DFGC: DFG-aware NoC Control based on Time Stamp Prediction for Dataflow Architecture

Tianyu Liu[1,2], Wenming Li[1,2,*], Zhihua Fan[1,2]
[1] State Key Lab of Processors, Institute of Computing Technology, CAS, Beijing, China
[2] School of Computer Science and Technology, UCAS, Beijing, China
{liutianyu, liwenming, fanzhihua}@ict.ac.cn

***Abstract*—Coarse-grained reconfigurable architectures (CGRAs) have been regarded as promising accelerating paradigm for the ever-evolving algorithms in multi domains. Obtaining high energy-efficiency on CGRAs relies heavily on the combination of mapping, timing (issuing) and routing to decrease run-time idle. Statically configuration-driven designs are widely adopted, but the leak of hardware flexibility leads to a heavy reliance on burdensome scheduling of compiler to avoid over-serialization. This draw a trade-off between hardware/software co-design in CGRA scheduling.**

Unlike a static-schedule oriented approach, we propose DFGC (DFG-aware NoC Control), a dataflow-driven CGRA which takes the advantages of fully exploring data parallelism in different kernels by using dataflow dynamic firing with low overhead. The DFGC compiler is responsible for analyzing critical data paths and generating *TimeStamp* predictions instead of per cycle configuration. The rough predicted results enable router and PE to be sensitive to the entire dataflow graph, thereby accelerating the whole computation process. The DFG-aware NoC design realize a combined scheduling technique of hardware dynamic decision-making together with static prediction. DFGC represents a paradigm of CGRA that is worth exploring, achieving hardware/software co-design without relying on sophisticated designed compiler. Experiments show that DFGC achieves $1.32\times$ energy efficiency improvement over a dataflow architecture and $1.8\times$ energy efficiency improvement over a state-of-the-art static configured CGRA.

***Index Terms*—CGRA, dataflow, hardware/software co-design, NoC**

I. Introduction

Coarse-grained reconfigurable architectures (CGRAs) have received increasing attention in recent years due to its characteristics of both approaching near-ASIC performance while providing flexibility. The efficiency of CGRA is primarily achieved by deploying various types of computations on the architecture and enabling high data parallelism through temporal and spatial execution [1]. This kind of complex allocation of hardware resources relies heavily on the static compilation by the compiler, for it takes on the task of providing static configurations on mapping, issuing and routing [2], [3].

However, when accelerating kernels with complex control flow loops, pure statically scheduled CGRAs suffers from insufficient parallelism due to over-serialization and workload imbalance [4]. This has created a higher demand for hardware flexibility and reconfigurability: The architecture should employ flexible and efficient methods to partition loops, and further exploit the data parallelism inherent in operations [5]. This is where the concept of dataflow comes into play in CGRAs. Work like Triggered Instruction [6], [7], Softbrain [8], SPU [9], SNAFU [10] and others maintain the mutual dependency between dataflow graph (DFG) nodes through their hardware execution mechanism, and gain efficiency from the dynamic issuing of dataflow. These work demonstrate a high degree of hardware flexibility.

The introduction of dynamic hardware mechanisms such as dataflow has broadened the choice of software-hardware co-design scheduling approaches. CGRAs can either achieve hardware simplification with the precisely defined hardware behavior of each cycle through a rigorous thus expensive schedule-space search, or add acceptable hardware flexibility to reduce the burden of software scheduling. Given the widespread discussion on the former, this paper believes that the latter deserves more profound exploration: a general solution is in need to maintain low overhead while providing hardware with a certain degree of flexibility, realizing joint optimization of mapping, scheduling, and routing without relying on highly complex compilers.

In this paper, we propose DFGC, a dataflow-driven CGRA. DFGC benefits from an extended functionality PE (Processing Element) and Router that are aware of the critical path of the entire DFG. It can not only keep low complexity of compiler design in terms of mapping and scheduling, but also utilize the flexibility of dataflow architectures more efficiently with low overhead. DFGC compiler generates rough prediction from the mapping result, and gives PE and router a view of the overall computation process, thereby achieving the goal of dynamic scheduling enhancement. This work achieves a co-design of software and hardware by using software to predict the reference factors for hardware decision-making process. We believe that under this technique, dataflow-driven CGRA is a paradigm worth exploring.

Our contributions are listed as follows:

- A distributed dataflow control mechanism on DFGC, as well as a supporting dataflow analyze model. Following the model, we propose the DFGTE algorithm, which predicts the possible significant transmission paths during the actual execution process. According to the mapping result, DFGTE can generate a rough *TimeStamp* to guide the scheduling of PE and NoC (Network-on-Chip).

* Corresponding author

2576-6996/23/$31.00 ©2023 IEEE
DOI 10.1109/ICCD58817.2023.00071

- The DFG-aware hardware design. DFGC utilizes both static predicted result and local real-time status to make hardware decisions on PE and NoC, which could accelerate execution and manage routing resources effectively. Through the technique, DFGC achieves software-hardware synergy in scheduling.
- Evaluation on DFGC, including comparison to dataflow architecture, CGRA, GPU and DSP.

The rest of the paper is organized as follows: Section II provides background and related work, Section III proposes the dataflow analyzing model and *TimeStamp* extraction algorithm. Section IV gives the detailed hardware implementation on router and PE, as well as the hardware schedule process. Section V evaluates, and Section VI concludes.

II. Background & Related Work

This section discusses the relationship between mapping, issuing and routing of CGRAs, to reach hardware efficiency. The merits and drawbacks of introducing dataflow dynamic firing into CGRA are also presented.

