Open-Source Processing-in-Memory (PiM) Architecture Design through FPGA Emulation: A Case Study Modeling Sieve

(Open-Source Computer Architecture Research)

OSCAR 2023

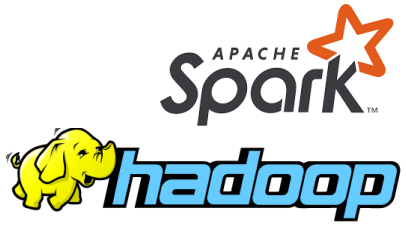
Khyati Kiyawat, Sergiu Mosanu, Mircea Stan, Kevin Skadron

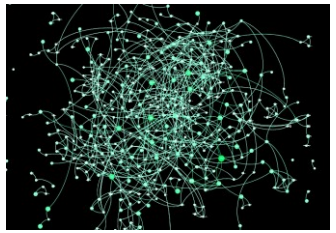
University of Virginia

UNIVERSITY of VIRGINIA

Towards In-memory compute

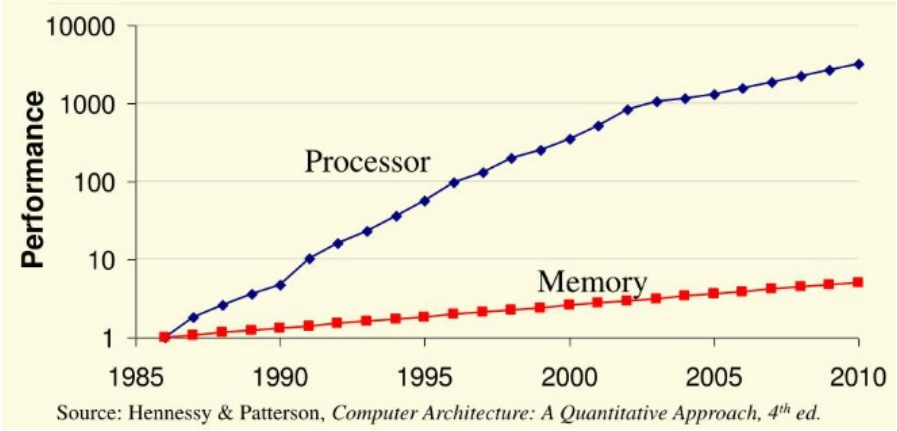
- Modern workloads are data-intensive





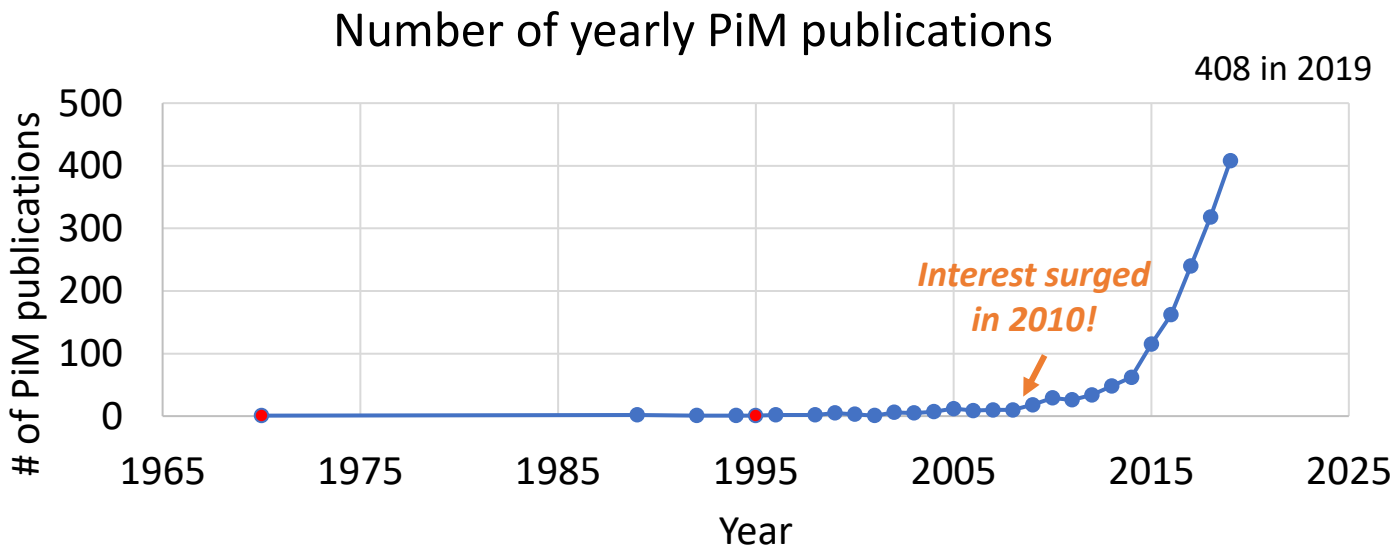
- Energy and performance cost of data movement is huge
  - 2-3 orders of magnitude more as compared to compute itself

- Necessitates Processing-in-memory architectures to address the ‘memory wall’ issue[1]



[1] Wulf, Wm A., and Sally A. McKee. "Hitting the memory wall: Implications of the obvious." *ACM SIGARCH computer architecture news* (1995)

Proliferation of PiM Architectures



Source: https://github.com/miglopst/PIM_NDP_papers/blob/master/paper_list.md

PiM-related publications are **increasing exponentially** each year

Unavailability of a unified modeling and evaluation framework that is fast and accurate

Challenges in Evaluating PiM Architectures

- Lack of standard PiM hardware

- Researchers today rely on
  - Hand calculations,
  - In-house simulators,
  - Statistical models for power/performance characterizations,
  - Modification of existing tools,
  - Chip fabrication, or
  - A combination of all

