Received 21 August 2023, accepted 16 September 2023, date of publication 26 September 2023, date of current version 29 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3319404

RESEARCH ARTICLE

3DRA: Dynamic Data-Driven Reconfigurable Architecture

JINHO LEE, (Member, IEEE), BURIN AMORNPAISANNON, (Graduate Student Member, IEEE), ANDREAS DIAVASTOS, AND TREVOR E. CARLSON, (Senior Member, IEEE)
Department of Computer Science, National University of Singapore, Singapore 117417

Corresponding author: Trevor E. Carlson (tcarlson@comp.nus.edu.sg)

This work was supported by the Singapore NRF under Grant NRF2018NCR-NCR002.

ABSTRACT Specialized accelerators are becoming a standard way to achieve both high-performance and efficient computation. We see this trend extending to all areas of computing, from low-power edge-computing systems to high-performance processors in datacenters. Reconfigurable architectures, such as Coarse-Grained Reconfigurable Arrays (CGRAs), attempt to find a balance between performance and energy efficiency by trading off dynamism, flexibility, and programmability. Our goal in this work is to find a new solution that provides the flexibility of traditional CPUs, with the parallelism of a CGRA, to improve overall performance and energy efficiency. Our design, the Dynamic Data-Driven Reconfigurable Architecture (3DRA), is unique, in that it targets both low-latency and high-throughput workloads. This architecture implements a dynamic dataflow execution model that resolves data dependencies at run-time and utilizes non-blocking broadcast communication that reduces transmission latency to a single cycle to achieve high performance and energy efficiency. By employing a dynamic model, 3DRA eliminates costly mapping algorithms during compilation and improves the flexibility and compilation time of traditional CGRAs. The 3DRA architecture achieves up to 731 MIPS/mW, and it improves performance by up to 4.43x compared to the current state-of-the-art CGRA-based accelerators.

INDEX TERMS Reconfigurable architectures, coarse-grained reconfigurable array, CGRA, accelerators, dynamic dataflow.

I. INTRODUCTION

While performance improvements of modern processors have continued with each new CPU generation, the rate of improvement over time has not kept pace with the performance seen in previous decades [1], [2], [3]. To overcome some of the limitations of traditional designs, specialized accelerators have started to become the norm in high-performance systems. State-of-the-art accelerators aim to achieve high performance with low-cost, energy-efficient designs. But, achieving these needs often sees sacrifices in flexibility, programmability, or both.

Several specialized accelerators provide high performance and energy efficiency for specific applications such as Graphics Progressing Units (GPUs) [4], AI Accelerators [5], [6], [7], Digital Signal Processors (DSPs) [8], graph processors [9], [10], and cryptography accelerators [11]. Unfortunately, these works tend to target single-workload classes and are unable to accelerate a broad set of applications. To provide both high-performance and energy-efficient designs for a wide range of applications, researchers have been investigating reconfigurable architectures such as Field Programmable Gate Arrays (FPGAs) and Coarse-Grained Reconfigurable Arrays (CGRAs) [12], [13], [14]. However, building a processor that is both flexible and easy to program while maintaining performance and efficiency has proven to be a difficult task.

Fast and efficient FPGA development requires a deep understanding of the specific FPGA architecture to achieve the frequency or performance targets needed. While High-Level Synthesis (HLS) tools [15] can speed FPGA development, reaching specific performance targets requires a trial-and-error approach to select the `#pragma` options that show the best performance. CGRAs offer an alternative approach, using components at a larger granularity (functional-unit level) than FPGAs. However, finding a

The associate editor coordinating the review of this manuscript and approving it for publication was Ludovico Minati.

mapping for a given Control Data-Flow Graph (CDFG) based on Modulo Scheduling [16] is known to be NP-Complete [17], [18], which limits their scalability and applicability. More importantly, typical compilation for these statically-scheduled CGRA hardware designs can take many hours [12].

These long compilation times, and general difficulty in CGRA compilation, stem mainly from limited connectivity between processing elements. More specifically, finding optimal (single-hop) data transfer routes between instructions is difficult, and even impossible for some applications, due to the limited connectivity of mesh networks that are typically used in CGRA implementations. To make matters worse, when there exist high fan-out instructions, communication resources such as wires and ports can be easily overwhelmed, increasing data transfer latency. In this situation, it is crucial to minimize the physical distance between dependent instructions to optimize routes and achieve high performance by minimizing the delivery latency.

Typically, a CGRA maps multiple instructions at each processing element. When dependent instructions are mapped at the same location, the latency and resource usage can be minimized. However, this technique can cause input contention or output contention as the instructions compete to use the same input and output ports. In addition, support for dynamic events in modern CGRA architectures is not typically provided. Dynamic events are those that are not resolved in a pre-determined amount of time. For example, a cache miss, versus a hit, could take a significantly longer time before the data is returned. The compiler for a CGRA, in the case of a statically scheduled architecture, needs to use the worst-case timing of these dynamic events to guarantee correctness. Thus, an instruction that can take a variable amount of time to execute, such as memory access, has to be scheduled assuming the longest possible memory access time. This can add significant latency to the computation, reducing performance.

In this work, we propose a new architecture, called the 3DRA, or the Dynamic Data-Driven Reconfigurable Architecture, that aims to overcome the limitations of traditional CGRAs that include the complexity of compilation, lack of the ability to efficiently handle dynamic events that take variable time, and the performance loss due to sub-optimal data delivery latency. The goal of this new architecture is to increase flexibility and reduce the application development effort for traditional CGRAs, while at the same time improving their performance and energy efficiency. The 3DRA accelerator employs dynamic data-driven execution and enables fast all-to-all connectivity between Processing Elements (PEs) to allow for more flexible and aggressive scheduling that significantly reduces latency and compilation time. Dynamic data-driven execution also improves programmability, as the programmer and the compiler are no longer responsible for statically defining the scheduling and the data dependency resolution of the instructions.

The proposed design implements a single-cycle, all-to-all bypass network to connect all the processing elements in the accelerator. This significantly reduces latency and improves data transfer efficiency as it eliminates contention events when multiple processing elements are trying to send data through the network. More importantly, the new hardware design proposed in this work overcomes the limitations of CGRA instruction mapping, as each processing element can now directly send data to any other processing element in the accelerator.

Below, we list the main contributions of this work:

- A reconfigurable architecture, called the 3DRA, that supports dynamic data-driven events and a single-cycle all-to-all communication network to (a) solve the extremely difficult programming challenge of CGRAs, (b) minimize data transfer latency, (c) achieve efficient dynamic adaptability to various workloads, and finally (d) minimize contention due to instruction PE placement and high-fan-out instructions;
- A detailed analysis of this new architecture, showing that the 3DRA achieves up to a 4.43x performance improvement over state-of-the-art CGRA accelerators and throughput of up to 13,748 MIPS while achieving power-efficiency of 731 MIPS/mW and an area-efficiency of 35,709 MIPS/mm^2.

