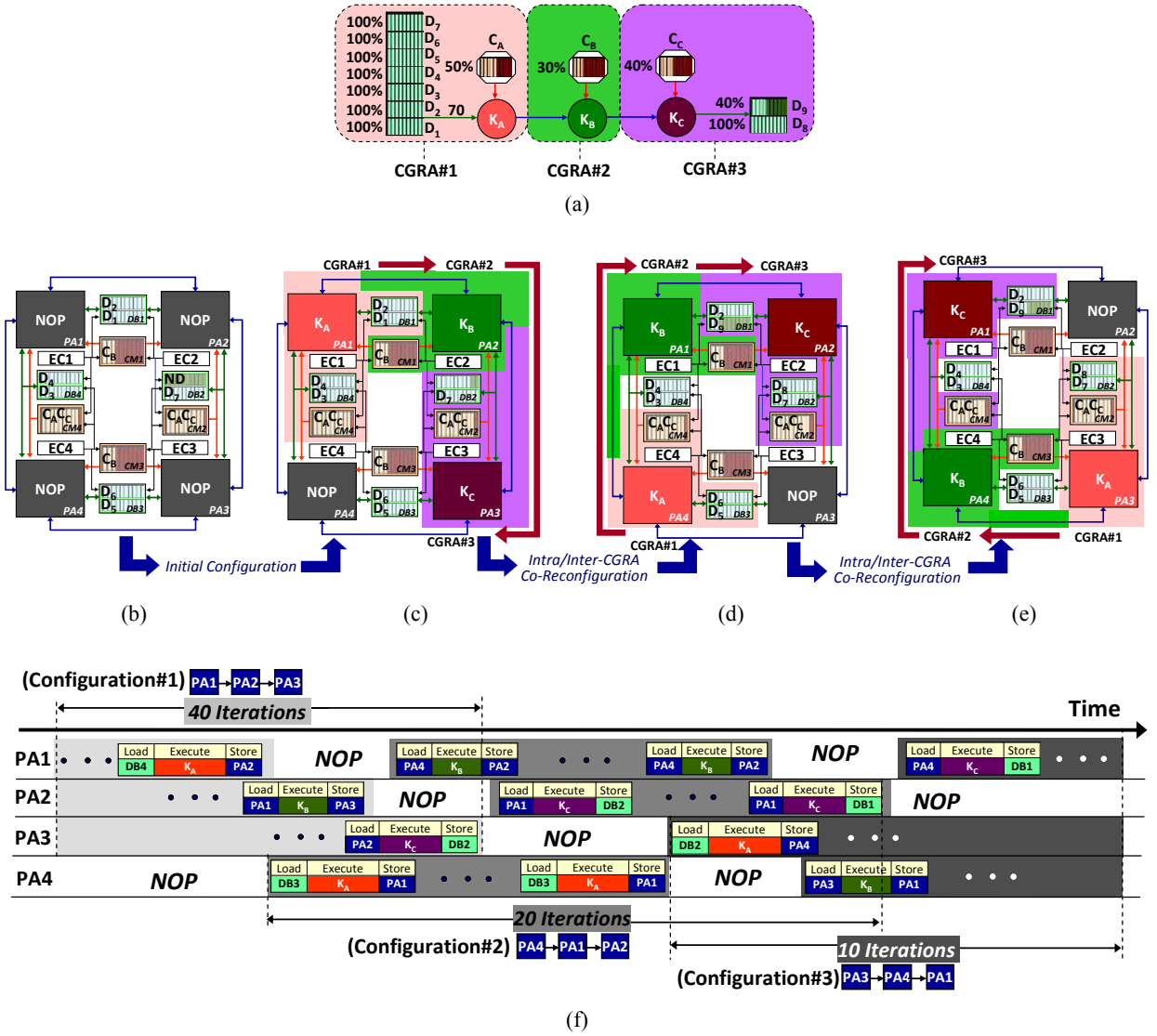


Fig. 12. Mapping the pipelining of the kernel-stream with four DBs on RSF (a) Pipelining of kernel-stream with four DBs, (b) Initialization, (c) Configuration#1, (d) Configuration#2, (e) Configuration#3, (f) Pipeline-scheduling.

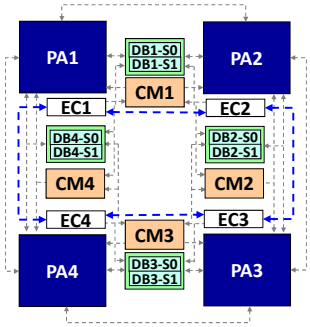


Fig. 13. Interconnection structure among ECs.

stream examples that illustrate why such data-transfer process is required for the synchronization between the PAs on the RSF.

A. Pipelining of kernel-stream with 2 DBs

In this subsection, we only consider a case of synchronization between adjacent PAs for the pipelining of kernel-stream with 2DBs – therefore, it means that the shifting configuration as Fig. 10 is unnecessary for this case. Fig. 14 shows such a pipeline-scheduling for Example#1 in Fig. 9. As shown in Fig. 14(b), each kernel (K_A~K_D) shows different execution time and kernel K_B running on PA2 takes the longest execution time among them. It means that PA1, PA3 and PA4 must wait for their preceding kernel-execution to be finished in the second iteration and over. Therefore, EC1, EC3 and EC4 must activate their own PAs after idle cycles for the synchronization.