- Hard to reproduce, scale, and explore design space

Survey of various modeling approaches

Example	Approach	Fidelity	Speed	Underlying Memory Model	Design Space Exploration	Full System Evaluation	Affordability Adoptability
Micron SDRAM Models	**Verilog Behavioral Simulation**	High	Low	Interface, State, Timing, Dataflow	Flexible, Behavioral	Compatibility, Correctness, Ø	Affordable, Tedious
PIMSim, MultiPIM	**Software Simulation**	Low, Medium	Low	Timing, Ø	Flexible	OS/Application, Power, Performance	Affordable, Familiar
FASED, FireSim	**FPGA Accelerated Simulation**	Medium	Medium	Timing, Ø	Flexible	OS/Application, Power, Performance	Cloud price, Familiar
LiME, MEG	**Approximate FPGA Emulation**	Low, Limited	High	Interface, Approx. Timing, Dataflow	Constrained	OS/Application, Approximate Performance	Platform price, FPGA toolflow
PiMulator	***FPGA Emulation***	*High*	*High*	*Interface, State, Timing, Dataflow Data layout*	*Full flexibility*	*OS/App, Power, Hardware Model*	*Cloud price, FPGA toolflow, LiteX*
PiDRAM	**DRAM use violation**	Maximum	Realtime	Physical memory	Constrained	OS/App, Real hardware	Platform price, FPGA toolflow
Terasys, FlexRAM	**Hardware tape-out**	Maximum	Realtime	Physical memory	Constrained	OS/App, Real hardware	Prohibitively Expensive, ASIC & PCB

Ø = insufficient

Open-Source FPGA-based Emulation

PiMulator: a fast and flexible processing-in-memory emulation platform

Authors: Sergiu Mosanu, Mohammad Nazmus Sakib, Tommy Tracy, Ersin Cukurtas, Alif Ahmed, Preslav Ivanov, Samira Khan, Kevin Skadron, Mircea Stan Authors Info & Claims

DATE '22: Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe • March 2022 • Pages 1473–1478

Why FPGA?

- ✓ Reconfigurable
- ✓ Opens more DSE
- ✓ Prototype for tape-out

Why Emulation?

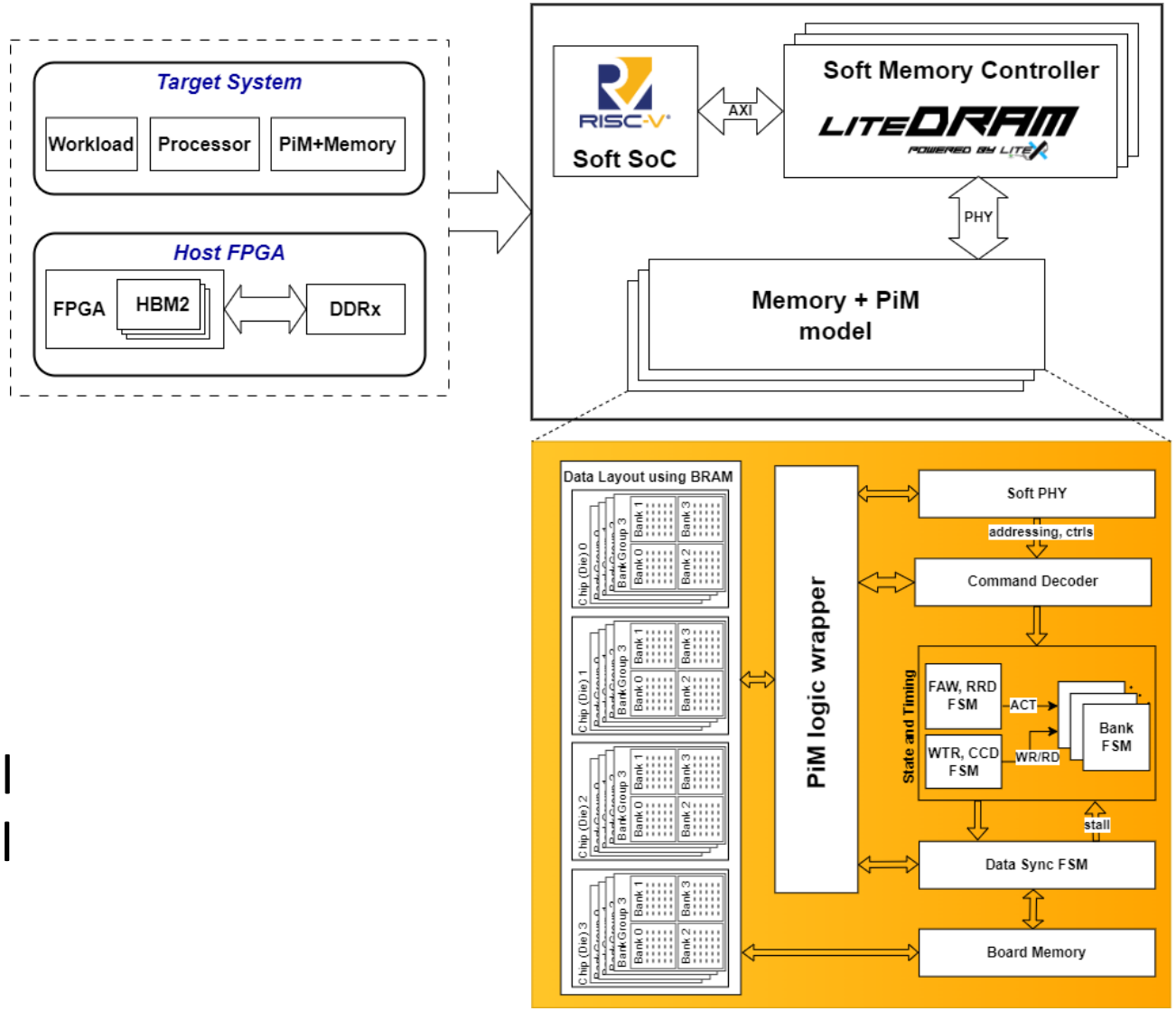
- ✓ Faster
  - 28× speedup than DRAMsim3[1]
- ✓ More accurate and reliable
  - Does not overlook any important hardware logic

[1] S. Li, Z. Yang, D. Reddy, A. Srivastava and B. Jacob, "DRAMsim3: a Cycle-accurate, Thermal-Capable DRAM Simulator," in IEEE Computer Architecture Letters.

