RF-CGRA: A Routing-Friendly CGRA with Hierarchical Register Chains

Rong Zhu, Bo Wang, Dajiang Liu*
College of Computer Science, Chongqing University, Chongqing 400044, China
{zhur,wangbocs,liudj}@cqu.edu.cn
*Corresponding author

***Abstract*—CGRAs are promising architectures to accelerate domain-specific applications as they combine high energy-efficiency and flexibility. With either isolated register files (RFs) or link-consuming distributed registers in each processing element (PE), existing CGRAs are all not friendly to data routing for data-flow graphs (DFGs) with a high edge/node ratio since there are many multi-cycle dependences. To this end, this paper proposes a Routing-Friendly CGRA (RF-CGRA) where hierarchical (intra-PE or inter-PE) register chains could be flexibly (wide range of chain length) and compactly (consuming fewer links among PEs) achieved for data routing, resulting in a new mapping problem that requires the improvement of a compiler. Experimental results show that RF-CGRA gets 1.19× performance and 1.14× energy efficiency of the state-of-the-art CGRA with single-cycle multi-hop connections (HyCUBE) while keeping a moderate compilation time.**

***Index Terms*—CGRAs, Modulo Scheduling, Data Routing**

I. INTRODUCTION

Coarse-Grained Reconfigurable Architectures (CGRAs) are a promising class of architectures due to their high performance and high power efficiency advantages. As shown in Fig. 1(a), a CGRA is typically composed of a processing element (PE) array, a data memory, and a configuration memory, where the PE array is composed of N × N PEs interconnected in a mesh topology. Each PE includes an arithmetic logic unit (ALU)-like function unit (FU), a local register file (RF), a result register, and some configuration registers. The FU can execute various fixed-point operations, including addition, subtraction, multiplication, etc. With fast context switching in the configuration registers of each PE, the functionality of FU and the connectivity style among PEs could be dynamically configured. By efficiently deploying hardware resources, CGRA enables many highly energy-efficient solutions for a wide range of applications.

With inherent parallelism in CGRA, the computation-intensive loop kernels are often mapped onto CGRAs for acceleration. To improve the mapping performance, modulo scheduling [1] is widely used to map a Data-Flow Graph (DFG) of a loop onto a time-extended CGRA resource graph, where the initiation interval (II) between adjacent loop iterations is the key metric of execution performance. In modulo scheduling, as the DFG is executed in a pipelined fashion, multi-cycle dependences need to be carefully routed such that different operand values of a target operation could arrive simultaneously to satisfy the timing requirement of pipelining. As the dependences may vary widely in amount and length, while the hardware resources for routing in CGRA are usually local and sparse, the routing of multi-cycle dependencies is a challenging task.

This work was supported by the National Natural Science Foundation of China under Grant 61804017.

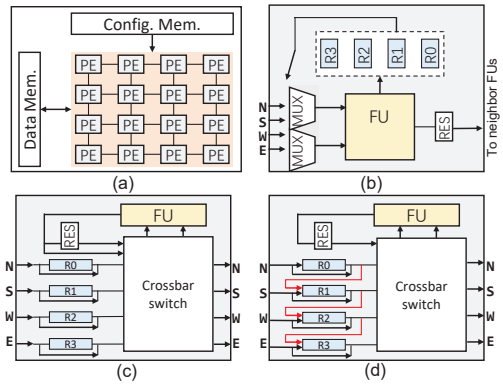

Fig. 1. CGRA architecture comparison. (a) A CGRA with 4× 4 PE array, (b) typical PE structure of ADRES [2], (c) PE structure of HyCUBE [3] and (d) our proposed PE structure.

To efficiently route and synchronize these multi-cycle (long) dependences, several approaches have been proposed. In the design space explorations of ADRES [2], as shown in Fig. 1(b), the RF is used to retain data in the current PE, which could be consumed by the FU within the PE or could even be consumed by the FUs of neighbor PEs [4] in future cycles. As FU may write or read different registers in the RF, the addressing of the RF adds extra configuration costs. Moreover, as the RFs in different PEs are isolated by FUs, they can hardly work in a chain way to route a dependence with the length greater than the RF size. Although the PE could bypass the FU to the result register (RES) such that the RFs in different PEs could be connected, the routing PE can not perform any other operations, leading to low resource utilization and poor pipelining performance. In HyCUBE [3], as shown in Fig. 1(c), the central RF in a PE is eliminated and the registers are moved directly into the incoming wires from each direction North, East, West, and South (N, E, W, S). Consequently, a long dependence could be routed by these distributed registers without RF addressing overhead. However, as registers are put on the input links of PEs, it is link-consuming for HyCUBE

