Amber: A 16-nm System-on-Chip With a Coarse-Grained Reconfigurable Array for Flexible Acceleration of Dense Linear Algebra

Kathleen Feng, *Graduate Student Member, IEEE*, Taeyoung Kong, *Member, IEEE*, Kalhan Koul, Jackson Melchert, Alex Carsello, Qiaoyi Liu, Gedeon Nyengele, Maxwell Strange, Keyi Zhang, Ankita Nayak, Jeff Setter, James Thomas, Kavya Sreedhar, Po-Han Chen, *Graduate Student Member, IEEE*, Nikhil Bhagdikar, Zach A. Myers, *Member, IEEE*, Brandon D'Agostino, *Graduate Student Member, IEEE*, Pranil Joshi, Stephen Richardson, Christopher Torng, Mark Horowitz, *Fellow, IEEE*, and Priyanka Raina, *Life Fellow, IEEE*

***Abstract*— Amber is a system-on-chip (SoC) with a coarse-grained reconfigurable array (CGRA) for acceleration of dense linear algebra applications, such as machine learning (ML), image processing, and computer vision. It is designed using an agile accelerator–compiler codesign flow; the compiler updates automatically with hardware changes, enabling continuous application-level evaluation of the hardware–software system. To increase hardware utilization and minimize reconfigurability overhead, Amber features the following: 1) dynamic partial reconfiguration (DPR) of the CGRA for higher resource utilization by allowing fast switching between applications and partitioning resources between simultaneous applications; 2) streaming memory controllers supporting affine access patterns for efficient mapping of dense linear algebra; and 3) low-overhead transcendental and complex arithmetic operations. The physical design of Amber features a unique clock distribution method and timing methodology to efficiently layout its hierarchical and tile-based design. Amber achieves a peak energy efficiency of 538 INT16 GOPS/W and 483 BFloat16 GFLOPS/W. Compared with a CPU, a GPU, and a field-programmable gate array (FPGA), Amber has up to 3902×, 152×, and 107× better energy-delay product (EDP), respectively.**

***Index Terms*— Coarse-grained reconfigurable array (CGRA), computer architecture, computer vision, image processing, machine learning (ML), reconfigurable accelerators, system-on-chip (SoC).**

Manuscript received 9 February 2023; revised 5 July 2023 and 21 August 2023; accepted 26 August 2023. Date of publication 22 September 2023; date of current version 27 February 2024. This article was approved by Associate Editor Dennis Sylvester. This work was supported in part by the DARPA DSSoC Grant, in part by the Stanford AHA Agile Hardware Center and Affiliates Program, Intel's Science and Technology Center (ISTC), Stanford SystemX Alliance, SRC JUMP 2.0 PRISM Center, NSF CAREER Award under Grant 2238006, Hellman Faculty Scholar Program, and Apple Stanford EE Ph.D. Fellowship. The work of Kathleen Feng was supported by the Department of Defense (DoD) through the National Defense Science and Engineering Graduate (NDSEG) Fellowship Program. *(Kathleen Feng and Taeyoung Kong contributed equally to this work.) (Corresponding author: Kathleen Feng.)*

Kathleen Feng, Taeyoung Kong, Kalhan Koul, Jackson Melchert, Alex Carsello, Qiaoyi Liu, Gedeon Nyengele, Maxwell Strange, Ankita Nayak, Jeff Setter, Kavya Sreedhar, Po-Han Chen, Nikhil Bhagdikar, Zach A. Myers, Brandon D'Agostino, Pranil Joshi, Stephen Richardson, Christopher Torng, Mark Horowitz, and Priyanka Raina are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: kzf@stanford.edu).

Keyi Zhang and James Thomas are with the Department of Computer Science, Stanford University, Stanford, CA 94305 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2023.3313116.

Digital Object Identifier 10.1109/JSSC.2023.3313116

I. Introduction

HARDWARE accelerators have emerged as the key method to improve performance and energy efficiency of applications, as Moore's law slows down. However, application changes quickly result in outdated accelerators. Accelerators designed for specific image processing, computer vision, and machine learning (ML) applications [1], [2], [3], [4], [5], [6] demonstrate high performance and efficiency when they are released but soon become obsolete, as new imaging or ML applications emerge.

Reconfigurable accelerators present a solution to balance the efficiency and performance offered by fixed-function, dedicated accelerators and the adaptability needed to support application changes. Field-programmable gate arrays (FPGAs) have demonstrated utility as reconfigurable accelerators and have been used for accelerating various application domains [7], [8], [9], [10], [11]. However, programmability comes with energy, delay, and area overheads. Reconfiguration speed affects how often the accelerator can switch applications and repurpose resources that may be sitting idle. In an FPGA, reconfiguration takes tens of milliseconds [12], which prevents users from switching applications in real time. Memory control logic is often implemented in the reconfigurable fabric itself, which is inefficient relative to specialized memory control logic. General purpose direct memory access (DMA) engines can be used but are less performant. Finally, reconfigurable accelerators have introduced dedicated hardware, such as digital signal processors and artificial intelligence engines [13] in order to close the gap with application-specific integrated circuits (ASICs), but it often goes underutilized for applications that do not need those operations.

Amber [14] is a reconfigurable system-on-chip (SoC) that overcomes the challenges faced by traditional reconfigurable accelerators and targets dense linear algebra applications, including image processing, computer vision, and ML. The core of Amber is a coarse-grained reconfigurable array (CGRA) of processing element (PE) and memory tiles for application acceleration. To reduce area, energy, and the complexity of running applications on a reconfigurable chip, Amber features the following contributions.

1) Fast dynamic partial reconfiguration (DPR) that allows the SoC to be reconfigured to run new applications very quickly and run several independent kernels in parallel.
2) Efficient streaming memories to support affine access patterns, common in dense linear algebra applications.
3) Low-overhead, distributed implementation of necessary, but infrequently used, complex arithmetic operations.
4) Hierarchical physical design with low-overhead river routing for the clock and other global signals.
5) Use of an agile accelerator–compiler design flow in which the compiler updates automatically with hardware changes, enabling continuous application-level evaluation of the hardware–software system.

Section II provides background on CGRAs and agile hardware design. The Amber architecture is described in Section III. Section IV covers the global buffer (GLB) and DPR. Section V describes the streaming memory controllers for affine patterns. Section VI explains how Amber supports complex operations with low overhead. Section VII covers physical design of Amber, and Section VIII describes the hardware design and compilation flow. Finally, Section IX shows results and comparisons of Amber against other reconfigurable platforms.

II. Background

As accelerators have become increasingly common, research into reconfigurable architectures, such as CGRAs, and agile accelerator–compiler design techniques, has also become more popular. CGRAs are a class of reconfigurable, programmable, spatial-style architectures that serve as a midpoint between the flexibility offered by FPGAs and the energy efficiency of ASICs [15]. Instead of being configurable at the bit-level-like FPGAs, which use lookup tables (LUTs) to hold configuration data, CGRAs are configurable at a word-level granularity. They use coarse-grained processing and memory elements with a word-level interconnect, typically arranged in a tile-based manner [16], [17], [18]. While less efficient compared with ASICs, CGRAs are more flexible due to the built-in configurable nature of their different components.

Traditionally, designers use a waterfall approach to build accelerators, in which they first study an application, create a hardware specification, perform logical and physical design, fabricate the hardware, and finally develop the software compiler to map the application to the hardware. However, a waterfall approach makes it hard for designers to adapt to changing application requirements. In contrast, an agile accelerator–compiler design approach borrows from agile software methodologies and allows the designer to make incremental updates to the hardware and corresponding software tool chain to create an accelerator [19]. Recently, an agile approach to designing CGRAs has been introduced, which couples CGRA hardware generation with automatic application compiler updates [20]. In this article, we demonstrate a CGRA architecture designed with an agile accelerator–compiler design flow.

III. Amber Architecture

Amber consists of a CGRA, a GLB, and an ARM Cortex-M3 CPU [21] (Fig. 1). The CGRA has 384 PE tiles and 128 memory tiles, with every fourth column composed of memory tiles (Fig. 2). Each PE supports INT16 and BFloat16 [22], [23] operations and contains a 64-byte register file (RF) (Section VI). Each memory tile has a 4-KB SRAM (Section V). Each PE and memory tile has switch boxes (SBs) and connection boxes (CBs) to connect to the array interconnect. The GLB sits between off-chip DRAM and the CGRA and streams input, output, and intermediate data to and from the CGRA. Each of the 16 GLB tiles contains two 128-KB SRAM banks, load and store units, a configuration network, and a data network (Section IV). The ARM Cortex M3 processor subsystem manages application execution by sending bitstreams to the CGRA and directing data movement. It contains 128-KB of SRAM and DMA units that transfer data to and from off-chip DRAM through an ARM Thin Links interface [24].

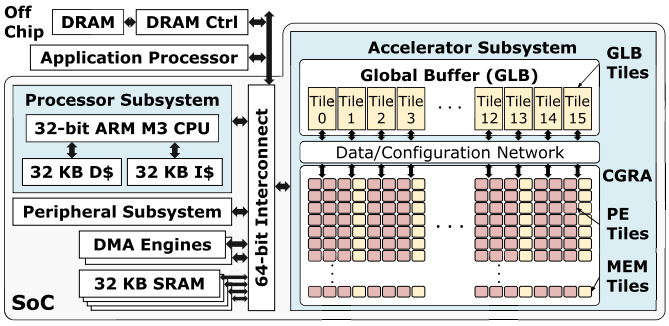


Fig. 1. Amber SoC architecture. The accelerator consists of a GLB and a CGRA. The processor subsystem controls application execution.