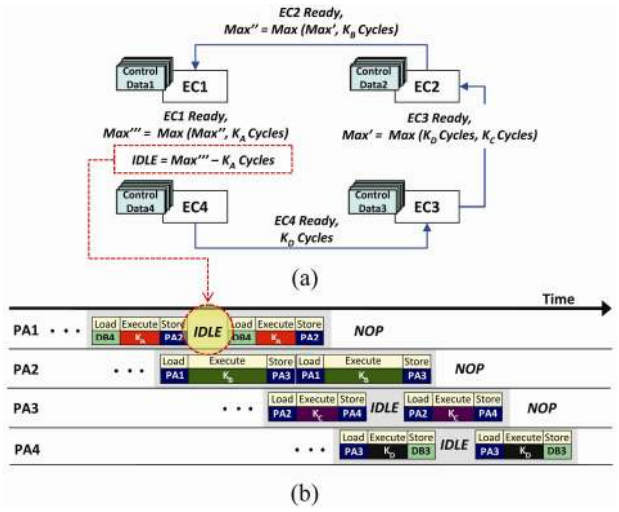


Fig. 14. Synchronization between adjacent PAs for the pipelining of kernel-stream with 2DBs (a) Calculation of '*IDLE*' cycles for PA1, (b) Pipeline-scheduling.

In the case of EC1, the '*IDLE*' cycles must be detected for the synchronized execution from PA1 to PA2. Fig. 14(a) shows such detection process by subtracting the kernel K_A cycles from the maximum execution cycles among K_A~K_D cycles – each kernel-cycle information is included in each control data (Control Data1 ~ Control Data4). The maximum execution cycles can be found by delivering the previous maximum cycles to the next EC in the reverse order of kernel-stream sequence − every EC checks whether one's own execution cycles are greater than the delivered one or not. Meanwhile, there is no need to detect '*IDLE*' cycle for EC3 and EC4 because these ECs are synchronized by '*Intermediate Done*' signal from the previous EC– PA3 and PA4 are not activated until their ECs receive this signal.

Fig. 15 shows every possible flow of the control signals among 4 ECs. Every EC may send or receive '*Intermediate Done*' signal because any EC can play a role of head, body or tail when a kernel-stream is mapped onto the RSF. Therefore, it is necessary that each EC is initialized with the control data specifying one's own role for running the kernel-stream on the RSF. Fig. 16 shows the field layout of such a control data. '*Partner*' field specifies the previous/next EC which sends or receives the control signal to/from the current EC. In addition, '*Sender*' and '*Receiver*' field are ce – every EC checks whether one's own execution cycles are greater than the delivered one or not. Meanwhile, used for defining the

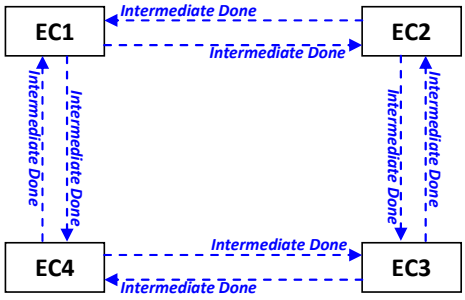


Fig. 15. Flow of '*Intermediate Done*' signals among 4 ECs.

Partner	Sender	Receiver	Head	Tail	DB_Sel	CM_Sel

Fig. 16. Control information for the synchronization between adjacent PAs.

role of the current EC. '*Head*' or '*Tail*' field mean whether the current EC is the starting part or the ending part of the kernel stream. The last two fields of '*DB_Sel*'/'*CM_Sel*' point out which CM/DB are preferentially used between two CMs/two DBs connected to the current EC.

B. Pipelining of kernel-stream with 3 DBs and over

If we consider the pipelining of kernel-stream with 3DBs and over on the RSF, the shifting configuration as Fig. 10 is necessary. It means that the flow of '*Intermediate Done*' signals should be changed immediately in order to achieve the shifting configuration. Fig. 17 shows such a kernel-stream with the shifting configuration as Fig. 11 - it iteratively runs on the RSF at 50 times as Fig. 17(a). In the case of the first 40 iterations, the flow of '*Intermediate Done*' signals as Fig. 17(b) enables Configuration#1 in the same manner of the example with two DBs in the previous subsection. However, for running the remaining 10 iterations, the shifting configuration (from Configuration#1 to Configuration#2) should occur on the RSF and it can be achieved by changing the signal flow from Figs. 17(b)-(d) – EC4 is activated by '*Intermediate Done*' signal from EC1 as Fig. 17(c).