to route multi-cycle dependencies. Moreover, as register latch and single-cycle hop are mutually exclusive in HyCUBE, there will be even fewer available registers for multi-cycle data routing if single-cycle hops are used. Consequently, HyCUBE is not friendly for mapping DFGs with a high edge/node ratio. Besides register routing, a long dependence can also be routed via data memory [5] by cutting it off and adding a pair of load-store operations. However, extra memory accessing operations may easily introduce access conflicts, leading to poor pipelining performance.

From previous works, we note that, with either isolated RFs or link-consuming distributed registers, existing CGRAs are all not friendly to data routing for long dependences, which frequently appear in DFG with a high edge/node ratio. To this end, as shown in Fig. 1(d), this paper proposes a Routing-Friendly CGRA (RF-CGRA) where hierarchical (intra-PE or inter-PE) register chains can be flexibly (wide range of clock cycles) and compactly (traversing fewer links among PEs) achieved for data routing. Our contributions are summarized as follows:

- We propose an energy-efficient Routing-Friendly CGRA, RF-CGRA, where intra-PE or inter-PE registers can work in a chain way. With the segment of register chain within a PE, as the red arrows shown in Fig. 1(d), data could be retained for multiple cycles without RF addressing overhead and PE link cost. With inter-PE register connection, registers among different PEs could be chained together to form a longer one such that long dependence could be easily routed and synchronized.
- We formulate the new mapping problem on RF-CGRA by adding inter-register edges to the modulo routing resource graph (MRRG [3]) with single-cycle multi-hop connection. With these extra paths, our mapping algorithm can find solutions with II near the minimal one for DFGs, especially for those with a high edge/node ratio.

II. Motivating Example

We illustrate the advantage of our proposed architecture with a motivation example. Fig. 2(a), (b), and (c) present the target mesh 2×2 CGRA with 8 links among PEs, the input loop with two references of the array x and the corresponding DFG with 2 load operations. Obviously, the original DFG has the opportunity of data reuse, and one load operation could be eliminated by adding a loop carried dependence (the blue arrow) with distance dif=4, as shown in Fig. 2(d). As the new DFG has only 4 operations while the target CGRA has 4 PEs, resource-constrained minimal II (ResMII) of 1 in modulo scheduling could be achieved. In that case, the length of the loop carried dependence is actually $t(b) - t(L_1) + II \times dif$ = 6, where t indicates the time step of an operation.

When mapping the new DFG onto traditional CGRA (ADRES-like) with II=1, as shown in Fig. 2(e), the 4 nodes (L_1, a, b and S_1) in each iteration are mapped onto PE_0, PE_1 and PE_2 and PE_3, respectively, where the blue, green, orange, yellow and light blue nodes indicate operations from iteration i, $i+1$, $i+2$, $i+3$ and $i+4$, respectively. To route the

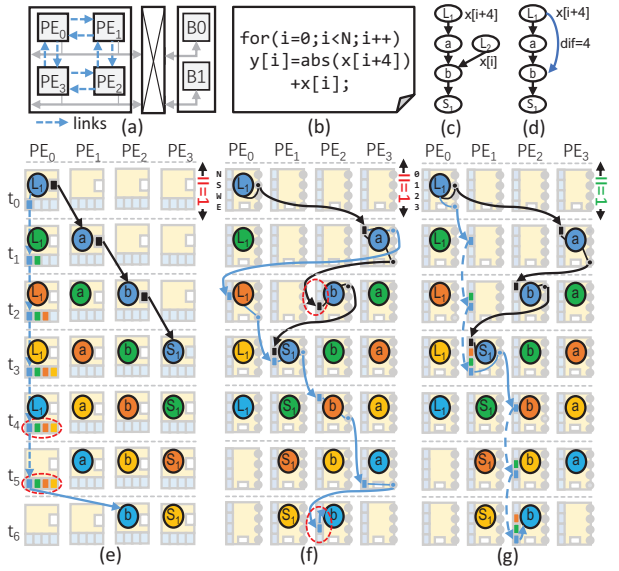

Fig. 2. A motivation example. (a) A 2×2 template CGRA with 8 links among PEs, (b) the input loop with two references of array x, (c) the DFG of the loop, (d) the new DFG with loop carried dependence after data-reuse, (e) failed mapping of the DDG on traditional CGRA with II=1, (f) failed mapping of the DDG on HyCUBE with II=1, and (g) successful mapping of the DDG on our proposed CGRA with II=1.