II. BACKGROUND

CGRAs, or Coarse-Grained Reconfigurable Arrays [12], [19], [20] are a type of accelerator from a category between Application-Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs) in terms of configurability and performance. They consist of a collection of functional units, memory units, and control logic, interconnected by a statically programmable fabric. This fabric allows the functional units to be dynamically connected and reconfigured based on the specific computational requirements of the application. CGRAs usually target high-performance computing kernels that consume a significant amount of time from an application and accelerate execution by providing a high level of parallelism. They are most commonly used in digital signal processing and multimedia processing applications.

Overall, CGRAs offer a good balance between performance and configurability, making them well-suited for specific types of computationally intensive applications. However, they tradeoff programmability for increased scalability. In CGRAs, the interconnection of functional units commonly employs a mesh network topology, wherein the units iteratively execute instructions that are stored within their configuration memories, as orchestrated by the compiler. The role of the compiler extends beyond instruction placement, to include the determination of both the data transfer pathway through the network and the associated transfer timing.

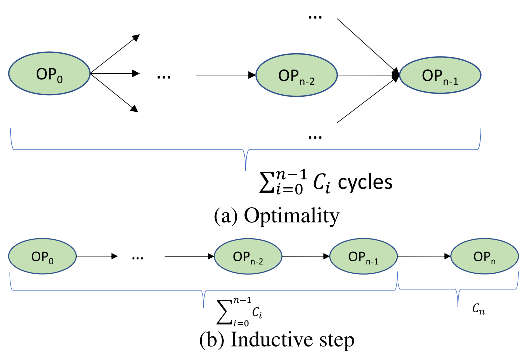


FIGURE 1. Example sub-tree. The path between OP_0 and OP_{n-1} is the longest path containing OP_n.a.

The compilation complexity in CGRAs grows exponentially as it is affected by factors such as the size of the target program and the number of functional units involved [21], [22]. This does not allow the offloading of large workloads, forcing users to accelerate only small inner loops in order to maximize performance and efficiency. Considering just the temporal aspect of the execution schedule, that is the data transfer timings through the interconnection network, the process of compilation for a CGRA is still complex even for simple workloads, and is in fact an NP-complete problem [17], [18].

Instead, in this work, we target fast communication between PEs. We accomplish this by deploying a fully-connected network that dynamically constructs the execution schedule based on the input Control Data-Flow Graph (CDFG). This eliminates the need for temporal knowledge of instruction execution and data transfer timings, leading to streamlined data communication, improved programmability with simplified compilation procedures, and ultimately higher performance and efficiency.

III. PROBLEM FORMULATION

Apart from simplifying the mapping and the execution of a CDFG, with 3DRA, our aim is to propose an accelerator that approaches optimal performance. In this section, we first define optimality in processing a CDFG and define the requirements that need to be met.

A. OPTIMAL DATAFLOW EXECUTION

Assuming a single-cycle latency for each compute operation (OP) in a dataflow graph, for each operation, the longest path to reach any operation requires n operations including itself. We therefore consider it to be the *optimal* processing when it is completed in $\sum_{i=0}^{n} C_i$ cycles, where it takes C_i cycles to process OP_i (as shown in Figure 1a).

Assuming this definition, to prove that an accelerator can guarantee optimality, we must show that for the base case, the inductive step holds. For the base case, the operations without a predecessor can be executed immediately using constant values without any delay. To show that the inductive step holds, we must show that a spatial accelerator can complete the execution of OP_n in Figure 1b in $\sum_{i=0}^{n} C_i + C_n$. In other words, the accelerator must transfer data between OP_{n-1} and OP_n without delay and execute OP_n as soon as the operands arrive.

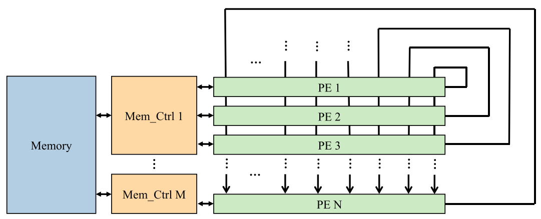


FIGURE 2. The high-level design of a 3DRA accelerator.

B. MINIMIZING DATAFLOW OVERHEADS

While optimality is the goal of this work, several constraints make this task difficult. We first analyze the causes of each source of delay, and then demonstrate how our microarchitecture can minimize each of these overheads.

Such data transfer delays can occur in three places: (1) inside the *network*, (2) at the *receiving PE*, or (3) at the *sending PE*. When data is transferred from a sender to a receiver through (1) the *network*, the delay can be affected by the distance between operations, the hop count, and also whether there is network contention. To minimize the network delay, a new type of network can be designed where the data can be sent to the destination immediately, regardless of the location or the activities of other PEs. When (2) a *receiving PE* takes data from multiple senders, depending on the number of input ports, they might not be processed at the same time. For example, if a PE has a single input port and processes an addition operation with two inputs, it will require at least two cycles to receive the operands. It can take even longer if multiple instructions are mapped to a single PE and they both compete for the use of the input ports. This type of delay caused by limited ports can happen at (3) a *sending PE*'s side too. When a sender has multiple receivers, it can take multiple cycles to generate and send out packets for them.

To minimize these delays caused by the *network*, in our proposed 3DRA design, we introduce a fully connected network using uni-directional broadcast channels, where we guarantee that the receiving PE will always receive an input packet in a single cycle. To eliminate delays at the *receiving PE* we allow input operands to be received simultaneously without contention at the input ports. Finally, to eliminate delays at the *sending PE*, we broadcast its output only when all of its receivers are ready to receive.

IV. THE 3DRA DESIGN

In this section, we describe the design and execution model of the proposed Dynamic Data-Driven Reconfigurable Architecture (3DRA). This accelerator design aims to optimize the computationally intensive inner loops that tend to dominate the execution time of many kernels. More specifically, we focus on minimizing communication delays

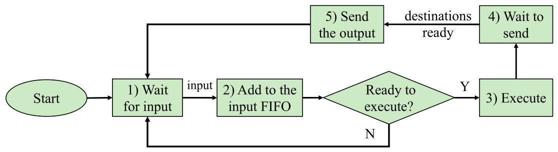


FIGURE 3. The execution model of a 3DRA processing element (PE).

and improving programmability by removing potential input or output contention while also reducing network latency.

A. PROPOSED ARCHITECTURE

Figure 2 shows the high-level design of a 3DRA accelerator. It consists of multiple Processing Elements (PEs), memory controllers, and a single local memory space (scratchpad) that operates on 32-bit data. During execution, a PE operates on a single instruction assigned by the compiler. When the operands of the instruction are produced and delivered by its producers, it executes the instruction and sends the output to the appropriate consumer if the receiver is ready to consume it. All these steps are done dynamically, alleviating the burden of compiler-driven orchestration typically associated with conventional CGRA architectures. As a result, the compilation process is simplified to a straightforward 1-to-1 instruction-to-PE mapping step.