2. Inter-CGRA Interconnection Structure

A. PE Array (PA)

The proposed synchronization method enables direct data-transfer between adjacent PAs. As shown in Fig. 18,

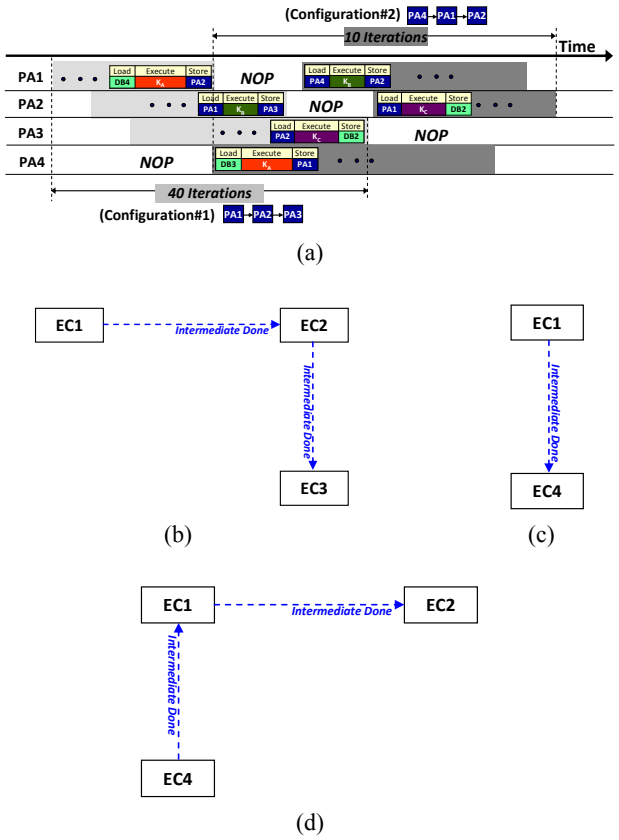


Fig. 17. Flow of '*Intermediate Done*' signals when Mapping the kernel-stream with three DBs (a) Pipeline-scheduling, (b) The signal flow for Configuration#1, (c) Activation of EC4 for shifting configuration, (d) The signal flow for Configuration#2.

each PA is connected with the neighbor two PAs as mesh-based structure in order to support efficient column-wise or row-wise direct data-transfer.

B. Configuration Memory (CM)

Fig. 19 shows the interconnection structure among adjacent ECs and CMs. It enables a CM to be shared by two adjacent PAs on the RSF. Therefore, the multiplexer (PAn INPUT MUX) is necessary for each EC to select one of two CMs for running PA. In addition, each CM controller can be activated by either one of two adjacent ECs.

C. Data Buffer (DB)

Fig. 20 shows the interconnection structure among adjacent ECs and DBs with the inner structure of a DB. The interconnection structure enables a DB to be shared by two adjacent PAs on the RSF likewise with CM - data-input ports of PAs are connected to the multiplexer (PAn INPUT MUX) for one's own EC to select one of two DBs. Meanwhile, in the viewpoint of intra-structure, a DB has two sets of buffers, each having three banks: one bank connected to the write bus and the other two banks connected to the read buses. The two-set structure facilitates simultaneous access to a DB from two adjacent PAs as well as the overlap of data-transfer with computation. Fig. 21 shows the inner structure of DB controller to enable such a DB-access. Each EC can operate one of two DB-sets at the same time and any combination of one-to-one mapping between the two ECs and the two DB-sets is possible.

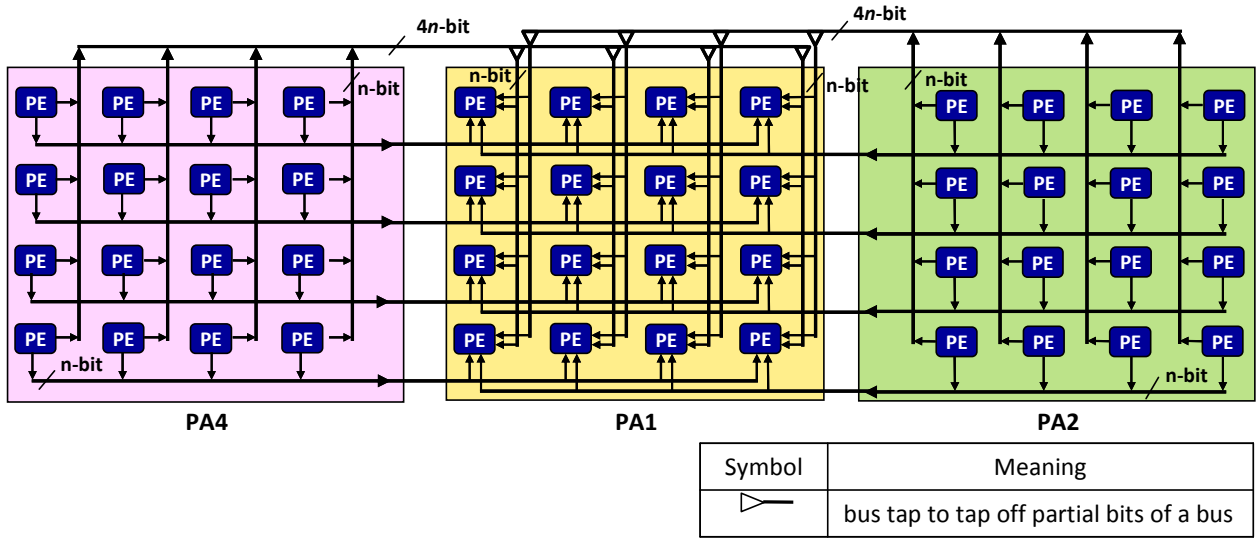


Fig. 18. Direct interconnection between adjacent Pas.