long dependence from L_1 at iteration i to b at iteration $i+4$, operation L_1 at each iteration would write loading data to the RF with different addresses, as the colored squares shown in Fig. 2(e). As the consumer operation (b at t_6) is far away and the RF size is limited, the RF will become full at t_3 such that later loading data cannot be written to the RF anymore. Consequently, it is failed to map the DDG onto traditional CGRA with II=1. Although a routing node could be inserted into the DFG to relieve the pressure of RF, it will also increase the ResMII.

When mapping the DDG onto HyCUBE with II=1, as shown in Fig. 2(f), the 4 light-blue squares on the left of PE indicate distributed registers in N, S, W, E directions, respectively, and the 4 dots on the right of PE indicate the output ports to PEs in N, S, W, E directions, respectively. After routing of the long dependence using these distributed registers, as the blue arrows shown in Fig. 2(f), the input link in N direction (blue square of PE_2 at t_4) and the input link in W direction (blue square of PE_2 at t_6) of PE_2 are both utilized. Since PE_2 in 2×2 has no input links in S and E directions, there is no input link left to route the 1-cycle dependence from operation a to b at time step t_2, as the dashed red circle shown in Fig. 2(f). Consequently, it is also failed to map the DFG onto HyCUBE with II=1. As a 2×2 HyCUBE has only 8 links (each PE has 2 input links) when II=1 while the DFG in Fig. 2(d) needs at least 9 registers (6 for the loop carried dependence and 3 for intra-iteration dependences), we can never find a valid mapping for the DDG on a 2×2 HyCUBE with II=1.

When mapping the DDG onto our proposed CGRA with II=1 with the same placement as the second one, as shown

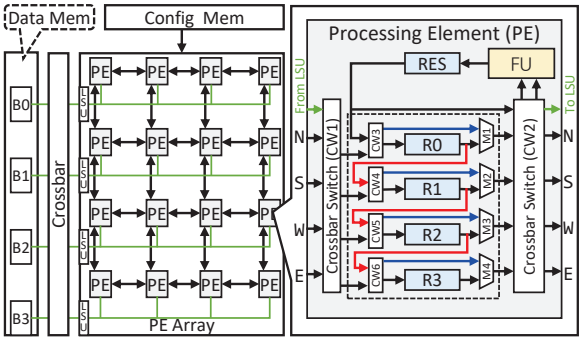

Fig. 3. A 4×4 RF-CGRA with a 4 distributed registers in each PE that can work in a chain way.

in Fig. 2(g), the data (blue squares) of L_1 in PE_0 is firstly written to the R_1 in PE_1 at t_1 via inter-PE register chain, shifted to R_2 in the same PE at t_2 via a intra-PE register chain, shifted to the R_3 in the same PE at t_3, shifted to R_1 in PE_2 at t_4 via a inter-PE link, shifted to the R_2 in the same PE at t_5, shifted to the R_3 in the same PE at t_6, and finally consumed by operation b on PE_2 at t_6. With independent registers, the dependences of (L_1, a), (a, b) and (b, S_1) can also be successfully routed, as black arrows and black squares shown in Fig. 2(g). Finally, we can find a valid mapping for the DFG on our proposed CGRA with II=1 using only 2 PE link instances. With hierarchical register chain, our proposed CGRA is very friendly to route long dependences in DFG. Thus, RF-CGRA is very friendly to map DFG with a high edge/node ratio.

