Many modern applications perform large-scale data processing that generates a high amount of data movement between the main memory and the CPU in a conventional computer, consuming significant energy [1]. Recent works have proposed new device innovations based on the principle of processing-using-memory (PUM) [3] to mitigate data movement overheads. The PUM paradigm takes advantage of direct electrical interactions between interconnected memory cells to perform primitive computational functions, in addition to the cells’ original role as data storage. As DRAM scaling issues continue to be difficult to solve [6, 7], researchers have been developing several emerging memory alternatives, including resistive memory technologies such as MRAM, PCM, and ReRAM. Resistive memories are attractive alternatives to DRAM as PUM-enabling technologies because of their ability to perform logically-complete bitwise Boolean operations (e.g., ReRAM-based NOR [5]) without relying on additional compute circuitry and because of their higher memory densities. Prior works have taken advantage of the computational capability of resistive crossbars [4, 9] to realize bit-serial computation (e.g., ripple-carry addition), and to run data-intensive applications (e.g., neural network inferencing). In a resistive crossbar array, it is possible to take an entire column of the array and perform a bitwise Boolean primitive with an entire second column of the array. As a result, for a column of size \( n \), we can potentially increase the throughput by a factor of \( n \) by performing whole-column operations, which can amortize the latency of bit-serial operations if \( n \) is large enough.

In our MICRO 2021 paper, we show that while large crossbars (i.e., \( n \geq 1024 \)) have traditionally been seen as an effective way to amortize bit-serial latencies and peripheral circuit area for PUM architectures, there is a fundamental limit to increasing \( n \). Even with aggressive technology scaling, it will be challenging to achieve a value of \( n \) that is much larger than 128 in practice (see Section 3.3 of the MICRO 2021 paper). This motivates us to design RACER (Resistive Accelerated Computation for Energy Reduction). RACER is the first PUM architecture that addresses this array size limitation, relying on a novel execution model we call bit-pipelining to operate on \( w \times n \) words at a time, for a \( w \)-bit word, and provide high throughput for a range of data-intensive operations.

Unlike prior works, which focus on a particular memory technology and logic primitive, RACER can be adopted for any resistive crossbar memory (including ReRAM and MRAM) that can perform any functionally-complete set of logic primitives in memory. In this work, we use redox-based RAM (ReRAM) as a motivating example, and base the architecture on small \( 64 \times 64 \) ReRAM tiles capable of performing whole-column NOR [5]. Our design decisions in RACER use state-of-the-art device-level information (see Section 3.2 of our MICRO 2021 paper) to drive the co-design of circuits, microarchitecture, and an ISA.

**Bit-Pipelining.** RACER exploits parallelism across multiple tiles by striping each bit of a \( w \)-bit word across \( w \) tiles, as shown in Figure 1 (Tile 0 holds the least-significant bit, or LSB). RACER can realize bit-serial operations by iteratively applying column-wide Boolean logic primitives (e.g., NOR in ReRAM) one tile at a time. As a simple example, RACER performs ripple-carry addition tile-by-tile: at each bit \( i \), it generates the sum and carry-out bits in Tile \( i \) for \( 64 \) different addition calculations simultaneously, and then propagates the carry-out bits to Tile \( i + 1 \) to compute the next bit.

To enable inter-tile data transfer, RACER uses buffers, \( 1 \times 64 \) crossbars (made of the same device as the tiles) that connect to a pair of tiles using programmable pass gates (Figure 1). Buffers allow tiles to transfer data without using area-intensive CMOS converters, and allow us to tightly lay them out next to the tiles, incurring minimal area overhead. RACER ensures that a buffer connects to only one tile at a time. For our addition example, after computing the carry-out bits in Tile \( i \), RACER connects Tile \( i \) to Buffer \( i \), copies the carry-out column into the buffer, and then connects the buffer to Tile \( i + 1 \). The buffer’s contents are finally copied into Tile \( i + 1 \), where the column is used as the carry-in bits for bit \( i + 1 \)’s addition.

Our decoupled tile-by-tile computation allows us to pipeline across bits in RACER. We group \( w \) tiles and their corresponding buffers together to form a pipeline (we set \( w = 64 \) to support up to 64-bit computation) that can support bit-pipelining. We observe that after Tile \( i \) passes its generated column to Tile \( i + 1 \), Tile \( i \) is free to perform another sequence of combinational logic operations, on another set of \( n \) computations (different from the \( n \) computations that Tile \( i + 1 \) is working on in parallel). This allows RACER to have up to \( w \times n \) instructions being carried out simultaneously in a set of \( w \) tiles, each operating on \( n \) words at once.

**Controlling RACER.** In RACER, we exploit our observation that many bit-serial operations perform the same Boolean primitives on each bit to design efficient control logic. For such operations, RACER generates a sequence of NOR primitives for one tile, and then propagates the sequence from tile to tile. A byte group (Figure 2a) contains the control circuitry to enable bit-pipelining across eight consecutive tiles, with one micro-op queue per tile. Each queue holds a sequence of micro-ops (i.e., NOR primitives), and can control which columns are operated on. RACER propagates primitives from the head of one micro-op queue to the tail of the next micro-op queue in a byte group. RACER can configure whether adjacent

---


**Note**: We use the term *resistive memory* to refer broadly to resistance-based non-volatile memories (e.g., PCM, MRAM, ReRAM), while we use ReRAM to refer specifically to oxide-based switches (often referred to as memristors).
byte groups are connected together to allow propagation across byte groups, effectively enabling the ability to perform bit-pipelining at the 8-, 16-, 32-, or 64-bit granularity (to match common data widths). The micro-ops drive selection voltages in per-tile decode & drive units, which serve as the interface between the technology-agnostic byte group circuits and the technology-specific crossbar voltages.

While RACER is optimized for bit-pipelined operations, we can use it to enable a number of shift-based, non-pipelined operations. We demonstrate how RACER can perform integer multiply, multiply–accumulate, integer divide, and CORDIC-based trigonometric functions and square root. While the execution behavior for non-pipelined operations is more complicated than that of bit-serial operations, we can leverage RACER’s bit-stripping and ReRAM buffers to efficiently perform these operations without additional hardware (see Section 5.3 of our MICRO 2021 paper).

Significance. We highlight two key contributions of RACER. First, the array size limitation is expected to hold for any resistive crossbar technology capable of performing whole-column operations. RACER’s decode & drive units are the only components that are device-specific, with the rest of the architecture and abstraction being fully device-agnostic, allowing for a wide range of technologies to overcome array size limitations (we discuss this in our follow-on work [8]). Second, RACER provides ultra-efficient, highly-scalable support for a wide range of workloads, going beyond simply eliminating movement on the memory bus. Compared to RTX-2070, RACER achieves a 12× speedup and 17× energy savings. While not shown in the figure for brevity, RACER achieves a 6.7× speedup and 1.3× energy savings compared to DC, as DC’s much larger SRAM cells and discrete CMOS logic limit its memory capacity, resulting in costly data swapping between DC and main memory.

References