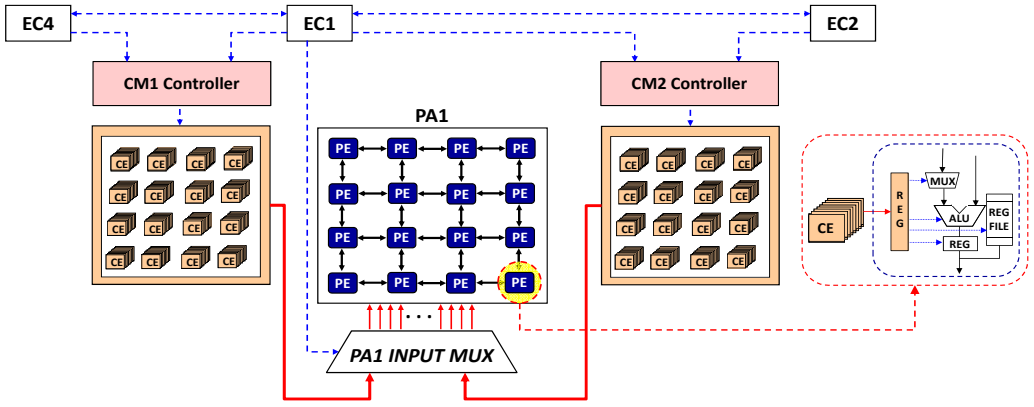


Fig. 19. Interconnection structure among adjacent ECs and CMs.

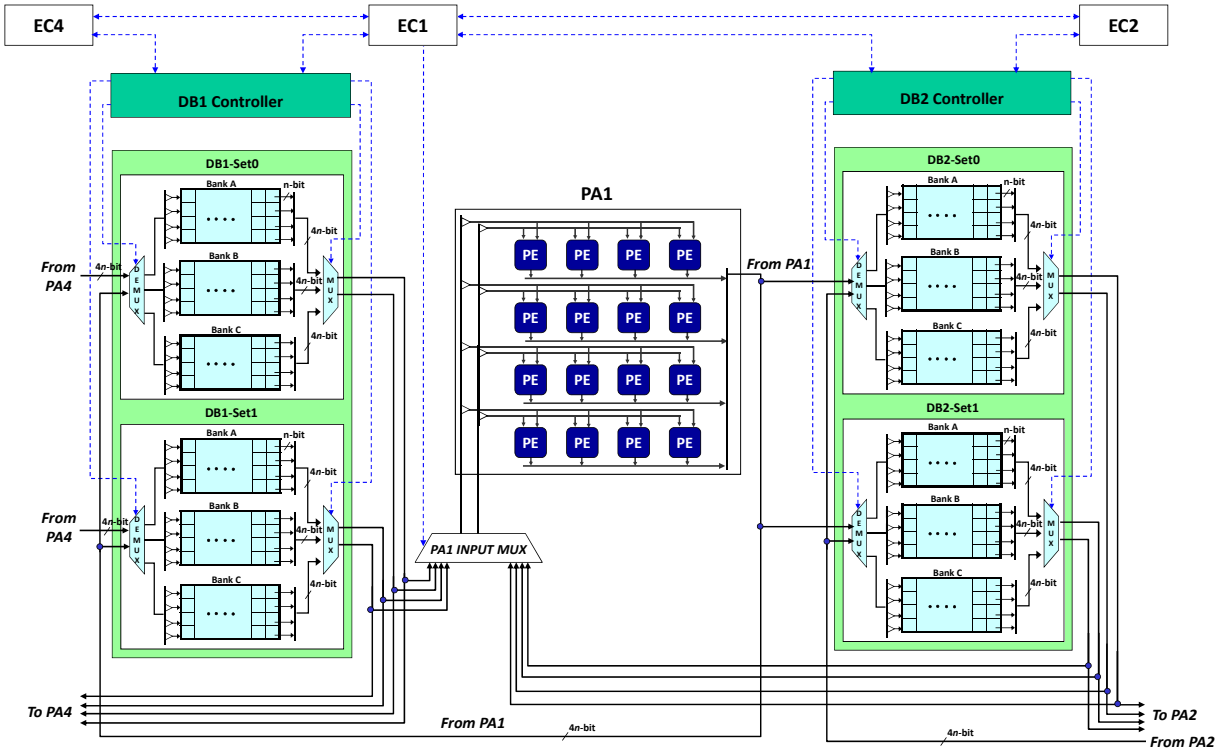


Fig. 20. Interconnection structure among adjacent ECs and DBs.

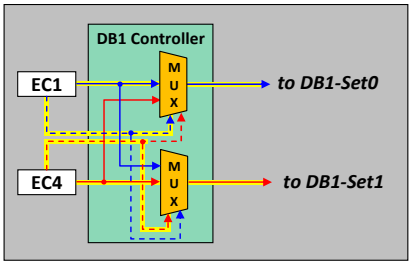


Fig. 21. Inner structure of DB controller with two adjacent ECs.

