Processing in Memory for Energy-Efficient Computing

Kevin Skadron
Harry Douglas Forsyth Professor

University of Virginia
Dept. of Computer Science

This work is supported in part by CRISP and PRISM, centers in JUMP 1.0 and 2.0, Semiconductor Research Corporation (SRC) programs sponsored by DARPA; grants from the NSF and Booz Allen Hamilton, and MIST - an NSF I/UCRC center

Why we work on ModSim
(from an academic perspective)

- Academia – tendency to look where the light - and tools - are good
  - Good tools enable expansion for a new research area
  - Key enabler for research we want to do ourselves
  - Helps make it easier to compare research results
- Developing these tools (simulators, benchmarks, etc.) is rewarding
  - Fun research and helps the community
  - High citation counts illustrate the impact

ModSim observations

- Finding the right primitives and flexibility is key to success
- Successful example: thermal
  - HotSpot derived from first principles, model wide variety of designs simple abstractions for modeling diverse architectures
    - Relies on external tools for power modeling, e.g. McPAT, and floorplan (ArchFP)
    - People can use other tools, e.g. CAD tools
  - Allowed us to publish several papers on thermal management
    - But many more published by others
  - Unfortunately, failed to build collaborative development community
    - Instead – semi-competing teams and tools
      - Competition isn't necessarily bad! In HotSpot's case, incorporated some external models into unified framework

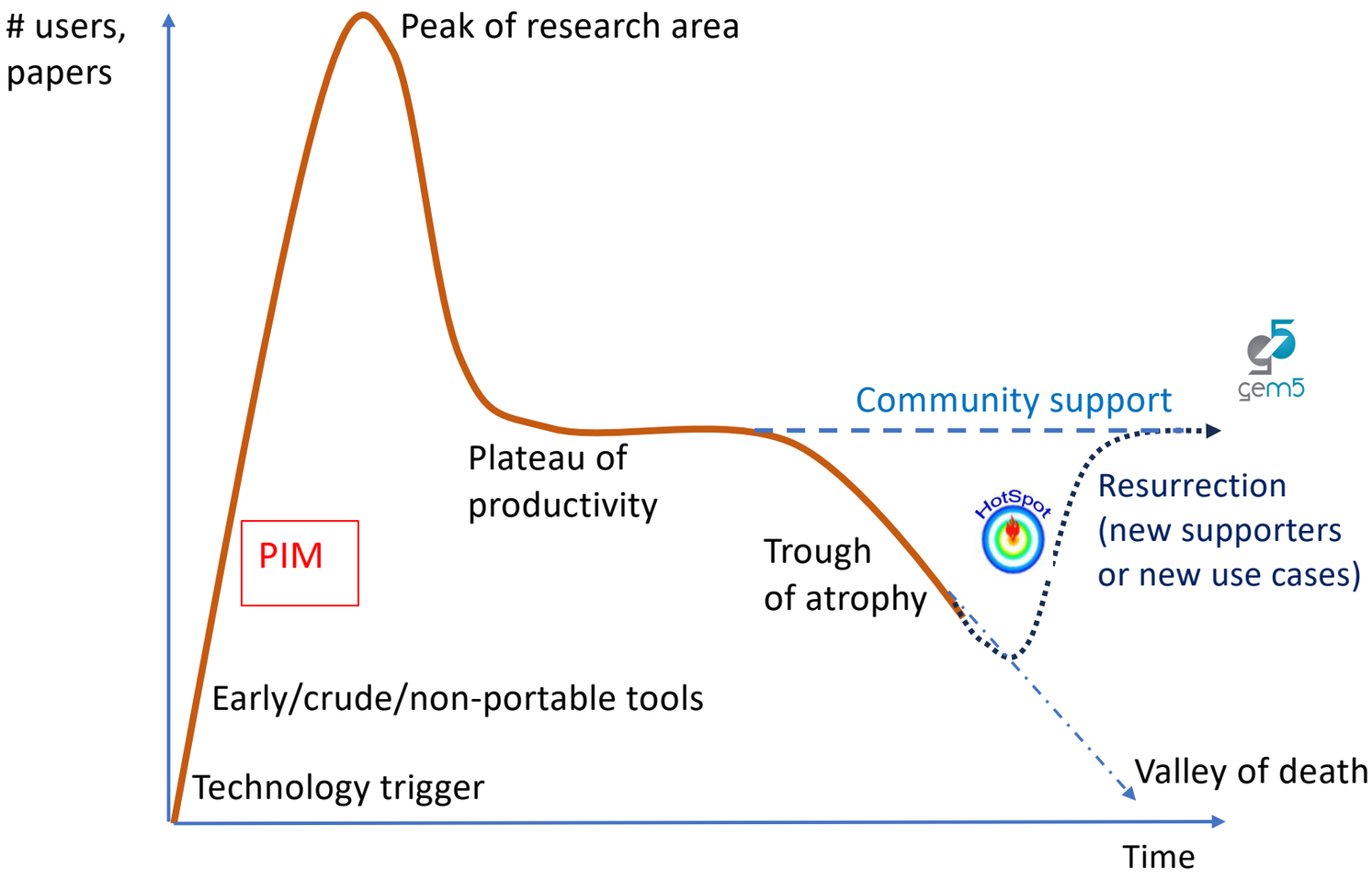
ModSim observations, cont.