The PEs are connected using an all-to-all bypass network that allows for single-cycle broadcast communication between any two communication points. The all-to-all communication is implemented using uni-directional broadcast channels in which the data transfer delay is minimized. The PEs are separated and allocated into clusters, with each cluster assigned a dedicated memory controller. Each memory controller incorporates an arbiter to handle multiple requests. To facilitate communication between PEs belonging to different clusters, the PE producing the data to be transmitted will broadcast its data to all other clusters.

1) DYNAMIC DATA-DRIVEN EXECUTION

3DRA and more specifically, its processing elements (PEs), follow a dynamic data-driven execution model. Figure 3 shows the execution flow of a processing element. Instructions are assigned to PEs in such a way to avoid the potential input and output contention that can happen when multiple instructions are mapped to the same PE. Each PE waits to receive input data that can be from other producing PEs. When the input arrives at the PE, it is first stored to the input FIFO. At the same time, the PE checks whether all of the operands needed have arrived. Similary to a dataflow machine, the execution of instructions relies exclusively on the availability of input operands and is determined between steps 2 and 3 in Figure 3. Furthermore, it does not impose constraints on the execution duration of any instruction, thus enabling 3DRA to handle instructions that exhibit varying time requirements, a capability that is absent in conventional CGRAs.

Once all of the operands are ready, the PE can start computing. After the execution of the instruction, the data is stored in the output register of the PE until all of its destination (consuming) PEs are ready to receive it. This is determined by the ready-to-send signal. Note that, all steps in 3DRA are pipelined to increase instruction throughput.

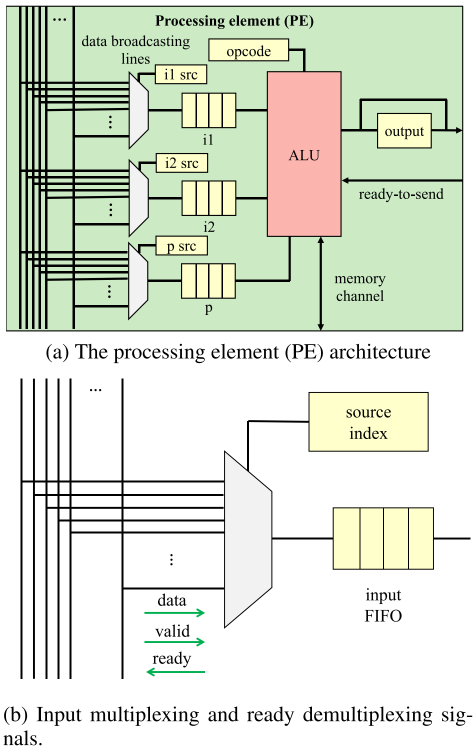


FIGURE 4. A PE receives and stores its input operands, in the input FIFO queues, through the data broadcasting lines. The executing instruction is stored in the opcode register. The result is stored in the output register and broadcast to all receiving PEs when ready.

2) PROCESSING ELEMENT DESIGN

Figure 4a presents the architecture of the processing element (PE). Each PE is configured based on the application that will be accelerated, and each instruction from the loop that is to be accelerated is assigned to a dedicated PE. During the configuration phase, the PEs, the input sources, and the operation types are specified. As highlighted in Section IV-A1, this approach helps mitigate the issues related to input contention, output contention, and programming complexity. The PEs are connected to each other using uni-directional data broadcasting lines that implement an all-to-all bypass network. This enables fast, contention-free communication between dependent PEs.

A PE is designed to handle an instruction where the source operands are specified in the registers connected to the data broadcasting lines, and the operation in the opcode register is executed when the source operands are ready in the input FIFOs. The output of the arithmetic logic unit (ALU) is stored in the output register and sent out when the ready-to-send signal is set to high; this indicates that all target PEs to receive the output are ready. If the computation happens when the ready-to-send signal is set to high, the result is immediately sent out, bypassing the output register. This enables low-latency communication with other functional units and PEs.

Figure 4b shows how the data broadcasting lines within a PE handle communication using valid and ready signals. A PE sends a ready signal to its source PE specified by the source index register that is initialized during the reconfiguration phase. By default, all ready signals of a PE are set to high. The ready signal of a source PE changes to low only if the input FIFO is full. A multiplexer connected to an input FIFO queue uses the source index register to select the source operands of the instruction to be executed. When the valid signal is set to high, the incoming data is queued into the input FIFOs (*i1* for operand 1, *i2* for operand 2, or *p* for predicate inputs). When the ALU is idle, each FIFO is checked for available data in every cycle.

As soon as all the operands arrive in the FIFOs, they are sent to the ALU to execute the instruction denoted by the opcode register. An ALU supports various operations including multiplication and division. The multiplier and the divider are pipelined to reduce the timing delay.

3) SINGLE CYCLE, NON-BLOCKING BROADCAST COMMUNICATION

We minimize the communication latency by implementing an all-to-all bypass network through uni-directional data broadcasting lines that allow for parallel, low-latency data exchange. All PEs send their output to all receiving PEs at the same time when all destinations are ready to receive data. Having a direct connection to every PE not only reduces transfer latency but also enables 3DRA to efficiently handle high fan-out instructions, with numerous destination instructions. Regardless of the number of transfers, an instruction can send its output to all of its destinations in a single cycle. Whereas, in traditional CGRAs using a mesh network, only a single packet can be sent at a time, increasing the data transmission latency. To check if all destinations are ready to receive, all of the ready signals through the data broadcasting lines are reduced as shown in Figure 6 in every cycle. Then, when all the ready signals are set to high, the source PEs are immediately informed by the AND reduction module to send its data. This method simplifies the scheduling process as all data transfers are handled dynamically and guarantee the incoming input data order and correct execution synchronization. The data arrive at the destination immediately, making sure they are not received out-of-order, which could be the case for packet-switched networks.

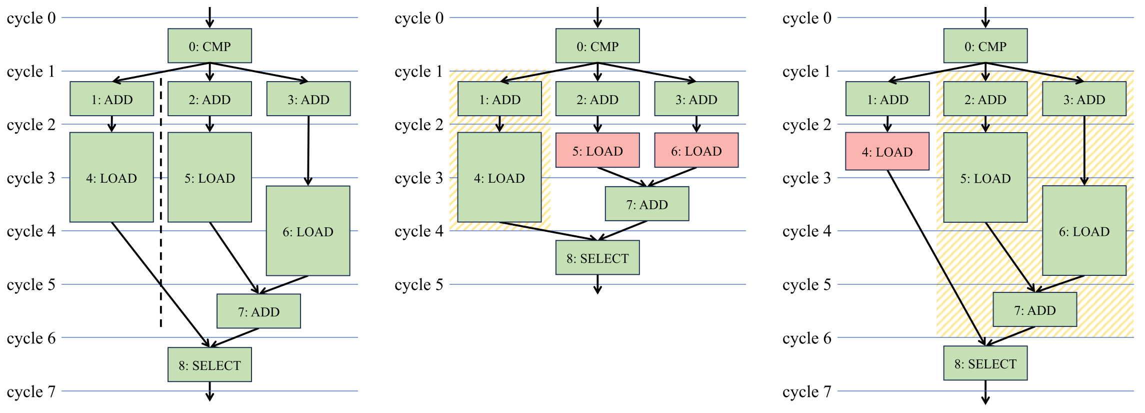
4) PREDICATION AND OPTIMIZING MEMORY OPERATIONS

The 3DRA implements predication using comparison (CMP) and selection (SELECT) instructions. CMP checks the equality of two operands and sends True or False to all receivers (instructions in both directions of a branch). CGT (greater than) and CLT (less than) instructions also work similarly. The receivers of the CMP operation check the validity of its CMP output and mark it in the output value using an extra bit (the validity bit). Instructions in both valid and invalid paths of the branch will execute regardless of the value of the outcome of the CMP instruction. However, the SELECT instruction will choose the correct output of the branch sequence that is marked as VALID.