PiMulator framework

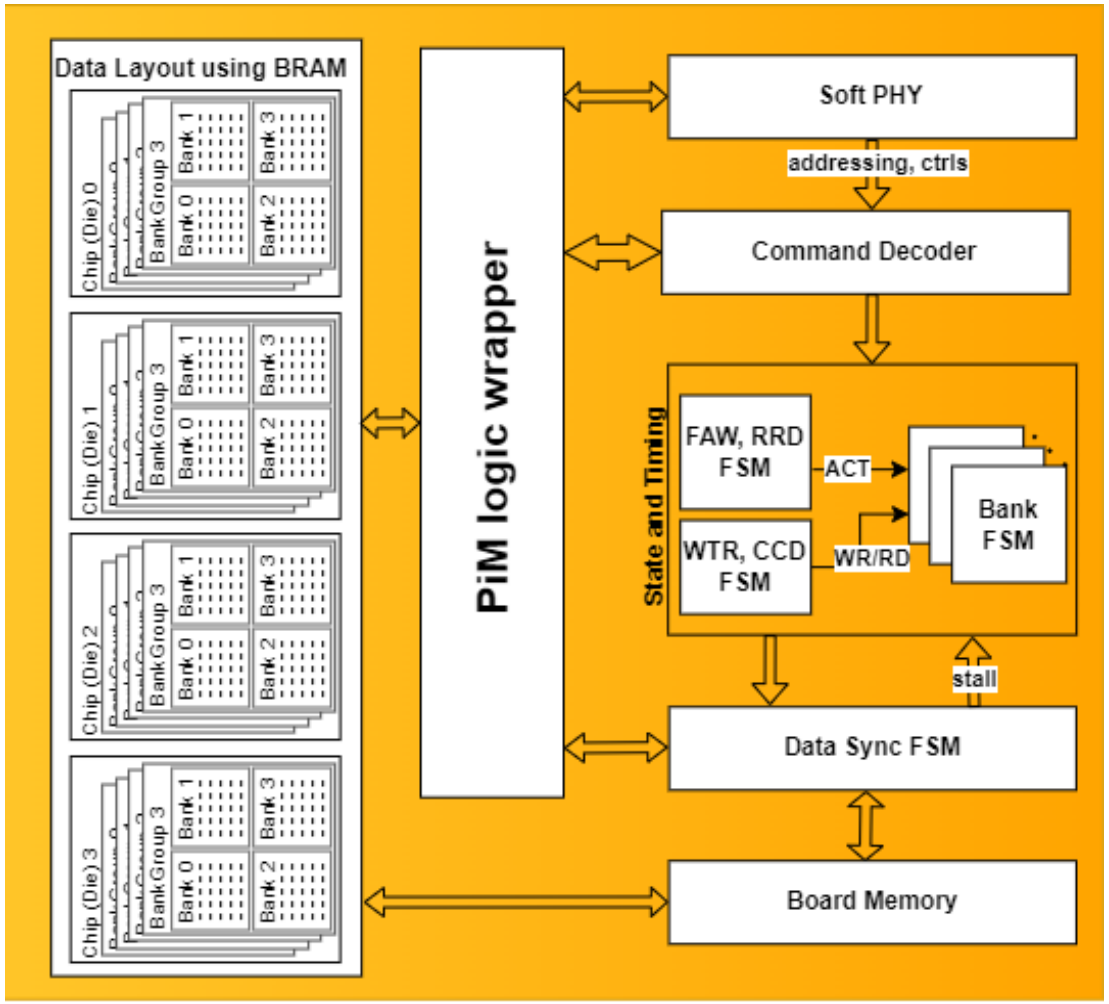
Separation of concerns

- Target system layer
  - Defined by configuration files
  - App, Processor, Memory, PiM
  - Runtime monitoring
- Host FPGA layer
  - Efficient resource utilization
- Logic model layer
  - Soft Nax/VexRISC-V CPU model
  - Soft LiteDRAM MEMCtrl model
  - Soft Memory+PiM model



Memory + PiM model

- Implements a parameterized DIMM channel in SystemVerilog
- Key elements include
  - a memory interface,
  - a command decoder,
  - a data bus, and
  - timing FSMs
- Flexible PiM logic wrapper



Parameterized SystemVerilog Implementation

PiMulator features

- Can **emulate different memory types** including DDRx, LPDDRx, GDDRx, and HBM2.
- Can be integrated with **LiteX**. LiteX provides several open-source IPs and utilities, and it supports various soft-core CPUs and FPGA boards. It also extends migen to define 100s of hardware.
- A **Data Synchronization Engine** is integrated into the model to expand the emulated memory's capacity using the available DRAM resources on the FPGA board.
- **FreezeTime**[1] technique of architectural virtualization to overcome the resource limitations of FPGAs and enable the modeling of large and complex compute and memory system

[1] S. Mosanu, *et al*, "FreezeTime: Towards System Emulation through Architectural Virtualization," Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Feb. 2023.