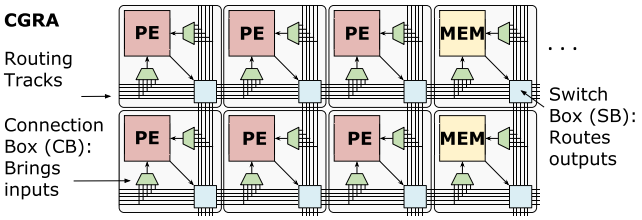


Fig. 2. PE and memory tiles in the CGRA, connected in an island-style architecture.

CGRA tiles communicate through statically configured 16- and 1-bit data interconnects (Fig. 3). SBs and CBs connect a tile core to the data interconnect. SBs have multiplexers that route data over five incoming and five outgoing 16-/1-bit routing tracks in each direction, and SB muxes optionally pipeline data. Adding more than five incoming or outgoing tracks to the SBs does not significantly improve application performance. However, decreasing the number of tracks from five hurts performance, because the router creates longer paths in the design due to the increased difficulty in routing the application [20]. Internally, SBs use the Wilton topology [25]. The Wilton topology rotates track numbers, meaning, for example, input west 2 (IW2) links to output north three (ON3). The Wilton topology increases routability, because it does not impose a restriction that a route from one side of the SB to the other side must use the same track number. Other topologies, such as disjoint [26], fail to route the targeted applications for this reason. CBs contain a 20:1 multiplexer that brings data from the interconnect tracks to the PE or memory core; one CB is instantiated for each input.

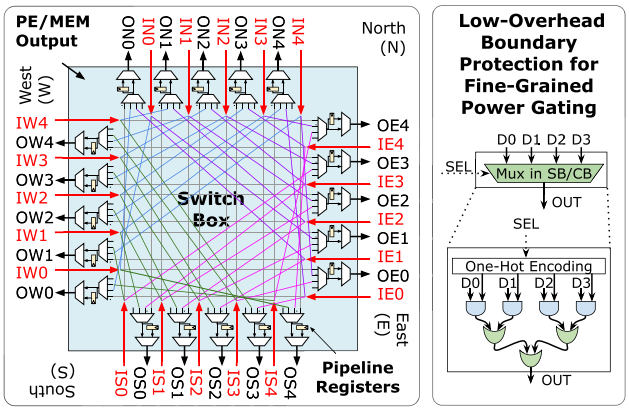


Fig. 3. CGRA SB features the Wilton topology to route data between tiles (left). Boundary protection logic for power-gating resides in the SB and CB multiplexers (right).

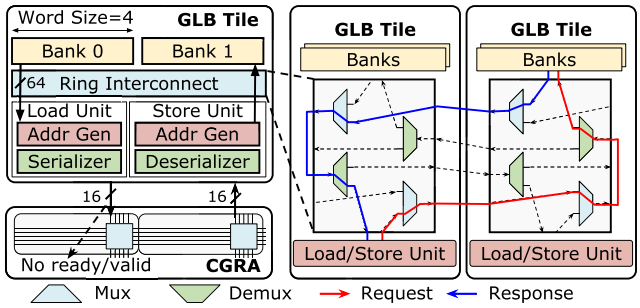


Fig. 4. GLB tile microarchitecture (left). By default, a load/store unit only accesses banks in the same GLB tile, bypassing the ring interconnect. Two GLB tiles connected via a ring interconnect (right). Fixed latency is guaranteed when a load unit accesses any bank in the two tiles as the request path (red) and the response path (blue) form a circle. Pipelining registers are not shown.

The CGRA supports tile-level power gating to turn unused tiles off. Each tile has power switches controlled by a configuration register. Instead of dedicated isolation cells, which add area and timing overheads, the multiplexers within the SBs and CBs contain boundary protection logic to prevent propagation of floating signals from an "off" domain into an "on" domain. The boundary protection multiplexers are based on [27] and [28] and have three stages: one-hot encoding, clamping, and an OR tree. One-hot encoding ensures that only one data port is selected, clamping all non-selected ports to zero. The OR tree produces the final output of the multiplexer. The boundary protection logic adds less than 1% area overhead and only 30 ps of delay.

IV. GLB Architecture Supporting DPR

Many workloads in imaging, vision, and ML domains are composed of multiple tasks that run concurrently or in a pipelined manner. For example, a computing system in an autonomous vehicle runs a camera pipeline to generate an RGB image, followed by several perception tasks, such as object detection. The GLB in Amber is designed to support concurrent streaming tasks with several input–output channels by splitting the GLB into 16 independent stream units called GLB tiles. It supports DPR, which can reconfigure up to 16 CGRA regions in parallel.

A. GLB Tile Microarchitecture

The GLB is a shared memory that streams data to and from the CGRA. Each of the 16 GLB tiles serves as an independent stream unit with its own memory (Fig. 4). Each GLB tile communicates with fixed columns of the CGRA, which avoids using an expensive crossbar between the GLB and the CGRA. This does not cause any limitations, since data can be routed to any part of the CGRA through its interconnect. A GLB tile has two 128-KB 64-bit wide (four-word wide) single-port SRAM banks, a load/store unit with an address generator (AG), and a serializer/deserializer to stream 16-bit CGRA words.

To run applications on Amber, the application compiler statically schedules data movement at compile time and extracts affine memory access patterns (Section V). The number of GLB tiles needed and configuration of the AGs are determined by the number of input–output channels and the access pattern of each channel, respectively. The compiler guarantees synchronization of data movement at the word level, eliminating the need for a handshaking interface between GLB tiles and the CGRA and saving area and energy.

Some channels in an application may need more memory than what one GLB tile can provide. In such cases, a GLB tile needs to access banks in other GLB tiles, which can introduce variable latency, as there are pipelining registers on the path to other tiles. However, the application compiler needs to know the memory access latency a priori to statically schedule data movement. To solve this, the GLB uses a configurable ring interconnect to enable a load/store unit to communicate with any bank in the neighboring GLB tiles while guaranteeing a fixed latency. When a request from a load unit enters the ring, it flows to the destination bank, and the bank then injects the response back into the ring to flow to the source load unit in a full circle with fixed latency (Fig. 4).

B. Dynamic Partial Reconfiguration

Amber can spatially map multiple tasks or temporally switch one task to another to maximize hardware utilization. This is done by DPR, which reconfigures part of the CGRA without affecting the rest. Amber uses the GLB to support high-throughput and parallel DPR with low hardware overhead. Amber shares GLB storage between application and configuration data and uses the GLB's parallel data ports to drive a high-speed, pipelined configuration network. Reusing the GLB for reconfiguration eliminates the need for dedicated storage and control logic for the reconfiguration system, reducing hardware overhead.

The CGRA has 22 656 32-bit configuration registers (90.6 KB), addressed by a 32-bit configuration address channel. Within the 32-bit configuration address, the most significant 16 bits indicate the *tile_id*, while the least significant 16 bits indicate the specific register within each tile. The configuration process in the CGRA operates on a columnwise basis, with each column having its own dedicated configuration channel. The channel comprises a 32-bit configuration address and the 32-bit configuration data. The configuration flows down the column, broadcasting to the tiles within it. Every tile in the column compares its *tile_id* to the most significant 16 bits of the configuration address, and

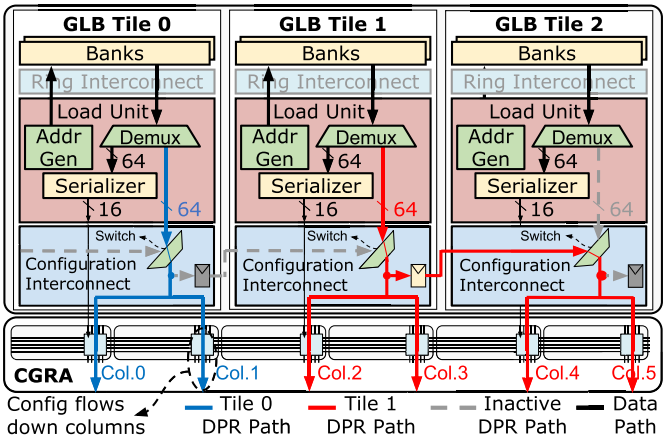


Fig. 5. DPR using GLB tiles and the configuration interconnect. GLB tile 0 only reconfigures the first two columns (blue), while GLB tile 1 can broadcast a bitstream to four columns through the configuration interconnect (red). Configuration flows down CGRA columns.

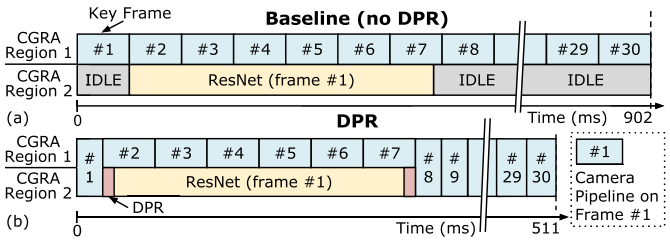


Fig. 6. Execution timeline of running a camera pipeline and ResNet-18 on the CGRA (a) without DPR and (b) with DPR.

only tiles with a matching *tile_id* store configuration data in corresponding registers. Therefore, even when the GLB broadcasts a bitstream to multiple columns (and ultimately multiple PE/memorys in the columns), only a single PE with a matching *tile_id* accepts the configuration data, while the remaining PEs disregard the bitstream. This simplifies the configuration network, as complex routers are not needed. Amber does not support configuring multiple PEs with the same bitstream in a single cycle.