When executing a load instruction, a request is sent to the memory controller through the memory channel and waits for the response to return. When the response arrives, the data is fetched and stored in the output register of the corresponding PE until the ready-to-send signal is set to high. To optimize the execution in the 3DRA and save resources such as time, memory ports, and power, invalid memory operations (LOAD instructions in the wrong path of a branch) are dropped dynamically and a dummy value is sent to the consuming instructions in the next cycle. This reduces the cost of unnecessary memory access to just a single cycle, while the dynamic nature of 3DRA will handle the changes in the scheduling as dependencies are resolved at runtime. To improve utilization, store instructions are executed by sending a write request to the memory controller without waiting for a response.

Figure 5 shows an example that illustrates how predication and dynamic memory accesses are optimized to improve performance in 3DRA when the LOAD instructions use a single port. The example in Figure 5a assumes a ternary operation `(condition ? a[i]: b[i] + c[i])` is mapped on a spatial architecture and statically programmed. When the output of the CMP instruction is True, the address of `a[i]` is calculated by the ADD (instruction #1), and the validity bit is set to VALID. The value is then read by the LOAD (instruction #4). Otherwise, when the output of CMP is False the LOAD instruction #4 (depending on the implementation of the CGRA) can send a dummy output attached with INVALID bit without actually fetching the data. In this case, LOAD instructions #5 and #6 are executed and send out the correct outputs along with a VALID signal. Next, the SELECT instruction (#8) picks a valid input operand between the outputs of instructions #4 and #7. Even though the LOAD instructions are not executed, given a statically mapped execution environment, the timing of instructions is predefined at compile time and cannot be changed at runtime. Therefore, in a traditional CGRA, the instructions (#7 and #8) should wait for all of the source operations (the worst-case execution latencies from all paths) to be completed before starting execution. This wastes valuable cycles when the valid path is the one on the left in Figure 5a.

In 3DRA, by dynamically handling the predication and memory accesses, the hardware naturally minimizes the execution time, saving the wasted cycles seen by static spatial architectures. Figure 5b, shows 3DRA's execution flow when the CMP instruction (#0) is True. In this case, the LOADs in the invalid direction (instructions #5 and #6) can be dropped completely, but they still have to send an output for their consumers to continue execution. Thus, they send dummy output values to ADD (#7) without accessing the memory. This way, it takes only 6 cycles to complete, while



(a) *Optimized statically programmed execution without data transfer delay.* Instructions on the left-hand side of the dotted line are valid only if the output of the CMP instruction (Instruction #0) is True, while instructions on the right-hand side are valid only if the output is False. Optimized static mappings [12] can detect that only a single instruction will execute from instructions #4 and #5, allowing them to be mapped in the same cycle despite the system having only one memory port.

(b) *3DRA's dynamic predication handling when the output of CMP is True (T).* The instructions in the dashed box on the left are valid, and only LOAD #4 is executed. 3DRA uses the predication value to drop the LOAD instructions that don't need to execute (#5 and #6) and leverages its dynamic scheduling capabilities to execute instructions #7 and #8 earlier. In this example, we save 2 cycles from not executing the corresponding LOADs.

(c) *3DRA's dynamic predication handling when the output of CMP is False (F).* The instructions in the dashed box on the right are valid, and LOADs #5 and #6 are executed in consecutive cycles as the memory controller can pipeline the requests but can only issue one memory operation per cycle. In this case, the execution time of 3DRA is the same as an optimized statically programmed execution.

FIGURE 5. Dynamic predication handling example when executing a ternary operation: *condition? a[i]: b[i] + c[i]*. The predication bit is sent as data in the dataflow graph. The example above assumes a single port, pipelined memory configuration where LOADs take 2 cycles to fetch the data. Note that, in a statically mapped CGRA, each instruction is programmed to execute in a predefined cycle (calculated by the compiler) regardless of the outcome of the condition (CMP instruction). They are only validated at the end of the block with the SELECT instruction, where the results of those marked as invalid (incorrect path of the branch) will be ignored.

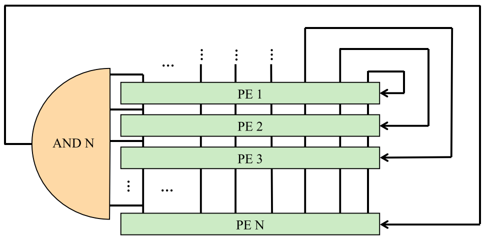


FIGURE 6. Ready signal reduction: Each PE is coupled with one AND reduction module. In every cycle, an AND module checks if all of its destination PEs are ready by reducing their ready signals in the Data Broadcasting Lines and informs its coupled PE to send its data.

it takes 8 cycles for a statically programmed execution. If the CMP (instruction #0) is False (Figure 5c), the following LOADs (instructions #5 and #6) will be executed in the same manner as the statically programmed execution, taking the full 8 cycles.

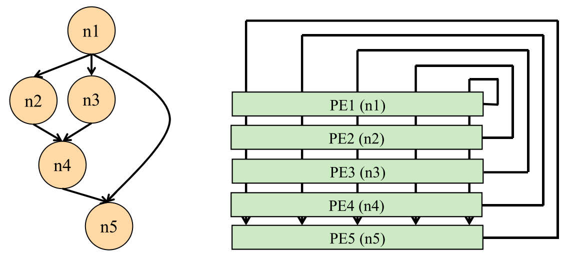
B. PROGRAMMABILITY

Programming the 3DRA is simpler than mapping to traditional CGRAs, as our design relies on dynamic execution that does not require the user to explicitly define the execution schedule. Configuring a PE only requires the source operands' indices to be used for receiving data from the appropriate source PEs and the instruction opcode that will define the operation it will execute. Taking into account the memory access latency of each memory operation is also not required in the proposed implementation. Instead, the 3DRA hardware determines when to compute and transfer data dynamically using the ready-to-send signals. In addition, because all PEs are homogeneous and the data transfers are handled through a non-blocking broadcast, the instructions can be placed in any PE. This eliminates costly optimization phases over resources and routing paths between instructions found in traditional CGRAs. Finally, this design builds on the operation-level reconfigurability of a CGRA to allow for fast reconfiguration.