Case Study- Modeling Sieve

2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)

Sieve: Scalable In-situ DRAM-based Accelerator Designs for Massively Parallel k-mer Matching

Lingxi Wu
University of Virginia
lw2ef@virginia.edu

Rasool Sharifi
University of Virginia
as3mx@virginia.edu

Marzieh Lenjani
University of Virginia
ml2au@virginia.edu

Kevin Skadron
University of Virginia
skadron@virginia.edu

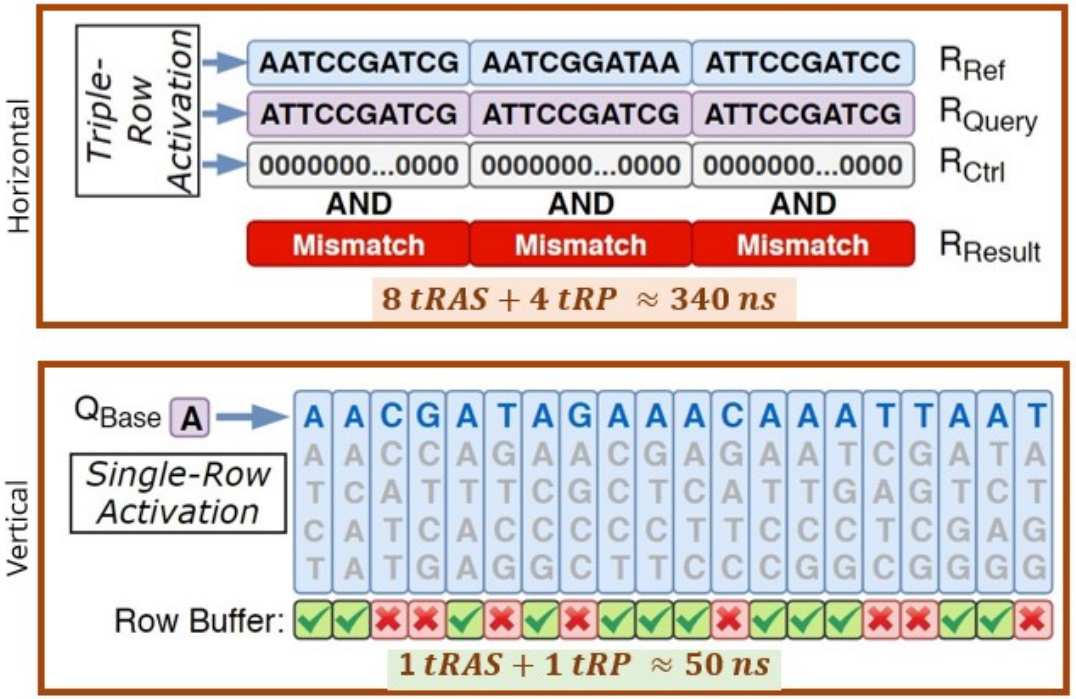
Ashish Venkat
University of Virginia
venkat@virginia.edu

<u>Objective:</u>

To extend PiMulator's modeling scope to include diverse PiM architectures at different levels of the DIMM hierarchy

- Validate software simulations by modeling it in hardware
- Facilitate larger design space exploration across the memory hierarchy

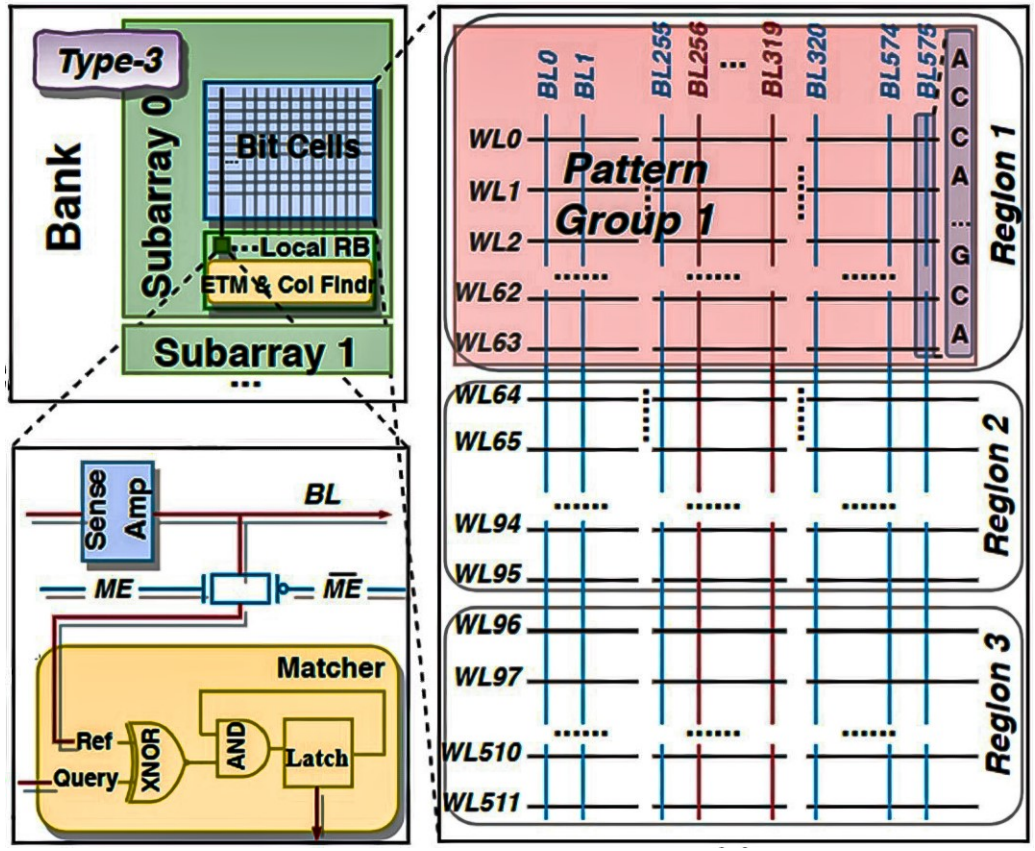
Why bit-serial architecture?



