# Versa: A 36-Core Systolic Multiprocessor With Dynamically Reconfigurable Interconnect and Memory

Sung Kim, *Graduate Student Member, IEEE*, Morteza Fayazi, *Graduate Student Member, IEEE*, Alhad Daftardar, *Graduate Student Member, IEEE*, Kuan-Yu Chen, *Graduate Student Member, IEEE*, Jielun Tan, Subhankar Pal, *Member, IEEE*, Tutu Ajayi, *Graduate Student Member, IEEE*, Yan Xiong, *Graduate Student Member, IEEE*, Trevor Mudge, *Life Fellow, IEEE*, Chaitali Chakrabarti, *Fellow, IEEE*, David Blaauw, *Fellow, IEEE*, Ronald Dreslinski, *Senior Member, IEEE*, and Hun-Seok Kim, *Member, IEEE*

***Abstract*— We present Versa, an energy-efficient 36-core systolic multiprocessor with dynamically reconfigurable interconnects and memory. Versa leverages reconfigurable functional units and systolic-enhanced ARM cores to adapt for different algorithm characteristics, providing optimized bandwidth, access latency, and data reuse. Hardware support for crucial thread-synchronization operations enables a tree-based algorithm with 6.5× improvement in synchronization latency. Measured on a diverse set of compute kernels, Versa's design features culminate in median energy-efficiency improvements of 37.2× and 11.6× over mobile CPU and GPU baselines, respectively.**

***Index Terms*— Accelerators, data movement, data reuse, energy efficiency, interconnect, multicore architecture, on-chip memory, programmability, reconfiguration, systolic arrays.**

## I. Introduction

KERNELS with diverse computation and data transfer are ubiquitous in emerging applications but are challenging to support in general-purpose processors. For instance, sparsity techniques [1], [2] continue to gain adoption for machine learning but rely on complex, irregular data structures. Similarly, irregular algorithms that operate on index structures (such as trees and adjacency formats) are ubiquitous in graph analytics [3], [4] and genomics pipelines [5], [6]. Nevertheless, existing programmable designs lack first-class support for the types of workloads above and fail to exploit their properties.

Multi-core CPUs are the gold standard in programmability but incur significant overhead for features, such as speculative out-of-order (OoO) execution and hardware-managed cache coherence. These features are crucial for fully general software and single-thread workloads but have limited utility in acceleration contexts. On the other hand, GPUs rely entirely on SIMD compute that amortizes control-related overheads. SIMD excels on computations with predictable dense data, and GPU programmers benefit from an SIMT programming model that lends the illusion of scalar execution. However, due to thread divergence, the underlying SIMD units are prone to underutilization and degraded performance on irregular workloads [7]. Field-programmable gate arrays (FPGAs) are highly configurable but incur non-trivial hardware overheads in exchange for gate-level reconfigurability. In addition, FPGAs exhibit long reconfiguration times at microsecond scales [8].

This work describes Versa [9] (Fig. 1): a general-purpose accelerator that exploits microarchitectural flexibility to support diverse algorithms. In contrast to existing designs that incorporate fixed compute, interconnect, and on-chip memory, Versa provides optimized modes for each of the previous ones that are reconfigurable at nanosecond scales. This enables kernel implementations and hardware that are co-optimized for per-algorithm characteristics and dynamic application needs.

Versa incorporates the following specific contributions:

1) a tiled accelerator architecture that combines lightweight scalar cores with reconfiguration techniques to achieve energy-efficiency and performance improvements up to 105× and 71.6×, respectively;
2) reconfigurable functional units—namely, crossbar interconnects and on-chip memories—that exploit mode-specific properties for optimized bandwidth and latency;
3) enhancements to industry-grade ARM cores that enable general-purpose, programmable systolic computation;
4) hardware support for fundamental thread-synchronization operations that reduce synchronization latency by 6.5×.

Prior designs most similar to Versa include the raw processor [10], Manticore [11], and the ET-SoC-1 from Esperanto Technologies [12] (the latter two developed concurrently with this work). Raw incorporates packet-switched routers that are instruction-addressable for stream-based dataflow, similar to Versa's instruction-level systolic computation. However, Versa's systolic mechanism is comparatively lightweight (e.g., without packet routers), does not require per-packet

Manuscript received August 16, 2021; revised October 25, 2021; accepted December 23, 2021. Date of publication January 31, 2022; date of current version March 28, 2022. This article was approved by Associate Editor Borivoje Nikolić. This work was supported in part by the U.S. Government and in part by the Defense Advanced Research Projects Agency (DARPA) under Grant FA8650-18-2-7864. *(Corresponding author: Sung Kim.)*

Sung Kim, Morteza Fayazi, Alhad Daftardar, Kuan-Yu Chen, Subhankar Pal, Tutu Ajayi, Trevor Mudge, David Blaauw, Ronald Dreslinski, and Hun-Seok Kim are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: sungmk@umich.edu).

Jielun Tan is with Qualcomm, San Diego, CA 92121 USA.

Yan Xiong and Chaitali Chakrabarti are with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2022.3140241.

Digital Object Identifier 10.1109/JSSC.2022.3140241

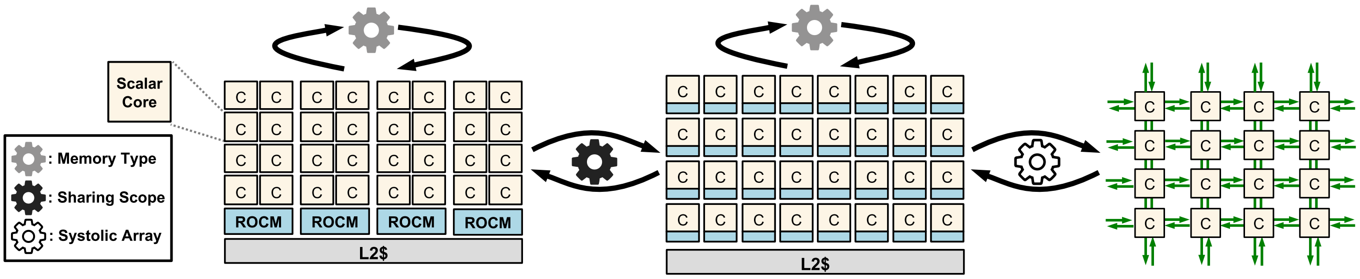


Fig. 1. Versa exploits reconfiguration of on-chip memory and compute to provide algorithm-optimized hardware characteristics.

initialization or link setup, and is physically constructed to mimic an application-specific integrated circuit (ASIC) design. Manticore incorporates hardware for DMA operations into core registers. This enables removal of explicit load/store instructions similar to Versa but still incurs DMA setup overhead and does not facilitate cross-thread spatial data reuse. ET-SoC-1 is a flexible accelerator targeted for machine learning that (like Versa) incorporates lightweight, general-purpose cores. To the best of our knowledge, ET-SoC-1 is the first industry design to incorporate memory that is reconfigurable as cache or scratchpad. We speculate that the ET-SoC-1's reconfigurable memory confers benefits similar to the Versa design; however, Esperanto has not disclosed details or performance metrics related to reconfigurability. A broader discussion of FPGA-based systems, reconfigurable ASICs, and additional manycore processors is provided in Section VI.

The remaining sections are organized as follows. Section II introduces the Versa architecture. Section III presents the reconfigurable functional units, systolic enhancements, and accelerated thread synchronization. Section IV details the prototype chip and experiment methodology, and Section V presents the measured results.

## II. Versa: A Reconfigurable Processor Design

This section discusses high-level design choices and introduces the Versa architecture.

### A. Key Objectives and Design Choices

An accelerator architecture that addresses the limitations from Section I should cover the following sub-criteria:

1) programmability using productive, high-level languages;
2) tolerance to irregular compute and data-access patterns (i.e., no thread divergence);
3) maximal support for contrasting (potentially unknown) workloads.

This work addresses the above with a combination of existing techniques and novel contributions:

1) ARM-based cores, coupled with a robust ecosystem of toolchain and compiler infrastructure;
2) multi-threaded execution backed by scalar core clusters, intrinsically tolerant to divergence;
3) reconfiguration and lightweight pipeline augmentations to support distinct, workload-optimizing hardware modes.

While not listed above, the premise of energy efficiency and performance dictates additional design traits. For instance, while the possible scopes and granularities of hardware reconfiguration are virtually unbounded, reconfiguration that is finer grained generally incurs greater overhead. This is the case with FPGA designs, as discussed in Section I. Thus, Versa incorporates reconfigurability that is limited to a few specific functional units, which are strategically positioned in the design hierarchy to maximize return on investment. These aspects are discussed in more detail next.