C. WALK-THROUGH EXAMPLE

To demonstrate how the 3DRA accelerates the execution of a loop, we present an example in Figure 7. For this example, we use a simple loop of 5 instructions with dependencies depicted in the Dataflow Graph (DFG) in Figure 7a. Each instruction in this example takes 1 cycle to complete. Because the 3DRA is using an all-to-all broadcast bypass network, we simply map instructions to the PEs sequentially as shown in Figure 7b. Note that, since each PE receives a dedicated instruction to execute, the number of PEs must be equal to or more than the number of instructions to be executed. While we currently require a 1-to-1 mapping for instructions to PEs, extending the 3DRA architecture to support multiple instructions per PE is possible, but beyond the scope of this current work.

Figure 7c shows the execution flow of a 3DRA accelerator with 3-entry input FIFO queues, with each block representing the input data received in each cycle. For example, in cycle 2, PE2, PE3 and PE5 have already received the output from



(a) Example DFG (b) Sequential mapping on 5 PEs

	PE 1	PE 2	PE 3	PE 4	PE 5
cycle 1	🟩				
cycle 2	🟧	🟩 n1	🟩 n1		n1
cycle 3		🟧 n1	🟧 n1	🟩 n2, n3	n1
cycle 4				🟧 n2, n3	🟩 n4
cycle 5					🟧 n4
cycle 6					

🟩 Iteration 1 🟧 Iteration 2 Op Arriving operand

(c) Execution flow with 3-entry input FIFO queues

	PE 1	PE 2	PE 3	PE 4	PE 5
cycle 1	🟩				
cycle 2		🟩 n1	🟩 n1		n1
cycle 3				🟩 n2, n3	
cycle 4					🟩 n4
cycle 5	🟧				
cycle 6		🟧 n1	🟧 n1		

🟩 Iteration 1 🟧 Iteration 2 Op Arriving operand

(d) Execution flow with single-entry input FIFO queues

FIGURE 7. **Mapping a sample DFG on 3DRA with 5 PEs and exposing loop level parallelism. The PEs firing computations are colored and the arriving operands are specified with the source node id.**

PE1 that executed instruction $n1$. In the next clock cycle, $n2$ (PE2) and $n3$ (PE3) send their outputs to $n4$ (PE4) in parallel, as supported by our broadcast network. The execution then continues until one complete iteration of the DFG is complete.

Since the example is a loop, the execution pattern is repeated, and assuming that each iteration is independent of the rest, the execution of subsequent iterations is pipelined and can start as soon as the executing resources are available. This allows the 3DRA to exploit the loop-level parallelism present in the application. To support this, however, it is necessary to have enough entries in the input FIFO queues to allow PEs that have finished the current loop iteration instruction to move to the next iteration and store their output in empty slots in the input FIFO queues of their consuming PEs. The benefit of this is depicted in Figure 7d, where we use single-entry input FIFO queues. Because a PE only sends its output if all of the destinations are ready to receive (*i.e.* have an empty slot in their input FIFO queue), a single-entry FIFO queue would block future iterations from being executed until that PE executes its instruction. In our example, PE1 can start executing the next iteration only after PE2, PE3, and PE5 consume the $n1$ output, which happens in cycle 4. Therefore, when using a single-entry input FIFO queue, the next iteration can only start in cycle 5.

TABLE 1. **Benchmark characteristics.**

Benchmark	# Nodes	# Edges	Domain
Conv2d	59	72	Deep learning
CRS	27	33	Graphic processing
ellpack	41	50	Graphic processing
GeMM	61	77	Linear algebra
stencil2d	27	33	Image processing

V. EXPERIMENTAL SETUP

We implemented the 3DRA in Chisel 3 [23] and synthesized it using Cadence Genus targeting a commercial 22 nm technology node. We use Synopsys VCS-MX v2015.09 for simulation, Synopsys PrimePower v2019.03 for power consumption measurements, and Cadence Innovus for place and route. We apply clock gating and power gating to 3DRA. We use the average switching activities across all the benchmarks in Table 1 to calculate 3DRA's power consumption. Unless otherwise specified, all comparisons are performed using synthesized power and area results as a fair comparison with REVAMP [20]. Table 2 outlines the configurations we evaluated in this study. We evaluated 3DRA with kernels of benchmarks which are selected in the state-of-the-art CGRA implementation [20] from various application domains (see Table 1). To match the precision and have a fair comparison, the instruction set and the data widths are the same with REVAMP [20]. In addition, we use identical input control- and data-flow graphs (CDFGs).

VI. EVALUATION

A. OVERALL IMPROVEMENTS

In our experiments, the 3DRA delivers up to 4.43x higher instructions per cycle (IPC) over the current state-of-the-art CGRA implementation (REVAMP [20]), due to our minimized communication overheads and the deeply pipelined execution using the input FIFO queues. In REVAMP, multiple instructions are placed in each PE while 3DRA uses more PEs and maps only a single instruction per PE. By increasing the number of PEs, we maximize the number of instructions executed in parallel. Finally, we show up to 731 MIPS/mW of power efficiency, which is again higher than the baseline.

B. QUALITY OF EXECUTION SCHEDULES

Figure 8 shows the impact of the FIFO size on a 64 PE 3DRA implementation with respect to Instructions Per Cycle (IPC). To highlight the benefit of dynamic data-driven execution, the 3DRA is compared with the state-of-the-art CGRA, REVAMP [20]. In such static designs, performance is calculated using Initiation Intervals (II) which is defined as the time gap between two consecutive iterations. Then, the iterations are repeated in every II cycles. For this comparison, its IPC is calculated as $\#Nodes/II$.

We observe that the impact of FIFO size is significant. The 3DRA tends to show higher IPC values than the baseline

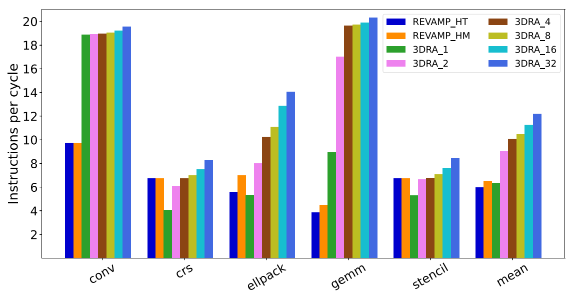


FIGURE 8. Instructions per cycle (IPC) comparison against a state-of-the-art CGRA implementation, REVAMP [20]. HT means that it uses heterogeneous PEs while HM uses homogeneous PEs. 3DRA is using 64 PEs for these results and the FIFO size is labeled at the end (3DRA_8 indicates an input FIFO size of 8 entries for this configuration). The mean represents the harmonic mean across benchmarks. When the FIFOs have more than 2 entries, 3DRA shows a higher mean IPC than REVAMP. In addition, when the FIFO size is greater than or equal to 4, 3DRA shows higher IPC in all benchmarks. The 3DRA accelerator shows up to a 4.43x IPC improvement over the homogeneous version when the FIFO size is fixed to 16.

and improves as the FIFO size grows. However, it can be slower than the baseline (e.g., *CRS* and *stencil2d*) when the FIFO size is small because the size of the queue defines the loop-level parallelism that can be exploited. Due to the nature of the broadcast communication network used in 3DRA, a single-entry FIFO queue would force the entire loop to execute before allowing the next iteration to start execution. Increasing the number of entries in the FIFO queue allows for more data to be stored at the same time and consequently additional iterations to be pipelined in 3DRA (as described in the example of Figure 7).