- More parallelism
- Saves cost of repeating query patterns across the columns
- Single-row activation saves energy
  - Triple-Row Activation takes 22% more energy
- Enables Early Terminate Mechanism (ETM)

Application: k-mer matching sits on the critical path of many genome analysis pipelines

Sieve Overview (Type 3, Subarray level)

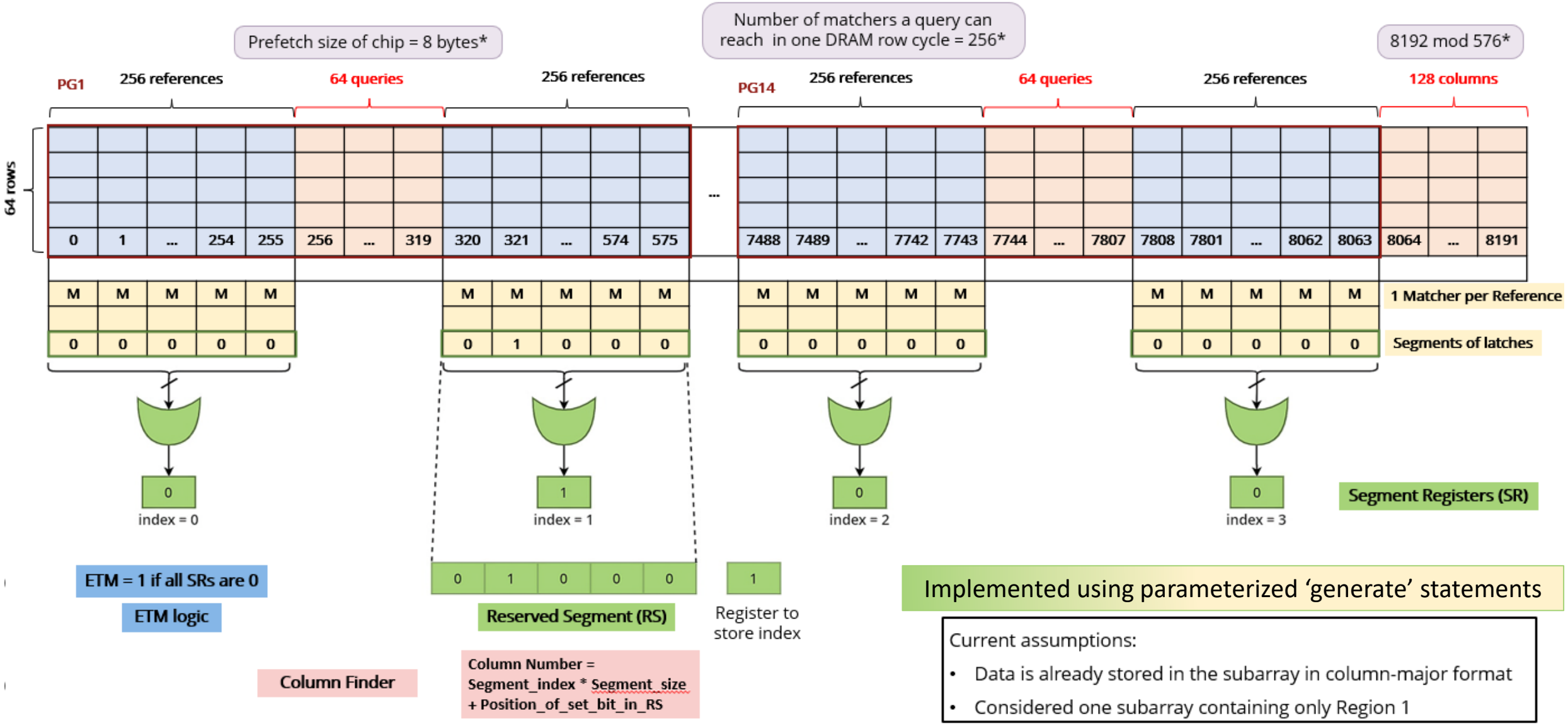


- Each subarray is partitioned into three regions:
- **Region-1** stores the interleaved reference and query k-mers
- **Region-2** stores the offsets to the starting address of payloads
- **Region-3** stores the actual payloads such as taxon labels

Data in Region-2/3 is stored in conventional row-major format

Sieve shows 326x/32x speedup and 74x/48x energy savings over CPU/GPU baseline

Pattern groups and Matching Implementation



Implemented using parameterized 'generate' statements

Current assumptions:

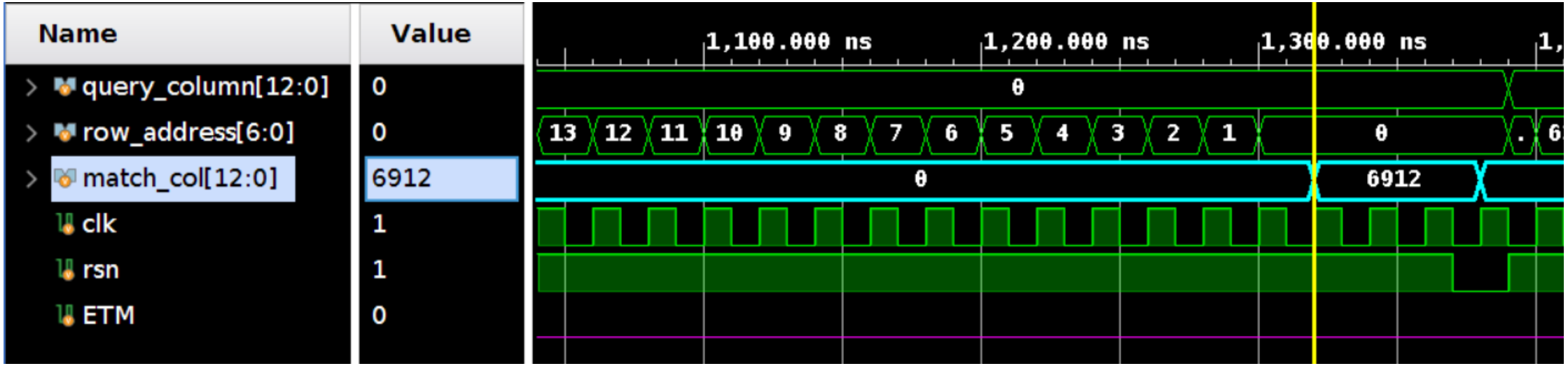
- Data is already stored in the subarray in column-major format
- Considered one subarray containing only Region 1