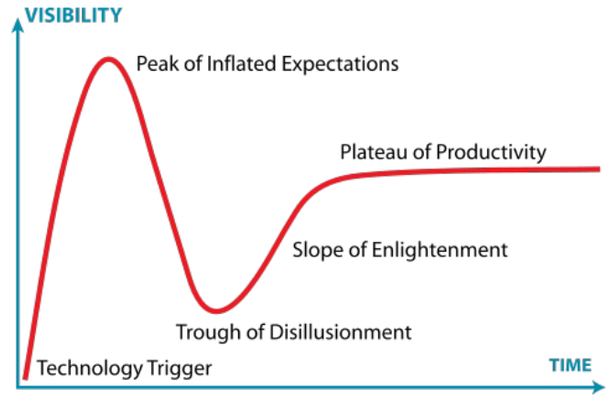
- Disappointing example: feedback control theory for dynamic adaptation
  - We used it in our thermal and power management techniques
  - But didn't find a *general* abstraction to make it easy to use with diverse architecture components
    - So not well explored in the architecture community
- In-between example: automata processing (VASim, AutomataZoo, etc.)
  - Developed simulation framework and benchmarks, continuing low-level use
  - I think we got the abstractions and infrastructure about right
  - But area remains small

Modeling lifecycle

- Early work – ad hoc models and benchmarks – crude and non-portable
  - Researchers just want to evaluate their architectural idea
  - Too early to think about building a community
  - But high barrier to entry in this research topic
- Somebody takes initiative to build general framework
- Tipping point: area gains momentum → framework is adopted by others → helps area gain further momentum
  - Tools lower barrier to entry
- Iterative development and support
  - This is often insufficiently supported by funding, publication opportunities, community
  - Tragedy of the commons, often simply a labor of love
- Gradual atrophy
  - PIs lose interest/move on, insufficient funding, popular research areas change, etc.
  - Building a *development community* can help with this
    - Help new users take ownership for continued support, relevance, and success
    - Gem5 has been a role model in this
    - Funding and recognition remain challenges
  - External dependencies may atrophy too!

Modeling lifecycle





Gartner hype cycle (Wikipedia)

Why Processing in Memory (PIM)?

- Many important applications bottlenecked by memory
  - Memory wall! (Term coined at UVA in mid-90s by Bill Wulf and Sally McKee)
- Data-access bandwidth available inside DRAM >> I/O bandwidth
- Spend lots of time and energy moving data - only to discard it
  - Eg, filtering in database table scan
- Energy efficiency is critical to success of computing field
  - From embedded systems to AI data centers
- Perhaps we can have our cake and eat it too – better performance and lower energy

Why PIM now?

- It was a hot topic many years ago and died out! What's new?
  - Pain and opportunity
- Pain back then: mitigated by improvements in caching, prefetching, etc. – also GPUs
- Pain now: Slowdown in Moore's Law, explosion in data, poor cache behavior, memory bottlenecks much more widespread, data center energy concerns
- Lack of opportunity then: lack of killer app, no clear market
- Opportunity now: Broad interest in accelerators; no other accelerator directly targets memory bottleneck
  - *Potential market - GEMV*
- PIM is just another accelerator - tackles the memory bottleneck
  - Happens to use DRAM as the implementation
  - But special in the potential to leverage existing DRAM market
- What is the "killer app" that will create a market?