III. RF-CGRA with Hierarchical Register Chain

As we can see in Fig. 3, RF-CGRA is a CGRA architecture that also consists of a 4-bank data memory, a configuration memory, and a 4×4 PE array organized in a 2-D mesh topology. The data memory and the PE array are connected with a crossbar so that each row of the PE array can load from / store to each bank of the data memory. The configuration memory stores the configuration contexts that can be loaded by the PE array to change its functionality and data path. For low power consideration, PEs in the same row share an LSU with a data bus, as the green lines shown in the left part of Fig. 3. Consequently, this architecture allows at most 4 load/store operations to work simultaneously. The main feature of RF-CGRA that distinguishes it from previous CGRAs is the register connection networks, named hierarchical register chain, enabling flexible data routing for DFG with a high edge/node ratio.

As the central RF in a traditional CGRA [2] is explicitly addressed, in pipelined execution fashion, the register address would be frequently changed for a multi-cycle dependence from different loop iterations, leading to extra configuration cost. Although the distributed registers in HyCUBE can work in a chaining way to avoid addressing overhead, a register in each input link of PE can only hold data for only II cycles (assuming that the register is enabled every II cycles and remain unchanged within the interval), leading to link-consuming data routing. Consequently, based on the register organization of HyCUBE, we propose a new RF structure in PE where registers in one PE can work in a parallel way or chaining way. As shown in the right part of Fig. 3, a PE of RF-CGRA also constitutes of an FU, 4 distributed registers, 1 result register, some configuration registers, some crossbar switches, and some multiplexers. With crossbar switches CW3, CW4, CW5, and CW6, a distributed register can receive data either from neighbor PEs or from adjacent distributed registers in the same PE.

Intra-PE Register Chain. By selecting the first input for 2×2 crossbar CW3, CW4, CW5 and CW6, the register R_0, R_1, R_2 and R_3 can work in a chaining way (along the red arrows in the RF shown in Fig. 3). With the configuration of multiplexer M1-M4 and crossbar switch CW2, the length of the intra-PE register chain can be flexibly determined, varying from 1 to 4, such that the input data can retain in the PE for up to 4×II cycles. Besides data latch, each distributed register can also be bypassed to achieve single-cycle multi-hop connectivity as that in HyCUBE. As register number is crucial to register chain while an ALU operation may easily occupy two input registers, we put crossbar switches (CW3, CW4, CW5, and CW6) rather than multiplexers to the input of the distributed registers. In this case, the input data of ALU operation can bypass some distributed registers along blue arrows shown in Fig. 3 such that they can be saved for the intra-PE register chain.

Inter-PE Register Chain. By selecting the second input for 2×2 crossbar CW3, CW4, CW5 and CW6, each distributed register, R_0, R_1, R_2 and R_3, can receive data from neighbor PEs, hold the data for II cycles and directly send the data to neighbor PEs. Consequently, registers in different PE can be connected and form a longer register chain such that relative long dependence can be easily routed. It is worth noting that the intra-PE register chain can start from any distributed register by the control of CW3, CW4, CW5 and CW6. As shown in Fig. 3, the earlier (start from the register with smaller index) the register chain starts, the longer the chain can be formed. To make inputs from North, South, West, East have equal opportunity to form a longer register chain, we also add a crossbar switch CW1 into the input wires.

From RF-CGRA above, we note that, with the input crossbar switch (CW1) and intra-PE register chain, a register can still be used for data routing from other directions even if it is bypassed for a single-cycle hop. Therefore, RF-CGRA is friendly to map DFGs with a high edge/node ratio.

IV. RF-CGRA Compiler

In this section, we present the RF-CGRA compiler that maps application kernels onto the architecture.

A. Mapping Problem Formulation

Resource Graph. The mapping is performed using modulo scheduling, where a new loop iteration can be initiated every II cycle. Given a DFG and a CGRA, we first determine the

