An Open-Source Framework for the Generation of RISC-V Processor + CGRA Accelerator Systems

Xiaoyi Ling, Takahiro Notsu†, Jason Anderson
Dept. of Electrical and Computer Engineering, University of Toronto
†*Fujitsu, Ltd., Kawasaki, Japan*
Email: xiaoyi.ling@mail.utoronto.ca, notsu.takahiro@fujitsu.com, janders@eecg.toronto.edu

2021 24th Euromicro Conference on Digital System Design (DSD) | 978-1-6654-2703-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/DSD53832.2021.00015

***Abstract*—We describe a framework for automated generation of hybrid processor/accelerator systems comprising a RISC-V processor, and a coarse-grained reconfigurable array (CGRA) for realizing compute-kernel acceleration. CGRAs are programmable hardware platforms having an array of coarse ALU-like processing elements, and word-wide programmable interconnect. The proposed framework integrates CGRAs generated by the open-source CGRA-ME tool [1], with the RISC-V processor from the PULP project [2]. In an experimental study, we use the framework to generate RISC-V+CGRA systems that provide an order-of-magnitude speedup vs. software and considerable speedup vs. a vector processor on several applications by leveraging the CGRA spatial and pipeline parallelism. As CGRA-ME permits a variety of different CGRAs to be modelled and mapped to, we believe the proposed framework represents a powerful open-source platform, enabling a variety of new research on processor/CGRA system architectures and programming models.**

I. Introduction

RISC-V [3] is an open-source instruction set architecture (ISA) gaining traction relative to proprietary alternatives such as ARM, particularly in low-power/embedded/mobile scenarios. Microprocessors in such scenarios are typically paired with hardware accelerators, to perform video processing, ML inference, or other tasks with superior energy efficiency, and providing higher performance and a better user experience. Programmable hardware is one means of implementing such application accelerators, including field-programmable gate arrays (FPGAs) and coarse-grained reconfigurable arrays (CGRAs). CGRAs are programmable hardware platforms comprised of an array of large coarse-grained (often ALU-like) blocks and word-wide configurable interconnect. CGRAs stand in contrast to FPGAs, which have fine-grained programmable elements (LUTs) and interconnect configurable at the individual logic-signal level. CGRA-ME [1] is an open-source framework for CGRA architecture modelling and exploration. In this work, we integrate a popular RISC-V implementation with CGRA accelerators generated by CGRA-ME, realizing a software framework for generation of RISC-V + CGRA-accelerator systems. We believe the framework will be broadly useful for research on processor+CGRA architectures, domain-specific compute acceleration and programming methodologies.

RISC-V is a family of ISAs, with the simplest being a 32-bit integer ISA with no multiplication/division instructions (labelled "RV32I"), ranging to more complex 64-bit ISAs with floating-point and atomics capabilities. The software ecosystem for RISC-V is maturing rapidly under active development, and currently includes Linux, `gcc` compiler support, debuggers, and so on. A key value proposition of the RISC-V ISA is that *any* individual or company is free to build and use a hardware implementation. SiFive (`www.sifive.com`), for example, is a start-up offering silicon RISC-V implementations, and there also exist many open-source RISC-V implementations.

CGRAs first appeared in the literature in the 1990s (e.g. [4]) and a significant number of CGRA architectures have been proposed by both academia (surveyed in [5]), as well as industry (e.g. [6]). Much of the prior work on CGRA architectures has been "point solutions" where an individual CGRA is proposed and bespoke CAD tools are developed for it. The CGRA-ME framework aims to provide for CGRAs a capability analogous to what VTR [7] and Gem5 [8] provide for fine-grained FPGAs and standard processors, respectively. CGRA-ME offers the ability to model a range of CGRA architectures, as well as to map applications onto them, thereby allowing exploration and comparison of architectures, as well as facilitating new research on CGRA CAD algorithms.

We provide a software framework for automated generation RISC-V+CGRA hybrid systems, where the processor and CGRA communicate through memory-mapped I/Os and shared memory. Verilog RTL is automatically generated, allowing simulation and verification, as well as physical implementation using an ASIC standard-cell methodology or as an FPGA overlay. Unlike recent related work on RISC-V+CGRA systems (e.g. [9]), our generation framework is not tied to the use of a specific CGRA; rather, *any* CGRA that can be modelled by the CGRA-ME framework can be used.

The contributions of this paper are as follows:

- A software framework that offers push-button automated generation of RISC-V+CGRA hybrid systems.
- A programming methodology for the generated hybrid systems.
- An experimental study using the ADRES CGRA [10] as a representative example, with the RISC-V+CGRA mapped into 45nm standard cells [11]. The study includes 4 applications mapped onto CGRAs of different sizes. We compare the performance and area of the generated hybrid systems with: 1) software running on the RISC-V, and 2) a RISC-V with vector extensions. Results show the RISC-V+CGRA provides an order-of-magnitude speedup vs. software alone, and $\sim$2$\times$ speedup vs. the vector processor.

978-1-6654-2703-6/21/$31.00 ©2021 IEEE
DOI 10.1109/DSD53832.2021.00015

- A demonstration of the framework's flexibility and genericness by interchanging the ADRES CGRA with the HyCUBE CGRA architecture [12] and evaluating the performance.