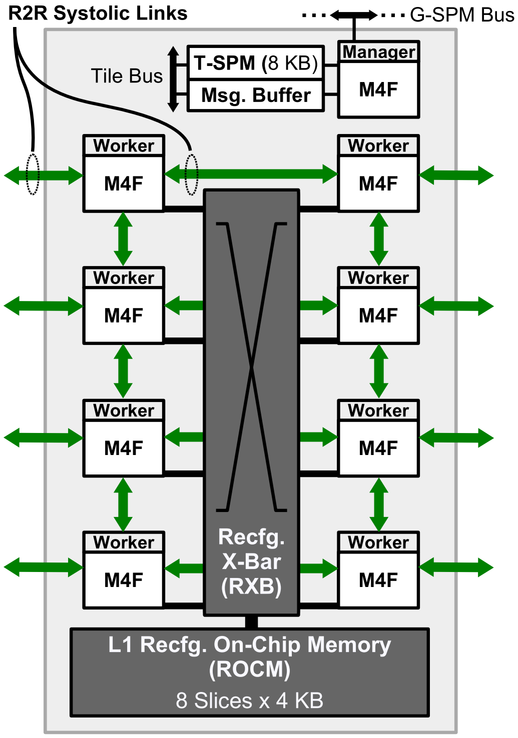


Fig. 2. Composition of a Versa compute tile.

### B. Compute Tiles

The majority of Versa's energy efficiency, performance uplift, and research novelty are attributed to features in its compute tiles (Fig. 2). A tile contains eight ARM Cortex-M4F "worker" cores that are responsible for the bulk of algorithmic computation. Workers are single-issue cores equipped with an IEEE 754-compliant single-precision (i.e., FP32) scalar floating-point unit (FPU). The inclusion of the FPU extends the base ARMv7-M ISA with DSP-oriented floating-point instructions and adds 32 additional operands (`s0–s31`) to the M4F register set. Bare-metal ARM binaries are loaded into a 16-kB instruction memory that resides in each M4F core. A tile also includes a Cortex-M4F "manager" core to handle supervisory tasks, including reconfiguration.

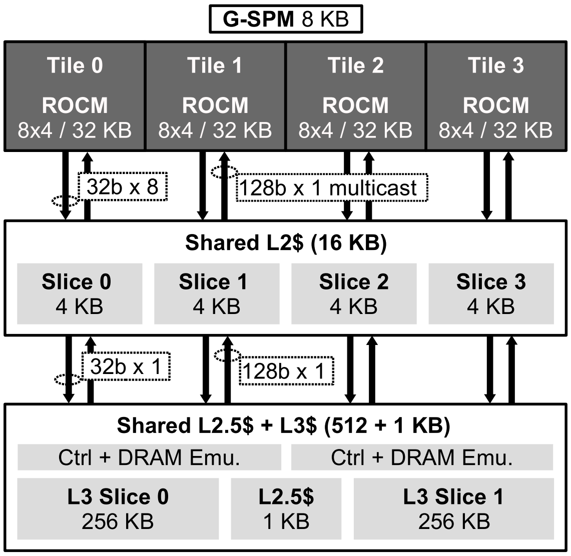


Fig. 3. Overview of the Versa prototype and memory hierarchy.

Versa workers have reduced functional coupling to the manager, unlike prior processors that explicitly partition management and computation. For instance, "synergistic processing elements" (SPEs) in the cell processor [13] could not directly access memory, instead of requiring manager-coordinated DMA transfer between memory and SPE scratchpads. In contrast, Versa workers can access system memory without manager intervention, significantly reducing software complexity.

Each tile contains reconfigurable resources to support multiple hardware modes—namely, the reconfigurable on-chip memory (ROCM), reconfigurable crossbar (RXB), and register-to-register (R2R) links. The ROCM and RXB reconfigure in a pair and compose to provide a multi-modal L1 memory with optimized characteristics; 32 kB of L1 memory is partitioned into eight slices (4 kB each), where the functionality of each slice is determined by RXB and ROCM submodes. Although it is possible to add reconfigurability in other levels of the memory hierarchy, early simulation experiments suggest diminishing returns. This is largely due to the intensity of data transfer at the L1 level, and the Cortex-M4F's relative sensitivity to load/store latency.[1] In addition to the ROCM and RXB, R2R links (illustrated by green arrows) augment the M4F processors to enable systolic computations in arbitrary 2-D spatial groupings. Further detail on the ROCM, RXB, and R2R-related enhancements are provided in Section III.

Large data structures that require cross-mode persistence are placed in a dedicated tile-level scratchpad memory (T-SPM). Notably, the T-SPM and a global-level scratchpad (G-SPM) external to the tiles facilitate accelerated thread-synchronization operations, which is discussed further in Section III-D. A small point-to-point message buffer between the manager and each worker facilitates the low-latency distribution of small runtime variables (e.g., parameters for load balancing).

TABLE I
COMPOSITION OF LOGICAL MEMORY MODES

| | | ROCM Sub-mode | | |
|---|---|---|---|---|
| | | Cache | Scratchpad | Queue |
| RXB Sub-mode | Private | ✓ | ✓ | - |
| | Shared | ✓ | ✓ | - |
| | Queue | - | - | ✓ |

### C. Memory Hierarchy and Tile-External Support

The four compute tiles in the Versa prototype are supported by two additional levels of cache (Fig. 3). Overall, L1–L3 have capacities that form an "hourglass" or inverted shape, typical for processors with high core counts. In addition, cache coherence and invalidation are explicitly software-managed; this mechanism significantly reduces hardware overhead and is also the mechanism used in current GPU products.

In terms of cache policies, Versa utilizes both read-allocate write-through and read-allocate write-back caches. This is due to relative advantages between cache policies that depend on the level of temporal locality. For instance, a write-through policy prevents the problem of "false sharing" [14], while write-back typically incurs less spurious traffic under high temporal locality.

The L2 is a 16-kB fixed-function shared cache (S.Cache; implemented with four 4-kB slices) that utilizes read-allocate write-through. Slices are statically cache line-interleaved.[2] In contrast to a small L2 write-back cache that would retain "victim" lines evicted from the L1 [15], the write-through L2 effectively serves as a staging area for data shared across tiles. For instance, while cache lines never experience write-back eviction to the L2, updates to cross-tile shared data still update valid cache lines in the L2, obviating the need for L3 access.

The last level of the cache hierarchy contains a 1-kB "L2.5" cache and 512-kB L3 cache. As discussed above, write-through stores necessarily propagate through the hierarchy. However, it is desirable to minimize the number of SRAM accesses in L3. Thus, in contrast to L2, L2.5 is a fully associative write-back cache that supports sub-block valid tracking [14]. Due to the small size of L2.5, only 256 bits are required for word-granularity sub-block valid state.

The last notable component outside the tiles is an 8-kB G-SPM that is accessible by managers. Similar to the T-SPM, the G-SPM facilitates the low-latency transfer of data related to control and supervisory tasks, in particular for the thread-synchronization optimizations discussed in Section III-D.

## III. Support for Algorithm-Driven Adaptation

Versa supports five composite modes (Table I) that can be dynamically configured at runtime, in addition to systolic R2R. The ROCM implements functionality that governs the memory type, namely, for ROCM-cache, ROCM-scratchpad, and ROCM-queue. The RXB implements functionality for the memory scope, namely, RXB-private, RXB-shared, and RXB-queue—a variation of the private scope that allows a pair of

[1] The M4F cores are augmented with a simple form of software-controlled prefetching to facilitate memory latency hiding. This prefetch mechanism leverages "store inversion," where cores may prefetch a cache line by storing its address to a special memory section. Effectively leveraging this mechanism involves compiler integration, which we leave for future work.

[2] The interleaving function is $s = \texttt{index} \bmod_4$, where $s$ is the slice number and $\texttt{index}$ is the cache line index bit-selected from an incoming address.

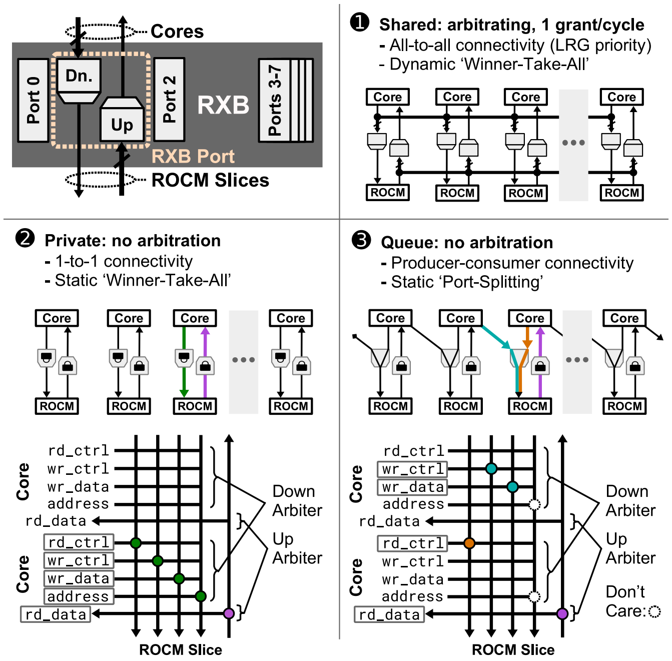

Cores
Port 0
Dn.
Up
Port 2
RXB
Ports 3-7
RXB Port
ROCM Slices
❶ Shared: arbitrating, 1 grant/cycle
- All-to-all connectivity (LRG priority)
- Dynamic 'Winner-Take-All'
Core
ROCM
❷ Private: no arbitration
- 1-to-1 connectivity
- Static 'Winner-Take-All'
❸ Queue: no arbitration
- Producer-consumer connectivity
- Static 'Port-Splitting'
rd_ctrl
wr_ctrl
wr_data
address
rd_data
Down Arbiter
Up Arbiter
Don't Care:
ROCM Slice

Fig. 4. RXB and overview of sub-modes. Diagrams in private and queue illustrate a sub-section of the crosspoint matrix.

cores to access the same slice simultaneously. The microarchitecture of the RXB and ROCM and functionality of their sub-modes is described in the following.

