Enabling Practical Processing Using Emerging Memories

Saugata Ghose
https://ghose.cs.illinois.edu/

EFCL & SAFARI Live Seminar Series • October 19, 2022
Emerging Applications Are Increasingly Data-Centric

- Database Analytics
- Graph Workloads
- Machine Learning
- Drones and Self-Driving Cars
- Precision Medicine
- Mobile Device Workloads
But Computer Systems Haven’t Kept Up

- **Beefy processing engines** (CPUs, GPUs, accelerators)
  - Several cores, often heterogeneous
  - Many performance and power optimizations

- **Designed for infrequent memory accesses**
  - Caches highly dependent on locality
  - Long, narrow off-chip memory channel to connect CPU with DRAM

Modern architectures remain compute-centric
Waste significant energy and time on data movement
Emerging Memories Offer New Compute Opportunities

- **Processing-in-memory (PIM), or near-data processing (NDP)**
  - Add some compute capability to memory
  - No need to move data across memory channel

- **Recent innovations have made PIM hardware viable**
  - New memory architectures and heterogeneous integration approaches
  - New ways of having memory devices interact with each other

- **Can enable efficient cloud-free computing at the edge**

Many PIM-capable architectures and memory devices are being explored in research today
Real PIM Prototypes Finally Exist!

### Processing-Near-Memory (PNM)
- **Discrete logic** in or near the memory chip
- Most popular w/ 3D-stacked DRAM or silicon interposers
- Examples
  - UPMEM PIM–DRAM
    - Big Data Accelerator
  - Samsung Aquabolt-XL (a.k.a. HBM-PIM, AxDIMM)
  - SK hynix GDDR6-AiM

### Processing-Using-Memory (PUM)
- **Electrical interactions between memory cells** w/ little additional logic
- Possible w/ DRAM, SRAM, ReRAM, MRAM, PCM, NAND flash
- Examples
  - Mythic Analog Matrix Processor (NAND flash)
  - IBM Hermes (PCM)
But Are These Ideas Currently Practical at Scale?

- **Novel architectures and devices aren’t enough**
  - What is the *lifetime* of the underlying devices?
  - How do *programmers* use these chips?
  - How can we design realistic chips that are *practical to manufacture*?

- **How big is the market?**
  - ASICs for machine learning: lots of interest
  - *Many other domains can benefit from PIM*
  - ASICs for other applications: a harder sell
  - Can we make *general-purpose data processors*?
  - Can we enable edge processing of ML + data?

We need to work across the entire compute stack to have a chance at the widespread adoption of PIM
A Roadmap for Completing a Full Stack for PIM

Emerging/novel hardware requires cooperation between software, architecture, circuits, and device physics
- Need to consider fundamental limits of materials
- Need to understand/capture evolving limits of devices based on manufacturing technologies

RACER [Truong+ MICRO 2021, Truong+ JETCAS 2022]
- Cross-stack PUM that architects around contemporary device limits
- Fully-synthesized front end, peripherals, interconnects
- 107x speedup, 189x energy savings vs. 16-core CPU
Introduction

RACER: Architecting Around Small Tiles

OSCAR: Designing Devices Around Practical Limits

Where Do We Go From Here?

Closing Thoughts
Many Memory Technologies Support PUM (Not Comprehensive)

- **DRAM**
  - Capable of **MAJ** (AND/OR), **NOT** (with special cells)
  - Examples: Ambit, SIMDRAm, ComputeDRAM, DRISA, Fulcrum

- **SRAM**
  - Capable of **AND**, **NOR**, **XOR**, **COPY** (all through sense amplifier mods)
  - Examples: Compute Caches, Neural Cache, Duality Cache, CAPE

- **NAND Flash**
  - Capable of **AND**, **OR**, **NAND**, **NOR**, **XOR**, **XNOR**, **MULT**
  - Examples: Parabit, Flash-Cosmos, Mythic Analog Matrix Processor

- **ReRAM**
  - Capable of **NOR**, **NAND**, **OR**, **IMP**, **MULT**
  - Examples: MAGIC, FELIX, RACER, Yang+ Nature 2013, ISAAC, PRIME

- **MRAM**
  - Capable of **MAJ**, **NAND**, **NOR**, **TCAM**
  - Examples: AC-DIMM, MRIMA

- **PCM**
  - Capable of **AND**, **OR**, **XOR**, **INV** (all through peripheral circuitry mods), **MULT**
  - Examples: Pinatubo, IBM HERMES
ReRAM: A Potential PUM Candidate