A. Scheduling: hardware/software tradeoff

The choice of scheduling method is vital to CGRA efficiency, because it determines the spatial-temporal allocation from workload to hardware substrate. The mapping and subsequent scheduling process are completely coupled in those pure static configured CGRAs, where joint optimal strategy of mapping, routing, and issuing are found by heuristic-based methods such as simulated annealing, to generate per-cycle hardware activity configurations [2], [11]–[14]. The search for an optimal solution is constrained by hardware resource limitations and specific execution model. These static scheduled CGRAs stands out for low hardware overhead but struggle to adapt to algorithms with complex control flows due to their lack of time-multiplexing mechanism for hardware resources, or the deficiencies in timing. For instance, DySER [15] only allows non-conflicting dataflow dependencies to coexist on the architecture at the same time, so as FIFER [16]. 4D-CGRA [17] allows conflicting nodes to be assigned to the same resource at the same time, but requires compiler to predict all fine-grained conflicting hardware behavior. Moreover, they often have strict requirements for the partitioning of computational stages to form predicated DFG [16], and need delay mechanism for synchronization [18]. Therefore, these CGRAs also have to bear the burden brought by the compiler, which is due to the complexity of the timing process and the time-consuming overhead of prediction [19].

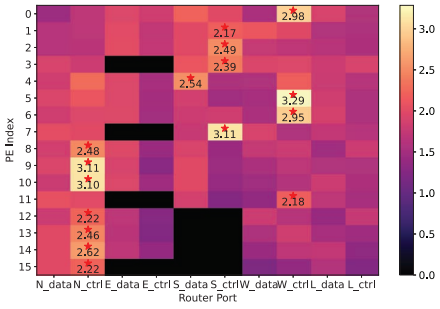
The execution model of dataflow-driven CGRA is fundamentally different from the above-mentioned statically compiled CGRA. Its scheduling method, that is, deciding when to execute a certain operation somewhere depends entirely on the control of dataflow, rather than static planning (predicting) in advance. Therefore, there is no need for compiler to analyze the allocation of hardware resources for each calculation cycle because hardware can autonomously explore data/instruction parallelism. A typical task scheduling technique on these architectures is static mapping and dynamic issuing, such as in [8], [20], [21]. Advanced mapping algorithms are also required to consider load balancing and routing overhead, and distribute DFG nodes or edges to specific function units or routers [22], [23]. Considering this, many work aims to incorporate hardware-specific execution model into mapping process, improving potential instruction/data-level parallelism and reducing NoC congestion through dynamic hardware scheduling. However, static mapping cannot accurately predict all actual runtime conditions.

Current new architectures combine the advantages of the aforementioned two approaches by adopting a co-scheduling strategy between software and hardware. This approach provides better adaptability to complex loop control, and brings higher requirements for software-hardware co-design. Although REVEL [13] combines two computing paradigms, it heavily relies on compiler's heuristic search for the globally optimal timing decision due to the consideration of balancing between dataflow PEs and systolic PEs. SNAFU [10] employs dynamic dataflow firing, which alleviates the burden on compilers for operation timing. However, it does not support conflicting mappings of dataflow nodes or edges to single PE or route, and can only speed up inner loops. According to the above analysis, a consensus can be reached that the selection of hardware scheduling mechanism evidently requires a hardware/software co-design trade-off: hardware flexibility and software complexity both matter.

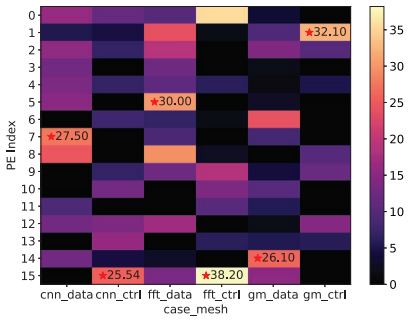
DFGC takes a different approach by utilizing a block-level dataflow control mechanism, which decouples hardware execution from static mapping and allows for dynamic scheduling and firing of instruction blocks. The dataflow control mechanism enables DFGC to better adapt to algorithms in multi-domains. As for efficiency, the scheduling on DFGC only requires minor assistance from the compiler in terms of timing. The compiler of DFGC does not need to plan the overall operation process (no need to generate timetable), but leaves the dynamic exploration of data parallelism to the hardware mechanism. It only needs to generate rough prediction *TimeStamp*s, to assist hardware scheduling and make acceleration. By filling in the limitations of hardware dynamic scheduling with a software-hardware co-scheduling approach, the hardware utilization of DFGC is improved.

B. Routing: bridge the gap

NoC is an indispensable module in CGRA and fills the gap between mapping and issuing, and its design impacts the actual hardware resource utilization. The interconnection on CGRA needs to be adapted to the architecture-specific execution mechanism on one hand, and on the other hand, it is also influenced by the actual real-time communication requirements. The requirement of generality calls for reconfigurable NoCs, whether driven by configurations of compiler or dynamically routing, to maintain high efficiency PE-PE communications [18]. Configuration-driven CGRAs generate routing results based on the edges of the DFG according to the mapping results. This serves as the foundation for



(a) Trans time in router (conv)



(b) Edge latency of DFG

Fig. 1: (a) The average period that a packet stays in router. Star mark the most overloaded router port on each PE, black represents unused ports, based on ideal 1-cycle-trans. (b) DFG edge latency, recorded by producers of dependencies.

the compiler to predict and set strict configurations for each cycle, for example, HyCube [24]. In this paper, we discuss the relationship between routing, mapping, and hardware resource status on dataflow-driven CGRAs and seek optimal solutions. We choose a typical dataflow-driven CGRA deployed with mesh NoC, which heavily relies on PE-PE communications of data/control signal transmission to execute DFG-like programs. At run-time, data from five directions (N, E, S, W, L) are passed to their destination Router in a FIFO order. An arbiter manages competing data paths using a round-robin policy to maintain fairness and balance the transmission pressure.