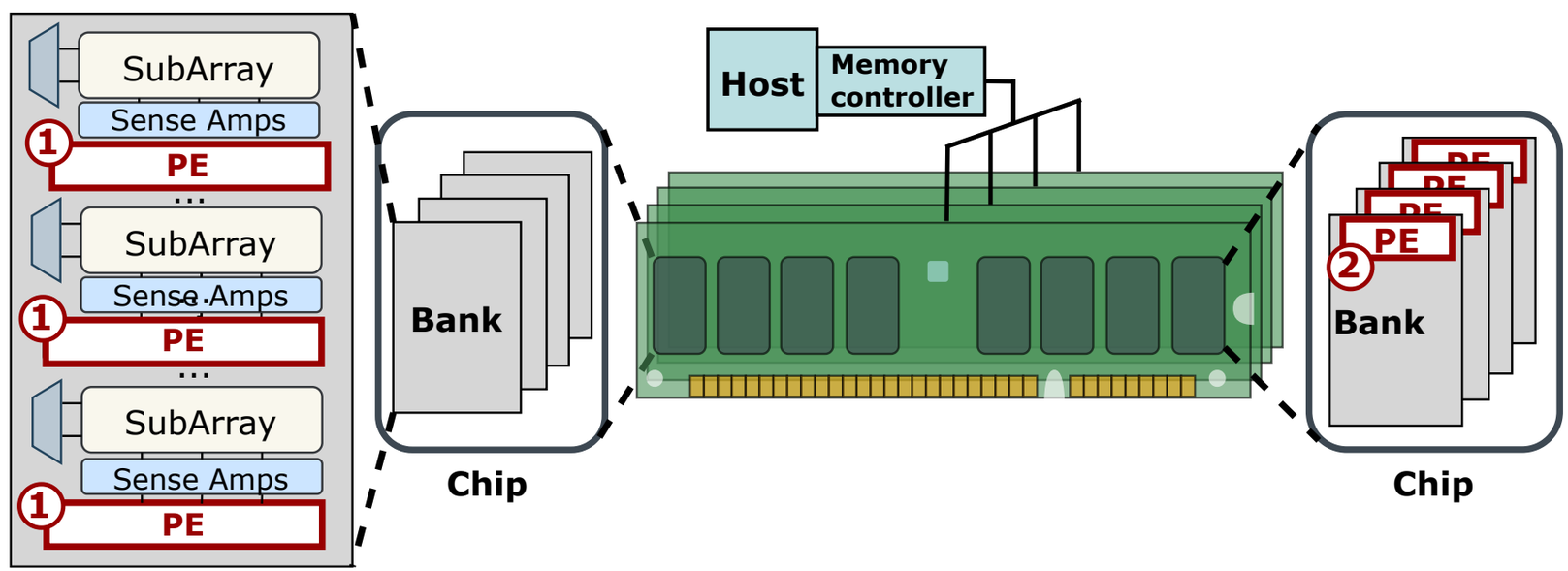
PIM vs. DRAM-based accelerator

- *Memory first* - PIM for commodity DRAM – “memory++”
  - Compatible with existing/near-term memory standards
  - But this limits design flexibility, e.g. regarding address interleaving, PIM logic overhead
  - Data layout and transformations may still require considerable data movement
    - Data movement/transformation can kill benefits of PIM
  - Allows incremental deployment
- *Accelerator-first* – DRAM as implementation technology for data-parallel accelerator
  - Can still be used as memory, but optimized for PIM
    - Allows much greater design flexibility: trade off capacity, maybe R/W perf for PIM
    - Can change interleaving, etc.
  - Still want to try to avoid copying data back and forth btw. main memory and PIM
  - Could be deployed in regular memory slots or CXL

Taxonomy of near-data/PIM in DRAM

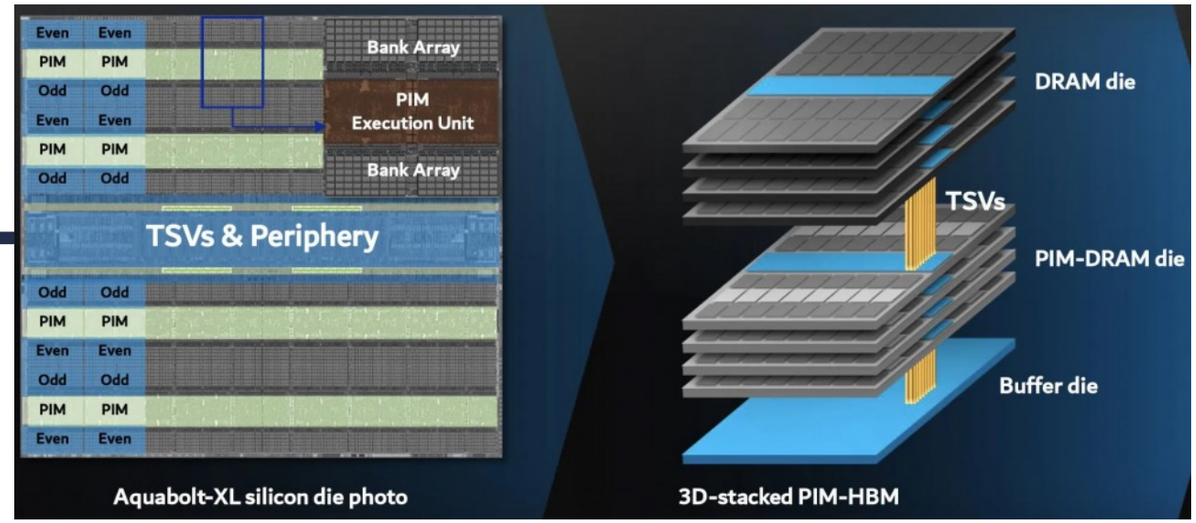
Closer to data

- In memory controller on processor chip
- In interface logic of DIMM or in logic layer of 3D stack
- Bank-level
- Subarray-level
- On the bitlines