II. Background

A. RISC-V

The RISC-V ISA was originally developed at the University of California, Berkeley, and is now maintained by the RISC-V foundation[1]. The ISA was designed with flexibility in mind. There is a base integer ISA that can be used directly as a lightweight RISC processor; the base processor has several dozen different instructions. Optional standard extensions can be added to the base, e.g. for floating point support. The RISC-V ISA was designed in the *accelerator era* – the ISA itself is extensible to recognize that modern processors are rarely used in isolation, but rather, with application-specific accelerators. As such, the RISC-V ISA is designed to include custom instructions tailored to application needs. The open-source aspect and the flexibility are attractive attributes of RISC-V, and many hardware platforms are being developed based on it. For example, PolarFire SoC FPGAs from Microchip contain a hardened multicore RISC-V processor on the FPGA die [13].

B. CGRA-ME (Modelling and Exploration) Framework

Fig. 1 shows the aspects of the CGRA-ME flow leveraged in this research. The blue boxes show the inputs to the framework: 1) a specification of a CGRA architecture through a C++ API or an XML-based language, and 2) an application dataflow graph (DFG). The DFG represents a compute-intensive kernel for CGRA acceleration. The architecture specification is parsed to create an in-memory CGRA device model – the central green block in Fig. 1. The mapper in CGRA-ME is architecture agnostic [14]: it accepts the device model as input, as well as the application DFG, producing mapping statistics and a configuration bitstream for the modelled CGRA. The in-memory device model can also be fed to a Verilog RTL generator. The red boxes in Fig. 1 represent commercial EDA tools. We use ModelSim for RTL simulation to verify functionality and acquire cycle-count data. We use Synopsys Design Compiler and Cadence Innovus to generate a standard-cell implementation, to assess post-layout area and critical-path delay.

III. Hybrid System Architecture Definition

A. RISC-V Processor Selection

We evaluated a number of open-source RISC-V ISA hardware implementations, and selected the CV32E40P processor [2], developed as part of the PULP platform [15][2] from ETH Zurich and the Univ. of Bologna. We chose this processor as it has been used in a wide range of projects (e.g. [16]), has strong documentation, and the Verilog was clear and easy to modify. The processor is a 32-bit in-order RISC-V core with a 4-stage pipeline with no memory attached to the core. The

[1]https://riscv.org/

[2]https://pulp-platform.org/

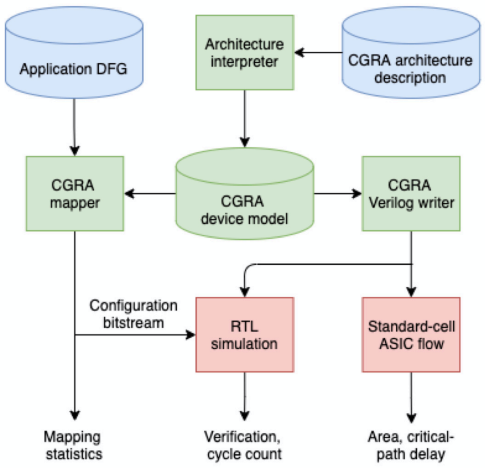

Fig. 1: CGRA-ME framework flow.

core is able to implement the RV32IMC (32-bit integer RISC-V with multiply/divide and compressed extension) ISA with optional floating-point extensions. It also supports several optional Xpulp custom extensions, for tasks such as bit manipulations and other ALU operations to achieve higher code density, performance, and energy efficiency. Considering that heavy computations are intended to be offloaded to the CGRA instead of being processed by the processor, we removed the floating-point, instruction-compression, and the custom Xpulp features for this work, thereby altering the processor to implement the RV32IM ISA.

B. CGRA Selection

While our framework is generic in the sense that it is not tied to a single CGRA architecture, we select a CGRA to be used as a test vehicle for demonstrating the framework and for the experimental study. To this end, we opted to use the ADRES CGRA architecture [10], as it is one of the most highly cited, and it had already been modelled within CGRA-ME. Fig. 2 shows a 4×4 ADRES CGRA. There are 4 rows and 4 columns of processing elements (PEs). Each PE contains an ALU capable of: add, subtract, multiply, shift, and logical operations. PEs in the top row of the CGRA share a register file (DRF); PEs in other rows each have their own local register file (LRF). PEs in the top row also have access to I/O ports. Each row of PEs has access to a memory port, shown on the left size of the figure. Regarding interconnect, PEs connect to their 4 NSEW neighbours, as well as diagonal neighbours (not shown in Fig. 2 for clarity). Toroidal connections "wrap-around' from the top/bottom (left/right) of each column (row). In addition to ADRES, we conduct experiments with the HyCUBE CGRA [12] (also modelled in CGRA-ME [17]) to verify the genericness of our framework.

C. Processor/CGRA Connectivity

A processor and a CGRA accelerator can be connected to one another in various ways, some characterized as *tightly coupled*

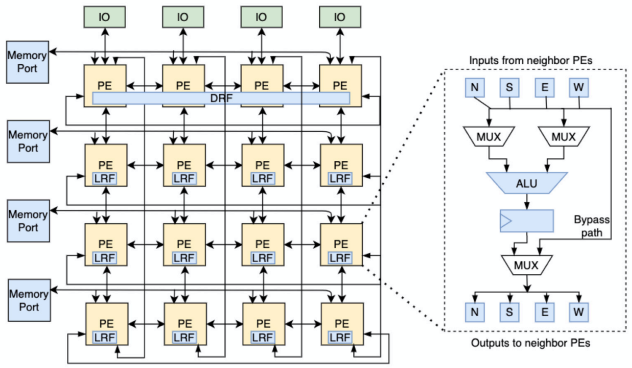

Fig. 2: 4×4 ADRES CGRA architecture.