Fig. 1 (a) demonstrates the average transmission time on different router ports of convolution (CONV-64) on a 4×4 PE array. Data latency on the router is caused by queuing delay for multiple packets from the same source, competition among packets with the same destination, and downstream congestion. The uneven distribution of data and control path transmission characteristics in the convolution example, particularly with the highest value 3.29 cycle in PE-5 (located at (1, 1)), ctrl port of direction W, can slow down the overall execution. This is primarily attributed to the suboptimal allocation of network resources during the execution of the DFG on the array, caused by the limited control over the overall execution process from a hardware perspective. As for the perspective of DFG, Fig. 1 (b) shows the average latency from the producer side of DFG edge in three general applications, represented by (trans_time/packet_num) on dependency edge. It can be observed that in FFT-64, PE-15 (located at (3, 3)) experiences the maximum level of congestion in the ctrl path. The DFG nodes located on this PE, along with their downstream nodes, are subjected to varying degrees of edge delay, leading to obstruction of intermediate results in the computation process. So as PE-1 when focusing on control signals of DFG in GEMM-64. Hence, a dynamic forwarding network that avoids latching intermediate results or critical dataflow control signals is necessary based on the above analysis. To better allocate on-chip network resources, the router and related structures inevitably need to go beyond the original hardware perspective, abandon the fair scheduling and instead providing optimal strategies from the global perspective of the entire DFG.

In this paper, DFGC employs a combination of software and hardware schedule mechanism to compensate for the limitations of pure hardware decision-making. Routing and execution on DFGC is determined in real time by both run-time hardware status and rough prediction results based on mapping result, in order to balance the actual resource demand of routing and the macro running state of the whole DFG. The mapping strategy only needs to consider the route length and resource distribution without considering the actual scheduling process. The scheduling scheme of DFGC not only achieves flexible control with lower hardware overhead, but also gains the advantages of dataflow control for complex computation modes, and greatly reduces the burden on the compiler, eliminating the need to predict and schedule configurations for each cycle.

C. Challenges

DFGC differs from common CGRAs in that it uses dataflow to dynamically handle resource issuing to improve utilization, and differs from the general dataflow structures in that it introduces partial compiler predictions to guide more efficient data transfer, balancing hardware and software in scheduling. We distill the main challenges in designing an enhanced dataflow-driven CGRA, which will be discussed in this paper.

❶ **Effectively modeling and predicting on DFG.** A DFG analyzing method is in need to identify the critical path that slows down the overall execution from the complex dataflow dependencies. This requires an optimization and abstraction of the original execution mechanism with a time-phase partitioning of execution process, without over-detailed prediction. As the internal execution order is somewhat decoupled from static compilation mapping result on dataflow-driven substrate, it is almost impossible to predict when the data on which the consumer depends will arrive at runtime [1]. DFGC uses dynamic routing and rough predictions to handle PE-PE communication, addressing the limitations of the router's perspective for overall speedup. The paper focuses on effectively modeling and predicting with this challenging approach.

❷ **Optimizing hardware decisions via DFG-aware design.** Router on DFGC is supposed to leverage the aforementioned modeling and prediction results to prioritize the transfer of critical nodes on the data path, while flexibly controlling

real-time routing resources with low control overhead. Our goal is to speed up the execution of the entire DFG by using this smart DFG-aware hardware, increasing the overall resource utilization of the PE array. It is crucial to analyze and evaluate the runtime resources and make optimal decisions for the whole process, which plays a critical role in reducing the idle state of the entire architecture. Finally, the benefits of the design and the comparison with other alternatives are presented through experiments.

III. DFGC Overview Design

The underlying hardware control mechanism and upper-level software execution model are closely related in the architecture, establishing the specific problem scenario for this article. This section introduces the overview design of DFGC and presents the modeling method for *TimeStamp* prediction.

A. DFGC Control Mechanism

After studying the evolution of prior dataflow architectures [4], [20], [25], [26], a trend is found that architectures are choosing coarse-grained (block-, loop-, kernel-level, etc.) dataflow execution to reduce control complexity on hardware. In this paper, DFGC also chooses a CGRA with integrated dataflow execution model as hardware substrate. The scheduling flexibility of DFGC falls between "Ordered dataflow" and "Tagged Dataflow" taxonomy of prior work [13], as it does not support instruction reordering within the same block but supports out-of-order control between different DFG edge instantiations. This enables DFGC to handle conflicting DFG nodes and edges independently. In addition, the duplicate instantiate operations further amortize the control overhead.

As shown in Fig. 2, all the configurations and instructions are transferred to CBUF before being packaged and broadcasted to different PEs via data mesh, while MICC is responsible for run-time schedule on PE array scale, including kernel switch and task level parallelism exploiting. DFGC eliminates the centralized control within a single DFG, which means that the switching of loops within a graph is achieved through distributed control on the PE array. This approach effectively reduces the number of routing hops required for control signals during instance switching and mitigates the significant, unpredictable factors introduced by centralized control in subsequent modeling processes. Additionally, it alleviates the burden on the NoC and enhances overall efficiency.