- **Data stored as a resistance**
  - 1S1R: a *selector device* in series with a *resistive memory switch*
  - Typically organized as a *crosspoint array*

- **Analog interaction between multiple ReRAM cells can perform meaningful computation**
  - Multi-level ReRAM cells can perform dot product, low-precision multiply
  - **MAGIC:** single-level ReRAM cells can perform **NOR** [Kvatinsky+ TCAS II 2014]

- **Similar operations demonstrated in DRAM, SRAM, MRAM**
ReRAM Arrays a Good Fit for Bit-Serial Compute

- **Bit-serial operations**
  - Perform operations one bit at a time
  - Example: ripple-carry add

- **NOR-capable ReRAM can perform many bit-serial functions**
  - Addition/subtraction
  - Content search (like a content-addressable memory, or CAM)

- **Bit-serial operations incur long latencies**

To compensate for long latencies, ReRAM can compute on whole columns of data at once to exploit data-level parallelism
Issues with Processing Using ReRAM

- The larger the crossbar size, the bigger the throughput for whole-column operations

- Whole-column operations limit the crossbar size
  - Each extra cell in a column adds current \( \rightarrow \) grows proportionally to crossbar size
  - Limited by the current carrying capacity of a wire
  - Large currents permanently damage the metal wires (see MICRO 2021 paper for analysis)

Whole-column operations are realistically possible only when column length < 200 cells!

How can small tiles deliver high throughput at low overhead?
RACER: Optimizing PUM for Small Memory Tiles

- State-of-the-art processing-using-memory architectures keep whole chunks of a word in a single tile

- In RACER, we distribute each bit of a word to a different tile
RACER: Dealing with Bit-Serial Ops Across Tiles

- We add 1x64 ReRAM column buffers
  - Enables tile-to-tile communication
  - Connects to an adjacent tile using pass gates

- We enable a new technique that we call **bit-pipelining**
  - Treat each tile as a pipeline stage
  - With $t$ tiles, we can operate on $t$ columns of words at once
- Core control circuitry for bit-pipelining

- Each bit (i.e., each tile) repeats the same exact operations

- NOR instructions (micro-ops) are stored in micro-op queues
  - Each tile has a dedicated queue
  - Queue $i$ sends its micro-ops to Queue $i+1$

- Enables efficient support of 8-/16-/32-/64-bit operands
Cluster: group of pipelines
- Clusters operate independently
- Only one set of control queues and peripheral circuits per cluster

Chip can contain however few/many clusters as needed
### An Architectural Abstraction for RACER

- **RACER core**: pipeline w/ 32 kB of data
  - Corresponds to 64 tiles connected w/ buffers
  - Each core has local access to data from 512 cores
  - Global network gives each core access to entire chip’s data (up to 8 GB)

- **Vectorized ISA for easy programmability**
  - 64 words at once
  - Can support non-bit-pipelined instructions†

#### Table: Instruction Set Architecture (ISA)

<table>
<thead>
<tr>
<th>Op.</th>
<th>Description/Notes</th>
<th>Op.</th>
<th>Description/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Two’s complement add</td>
<td>ABS</td>
<td>Absolute value</td>
</tr>
<tr>
<td>SUB</td>
<td>Two’s complement subtract</td>
<td>MUX</td>
<td>Multiplex (i.e., choose)</td>
</tr>
<tr>
<td>POPC</td>
<td>Population count</td>
<td>RELU</td>
<td>Rectified linear unit</td>
</tr>
<tr>
<td>CMPEQ</td>
<td>Check equality</td>
<td>LSHIFT</td>
<td>Left shift by 1</td>
</tr>
<tr>
<td>FUZZY</td>
<td>Fuzzy search</td>
<td>RSHIFT</td>
<td>Right shift by 1</td>
</tr>
<tr>
<td>MUL†</td>
<td>Multiply (only 8-/16-/32-bit)</td>
<td>SQRT†</td>
<td>CORDIC square root</td>
</tr>
<tr>
<td>MAC†</td>
<td>Multiply–accumulate</td>
<td>SIN†</td>
<td>CORDIC sine</td>
</tr>
<tr>
<td>DIV†</td>
<td>Division (returns quotient &amp; remainder)</td>
<td>COS†</td>
<td>CORDIC cosine</td>
</tr>
<tr>
<td>MAX</td>
<td>Searches for the maximum number</td>
<td>EXP†</td>
<td>CORDIC exponent</td>
</tr>
<tr>
<td>MIN</td>
<td>Searches for the minimum number</td>
<td>CAS</td>
<td>Compare and swap</td>
</tr>
</tbody>
</table>

