Hardware Abstractions and Hardware Mechanisms to Support Multi-Task Execution on Coarse-Grained Reconfigurable Arrays

Taeyoung Kong, Kalhan Koul, Priyanka Raina, Mark Horowitz, and Christopher Torng

Stanford University

{kongty,kkoul,praina,horowitz,ctorng}@stanford.edu

Abstract

Domain-specific accelerators are used in various computing systems ranging from edge devices to data centers. Coarse-grained reconfigurable arrays (CGRAs) represent an architectural midpoint between the flexibility of an FPGA and the efficiency of an ASIC and are a promising candidate for servicing multi-tasked workloads within an application domain. Unfortunately, scheduling multiple tasks onto a CGRA is challenging. CGRAs lack abstractions that capture hardware resources, leaving workload schedulers unable to reason about performance, energy, and utilization for different schedules. This work first proposes a CGRA architecture that can flexibly partition key resources, including the global buffer memory capacity, the global buffer memory bandwidth, and the compute resources. Partitioned resources serve as hardware abstractions that decouple compilation and resource allocation. The compiler uses these abstractions for coarse-grained resource mapping, and the scheduler uses them for flexible resource allocation at run time. We then propose two hardware mechanisms to support multi-task execution. A flexible-shape execution region increases the overall resource utilization by mapping multiple tasks with different resource requirements. Dynamic partial reconfiguration (DPR) enables a CGRA to update the hardware configuration as the scheduler makes decisions rapidly. We show that our abstraction can help automatic and efficient scheduling of multi-tasked workloads onto our target CGRA with high utilization, resulting in 1.05x–1.24x higher throughput and a 23–28% lower latency in a multi-tasked cloud workload and 60.8% reduced latency in an autonomous system workload when compared to a baseline CGRA running single tasks at a time.

1. Introduction

Domain-specific accelerators have gained growing interest in recent years as they provide improved performance and energy efficiency over general-purpose processors. Application-specific integrated circuits (ASICs) [8, 18, 21] show the highest performance and efficiency as they are specialized for target applications such as image processing or machine learning (ML). However, the ASIC design process can span multiple years, and fixed-function accelerators quickly become obsolete as applications continue to evolve. Some works deploy applications on FPGAs [12, 16, 17]. FPGAs enable reconfiguration of the underlying hardware and can accelerate diverse workloads, but their bit-level flexibility incurs high area and energy overheads. Coarse-grained reconfigurable arrays (CGRAs) are promising architectures that lie between ASICs and FPGAs. A CGRA has arithmetic units and a routing system that are configurable in word-level granularity, providing flexibility at a lower overhead than a FPGA. With its unique advantages, a CGRA can be widely adopted in domains with high performance, power, and flexibility requirements.

As hardware accelerators are deployed in various scenarios, the demand for multi-task execution support on hardware is growing. For example, many vendors [21, 13] offer INFerence-as-a-Service, where multiple tenants share the same hardware to run inference tasks. Also, an autonomous system handles concurrent tasks to process various types of data from numerous sensors. Some works have explored multi-task execution support in ASICs and FPGAs. PREMA [11] and Planaria [14] propose a systolic array that supports multi-tenancy by temporal and spatial multiplexing, respectively. [35, 29, 34] propose an FPGA virtualization framework with multi-tenancy support. However, multi-task execution support on CGRAs has not been explored much thus far. A noteworthy exception is ChordMap [27] which schedules multiple tasks captured in synchronous data flow graphs onto a CGRA. However, it assumes that all tasks are known a priori, whereas in a multi-tenant cloud or multi-tasked edge workload scenario, tasks may arrive dynamically and require schedulers to react to maximize utilization.

Unfortunately, scheduling multiple tasks onto a CGRA is challenging as it lacks abstractions capturing hardware resources. In this paper, we propose hardware abstractions of a CGRA by partitioning key hardware resources. Both compilers and schedulers can exploit the abstrac-

tions to reason about performance, energy, and utilization. We also develop hardware mechanisms that allow fast and flexible multi-task execution on a CGRA, which schedulers exploit to improve hardware utilization. We evaluate our CGRA with two different multi-tasked workload scenarios to show the potential. Our key contributions are:

- ① We propose a CGRA architecture that can flexibly re-partition key resources, including the Global Buffer (GLB) memory capacity, the GLB memory bandwidth, and the compute resources. Specifically, we partition the GLB into GLB-slices and the tile array into array-slices, which serve as hardware abstractions. The compiler uses these abstractions for coarse-grain resource mapping, while the scheduler uses them for flexible resource allocation.
- ② We propose two hardware mechanisms to support multi-task execution on the CGRA. First, the CGRA can form a flexible-shape execution region at run time. It improves resource utilization by enabling a scheduler to allocate GLB-slices and array-slices flexibly. Second, we propose a fast-DPR method to reconfigure the underlying hardware rapidly according to scheduler decisions. It also supports run time relocation of a task to any available array-slice without software intervention.
- ③ We quantify the benefits of our proposed mechanisms on two different examples. Our CGRA with flexible execution regions and fast-DPR shows 1.05x–1.24x higher throughput and 23–28% lower latency in a cloud system scenario and 60.8% reduced latency in an autonomous system scenario than the baseline CGRA.

2. Architectural Support for Multi-Task Execution on a CGRA

In this section, we explore the architectural support needed for multi-task execution on a CGRA. Section 2.1 first introduces a baseline CGRA architecture with common features present in many reconfigurable accelerators [7, 32, 15, 6, 1, 28]. Section 2.2 then introduces how we abstract the hardware resources in the CGRA for the scheduler by partitioning the global buffer (GLB) and the tile array into GLB-slices and array-slices, respectively. We further develop hardware mechanisms that enable multi-task execution on top of these abstractions (Section 2.3), including flexible-shape execution regions and dynamic partial reconfiguration (DPR).

2.1. Baseline CGRA Architecture

Our baseline CGRA consists of a tile array with processing element (PE) and memory (MEM) tiles and a global buffer (GLB) (Figure 1). We leverage almost the same hardware configuration used in the Amber SoC [7]. The CGRA has 32x16 tiles with 384 PE tiles and 128 MEM tiles, and tiles communicate through a statically

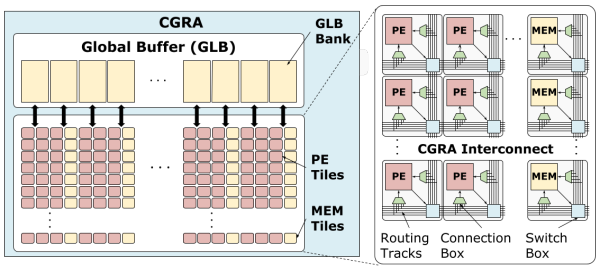


Figure 1: Baseline CGRA block diagram corresponding to [23].

App.	Task	Ver.	Tpt.	Array slices	GLB slices
ResNet-18	conv2_x	a	64	2	7
		b	256	6	7
	conv3_x	a	64	2	4
		b	256	6	4
	conv4_x	a	64	2	6
		b	256	6	6
	conv5_x	a	64	2	20
		b	128	6	20
MobileNet	conv_dw[1] _pw_2_x	a	52	2	4
		b	208	5	4
	conv_dw _pw_3_x	a	52	2	4
		b	104	3	4
	conv_dw _pw_4_x	a	52	2	4
		b	104	3	4
Camera pipeline	Camera pipeline	a	3	4	4
		b	12	6	14
Harris	Harris	a	1	2	4
		b	2	4	7
		c	4	7	14

Table 1: Variants of tasks with different resource usage and throughput. ResNet-18 and MobileNet consist of several layers, and one or more layers form a single task. The unit of throughput (Tpt.) for ResNet-18 and MobileNet is MACs/cycle and for camera pipeline and harris it is pixels/cycle.

configured mesh interconnect. A PE tile is extended from Amber version to support MAC operation. Each node in the interconnect has five incoming and five outgoing tracks in each direction, and switch boxes route data from incoming tracks to outgoing tracks. Connection boxes select data from incoming tracks and route it to the PE or MEM tile cores. The GLB consists of 32 banks, with each bank containing 128 KB of SRAM. Each GLB bank directly communicates with the tile array through IO tiles located at the top of the array.

2.2. A Scheduler-Visible Abstraction of Hardware Resources

We focus on three key hardware resources within the CGRA (Figure 1): the GLB memory capacity, the GLB

[1] A *conv_dw_pw* refers to a merged task of a depth-wise convolutional layer and a point-wise convolutional layer.