## A. Reconfigurable Crossbar

The RXB [Fig. 4 (top left)] contains eight bidirectional ports, equal to the number of worker core and ROCM slice pairs. Each port is implemented with a pair of arbiters, one per upstream and downstream direction. Arbiters operate on a per-direction basis—rather than per-signal basis—to amortize crossbar overhead (e.g., muxes and arbitration logic). The pair of arbiters is reconfigurable to one of three RXB sub-modes (Fig. 4): 1) RXB-shared; 2) RXB-private; or 3) RXB-queue (access from a pair of cores). These sub-modes function as follows.

1) *RXB-Shared:* The crossbar provides all-to-all "winner-take-all" connectivity between workers and ROCM slices, with least recently granted (LRG) arbitration [16].
2) *RXB-Private:* The arbiter crosspoints are statically fixed in both directions with "winner-take-all," locking connections vertically between worker and ROCM pairs.
3) *RXB-Queue:* Upstream arbiters use "winner-take-all" as in RXB-private, whereas downstream arbiters are statically "split" between a pair of cores.

In RXB-shared, the ROCM slices appear as a single shared memory that is accessible by all workers in the tile. This is often beneficial for workloads with data structures that are accessed (and reused) across multiple cores. In addition, the 8× increase in capacity provided by RXB-shared reduces spills and re-fetches from subsequent memory levels for larger data footprints. In RXB-private, the locking of crosspoints between worker and ROCM slices not only eliminates bank contention but also obviates arbitration. This results in up to 10.6× improvement in bandwidth and latency relative to shared mode (i.e., contention elimination + arbitration skipping), with a lower bound of 1.33× improvement due to arbitration skipping alone. RXB-queue supports common streaming DSP and filtering kernels. In contrast to the "winner-take-all" pattern used in RXB-private and RXB-shared, RXB-queue resolves structural contention by sub-partitioning ownership of signals inside an RXB port. This "port-splitting" enables simultaneous reader–writer access to ROCM slices and effectively doubles bandwidth over the same port. Notably, FIFO semantics in queue mode enables full reuse of existing crossbar signals, without signal duplication or widening of buses.

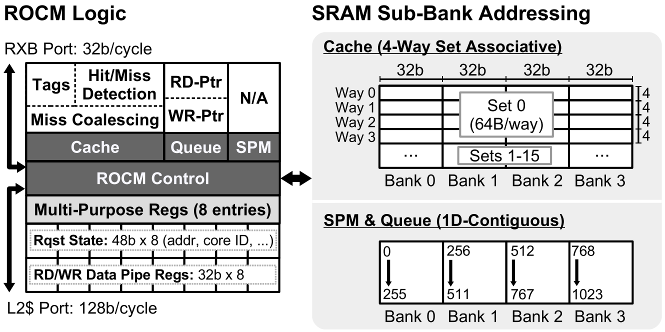


Fig. 5. Sub-mode logic and SRAM components in the ROCM.

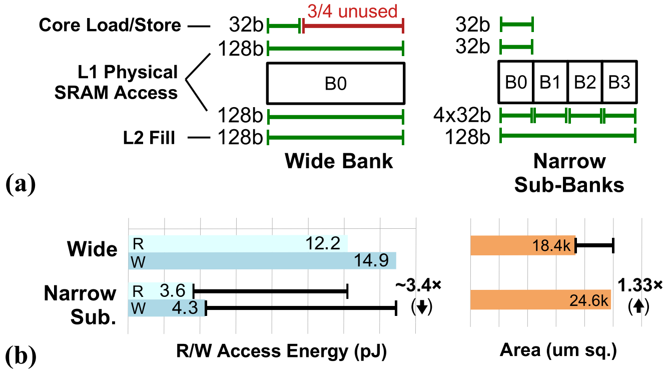


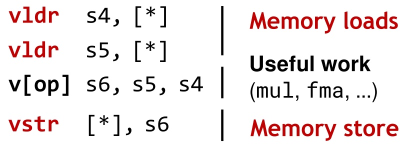
Fig. 6. Comparison of banking schemes (wide versus sub-banked): (a) data utilization per access and (b) design costs.

## B. Reconfigurable On-Chip Memory

Similar to the RXB, the L1-ROCM (Fig. 5) contains logic to support multiple sub-modes as follows.

1) *ROCM-Cache:* A four-way set-associative read-allocate write-through cache. Coherence is software-managed.
2) *ROCM-Scratchpad:* An explicitly managed scratchpad memory. Main-memory accesses bypass L1 and are forwarded directly to L2.
3) *ROCM-Queue:* An explicitly managed FIFO queue between core pairs. Main-memory accesses are forwarded to L2 (as with ROCM-scratchpad).

The cache sub-mode logic is primarily composed of cache tags, support for non-blocking cache requests (i.e., multiple outstanding requests), hit/miss detection, and coalescing logic. Coalescing refers to the case when an existing (in-flight) request with a cache line match is merged with an incoming



```
vldr   s4, [*]
vldr   s5, [*]
v[op]  s6, s5, s4
vstr   [*], s6
```

Fig. 7. Conventional instruction sequence: obligatatory memory access overheads are interleaved with computation.

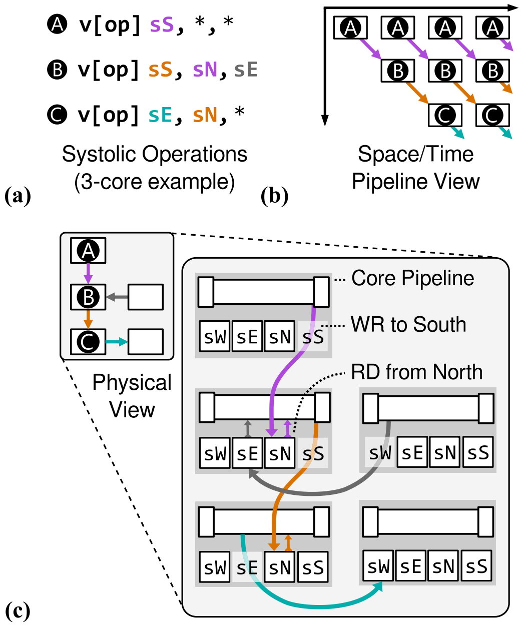


Fig. 8. Example computation with R2R: (a) instruction sequence, (b) logical (pipeline) view, and (c) physical view of read/write conventions. R2R-writes bypass the local register file (RF), while R2R-reads utilize local registers. Cardinal directions are inverted for writes to maintain spatial symmetry.

request such that dispatch of a new L2 request is unnecessary (saving both bandwidth and request slots).

Logic to support ROCM-scratchpad and ROCM-queue is minimal. Aside from interface logic to forward requests that have addresses pointing to main memory, an ROCM slice configured as scratchpad is functionally equivalent to a simple SRAM. With a 4-kB physical capacity (1024 32-bit words), additional state for FIFO operation in ROCM-queue consists of two 10-bit read/write pointers.

SRAM is reused across modes, with interface overhead limited to combinational logic that maps logical addresses to physical sub-banks [Fig. 5 (right)]. The choice of 32-bit sub-banks is driven by the native load/store width of the cores (Fig. 6). While a single 128-bit bank amortizes SRAM periphery more effectively, 75% of the physical SRAM read width would be unutilized for 32-bit scalar accesses [Fig. 6(a)]. Thus, sub-banking reduces common-case access energy by 3.4× in exchange for 33% more area [Fig. 6(b)].

### C. Systolic Execution

Systolic arrays have been recently applied in ASIC designs [17]–[20] but lack a general-purpose, programmable counterpart. While systolic arrays leverage registers for data movement, inter-core data transfer in load/store architectures necessitates cache or main-memory access. This reliance on explicit load/store instructions for data transfer has two impacts.

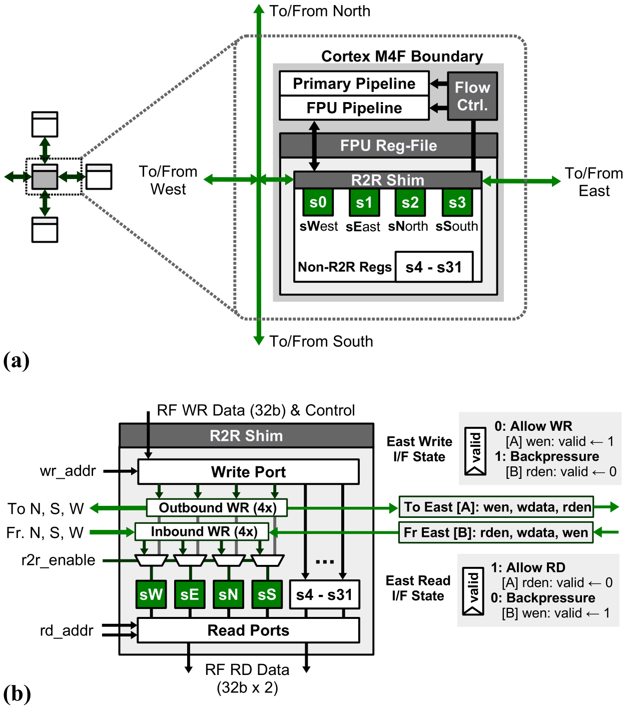


Fig. 9. R2R microarchitectural implementation: (a) integration with the ARM Cortex-M4F pipeline and (b) interception of operands in the R2R Shim and 2-bit link-state control.