- Segment_size = 2^n (multiplication can be implemented using shifter)
- Position_of_set_bit_in_RS using Shift and Count logic

* These numbers are for DDR3_micron_32M_8B_x4_sg125

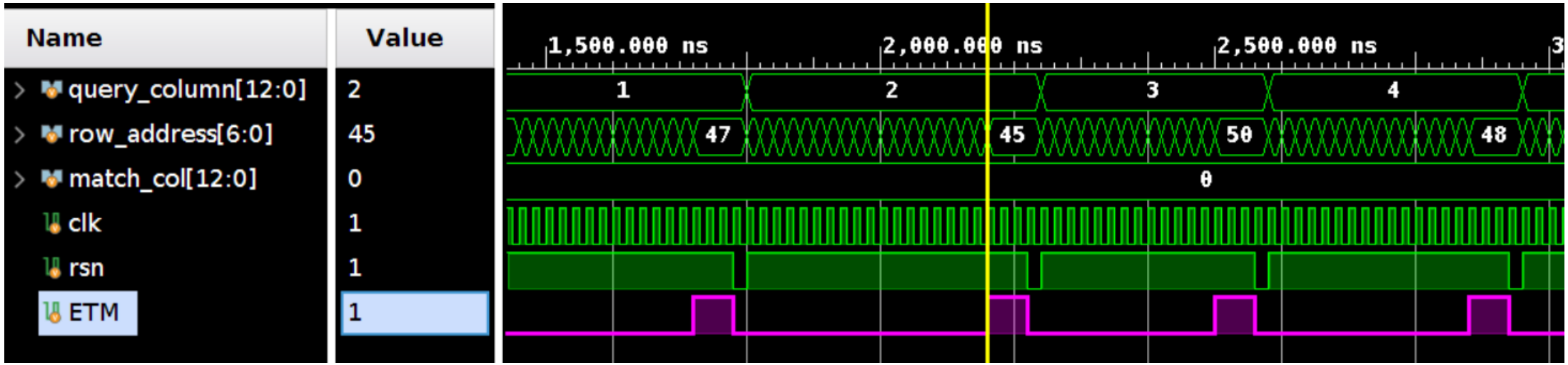
Matching Column Number

- Continued till all the rows were matched
  - since there was no ETM
- After matching row #0, the segment register with a 1 is copied to reserved segment, column finder computes the matching column number, here, 6912.