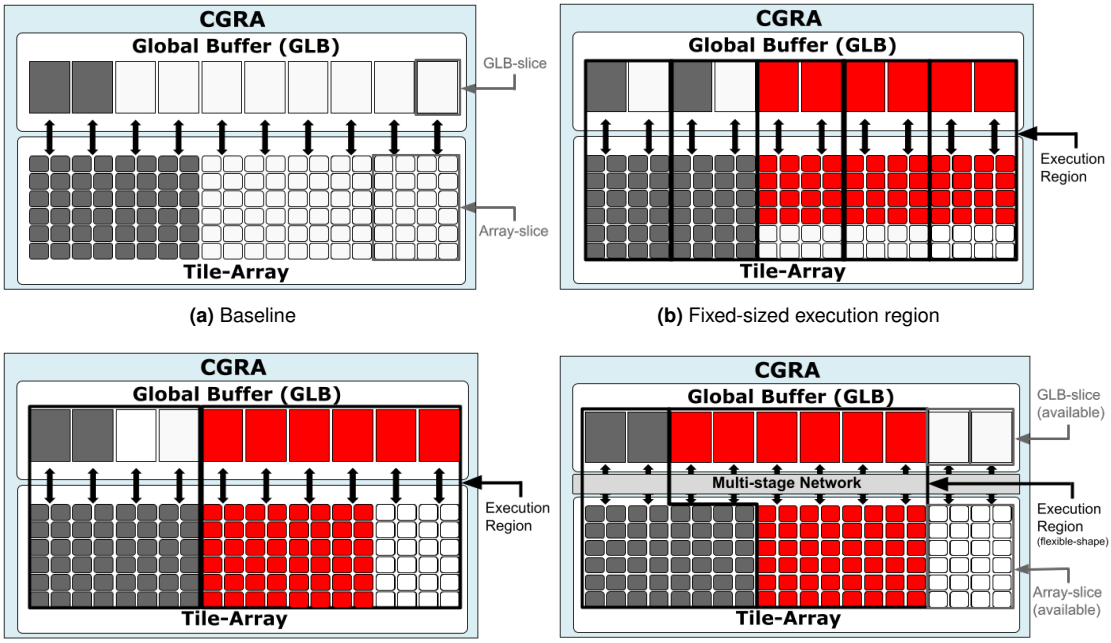


Figure 2: Resource allocation in the baseline CGRA and a CGRA with three different execution regions. Resources colored grey represent the blocks occupied by a current-running task, and those colored red represent blocks occupied by a next-running task.

memory bandwidth, and the compute resources within the tile array. When a task is compiled in the Amber toolchain [23], a compiler converts it into a dataflow graph where each node and edge represents a hardware resource and communication, respectively. Specifically, GLB banks are used for medium-sized storage and communication to the host and tile array, and PE and MEM tiles are used for computation and as small scratchpads. The dataflow graph can derive the usage of memory capacity, memory bandwidth, compute units, and throughput.

We abstract the hardware resources by partitioning the GLB and tile array into homogeneous GLB-slices and array-slices, respectively. For example, we can abstract each GLB bank within our CGRA as a GLB-slice and every set of four columns in the tile array (48 PE tiles and 16 MEM tiles) as an array-slice. This abstraction serves as a middle layer that decouples offline bitstream generation by a compiler and run time resource allocation by a scheduler. During compilation, we represent the resource usage of each task using these abstracted GLB-slices and array-slices. For instance, a *conv2_x* layer in [19] utilizes 750KB of GLB memory capacity, 17.3MB/s of memory bandwidth, 80 PE tiles, and 17 MEM tiles and achieves 64 OPs/cycle throughput at a 500MHz clock frequency. The task is abstracted as seven GLB-slices and two array-slices in coarse-grain resource slice usage. It is possible to produce variants of the same task with different resource usage and throughput by tweaking the compiler. For example, increasing the unroll factor of the same task by four would achieve 4x throughput (256 OPs/cycle) with 288 PE tiles, 33 MEM tiles, and the same GLB memory capacity and bandwidth, which is abstracted as seven GLB-slices and six array-slices. Our approach allows for pre-computation of bitstreams that support different resource usage and throughput to be cached in on-chip storage to support fast dynamic partial reconfiguration, as discussed later. Table 1 summarizes the resource usage and throughput for several different variants of tasks. At run time, a scheduler leverages the hardware slice abstraction to decide which variant of tasks to choose, which resources to allocate, and when to execute.