1) *Performance Costs (Fig. 7):* Explicit load/store instructions consume pipeline cycles, stalling pipeline ALUs even if they are physically available for use.
2) *Energy Costs:* Load/store instructions entail underlying operations that are significantly more dissipative than register access, e.g., SRAM access and cache traversal.

Versa performs systolic array computation with R2R tunneling. R2R directly connects adjacent cores to enable programmable systolic computation and enhanced spatial data reuse. Compared to existing multi-threaded programming models, R2R confers the following benefits:

1) implicit data movement that increases utilization of pipeline ALUs, up to the physical limit;
2) inter-core data transfer that is R2R only, eliminating the energy cost of cache and SRAM access.

Fig. 8(a) shows an example R2R instruction sequence across three cores. In place of explicit loads and stores, data movement with R2R is implicitly controlled by the inclusion of spatial registers as source or destination operands. This enables fine-grained, multistep systolic computations [Fig. 8(b)] with energy and performance benefits similar to an ASIC equivalent. Fig. 8(c) shows the physical view of data movement between spatial registers. We note that the semantics of R2R requires a convention for the physical location of a shared spatial register; in Versa, the write destination always resides in the remote core. For example, "write to south" and "read from north" both physically refer to the reader's "north" register.

R2R integrates seamlessly in the Cortex-M4F pipeline with minimal overhead. If enabled at runtime (i.e., in software), the FPU registers s0–s3 are aliased to scalar data links in the <W, E, N, S> directions, respectively [Fig. 9(a)].

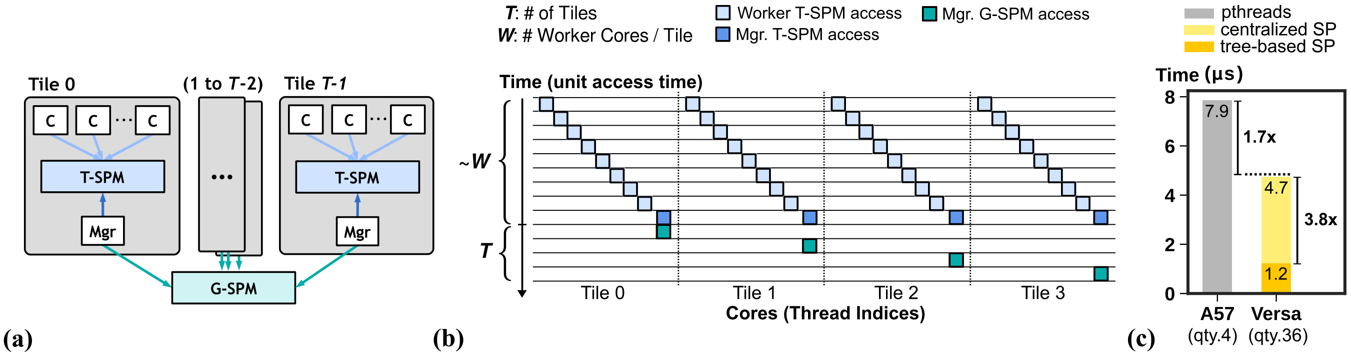


Fig. 10. Accelerated thread-synchronization barriers. (a) Distributed scratchpads decentralize atomic updates to barrier data. (b) Serialization is reduced by partitioning into parallel sections. (c) Comparison with alternative barrier implementations.

Link state (i.e., data valid tracking) requires 2 bits per bidirectional link. When an instruction writes to an R2R register, data from register write-back—normally directed to the local RF—are instead intercepted by the "R2R Shim" [Fig. 9(b)] and forwarded to an adjacent core. An R2R-write updates the link state and allows a matching R2R-read to proceed at the neighbor. In contrast, R2R-reads proceed if the link state is valid but utilize the local RF. Symmetry in read/write logic across cores minimizes timing impact and prevents the creation of new critical paths. Finally, flow control that prevents stale reads and destructive writes is implemented by tie-in with existing pipeline stall mechanisms. While this work augments the FPU RF to demonstrate the R2R concept with floating point workloads, the integer pipeline may be similarly extended (if desired) with no loss of generality.

### *D. Hierarchical Thread Primitives*

Thread-synchronization barriers are fundamental operations that are notorious performance bottlenecks in parallel programs [21], [22]. For instance, thread barriers may consume up to 60% of execution time in complex parallel workloads [23]. Because barriers constitute a non-trivial fraction of parallelization overhead, application developers go to significant lengths to both minimize their usage and develop low-level optimizations. Prior efforts include entirely new algorithm implementations [24], [25], in addition to a significant body of work on underlying synchronization-related primitives [26]. Nevertheless, thread barriers can be minimized but typically not eliminated entirely.

Thread synchronization entails serialized, atomic updates to shared variables that track whether threads can safely enter or exit a region of parallel execution. In conventional CPUs, each atomic update can require hundreds to thousands of cycles (i.e., microseconds), largely due to deep coherent caches that centralize barrier data. In addition, barrier operations have poor scalability in manycore designs and, in the worst case, exhibit $O(N)$ scaling with respect to core count.

Versa abolishes cache altogether and instead utilizes scratchpads distributed at the tile (T-SPM) and global (G-SPM) levels for lock and barrier operations. In addition to controlled latency and full support for Cortex-M exclusive access extensions, the T-SPM and G-SPM enable decentralized updates in a tree-based strategy [Fig. 10(a)]. The hierarchical (tree-based) strategy partitions atomic updates across tiles into sections that run in parallel [Fig. 10(b)], improving scalability from $O(WT)$ to $O(W+T)$, where $W$ and $T$ are the number of workers per tile and number of tiles, respectively. Although we focus on thread barriers for this work, the flexibility of the T/G-SPMs facilitates the full range of thread-synchronization primitives built on atomic memory access (e.g., semaphores and multithreaded data structures).

The tree-based scratchpad barrier is evaluated [Fig. 10(c)] against two baselines: a centralized scratchpad barrier that is measured with Versa in RTL simulation and an off-the-shelf barrier implementation from the popular `pthreads` library, measured on a quadcore ARM A57 CPU. The centralized scratchpad-based approach alone achieves a 1.7× speedup compared to the cache-based barrier from `pthreads` on the CPU. Adding the tree-based strategy yields an additional 3.8× speedup—or 6.5× total—despite a 9× higher thread count. Thus, Versa accelerates barrier operations such that thread synchronization contributes marginally to parallelization overheads.

### *E. Reconfiguration Control*

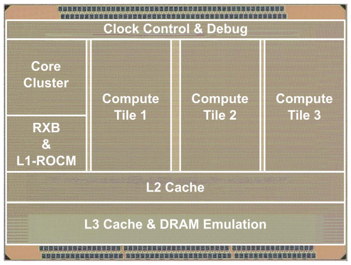
Typical reconfigurable systems (e.g., FPGAs) have combinatorially large configurations that necessitate large bitstreams and significant on-chip storage. In contrast, the finite set of Versa modes is governed by a single memory-mapped register (MMR) in each tile, accessible by the manager core. 4-bits control RXB and ROCM sub-modes (2 bits each). Thus, memory mode transitions involve a simple MMR write and complete in two cycles. Versa software libraries currently support (optional) memory reconfiguration by managers during thread synchronization, which guarantees that reconfigurable resources are not accessed during the two-cycle reconfiguration window. In contrast, the enable/disable for systolic R2R is tied to 1 bit in an MMR local to each worker core, and R2R registers are guarded from automatic compiler usage in regions of code where R2R is enabled (i.e., correctness for R2R is compiler-enforced).

## IV. Prototype Chip and Evaluation Methodology

The following describes the test chip and experiment design.

### *A. Chip Implementation*

Versa is fabricated in a 28-nm CMOS process and occupies 12-mm$^2$ die area (Fig. 11). The design is implemented hierarchically with tiles and L3 as hard partitions. L3 also includes logic for DRAM timing emulation, which is disabled for this



| | |
|---|---|
| Technology | 28 nm |
| Area (mm²) | Die: 12 (Ex. pads: 11.3) |
| Core Config | 36 Cores (9 Cores / Tile) |
| Core μ-arch | Arm Cortex-M4F, R2R |
| Total SRAM | 1.27 MB |
| Voltage | 0.55 - 1.0V (MEP @ 0.60V) |
| Frequency | 12 - 510 MHz |
| Power | 7.9 – 811.2 mW |
| Energy Eff. | 14.7 – 36.4 GFLOPS/W |
| Performance | 0.74 – 11.9 GFLOPS |

Fig. 11. Left: die photograph. Right: summary characteristics.