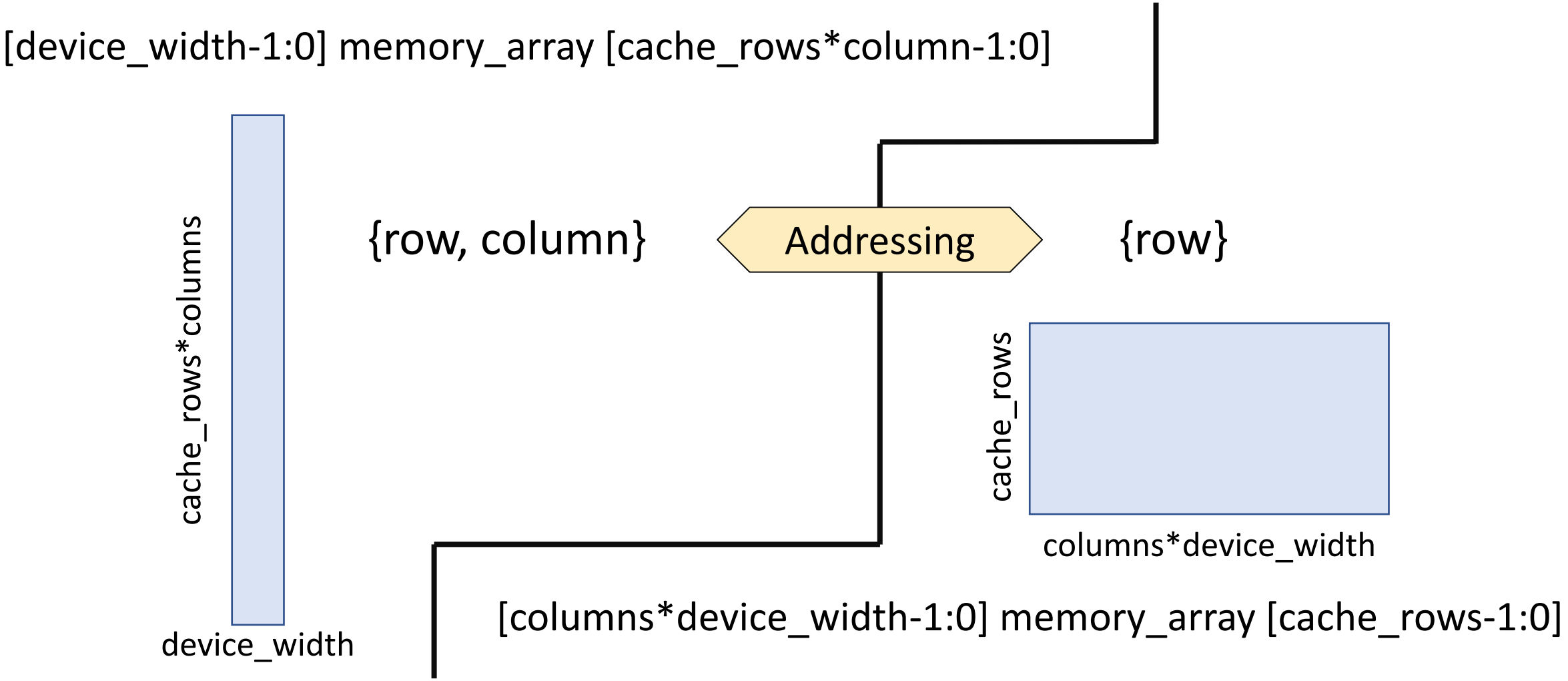


ETM saves energy and time

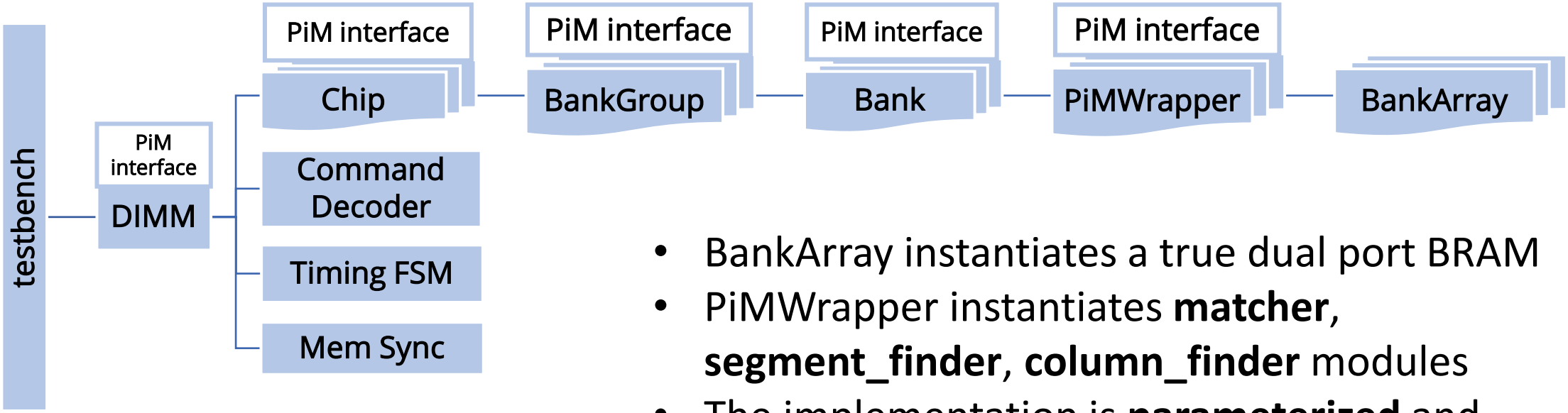
- For query 1, 2, 3, 4 :
  - As they do not match with any reference
  - ETM signal is set after all the segment registers shows mismatch
  - After row number 46, 44, 49, 47 respectively in this case



Updating the memory array



Integrating PiM wrapper with DIMM



- BankArray instantiates a true dual port BRAM
- PiMWrapper instantiates **matcher**, **segment_finder**, **column_finder** modules
- The implementation is **parameterized** and **scalable** (based on the availability of FPGA resources)

Insights from the functional modeling

BRAM utilization by memory (0.78%):

- A subarray with 64 rows and 8192 columns uses 64KB BRAM
- BRAM available on Xilinx Virtex-7 FPGA Board is 8MB
- Possible to scale to multiple subarrays and banks (4 BG * 4 B * 8 SA)

Opportunities to explore new design space

- 128 columns not needed for reference and query patterns can be used to:
  1. Preload two batches of query
  2. Introduce ECC/parity bits for the memory
     - Overheads of adding a parity checker (XOR) could not be justified for Sieve but worth considering for more generic architectures

To be able to visualize end to end integration and analyze power/timing reports

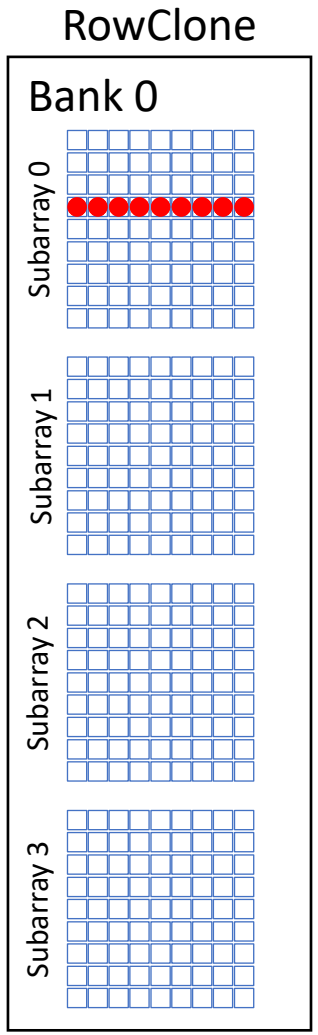
Ongoing work

- Compile the testbench to interact with memory controller in PiMulator
- Update timing FSM, and Command decoder to support custom PiM commands
- Scale the Sieve implementation for multiple subarrays and banks
  - Multiplex queries such that Query A/B -> Bank A/B, or Query C/D-> subarray C/D
  - Need to work on the addressing mechanism
- Running real-world workloads on the emulated PiM + Host system
- Making the *PiMWrapper* generic to include other bit-serial architectures