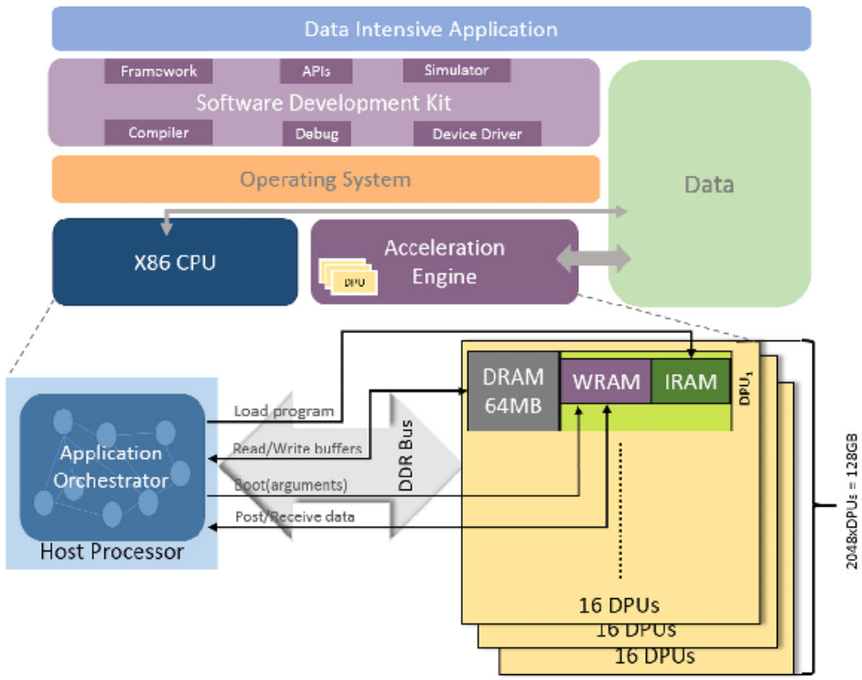


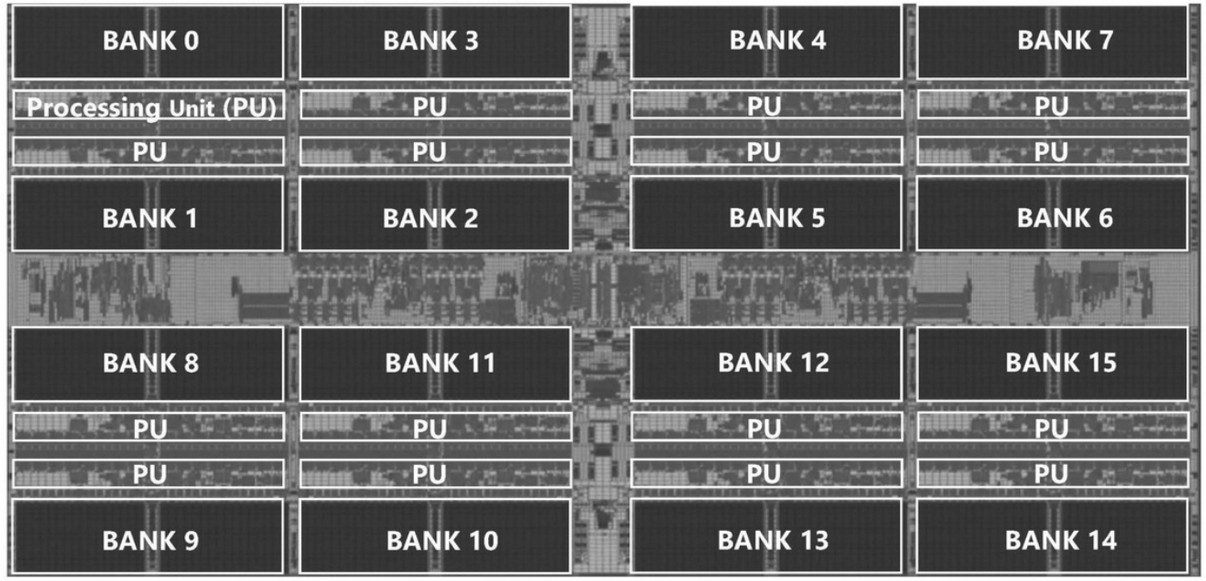
Notable early PIM products–all bank level

- Samsung Aquabolt
- SK Hynix AiM
- UPMEM



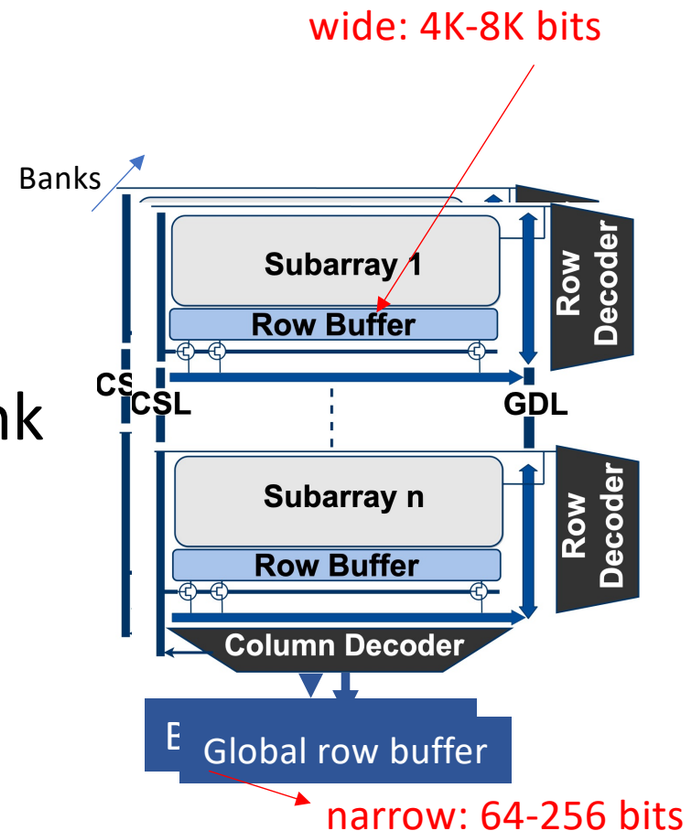
(all images taken from vendor information on the Internet)





Why subarray level ("in-situ")?