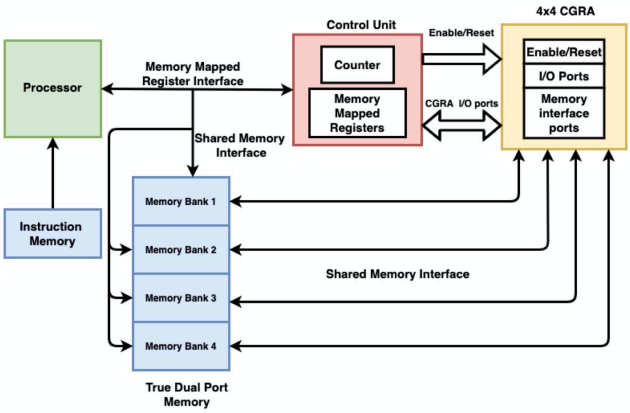

Fig. 3: Block diagram for hybrid RISC-V + CGRA system.

and others *loosely coupled*. In the tightly coupled approach, the CGRA may for example have direct access to the processor's register file or its internal pipeline and/or ALU registers. While such an approach may offer advantages in terms of requiring fewer explicit data transfers between the processor and CGRA, we believe it would be challenging to make generic and automated in terms of the range of CGRA architectures that could be used with the processor in an auto-generated fashion. We therefore opted for a more loosely coupled approach.

Fig. 3 shows a block diagram of the RISC-V+CGRA hybrid system, elaborated upon in this and subsequent sections. The processor is on the left (green); the CGRA on the right (yellow); memory (blue) and control (red) are in the centre. The CGRA is designed to communicate with the RISC-V via two types of interfaces: memory-mapped I/O ports on the CGRA and through shared memory between the processor and CGRA. Two hybrid systems are implemented and tested in our experimental study, having 4×4 and 8×8 CGRA accelerators as subsystems, to show the flexibility of our platform to handle CGRAs of different sizes, and explore area/performance trade-offs. As is apparent in Fig. 2, different sized ADRES CGRAs will have different numbers of I/Os and memory ports. The hybrid system is modified accordingly (and automatically, described below), based on the number of available CGRA I/Os and memory ports.

I/O ports are used for two purposes: 1) As pointers (addresses) to shared memory, for example, to indicate to the CGRA the location of input data, or the location where output data should be stored; and, 2) as direct data inputs and outputs to/from the CGRA, obviating the need to communicate data via shared memory. Both purposes are leveraged in our experimental study. The CGRA I/O ports are accessible to the RISC-V processor via loads/stores from memory-mapped registers.

In the hybrid system, we employ a Harvard memory architecture, where the instruction and data memories are separated. The instruction memory is single ported, as only the processor accesses this. The data memory is dual-ported and partitioned into banks, with two underpinning rationales: First, the RISC-V and CGRA should be able to work independently, meaning the RISC-V processor should be capable of continuing executing (and accessing data) while the CGRA concurrently performs operations. This necessitates dual-port memory functionality. Second, the need for banking is so multiple memory ports of the CGRA can be used concurrently, increasing memory-access parallelism to achieve higher computational throughput. The example in Fig. 3 shows four memory banks, which the processor "sees" as a single memory address space, and the CGRA sees as individual banks that can be accessed independently.

D. CGRA Control

For the proposed hybrid system, the processor is responsible for controlling and transferring data to/from the CGRA. The RISC-V processor executes a software program and invokes the CGRA for a computationally-intensive program hot spot – a compute kernel. The CGRA is pre-configured to accelerate the specific application. That is, the CGRA is configured with a bitstream for the kernel's DFG produced by the mapper in CGRA-ME, shown in Fig. 1.

As shown in Fig. 3, a control unit resides between the RISC-V and the CGRA. The control unit contains memory-mapped registers and a counter. There is one memory-mapped register for every CGRA I/O port, allowing the processor to write/read from the CGRA I/Os. Two additional memory-mapped locations, *Enable* and *Reset*, allow the processor to enable and reset the CGRA.

CGRAs generated by the CGRA-ME framework are capable of streaming dataflow-style computations – the CGRAs do not support presently control-flow instructions, e.g. branch. The processor must manage the CGRA execution, including determining when the CGRA is "done" its work. This is handled through the counter, and a third memory-mapped location, *Done*. When a DFG is mapped onto a CGRA, the CGRA-ME framework reports the latency (in clock cycles) for the mapping, and also the initiation interval (II). With this information, the programmer can determine how many clock cycles the CGRA needs to process a specified number of data inputs. For example, if the mapping latency is 3, and the II=1, and the programmer intends to have the CGRA execute

10 data inputs, the total number of CGRA cycles required is $10 \times 1 + (3 - 1) = 12$. This value is written to the counter, as an initial value, before enabling the CGRA. The counter counts down and when it reaches 0, the CGRA is automatically disabled, and a '1' is written to the Done register. The processor can poll the Done register to determine when the CGRA has completed its work.

IV. Automated Hybrid System Generation

We modified CGRA-ME to automatically generate RISC-V+CGRA systems. The automation flow is illustrated in Fig. 4. The inputs are the CGRA architecture specification (as before) and the desired memory size, **M**, of the hybrid system: the size of the shared data memory between the processor and CGRA. Existing CGRA-ME functionality parses the architecture description into an in-memory device model. From this in-memory model, our hybrid-system writer extracts the number of CGRA I/O ports, **I**, as well as the number of CGRA memory ports, **P**. I/Os and memory ports are primitive elements in CGRA-ME that an architect may use in defining their desired CGRA architecture.