C. NUMBER OF PROCESSING ELEMENTS

As a PE handles a single instruction, the maximum IPC is limited by the number of PEs. In the workloads characterization study [19], it is pointed out that the regions that take about 90% of the execution are composed of 51 to 264 instructions. By having up to 256 PEs, 3DRA can cover most of the highly repeated regions. In addition, all of the benchmarks used in the state-of-the-art CGRA [20] can be covered with 64 PEs. Thus, three versions of 3DRA having 64, 128, and 256 PEs are evaluated in this work.

In Table 2, the impact of the number of PEs and frequency on power and area is shown where the FIFO size is fixed to 16 entries. When the number of PEs grows, the frequency drop becomes larger as it becomes more difficult to optimize the design's placement and routing due to the complexity of the broadcasting network implementation. However, the possible (upper-bound) throughput of the system continues to improve despite the frequency decreases. For example, when we double the PE count from 64 to 128, the throughput can double while the frequency reduces by just 21%; this results in a maximum throughput improvement of 58%. A further doubling also sees a 100% increase in PEs, but now with a 36% decrease in frequency. The result continues to be a net win for scalable applications, with a 28% increased throughput over the 128 PE case. While out of scope for

TABLE 2. Impact of PE count and frequency on power and area. The input FIFO size is 16. To compare fairly with REVAMP, we use the power and area from a synthesized design that excludes the estimated interconnect area.

# of PEs	Frequency (MHz)	Power[1] (mW)	Power[2] (mW)	Area[1] (mm^2)	Area[2,3] (mm^2)	Area[2,4] (mm^2)
64	690	18.9	18.8	0.555	0.511	0.385
	100	2.9	2.8			
128	546	28.1	26.2	1.115	1.007	0.774
	100	5.4	5.0			
256	351	32.9	31.3	2.232	2.072	1.566
	100	9.9	9.4			

(1) Based on the placed-and-routed design
(2) Based on the synthesized design
(3) Including the interconnect area estimate calculated by Cadence Genus
(4) Excluding the interconnect area estimate calculated by Cadence Genus

this current work, we expect an increase in performance for most workloads with additional unrolling. Synthesis results are also included for a fair comparison with REVAMP [20].

D. POWER EFFICIENCY

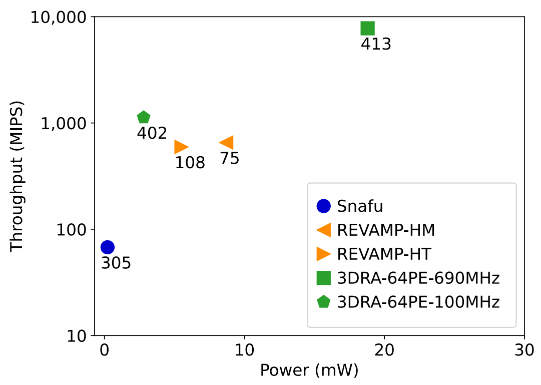
Figure 9a shows the power efficiency of the 3DRA in two configurations compared to previously proposed power-efficient CGRA implementations [20], [24]. The 3DRA accelerator shows the highest level of power efficiency on average, at 413 MIPS/mW with 64 PEs running at 690 MHz. REVAMP [20] can reach up to 177 MIPS/mW using heterogeneous PEs. Snafu [24], an ultra-low-power CGRA, reports that it achieves 305 MIPS/mW. It is said to be synthesized using sub-28 nm technology. These results demonstrate that the 3DRA can exploit a large amount of parallelism (and performance) with higher power efficiency compared to current, state-of-the-art, reconfigurable architectures.

E. POWER BREAKDOWN

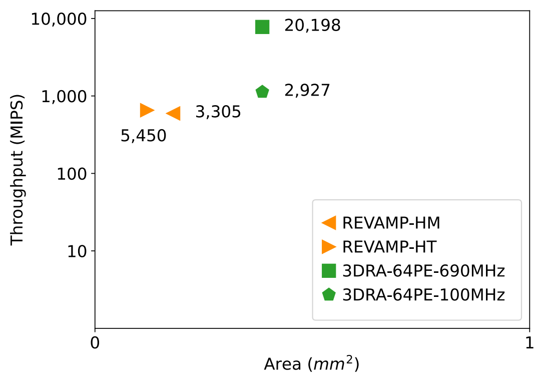
In Table 3, the power breakdown is shown consisting of compute units, FIFOs, control units, and memory controllers. Most of the power in 3DRA is spent on computation and buffering incoming data at input FIFOs. FIFOs take a large portion of the total power consumption in the design and consume around 45 % of the total power. The size of the FIFOs can be reduced when the power budget is tight or the target application kernels that will be run on 3DRA do not benefit from bigger FIFOs such as *Conv2d* as shown in Figure 8.

F. AREA EFFICIENCY

The areas of synthesized 64 PE, 128 PE and 256 PE versions of 3DRA are 0.385 mm^2, 0.774 mm^2, 1.566 mm^2, respectively. In the best case, 3DRA can reach up to 35,709 MIPS/mm^2 with an average area efficiency of 20,198 MIPS/mm^2, with 64 PEs running at 690 MHz. This is approximately 3.7× the area efficiency of the REVAMP-HT, the heterogeneous version that demonstrates a 5,450 MIPS/mm^2 area efficiency. Snafu [24] does not reveal the exact area and only mentions it is much smaller than 1 mm^2. Due to the lack of information, we are not able to



(a) Power Efficiency (MIPS/mW)



(b) Area Efficiency (MIPS/mm^2)

FIGURE 9. Efficiency compared to state-of-the-art CGRAs when 3DRA is synthesized using 16 FIFO entries. Throughput numbers are averages across all benchmarks. We cannot accurately list Snafu's [24] area efficiency as they don't provide an exact size (reported as lower than 1 mm^2).

TABLE 3. Power consumption breakdown of the 3DRA with 64 PEs and 16 input FIFO entries.

		Power (mW)		
	#	Static	Dynamic	Total (Total %)
Compute Unit	64	0.08	8.86	8.94 (47.32%)
FIFO	3×64	0.07	8.41	8.48 (44.89%)
Control Unit & Memory Controller	-	0.01	1.46	1.47 (7.79%)
Total		0.16	18.73	18.89

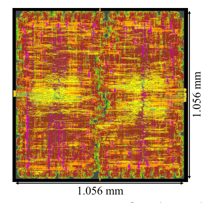
compare 3DRA with this prior work in terms of performance per area. Figure 10a shows the layout of 3DRA with 128 PEs and 16 FIFO entries.