2.3. Hardware Mechanisms

Flexible-Shape Execution Regions. To manage multiple tasks that are concurrently running, we need a way to monitor hardware resources and the status of tasks, that are build upon the abstractions described above. We introduce an *execution region*, a sub-region of the CGRA on which a single task is mapped and executed. An execution region consists of one or more GLB-slices and array-slices. The flexibility to form different sizes and shapes of execution regions gives the scheduler a simplified and quantized view of hardware resources while providing enough information to allocate resources to each task to maximize resource utilization in multi-tasked workloads.

Figure 2 compares different mechanisms to form an execution region and how they affect resource allocation.

The blocks colored in gray represent resources occupied by the currently running task, and those colored in red represent resources allocated to the next-running task. The baseline CGRA (Figure 2a) is unaware of our hardware slice abstraction, and the entire CGRA serves as a single large execution region. Since an existing task is already mapped onto the CGRA, subsequent tasks are always forced to wait until the previous tasks finish and release the single execution region.

The simplest mechanism to form an execution region is only to support fixed-sized regions. For example, all execution regions in Figure 2b consist of two GLB-slices and one array-slice. Fixed-sized regions are not optimal. Since each task must fit within the fixed-sized execution region, the largest task with the highest resource usage determines the size. On the other hand, when there are several available execution regions, a task can be unrolled and mapped in parallel to achieve higher throughput (e.g., the next-running task is unrolled by three in Figure 2b). This method does not require much architectural change, and the implementation of a scheduling algorithm can be straightforward given the assumption that all target tasks fit within an execution region. However, although unrolling increases throughput, optimization across the unrolled dimension can be challenging to support.

Another method is to support variably sized execution regions by merging multiple fixed-sized regions. We define the unit size of a region as in the fixed-sized region case, but we can merge multiple unit regions to form a larger execution region. For example, in Figure 2c, three unit-sized regions are merged to execute the next-running task (colored in red). The benefit of variably sized execution regions is to allow compilation optimization across the unrolled dimension. For example, a *camera pipeline* task with 3 pixels/cycle throughput uses four array-slices (Table 1). Naively unrolling it by four achieves 12 pixels/cycle throughput using 16 array-slices. However, the compiler can optimize to time-multiplex PE tiles and achieve 12 pixels/cycle throughput with only six array-slices. Support for a variably sized region still allows for the pre-computation of bitstreams for multiple variants of tasks with different resource usage and throughput. However, this approach may still suffer from low resource utilization since the ratio of GLB-slices and array-slices within an execution region always remains the same.

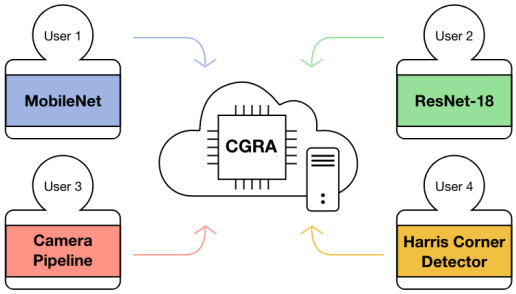
Therefore, we propose *flexible-shape execution regions* in which GLB-slices and array-slices are no longer coupled. Decoupling of GLB-slices and array-slices enables finer-grained resource allocation. For example, Figure 2d shows how an execution region can be allocated any number of GLB-slices and array-slices, forming a non-rectangular shape, with remaining array-slices and GLB-slices available to be used by other tasks. The support for flexible-shape execution regions improves resource utilization, especially for multi-tasked workloads where memory-intensive and compute-intensive tasks are mixed. However, it may require additional communication between the GLB-slices and the array-slices. In this work, we limit the placement of GLB-slices and array-slices within an execution region to be contiguous to simplify our study. Design space exploration on flexible placement support and the required network remains as future work. Section 3.1 describes the benefits of these mechanisms in more detail with a cloud system example.

Dynamic Partial Reconfiguration. Dynamic partial reconfiguration (DPR) is a mechanism to update the hardware configuration in reconfigurable architectures. We propose fast-DPR following the DPR mechanism proposed in Amber SoC [7], but with added features to exploit hardware abstractions. In Amber, every other GLB bank stores the configuration bitstreams and independently streams configuration into two columns of the tile array. Also, clocks and configuration signals are distributed down each column together, enabling reconfiguring the tile array at high clock frequency without pipeline stages. In our CGRA, we also reuse GLB blocks to store and stream bitstreams to the tile array and follow the same clock distribution network. Unlike Amber, however, one GLB bank streams configuration into one array-slice (in turn, four columns of the tile array) as an array-slice is the minimum unit of execution regions.