The structure of PE is illustrated in the lower part of Fig. 2, where each PE contains a control unit (CU) for block scheduling, maintaining inner PE block info and status information for dataflow dynamic firing. A series of combinational logic forms block matching mechanism, prefetching blocks for the decoupled pipeline before execution. Furthermore, an active and ack control protocol with block tokens and credits enables pipelined execution of blocks inner PE, and keeps dataflow execution dependencies cross PEs. Dynamic scheduling occurs in a block-level fashion, whenever tokens and credits from upstream and downstream of a dataflow node meet certain requirements, the node will trigger an Instance and decouple

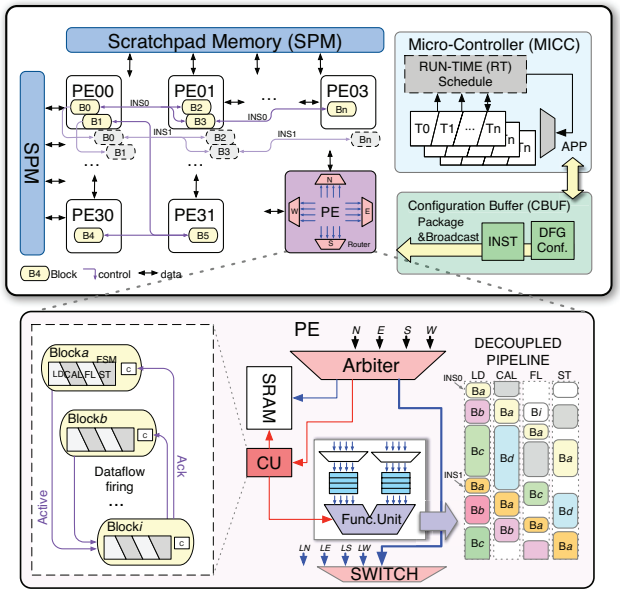


Fig. 2: DFGC control mechanism diagram.

its computation stages (*LD*, *CAL*, *FLOW*, and *STORE*) for parallel execution. The reconfigurable control mechanism makes it possible for DFGC to deal with many kinds of computation manner and complex task-level parallelism.

B. DFGC Analyzing Model

1) The Setup of "Instance" and Challenges: The mapping strategy in DFGC is partially developed based on prior work [27], where the memory access behavior, data parallelism and data dependencies in the calculation process of the algorithms are comprehensively considered in this figure, unifying their dataflow representation. It is worth noting that the multiple instantiation and pipelined execution of DFG edges on DFGC is particularly suitable for complex calculations with contextual dependencies through iterations. Apart from complex control loops, it is usually used in cycles for direct accumulations, where an Instance of instruction can directly compute over the result of its previous Instance. DFGC's energy efficiency is also benefited from the amortizing of control overhead in multiple Instances of an edge. Furthermore, for commonly unrelated loop executions, DFGC can also distribute them across parallel edges and nodes, maximizing instruction-level parallelism through decoupling the pipeline and utilizing the dynamic dataflow firing. This distinction is what sets DFGC apart from most CGRAs and also the reason why most previous static scheduling strategies are not applicable to DFGC such as [5].

The DFGC execution mechanism provides flexibility and a wide range of mapping options, but also presents challenges. This parallel execution pattern through different iterations emphasizes the importance of spatially distributing computational resources, i.e., load balancing. Furthermore, almost every instruction is ready to be executed in each cycle during execution, and each function unit generates a result almost

Algorithm 1 The DFG *TimeStamp* Extraction Algorithm

Input: $\mathcal{G}$ - DFG, $\mathcal{P}$ - PE array, $\mathcal{A}$ - a mapping $\mathcal{G} \rightarrow \mathcal{P}$
Output: $\mathcal{T}$ - *TimeStamp* of ($\mathcal{A}$, $\mathcal{B}$ - blocks, $\mathcal{P}$)

```
1: N ← nodes(G), B ← Instance_expansion(N, G)
2: T ← null, pCosts ← null
3: for each block i in sorted (B, G) do
4:     ranked_costs ← null
5:     if i.children is not empty then
6:         for each block c in i.children do
7:             cost ← CALPATHCOST(c, B, pCosts)
8:             ranked_costs.append(c, cost)
9:         end for
10:        tp ← ranked_costs.Sort
11:        T ← T + tp                    ▷ merge duplicate item.
12:    end if
13: end for
14: Sort_Normalization(T)
15: return T
16: function CALPATHCOST(block n, B, pCosts)
17:    ...    ▷ return pCosts[n] or n.exeLatency are omitted.
18:    mcost ← 0
19:    for each block c in n.children do
20:        cost ← CALPATHCOST(c, B, pCosts)
21:        mcost ← max(mcost, cost + transLatency(n, c))
22:    end for
23:    pCosts[n] ← n.execLatency + mcost
24:    return pCosts[n]
25: end function
```

every cycle, leading to a high volume of data transmission between PEs and intense competition on NoC. Therefore, real-time and precise control and scheduling of data transfers within the NoC are crucial to maximize the utilization.

2) DFG TimeStamp Prediction: Based on the above analysis, we have employed a load-balance-centric dataflow instruction scheduling algorithm based on [28], which ensures that different components within the same execution unit are utilized synchronously to a maximum extent.