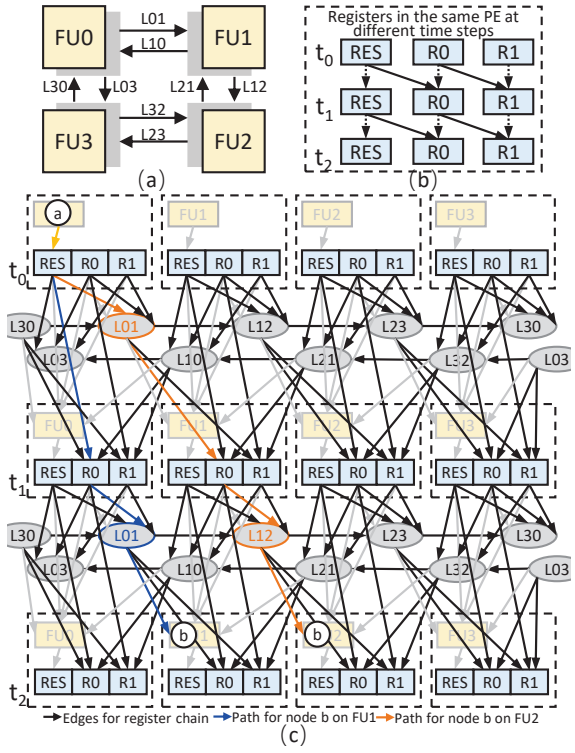

Fig. 4. MRRG for RF-CGRA. (a) 2×2 RF-CGRA, (b) MRRG fragment for registers and (c) detailed MRRG supporting hierarchical register chain.

minimal II (MII) according to the resource-constrained MII (ResMII) and the recurrence-constrained MII (RecMII). With a determined II, we create a time-extended CGRA resource graph as Modulo Routing Resource Graph (MRRG) [1]. An MRRG is a directed graph $G(V_G, E_G)$, where each vertex $v \in V_G$ indicates a CGRA resource and each edge $e \in E_G$ indicates connectivity between two resources. The nodes in V_G indicate two types of resources, including routing resources, V_G^R (e.g., register, etc.) and function units, V_G^F.

In contrast to HyCUBE, RF-CGRA supports intra-PE register chains. Thus, we need to add extra edges between registers in MRRG to present this intra-PE register chain. As shown in Fig. 4(a), there are also 8 links forming a routing fabric to transfer data between neighbor FUs in the 2×2 RF-CGRA. With extra connections among registers in the same PE, the MRRG fragment for registers of a PE would include edges between different registers. As shown in Fig. 4(b), assuming that there are 1 result register (RES) and 2 distributed registers (R0 and R1) in a PE, besides self-to-self edges (dashed arrows), there is also inter-register edges (solid arrows) RES→R0 and R0→R1 indicating intra-PE register chain. The details of MRRGs of the 2×2 RF-CGRA with II=2 are shown in Fig. 4(c), where other routing resources such as MUX and Crossbar and self-to-self register edges are neglected. The RF-CGRA instance shows that both inter-PE and intra-PE register chains can be achieved. Along the 1^{st} blue arrow and the link (L01) in Fig. 4(c), we can perform the intra-PE register chain consuming only 1 link in routing fabric. Along the orange edges and links (gray ovals, L01 and L12) in Fig. 4(c), we can perform an inter-PE register chain. Therefore, the MRRG derived from RF-CGRA is much more flexible for multi-cycle dependence routing.

Algorithm 1: Mapping Algorithm

Input: DFG, RF-CGRA
Output: DFG mapped on MRRG with minimized II
1 RecMII ← DependenceAnalysis(DFG);
2 ResMII ← $\max\left(\frac{\#all\ nodes\ in\ DFG}{\#FUs\ in\ RF-CGRA}, \frac{\#memory\ nodes\ in\ DFG}{\#Banks\ in\ data\ memory}\right)$;
3 II ← $\max(RecMII, ResMII)$;
4 **while** *Mapping is not valid* **do**
5 nodeTimesteps ← Scheduling(DFG, II);
6 orderedNodes ← NodeSorting(DFG, nodeTimeSteps);
7 MRRG ← CreateMRRG(RF-CGRA, II);
8 timeStep ← GetTimeStep(node, nodeTimeSteps);
9 **for** *node of orderNodes* **do**
10 **for** *unmapped FU at slot (timeStep%II) of MRRG* **do**
11 cost ← CalculateRouteCost(node, FU);
12 FUCostSet ← FUCostSet + (FU, cost);
13 **end**
14 OptimalFU ← $\min_{FU}\ FUCostSet$;
15 PlaceAndRoute(node, optimalFU);
16 **end**
17 II ← II+1;
18 **end**

Problem Definition. Given a scheduled DFG $D(V_D, E_D)$ and an II-time extended MRRG $G_{II}(V_G, E_G)$ constructed from an RF-CGRA instance, the problem is to find a mapping $\phi = (\phi_V, \phi_E)$ from D to G_{II} such that:

1) each node $v \in V_D$ should have one-to-one mapping to a node $\phi_V(v) \in V_G^F \subset V_G$.
2) each edge $e = (p, q) \in E_D$ should map to a set of routing resources $S_{pq} \subset V_G^R$ connecting $\phi_V(p)$ and $\phi_V(q)$, and S_{pq} should include $L = t(q) - t(p) + II \cdot dif_e$ register vertexes for timing requirement, where $t(p)$, $t(q)$ and dif_e indicates the time steps of operator p, q and dependence distance of edge e.