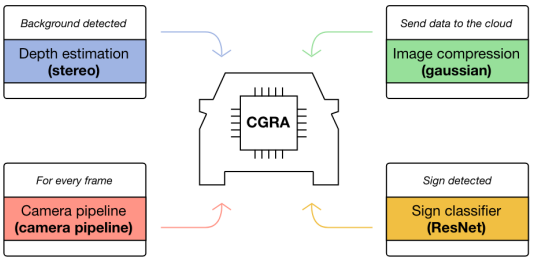
We added a feature to relocate bitstreams at run time to exploit hardware abstractions further. In Amber, the compiler generates region-aware bitstreams; the bitstreams for one region cannot be reused in different regions even though the two regions are homogeneous. This limitation comes from the fact that the address of each configuration register in different columns has a distinct column #id. On the other hand, our compiler generates region-agnostic bitstreams by assuming that the task is always mapped to the leftmost region. We also added a register indicating the destination region of DPR to GLB banks. When the host processor triggers DPR, GLB banks read the register and stream bitstreams to the target region via the network between the GLB and the tile array. With this bitstream relocation feature, a user can pre-load bitstreams of the next task to the GLB in advance and rapidly map it to any next available region just by writing to a single register.

3. Evaluation

We evaluate the benefits of multi-task execution support under two different workload scenarios. In a cloud system example scenario (Section 3.1), our CGRA with flexible-shape execution regions enables 1.05x-1.24x higher throughput and 23-28% lower normalized turnaround time (NTAT) over the baseline CGRA. In an autonomous system example scenario (Section 3.2), our CGRA enables 60.8% reduced total latency.



(a) Cloud system example



(b) Autonomous system example

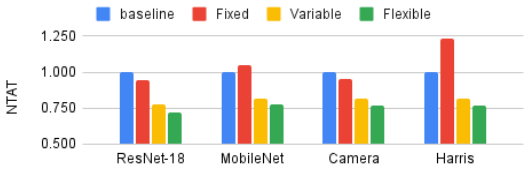
Figure 3: (a) Cloud system example scenario with four tenants submitting requests to the CGRA. Each tenant is assigned with a task from *MobileNet*, *ResNet-18*, *camera pipeline*, and *Harris*, respectively. (b) Autonomous system example with tasks that may be triggered under conditions.

3.1. Example 1: Cloud System

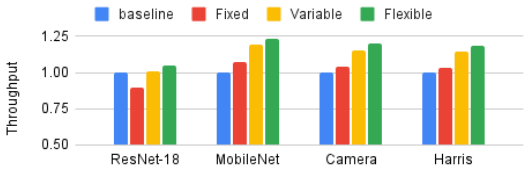
Overview. In this example, we construct a synthetic cloud computing scenario that models real-world examples in which the CGRA serves application requests from multiple users (Figure 3a). We construct the multi-tasked workload using kernels from machine learning (ML) and image processing domains, including ResNet-18 [19] and MobileNet [20] from the ML domain, and camera pipeline and Harris corner detector from the image processing domain. Table 1 summarizes the benchmark tasks and their resource requirements.

To generate the multi-tasked workload, we assume four tenants share the CGRA and are assigned one of the four target applications. Each tenant sends a request to the CGRA following a Poisson distribution. Whenever a new task arrives, or an existing task finishes, the scheduler is triggered and runs a greedy algorithm to schedule the next available task. The scheduler checks if dependencies are met before scheduling the task (e.g., in ResNet-18, *conv2_x* depends on *conv1_x*). If there is more than one version of a task that can be mapped onto the available resources, the greedy scheduler always chooses the one with the highest throughput.

Metrics. We measure *Normalized Turn-Around Time* and *throughput* to compare the baseline CGRA and the three partitioning mechanisms described in Section 2.3.



(a) NTAT



(b) Throughput

Figure 4: Evaluation in a cloud system example. (a) NTAT and (b) throughput for each task with fixed-sized, variably sized, and flexible-shape resource partitioning, normalized to the baseline CGRA. Flexible-shape partitioning decreases NTAT by 23-28% and increases throughput by 1.05x-1.24x.