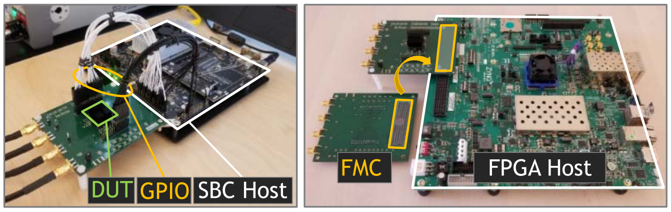


Fig. 12. Setups for chip testing. Left: Linux single-board-computer debug host, primarily for bringup and data collection. Right: FPGA interface for emulated MPSoC integration.

TABLE II
HARDWARE BASELINES

| | Cores | Clock | Architecture | Feature Note |
|---|---|---|---|---|
| ARM A57 | 4 | 1.8 GHz | ARMv8-A | 2-way superscalar, OoO |
| NVIDIA TX1 | 256 | 1.0 GHz | Maxwell Gen.2 | Eight 32-lane SIMD units |

work and left for future evaluation with Versa's prefetching features. A single unstructured clock tree is sufficient to meet the 2-ns $T_{clk}$ target, with 520-ps mean insertion delay and 70-ps mean global skew (3.5% of $T_{clk}$). At the nominal voltage (1.0 V), the system operates at 510 MHz, corresponding to 811.2 mW and 11.9-GFLOPS power and performance, respectively.

The test chip incorporates parallel boundary scan interfaces for test and debug, in addition to a VITA 57.1 FMC interface. Automated infrastructure for data collection is developed in python and C++ and runs on a Linux single-board computer [Fig. 12 (left)]. The FMC interface [Fig. 12 (right)] enables emulated integration with an FPGA MPSoC device, in particular with hard application-class host cores.

### *B. Baselines and Test Methodology*

Versa is evaluated against two energy-efficient hardware baselines. The baselines (Table II)—a quad-core ARM A57 CPU and 256-core Maxwell Gen.2 GPU (both integrated in NVIDIA's Tegra X1 SoC)—are mobile-class processors fabricated in a comparable 20-nm CMOS process. All measurements are performed with uncapped power states and averaged across the latter half of 1000 iterations with the first half discarded for cache warmup. GPU measurements utilize custom timers implemented in PTX assembly to obtain nanosecond-resolution timings free of host-side overheads. CPU measurements use nanosecond-resolution timers from `std::chrono`.

Power dissipation for the Versa chip is measured from a benchtop power supply, while the CPU and GPU measurements leverage on-board I2C-addressable current monitors on the Jetson TX1 platform. All major portions of the test chip (core logic, test and debug circuitry, and SRAM) are included in measurements and recorded as a lump number from a single supply. Digital I/O and a tuneable clock generator reside on separate voltage domains and are excluded but contribute marginally to power dissipation. The current monitors on the TX1 enable independent measurement of CPU and GPU power, but to the best of our knowledge do not compensate for dc–dc converter losses or DRAM power.[3]

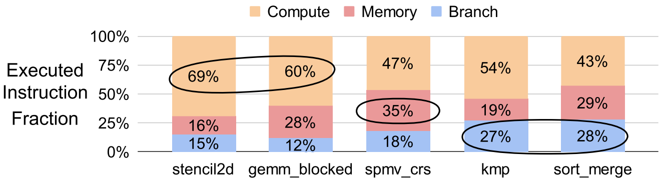


Fig. 13. Evaluation kernels from MachSuite: kernels are selected to capture a range of characteristics.

Five kernels from MachSuite [27] are selected based on the mix of instruction types, capturing representative diversity (Fig. 13). Stencil2D (2-D convolution) and GeMM (matrix mult.) exhibit regular data access to dense data, while KMP (string search) and SpMV (sparse matrix–vector mult.) have data-dependent variation in access patterns. Mergesort is a branch and synchronization-heavy comparison-based sort. We select 2 Versa modes per kernel based on analysis of the MachSuite reference kernels and modulate data sizes up to 512 kB. The CPU is evaluated with the reference kernels, while the GPU kernels use hand-optimized CUDA. CUDA kernels were developed with an effort-level of 2–4 weeks per kernel, guided by `nvprof` profiling and analysis with hardware performance counters. Best effort optimizations were utilized wherever possible, including memory coalescing, data tiling, scratchpad ("CUDA shared mem.") usage, divergence optimization, and sweeps of thread-block and grid sizes. Kernels for all platforms use FP32 as the primary workload datatype, which is predominant in ML research, sparse HPC, graph analytics, and genomics. We note that an evaluation with integer datatypes is largely redundant since benefits from memory reconfiguration are orthogonal, and the performance of the Cortex-M4F FPU pipeline is a lower bound relative to integer performance [28].

## V. MEASURED RESULTS AND ANALYSIS

Section V-A presents the measured results for MachSuite test kernels at nominal voltage, followed by voltage-scaling measurements.

We note that because hardware reconfiguration time is marginal, the measured results for kernel sequences (with reconfiguration interleaved) are virtually identical to the sum of runtimes for kernels executed in isolation. This property also holds for multi-modal kernels that are decomposed into distinct phases.

[3] The TX1 uses two Samsung 16-Gbit LPDDR4 chips (part no. K4F6E3S4HM) soldered on-board. According to manufacturer datasheets, the two DRAM chips dissipate between 150 and 400 mW. This would translate to less than 10% of either CPU or GPU power.

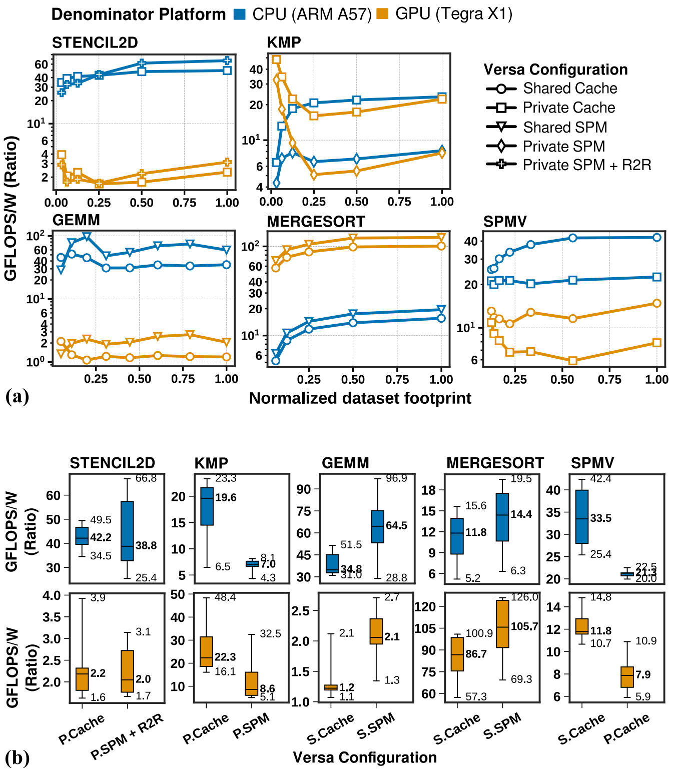


Fig. 14. Energy-efficiency improvement over CPU and GPU baselines: (a) trends across dataset footprints and (b) summary boxplots. Boxplots are labeled on min, median, and max values.

## A. Nominal Energy Efficiency and Performance

The benefit of Versa reconfiguration is illustrated by trends in energy-efficiency across dataset footprints[4] [Fig. 14(a)] and median overall improvements [Fig. 14(b)]. We find that the best mode varies not only across the test suite but also within individual kernels. Energy-efficiency improvements between Versa modes extend up to 3.17×, with 1.53× disparity on average (Table III). This is a key result that indicates the lack of any mode that could serve as a static replacement for reconfigurability. For instance, Stencil2D with Versa private cache (P.Cache) yields 1.37× higher GFLOPS/W relative to private SPM+R2R at small data sizes, but the advantage between modes is inverted at larger sizes. This result is due to the use of R2R to share and reuse overlapped input patches across cores and cache pressure as dataset footprint increases. Overall, Versa achieves 42.2×/2.2× median improvement over the CPU/GPU for Stencil2D.

For KMP, P.Cache consistently outperforms private SPM (P.SPM) despite $Q-1$ reuses of the query per search iteration, where $Q$ is the query length. Since scratchpads require explicit buffering, the number of loads/stores per iteration is nearly doubled relative to cache mode. Given a short query string (tests use $Q=4$), reuse is low and SPM buffering dominates. This results in 2.62× disparity between modes, with 19.6×/22.3× improvement over the CPU/GPU with P.SPM.