VI. EXPERIMENTS AND RESULTS

1. Architecture Implementation

To demonstrate the quantitative effectiveness of the proposed approaches, we have implemented three different organizations of multi-CGRA with variation in the number of CGRAs as shown in Table 2 – 4, 8, 12, and 16 CGRAs that include the identical CGRAs specified in Table 1. We have designed them at RT-level (RTL) using Verilog HDL and synthesized gate-level circuits using Design Compiler [11] with 90 nm generic library [11] to analyze hardware cost.

2. RTL-synthesis Results

A. Area evaluation

Table 3 shows area cost evaluation for the three cases of multi- CGRA with increasing the number of CGRAs. In the case of 4 CGRAs, the area costs of the CCF and the RSF have only increased by 10% and 8% respectively compared with the BASE. However, as the number of CGRAs continues to increase from 8 to 16, the area of the CCFs significantly increases by 19%~42% due to its heavy interconnections and switching logics.

Table 1. Single CGRA implementation at RT-level with Verilog

Components	Parameters	Value
Processing Element (PE) Array	Bit-Width of Registers in a PE	16-bit
	Number of Registers in a PE	4
	Number of PEs	4x4(16)
Configuration Memory (4KB)	Bit-Width of a Configuration Element (CE)	32-bit
	Number of Layers for a CE	64
	Number of CEs	4x4(16)
Data Buffer (1.5KB)	Number of Sets	2
	Number of Banks in a Set	3
	Bit-Width of a Bank (Dual-Port A/B)	32/64-bit
	Number of Layers for a Bank (Dual-Port A/B)	64/32

Table 2. Multi-CGRA implementation at RT-level with Verilog

Architecture	Number of CGRAs
BASE(Only Bus-Connected)	4, 8, 12, 16
Completely Connected Fabric (CCF)	
Ring-Shaped Fabric (RSF)	

Table 3. Area comparison

No'[1]	Arch'	Gate Equivalent			Increased[5] (%)		
		Net[2]	Logic[3]	Total[4]	Net	Logic	Total
4	BASE	311,216	5,813,381	6,124,597	-	-	-
	CCF	562,318	6,167,122	6,729,440	81	6	**10**
	RSF	491,591	6,094,900	6,586,491	58	5	**8**
8	BASE	624,679	11,626,367	12,251,046	-	-	-
	CCF	1,585,952	13,016,128	14,602,080	154	12	**19**
	RSF	1,117,076	12,510,034	13,627,110	79	8	**11**
12	BASE	939,966	17,438,521	18,378,487	-	-	-
	CCF	3,274,543	20,465,091	23,739,634	248	17	**29**
	RSF	1,940,685	19,226,564	21,167,249	106	10	**15**
16	BASE	1,254,581	23,250,721	24,505,302	-	-	-
	CCF	5,706,329	28,993,032	34,699,361	355	25	**42**
	RSF	2,896,255	26,275,472	29,171,727	131	13	**19**

No'[1]: Number of CGRAs, **Net[2]**: Net interconnect area**, Logic[3]**: Total cell area, **Total[4]**: Net[1]+Logic[2], **Increased[5]**: Increase rate of area compared with BASE, ((CCF or RSF)/BASE–1)X100

Table 4. Critical path delay comparison

No' [1] / Arch'	4		8		12		16	
	D[2](ns)	Inc[3](%)	D[2](ns)	Inc[3](%)	D[2](ns)	Inc[3](%)	D[2](ns)	Inc[3](%)
BASE	3.4	-	3.4	-	3.4	-	3.4	-
CCF	3.88	**14.12**	3.89	**14.41**	3.99	**17.35**	4.26	**25.29**
RSF	3.74	**10**	3.74	**10**	3.74	**10**	3.74	**10**

No'[1]: Number of CGRAs**, D[2]**: Critical path delay**, Inc[3]**: Increase rate of delay compared with BASE, ((CCF or RSF)/BASE–1)X100

On the other hand, the RSFs have gradually increased the area by 11%~19% because the interconnections and switching logics between two adjacent PAs are only added according to increasing the number of CGRAs. Therefore, the proposed RSF is more area-efficient fabric compared with the CCF.

B. Delay evaluation

Table 4 shows the comparison of critical path delay in the three cases of multi-CGRA varying the number of CGRAs. In the case of the CCF, the critical path delay has considerably increased by 14.12%~25.29% compared with the BASE. This is because more complex switching logics enabling the full connectivity are included in the set of critical paths of the CCF when the number of CGRAs increases. However, the RSFs show the same increase rate (10%) of delay regardless of the number of CGRAs because only adding CGRAs with keeping ring-shape does not affect the critical path delay. Therefore, the proposed RSF is more efficient than the CCF in terms of the critical path delay.