Unlike other CGRA architectures [2], [29] that use a DMA or a serialized interface to configure the CGRA, Amber reuses the GLB to store bitstreams and stream them to CGRA columns. Multiple sets of bitstreams can be preloaded into the ample GLB memory and subsequently reused multiple times, eliminating the need for fetching them from off-chip DRAM for every reconfiguration. By default, each GLB tile is responsible for reconfiguring PE/memory tiles within the two columns positioned below the corresponding GLB tile. During the reconfiguration process, the GLB tiles assigned to reconfigure the corresponding subregion work in unison to stream their respective bitstreams. The load unit in each GLB tile reads the bitstream in four-CGRA-word (64 bit) chunks (32-bit configuration address and 32-bit configuration data) and sends it to a switch in the configuration interconnect every cycle, bypassing the serializer (Fig. 5), which then multicasts bitstreams to the two CGRA columns below. For instance, bitstreams from GLB tile 0 in Fig. 5 (blue) flow down to columns 0 and 1. In this default setting, bitstreams have to be sorted by their destination CGRA tiles and preloaded into corresponding GLB tiles before reconfiguration. While this preprocessing incurs some overhead in the processor, it can be amortized when the same bitstream is used for reconfiguration several times once preloaded into the GLB.

Using multiple GLB tiles for reconfiguration enables high-throughput, parallel DPR, where multiple regions of the CGRA can be reconfigured concurrently. However, some GLB tiles may not be available or may not have enough empty space to store bitstreams at runtime. In this case, a single GLB tile can deliver a bitstream to more than two CGRA columns by configuring switches in the configuration interconnect to chain GLB tiles and forward bitstreams to the right. For example, in Fig. 5, the load unit in GLB tile 1 injects the bitstream into a switch in the configuration interconnect, which is configured to broadcast to columns 2–5 (red). The circuit-switched configuration interconnect provides flexibility in multicasting the bitstream to a varying number of CGRA columns with low hardware overhead: one multiplexer and one set of pipeline registers per GLB tile.

Once the bitstream enters the CGRA, it flows down a configuration channel along with the clock signal (Section VII). This allows the bitstream to reach the bottom of the CGRA at 1 GHz. Due to parallel streaming of bitstreams using GLB tiles and a high-frequency configuration channel, Amber achieves a maximum DPR throughput that is 80× higher than DynPaC [29], which relies on a DMA controller, and 25 600× higher than Eyeriss [2], which utilizes a scan chain interface (64 GB/s versus 800 MB/s versus 2.5 MB/s).

The CGRA operates on large tiles of an input image, and Amber only allows reconfiguration when a kernel finishes the execution of a tile. The intermediate output of the kernel is stored in either memory tiles or GLB tiles, and reconfiguration does not change the contents of data stored in the memories. After reconfiguration, if a new kernel needs the outputs of a previous kernel, it continues processing the data stored in memory tiles or GLB tiles without saving it to temporary memories. If a new kernel is from a distinct application and needs memory space in memory tiles, data is temporarily saved to GLB tiles (or DRAM) and are restored to memory tiles later.

We evaluate the benefits of dynamic resource allocation on Amber using DPR with a synthetic autonomous system workload. In this workload, an image sensor produces a RAW image. A camera pipeline task processes every frame to generate an RGB image, which then goes through ResNet-18 for object detection if it is a key frame (e.g., every ten frames). In the baseline (no dynamic resource allocation), we partition the CGRA into two regions with an equal amount of hardware resources. For key frames, each CGRA region executes camera pipeline and ResNet, while for non-key frames, the ResNet-dedicated CGRA region sits idle (Fig. 6(a)). In contrast, Amber exploits DPR to dynamically reallocate hardware resources at runtime to minimize the latency (Fig. 6(b)). Amber allocates the entire CGRA to run a camera pipeline for frame 1. Once it finishes, as frame 1 is a key frame, DPR rapidly switches half of the CGRA to run ResNet. After ResNet finishes, the resources allocated for ResNet are reconfigured to run camera pipeline again until the next key frame. With DPR, the maximum frames per second (frames/s) Amber can achieve is 58.7, which is 1.8× faster than without DPR (33.3 frames/s).

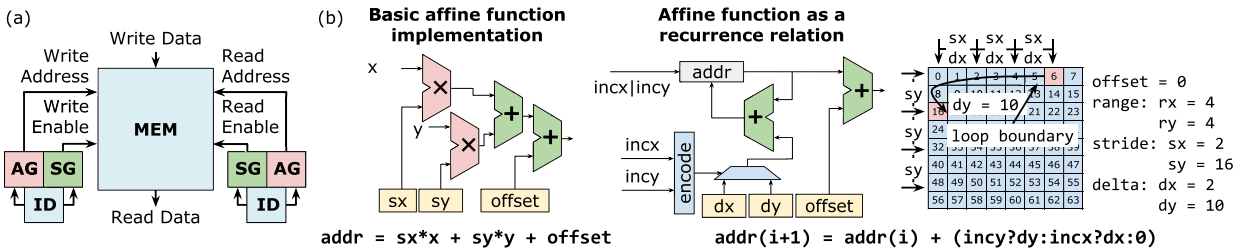


Fig. 7. (a) Example of a dual-port memory with streaming memory controllers for affine access patterns. The ID provides values to the AG and SG, which generate addresses and enable signals, respectively. (b) Affine access patterns are implemented using recurrence relations, which eliminate the use of expensive multipliers.

V. Streaming Memories for Affine Patterns

Accelerators often contain DMA engines to orchestrate data movement to and from the accelerator [30]. However, for the dense linear algebra applications that Amber targets, DMA engines are too general and lead to unnecessary area and energy overheads. Instead, because these applications demonstrate highly regular affine access patterns, we specialize the memory controllers within Amber to match these patterns. All on-chip memories (GLB, memorys, and RFs) in Amber have optimized internal affine memory controllers, whose parameters are extracted from the application by the compiler [31].

Affine access patterns can be expressed in the form of nested loops. For example, a two-level affine loop nest looks like

```
for y in range 0:ry
    for x in range 0:rx
        addr = sx*x + sy*y + offset
```

where the ranges (`rx` and `ry`) and strides (`sx` and `sy`) are configurable parameters set from the application. The GLB, memory, and PE RF support up to four-, six-, and two-level loop nests, respectively. The GLB loops over batches of images and the outer tiling loops, the memory is designed to have the most flexibility and supports the most number of loop configurations, and the PE RF supports only two levels due to its small size (32 16-bit words) and use as a local scratchpad inside the PE. We connect the affine memory controllers to the read and write ports of foundry-provided SRAMs (Fig. 7). Each controller consists of three components. The iteration domain (ID) specifies the range of memory operations (`0:rx` and `0:ry`). The ID is implemented by a set of counters, one for each loop in the pattern. The AG computes the address as an affine expression of strides and ID counter values. Finally, the schedule generator (SG) produces the read or write enable as an affine expression of a set of strides and ID counter values.

A. Streaming Memory Optimizations

Internal memory controllers in memory save area and energy compared with using PEs for address generation and eliminate sending memory requests across the interconnect. The hardware implementation of the streaming memory controllers contains a number of optimizations to further reduce overhead. To eliminate the use of multipliers, the affine patterns are calculated as recurrence relations, which use deltas (`dx` and `dy`) calculated from the strides and ranges (Fig. 7(b)). The deltas represent how much the output of the multiplier increases with each update.

The GLB and memory contain additional optimizations. They use wide-fetch SRAMs; each SRAM word is 64 bits wide, which fits four 16-bit CGRA words. A wide-fetch SRAM has a lower access energy per byte, 0.81 versus 1.65 pJ for single fetch (16 bits), for the same capacity 4-KB SRAM in 16-nm technology. Using a wide-fetch SRAM in memory allows us to expose two input and two output ports by aggregating inputs from the CGRA with serial in, parallel out (SIPO) buffers and separating SRAM words into CGRA words with parallel in, serial out (PISO) buffers. The memory tile also shares streaming memory controllers when possible across different internal modules. Reads from the SIPO buffers are always followed by writes to the SRAM, and reads from the SRAM are followed by writes to the PISO buffers, which allows them to share the same ID. Fig. 8 shows the final memory architecture.

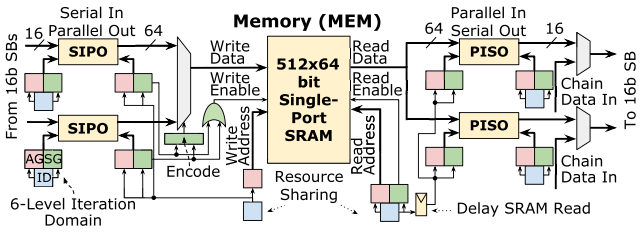


Fig. 8. Memory tile with a wide-fetch SRAM, streaming memory controllers, and SIPO and PISO buffers.

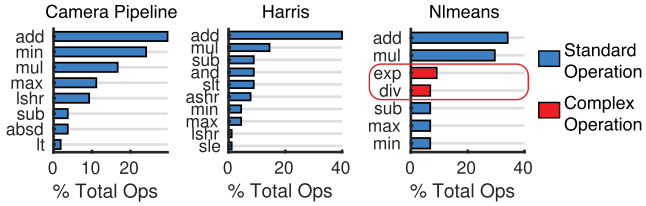


Fig. 9. Operations in camera pipeline, Harris corner detector, and non-local means (nlmeans). The circled complex operations account for 15% of the total operations in nlmeans.