VII. RELATED WORK

We classify related work based on their instruction placement and issue schemes using the taxonomy from [25].

A. DYNAMIC PLACEMENT, DYNAMIC ISSUE

DynaSpAM [26] uses a spatial architecture coupled with an Out-of-Order (OoO) processor that generates OoO execution schedules for it to enable dynamic scheduling. However, the issuing of instructions follows a fixed schedule that can unnecessarily delay operations, even though their operands arrive early. Therefore, its performance is limited by the host processor.



(a) Layout of placed-and-routed 3DRA with 128 PEs, and 16 input FIFO entries.

Param.	3DRA	REVAMP	Snafu
Freq (MHz)	100,351, 546,690	100	25
PE count	64, 128,256	36 (6x6)	36 (6x6)
Datapath width	32-bit	32-bit	n/a
Memory Ports	12	12	12

(b) Architecture parameters, n/a=not available.

FIGURE 10. Architecture and layout information.

B. STATIC PLACEMENT, DYNAMIC ISSUE

Most spatial architectures employ static placement solutions to minimize the routing costs on the fabric. However, their processing units are using a point-to-point network that increases network latency, due to multi-hop data communication. If the placement cannot be optimized, it will result in significantly reduced performance [25], [27]. Wavescalar [28] builds clusters of processing elements to allow for higher scalability and reduces network latency within clusters by connecting processing elements with a bus. This solution is limited by the capabilities of the compiler as it requires smarter placement policies to allow for shorter communication between neighboring instructions/processing elements.

Versat [29] consists of several PEs that are fully connected using a wide data bus, which enables single-cycle data access between any PEs similar to 3DRA. However, the PEs are programmed based on a 672-bit word-level configuration. The number of execution cycles is fixed to ensure that all the operations are completed before sending data, which can stall operations that are already ready to send outputs. In addition, it has just 2 pipeline stages, which can limit the throughput. In contrast, the PEs of the 3DRA architecture are independently programmed, and they send data whenever the output value is ready and the destination PEs are ready to receive them. In addition, the input buffers in every PE enable 3DRA to pipeline the target kernel at a much deeper level, which results in higher throughput.

Snafu [24] achieves very low power consumption by statically programming a crossbar network-on-chip with multi-hop data transfers. However, its buffer-less network design requires complex communication scheduling to avoid conflicts when data are sent on a channel.

RipTide [30] extends Snafu using a hardware-software co-design methodology to achieve higher performance and efficiency. The control-flow instructions are handled in the Network on Chip (NoC) to increase the utilization of PEs and the compiler handles the placement and routing considering the control-flow handling on NoC. In 3DRA, by using an all-to-all bypass network, the placement, and routing are heavily simplified without requiring complex compiler support. It also enables 3DRA to achieve low latencies while the mesh network can cause longer latencies due to multi-hop data transfers. RipTide reports 17 % higher performance over Snafu [24] which is still lower than the throughput we achieve in this work. However, they achieve up to 1,970 MIPS/mW which is higher than 3DRA's power efficiency. While RipTide outperforms 3DRA from an energy-efficiency perspective, our solution is the first to present a comprehensive, energy-efficient solution, that is easy to program, and handles dynamic events. Unfortunately, the workloads used to evaluate the two solutions are different, so a direct comparison isn't yet possible.

C. STATIC PLACEMENT, STATIC ISSUE

Statically building the placement and the issuing of instructions sacrifices flexibility with power efficiency [19], [31], [32]. To enable static issue, the routing must be done to guarantee that the operands are ready by issue time. This is a complicated task and often takes hours to complete when performed using Iterative Modulo Scheduling [16]. HyCUBE [12] improves communication performance by enabling single-cycle multi-hop data transfer on a mesh network. However, the wires on the mesh network can be used only once per cycle, creating contention between communicating PEs. While HiMap [33] allows for more efficient mappings for large CGRAs, it is limited to multi-dimensional kernels and is not generic.

D. ARTIFICIAL INTELLIGENCE

To enhance both compilation time and mapping quality, artificial intelligence methodologies such as reinforcement learning or graphical neural networks [34], [35], [36] are frequently employed. However, their success is not guaranteed due to the hardware limitations of the conventional CGRAs, like a mesh interconnection network and instruction execution timing constraints.

E. COMMUNICATION NETWORKS

All-to-all bypass networks are widely used in out-of-order CPUs to reduce the communication delays between operations [37]. By using a bypass network, the result of an instruction can be immediately forwarded to the next instruction that uses it as an input operand bypassing the register file and/or the memory, thus reducing access latency. To achieve this in a CPU that detects dependencies at runtime, a dedicated control module is required along with the bypass network to manage tagging and network arbitration. This dynamic tagging and arbitration require complex hardware, especially as the number of functional units increases, since the control module must plan and monitor the entire execution flow in every cycle. In addition, the complexity increases when the CPU has deep pipeline stages [38]. In 3DRA, we implement dynamic dataflow execution where each processing element can independently determine whether it's ready to execute or send the output to the destinations. Therefore, allows us to create a much simpler all-to-all bypass network without tagging and a central control unit.

VIII. CONCLUSION

In this work, we propose the 3DRA reconfigurable accelerator that implements a dynamic dataflow execution model to resolve data dependencies at runtime. This accelerator also builds on a single-cycle, non-blocking broadcast communication system to utilize a low latency network which reduces the latency to nearly zero. This results in up to 4.43x increase in throughput compared to previous works and power efficiency of up to 731 MIPS/mW.

REFERENCES

[1] R. R. Schaller, "Moore's law: Past, present and future," *IEEE Spectr.*, vol. 34, no. 6, pp. 52–59, Jun. 1997.

[2] C. Märtin, "Multicore processors: Challenges, opportunities, emerging trends," in *Proc. Embedded World Conf.*, 2014, p. 1.

[3] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2011, pp. 365–376.

[4] J. Lee, M. Samadi, Y. Park, and S. Mahlke, "Transparent CPU-GPU collaboration for data-parallel kernels on heterogeneous systems," in *Proc. 22nd Int. Conf. Parallel Archit. Compilation Techn.*, Sep. 2013, pp. 245–255.

[5] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, Mar. 2020.

[6] T. Luo, S. Liu, L. Li, Y. Wang, S. Zhang, T. Chen, Z. Xu, O. Temam, and Y. Chen, "DaDianNao: A neural network supercomputer," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 73–88, Jan. 2017.

[7] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[8] L. Calicchia, V. Ciotoli, G. C. Cardarilli, L. D. Nunzio, R. Fazzolari, A. Nannarelli, and M. Re, "Digital signal processing accelerator for RISC-V," in *Proc. 26th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2019, pp. 703–706.

[9] M. E. Coimbra, A. P. Francisco, and L. Veiga, "An analysis of the graph processing landscape," *J. Big Data*, vol. 8, no. 1, pp. 1–41, Apr. 2021.