The hybrid system writer produces three Verilog RTL files. The first is RTL for the RISC-V/CGRA interface, as shown in the middle of Fig. 3. This RTL module contains **I** registers: one for every CGRA I/O. It also contains the control unit for starting/stopping the CGRA execution, as described above. The I/O registers, as well as the registers for Enable, Reset, Done, and the counter, are memory mapped, at a user-provided base address location higher than the maximum data-memory address.

The second RTL file produced is the dual-port banked memory. The user-specified memory of size **M** is partitioned into **P** banks automatically, with each bank automatically connected to one of the **P** memory ports of the CGRA. Presently, we require that banks be of equal size, so that the memory is evenly divisible into the banks. Finally, as shown in Fig. 4, we also generate a top-level Verilog model instantiating and connecting the RISC-V processor, memory, interface module, and the CGRA. This top-level model is used for RTL simulation of the complete RISC-V+CGRA system.

With this generic approach to hybrid system generation, there is no dependence on the specific CGRA architecture used in the hybrid system. The Verilog for the CGRA itself is produced by the existing Verilog writer within CGRA-ME. The CGRA's internal composition may be *any* CGRA modellable by the CGRA-ME framework.

V. Programming Model Illustration/Case Study

Four applications are used to evaluate the hybrid system, and this section will focus on one of them, Matrix Multiply (MM), as a representative example to illustrate the detailed implementation. The other applications are described in the experimental study.

We implement a multiply of matrix A, with size 20×40, and matrix B, with size 40×20, producing a total of 400 outputs. The CGRA is programmed with an application that computes one value of the output matrix. That is, the CGRA performs

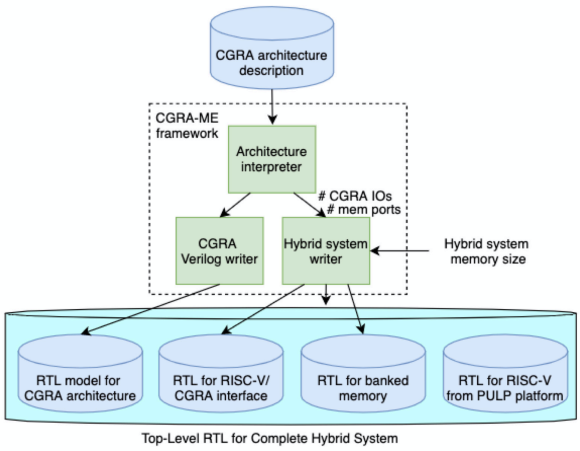

Fig. 4: Hybrid system generation flow.

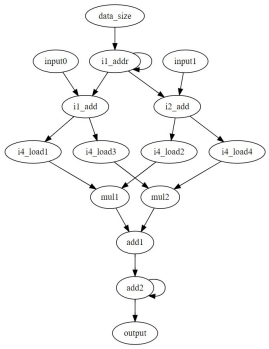

Fig. 5: Matrix multiply dataflow graph for 4×4 CGRA.

the dot product of a row of A and a column of B: 40 multiply-accumulate (MAC) operations. Fig. 5 shows the dataflow graph of the MM application for the 4×4 CGRA. For the DFG in Fig. 5, two input ports (nodes labeled *input0* and *input1* in the DFG) are used to indicate the address where the CGRA begins to fetch data (for a row of A and a column of B), and one output port is used to send the computed value back to the processor. Addresses updates happen inside the CGRA in parallel with computations, as well as loading/storing data. Since the 4×4 ADRES CGRA, used in our case study, has four memory interface ports, we are able to map two MAC operations onto the CGRA, leveraging all 4 memory ports each clock cycle. In a larger 8×8 CGRA, two copies of the DFG can be accommodated, using 8 memory ports.

In our implementation, for the 4×4 CGRA, matrix input data must be assembled in banks as follows: memory banks 1 and 3 store data from the rows of Matrix A and memory banks 2 and 4 store data from the columns of Matrix B. Banks 1 and 3 contain matrix data in row-major order; banks 2 and 4 column matrix data in column-major order. This data layout allows the CGRA to stream data from all banks every clock cycle in the correct order. Data in banks 1 and 3 is multiplied by data in banks 2 and 4, respectively, and accumulated.

For the RISC-V processor, hand-written assembly is used since there is presently no compiler for a high-level language

that can satisfy the special needs of the hybrid system connectivity and control – this is intended for our future work. The assembly is compiled by the RISC-V C cross-compiler toolchain provided by the RISC-V foundation [3]. The RISC-V program passes the starting addresses of rows/columns to the CGRA via the memory-mapped I/Os, and then invokes the CGRA to compute one entry in the result matrix.

The CGRA dot-product application's DFG has an II=1 after mapping with the CGRA-ME mapper onto the 4×4 ADRES CGRA, and a latency of 5. With two MACs/cycle, the total number of clock cycles needed for a dot product containing 40 MAC operations is $20 + (5 - 1) = 24$ cycles. The RISC-V invokes the CGRA then polls on the memory-mapped Done register to determine when the CGRA has completed its work. The processor then reads from the CGRA's output port (another memory-mapped I/O port) and writes the dot product result back to memory, then it updates starting addresses to the CGRA input ports, resets the CGRA to clear the output port and enables it again to compute the next dot product of a row and a column until all 400 outputs are produced.