C. Power evaluation

We have evaluated the power consumption of the three cases of multi-CGRA with increasing the number of CGRAs as shown in Table 5. First of all, both the CCF and the RSF including 4 CGRAs show insignificant increase rate (10.68% and 5.11%) of power compared with the BASE. However, the power of the CCFs with more CGRAs (8~16) seriously increases by 91.72%~97.70% because of its huge interconnections and switching logics. Mean while, the RSFs with more CGRAs (8~16) show the increase rate of power ranging from 7.31% to 21.73% because relatively fewer interconnections and switching logics are added according to increasing the number of CGRAs. Therefore, the proposed RSF is more power-efficient fabric compared with the CCF.

3. Performance/energy Evaluation

Table 6 shows that the test benches of kernel-streams that are classified by two criteria – The first criterion is the number of interdependent kernels and the second one is the number of utilized DBs. The first criterion is for evaluating the pipelining of the kernel-streams on the three cases of multi-CGRA with varying number of CGRAs – we assume that the pipelining of the kernel-streams runs on the multi-CGRA whose number of CGRAs is equal to the number of the kernels. The second criterion subdivides the four cases of kernel-streams into more cases that require different number of DBs – in this case, a DB includes operand-data for the iterative running at 16 times. Therefore, we can evaluate how inter-CGRA reconfiguration on the RSF works well for the kernel-streams that require two DBs ~ the most DBs. These test benches consist of several DSP algorithms in order to fully utilize arithmetic and storage resources in PAs.

Table 5. Power comparison

No'[1]	Arch'	Gate Equivalent			Increased[5] (%)		
		Net[2]	Logic[3]	Total[4]	Net	Logic	Total
4	BASE	0.5463	0.2263	0.7726	-	-	-
	CCF	0.5911	0.2640	0.8551	8.20	16.66	**10.68**
	RSF	0.5578	0.2543	0.8121	2.11	12.37	**5.11**
8	BASE	1.0307	0.3377	1.3684	-	-	-
	CCF	1.8859	0.7586	2.6445	82.97	124.64	**93.26**
	RSF	1.1212	0.4287	1.5499	8.78	26.95	**13.26**
12	BASE	1.5123	0.4595	1.9718	-	-	-
	CCF	2.8085	1.0898	3.8983	85.71	137.17	**97.70**
	RSF	1.6698	0.7305	2.4003	10.42	58.98	**21.73**
16	BASE	2.0252	0.6400	2.6652	-	-	-
	CCF	3.8230	1.2867	5.1097	88.77	101.05	**91.72**
	RSF	2.0955	0.7644	2.8599	3.47	19.44	**7.31**

No'[1]: Number of CGRAs, **Net[2]**: Net switching power, **Logic[3]**: Cell internal power, **Total[4]**: Net[1]+Logic[2], **Increased[5]**: Increase rate of power compared with BASE, ((CCF or RSF)/BASE–1)X100

Table 6. Kernel-Streams Characteristics

No' of Kernels	Pipelining of Kernel Streams with Increasing Number of Utilized DBs (Number of Iterations)				
	2 DBs (32 Iterations)	4 DBs (64 Iterations)	8 DBs (128 Iterations)	12 DBs (192 Iterations)	16 DBs (256 Iterations)
4	KS4T1	KS4T2	-	-	-
8	KS8T1	KS8T2	KS8T3	-	-
12	KS12T1	KS12T2	KS12T3	KS12T4	-
16	KS16T1	KS16T2	KS16T3	KS16T4	KS16T5

We have evaluated performance of the test benches kernel-streams running on the three cases of multi-CGRA with increasing the number of CGRAs as Fig. 22. In all cases of test benches, the CCF and the RSF are much faster than the BASE because they allows direct data-transfer without DB read/write operations passing through on-chip communication architecture. In addition, the CCF and the RSF show much higher performance improvement when the number of utilized DBs/CGRAs increases. It means that inter-CGRA reconfiguration technique really comes into its own when more DBs are utilized on more CGRAs. In addition, the RSF is a little bit faster than the CCF in all cases because shorter critical path delay of the RSF more than makes up for slightly increased execution cycles on the RSF caused by inter-CGRA reconfiguration. It also means that the pipelining of the kernel-streams with more than four DBs continually runs on the RSF by the shifting configuration without performance degradation.

Fig. 23[1] shows the energy saving of the test benches running on the RSF compared with the CCF with varying the number of CGRAs. First of all, the RSF including 4 CGRAs show modest energy saving (8.39% and 7.26%) compared with the CCF as Fig. 14(a) because stark differences between two fabrics are elusive under 4 CGRAs. However, the energy saving on the RSFs with

[1] We have omitted the BASE in Fig. 14 in order to clarify energy difference between the CCF and the RSF. By the way, in this case, the CCF and the RSF reduce the energy by 79~97% compared with the BASE.