After mapping, instructions are packed to nodes and form DFG, which is then used as input to extract prediction result for PE and NoC (Algorithm 1, DFGTE) in DFGC compiler. DFGTE plans the control signals and data interactions between code blocks following the hardware control protocol, enabling coarse-grained inter-block interaction priority prediction (higher *TimeStamp* has higher priority). For data memory access operations, we extract the destination SPM and set up virtual DFG nodes on the neighboring PE to simulate the actual control and data transfer scenarios. DFGTE expands the static collection of DFG nodes into dynamic executable blocks based on the initial mapping configuration, adhering to dataflow firing rules. For a small-scale instantiation of nodes within a single graph, a top-down search approach is adopted by expanding the nodes and evaluating the overhead. For cross-DFG Instances, the recursive search for edge weights between graphs is no longer performed (line 1). By calculating the costs (routing & execution) of downstream blocks for each block, DFGTE prioritizes the transmission and execution behavior for each instruction block during execution (line 7, 16–25), forming *TimeStamp*. The transmission cost is determined solely based on the routing hops for the target signals/data, without considering any potential network congestion.

DFGTE is ingenious in generating *TimeStamp* predictions with the triggered execution of blocks as the anchor point, because despite the unpredictable trigger time of specific instructions, the Instance iteration and execution on DFG edges strictly adhere to dataflow dependencies. The *TimeStamp* design enables the compiler to have clearer optimization objectives during mapping: maximizing computational resource utilization while offloading conflict DFG execution to hardware through global prediction parameters. In addition, the size of the *TimeStamp* under this design depends entirely on DFG depth and the number of Instance of specific kernel. The former generally does not exceed 10, and the *TimeStamp* file can also be replicated when the number of iterations is large. Therefore, this design extremely reduces complexity and storage required by generating per-cycle static timetables.

IV. DFGC DFG-aware NoC Design

This scection illustrates the detailed DFG-aware NoC control process from inner-PE to inter-PEs. DFGC realizes hardware-software co-design through this approach.

A. TimeStamp Table Monitor on PE

Once generated by the compiler, the *TimeStamp* information, along with other dataflow configuration, is broadcasted to all PEs and stored in *TimeStamp* Table Monitor (TPM) and Block Status Control (BSC), respectively. Fig. 3 shows the whole scheduling process from inner-PE dynamic firing to inter-PE NoC control. As in (a), ❶ BSC receives control signals from PE and external sources, updating local block status. Meanwhile, idle pipeline components request access from BSC, which places matched blocks in the block pool, before retrieving the block with the highest "TP" value from the TPM. The *TimeStamp* value is a 9-bit (32 blocks per PE) value generated from the prediction result (DFGTE, line 14). ❷ The selected block in the pipeline undergoes decomposition and execution, with its instructions continuously issuing request signals or data to local and external PEs, shown in (b).

❸ Upon completion of a block's execution, it sends control signals to the upstream and downstream blocks within the same DFG. These signals, along with the packets generated in the pipeline, are sent back to the CU for message packaging. Each packet checks the *TimeStamp* information of the block it belongs to from the TPM, and then uses it again as the priority of the subsequent routing on the NoC to speed up the transmission of the critical path.

B. DFG-aware run-time Router

❹ In each router, packets from local PE (L), as well as from direction N, E, S, W, are sent to the respective ports and awaits forwarding. Fig. 3 (c) illustrates hardware design we

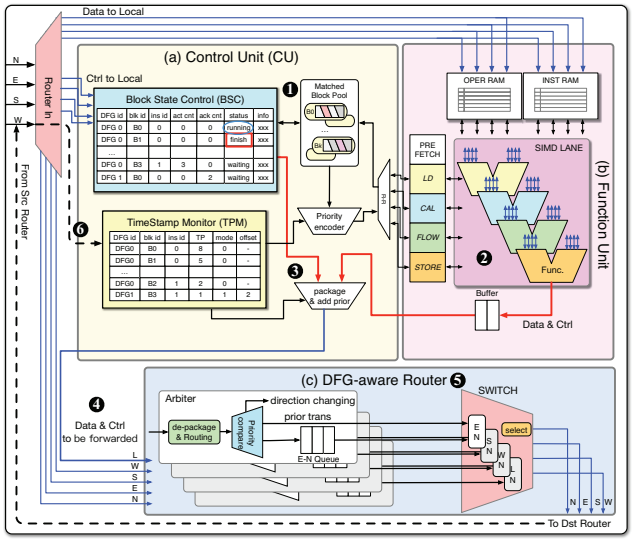


Fig. 3: DFGC PE and NoC hardware design.

implemented on router to accommodate *TimeStamp* prediction and improved scheduling of on-chip data transfers.

1) Avoid queuing delay: In each port, the arbiter continuously maintains the priority of the packet at the bottom of the forwarding queue. Whenever a new packet has a higher priority than the target value, the new packet is exempted from queuing and is directly sent to the SWITCH via the priority transmission channel. The SWITCH arbitrates between competing packets in each cycle, selecting the higher-priority packet, otherwise maintaining the round-robin (R-R) mode for fairness when packets from different directions have no priority difference. In this case, the combination of queuing bypass and preemptive switch selection design allow the DFGC's NoC to avoid latching intermediate data or control signal caused by queuing delays.

2) Relieve competing and downstream blocking: ❺ DFGC has set up a hardware-autonomous routing mechanism on the NoC, that is, the Arbiter decides the direction of packet transmission based on run-time blocking situation on port, and send the packet to a relatively idle queue without loss of transmission distance (multicast is not supported). The autonomous routing decision enables the efficient scheduling of DFGC NoC based on real-time congestion conditions.

3) The reconfigurable TimeStamp: ❻ DFGC also supports *TimeStamp* calibration at run-time. When the "mode" bit of the TPM entry is set to 1, the "offset" value will be added to the TP value as an update. On the one hand, MICC can adjust the execution priority of a kernel relative to other kernels in real time through this mechanism in a multi-DFG parallel execution scenario. On the other hand, the block can provide its last ready upstream producer with a higher priority through the ack control packet, which could obtain a more balanced execution and transmission in the next Instance. This offset backward delivery design serves as a complement in the event of a serious inaccurate prediction.