VI. Experimental Study

A. Setup

We evaluate the RISC-V+CGRA hybrid systems on four applications and compare with two different baselines: 1) RISC-V running software alone, and 2) RISC-V with a vector unit [18]. For the hybrid systems, we compare two systems with different sizes of CGRA: a 4×4 CGRA and an 8×8 CGRA. For the vector processor, we evaluate two systems with different numbers of lanes: 4 lanes and 8 lanes. We use ModelSim to simulate the designs for functional correctness, and also to gather the number of clock cycles required for each benchmark application. The RISC-V, CGRAs and the vector processor use a common clock, with the rationale being that the three platforms exhibit roughly the same critical path delay, as demonstrated below in the ASIC implementation results.

Four applications are used: matrix multiply (MM), K-nearest neighbor (KNN), a finite impulse response (FIR) filter and polynomial evaluation ($POLY$) via Taylor expansion of e^x. The applications were selected to leverage both memory-mapped and shared-memory communication between the processor and CGRA. RISC-V binary programs for the benchmark applications are stored in the instruction memory. The CGRA configuration bitstream for each application is stored in a Verilog file and fed into the CGRA during the configuration process. We perform a cycle-accurate RTL simulation of a processor-only version of each application, the processor+CGRA version of each application, and also the vector processor.

We described MM in the previous section. We briefly review the other three applications: KNN uses Euclidean distance to classify objects using a training set of 200 examples with 2 features. Feature data of the training examples is stored in memory and feature data of the objects is input to CGRA I/O ports from the RISC-V. One object is processed each cycle for the 4×4 CGRA. The CGRA outputs the sum of the square of the distance between the example and the object for both features. For the 8×8 CGRA, two instances are implemented in parallel to produce two outputs. Additional instances could not be accommodated in the 8×8, owing to the # of memory ports required. All outputs are saved into memory banks by the CGRA.

FIR computes 200 outputs of a 3-tap FIR filter, whose coefficients are embedded in the DFGs input to the CGRA-ME mapper. The mapper places the coefficients into constant units present in each ADRES PE (omitted in Fig. 2 for clarity). For the 4×4 CGRA, one output is produced each cycle and for an 8×8 CGRA, two outputs are generated in parallel. Outputs are saved into memory banks by the CGRA.

$POLY$ computes the first four terms of a Taylor expansion for e^x. This is performed 60 times for different values of x, with x automatically incremented by the CGRA. The computations are performed in fixed point. For the 4×4 CGRA, one output for e^x is produced each cycle. while for the 8×8 CGRA, three outputs are produced every clock cycle. Outputs are saved into memory banks by the CGRA.

Source code for all benchmarks is included with the CGRA-ME download distribution.

B. Results

We first compare the hybrid CGRA systems with software alone on the RISC-V, then compare with the vector processor.

1) RISC-V+CGRA hybrid vs. RISC-V alone: Cycle-latency results are shown in Table I. The first column lists the application name. The second column gives the cycle count for software alone on the RISC-V. The third and fourth columns give cycle counts for the hybrid systems with 4×4 and 8×8 CGRAs, respectively. In these columns, the numbers in paratheses give the cycle-count speedup vs. software alone. The last row of the table shows the average speedup across all benchmarks relative to RISC-V software.

From Table I, we observe speedups from 11-19× for the hybrid system with the 4×4 CGRA. Larger speedups from 14-37× are observed for hybrid system with the 8×8 CGRA. The larger CGRA can accommodate two parallel instances of the DFGs for the MM, KNN and FIR benchmarks, and three parallel instances of the DFG for the $POLY$ benchmark. However, the relative speedup comparing the larger and smaller CGRA are not precisely 2× and 3×, owing to some fixed overhead shared by the hybrid systems that cannot be accelerated.

The large cycle-count speedups observed for the hybrid system vs. software alone arise from a number of factors:

- Spatial and pipeline parallelism:
  1) The CGRA computations are pipelined and all applications have an II=1, in the steady state producing outputs every clock cycle. For example, Taylor Expansion of e^x required 8 clock cycles for an output in software, while the CGRA can generate a result every clock cycle.
  2) The CGRA can update addresses for memory interface ports in parallel with computations, but software requires separate instructions for this.

TABLE I: Cycle latency of benchmarks executing: 1) in software on RISC-V, 2) on two types of hybrid RISC-V+CGRA systems, 3) on a RISC-V processor with vector extensions.

	System				
Benchmark	**RISC-V software**	**Hybrid 4x4**	**Hybrid 8x8**	**4-lane Vec. Proc.**	**8-lane Vec. Proc.**
MM	178918	16518 (10.83)	12518 (14.29)	29600 (6.04)	15200 (11.77)
KNN	18042	1296 (13.92)	670 (26.93)	2814 (6.41)	1410 (12.8)
FIR	4005	208 (19.25)	108 (37.08)	545 (7.35)	281 (14.25)
POLY	844	68 (12.41)	28 (30.14)	157 (5.38)	82 (10.29)
Avg. Speedp	**1**	**14.1**	**27.11**	**6.3**	**12.3**

TABLE II: Timing and area analysis. Timing results for RISC-V and CGRAs are post-layout; timing results for the vector units are post-synthesis.