From the problem defined above, we note that the most different feature distinguishing our definition from others is routing in the second constraint. Besides finding a series of connected routing resources for a dependence in DDG, we should also precisely control the number register routing resource.

B. Efficient Solution

To solve the mapping problem formulated above, we propose routing-cost-based mapping algorithm presented in Algorithm 1. First, we determine the MII according to the RecMII and the ResMII, where loop carried dependence, the number of FU, and the number of memory banks are all taken into consideration (line 1-3). Then, we perform mapping attempts in a node-by-node fashion in a while loop (line 4-17) until a valid mapping is obtained. In the loop, we first do operator scheduling on the DFG such that each node will be assigned a time step. Then, we sort DFG nodes according to their time steps (line 6). If several nodes are at the same time step, their orders will be further determined by their parent nodes' output degree since nodes associated with high-degree parent nodes

TABLE I
BENCHMARK CHARACTERISTICS

Benchmark	N_{node}	N_{medge}	N_{wedge}	MII	R_{node}	R_{link}
div	39	16	130	3	3.33	90.22%
innerc_0300	52	29	224	4	4.31	116.61%
susan_smo	47	12	97	3	2.06	67.32%
seidel_2d	16	8	43	1	2.69	89.53%
sobel	29	8	83	2	2.86	86.44%
LoG	24	8	74	2	3.08	77.02%
r3000_p_quo	31	8	44	2	1.42	45.82%
dummies	37	3	55	3	1.49	38.20%

are challenging to be routed. Then, for each node and for each unmapped FU at slot (timeStep % II), we select the path with minimal cost using Dijkstra's shortest path algorithm, which is of polynomial time complexity.

The cost function is crucial to the efficiency of the above algorithm. As that in HyCUBE, we also use static routing cost (SRC) and the used adjacent resources cost (UARC). In static routing cost, the number of previously unmapped links is of main consideration. With this cost, routing data via intra-PE register is encouraged since it would consume fewer links. As shown in Fig. 4(c), if node a of a 2-cycle dependence ($a \rightarrow b$) is placed on FU_0 at time step t_0, placing node b on FU_1 (blue path) is better than placing it on FU_2 (orange path), since it consumes only one link (L01).

V. EXPERIMENTAL EVALUATION

A. Experiment Setup

To see the effectiveness of RF-CGRA, a series of comparisons are done against two baseline architectures: Tra-CGRA [2] and HyCUBE [3]. For a fair comparison, the number of registers for the PE in the three CGRAs is all set to 4. We implemented 4 $\times$ 4 CGRA instances of Tra-CGRA and RF-CGRA in Verilog using Pillars toolchain [6] and then mapped them onto 45 nm process. For HyCUBE, as it uses a different process and involves special timing control that is difficult to be re-implemented, we directly normalized the implementation results of HyCUBE to 45 nm according to the proportion between HyCUBE's result and Tra-CGRA's result reported in [3].

We select 8 loops from Polybench [7], MiBench [8], Spec2006 [9] and image processing, which are described from several aspects. As shown in Table I, N_{node}, N_{medge} and N_{wedge} indicate the number of operations, the number of multi-cycle dependences and the sum of weighted dependence (taking dependence length as weight), where dependence length is obtained after initial naive scheduling. As dependence number and length are both crucial to the utilization of register and links, the edge/node rations (R_{node}=$\frac{N_{wedge}}{N_{node}}$) are presented. As the number of link instances in the target MRRG plays a fatal role for dependence routing, the edge/link rations (R_{link}=$\frac{N_{wedge}}{N_{link}}$) are also presented considering the MII, where N_{link} indicates the number of links in the MII-extended MRRG for each kernel.