In summary, DFGC achieves optimization of NoC and inner-PE scheduling through a combination of software and hardware techniques. It employs software predictions to capture rough hardware execution and communication behaviors, and uses real-time hardware-based result selection to refine the software predictions. This method assists the distributed control of hardware with software DFG-aware global prediction, realizes a novel scheduling method, and improves the efficiency of the overall architecture.

V. Evaluation

A. Experiment Settings

Methodology: We implemented DFGC in RTL and synthesized the hardware using an industrial 12 nm process, meeting timing at a 1 GHz clock frequency. The DFGC simulator was also developed and was calibrated to cycle-level accuracy. Evaluation shows that DFGC has peak performance of 1TFLOPS with a chip area of 26.9mm^2 for 3.8 watts.

Benchmark: We compare DFGC to two kinds of baselines. First is CGRAs (or accelerators) including the dataflow architecture LRPPU [21], pure statically scheduled HyCube [24], as well as the hybrid designed REVEL [13]. Second, the energy efficiency of DFGC is evaluated compared to the three work: TMS320C6678 [29], NVIDIA Tesla V100 [30] and Jetson AGX Xavier, representing high-parallel general purpose approaches. As DFGC is mapped to 12nm process, all the evaluation parameters are scaled (1.42× per generation due to 0.7 scaling of feature size, with the power remaining the same as core voltage is kept constant) for fair comparison.

Applications: We implemented 2D fast-fourier transform (FFT, 16–128, 1K–8K), matrix-matrix multiply (GEMM, 32–256), matrix-vector multiply (MV, 32–256), and convolution (CONV in ResNet-18, 50). These kernels play a crucial role in various critical applications such as signal processing, graph analysis, machine learning (including deep neural networks), and communications. The DFGC compiler is developed based on the LLVM [31] infrastructure. All applications go through the process of DFG loop splitting, DFG mapping and *TimeStamp* prediction from `C`-code in the compiler, before assembly code and configuration files are finally generated for DFGC.

B. Power and Area Breakdown

Table I shows the hardware design parameters, as well as the evaluated power and area breakdown of essential modules in DFGC. The high parallelism of the hardware helps to amortize

TABLE I: Design Parameters and Breakdown

Module	Destription/Para.	Area (mm^2)	Power (mW)
single PE	Func. Units (SIMD)	0.7 (2.6%)	73.9 (1.9%)
	Control Unit	0.04 (0.2%)	35.1 (0.9%)
	Buffer	0.5 (1.9%)	70.9 (1.9%)
PE Array	4 × 4 PE	20.5 (76%)	2878.4 (75.7%)
NoC	16 × Router, 2-mesh	1.4 (5.3%)	265.5 (7%)
SPM	6MB, 8 bank, L shape	4.2 (16%)	534.6 (14.1%)
MICC&CBUF	-	0.8 (2.9%)	121.5 (3.1%)
Total		26.9mm^2	3.8W

TABLE II: Hardware Comparison w.r.t. the State-of-the-art CGRA

Design	Fabric size	Scheduling Method	NoC Design	Conflict DFG	Energy Effic. (GFLOPS/W)
DFGC	4×4	static mapping – dynamic issuing	static & dynamic	support	189 (gmean)
HyCube [24]	4×4	static mapping – static issuing	static, bufferless, multi-hop	not support	90–120
REVEL [13]	5×5	static mapping – hybrid	static & dynamic NoCs (2×)	not support	85–110

the control overhead, and the design employed for *TimeStamp* scheduling incurs little overhead (0.03% per PE in total area).

C. Comparison with CGRAs

Due to the shared focus on enhancing dynamic firing of dataflow, LRPPU is chosen as a comparative reference for DFGC in the dataflow architectures. As the original design of LRPPU primarily focused on the CNN-specific domain and had limited support for various kernels, we extract its centralized "iterative pipeline execution" mechanism and implement it on the hardware substrate of DFGC. This serves as the basis for comparing DFGC with LRPPU, which is deployed with the same floating-point computation units and configured to the same peak performance as DFGC. The following hardware evaluations of LRPPU are conducted under the same 12nm process technology. This comparison highlights the advanced scheduling and distributed control of DFGC.

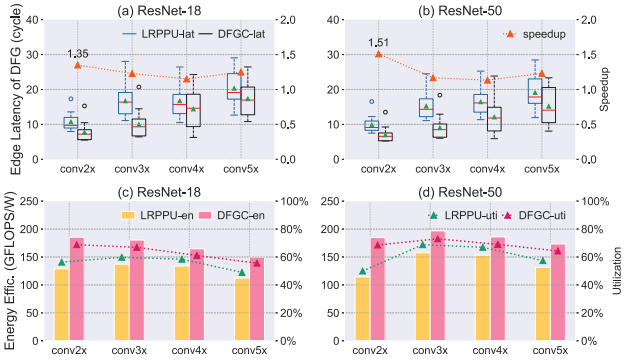


Fig. 4: Performance & Energy efficiency versus LRPPU.

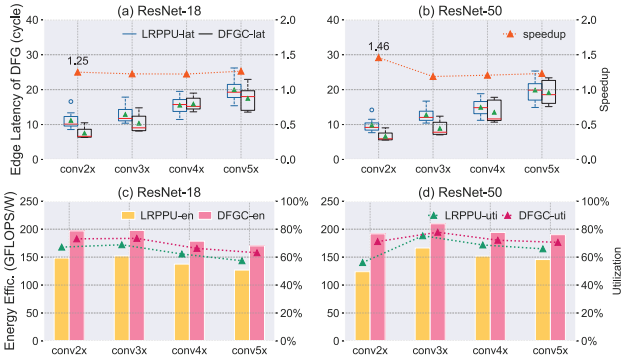


Fig. 5: Comparison under fabric size 2×4.