Strategies to emulate other PiM architectures

RowClone

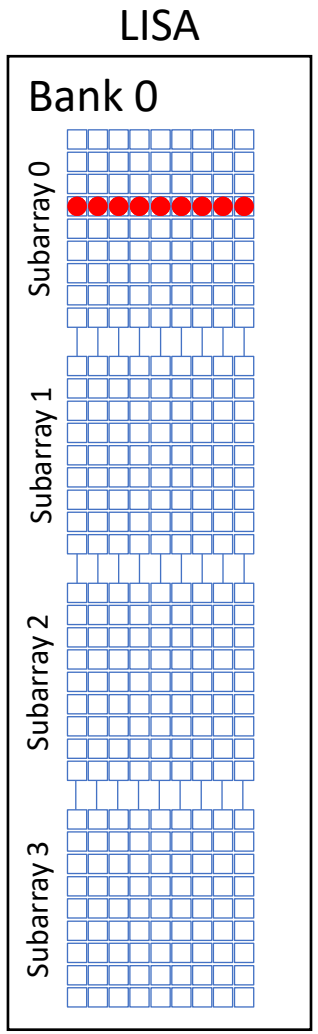
- Augment DSync tag table
  - support linking multiple memory rows to one local row
  - accounting for subarray membership
- Additional ACT (ReActivating) state in FSM
- Data flow between banks



Strategies to emulate other PiM architectures

LISA

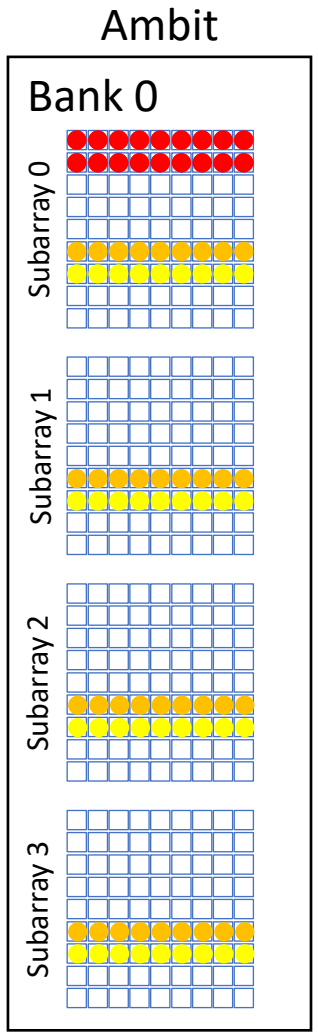
- Neighbor subarray membership = subarray line network



Strategies to emulate other PiM architectures

<u>Ambit</u>

- Model dedicated Ambit rows with LUTRAM
- Model AND, OR, NOT with LUTs
- RowClone → RowClone → Triple Row Activation
- Model timing with 3rd ACT (ReActivating) state



Summary

https://github.com/hplp/PiMulator

Model your PiM architectures using PiMulator,
Help us make the platform more robust and versatile

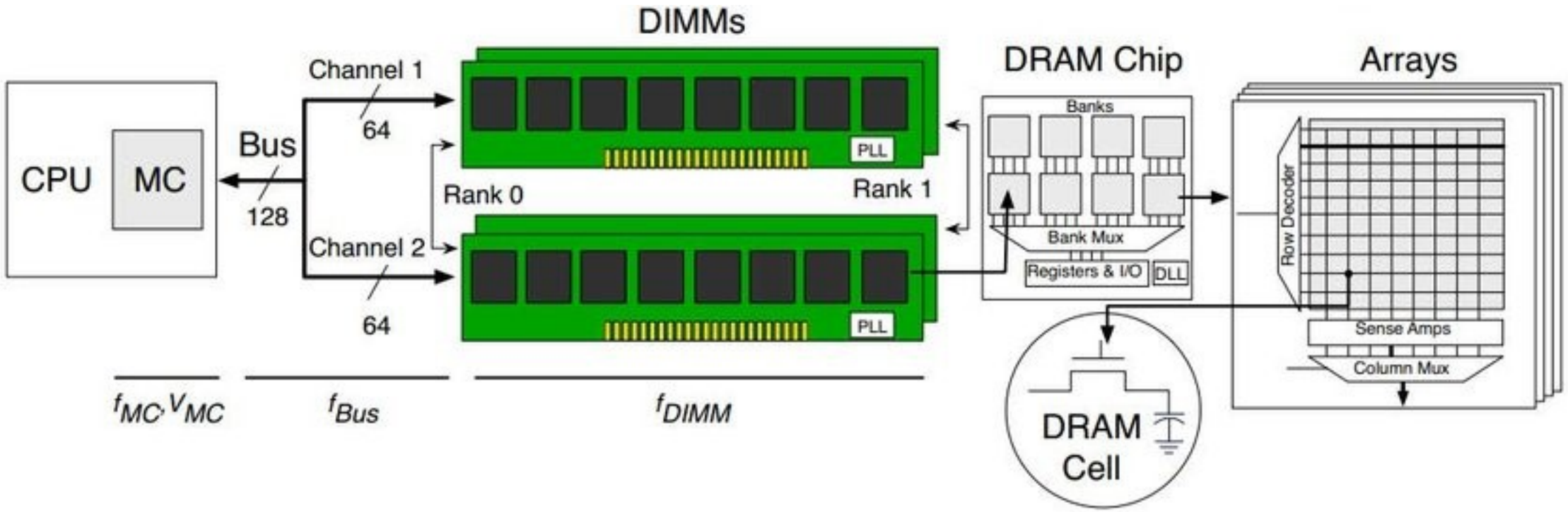
- Open-Source
- Software-like flexibility
- Hardware-like speed and fidelity
- Complex PiM prototyping and evaluation
- Design Space Exploration

Thank you! Any questions?

✉ khyati@virginia.edu

Backup Slides

Memory module structure



Jia, Gangyong & Li, X. & Yuan, Y. & Wan, J. & Jiang, Congfeng & Dai, Dong. (2014). PseudoNUMA for reducing memory interference in multi-core systems. 46. 39-46.