Architecture	Optimization	CP (ns)	Area (μm^2)
RISC-V	Area-optimized	5.95	61489
	Timing-optimized	3.86	70011
CGRA 4×4	Area-optimized	(Baseline) 4.13	120103
	Timing-optimized	(Baseline) 2.22	138089
CGRA 8×8	Area-optimized	(Baseline) 4.33	472663
	Time-optimized	(Baseline) 2.41	534580
4-lane Vec. Proc.(Syn)	Area-optimized	6.46	211890
	Timing-optimized	2.05	232240
8-lane Vec. Proc.(Syn)	Area-optimized	6.51	390447
	Timing-optimized	2.07	427074

TABLE III: Bypass delays (ns)

	Delay (ns)	
CGRA Size	Area-optimized	Timing-optimized
4×4	0.38	0.26
8×8	0.46	0.30

- Branches: CGRA computations are "branch free" because a counter is used to determine when the CGRA's work is finished. Conversely, the processor needs to use compare-and-branch instructions to detect when a loop trip-count is reached, requiring several clock cycles.
- Memory access parallelism: the CGRAs are connected to multiple memory banks and memory interface ports can perform loads and stores at the same time. For example, for the FIR filter implemented on an 8×8 CGRA, six input data words are fetched from memory and two outputs are written to memory each clock cycle in steady state. In the processor, each load and store will require separate (non-parallel) memory accesses.

Synopsys Design Compiler is used to synthesize the RISC-V and the CGRAs to the NanGate 45nm standard-cell library [11]. Cadence Innovus is used to place and route the processor and CGRAs. We performed a layout extraction, and analyzed post-route performance using Synopsys PrimeTime. We synthesized, placed and routed two versions of the processor and CGRAs: area-optimized and timing-optimized. For the area-optimized, we supplied no performance constraint to synthesis and set a tight area constraint, causing it to select the smallest cells. For the timing-optimized, we supplied a tough-to-meet $1ns$ clock-period constraint to Design Compiler. The CGRA layout was floorplanned to keep PEs within rows/columns in the proper order in the physical layout.

Area and critical path delay are evaluated separately for the RISC-V processor and the CGRAs. The post-layout results are presented in top portion of Table II[3]. The area-optimized RISC-V consumes 61.4Kμm^2 and has a critical path delay of $\sim$6ns. The timing-optimized RISC-V consumes 70.0Kμm^2, with a delay of $\sim$3.9ns. The 4×4 CGRA consumes about twice the area as the RISC-V. Unsurprisingly, the 8×8 CGRA consumes about $4\times$ the area of the smaller CGRA.

The CGRA critical-path delay values in Table II represent floor values, beginning at a processing element, passing through the interconnect to an immediate neighbour processing element, passing through that neighbor's ALU, and then reaching the local register file. Thus, the critical-path delay values in the table reflect an application mapping scenario where all communicating PEs are placed immediately adjacent to one another. Such an ideal placement is not always possible, however, and the ADRES CGRA permits longer connections between PEs to be formed by combinationally *bypassing* one or more PEs (see bypass path in Fig. 2). The delay costs associated with using one of such bypass connections is shown in Table III. By combining the floor critical path delay data in Table II, with the number of bypasses used for each application in its CGRA mapping (reported by the CGRA-ME mapper), and the bypass delay data in Table III, one can compute the final critical path for each application mapped onto the CGRA. This is shown in Table IV. For each application, the number of bypasses on the longest path is presented. The last row of the table shows the final computed critical path delay. The CGRA critical-path delays are close to the RISC-V critical-path delay. As such, we believe it a reasonable design decision that the RISC-V and CGRA use the same clock signal. This makes the speedups given in Table I also valid for wall-clock time.

2) Comparison with a RISC-V Vector Processor: An alternative approach to acceleration is through the use of a vector processor, which uses the single-instruction-multiple-data (SIMD) paradigm. We use the open-source RISC-V^2 [18] vector processor to execute the same 4 applications and assess performance and area relative to hybrid CGRA-based systems. RISC-V^2 is a RISC-V processor with vector extensions, supporting instructions like vector-add `vadd` and vector-multiply `vmul`. We removed the floating point and divider units from the RISC-V^2 to permit a more fair comparison with the

[3]The area results do not include the RAMs, which can be sized according to the application needs.

TABLE IV: Bypass counts and critical path delays on ADRES CGRA.

Application	MM				KNN				FIR				POLY			
CGRA Size	4×4		8×8		4×4		8×8		4×4		8×8		4×4		8×8	
Bypass Number	2		3		3		3		1		3		1		3	
Optimization	Area	Time	Area	Time	Area	Time	Area	Time	Area	Time	Area	Time	Area	Time	Area	Time
Final CP (ns)	4.89	2.74	5.71	3.31	5.27	3.00	5.71	3.31	4.51	2.48	5.71	3.31	4.51	2.48	5.71	3.31

CGRA capabilities. The processor is configurable with different numbers of lanes, ranging from 2 to 16. Each lane contains one execution unit. We opted for 4-lane and 8-lane versions for this study, owing to them having similar area to the two CGRA sizes used (discussed below).

The four applications were implemented to run on the vector processor using the compiler provided by the RISC-V^2 project. Complete Verilog RTL for RISC-V^2 is available, allowing us to perform cycle-accurate simulation for verification and to extract cycle latencies. We modified the memory model used in RISC-V^2 to match with the hybrid CGRA systems above, namely, single-cycle loads/stores from/to memory[4]. With the memory-model change, we can make an apples-to-apples comparison between application execution on the RISC-V^2 and the other systems considered. Applications execute partially on the scalar RISC-V, which invokes the vector-accelerator for compute-intensive portions of applications using vector instructions.