VI. Low-Overhead Complex Arithmetic

Image processing, computer vision, and ML applications require, but infrequently use, complex arithmetic operations, including BFloat16 division, natural logarithm, sine, and exponential. For example, in non-local means (nlmeans), which is used in image denoising, 15% of all operations are complex (Fig. 9). However, Harris corner detector and camera pipeline do not contain any complex operations. The CPU is able to perform these operations, but going to the CPU incurs a high performance penalty. On the other hand, including complex operations in the ALU of the PE is very expensive in terms of area. Amber takes a unique approach toward supporting com-

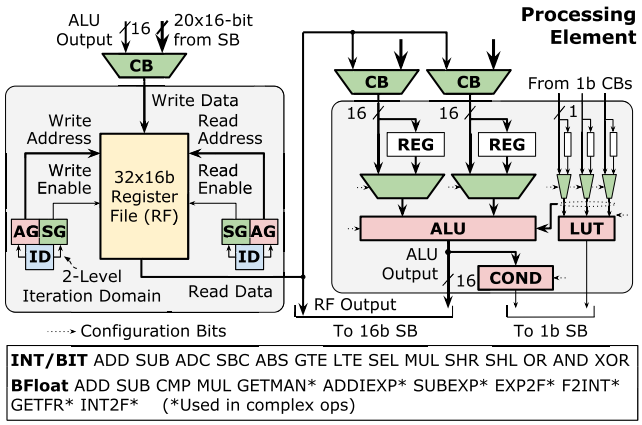


Fig. 10. PE architecture. Inputs come from other tiles or from the local RF. The PE has INT16 and BFloat16 operations, as well as additional operations to support complex arithmetic.

plex arithmetic operations and chains multiple tiles together to approximately perform the equivalent operations with very little area overhead.

The PE core within the PE tile has an ALU and a LUT to perform arithmetic and bit operations, respectively. It also contains a 64-byte RF as local memory (Fig. 10). The 16-bit inputs to the ALU come from either the CBs or from the RF. The outputs from the ALU go to the SB to be sent to another tile or are written into the RF. The 1-bit inputs to the LUT are taken from the 1-bit CB, and the outputs from the LUT are sent to the 1-bit SB to send to other tiles. The PE supports INT16 and bit operations, as well as BFloat16 operations.

In addition, the PE contains supplementary operations that, with multiple chained PE and memory tiles, implement complex arithmetic operations. The supplementary operations manipulate the BFloat16 representation $B = 1.f \times 2^x$, where B is the BFloat16 number, with 7 bits for the mantissa, f, and 8 bits for the exponent, x (the bias is not shown). The PE has eight supplementary functions. `GETMAN` returns the mantissa f. `ADDIEXP` adds an integer to the exponent x. `SUBEXP` subtracts the exponents of two BFloat16 values. `EXP2F` returns the exponent x as a BFloat16. `F2INT` converts BFloat16 into INT16 by rounding the numeric value to the nearest integer. `GETFR` returns the fractional part of the BFloat16 numeric value. `INT2F` converts INT16 to BFloat16. The supplementary operations add only 0.2% area in each PE tile (8124- versus 8108-μm^2 layout area).

Amber performs complex operations by using multiple PEs and memorys together (Fig. 11). For example, division is equivalent to multiplying, using one PE, the dividend by the reciprocal of the divisor. The reciprocal is found using one PE to get the mantissa, one memory to lookup the reciprocal of the mantissa, and one PE to subtract the original exponent from the reciprocal of the mantissa's exponent. This method of performing complex operations has minimal effect on accuracy; for nlmeans, the output image from Amber has a structural similarity index of 0.967 (with the maximum value being 1) with the output image from a CPU. Applications that use complex operations and operate in INT16 must first convert to BFloat16 and then convert back to INT16 after performing the operation.

Compared with a CGRA that supports these complex operations entirely in one PE, the Amber PE is 37.8% smaller in area (13 056- versus 8124-μm^2 layout area). However, the area required to perform each complex operation is much greater, since it requires using more than one PE. Fig. 12 shows the area trade-off between having a CGRA with dedicated complex operations in each PE and the Amber CGRA, analyzed across a variety of applications. For the majority of applications, which do not have complex operations, up to 34.5% less area is used on the CGRA with the approach taken by Amber. For nlmeans, which has exponentials and divisions, Amber uses 148 more tiles and 64.7% more area. However, Amber is still able to fully accelerate nlmeans on the CGRA, which is 29.7× faster than offloading the complex operations to the CPU.

VII. Physical Design Methodology

Scaling the physical design of a CGRA from a small array (4×4 and 4×8) [32], [33], [34] to the 16×32 array in Amber requires taking a bottom-up hierarchical approach to reduce physical design time. However, this is challenging, because the physical design tools do not have complete knowledge of the design. To address this, we perform special handling of clock distribution, global signals, and timing constraints, as described below.

The GLB, PE, and memory tiles are laid out independently and converted into macros, which are then replicated and repeated to create the CGRA and GLB. The PE and memory tiles have the same height to make it easy to lay out the tiles in the array, with the memory being wider than the PE. Each column has the same tile type. In the CGRA, the tiles are fully abutted to reduce area. The GLB and CGRA are connected to the processor subsystem at the top level to form the SoC.

Replicating tiles creates challenges when each tile must be treated differently. For example, each tile needs a tile ID to uniquely address it for configuration, yet each replicated macro is internally the same. Furthermore, abutment leaves no space between tiles to assign tile IDs. Instead, we observe that tile IDs are constants that do not switch and build each tile with a set of output pins that drive constants with a "GSVSG" pattern—signals ("S") with alternating logic 0 ("G") and logic 1 ("V") constants. At the tile array level, the tile ID is formed by drawing small metal strips to the correct logic levels. Our approach for assigning tile IDs requires no additional standard cells or area overhead and preserves tile abutment.

Another consequence of tile abutment is that there is no space between tiles to insert clock buffers. Rather than using a balanced clock tree, Amber distributes clocks down each CGRA column in clock rivers (Fig. 13). The naïve clock river approach sends the clock down both the PE and memory columns, but clock skew accumulates between columns of different tile types, a key challenge in the physical design of heterogeneous spatial arrays. Even with a small skew difference, e.g., 4 ps (20-ps PE delay versus 24-ps memory delay), after many tiles, the clock will reach the bottom of the PE columns before the memory columns. Instead, in our design, the clock only travels down a single tile type, the PE

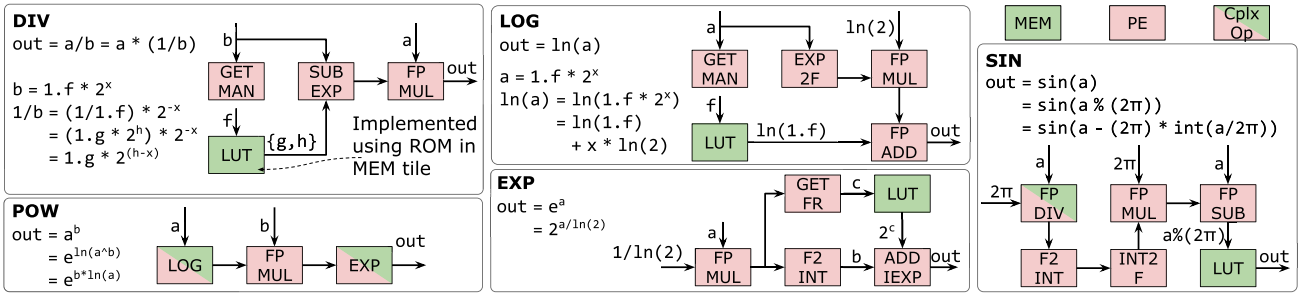


Fig. 11. Using multiple PE and memory tiles, Amber supports BFloat16 division (DIV), logarithm (LOG), exponential (EXP), sine (SIN), and power (POW).

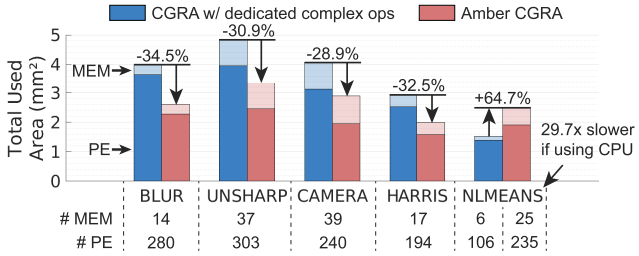


Fig. 12. Area utilization trade-offs for complex arithmetic support. Nlmeans contains complex operations, while the other applications do not. Without complex arithmetic support on the CGRA, nlmeans would be 29.7× slower.

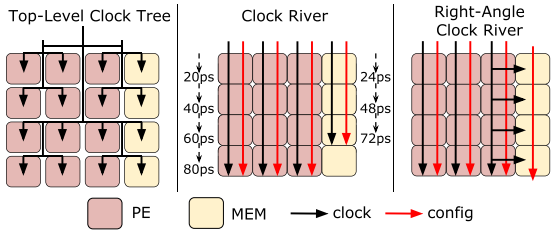


Fig. 13. Clock and configuration signal distribution method (right). The clock flows down the PE columns and travels right to the memory tiles, which helps match pass-through delays between the PE and memory tiles compared with a simple clock river approach (middle). A traditional clock tree does not allow creating a fully abutted layout (left).

tiles, and the PE tiles pass the clock to the memory tiles on the right. The right-angle clock river eliminates the need to precisely match delays through the PE and memory tiles, and we are able to send the CGRA configuration data together with the clock, which enables low-overhead fast configuration. However, the delay of the clock path through the PE to the right-adjacent memory is around 50 ps, so if a memory to PE path lies in the critical path through an application, the right-angle clock river slows it down by that much.