#### Boolean Operations

| NOR  | Bitwise NOR                   | OR   | Bitwise OR                    |
| NAND | Bitwise NAND                  | AND  | Bitwise AND                   |
| NOT  | Bitwise NOT                   | XOR  | Bitwise XOR                   |

#### Data Transfer Operations

| MOV  | <MOV buff [dst] = buff [src]> | SHIFT | Parallel data shift          |
|      | Moves data stored in buffers of core src to buffers of core dst |      | dst = src + stride           |

#### Configuration Operations

| SET  | <SET start, stop, stride>    | UNSET | Turns off all RACER cores that are active |
|      | Turns on RACER core i for i ∈ range(start, stop, stride) |      |                                |
Methodology

- Iso-area comparisons to four state-of-the-art platforms
  - **Baseline:** 16-core Xeon 8253 CPU + 8GB off-chip DRAM
  - **eMRAM:** 16-core Xeon 8253 CPU + 8GB on-chip MRAM
  - **RTX-2070:** GeForce RTX 2070 GPU
  - **DC:** Duality Cache, a compute-in-SRAM architecture [Fujiki+ ISCA 2019]

- We **model RACER at multiple levels of the stack**
  - Device-level ReRAM characteristics modeled using VerilogA with in-house device measurements
  - Control and peripheral circuits synthesized using FreePDK 15 nm
  - RACER ISA microbenchmarks executed using in-house simulator
  - Baseline modeled using MARSSx86 + DRAMSim2 + McPAT

- **Full paper:**
  [https://ghose.cs.illinois.edu/papers/21micro_racer.pdf](https://ghose.cs.illinois.edu/papers/21micro_racer.pdf)

- **Simulation framework open-sourced:**
  [https://doi.org/10.5281/zenodo.5495803](https://doi.org/10.5281/zenodo.5495803)
RACER Increases Performance vs. CPU/GPU

107× speedup vs. CPU
thanks to RACER’s tile-/pipeline-/cluster-level parallelism

71× speedup vs. eMRAM
as embedded memory does not reduce frequent data movement

12× speedup vs. GPU

7× speedup vs. DC (not shown)
thanks to RACER’s in-situ computation and tile-/pipeline-/cluster-level parallelism
RACER Significantly Reduces Energy

189× savings vs. CPU
thanks to RACER’s in-situ computation and fast low-power circuitry

94× savings vs. eMRAM
as embedded memory mostly reduces only the energy used by off-chip network

17× savings vs. GPU

1.3× savings vs. DC (not shown)
5× savings vs. DC for applications that trigger frequent data swapping
How Can ReRAM Computing Save Energy?

- **ReRAM write energies today are high: 1 pJ**
  - However, *scales favorably with V, R, speed*
  - Advanced assumptions: 1000x reduction is not unreasonable
  - **Wire charging energies are amortized** over 10–100 simultaneous operations

- **CMOS logic energies**
  - Gate charge energy is still tiny: 100x less than even advanced ReRAM
  - However, *wire charging energies can be 100–1000x larger* – no amortization

- **Control circuitry is a key aspect of energy consumption**
  - RACER: efficient million-way parallelism
  - Amortizes instruction generation, control circuitry

---

### ReRAM Energy Calcs

<table>
<thead>
<tr>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1V</td>
<td>V</td>
</tr>
<tr>
<td>0.01 fJ</td>
<td></td>
</tr>
<tr>
<td>0.2 fF/um</td>
<td></td>
</tr>
</tbody>
</table>

### CMOS Logic Energy Calcs

<table>
<thead>
<tr>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Voltage</td>
<td>1 V</td>
</tr>
<tr>
<td>Gate Capacitance</td>
<td>10 aF</td>
</tr>
<tr>
<td>Gate Charge Energy (0.5 CV^2)</td>
<td>0.01 fJ</td>
</tr>
<tr>
<td>Wire Cap/ Length</td>
<td>0.2 fF/um</td>
</tr>
</tbody>
</table>

- **Write Charge Energy**
  - 0.1 um wire: 0.01 fJ
  - 1 um wire: 0.1 fJ
  - 10 um wire: 1 fJ
  - 100 um wire: 10 fJ
Introduction

RACER: Architecting Around Small Tiles

OSCAR: Designing Devices Around Practical Limits

Where Do We Go From Here?