- Keeps compute very close to data – very low energy, minimizes long global wires
- Can operate on entire row buffer (~4-8 Kbits) between each DRAM row access
  - Logic ops can happen much faster than row read/write
- Potential for parallelism across subarrays + banks
- Beneficial even if only added to one subarray per bank
- Academic work has shown potential for both
  - Analog - requires triple row activation and other changes, but little/no circuitry at subarray row buffer
  - Digital - uses traditional read/write, places some gates at interface to row buffer
    - Can be 1) bit-serial, row-parallel or 2) scalar, bit-parallel

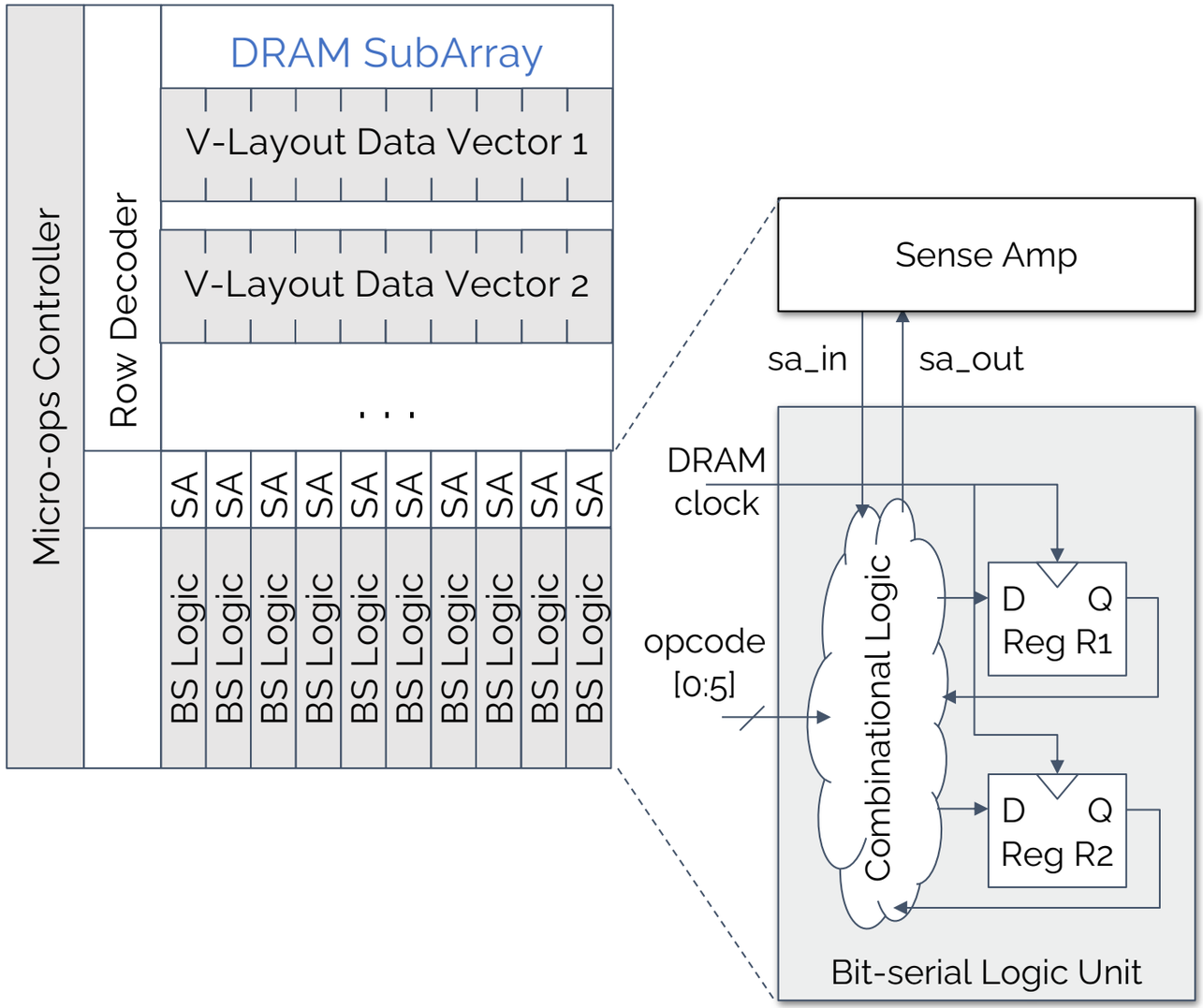


DRAM-AP: General bit-serial Associative Processing

(IISWC'24)

- Vertical data layout (so far)
- Bit-serial logic in subarrays
  - With a few 1-bit registers
  - Only need a few primitives to obtain considerable speedup
- Vector computing model
- Programmable micro-ops