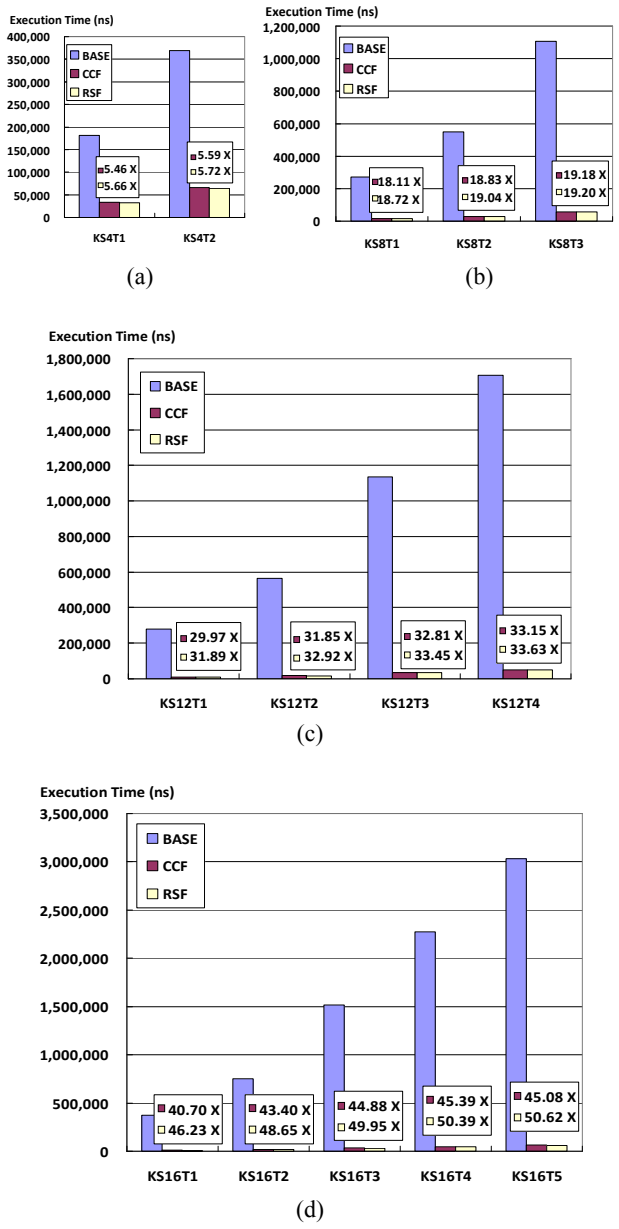


Fig. 22. Performance comparison (a) 4 CGRAs, (b) 8 CGRAs, (c) 12 CGRAs, (d) 16 CGRAs.

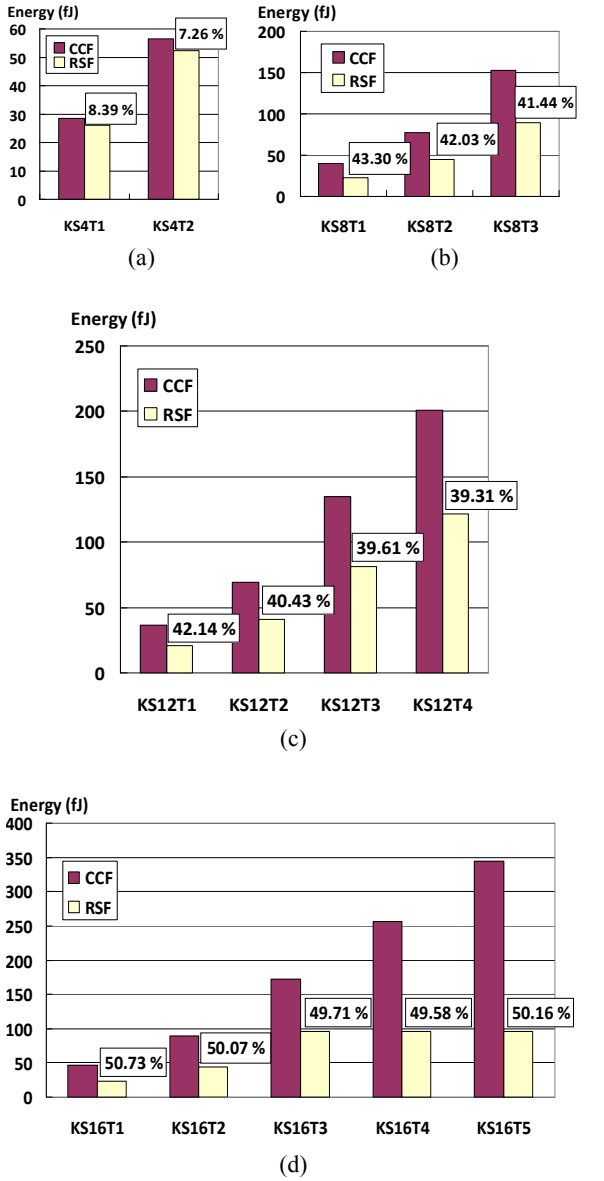


Fig. 23. Energy comparison (a) 4 CGRAs, (b) 8 CGRAs, (c) 12 CGRAs, (d) 16 CGRAs.

more CGRAs (8~16) increases by much (39.31%~50.73%) because the CCFs with more CGRAs (8~16) consist of huge interconnections and complex switching logics whereas relatively fewer interconnections and switching logics are added on the RSFs. Therefore, the proposed RSF is more energy-efficient fabric compared with the CCF when running pipelining of kernel-stream.