Closing Thoughts
RACER builds upon the 1S1R-based MAGIC logic family
• Architecture is independent of access topology and logic family
• Are there other options besides 1S1R and MAGIC?

What constraints does MAGIC impose?
• Required ratios between set, reset, logic voltages
• Required voltages to enable different
  • Are these compatible with typical ReRAM devices?
• What are other logic families that can work with RACER

How do we integrate different logic families with the architecture?

Are we really stuck with 1S1R?
Existing 1S1R Logic Is Highly Constrained

- **MAGIC** [Kvatinsky+ TCAS II 2014]
  - NOR logic family
  - Switching constraints: \( 2V_{\text{reset}} < V_{\text{logic}} < V_{\text{set}} \)

- **FELIX** [Gupta+ ICCAD 2018]
  - NAND, NOR, OR logic family
  - Same switching constraints same as MAGIC

- Change assertion voltages on input columns to perform logic other than NOR

<table>
<thead>
<tr>
<th>Input Assertion Voltage (V)</th>
<th>Voltage Drop (V) Over Output When ...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input 000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0.75</td>
<td>0</td>
</tr>
<tr>
<td>0.67</td>
<td>0</td>
</tr>
</tbody>
</table>
Visualizing the Constraints

- **MAGIC/FELIX**
  - $2V_{\text{reset}} < V_{\text{logic}} < V_{\text{set}}$
  - $V_{\text{logic}}$ can be $V_{\text{nor}}$ or $V_{\text{nand}}$
- **Typical (real) devices:** $V_{\text{set}} < 2V_{\text{reset}}$

Can we design a logic family that is compatible with typical devices?
**OSCAR NOR**
- Adds a fourth load resistor cell to balance voltage division
  - Does not require repeated initialization
  - We set it to 0V
- Assert $V_{\text{nor}}$ to input cells
- Assert $V_{\text{nor}} + \Delta$ to output cell ($\Delta$ makes operation non-destructive)
- **Constraints:** $V_{\text{nor}} > 4V_{\text{set}}$
- **No constraints on** $V_{\text{reset}}$

**OSCAR OR**
- Destructive: one of the input is the output
- Assert $V_{\text{or}}$ to one input
- Assert GND to the other input (this is also the output)
- **Constraints:** $V_{\text{set}} < V_{\text{or}} < 2V_{\text{reset}}$
Decode & Drive Units Support Multiple Logic Families

Data/micro-op propagation for bit-pipelining
What About Access Topologies?

- Nothing about RACER is specific to 1S1R!

- What happens if we use 1T1R arrays instead?
  - **Proven selector technology**
    » More robust than 1S1R
    » Eliminates half-select energy costs
  - **New select line**
    » Driven by $V_{\text{sel}}$
    » May be able to use $V_{\text{sel}}$ to reduce voltage drop
  - **BEOL integration harder w/ controllers**
    » Easier to lay out control circuitry side-by-side with array
    » **Reduces number of active pipelines by 24%**
RACER + OSCAR Provides Healthy Benefits

- **MAGIC NOR:** MICRO 2021 work
- **MAGIC NAND**
- **FELIX:** NAND, NOR, OR logic family
- **OSCAR:** our new NOR, OR logic family w/ significantly relaxed switching constraints

**OSCAR increases performance by 30% vs. MAGIC NOR**

**OSCAR increases energy savings by 37% vs. MAGIC NOR**
### CASCADE

- State-of-the-art neural network (NN) accelerator [Chou+ MICRO 2019]
- Over an order of magnitude throughput and energy improvements over CMOS-based NN accelerator (DaDiaNao)

### RACER outperforms CASCADE

- RACER+OSCAR: **3.16x throughput improvement** on average
- CASCADE outperforms RACER for sparse matrices
- RACER is better for edge computing: can run many non-NN operations & microbenchmarks that CASCADE can’t
Introduction

RACER: Architecting Around Small Tiles

OSCAR: Designing Devices Around Practical Limits

Where Do We Go From Here?