Configuration data is a part of the CGRA's global signals, which also include a read path to read out configuration data. To successfully meet timing at the CGRA level, each tile has constraints to handle global signals and break combinational loops that form across multiple tiles. A clock passing through a PE tile is constrained with a max delay of 50 ps. Configuration signals pass through each tile with a max delay of 30 ps after the clock. With these tight constraints and the configuration signals traveling with the clock, Amber only needs one set of pipeline registers halfway down the rows to configure at 1 GHz. The configuration readout path is more relaxed, with a 300-ps pass-through delay, since it is designed to be a multicycle debug path that does not need to meet high-performance requirements.

Finally, putting the tiles together in an array creates combinational loops in the design, since the tiles have purely combinational paths from input to output. Using the `set_case_analysis` command, we activate the pipeline registers in each tile's SB to break the loops during liberty (lib) file generation, a format that defines timing and power properties for a generated macro. Using the customized lib files, the timing tools will not see inter-tile combinational paths when running top-level timing checks.

VIII. Accelerator–Compiler Design Flow

To design the CGRA, we use the open-source domain-specific language (DSL)-based accelerator–compiler codesign flow from [20], which uses high-level DSLs to design the hardware. The flow generates the application compiler collateral automatically from the hardware specification (Fig. 14). The PE, memory, and interconnect in the CGRA are specified in their own DSLs—PEak, Lake, and Canal, respectively. Each DSL has its own compiler to generate both the hardware RTL description and the collateral for the application compiler to map applications onto the hardware. The collateral includes PE rewrite rules, which dictate how to map operations onto each PE tile, memory rewrite rules, which transform memory accesses into memory tile access patterns, and the routing graph, which describes how tiles are connected to each other.

The application compiler flow transforms a Halide program into a bitstream that configures the CGRA. Halide [35] is a DSL in C++ that divides a program into an *algorithm* and a *schedule*. The algorithm describes how to compute each output pixel as a function of the inputs, and the schedule describes the order in which the intermediates and output pixels are computed. Koul et al. [20] extend the Halide compiler to target CGRAs. Given an application, the polyhedral scheduler first transforms the program into a dataflow graph of operations. The mapper converts the dataflow graph into a graph of PE/memory tiles, which is then placed and routed on a physical CGRA. Pipelining activates registers in the interconnect on long paths to increase the maximum clock frequency. Finally, the bitstream generator produces the configuration bitstream. The Halide compiler also produces the ARM M3 CPU code, enabling application execution on Amber.

IX. Results

Amber is fabricated in Taiwan Semiconductor Manufacturing Company (TSMC) 16-nm FinFET technology with a

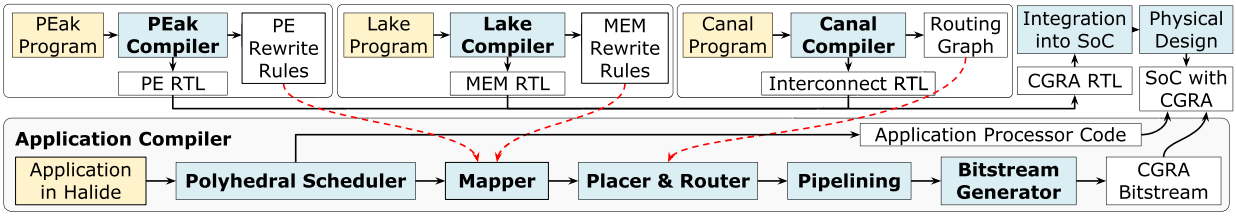


Fig. 14. Agile accelerator–compiler codesign flow that uses high-level DSLs to describe each of the components of the CGRA. The end-to-end flow for compiling and mapping Halide applications onto Amber uses collateral from each of the hardware DSLs.

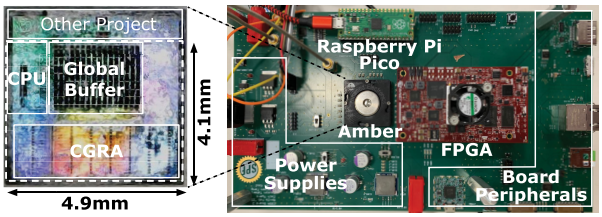


Fig. 15. Amber die photograph (left) and measurement and testing setup (right).

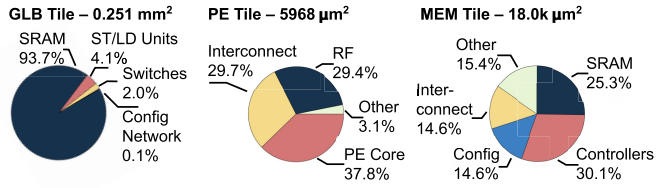


Fig. 16. Standard cell area breakdown of the GLB, PE, and memory tiles.

TABLE I
LAYOUT AND STANDARD CELL AREA (AFTER SIGNOFF) FOR AMBER

	Layout Area (mm^2)	Std Cell+SRAM Area (mm^2)	Density
Amber	20.1	–	–
Global Buffer	4.91	4.69	95.3%
GLB Tile (×16)	0.292	0.251	85.9%
GLB Controller	0.007	0.004	47.1%
CGRA	7.24	6.14	84.7%
MEM (×128)	0.024	0.018	74.0%
PE (×384)	0.008	0.006	73.5%
Processor System	0.256	–	–
Fill	7.69	–	–

TABLE II
ENERGY BREAKDOWN FOR FULLY UNROLLED, PIPELINED BLUR

	Energy (mJ/frame)	Percentage
GLB (14 / 16)	1.52	56.2%
PE (280 / 384)	0.641	23.7%
MEM (56 / 128)	0.388	14.3%
Processor	0.0616	2.28%
Other	0.0946	3.49%
Total	**2.71**	**100%**

total area of 20.1 mm^2. Fig. 15 shows the die photograph and testing setup. The GLB closes timing at 1 GHz, while the PE and memory are designed to run at a maximum frequency of 909 MHz. In the testing setup, Amber is housed in the socket in the middle of the board, and the application configuration bitstreams are generated off chip. The Raspberry Pi Pico programs on-board power supply modules to generate the core and I/O voltage and controls the clock generator. The nominal core voltage is 0.9 V, and all reported results are measured at 0.9 V unless otherwise noted.

Table I shows the breakdown of the layout area and standard cell area after signoff, with the CGRA and GLB comprising a majority of the area. Fig. 16 shows the area breakdown of the GLB tile, PE tile, and memory tile. Density is the percentage of the total layout area that is comprised of standard cells and macros, as opposed to fill cells. The densities of the GLB and CGRA are high, because they consist mostly of macros of GLB tiles and PE/memory tiles, respectively. The GLB controller density is relatively low, because it is a small module, and it was not necessary to make it very area efficient. The streaming memory controllers take up 30% of the area in the memory tiles. The area of the CGRA is 50% memory and 50% PE; the memory is 3× the size of the PE. The processor subsystem is very small and makes up 1.3% of the total area.

A. Unrolling and Pipelining Applications

The schedule of an application, that is, how the application is mapped onto the CGRA, affects its runtime, energy consumption, and energy-delay product (EDP). We employ scheduling techniques, such as unrolling and pipelining, to increase performance. Fig. 17 shows the effect of unrolling and pipelining. Unrolling (or parallelizing) the outer loop of an application kernel effectively replicates the kernel and increases throughput. The maximum unrolling is limited by the number of GLB tiles, the number of PE tiles, the number of memory tiles, and/or routing resources. Fully unrolling to the maximum unrolling/parallelization factor results in 2.0×–12.1× runtime improvement, 1.1×–7.1× lower energy consumption, and 2.2×–86× EDP improvement.

Pipelining the application by using registers in the interconnect, PEs, and memorys further increases application performance on Amber. The amount that each application is pipelined varies, as it depends on how the application is mapped onto the CGRA and the critical path through the array [36]. Pipelining results in 6.4×–24.7×, 1.8×–9.1×, and 12×–221× improvements in performance, energy, and EDP, respectively. Table II breaks down the energy consumed by the unrolled and pipelined blur application, and Table III shows the amount of resources used by all the unrolled and pipelined applications. Temporal occupancy (utilization) is the percentage of cycles for which the CGRA is active during the execution of the entire application. The size of the input image for each application is larger than the storage available in the memory tiles, so the image is split into several tiles, and each tile is streamed through the application mapped on the CGRA. While we try to fully overlap computation time on the CGRA with the data loading and storing time from/to the GLB in steady state, the CGRA is idle during the time the first tile is loaded and the last tile is stored. Furthermore, the CGRA is

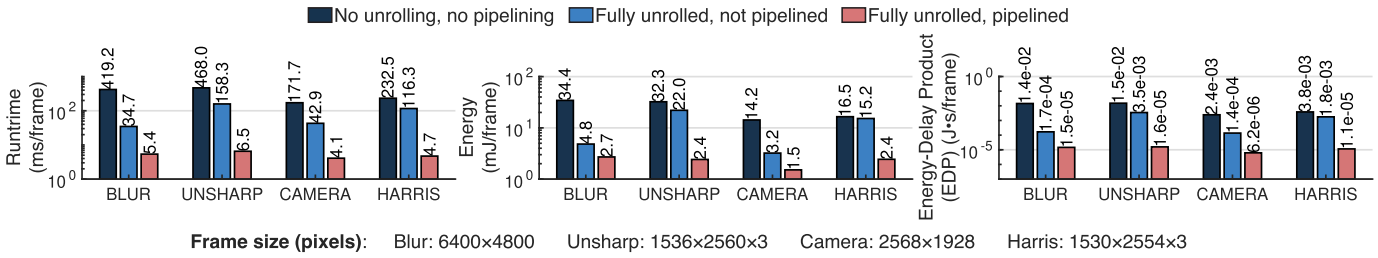


Fig. 17. Effect of unrolling and pipelining on runtime, energy, and EDP.