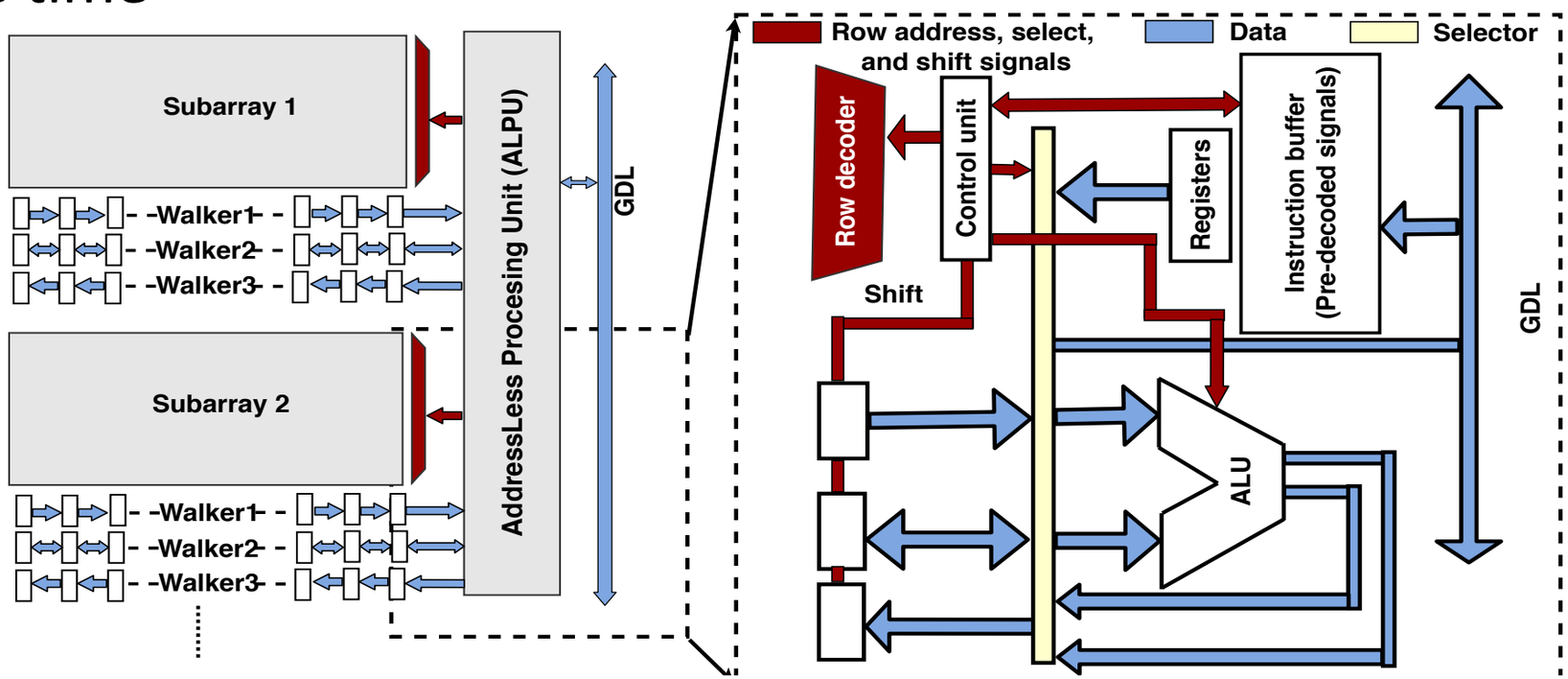
- Working on an approach that minimizes changes to data layout



Fulcrum: Flexible, word-granularity, subarray-level PIM

[HPCA'20, in collaboration with UCSB and Micron]

- Bit-serial: no support for data dependencies, conditional operations, sparse access, etc.; high overhead for multiplication
- Key insights: 1) leverage row-wide fetch but avoid high area cost and functional limitations of row-wide logic; 2) leverage faster column cycle time vs. row cycle time
- Fulcrum solutions:
  - Word-granularity ALU
  - Shift-based access
    - Row buffer→walker
  - Simple controller
    - Issuing shifts based on a predicate
    - Programmable, semi-general-purpose



Low-hanging fruit: database table scans for OLAP

(joint w/ Jignesh Patel, CMU and José Martínez, Cornell, and our students: Akhil Shekar, Lingxi Wu, Kevin Gaffney, Martin Prammer, and Helena Caminal)

- Online analytics processing (OLAP) - huge market for potential accelerator
  - eg, Google found that BigQuery, consumed about 10% total cycles within the fleet
- Memory bound: typical query scans all records yet selects only a small subset
  - CPU fetches all records into CPU, mostly performs only simple/range comparisons
  - Then throws away most of them, wasting lots of time and energy

Find the top 10 product categories most popular with customers in their 20s

```
SELECT
  p_category,
  SUM(o_quantity) AS popularity
FROM order, product, customer
WHERE o_p_key = p_key
  AND o_c_key = c_key
  AND c_age BETWEEN 20 AND 29
GROUP BY p_category
ORDER BY popularity DESC
LIMIT 10;
```