Closing Thoughts
PUM Can Open Up New Domains of Computing

- **Edge devices capable of data analytics in the field**
  - RACER’s 180x+ energy savings enable new local uses of compute
  - Avoids costly overheads of sending raw data across the network to the cloud
  - Combined with non-volatility, can enable **highly intermittent compute**
  - Many use cases
    - Smart infrastructure sensors
    - Mini delivery drones
    - Battery-operated genome & virus sequencing
    - Satellite image processing & analytics

- **Highly-efficient data processing in the cloud**
  - Multi-tenant systems can’t have custom ASICs
  - RACER’s reconfigurability can allow it to accelerate many data-intensive programs
  - Improves **scalability & sustainability** of cloud servers

So why aren’t we building PUM-based computers already?
Emerging memory devices not yet ready for prime time
- Voltages need to come down, on/off ratios need to improve
- **Reliability and device lifetimes must increase**
  - Current ReRAM achieves $10^{12}$ writes in research, $10^4 - 10^6$ at scale
  - RACER experiments: 10-year lifetime requires $10^{14}$ writes
- 1T1R devices may offer better viability than 1S1R

Digital PUM: whole-column operations limit the crossbar size
- **Each extra cell in a column adds current** → grows with crossbar size
- Limited by the current carrying capacity of a wire
- Difficult to do whole-column ops on columns > 200 tiles

Analog PUM: multi-level resistances need significant device precision
- Significant trouble controlling intermediate Rs
- Peripheral circuitry overhead can dominate
Several Other Hardware Challenges Remain

- DRAM-, SRAM-based PUM have their own challenges
  - DRAM often requires 1000s-wide vectors and/or transpose units
  - SRAM has low density, and is often limited w.r.t. parallelism

- What about controlling the PUM arrays?
  - Lots of works focus on the cool device or architecture, but forget the circuit and wiring overhead of distributing instructions
  - We’re often dealing with million-way parallelism
  - How do we handle branches and divergence?
  - How do we coordinate operations across arrays without bottlenecks?
Even If We Solve Devices, We Still Need a Full Stack!

- There is no complete HW/SW stack for PUM (or PIM)

- We don’t really know which hardware design(s) will win
  - What underlying memory devices?
  - What primitives?
  - What microarchitecture?

- Difficult to make software with so many unknowns

- Wait! We’ve solved this before for conventional computers…

We need to think about general abstractions that can easily be ported to a wide range of microarchitectures

It’s now time for a PIM ISA!
Programmers Need More Than Just a PIM ISA

- Once PIM HW exists, programmers must be able to use it easily

**How do we write programs?**

- Develop **programming models** for PIM
  - Tough sell: force them to **learn a new programming model**
  - Initial path to broad adoption: **adapt PIM to existing, friendly models**
  - Should enable mechanisms such as threads, shared memory, coherence/consistency

- Develop **compilers** that
  - Automatically identify opportunities for PIM
  - Generate PIM-compatible binaries

- Provide **support to partition data** across PIM cores and/or memory arrays

**How do we use PIM in the context of a whole system?**

- Develop an **OS or runtime** that can manage PIM execution

- Enable support for **multitenancy**
  - Spatial multiplexing & context switching support
  - Memory virtualization & protection
Introduction

RACER: Architecting Around Small Tiles

OSCAR: Designing Devices Around Practical Limits

Where Do We Go From Here?

Closing Thoughts
Conclusion

- Processing-using-memory (PUM) provides an opportunity to enable new types of computing
  - In-the-field data analytics for edge computing devices
  - Widely-usuable data processing acceleration for the cloud

- Emerging/novel hardware requires cooperation between architects, circuit designers, and device physicists
  - Example approach: RACER [Truong+ MICRO 2021, Truong+ JETCAS 2022]
    - Cross-stack PUM that architects around device limits
    - Fully-synthesized front end, peripherals, interconnects
    - 107x speedup, 189x energy savings vs. 16-core CPU
  - Architects/circuit designers need to be informed of & drive device research
  - We need more practical PIM-capable devices, longer lifetimes

- We need to co-improve device physics and architectures
- We need to start tackling difficult problems in the software stack
Thanks to My Collaborators

- ARCANA Research Group: https://arcana.cs.illinois.edu/
  - Minh S. Q. Truong
  - Eric Chen
  - Deanyone Su
  - Alex Glass
  - Ali Hoffmann
  - Shivani Prasad
  - Raghav Gupta

- CMU Data Storage Systems Center https://www.dssc.ece.cmu.edu/
  - James A. Bain
  - L. Richard Carley
  - Liting Shen
  - Marek Skowronski

- Supporters of ARCANA’s PUM Research
  - Scott Center for Energy Innovation at Carnegie Mellon University
  - Sandia National Laboratories
  - Apple Ph.D. Fellowship for Minh Truong
Enabling Practical Processing Using Emerging Memories

Saugata Ghose
https://ghose.cs.illinois.edu/

EFCL & SAFARI Live Seminar Series • October 19, 2022