TABLE III

EFFICIENCY AND PERFORMANCE IMPROVEMENT FROM DYNAMIC RECONFIGURATION BETWEEN MODES. STARS INDICATE WHETHER THE MODE ADVANTAGE IS CONSISTENT (★) OR MIXED (☆)

| Kernel | | Mode A | | Mode B | Relative Gain |
|---|---|---|---|---|---|
| STENCIL2D | ☆ | P. Cache | ☆ | P. SPM+R2R | 1.26× |
| KMP | | P. SPM | ★ | P. Cache | 2.62× |
| GEMM | ☆ | S. Cache | ☆ | S. SPM | 1.73× |
| MERGESORT | | S. Cache | ★ | S. SPM | 1.22× |
| SPMV | | P. Cache | ★ | S. Cache | 1.57× |
| **All Kernels** | | | | | **1.53×** |

TABLE IV

MEDIAN IMPROVEMENTS ACROSS ALL KERNELS

| | GFLOPS/W | | GFLOPS | |
|---|---|---|---|---|
| Kernel | CPU | GPU | CPU | GPU |
| STENCIL2D | 42.2× | 2.25× | 6.92× | 0.16× |
| KMP | 19.6× | 22.3× | 3.18× | 1.51× |
| GEMM | 64.5× | 2.18× | 10.5× | 0.15× |
| MERGESORT | 14.4× | 105× | 2.33× | 71.6× |
| SPMV | 33.5× | 11.8× | 5.42× | 1.80× |
| **All Kernels** | **37.2×** | **11.6×** | **5.86×** | **0.78×** |

GeMM utilizes data tiling in S.Cache and shared scratchpad modes (S.SPMs) and has a reuse/buffering tradeoff similar to KMP. However, the benefit of quadratic data reuses outweighs scratchpad buffering cost, resulting in 1.73× median mode advantage for S.SPM and 64.5×/2.1× improvement over the CPU/GPU.

On Mergesort, Versa attains 2.33× and 71.6× speedups over the CPU and GPU, respectively, translating to 14.4× and 105× energy-efficiency improvements. GPU profiling indicates bottlenecks in parallel synchronization and branch-heavy comparison operations. Results from Mergesort suggest that Versa's independent scalar cores and tree-based scratchpad barriers are highly effective in practice.

For SpMV, the Versa kernels allocate one or more sparse dot product operations per worker core, implying that accesses between cores are non-overlapping. However, since MachSuite provides matrices in the CSR format, sparse values are packed contiguously in memory such that multiple sparse rows frequently reside in the same cache lines. This produces implicit cache prefetching of sparse matrix values and column pointers into the S.Cache. Synergistic prefetching does not occur in P.Cache, resulting in a mode advantage for S.Cache up to 1.9× and 33.5×/11.8× improvement against the CPU/GPU.

Across all kernels, energy-efficiency improvements extend up to 64.5×/105× against the CPU/GPU, with median improvements of 37.2× and 11.6× overall [Table IV (left)].

In terms of performance, Versa consistently outperforms the CPU, but results are mixed against the GPU [Table IV (right)]. The 5.86× median improvement in performance is obtained relative to the CPU; we estimate that hardware parallelism accounts for roughly half of the disparity, with the other half attributed to reconfiguration benefits. GPU performance ratios range from 0.15× to 71.6× on GeMM and MergeSort, respectively. While GPUs are known to excel on linear-algebra computations (which are prevalent in graphics applications), the difference in core counts is a plausible explanation for

[4] Dataset footprint is the size in bytes for a kernel's input and output arguments.

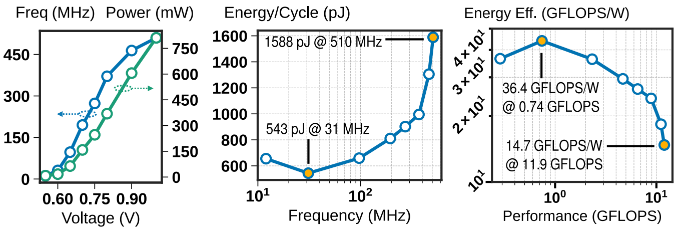


Fig. 15. Chip characteristics under voltage scaling. Left: system frequency and power. Middle: energy per cycle. Right: performance–efficiency Pareto curve.

the performance shortfall. For instance, multiplying Versa's performance by a factor of 8 to normalize core count results in performance improvements of 1.28× and 1.2× for Stencil2D and GeMM, respectively. However, it is unlikely that Versa's power dissipation would scale linearly, in addition to production-related overheads and design margins that the Versa prototype lacks. Nevertheless, these observations are promising and suggest that Versa is comparable on dense kernels and within the margin of error created by differences in implementation methodology.

### *B. Voltage-Scaled Characteristics*

We conclude the measurement results with chip characteristics under voltage scaling, examined on a non-terminating 11-tap FIR filter. This gives a reasonable measure of "peak-practical" performance and efficiency since the FIR kernel is compute-oriented but retains a realistic number of memory accesses. At the far ends of the 0.55–1.0-V operating range, the chip dissipates 7.9 mW (at 31 MHz) and 811 mW (at 510 MHz) [Fig. 15 (left)]. Energy per cycle [Fig. 15 (middle)] varies from 543 to 1588 pJ/cycle. The minimum energy point (MEP) is observed at 0.6 V, resulting in 2.47× improvement over the nominal voltage. Thus, Pareto-optimal operation [Fig 15 (right)] corresponds to 11.9-GFLOPS performance at 1.0 V or 36.4-GFLOPS/W energy efficiency at 0.6 V.

## VI. Related Work

In addition to the designs discussed in Section I, this work can be compared to recent FPGA systems, reconfigurable ASICs, and general-purpose processors with programmable interconnect. We also summarize relevant work on application phase-based dynamic reconfiguration and optimization.

Whatmough *et al.* [29] and Schiavone *et al.* [30] integrated embedded FPGAs and fixed-function accelerators in heterogeneous SoCs; kernels are accelerated by offloading to different subsystems. In contrast, Versa is a standalone accelerator architecture that reconfigures to support different algorithm needs.

The ASICs from [31]–[33] leverage programmable interconnect or packet routers in targeted applications. For instance, the systolic array in [31] uses reconfigurable routers to support both dense and sparse linear-algebra operations, the ASIC from [33] reconfigures between phases of a single sparse matrix-multiplication algorithm, and Smets *et al.* [32] used programmable routers to support multiple image processing kernels. The ASICs above employ reconfigurability to broaden the capabilities of a fixed-function design. In contrast, Versa is the first to demonstrate how reconfigurability—in a general-purpose processor—can enhance programmability and performance.

The designs presented in [10], [11], and [34]–[36] are fabricated multi-core processors capable of spatial data transfer that resembles systolic array dataflow. Versa also leverages programmable interconnect, but for the purpose of reconfiguration in conjunction with ROCM memory modes. Furthermore, systolic dataflow in Versa is not emulated but instead employs direct R2R links, with performance and energy–efficiency characteristics closer to those of an ASIC design.

Versa can be compared to recent coarse-grained reconfigurable array (CGRA) designs with dataflow execution [37]–[42]. Dataflow machines [41] depart from von Neumann (program counter-based) execution and instead traverse dataflow graphs explicitly. Data values and associated data tags flow through distributed on-chip memories and compute resources according to graph dependencies ("firing rules"). In comparison, Versa retains PC-based execution and reconfigures at higher levels of abstraction but follows dataflow-like firing rules for systolic R2R. In this sense, Versa is closer to a "hybrid dataflow machine" or "pseudo-systolic processor" [43] that incorporates von Neumann systolic processing elements.

Phase-based optimization [44] for adaptive hardware is a well-established research area, with prior work that spans the hardware stack [45]–[51]. Effler *et al.* [45] and Maas *et al.* [46] examined program features and learning-based approaches to optimize main-memory allocation in real server workloads. Huang *et al.* [47] optimized the last-level cache capacity in an off-the-shelf Xeon processor using L1 and L2 cache performance counters to build a phase-based heuristic controller. The works [48]–[51] are architectural (cycle-level simulation) studies that optimize on-chip memory types, cache sizes, and pipeline resources. For instance, Pal *et al.* [49] and Feng *et al.* [51] leveraged explicit workload phases and varying data sparsity to reconfigure on-chip resources according to power–performance tradeoffs. Similarly, Dubach *et al.* [48] and Pal *et al.* [50] adapted microarchitectural parameters but used learning-based models to detect and exploit implicit workload phases. While we leave an in-depth study of phase-based optimizations on Versa for future work, our hardware is designed to be compatible with both explicit and implicit phase-based methods, as described above.