Turn-Around Time (TAT) is the interval from the time of request to submit a task to the time of task completion. *Normalized Turn-Around Time* (NTAT) is the ratio of the TAT to the execution time, which represents the relative delay of a task (Equation (1) - (2)). We calculate NTAT for each request and the arithmetic average for each application. We also measure the average throughput for each application to demonstrate the performance benefit.

$$TAT = wait_time + execution_time \quad (1)$$

$$NTAT = TAT \,/\, execution_time \quad (2)$$

Results. Figure 4 illustrates the relative improvements in NTAT and throughput for flexible-shape execution regions compared to fixed- and variably-sized execution regions. Even with a simple greedy scheduling algorithm, we achieve 23–28% decreased NTAT and 1.05x–1.24x higher throughput. Note that we only pre-compile each task to two different variants in this case study (Table 1), and a scheduler greedily selects the one with higher throughput if resources are available. Co-optimizing compilation and scheduling policy may improve NTAT and throughput further, which remains future work.

3.2. Example 2: Autonomous System

Overview. In this case study, we construct a synthetic edge system scenario modeling the real world in which multiple tasks from image processing and ML domains execute in parallel and can dynamically trigger. Specifically, we develop an autonomous system scenario as described in Figure 3b following a methodology used in [30]. [2] The system takes a RAW image in Bayer encoding format (RGGB) from sensors at 30 fps and first runs a *camera*

[2]We also changed the tasks to simplify the example.

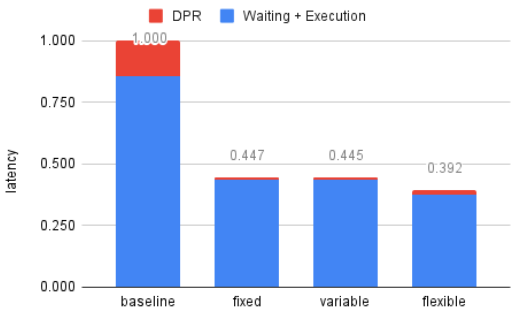


Figure 5: The average latency of an autonomous system example with different execution regions. The values are normalized to the result of the baseline. A red bar indicates the time spent for reconfiguration, and a blue bar indicates the sum of wait time and execution time. To show the benefit of fast-DPR (Section 2.3), we assume the baseline CGRA uses AXI4-Lite interface for DPR, while others use fast-DPR.

pipeline task on the CGRA to convert to an RGB image. Once the CGRA generates an RGB image, the system runs object detection and dynamically decides on the next tasks. [3] When an event happens (e.g., detection of a specific background), it processes the event and executes the corresponding tasks (e.g., *depth estimation*). Except for a *camera pipeline* that runs every frame, we set the period from one event to the next same event to follow a uniform random distribution between 3–7 frames.

Results. We evaluate the benefit of hardware resource partitioning and fast DPR by comparing our proposed CGRA to the baseline CGRA with AXI4-Lite-based DPR. Specifically, the baseline CGRA maps only one task at a time. When more than one event occurs, the baseline handles each task one by one and reconfigures using sequential AXI4-Lite configuration transactions. In the proposed CGRA with multi-task execution support, we exploit flexible-shape resource partitioning to concurrently run more than one task on the CGRA when possible. Also, we use the parallel and high-frequency DPR mechanisms in Section 2.3 to configure bitstreams. We compute the arithmetic average of the latency over all frames. As described in Figure 5, our techniques enable a 60.8% latency reduction compared to the baseline. With fast DPR, reconfiguration takes less than 5% of the total latency, an appreciable reduction from 14.4% in the baseline.

4. Related Work

As Deep Neural Networks (DNNs) are widely used in various domains, DNN accelerators [18, 17, 8, 9, 10, 25] have emerged and been deployed in the cloud system [21, 13]. To that end, many prior works have explored multi-tenancy support on DNN accelerators in cloud systems.

[3]This work assumes that object detection is executed in another hardware in the system (e.g. GPU or ASIC).

Multi-task execution support is also studied in FPGAs targeting both cloud and edge computing. However, a non-negligible portion of FPGA resources is typically reserved for controlling multi-task execution, ultimately decreasing the available computing resources. ChordMap [27] explores the automated mapping of multi-tasked applications onto a CGRA, but it is limited to mapping multiple tasks within streaming applications with all tasks known a priori. Our work proposes hardware abstractions and mechanisms, which both compilers and schedulers can exploit and co-optimize to improve resource utilization in both cloud and edge systems.