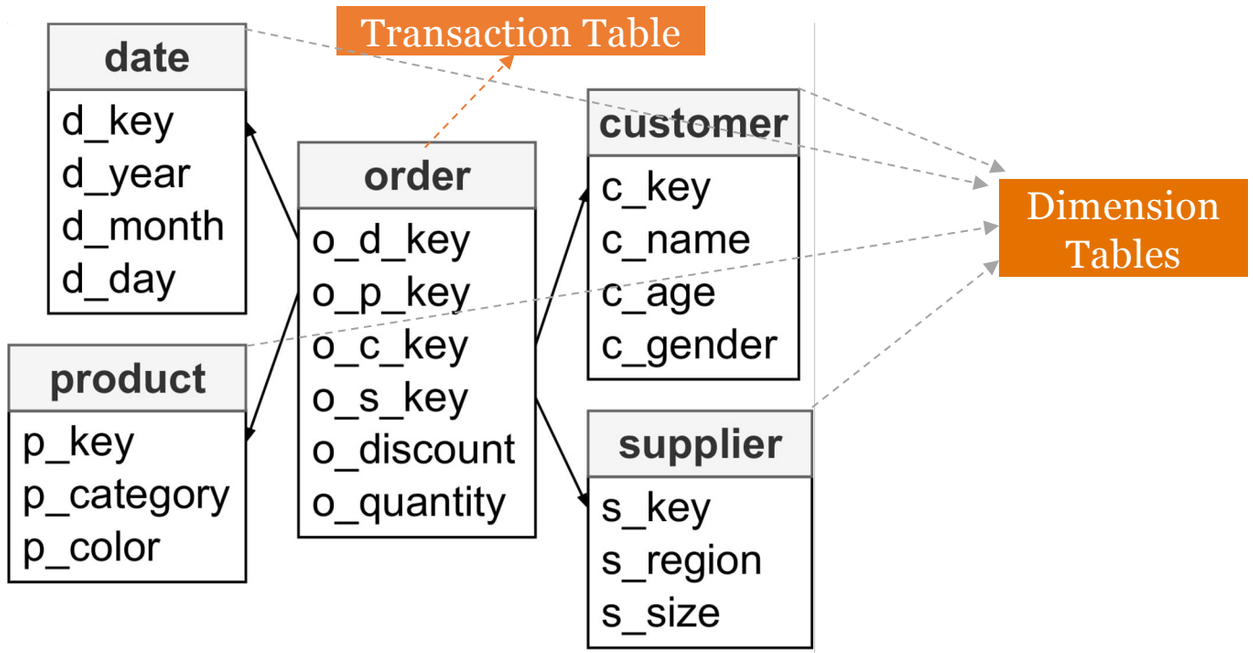
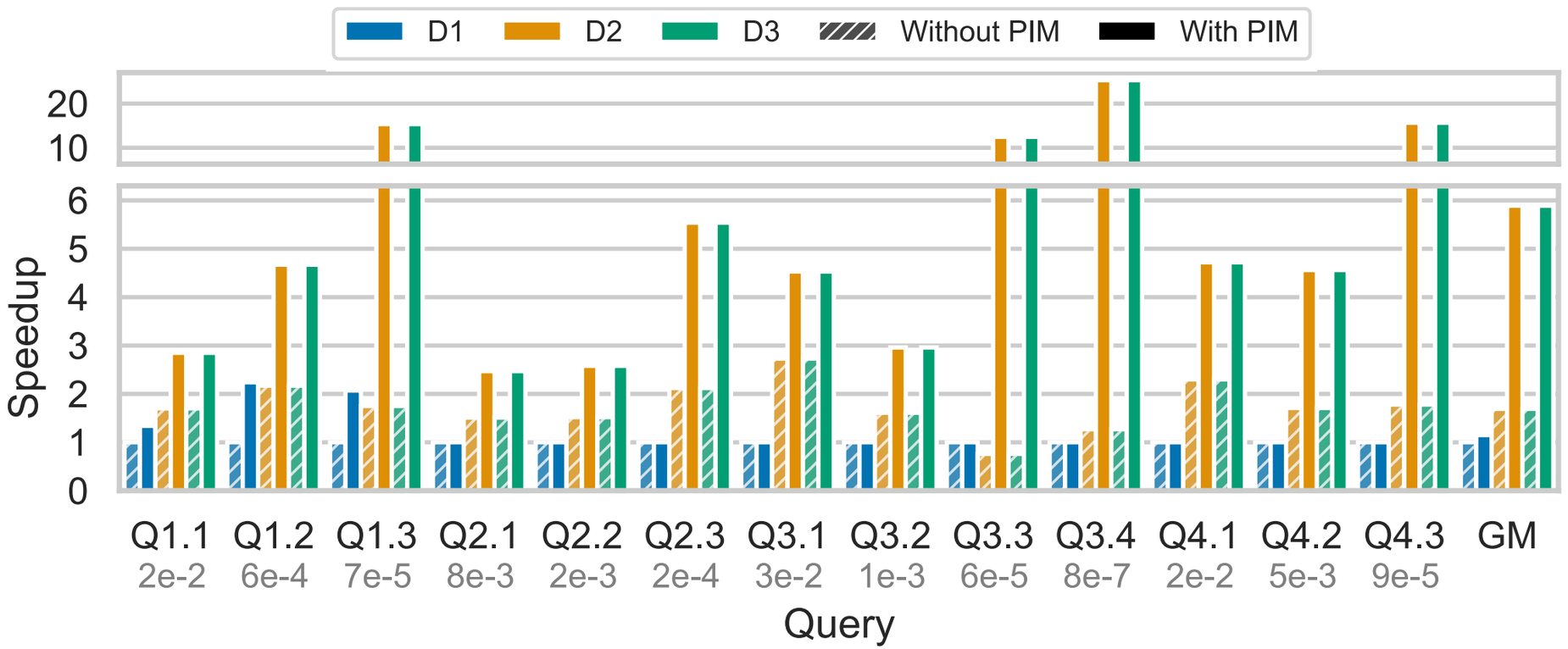


Table Scan: Apply **selection predicates** on the dimension table for the "WHERE" clause