B. Hardware Overhead

Table II shows the delay of critical path, area and power of the three architectures. According to Table II, the critical path

TABLE II
CRITICAL PATH DELAYS, POWER AND AREA

Arch	Crit.Delay (ns)	Freq. (MHz)	Power (mW)	Area (um^2)
Tra-CGRA	2	500	465	883376
HyCUBE	3.55	282	368	1153797
RF-CGRA	3.6	276	381	1195635

delays in HyCUBE and RF-CGRA are almost the same, it is because their delay of critical path depends on the allowed number of hops within a single cycle (allow 4 hops at most in both architectures). As Tra-CGRA could work at a higher frequency, it has the highest power consumption. We also note that the differences in power and area between RF-CGRA and HyCUBE are very small, and there is only 3.51% growth in total chip power and 3.62% growth in total chip area for RF-CGRA compared to HyCUBE. The extra power and area growth mainly comes from the input crossbar switch (CW1) of each PE. From the above hardware evaluation, we know that some additional connections in RF-CGRA which based on HyCUBE have little effect on chip area and power consumption.

C. Performance Comparison

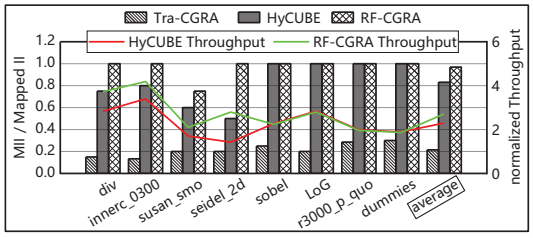

Fig. 5. Performance comparison.

We compare the performance of 8 loop kernels onto RF-CGRA with RF-CGRA compiler, Tra-CGRA with RAMP [10] and HyCUBE with HyCUBE compiler. As the bars are shown in Fig. 5, both RF-CGRA and HyCUBE can achieve the MII on some kernels while Tra-CGRA cannot achieve MII on any kernels. As both RF-CGRA and HyCUBE are equipped with single-cycle multi-hop fabrics, they have relatively sufficient routing resources, and therefore they are both friendly for application mapping. As compared to HyCUBE, RF-CGRA could achieve MII on 7 kernels out of 8 while HyCUBE can only achieve MII on 4 kernels. When mapping kernels of both a high edge/node ratio and a high edge/link ratio (such as div, $innerc\_0300$ and $seidel\_2d$) onto HyCUBE's MRRG, the required number of register instances is approaching the number of link instances (48 links at each time slot) while the number of link instances determined the available registers, leading to difficulty of getting all the edges routed validly. In contrast, with the intra-PE register chain, the number of available registers in RF-CGRA is not limited by links between PE. Therefore, RF-CGRA could still achieve valid mapping of kernels with a high register utilization rate. It is worth noting that even though the edge/node ratio and edge/link ratio of kernel $susam\_smo$ are not high (2.06 and 67.3% respectively), it has many nodes of high output-degree, leading

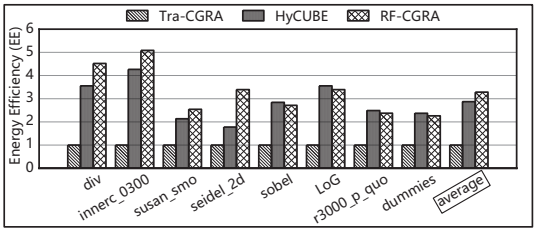

Fig. 6. Energy Efficiency comparison.

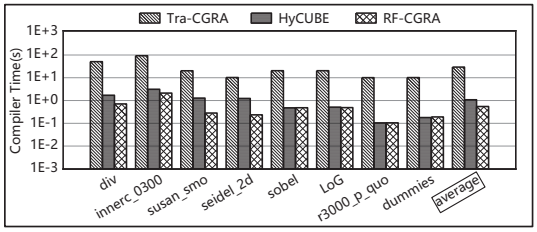

Fig. 7. Compilation time comparison in logarithmic form.