## VII. Conclusion

This article introduces Versa, a flexible multi-core accelerator that optimizes for diversity in computation and data-access patterns. The premise is that given dynamic compute and data-access characteristics, any static hardware design will exhibit suboptimal energy efficiency and performance. Versa addresses this issue through fast, nanosecond-scale reconfiguration between distinct hardware modes. Reconfigurable functional units exploit mode-dependent operating guarantees (such as privatized data) to optimize microarchitectural characteristics. In addition, Versa's scalar cores are augmented to support a new class of programmable, R2R systolic array computation. To minimize and prevent performance bottlenecks on synchronization-heavy workloads, Versa employs distributed scratchpads that facilitate a tree-based

TABLE V
SIMT/SPMD-STYLE THREAD IDENTIFICATION

| **Function Signature** | **Description** |
|---|---|
| `uint core_id()` | Physical core ID [0:8] |
| `uint tile_id()` | Physical tile ID [0:3] |
| `uint core_row_id()` | Physical row ID [0:3] |
| `uint core_col_id()` | Physical col. ID [0:1] |
| `bool core_on_edge(enum <L\|R\|T\|B>)` | Chip boundary |
| `bool is_manager()` | Manager identification |

TABLE VI
THREAD-SYNCHRONIZATION PRIMITIVES

| **Function Signature** | **Description** |
|---|---|
| `void sync_managers()` | Managers only |
| `void sync_workers()` | Workers in tile |
| `void sync_tile()` | All cores in tile |
| `void sync_global()` | All cores in chip |
| `void mutex_init(lock_t* lock)` | Low-level mutex |
| `void mutex_lock(lock_t* lock)` | Acquire |
| `void mutex_unlock(lock_t* lock)` | Release |

TABLE VII
MEMORY BARRIERS AND INVALIDATION CONTROL

| **Function Signature** | **Description** |
|---|---|
| `void mem_barrier_tile()` | Tile-level, full memory barrier |
| `void mem_barrier_global()` | Global, full memory barrier |
| `void mem_inv_tile()` | Tile-level, invalidation only |
| `void mem_inv_global()` | Global, invalidation only |

TABLE VIII
HEAP-STYLE MEMORY ALLOCATION

| **Function Signature** | **Description** |
|---|---|
| `void* malloc_sp(`<br>`size_t bytes, enum <tsp\|gsp>)` | Scoped SP malloc |
| `void free_sp(enum <tsp\|gsp>)` | Scoped SP free |
| `void* malloc_l0(size_t bytes)` | IMEM malloc |
| `void free_l0()` | IMEM free |

thread-synchronization algorithm. The techniques above are demonstrated in a 28-nm prototype chip that incorporates industry-grade IP cores and system components. Against comparable CPU and GPU baselines, the prototype achieves 37.2× and 11.6× median improvements in energy efficiency, respectively, on a set of diverse kernels. Overall, this work indicates that compute and memory reconfiguration are highly effective and may be broadly applicable to future accelerator designs.

## APPENDIX A
## PROGRAMMER-VISIBLE APIs

The following lists a subset of frequently utilized Versa C++ library functions and preprocessor macros. Trivial preprocessor macros defining chip parameters (e.g., `CORES_PER_TILE`) are omitted. See Tables V–XI.

TABLE IX
WORKER–MANAGER MESSAGE BUFFER

| **Function Signature** | **Description** |
|---|---|
| `msg_t msg_pop()` | Pop buffer element |
| `void msg_push(msg_t v)` | Push buffer element |
| `void msg_bcast_send(`<br>`msg_t* a, uint n_items)` | Broadcast send (manager) |
| `void msg_bcast_recv(`<br>`msg_t* a, uint n_items)` | Broadcast recv. (workers) |
| `bool msg_buf_empty(enum <i\|o>)` | I-buf/O-buf status |
| `bool msg_buf_full(enum <i\|o>)` | I-buf/O-buf status |

TABLE X
RECONFIGURATION CONTROL

| **Function Signature** | **Description** |
|---|---|
| `void l1_config(`<br>`enum <sp\|ca\|q>, enum <priv\|shrd>)` | L1 control |
| `void r2r_config(bool enable)` | R2R control |

TABLE XI
R2R INTRINSICS (PREPROC. DIRECTIVES)

| **Macro Signature** | **Description** |
|---|---|
| `R2R_ARITH(<asm_op>,` | Systolic compute; instruction name |
| `<var\|dst_dir>,` | Dest. R2R reg. or variable |
| `<var\|src_dir>,` | Src. R2R reg. or variable |
| `<var\|src_dir>)` | Src. R2R reg. or variable |
| `R2R_POP(<var>, <src_dir>)` | Systolic pop (`mov` instruction) |
| `R2R_PUSH(<dst_dir>, <var>)` | Systolic push (`mov` instruction) |
| `R2R_LD(<dst_dir>, <addr>)` | Systolic load |
| `R2R_ST(<addr>, <src_dir>)` | Systolic store |

## ACKNOWLEDGMENT

The authors would like to thank Arm Ltd. for IP and design support and the reviewers for thoughtful improvements to the manuscript. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.

## REFERENCES

[1] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. ICLR*, 2015, pp. 1–14.

[2] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *Proc. ICLR*, 2017, pp. 1–13.

[3] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," Stanford InfoLab, Stanford, CA, USA, Tech. Rep. 1999-66, 1999. [Online]. Available: http://ilpubs.stanford.edu:8090/cgi/export/422/BibTeX/ilprints-eprint-422.bib

[4] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proc. IEEE 12th Int. Conf. Data Mining*, Dec. 2012, pp. 181–213.

[5] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature Methods*, vol. 9, no. 4, pp. 357–359, 2012.

[6] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Briefings Bioinf.*, vol. 11, no. 5, pp. 473–483, Sep. 2010.

[7] P. Xiang, Y. Yang, and H. Zhou, "Warp-level divergence in GPUs: Characterization, impact, and mitigation," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2014, pp. 284–295.

[8] D. Koch, *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications*. Cham, Switzerland: Springer, 2012.

[9] S. Kim *et al.*, "Versa: A dataflow-centric multiprocessor with 36 systolic ARM cortex-M4F cores and a reconfigurable crossbar-memory hierarchy in 28nm," in *Proc. Symp. VLSI Circuits*, Jun. 2021, pp. 1–2.

[10] M. Taylor *et al.*, "The raw processor: A composeable 32-bit fabric for embedded and general purpose computing," in *Proc. HotChips*, 2001, pp. 1–4.

[11] F. Zaruba, F. Schuiki, and L. Benini, "Manticore: A 4096-core RISC-V chiplet architecture for ultraefficient floating-point computing," *IEEE Micro*, vol. 41, no. 2, pp. 36–42, Mar. 2021.

[12] D. Ditzel *et al.*, "Accelerating ML recommendation with over a thousand RISC-V/tensor processors on Esperanto's ET-SoC-1 chip," in *Proc. IEEE Hot Chips 33 Symp. (HCS)*, Aug. 2021, pp. 1–23.

[13] J. Kahle, "The cell processor architecture," in *Proc. 38th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Nov. 2005, p. 3.

[14] I. Singh, A. Shriraman, W. W. L. Fung, M. O'Connor, and T. M. Aamodt, "Cache coherence for GPU architectures," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2013, pp. 578–590.

[15] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *Proc. 17th Annu. Int. Symp. Comput. Archit.*, May 1990, pp. 364–373.

[16] S. Satpathy *et al.*, "A 4.5Tb/s 3.4Tb/s/W 64×64 switch fabric with self-updating least-recently-granted priority and quality-of-service arbitration in 45nm CMOS," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2012, pp. 478–480.

[17] 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.

[18] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ISCA*, 2017, pp. 1–12.

[19] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerging Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[20] H. Genc *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 1–6.

[21] M. Roth, M. J. Best, C. Mustard, and A. Fedorova, "Deconstructing the overhead in parallel applications," in *Proc. IISWC*, 2012, pp. 59–68.

[22] R. M. Yoo, C. J. Hughes, K. Lai, and R. Rajwar, "Performance evaluation of Intel®transactional synchronization extensions for high-performance computing," in *Proc. SC*, 2013, pp. 1–11.

[23] S. Xiao and W.-C. Feng, "Inter-block GPU communication via fast barrier synchronization," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. (IPDPS)*, Apr. 2010, pp. 1–12.

[24] H. Yun, H.-F. Yu, C.-J. Hsieh, S. V. N. Vishwanathan, and I. Dhillon, "NOMAD: Non-locking, stOchastic multi-machine algorithm for asynchronous and decentralized matrix completion," 2013, *arXiv:1312.0193*.

[25] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Proc. NeurIPS*, 2015, pp. 2737–2745.

[26] M. L. Scott, "Shared-memory synchronization," in *Synthesis Lectures on Computer Architecture*, vol. 8, no. 2. San Rafael, CA, USA; Morgan & Claypool Publishers, 2013.

[27] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2014, pp. 110–119.

[28] *Arm Cortex-M4 Processor Technical Reference Manual*, Arm Ltd., Cambridge, U.K., 2020.

[29] P. N. Whatmough *et al.*, "A 16nm 25 mm2 SoC with a 54.5x flexibility-efficiency range from dual-core arm cortex-A53 to eFPGA and cache-coherent accelerators," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C34–C35.

[30] P. D. Schiavone *et al.*, "Arnold: An eFPGA-augmented RISC-V SoC for flexible and low-power IoT end nodes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 4, pp. 677–690, Apr. 2021.

[31] M. Anders *et al.*, "2.9TOPS/W reconfigurable dense/sparse matrix-multiply accelerator with unified INT8/INTI6/FP16 datapath in 14NM tri-gate CMOS," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 39–40.

[32] S. Smets, T. Goedeme, A. Mittal, and M. Verhelst, "2.2 A 978GOPS/W flexible streaming processor for real-time image processing applications in 22nm FDSOI," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2019, pp. 44–46.

[33] D.-H. Park *et al.*, "A 7.3 M output non-Zeros/J, 11.7 M output non-Zeros/GB reconfigurable sparse matrix-matrix multiplication accelerator," *IEEE J. Solid-State Circuits*, vol. 55, no. 4, pp. 933–944, Apr. 2020.

[34] A. M. Jones and M. Butts, "TeraOPS hardware: A new massively-parallel MIMD computing fabric IC," in *Proc. IEEE Hot Chips 18 Symp. (HCS)*, Aug. 2006, pp. 1–15.