Cycle-latency results for the RISC-V^2 are shown in the right-most two columns of Table I. The speedups vs. the RISC-V alone are appreciable: 6.3× and 12.3× for the 4-lane and 8-lane vector processors, respectively. Speedups exceed lane counts as the vector processor eliminates many compare/branch instructions in the scalar RISC-V software. Note however, these average speedups are smaller than those afforded by using a CGRA for acceleration. The vector processor is able to fetch and process multiple data items depending according to the # of lanes, but it still requires several cycles to compute results. One instruction executes at a time and the compute units are arranged in a 1D configuration. In contrast, in the CGRA, the DFG specifying the computations is embedded into a 2D array of computational units, permitting pipeline parallelism, where different types of computations are overlapped in time.

The bottom rows of Table II give the area and speed performance of the vector unit (not including the RISC-V). Regarding area, the 4×4 CGRA consumes less area than the 4-lane vector processor, but the 8×8 CGRA is larger than the 8-lane processor because the 8-lane vector processor is double the size of the 4-lane vector processor, whereas 8×8 CGRA is $\sim$4× the area of the smaller CGRA. We investigated why the area of the 4×4 CGRA having 16 ALUs is smaller than the 4-lane vector processor having just 4 ALUs. In the area-optimized 4-lane vector unit, only 29% of the area is consumed by the execution units containing its ALUs. 54% of area is consumed by the issue unit, with the majority of this consumed by the vector register files – one per lane. 11% of the area is consumed by the memory unit, and the remaining 6% of area in other circuits. The vector unit's area is not dominated by its ALUs, but rather, by the area of its other internal structures.

[4]The RISC-V^2 originally used an L1 cache.

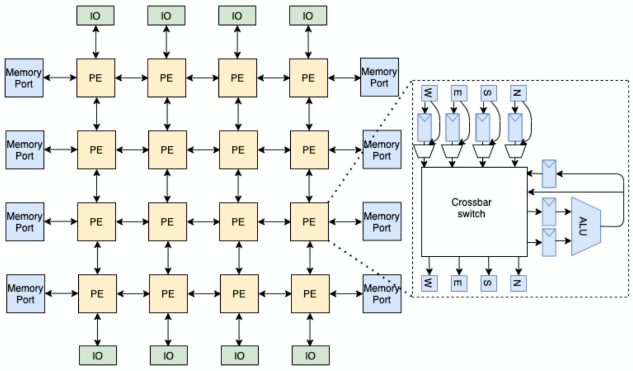

Fig. 6: HyCUBE CGRA architecture.

Table II also reports the critical-path delay for the RISC-V^2 vector unit. The delays reported are post-synthesis not post-layout, and therefore likely to be underestimates. While the source code for the vector unit went through synthesis without issues, when we pushed the technology-mapped gate-level netlists through place/route, we encountered timing problems related to very high fanout nets. Nevertheless, the post-synthesis delays are similar to the CGRA and RISC-V delays.

Overall, the performance results demonstrate that our framework, while generic and automatic, can generate RISC-V+CGRA hybrid systems having performance competitive with alternative acceleration approaches.

C. RISC-V with a HyCUBE CGRA

Finally, to illustrate the flexibility of our RISC-V+CGRA hybrid system generation framework, we "swapped out" the ADRES CGRA architecture with the HyCUBE CGRA architecture [12], illustrated in Fig. 6. HyCUBE has a considerably different PE containing a crossbar switch with 6 inputs (from the four neighboring PEs, and from the ALU's output in combinational/registered form) and 6 outputs (to the same neighboring PEs and to the two ALU operands). Multiple paths through the crossbar can be used concurrently, increasing routing flexibility vs. ADRES. In our HyCUBE architecture, I/Os are present on the top/bottom; memory ports are present on the left/right. Relative to ADRES with the same # of PE rows/columns, HyCUBE has double the number of I/Os and memory ports.

With double number of I/Os and memory ports and improved routability, applications whose mappings are limited by the number of ports and I/Os can be mapped to smaller size of HyCUBE CGRA compared to ADRES. Table V shows cycle latency results for a hybrid system with a 4×4 HyCUBE CGRA and compares with the ADRES-based hybrid systems described above. For three benchmarks, MM, KNN and FIR, the DFG

TABLE V: Cycle latency of benchmarks on hybrid system with 4×4 HyCUBE CGRA.

	MM	KNN	FIR	POLY
Hybrid 4×4 HyCUBE	13718	682	111	71
Speedup vs. Hybrid 4×4 ADRES	1.20	1.90	1.87	0.96
Speedup vs. Hybrid 8×8 ADRES	0.91	0.98	0.97	0.39

used for the 8×8 ADRES-based hybrid system was able to be mapped to the 4×4 HyCUBE. For these benchmarks, the 4×4 HyCUBE system has similar cycle counts to the 8×8 hybrid system, with the small differences due to different numbers of pipeline stages in the DFG's CGRA mapping for HyCUBE. For $POLY$, the 4×4 HyCUBE could not accommodate the larger DFG used for the 8×8 ADRES – the mapper could not find a routing in the smaller CGRA. For this case, the 4×4 ADRES and HyCUBE have similar latency.