[10] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–13.

[11] A. Faz-Hernández, J. López, and R. Dahab, "High-performance implementation of elliptic curve cryptography using vector instructions," *ACM Trans. Math. Softw.*, vol. 45, no. 3, pp. 1–35, Sep. 2019.

[12] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.

[13] 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.*, vol. 52, no. 6, pp. 1–39, Nov. 2020.

[14] C. Kim, M. Chung, Y. Cho, M. Konijnenburg, S. Ryu, and J. Kim, "ULP-SRP: Ultra low power Samsung reconfigurable processor for biomedical applications," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2012, pp. 329–334.

[15] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016.

[16] B. R. Rau, "Iterative modulo scheduling: An algorithm for software pipelining loops," in *Proc. 27th Annu. Int. Symp. Microarchitecture (MICRO)*, 1994, pp. 63–74.

[17] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "EPIMap: Using epimorphism to map applications on CGRAs," in *Proc. DAC Design Autom. Conf.*, Jun. 2012, pp. 1280–1287.

[18] L. Chen and T. Mitra, "Graph minor approach for application mapping on CGRAs," *ACM Trans. Reconfigurable Technol. Syst. (TRETS)*, vol. 7, no. 3, pp. 285–292, 2014.

[19] V. Govindaraju, C.-H. 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, Sep. 2012.

[20] T. K. Bandara, D. Wijerathne, T. Mitra, and L.-S. Peh, "REVAMP: A systematic framework for heterogeneous CGRA realization," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst. (ASPLOS)*, Feb. 2022, pp. 918–932.

[21] S. Dave, M. Balasubramanian, and A. Shrivastava, "RAMP: Resource-aware mapping for CGRAs," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[22] Z. Zhao, W. Sheng, Q. Wang, W. Yin, P. Ye, J. Li, and Z. Mao, "Towards higher performance and robust compilation for CGRA modulo scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 9, pp. 2201–2219, Sep. 2020.

[23] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a Scala embedded language," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, 2012, pp. 1216–1225.

[24] 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 *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 1027–1040.

[25] D. Burger, S. W. Keckler, K. S. McKinley, M. Dahlin, L. K. John, C. Lin, C. R. Moore, J. Burrill, R. G. McDonald, and W. Yoder, "Scaling to the end of silicon with EDGE architectures," *Computer*, vol. 37, no. 7, pp. 44–55, Jul. 2004.

[26] F. Liu, H. Ahn, S. R. Beard, T. Oh, and D. I. August, "DynaSpAM: Dynamic spatial architecture mapping using out of order instruction schedules," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 541–553.

[27] A. Parashar, M. Pellauer, M. Adler, B. Ahsan, N. Crago, D. Lustig, V. Pavlov, A. Zhai, M. Gambhir, A. Jaleel, R. Allmon, R. Rayess, S. Maresh, and J. Emer, "Triggered instructions: A control paradigm for spatially-programmed architectures," in *Proc. 40th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2013, pp. 142–153.

[28] S. Swanson, K. Michelson, A. Schwerin, and M. Oskin, "WaveScalar," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2003, pp. 291–302.

[29] J. D. Lopes, M. P. Véstias, R. P. Duarte, H. C. Neto, and J. T. de Sousa, "Coarse-grained reconfigurable computing with the Versat architecture," *Electronics*, vol. 10, no. 6, p. 669, Mar. 2021.

[30] G. Gobieski, S. Ghosh, M. Heule, T. Mowry, T. Nowatzki, N. Beckmann, and B. Lucia, "RipTide: A programmable, energy-minimal dataflow compiler and architecture," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2022, pp. 546–564.

[31] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in *Field Programmable Logic and Application*. Berlin, Germany: Springer, 2003, pp. 61–70.

[32] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "Baring it all to software: Raw machines," *Computer*, vol. 30, no. 9, pp. 86–93, 1997.

[33] D. Wijerathne, Z. Li, A. Pathania, T. Mitra, and L. Thiele, "HiMap: Fast and scalable high-quality mapping on CGRA via hierarchical abstraction," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 10, pp. 3290–3303, Oct. 2022.

[34] Z. Li, D. Wu, D. Wijerathne, and T. Mitra, "LISA: Graph neural network based portable mapping on spatial accelerators," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Apr. 2022, pp. 444–459.

[35] Y. Zhuang, Z. Zhang, and D. Liu, "Towards high-quality CGRA mapping with graph neural networks and reinforcement learning," in *Proc. 41st IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2022, pp. 1–9.

[36] X. Kong, Y. Huang, J. Zhu, X. Man, Y. Liu, C. Feng, P. Gou, M. Tang, S. Wei, and L. Liu, "MapZero: Mapping for coarse-grained reconfigurable architectures with reinforcement learning and Monte-Carlo tree search," in *Proc. 50th Annu. Int. Symp. Comput. Archit.*, Jun. 2023, pp. 1–14.

[37] A. D. Kahlich and R. R. Godard, "Computer processor employing bypass network using result tags for routing result operands," U.S. Patent 9 965 274, May 8, 2018.

[38] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors," in *Proc. 24th Annu. Int. Symp. Comput. Archit. (ISCA)*, 1997, pp. 206–218.

JINHO LEE (Member, IEEE) is currently pursuing the Ph.D. degree with the National University of Singapore, under the supervision of Dr. Trevor E. Carlson. His research interests include range from dataflow-based general-purpose reconfigurable architectures to domain-specific accelerators, specifically graph accelerators.

BURIN AMORNPAISANNON (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the National University of Singapore, under the supervision of Dr. Trevor E. Carlson and Dr. Li-Shiuan Peh. His research interests include understanding physical attack vectors, hardware trojans, efficient neuromorphic computing, and computer architecture in general.

ANDREAS DIAVASTOS received the Ph.D. degree in computer architecture from the University of Cyprus. He did his Postdoctoral Fellowship with the Computer Architecture Group, National University of Singapore (NUS), and then, he joined Universitat Politècnica de Catalunya (UPC), as a Distinguished Researcher. He is currently a Research Associate with the Computer Architecture Group, NUS. His research interests include processor and accelerator architectures, hardware-software co-design, parallel programming and execution models, automatic parallelization, and high-performance computing (HPC). He developed the SWITCHES parallel runtime systems, including the first auto-tuning system for static schedules for task data-flow applications on multi- and many-core systems.

TREVOR E. CARLSON (Senior Member, IEEE) is currently an Assistant Professor with the National University of Singapore (NUS). His research interests include efficient general-purpose processing, secure systems, AI acceleration, and simulation methodologies. He co-develops the Sniper Multi-Core Simulator, which is being used by hundreds of researchers in academia and industry, to evaluate the performance and power efficiency of next-generation systems. He has over 20 years of computer systems and architecture experience in both industry and academia, and his work has received six best papers or best paper nominations in conferences, such as the International Symposium on Microarchitecture (MICRO) and the International Symposium on Performance Analysis of Systems and Software (ISPASS).