VIII. CONCLUSIONS

Coarse-grained reconfigurable architecture (CGRA) has emerged as a suitable solution for embedded systems but there is a limit when a CGRA is expected to improve the performance of an entire application. This is because single CGRA is sequentially optimized for the parallelized computations in a kernel at a time whereas the overall speedup of the entire application can be achieved by kernel level parallelism (KLP) that several kernels concurrently run. Therefore, CGRA-based multi-core architectures have appeared to support diverse KLPs.

However, the existing CGRA-based multi-core architectures suffer from much energy and performance bottleneck because the existing multi-CGRA structures are not flexible enough to adaptively support various cases of the KLP. It means that the resources in the multi-CGRAs cannot be efficiently utilized under monotonous aggregation of several CGRAs. To overcome the limitations, we have proposed the new ring-based sharing fabric (RSF) for improving the flexibility level of the CGRA-based multi-core architectures focusing on the kernel-stream type of the KLP. In addition, the novel inter-CGRA reconfiguration technique based on the RSF has been introduced for efficient pipelining of kernel-stream. Experimental results show that the proposed approaches improve performance by up to 50.62 times and reduce energy by up to 50.16% when compared with the existing architecture model.

REFERENCES

[1] Reiner Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," in *Proc. of Design Automation and Test in Europe Conf.*, pp. 642-649, Mar. 2001.

[2] Aaron Wood, Adam Knight, Benjamin Ylvisaker, and Scott Hauck, "Multi-kernel floorplanning for enhanced CGRAs," in *Proc. of IEEE Int. Conf. on Field-Programmable Logic and Application (FPL),* pp. 157-164, Aug. 2012

[3] Minsoo Kim, Joon Ho Song, Do-Hyung Kim, and Shihwa Lee, "Hybrid Partitioned H.264 Full High Definition Decoder on Embedded Quad-core," in *Proc. of IEEE Int. Conf. on Consumer Electronics (ICCE),* pp. 279-280, Jan 2012.

[4] Kosuke Nishihara, Atsushi Hatabu, and Tatsuji Moriyoshi, "Parallelization of H.264 video decoder for embedded multicore processor," in *Proc. of IEEE Int. Conf. on Multimedia and Expo,* pp. 329-332, April 2008.

[5] Minsoo Kim, Joonho Song, Dohyung Kim, and Shihwa Lee, "H.264 decoder on embedded dual core with dynamically load-balanced functional partitioning," in *Proc. of IEEE Int. Conf. on Image Processing (ICIP),* pp. 3749-3752, Sept 2010.

[6] Seunghun Jin, Sang-Heon Lee, Moo-Kyoung Chung, Yeon-Gon Cho, and Soojung Ryu, "Implementation of a Volume Rendering on Coarse-grained Reconfigurable Multiprocessor," in *Proc. of IEEE Int. Conf. on Field-Programmable Technology (FPT),* pp. 243-246, Dec. 2012.

[7] Navneet Basutkar, Ho Yang, Peng Xue, Kitaek Bae, and Young-Hwan Park, "Software-Defined DVB-T2 Receiver Using Coarse-Grained Reconfigurable Array Processors," in *Proc. of IEEE Int. Conf. on Consumer Electronics (ICCE),* pp. 580-581, Jan. 2013.

[8] Karel H. G. Walters, André B. J. Kokkeler, Sabih H. Gerez, and Gerard J. M. Smit, "Low-Complexity Hyperspectral Image Compression on a Multi-tiled Architecture," in *Proc. of IEEE NASA/ESA Conf. on Adaptive Hardware and Systems,* pp. 330-335, July. 2009.

[9] Haitao Wei, Junqing Yu, Huafei Yu, Mingkang Qin, and Guang R. Gao, "Software Pipelining for Stream Programs on Resource Constrained Multicore Architectures," *IEEE Trans. on Parallel and Distributed Systems,* vol. 23, no. 12, pp. 2338-2350, Dec. 2012.

[10] Wei Han, Ying Yi, Mark Muir, Ioannis Nousias, Tughrul Arslan, and Ahmet T. Erdogan, "Multicore Architectures with Dynamically Reconfigurable Array Processors for Wireless Broadband Technologies," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* vol. 28, no. 12, pp. 1830-1843, Dec. 2009.

[11] http://www.synopsys.com

Yoonjin Kim received the B.S. degree in information and communication engineering from Sungkyunkwan University, Seoul, South Korea, in 2003, the M.S. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2005, and the Ph.D. degree in computer engineering from Texas A&M University, College Station, in 2009. From 2009 to 2010, he was a Senior R&D Staff Member with the Samsung Advanced Institute of Technology (SAIT), Gyeonggi, South Korea. In 2010, he joined the faculty of the Department of Computer Science at Sookmyung Women's University in Seoul, South Korea. His research interests include embedded systems, computer architecture, VLSI/system-on-chip design, and hardware/software co-design.

Heesun Kim received the B.S. degree and the M.S. degree in computer science from Sookmyung Women's University, Seoul, South Korea, in 2012 and 2014. She is currently a R&D Staff Member with the Dongbu HiTek, Gyeonggi, South Korea. Her research interests include embedded systems and reconfigurable computing.