TABLE III
APPLICATION RESOURCE UTILIZATION AND FREQUENCIES

Application	Blur	Unsharp	Camera	Harris	ResNet-18
Output Rate (pixels/cycle)	14	9	3	2	4
Temporal Occupancy	72%	83%	73%	83%	Table V
Frequency (MHz)	415	260	320	425	200
# PE / 384	280	303	294	194	272
# MEM / 128	56	36	34	12	56
# GLB / 16	14	9	12	6	12
# 1-bit Routing Tracks / 10240	560	296	417	348	450
# 16-bit Routing Tracks / 10240	2128	1892	1410	2236	2684

TABLE IV
XILINX VIRTEX ULTRASCALE+ VU9P FPGA PROPERTIES COMPARED WITH AMBER

	FPGA		Amber
Processing	Logic Cells	2586 K	384 PEs
	Flip-Flops	2364 K	
	DSP Slices	6840	
	LUTs	1182 K	
Memory	Block RAM	75.9 Mb	4.5 MiB
	UltraRAM	270 Mb	

TABLE V
LATENCY AND ENERGY COMPARISON OF AMBER WITH A XILINX VIRTEX FPGA ON RESNET-18 LAYERS

	Latency (ms)		Energy (mJ)		Temporal Occupancy
Layer	Amber	FPGA	Amber	FPGA	Amber
conv2_x	4.44	4.53	1.51	13.3	84.8%
conv3_1	2.24	2.27	0.75	6.7	84.9%
conv3_x	4.44	4.53	1.51	13.3	84.8%
conv4_1	2.28	2.27	0.78	6.7	82.4%
conv4_x	4.34	4.53	1.47	13.3	86.8%
conv5_1	2.33	2.27	0.79	6.7	80.6%
conv5_x	5.40	5.91	1.84	17.3	69.7%

idle when we switch the bitstream in the case of DPR. These lead to a temporal occupancy of less than 100%. In addition, the frequency that each application runs at is dependent on how the application is placed, routed, and pipelined on the CGRA, which is similar to FPGAs, where applications rarely run at the maximum achievable frequency of the FPGA. Routes between sequential elements may span across multiple tiles, which will decrease the frequency since our CGRA interconnect allows for single-cycle multi-hop connections, as opposed to interconnects that are single cycle single hop.

B. Comparison Against Other Platforms

We compare Amber against an ARM Cortex A57 CPU, an Intel Xeon CPU, an NVIDIA Tesla K40 GPU, and a Xilinx Virtex Ultrascale FPGA. The FPGA is in the same technology node as our CGRA, 16-nm FinFET, and its properties are summarized in Table IV. Applications running on the CPUs and GPU use Halide's autoscheduler [37] to find the best application schedule for those platforms. Amber achieves 160×–1200×, 678×–3902×, 498×–988×, 12×–152×, and 20×–107× better EDP over the Cortex A57, 1-core and 12-core Xeon CPUs, GPU, and FPGA, respectively (Fig. 18). For ResNet-18 layers in INT16 precision, Amber uses up to 9.4× less energy and has up to 10.3× lower EDP than the FPGA (Table V).

X. RELATED WORK

Recently published, state-of-the-art, programmable accelerator chips include 2 × 2 dielets with CGRAs for signal processing and linear algebra [38], an SoC with an embedded FPGA [39], a RISC-V vector machine [40], and a 496-core RISC-V processor [41]. Rathore et al. [38] have more than double the number of PEs, and their PEs are denser and more specialized in precision and operations. Even so, Amber shows comparable energy efficiency. Whatmough et al. [39] see its efficiency determined by how much of its resources can be utilized. Schmidt et al. [40] support a variety of floating point and integer operations but latency varies across functional units. Finally, Rovinski et al. [41] send memory requests across the interconnect to the on-chip SRAMs. With its DPR, internal streaming memory controllers, and low-overhead complex arithmetic, Amber is 1.7× more energy efficient with 36.7× better throughput (Table VI).

Prior work has also explored using CGRAs as programmable accelerators and an alternative to fixed-function accelerators. Previously proposed CGRAs include Plasticine [18], ultra-elastic CGRAs [32], CGRA-ME [42], DySER [16], and ADRES [17]. While they are programmable

TABLE VI
SUMMARY OF ACHIEVED PERFORMANCE AND COMPARISON WITH STATE-OF-THE-ART RECONFIGURABLE ACCELERATORS

	This Work	ISSCC 2022 [38]	VLSI 2019 [39]	ISSCC 2021 [40]	VLSI 2019 [41]
Architecture	SoC with CGRA	2×2 dielet with CGRA	SoC with FPGA	Multicore Vector CPU	Multicore CPU
Node	TSMC 16-nm	16-nm	TSMC 16-nm	TSMC 16-nm	TSMC 16-nm
Area (mm^2)	20.1	25	25	24.01	15.25
Precision	BF16, INT16/32/64	Fixed 16	FP16/32/64, INT16/32/64	FP16/32/64, INT32/64	INT32
SRAM (MiB)	4.5	0.06	9.025	4.5	3.875
Voltage (V)	0.84-1.29	0.35-0.9	0.5-1.0	0.55-1.0	0.60-0.98
Frequency (MHz)	955 @ 1.29 V	1100 @ 0.8 V	> 1000	1440	10-1400
Peak GOPS	367 @ 1.29 V	3449.6	10-56.2	368.4	695
GOPS/W	538 @ 0.84 V	568 @ 0.8 V	312.4	209.5	93.04

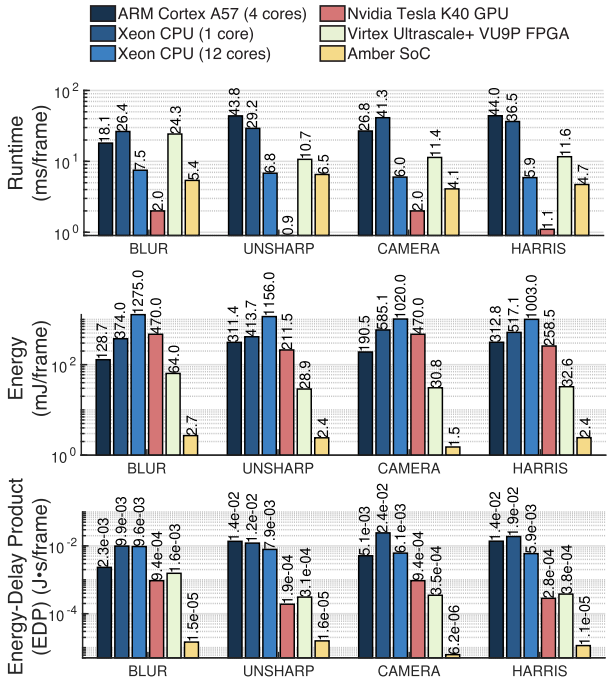


Fig. 18. Performance, energy, and EDP compared with CPU, GPU, and FPGA for image processing and computer vision applications.

accelerators, they are evaluated in simulation or on FPGAs. In addition, the application compilers for previous CGRAs are designed manually, as opposed to the compiler for Amber, which is automatically updated from the hardware specification.

XI. CONCLUSION

Amber is a reconfigurable SoC for flexible and efficient acceleration of imaging, vision, and ML applications. The demonstrated techniques—DPR, efficient streaming memories, and low-overhead complex arithmetic—achieve efficient *domain* (rather than single application) *acceleration*. In addition, due to its repetitive architecture, Amber employs a hierarchical-based physical design approach and uses a right-angle clock river to efficiently distribute the clock. Amber is designed using an agile accelerator–compiler design flow, which allows for continuous application-level evaluation of the hardware design.

REFERENCES

[1] B. Zimmer et al., "A 0.32–128 TOPS, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm," *IEEE J. Solid-State Circuits*, vol. 55, no. 4, pp. 920–932, Apr. 2020.

[2] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[3] Y. Chen et al., "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.

[4] Q. Zhang et al., "A 22 nm 3.5 TOPS/W flexible micro-robotic vision SoC with 2 MB eMRAM for fully-on-chip intelligence," in *Proc. IEEE Symp. VLSI Technol. Circuits*, Jun. 2022, pp. 72–73.

[5] S. Komatsu, M. Kimura, A. Okawa, and H. Miyashita, "Milbeaut image signal processing LSI chip for mobile phones," *Fujitsu Sci. Tech. J.*, vol. 49, no. 1, pp. 17–22, Jan. 2013.

[6] G. P. Stein, E. Rushinek, G. Hayun, and A. Shashua, "A computer vision system on a chip: A case study from the automotive domain," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR) Workshops*, Sep. 2005, p. 130.