Multi-Task Execution on DNN Accelerators. Some DNN accelerators service multi-DNN tasks at the software level. AI-MT [2] and Layerweaver [31] propose a scheduling policy to mix compute- and memory-intensive tasks to increase hardware utilization. PREMA [11] implements preemptible NPUs to support multi-tenancy via temporal multiplexing. Many works add flexibility to an accelerator to accommodate multiple DNN tasks. Planaria [14] introduces a flexible systolic array with dynamic architecture fission to map multiple DNN tasks. [26] suggests a multi-directional network to support up to four DNN tasks with different dataflow. Other works [24, 3] explore a computing system with multiple DNN accelerators with different hardware characteristics. While these works only support DNN workloads, our work can support any applications that can be mapped onto a CGRA.

Multi-Task Execution on FPGAs. In FPGAs, multi-task execution support has been explored in the context of virtualization. Some works divide an FPGA into a static region, a *shell*, which serves as glue logic between the host and the FPGA, and a dynamic region, a *role*, which handles the computation of tasks. [4, 5, 33] partition a physical FPGA into several fixed-size virtual blocks and share them across multiple tasks. AmorphOS [22] presents a hardware abstraction of an FPGA, *Morphlet*, which dynamically alters its size based on resource requirements. ViTAL [35] provides a full-stack framework to run multiple tasks with different sizes on homogeneous regions. [34] supports running multi-DNN tasks on an FPGA by dividing hardware resources into multiple PE cores and spatially multiplexing them, while [30] evaluates the benefits of temporal multiplexing of FPGAs using DPR for vision applications on embedded devices. While these works only target scenarios where underlying applications change infrequently because of long reconfiguration time of FPGAs, our work can support both cloud systems and real-time edge systems due to rapid DPR.

5. Conclusion

Multi-task execution support on accelerators is becoming increasingly relevant in both cloud and edge systems and

has the potential to improve performance through better hardware utilization. This work proposes abstracting hardware resources within a CGRA into coarser-grained units with which a workload scheduler can quickly make decisions. Based on the proposed abstraction, we develop hardware mechanisms to support multi-task execution through flexible-shape hardware partitioning and high-throughput dynamic partial reconfiguration. Our evaluations modeling both a cloud and an edge system scenario suggest that the abstraction and hardware mechanisms can enable automatic schedulers to achieve high performance in multi-tasked workloads on future CGRAs.

References

[1] Giovanni Ansaloni, Paolo Bonzini, and Laura Pozzi. Egra: A coarse grained reconfigurable architectural template. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(6):1062–1074, 2010.

[2] Eunjin Baek, Dongup Kwon, and Jangwoo Kim. A multi-neural network acceleration architecture. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 940–953. IEEE, 2020.

[3] Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu. Google neural network models for edge devices: Analyzing and mitigating machine learning inference bottlenecks. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 159–172. IEEE, 2021.

[4] Stuart Byma, J. Gregory Steffan, Hadi Bannazadeh, Alberto Leon-Garcia, and Paul Chow. Fpgas in the cloud: Booting virtualized hardware accelerators with openstack. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 109–116, 2014.

[5] Stuart Byma, J. Gregory Steffan, Hadi Bannazadeh, Alberto Leon-Garcia, and Paul Chow. Fpgas in the cloud: Booting virtualized hardware accelerators with openstack. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 109–116, 2014.

[6] Fabio Campi, Antonio Deledda, Matteo Pizzotti, Luca Ciccarelli, Pierluigi Rolandi, Claudio Mucci, Andrea Lodi, Arseni Vitkovski, and Luca Vanzolini. A dynamically adaptive dsp for heterogeneous reconfigurable platforms. In *2007 Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6. IEEE, 2007.

[7] Alex Carsello, Kathleen Feng, Taeyoung Kong, Kalhan Koul, Qiaoyi Liu, Jackson Melchert, Gedeon Nyengele, Maxwell Strange, Keyi Zhang, Ankita Nayak, et al. Amber: A 367 gops, 538 gops/w 16nm soc with a coarse-grained reconfigurable array for flexible acceleration of dense linear algebra. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pages 70–71. IEEE, 2022.

[8] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 44(3):367–379, 2016.

[9] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[10] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622. IEEE, 2014.

[11] Yujeong Choi and Minsoo Rhu. Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 220–233. IEEE, 2020.

[12] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. A configurable cloud-scale dnn processor for real-time ai. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14. IEEE, 2018.

[13] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. A configurable cloud-scale dnn processor for real-time ai. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14. IEEE, 2018.

[14] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, Cliff Young, and Hadi Esmaeilzadeh. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 681–697, 2020.