Fig. 4 presents the performance and energy efficiency of DFGC versus LRPPU. The presented data showcases the averaged results of a series of multi-layer computations conducted within each "convx" module of ResNet, representing the outcome of each individual layer operation. Fig. 4 (a) and (b) illustrate the maximum speedup of DFGC over LRPPU in the ResNet-18 and 50, which are 1.35× and 1.51×, respectively, both occurring at conv2x. Additionally, the overall geometric mean is 1.25× speedup in these two networks. (a) and (b) also demonstrate the scheduling effectiveness of DFGC's *TimeStamp* prediction on the delay of DFG edges across 16 PEs, resulting in a significant alleviation of PE pressure on the highest edge delay. In conv3x, ResNet-18, DFGC effectively schedules 28118 data packets, resulting in an average reduction of 40.67% in the delay of each edge's consumer side during execution. DFGC exhibits an average edge latency reduction of 1.42× and 1.45× compared to LRPPU in ResNet-18 and 50. As for energy efficiency, Fig. 4 (c) and (d) demonstrates a geometric mean 1.33× and 1.35× improvement in ResNet-18 and 50, respectively, where the highest increase in utilization of the computation components is observed in the conv2x, ResNet-50, where it has been improved from 49.9% to 68.6%.

To evaluate the scalability of our design, the comparison is also deployed on a 2×4 fabric substrate, shown in Fig. 5. It can be observed that on a downscaled array, the scheduling mechanism of DFGC can still maintain a geometric mean 1.32× and 1.35× energy efficiency improvement, with 1.24× and 1.27× speedup in ResNet-18 and 50, respectively.

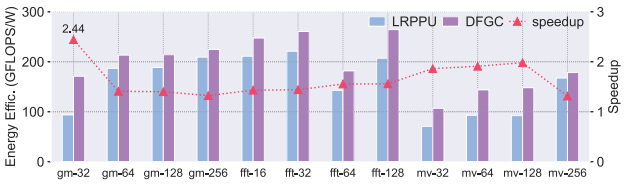


Fig. 6: Performance and energy improvement of other kernels.

The evaluation results of GEMM, FFT, and MV are shown in Fig. 6. DFGC achieves the best acceleration and energy efficiency improvements in GEMM-32, with a speedup of 2.44× and an energy efficiency gain of 1.82×. Compared to the hardware scheduling mechanism in LRPPU, DFGC achieves gmean of 1.32× and 1.64× efficiency and performance improvements, respectively. Lastly, DFGC exhibits a gmean energy efficiency of 189 GFLOPS/W in small-scale GEMM, FFT, and MV computations. Furthermore, the time cost of generating *TimeStamp* in all experiments can be negligible compared to the kernel execution time.

As for comparison with the other CGRA designs, due to different design objectives and target applications, there is a significant disparity in peak performance among these designs. We made efforts to mitigate the influence of peak performance and process technology disparity during the evaluation. Here,

the hardware comparison is listed in TABLE II. It is shown that DFGC has an approximately $1.8\times$ and $2\times$ energy efficiency improvement over HyCube and REVEL, respectively.

D. Comparison with GPU and DSP

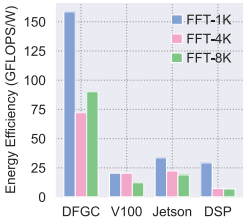


Fig. 7: Energy effic. compared to GPU and DSP.

Due to the high peak performance of DFGC (1 TFLOPS), it is evident that the comparisons with those lightweight CGRAs are not entirely fair. DFGC actually has significant advantages when computing extremely large-scale kernels. Therefore, we provide an energy efficiency comparison between DFGC, GPU, and DSP for large-scale FFT operations, to show the overall superiority of the architecture. As Fig. 7 exhibits, DFGC reaches the highest energy efficiency of 158 GFLOPS/W, at 1K point FFT. The results demonstrate a $5.9\times$ gmean improvement of DFGC over V100, $4.2\times$ over Jetson, and $8.9\times$ over DSP in large scale FFT. This further validates the advantages of DFGC over general-purposed computing solutions.

VI. Conclusion

In this work, we analyzed the relationship of mapping, issuing and routing on two mainstream CGRA design approaches (static or dynamic scheduling) and highlighted the significance of software-hardware co-design. Experiments have confirmed the superiority of DFGC in terms of performance and energy efficiency, achieved by the DFG-aware design. DFGC fills the gap between software mapping and hardware scheduling on high flexible architectures such as dataflow. The proposed *TimeStamp* method can be migrated and integrated according to various architectures and their mapping schemes, enabling better scheduling. In our future research, we will consider using actual data transmission condition statistics on PE array to guide the formation of more optimal mapping solutions.

Acknowledgments

This work was supported by National Key Research and Development Program (Grant No. 2022YFB4501404), CAS Project for Young Scientists in Basic Research (Grant No. YSBR-029), CAS Project for Youth Innovation Promotion Association, Open Research Projects of Zhejiang Lab (Grant No. 2022PB0AB01) and Beijing Nova Program (NO. 20220484054).

References

[1] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, and S. Wei, "A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications," *ACM Comput. Surv.*, 2019.