[7] A. Putnam et al., "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014, pp. 13–24.

[8] C. Farabet et al., *Large-Scale FPGA-Based Convolutional Networks*. Cambridge, U.K.: Cambridge Univ. Press, 2011.

[9] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-HD: Fast flexible FPGA-based framework for refreshing hyperdimensional computing," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*. New York, NY, USA: Association for Computing Machinery, Feb. 2019, pp. 53–62, doi: 10.1145/3289602.3293913.

[10] L. Wu et al., "FPGA accelerated INDEL realignment in the cloud," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 277–290.

[11] Y. Chen, R. Yin, B. Gao, L. Peng, and M. Gong, "Ray tracing on single FPGA," in *Proc. IEEE Asia–Pacific Conf. Image Process., Electron. Comput. (IPEC)*, Apr. 2021, pp. 1290–1294.

[12] M. Nguyen, R. Tamburo, S. Narasimhan, and J. C. Hoe, "Quantifying the benefits of dynamic partial reconfiguration for embedded vision applications," in *Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2019, pp. 129–135.

[13] Xilinx. (2020). *Versal: The First Adaptive Compute Acceleration Platform (ACAP)*. [Online]. Available: https://docs.xilinx.com/v/u/en-U.S./wp505-versal-acap

[14] A. Carsello et al., "Amber: A 367 GOPS, 538 GOPS/W 16 nm SoC with a coarse-grained reconfigurable array for flexible acceleration of dense linear algebra," in *Proc. IEEE Symp. VLSI Technol. Circuits*, Jun. 2022, pp. 70–71.

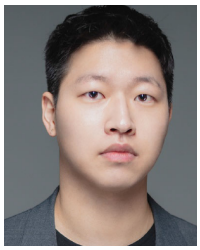
[15] A. Vasilyev, N. Bhagdikar, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz, "Evaluating programmable architectures for imaging and vision applications," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–13.

[16] V. Govindaraju et al., "DySER: Unifying functionality and parallelism specialization for energy-efficient computing," *IEEE Micro*, vol. 32, no. 5, pp. 38–51, Sep. 2012.

[17] F.-J. Veredas, M. Scheppler, W. Moffat, and B. Mei, "Custom implementation of the coarse-grained reconfigurable ADRES architecture for multimedia purposes," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2005, pp. 106–111.
[18] R. Prabhakar et al., "Plasticine: A reconfigurable architecture for parallel patterns," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 389–402.
[19] R. Bahr et al., "Creating an agile hardware design flow," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.
[20] K. Koul et al., "AHA: An agile approach to the design of coarse-grained reconfigurable accelerators and compilers," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 2, pp. 1–34, Jan. 2023, doi: 10.1145/3534933.
[21] *ARM Cortex-M3*. Accessed: Feb. 9, 2023. [Online]. Available: https://developer.arm.com/Processors/Cortex-M3
[22] *The BFLOAT16 Numerical Format*. Accessed: Feb. 9, 2023. [Online]. Available: https://cloud.google.com/tpu/docs/bfloat16
[23] *BFLOAT16 Hardware Numerics Definition*. Accessed: Feb. 9, 2023. [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/bf16-hardware-numerics-definition-white-paper.pdf
[24] *ARM CoreLink TLX-400 Network Interconnect Thin Links Supplement to ARM CoreLink NIC-400 Network Interconnect Technical Reference Manual*. Accessed: Feb. 9, 2023. [Online]. Available: https://developer.arm.com/documentation/dsu0028/f
[25] M. I. Masud and S. J. E. Wilton, "A new switch block for segmented FPGAs," in *Field Programmable Logic and Applications*, P. Lysaght, J. Irvine, and R. Hartenstein, Eds. Berlin, Germany: Springer, 1999, pp. 274–281.
[26] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*. Boston, MA, USA: Addison-Wesley, 1993.
[27] A. Nayak et al., "A framework for adding low-overhead, fine-grained power domains to CGRAs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 846–851.
[28] A. Nayak et al., "Improving energy efficiency of CGRAs with low-overhead fine-grained power domains," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 2, pp. 1–28, Jun. 2023, doi: 10.1145/3558394.
[29] C. Tan et al., "DynPaC: Coarse-grained, dynamic, and partially reconfigurable array for streaming applications," in *Proc. IEEE 39th Int. Conf. Comput. Design (ICCD)*, Oct. 2021, pp. 33–40.
[30] H. E. Sumbul et al., "System-level design and integration of a prototype AR/VR hardware featuring a custom low-power DNN accelerator chip in 7 nm technology for Codec Avatars," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2022, pp. 1–8.
[31] Q. Liu et al., "Unified buffer: Compiling image processing and machine learning applications to push-memory accelerators," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 2, pp. 1–26, Mar. 2023, doi: 10.1145/3572908.
[32] C. Torng, P. Pan, Y. Ou, C. Tan, and C. Batten, "Ultra-elastic CGRAs for irregular loop specialization," in *Proc. IEEE Int. Symp. High-Performance Comput. Archit. (HPCA)*, Feb. 2021, pp. 412–425.
[33] 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.
[34] A. Parashar et al., "Triggered instructions: A control paradigm for spatially-programmed architectures," in *Proc. 40th Annu. Int. Symp. Comput. Archit. (ISCA)*. New York, NY, USA: Association for Computing Machinery, Jun. 2013, pp. 142–153, doi: 10.1145/2485922.2485935.
[35] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," in *Proc. 34th ACM SIGPLAN Conf. Program. Lang. Design Implement.* New York, NY, USA: Association for Computing Machinery, Jun. 2013, pp. 519–530, doi: 10.1145/2491956.2462176.
[36] J. Melchert, Y. Mei, K. Koul, Q. Liu, M. Horowitz, and P. Raina, "Cascade: An application pipelining toolkit for coarse-grained reconfigurable arrays," 2022, *arXiv:2211.13182*.
[37] R. T. Mullapudi, A. Adams, D. Sharlet, J. Ragan-Kelley, and K. Fatahalian, "Automatically scheduling Halide image processing pipelines," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 1–11, Jul. 2016, doi: 10.1145/2897824.2925952.
[38] U. Rathore, S. S. Nagi, S. Iyer, and D. Markovic, "A 16 nm 785 GMACs/J 784-core digital signal processor array with a multilayer switch box interconnect, assembled as a 2 × 2 dielet with 10 μm-pitch inter-dielet I/O for runtime multi-program reconfiguration," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 52–54.
[39] P. N. Whatmough et al., "A 16 nm 25 mm^2 SoC with a 54.5× flexibility-efficiency range from dual-core arm cortex-A53 to eFPGA and cache-coherent accelerators," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. 34–35.
[40] C. Schmidt et al., "An eight-core 1.44 GHz RISC-V vector machine in 16 nm FinFET," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 58–60.
[41] A. Rovinski et al., "A 1.4 GHz 695 Giga Risc-V Inst/s 496-core manycore processor with mesh on-chip network and an all-digital synthesized PLL in 16nm CMOS," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. 30–31.
[42] S. A. Chin et al., "CGRA-ME: A unified framework for CGRA modelling and exploration," in *Proc. IEEE 28th Int. Conf. Appl.-Specific Syst., Architectures Processors (ASAP)*, Jul. 2017, pp. 184–189.

Kathleen Feng (Graduate Student Member, IEEE) received the B.S.E. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2018, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2020, where she is currently pursuing the Ph.D. degree in electrical engineering under the supervision of Prof. Priyanka Raina.

Her current research focuses on domain-specific hardware architectures and hardware–software codesign.



Taeyoung Kong (Member, IEEE) received the B.S. degree in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2017, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2020, where he is currently pursuing the Ph.D. degree under the supervision of Prof. Mark Horowitz.

Specifically, he is working on architecture and runtime software for multi-tenancy support on coarse-grained reconfigurable arrays (CGRAs) and reconfigurable architectures. His current research focuses on domain-specific architectures and reconfigurable architectures for machine learning and image processing.

Kalhan Koul received the B.S. degree (Hons.) in electrical engineering and the B.A. degree from the Plan II Honors Program, College of Liberal Arts, The University of Texas at Austin, Austin, TX, USA, in 2018. He is currently pursuing the Ph.D. degree in electrical engineering with Stanford University, Stanford, CA, USA, under the supervision of Prof. Priyanka Raina.

He was an Intern at Apple Inc., Cupertino, CA, USA; Micron Inc., San Jose, CA, USA; and Silicon Labs, Austin. Specifically, he is working on automatically mapping applications, ranging from machine learning to image processing, onto coarse-grained reconfigurable arrays (CGRAs) and reconfigurable logic devices. His current research focuses on domain-specific hardware architectures and design methodology.

Jackson Melchert received the B.S. degree in electrical and computer engineering and computer science from the University of Wisconsin-Madison, Madison, WI, USA, in 2019. He is currently pursuing the Ph.D. degree in electrical engineering with Stanford University, Stanford, CA, USA, under the supervision of Prof. Priyanka Raina.

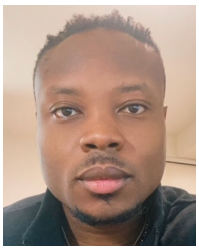
He is broadly interested in optimizing configurable hardware to approach the performance and efficiency of application-specific accelerators.