[15] Graham Gobieski, Ahmet Oguz Atli, Kenneth Mai, Brandon Lucia, and Nathan Beckmann. Snafu: an ultra-low-power, energy-minimal cgra-generation framework and architecture. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 1027–1040. IEEE, 2021.

[16] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Jincheng Yu, Junbin Wang, Song Yao, Song Han, Yu Wang, and Huazhong Yang. Angel-eye: A complete design flow for mapping cnn onto embedded fpga. *IEEE transactions on computer-aided design of integrated circuits and systems*, 37(1):35–47, 2017.

[17] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 75–84, 2017.

[18] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[21] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[22] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J Rossbach. Sharing, protection, and compatibility for reconfigurable fabric with {AmorphOS}. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 107–127, 2018.

[23] Kalhan Koul, Jackson Melchert, Kavya Sreedhar, Leonard Truong, Gedeon Nyengele, Keyi Zhang, Qiaoyi Liu, Jeff Setter, Po-Han Chen, Yuchen Mei, Maxwell Strange, Ross Daly, Caleb Donovick, Alex Carsello, Taeyoung Kong, Kathleen Feng, Dillon Huff, Ankita Nayak, Rajsekhar Setaluri, James Thomas, Nikhil Bhagdikar, David Durst, Zachary Myers, Nestan Tsiskaridze, Stephen Richardson, Rick Bahr, Kayvon Fatahalian, Pat Hanrahan, Clark Barrett, Mark Horowitz, Christopher Torng, Fredrik Kjolstad, and Priyanka Raina. Aha: An agile approach to the design of coarse-grained reconfigurable accelerators and compilers. *ACM Trans. Embed. Comput. Syst.*, apr 2022. Just Accepted.

[24] Hyoukjun Kwon, Liangzhen Lai, Michael Pellauer, Tushar Krishna, Yu-Hsin Chen, and Vikas Chandra. Heterogeneous dataflow accelerators for multi-dnn workloads. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 71–83. IEEE, 2021.

[25] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. *ACM SIGPLAN Notices*, 53(2):461–475, 2018.

[26] Jounghoo Lee, Jinwoo Choi, Jaeyeon Kim, Jinho Lee, and Youngsok Kim. Dataflow mirroring: Architectural support for highly efficient fine-grained spatial multitasking on systolic-array npus.

In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 247–252. IEEE, 2021.

[27] Zhaoying Li, Dhananjaya Wijerathne, Xianzhang Chen, Anuj Pathania, and Tulika Mitra. Chordmap: Automated mapping of streaming applications onto cgra. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(2):306–319, 2022.

[28] Leibo Liu, Dong Wang, Min Zhu, Yansheng Wang, Shouyi Yin, Peng Cao, Jun Yang, and Shaojun Wei. An energy-efficient coarse-grained reconfigurable processing unit for multiple-standard video decoding. *IEEE Transactions on Multimedia*, 17(10):1706–1720, 2015.

[29] Joel Mbongue, Festus Hategekimana, Danielle Tchuinkou Kwadjo, David Andrews, and Christophe Bobda. Fpgavirt: A novel virtualization framework for fpgas in the cloud. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 862–865. IEEE, 2018.

[30] Marie Nguyen, Robert Tamburo, Srinivasa Narasimhan, and James C Hoe. Quantifying the benefits of dynamic partial reconfiguration for embedded vision applications. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 129–135. IEEE, 2019.

[31] Young H Oh, Seonghak Kim, Yunho Jin, Sam Son, Jonghyun Bae, Jongsung Lee, Yeonhong Park, Dong Uk Kim, Tae Jun Ham, and Jae W Lee. Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 584–597. IEEE, 2021.

[32] Artem Vasilyev, Nikhil Bhagdikar, Ardavan Pedram, Stephen Richardson, Shahar Kvatinsky, and Mark Horowitz. Evaluating programmable architectures for imaging and vision applications. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[33] Jagath Weerasinghe, Francois Abel, Christoph Hagleitner, and Andreas Herkersdorf. Enabling fpgas in hyperscale data centers. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pages 1078–1086, 2015.

[34] Shulin Zeng, Guohao Dai, Hanbo Sun, Kai Zhong, Guangjun Ge, Kaiyuan Guo, Yu Wang, and Huazhong Yang. Enabling efficient and flexible fpga virtualization for deep learning in the cloud. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 102–110. IEEE, 2020.

[35] Yue Zha and Jing Li. Virtualizing fpgas in the cloud. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 845–858, 2020.