to overuse of single-cycle multi-hop routings. As register latch and single-cycle hop can coexist for a register in RF-CGRA while they are mutually exclusive in HyCUBE, RF-CGRA could still achieve a smaller II than HyCUBE. As for kernel $sobel$, although the edge/node ratio is very high, many of the edges share the same parent nodes, and therefore path sharing is encouraged, saving more routing resources to achieve MII. As for kernel LoG, $r3000\_p\_quo$ and $dummies$, they all have relative low edge/ration ratio. Therefore, both HyCUBE and RF-CGRA could achieve valid mappings for these kernels at MII. On average, RF-CGRA could achieve II of 2.65, while HyCUBE and Tra-CGRA could achieve II of 3.13 and 13.12, respectively. Considering the three architectures have different frequencies, the actual performance (i.e., throughput) should be calculated using $\frac{1}{MappedII \times Crit.Delay}$. As the curve shows in Fig. 5, taking Tra-CGRA as the baseline, the normalized throughput of HyCUBE and RF-CGRA are presented. On average, RF-CGRA could get 2.69× and 1.19× speedup on throughput, as compared to Tra-CGRA and HyCUBE, respectively.

D. Energy Efficiency Comparison

For further overall performance of hardware comparison, we use the energy efficiency to measure hardware performance, and it is obtained by dividing the value of throughput by the value of power (normalization based on Tra-CGRA) of the architecture, which integrates the power consumption, the critical path delay and software pipelining performance of an architecture. Therefore, energy efficiency can effectively represent the overall performance of hardware. Fig. 6 shows the value of energy efficiency for different loop kernels mapped onto the three architectures. RF-CGRA can achieve 3.28× and 1.14× energy efficiency, as compared to Tra-CGRA and HyCUBE, respectively.

E. Compile Time Comparison

Fig. 7 presents the compilation time comparison of different architectures. As both HyCUBE and RF-CGRA have more flexible interconnections, it is easier to find valid mapping as compared to Tra-CGRA. Therefore, the compilation time of HyCUBE and RF-CGRA are both much less, about 3.68% and 1.86% of that on Tra-CGRA. As compared to HyCUBE, RF-CGRA has additional connections among registers, and therefore the connection flexibility of RF-CGRA is further improved. For the former 4 kernels in Fig. 7, as HyCUBE cannot obtain valid mappings at MII, it involves more number of mapping attempts. Thus, their compilation time is longer than those of RF-CGRA. For the latter 4 kernels, as both HyCUBE and RF-CGRA could obtain valid mappings at MII, there is a slight difference in compilation time. On average, the compilation time of RF-CGRA is 50.63% of that on HyCUBE.

VI. CONCLUSION

CGRA is a promising architecture to accelerate compute-intensive loops. However, many real-life loop DFGs are of a high edge/node ratio, and they are difficult to be mapped with existing CGRAs. To alleviate this problem, in this paper, we introduce a novel CGRA architecture, RF-CGRA, which could achieve intra-PE and inter-PE register chains with little hardware cost. Then we present a corresponding mapping approach based on the existing algorithm to find optimized solutions efficiently. The experimental results show that RF-CGRA can achieve improved performance and energy efficiency as compared to state-of-the-art CGRAs.

REFERENCES

[1] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," in *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, ser. DATE '03. USA: IEEE Computer Society, 2003, p. 10296.

[2] B. Mei, S. Vernalde, D. Verkest, H. Man, and R. Lauwereins, "Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix," 09 2003, pp. 61–70.

[3] M. Karunaratne, A. K. Mohite, T. Mitra, and L. Peh, "Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect," *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2017.

[4] F. Bouwens, M. Berekovic, B. De Sutter, and G. Gaydadjiev, "Architecture enhancements for the adres coarse-grained reconfigurable array," in *International Conference on High-Performance Embedded Architectures and Compilers*. Springer, 2008, pp. 66–81.

[5] S. Yin, X. Yao, T. Lu, L. Liu, and S. Wei, "Joint loop mapping and data placement for coarse-grained reconfigurable architecture with multi-bank memory," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16. New York, NY, USA: Association for Computing Machinery, 2016.

[6] Y. Guo and G. Luo, "Pillars: An integrated cgra design framework."

[7] J. Karimov, T. Rabl, and V. Markl, "Polybench: The first benchmark for polystores," in *Technology Conference on Performance Evaluation and Benchmarking*. Springer, 2018, pp. 24–41.

[8] 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 *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, 2001, pp. 3–14.

[9] V. Packirisamy, A. Zhai, Wei-Chung Hsu, P. Yew, and T. Ngai, "Exploring speculative parallelism in spec2006," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 77–88.

[10] S. Dave, M. Balasubramanian, and A. Shrivastava, "Ramp: Resource-aware mapping for cgras," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.