[2] J. Weng, S. Liu, V. Dadu, Z. Wang, P. Shah, and T. Nowatzki, "DSAGEN: Synthesizing Programmable Spatial Accelerators," in *ISCA*, 2020.

[3] T. Nowatzki, M. Sartin-Tarm, L. De Carli, K. Sankaralingam, C. Estan, and B. Robatmili, "A general constraint-centric scheduling framework for spatial architectures," *ACM SIGPLAN Notices*, 2013.

[4] L. Chen, J. Zhu *et al.*, "An Elastic Task Scheduling Scheme on Coarse-Grained Reconfigurable Architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 12, pp. 3066–3080, 2021.

[5] M. Kou, S. Wei, and S. Yin, "TAEM: Fast Transfer-Aware Effective Loop Mapping for Heterogeneous Resources on CGRA," in *DAC*, 2020.

[6] A. Parashar, M. Pellauer *et al.*, "Triggered instructions: A control paradigm for spatially-programmed architectures," in *ISCA*, 2013.

[7] A. Parashar, M. Pellauer, M. Adler, B. Ahsan, N. Crago, D. Lustig, V. Pavlov *et al.*, "Efficient Spatial Processing Element Control via Triggered Instructions," *IEEE Micro*, vol. 34, no. 3, pp. 120–137, 2014.

[8] T. Nowatzki, V. Gangadhar, N. Ardalani, and K. Sankaralingam, "Stream-Dataflow Acceleration," in *ISCA*, 2017.

[9] V. Dadu, J. Weng, S. Liu, and T. Nowatzki, "Towards General Purpose Acceleration by Exploiting Common Data-Dependence Forms," in *MICRO*, 2019.

[10] G. Gobieski, A. O. Atli, K. Mai, B. Lucia, and N. Beckmann, "Snafu: An Ultra-Low-Power, Energy-Minimal CGRA-Generation Framework and Architecture," in *ISCA*, 2021.

[11] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "EPIMap: Using epimorphism to map applications on CGRAs," in *DAC*, 2012.

[12] S. Dave, M. Balasubramanian, and A. Shrivastava, "RAMP: Resource-aware mapping for CGRAs," in *DAC*, 2018.

[13] J. Weng, S. Liu, Z. Wang, V. Dadu, and T. Nowatzki, "A Hybrid Systolic-Dataflow Architecture for Inductive Matrix Algorithms," in *HPCA*, 2020.

[14] G. Gobieski, A. Nagi *et al.*, "MANIC: A Vector-Dataflow Architecture for Ultra-Low-Power Embedded Systems," in *MICRO*, 2019.

[15] V. Govindaraju, C. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, "DySER: Unifying Functionality and Parallelism Specialization for Energy-Efficient Computing," *IEEE Micro*, vol. 32, no. 5, pp. 38–51, 2012.

[16] Q. M. Nguyen and D. Sanchez, "Fifer: Practical Acceleration of Irregular Applications on Reconfigurable Architectures," in *MICRO*, 2021.

[17] M. Karunaratne, D. Wijerathne, T. Mitra, and L.-S. Peh, "4D-CGRA: Introducing Branch Dimension to Spatio-Temporal Application Mapping on CGRAs," in *ICCAD*, 2019.

[18] Y. Zhang, A. Rucker, M. Vilim, R. Prabhakar, W. Hwang, and K. Olukotun, "Scalable Interconnects for Reconfigurable Spatial Architectures," in *ISCA*, 2019.

[19] T. Nowatzki, N. Ardalani, K. Sankaralingam, and J. Weng, "Hybrid Optimization/Heuristic instruction scheduling for programmable accelerator codesign," in *PACT*, 2018.

[20] C. Nicol, "A coarse grain reconfigurable array (CGRA) for statically scheduled data flow computing," *Wave Computing White Paper*, 2017.

[21] X. Wu, Z. Fan, T. Liu, W. Li, X. Ye, and D. Fan, "LRP: Predictive output activation based on SVD approach for CNN s acceleration," in *DATE*, 2022.

[22] Y. Li, M. Wu, X. Ye, W. Li, R. Xue, D. Wang, H. Zhang, and D. Fan, "An efficient scheduling algorithm for dataflow architecture using loop-pipelining," *Information Sciences*, vol. 547, pp. 1136–1153, 2021.

[23] J. Fowers, K. Ovtcharov, M. Papamichael *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *ISCA*, 2018.

[24] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "HyCUBE: A CGRA with Reconfigurable Single-cycle Multi-hop Interconnect," in *DAC*, 2017.

[25] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore, "Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture," in *ISCA*, 2003.

[26] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun, "Plasticine: A reconfigurable architecture for parallel patterns," in *ISCA*, 2017.

[27] Y. Wang, W. Li, T. Liu, L. Zhou, B. Wang, Z. Fan, X. Ye, D. Fan, and C. Ding, "Characterization and Implementation of Radar System Applications on a Reconfigurable Dataflow Architecture," *IEEE Computer Architecture Letters*, vol. 21, no. 2, pp. 121–124, 2022.

[28] X. Shen, X. Ye, X. Tan, D. Wang, Z. Zhang, and D. Fan, "POSTER: An Optimization of Dataflow Architectures for Scientific Applications," in *PACT*, 2016.

[29] D. Wang, M. Ali *et al.*, "Synthetic aperture radar (SAR) implementation on a TMS320C6678 multicore dsp," Texas Instruments, 2015.

[30] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter, "NVIDIA Tensor Core Programmability, Performance amp; Precision," in *IPDPSW*, 2018.

[31] C. Lattner and V. S. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," in *CGO*, 2004.