The results illustrate that application performance will depend significantly on the selected CGRA architecture, and that the proposed automated RISC-V+CGRA hybrid system generation framework can be applied for researching which architectures are best suited to a given basket of applications.

VII. Related Work

Fiolhais et al. [9] describe a hybrid system with multiple RISC-V and CGRA cores. Two types of RISC-V are used, an area-optimized variant, and a pipelined variant with higher performance. A specific type of CGRA, called Versat [19], is used in the design. Das et al. [20] describe a RISC-V+CGRA system for sensor data analysis and they compare their system with a multi-core processor system.

In relation to the above, our work offers a generic framework for RISC-V + CGRA system generation, that is not tied to a specific CGRA or application domain.

VIII. Conclusions and Future Work

We described an open-source framework for the generation of RISC-V + CGRA hybrid systems, combining an established RISC-V pipelined processor implementation with CGRA accelerators as generated by the CGRA-ME framework. The processor and CGRA communicate through memory-mapped access to the CGRA's I/O ports, and through shared memory. Hybrid system generation is automatic and generic – it is not tied to any specific CGRA architecture. Any CGRA that can be modelled by the CGRA-ME framework can be incorporated into a hybrid system. In a case study using the ADRES CGRA on a set of four benchmark applications, speedups of 11-19× are observed vs. RISC-V software alone for a 4×4 CGRA+RISC-V, and 14-37× for an 8×8 CGRA+RISC-V. Speedups of $\sim$2× are observed for the RISC-V+CGRA systems vs. a RISC-V with a vector unit. We also showed the flexibility of the framework through use of an alternative CGRA architecture, HyCUBE. We believe the framework will enable a variety of new research on RISC-V+CGRA systems and associated programming methodologies.

Future directions are many, including improving the programming model to allow high-level languages to be used, and analyzing the power efficiency of RISC-V+CGRA systems.

References

[1] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi, and J. Anderson, "CGRA-ME: A unified framework for CGRA modelling and exploration," in *IEEE ASAP*, 2017, pp. 184–189.

[2] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold RISC-V core with DSP extensions for scalable iot endpoint devices," *IEEE TVLSI*, vol. 25, no. 10, pp. 2700–2713, 2017.

[3] RISC-V Foundation, *The RISC-V Instruction Set Manual*, 2019. [Online]. Available: https://riscv.org/

[4] S. C. Goldstein, H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor, and R. Laufer, "PipeRench: a coprocessor for streaming multimedia acceleration," in *ACM/IEEE ISCA*, 1999, pp. 28–39.

[5] 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, Oct. 2019.

[6] 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 *IEEE FPT*, 2012, pp. 329–334.

[7] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, "The VTR project: Architecture and CAD for FPGAs from Verilog to Routing," in *ACM FPGA*, 2012, p. 77–86.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, Aug. 2011.

[9] L. Fiolhais, F. Gonçalves, R. P. Duarte, M. Véstias, and J. T. de Sousa, "Low energy heterogeneous computing with multiple RISC-V and CGRA cores," in *IEEE ISCAS*, 2019, pp. 1–5.

[10] 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 *FPL*, 2003, pp. 61–70.

[11] J. E. Stine and et al., "FreePDK: An open-source variation-aware design kit," in *IEEE MSE*, 2007, pp. 173–174.

[12] M. Karunaratne, A. K. Mohite, T. Mitra, and L. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," in *ACM/IEEE DAC*, 2017.

[13] "PolarFire SoC," https://www.microsemi.com/product-directory/soc-fpgas/5498-polarfire-soc-fpga, Microchip Corp.

[14] M. J. P. Walker and J. H. Anderson, "Generic connectivity-based CGRA mapping via integer linear programming," in *IEEE FCCM*, 2019, pp. 65–73.

[15] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "PULP: A parallel ultra low power platform for next generation IoT applications," in *IEEE Hot Chips*, 2015.

[16] D. Rossi, F. Conti, M. Eggiman, S. Mach, A. D. Mauro, M. Guermandi, G. Tagliavini, A. Pullini, I. Loi, J. Chen, E. Flamand, and L. Benini, "A 1.3TOPS/W @ 32GOPS fully integrated 10-core SoC for IoT end-nodes with 1.7μw cognitive wake-up from MRAM-based state-retentive sleep mode," in *IEEE ISSCC*, 2021, pp. 60–62.

[17] S. A. Chin, K. P. Niu, M. Walker, S. Yin, A. Mertens, J. Lee, and J. H. Anderson, "Architecture exploration of standard-cell and FPGA-overlay CGRAs using the open-source CGRA-ME framework," in *ACM ISPD*, 2018, p. 48–55.

[18] K. Patsidis, C. Nicopoulos, G. C. Sirakoulis, and G. Dimitrakopoulos, "RISC-V^2: A scalable RISC-V vector processor," in *IEEE ISCAS*, 2020, pp. 1–5.

[19] J. D. Lopes and J. T. de Sousa, "Versat, a minimal coarse-grain reconfigurable array," in *High Performance Computing for Computational Science – VECPAR 2016*, I. Dutra, R. Camacho, J. Barbosa, and O. Marques, Eds. Cham: Springer International Publishing, 2017, pp. 174–187.

[20] S. Das, K. J. M. Martin, P. Coussy, and D. Rossi, "A heterogeneous cluster with reconfigurable accelerator for energy efficient near-sensor data analytics," in *IEEE ISCAS*, 2018.