[35] B. Bohnenstiehl *et al.*, "KiloCore: A 32-nm 1000-processor computational array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 891–902, Apr. 2017.

[36] J. P. Cerqueira, T. J. Repetti, Y. Pu, S. Priyadarshi, M. A. Kim, and M. Seok, "Catena: A near-threshold, sub-0.4-mW, 16-core programmable spatial array accelerator for the ultralow-power mobile and embedded Internet of Things," *IEEE J. Solid-State Circuits*, vol. 55, no. 8, pp. 2270–2284, Aug. 2020.

[37] K. Sankaralingam *et al.*, "Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture," in *Proc. 30th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2003, pp. 422–433.

[38] K. Sankaralingam *et al.*, "Distributed microarchitectural protocols in the TRIPS prototype processor," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2006, pp. 480–491.

[39] S. Swanson *et al.*, "The WaveScalar architecture," *ACM Trans. Comput. Syst.*, vol. 25, no. 2, pp. 1–54, May 2007.

[40] 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.

[41] F. Yazdanpanah, C. Alvarez-Martinez, D. Jimenez-Gonzalez, and Y. Etsion, "Hybrid dataflow/von-Neumann architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1489–1509, Jun. 2014.

[42] T. Nowatzki, V. Gangadhar, N. Ardalani, and K. Sankaralingam, "Stream-dataflow acceleration," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 416–429, Sep. 2017.

[43] K. T. Johnson, A. R. Hurson, and B. Shirazi, "General-purpose systolic arrays," *IEEE Comput.*, vol. 26, no. 11, pp. 20–31, Nov. 1993.

[44] K. Criswell and T. Adegbija, "A survey of phase classification techniques for characterizing variable application behavior," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 1, pp. 224–236, Jan. 2020.

[45] T. C. Effler *et al.*, "Evaluating the effectiveness of program data features for guiding memory management," in *Proc. Int. Symp. Memory Syst.*, Sep. 2019, pp. 383–395.

[46] M. Maas, D. G. Andersen, M. Isard, M. M. Javanmard, K. S. McKinley, and C. Raffel, "Learning-based memory allocation for C++ server workloads," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Mar. 2020, pp. 541–556.

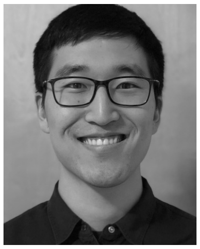
[47] Z. Huang, J. A. Joao, A. Rico, A. D. Hilton, and B. C. Lee, "DynaSprint: Microarchitectural sprints with dynamic utility and thermal management," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 426–439.

[48] C. Dubach, T. M. Jones, and E. V. Bonilla, "Dynamic microarchitectural adaptation using machine learning," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 4, pp. 1–28, 2013.

[49] S. Pal *et al.*, "Transmuter: Bridging the efficiency gap using memory and dataflow reconfiguration," in *Proc. ACM Int. Conf. Parallel Archit. Compilation Techn.*, Sep. 2020, pp. 175–190.

[50] S. Pal, A. Amarnath, S. Feng, M. O'Boyle, R. Dreslinski, and C. Dubach, "SparseAdapt: Runtime control for sparse linear algebra on a reconfigurable accelerator," in *Proc. MICRO 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2021, pp. 1005–1021.

[51] S. Feng *et al.*, "CoSPARSE: A software and hardware reconfigurable SpMV framework for graph analytics," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 949–954.



**Sung Kim** (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the University of Washington, Seattle, WA, USA, in 2015 and 2018, respectively. He is currently pursuing the Ph.D. degree in electrical engineering and computer science with the University of Michigan, Ann Arbor, MI, USA.

His research interests include computer architecture, hardware–software co-design, and accelerator programmability.

**Morteza Fayazi** (Graduate Student Member, IEEE) received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2017, and the M.S. degree in electrical engineering and computer science from the University of Michigan, Ann Arbor, MI, USA, in 2020, where he is currently pursuing the Ph.D. degree.

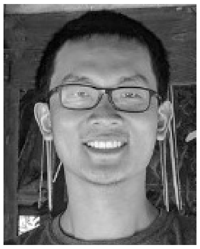
His research interests include EDA, machine learning, software development, and computer architecture.

**Alhad Daftardar** (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2020. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Michigan, Ann Arbor, MI, USA.

His research interests are in energy-efficient circuits and architectures for machine learning, signal processing, and communications.

**Kuan-Yu Chen** (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2018. He is currently pursuing the Ph.D. degree with the University of Michigan, Ann Arbor, MI, USA.

His current research interests include digital circuit design, accelerators, and computer architecture.

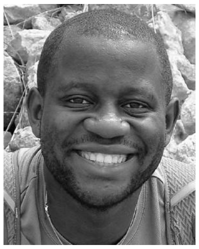


**Jielun Tan** received the B.S.E. and M.S. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2017 and 2019, respectively.

He is currently a hardware engineer at Qualcomm, San Diego, CA, USA. His research interests include hardware development for RISC-V, massively parallel systems, and emerging technologies.

**Subhankar Pal** (Member, IEEE) received the B.E. degree in electrical and electronics engineering from the Birla Institute of Technology and Science at Pilani (BITS-Pilani), Pilani, India, in 2014, and the M.S. and Ph.D. degrees from the University of Michigan, Ann Arbor, MI, USA, in 2018 and 2021, respectively.

He was previously with NVIDIA, Bengaluru, India, where he worked on pre-silicon verification and bring-up on multiple generations of GPUs. He currently works at IBM Research, Yorktown Heights, NY, USA.



**Tutu Ajayi** (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from Texas A&M University, College Station, TX, USA, in 2008, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 2017 and 2020, respectively.

He is currently a Post-Doctoral Research Fellow with the University of Michigan. His research interests include computer architecture, field-programmable gate array (FPGA) platforms, and VLSI EDA.

**Yan Xiong** (Graduate Student Member, IEEE) received the M.S. degree in optoelectronic information science and engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2017, and the M.S. degree in electrical engineering from Arizona State University, Tempe, AZ, USA, in 2018, where he is currently pursuing the Ph.D. degree in electrical engineering.

His research interests span the areas of speech processing, low power systems, and energy-efficient deep learning algorithms.

**Trevor Mudge** (Life Fellow, IEEE) received the Ph.D. in computer science from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1977.

He is currently the Bredt Family Professor of computer science and engineering with the University of Michigan, Ann Arbor, MI, USA. He has authored numerous articles on computer architecture, programming languages, VLSI, and computer vision. He has chaired more than 56 related theses.

Dr. Mudge received the ACM/IEEE CS Eckert-Mauchly Award in 2014 and the UIUC Distinguished Alumni Award.

**Chaitali Chakrabarti** (Fellow, IEEE) received the B.Tech. degree in electronics and electrical communication engineering from IIT Kharagpur, Kharagpur, India, in 1984, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland at College Park, College Park, MD, USA, in 1986 and 1990, respectively.

She is currently a Professor with the School of Electrical Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA. Her interests include VLSI signal processing and communication systems, low-power embedded systems, and portable medical imaging.

**David Blaauw** (Fellow, IEEE) received the B.S. degree in physics and computer science from Duke University, Durham, NC, USA, in 1986, and the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1991.

Until 2001, he worked at Motorola Inc., Austin, TX, USA, where he was the Manager of the High Performance Design Technology Group. Since 2001, he has been on the faculty of the University of Michigan, Ann Arbor, MI, USA, where he is currently the Kensall D. Wise Collegiate Professor of electrical engineering and computer science. He is also the Director of the Michigan Integrated Circuits Lab (MICL). His previous research interests include ultralow-power wireless sensors and low-power analog circuit techniques for millimeter systems. His group also introduced so-called near-threshold computing, which has since become a common concept in semiconductor design. Most recently, he has pursued research in cognitive computing using analog in-memory neural networks and genomics for precision health. He has published over 600 articles and holds 65 patents.

Dr. Blaauw was a member of the IEEE International Solid-State Circuits Conference (ISSCC) Analog Program Subcommittee and the General Chair of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED). He was awarded MIT Technology Review's "one of the year's most significant innovations." He received numerous best paper awards, the Motorola Innovation Award, and the 2016 SIA-SRC Faculty Award for "lifetime research contributions" to the U.S. semiconductor industry.

**Ronald Dreslinski** (Senior Member, IEEE) received the B.S.E., M.S.E., and Ph.D. degrees from the University of Michigan, Ann Arbor, MI, USA, in 2001, 2003, and 2011, respectively.

He is currently an Associate Professor of computer science and engineering at the University of Michigan. His work focuses on hardware and circuit designs for a post-Moore's Law world.

Dr. Dreslinski received the 2015 IEEE Young Computer Architect Award.

**Hun-Seok Kim** (Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2001, and the Ph.D. degree in electrical engineering from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 2010.

He is currently an Assistant Professor with the University of Michigan, Ann Arbor, MI, USA. His research focuses on system analysis, novel algorithms, and VLSI architectures for low-power/high-performance wireless communications, signal processing, computer vision, and machine learning systems.

Dr. Kim was a recipient of the DARPA Young Faculty Award in 2018 and the NSF CAREER Award in 2019. He is also an Associate Editor of IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, and the IEEE SOLID-STATE CIRCUITS LETTERS.