Alex Carsello received the B.S. degree in electrical engineering and computer engineering from Washington University in St. Louis, St. Louis, MO, USA, in 2017, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2020, where he is currently pursuing the Ph.D. degree in electrical engineering with the AHA! Agile Hardware Center under the supervision of Prof. Mark Horowitz.

He is interested in agile physical design tools and methodologies, reconfigurable computing, and domain-specific architectures for image processing and machine learning.

Qiaoyi Liu received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2017, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2020, where he is currently pursuing the Ph.D. degree in electrical engineering under the supervision of Prof. Mark Horowitz.

His research focuses on computer architecture and compilation for domain-specific accelerators, with a particular interest in optimizing schedules and memory mappings for compute-intensive tensor applications, such as image processing and deep learning.



Gedeon Nyengele received the B.S. degree in electrical engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2017. He is currently pursuing the Ph.D. degree in electrical engineering with Stanford University, Stanford, CA, USA, under the supervision of Prof. Mark Horowitz, with a focus on novel methodologies for system-on-chip design.

Maxwell Strange received the B.S. degree in computer engineering and computer science from the University of Wisconsin-Madison, Madison, WI, USA, in 2017.

His research focuses on developing infrastructure and tools to facilitate agile hardware development as part of the ongoing efforts by the AHA! Agile Hardware Center, Stanford University, Stanford, CA, USA. His research interests also include domain-specific hardware architectures, hardware/software codesign, and embedded systems design.

Keyi Zhang received the B.S. degree in computer science and engineering from Bucknell University, Lewisburg, PA, USA, in 2017, the M.S. degree in computer science from Stanford University, Stanford, CA, USA, in 2021, and the Ph.D. degree in computer science from Stanford University in 2022 under the supervision of Prof. Mark Horowitz.

He is currently a Lead Systems Engineer at EfficientAI, Santa Clara, CA, USA, a startup focusing on designing highly efficient and ultralow-power reconfigurable fabrics. His research interests include software and hardware codesign, compiler optimization, hardware generator frameworks, and debugging generated hardware systems.

Dr. Zhang was a recipient of the Apple Ph.D. Fellowship in integrated systems.

Ankita Nayak received the master's degree in computer engineering from Columbia University, New York, NY, USA, in 2011, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2023.

From 2011 to 2013, she worked at Intel, Santa Clara, CA, USA, on developing low-power design methodologies. In 2013, she joined Qualcomm, Santa Clara, CA, USA, where she led various research initiatives for enhancing power-performance-area for Snapdragon IPs. She is currently with the Wireless Research and Development Group leading on-device machine learning initiatives for 5G New Radio (NR) modems.

Jeff Setter received the B.S. degree in electrical and computer engineering from Cornell University, Ithaca, NY, USA, in 2015, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2023, with a focus on compiling halide applications to reconfigurable accelerators.

He is currently working at Google, Mountain View, CA, USA, on the compiler team for the machine learning accelerator on Pixel devices. His research interests include machine learning, image processing, compilers, software–hardware codesign, and domain-specific accelerators.

James Thomas received the S.B. in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2016, and the Ph.D. degree in computer science from Stanford University, Stanford, CA, USA, in 2022, under the supervision of Prof. Pat Hanrahan and Prof. Matei Zaharia in the AHA and DAWN groups.

He was an Intern at Xilinx, San Jose, CA, USA, and Databricks, San Francisco, CA, USA. He currently works at an artificial intelligence startup in Palo Alto, CA, USA. His research was on building tools to simplify the development of data processing applications on field-programmable gate arrays (FPGAs).

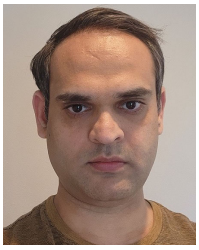
Kavya Sreedhar received the B.S. degree in electrical engineering and the B.S. degree in business, economics, and management from the California Institute of Technology, Pasadena, CA, USA, in 2019, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2021, where she is currently pursuing the Ph.D. degree in electrical engineering under the supervision of Prof. Mark Horowitz.

She was an Intern with Meta, Menlo Park, CA, USA; NVIDIA, Santa Clara, CA, USA; Apple, Cupertino, CA, USA; Microsoft, Redmond, WA, USA; and Intel, Santa Clara. Her research interests include hardware design for cryptography and machine learning applications.

Ms. Sreedhar received the Stanford's Knight-Hennessy Graduate Fellowship from 2019 to 2022 and the Quad Fellowship from 2023 to 2024 for her research.

Po-Han Chen (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering and computer science and the M.S. degree in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2016 and 2018, respectively. He is currently pursuing the Ph.D. degree in electrical engineering with Stanford University, Stanford, CA, USA, under the supervision of Prof. Priyanka Raina.

He was a Digital Circuit Designer at MediaTek, Hsinchu, Taiwan, where he worked on developing hardware architectures for image processing pipelines. Most of his previous works were related to computational photography algorithms, such as digital refocusing. He is interested in designing hardware accelerators. He is focusing on analyzing and designing the architecture of coarse-grained reconfigurable arrays (CGRAs) to create high-performance, energy-efficient, and reconfigurable computing platforms.



Nikhil Bhagdikar received the M.S. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2009. He is currently pursuing the Ph.D. degree in electrical engineering with Stanford University, Stanford, CA, USA, under the supervision of Prof. Mark Horowitz, with a focus on optimizing energy and area efficiency of coarse-grained reconfigurable arrays.

Zach A. Myers (Member, IEEE) received the B.S. degree in electrical engineering from UC Davis, Davis, CA, USA, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, where he is currently pursuing the Ph.D. degree, with a focus on the topic of sequence detection for high-speed links.

Outside research, he has a lot of experience in board design for various projects ranging from microfluidics to IC chip testing.

Brandon D'Agostino (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering and computer engineering and the M.S. degree in electrical engineering from the University of Connecticut, Storrs, CT, USA, in 2020 and 2021, respectively. He is currently pursuing the Ph.D. degree in electrical engineering with Stanford University, Stanford, CA, USA.

His research interests include embedded systems, reconfigurable computing platforms, and formal hardware verification methods.

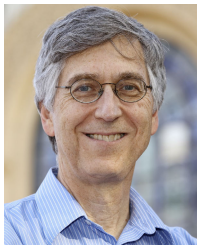
Pranil Joshi is currently pursuing the M.S. degree in electrical engineering with Stanford University, Stanford, CA, USA.

Stephen Richardson received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA.

He has worked in industry at Weitek, San Jose, CA, USA, and MIPS, San Jose, and at Sun Microsystems, Santa Clara, CA, USA, and Hewlett-Packard Research Labs, Palo Alto, CA, USA. He is currently a Research Associate with the Electrical Engineering Department, Stanford University.

Christopher Torng received the B.S., M.S., and Ph.D. degrees from Cornell University, Ithaca, NY, USA, in 2012, 2016, and 2019, respectively, all in electrical and computer engineering.

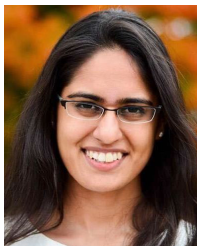
From 2019 to 2022, he was a Post-Doctoral Researcher with Stanford University, Stanford, CA, USA, operating in the leadership of the AHA Agile Hardware Project. Since 2023, he has been with the University of Southern California, Los Angeles, CA, USA, where he is currently an Assistant Professor of electrical and computer engineering. His research interests are in domain-specific hardware architectures and agile hardware design methodologies.



Mark Horowitz (Fellow, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 1978, and the Ph.D. degree from Stanford University, Stanford, CA, USA, in 1984.

He has worked on many processor designs, from early RISC chips to distributed shared memory multiprocessors, and in 1990, he took leave from Stanford University to help start Rambus Inc., San Jose, CA, USA, a company designing high-bandwidth memory interface technology. His work at both Rambus Inc. and Stanford University drove high-speed link designs for many decades. In the 2000s, he started a collaboration with Marc Levoy in computational photography, which led to light-field photography and microscopy. He is currently the Yahoo! Founders Professor at Stanford University, where he is the Chair of the Electrical Engineering Department. He remains interested in learning new things and building interdisciplinary teams. His current research interests include updating both analog and digital design methods, agile hardware design, and applying engineering to biology.

Dr. Horowitz is a fellow of the Association for Computing Machinery (ACM). He is a member of the National Academy of Engineering and the American Academy of Arts and Science.



Priyanka Raina (Life Fellow, IEEE) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology Delhi, New Delhi, India, in 2011, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2013 and 2018, respectively.

She was a Visiting Research Scientist with NVIDIA Corporation, Santa Clara, CA, USA, in 2018. She is currently an Assistant Professor of electrical engineering with Stanford University, Stanford, CA, USA, where she works on domain-specific hardware architectures and agile hardware–software codesign methodology.

Dr. Raina is a 2018 Terman Faculty Fellow. She was a co-recipient of the Best Demo Paper Award at VLSI 2022, the Best Student Paper Award at VLSI 2021, the IEEE JOURNAL OF SOLID-STATE CIRCUITS (JSSC) Best Paper Award in 2020, the Best Paper Award at MICRO 2019, and the Best Young Scientist Paper Award at European Conference on Solid-State Circuits (ESSCIRC) 2016. She has won the National Science Foundation (NSF) CAREER Award in 2023, the Intel Rising Star Faculty Award in 2021, and the Hellman Faculty Scholar Award in 2019. She was the Program Chair of the IEEE Hot Chips in 2020. She serves as an Associate Editor for the IEEE SOLID-STATE CIRCUITS LETTERS.