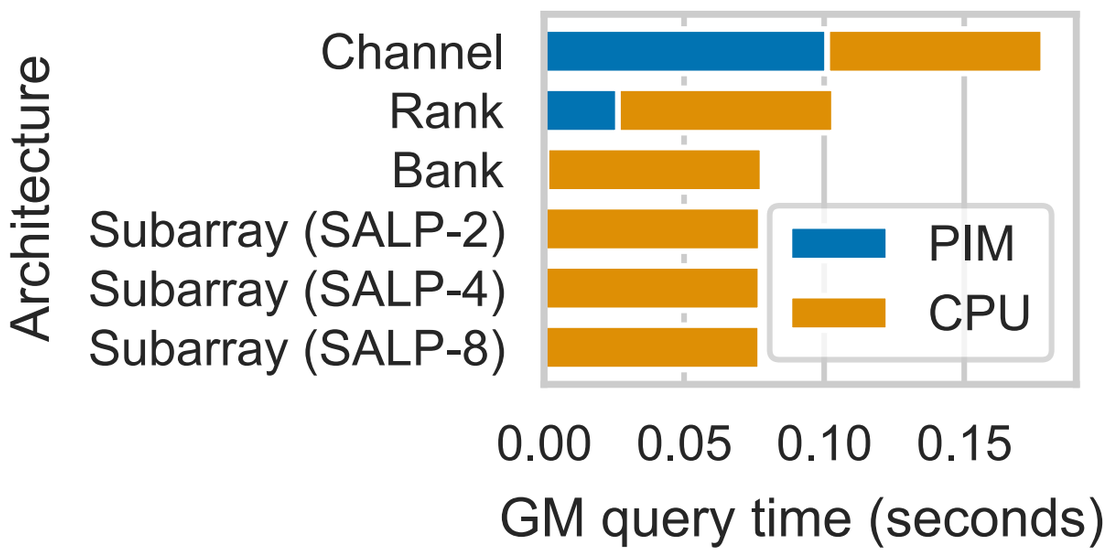
Key insight: convert joins to table scan

- Denormalization: joins are expensive, so prejoin everything, create one WideTable
  - Used in some industry products
- Denormalization yields almost 2X speedup in DuckDB
- PIM yields almost 4X further speedup



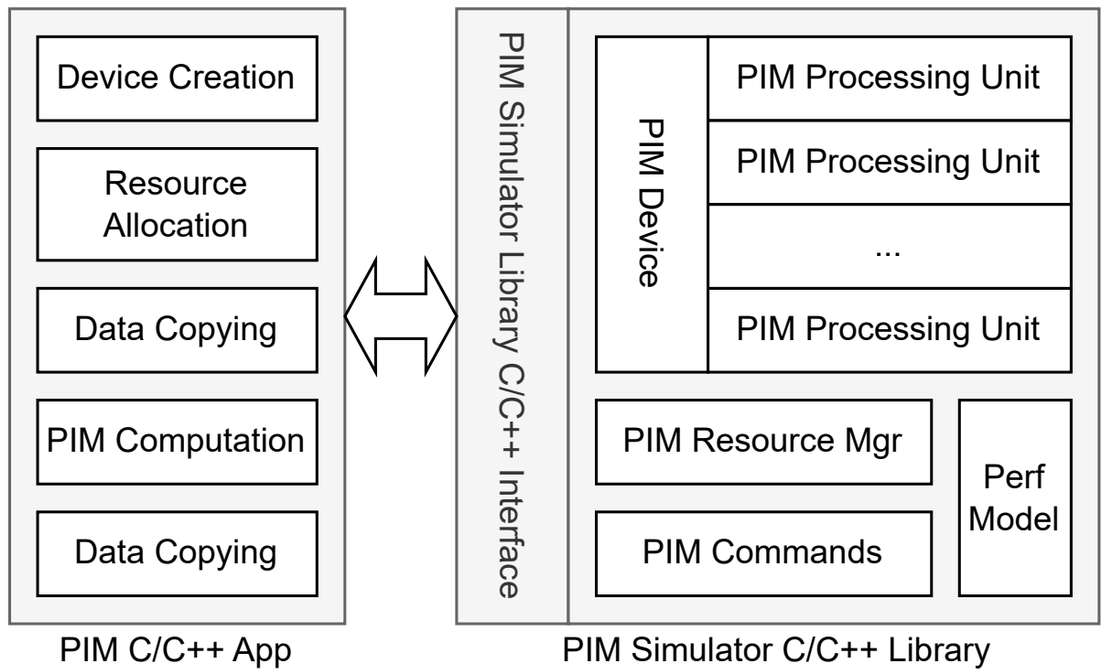
Data analytics: Bank-level PIM sufficient!

- Even with bank-level PIM, bottleneck is fetching selected records
- More aggressive PIM doesn't help
- Bank-level unit can be tiny
- Just equality and range checki



Evaluating PIM design space (IISWC'24)

- PIMeval simulator
- Just released
- Abstract model of PIM processing unit
- Can be integrated anywhere in DRAM hierarchy
- Can specify PIM-PU functional capabilities
- Also models data allocation, copying, etc.
- Performs both functional and performance modeling
  - Performance model derived from DRAMsim3



PIMbench benchmark suite (continuing to add new benchmarks)

Domain	Application Name	Memory Access Pattern		Execution Type
		Sequential	Random	
Linear Algebra	Vector Addition	✓		PIM
	AXPY	✓		PIM
	Matrix-Vector Mult. (GEMV)	✓		PIM
	Matrix-Matrix Mult. (GEMM)	✓		PIM
Sort	Radix Sort	✓	✓	PIM + Host
Cryptography	AES-Encryption	✓	✓	PIM
	AES-Decryption	✓	✓	PIM
Graph	Triangle Count	✓	✓	PIM
Database	Filter-By-Key	✓		PIM + Host
Image Processing	Histogram	✓		PIM
	Brightness	✓		PIM
	Image Down Sampling	✓		PIM
Supervised Learning	K-nearest neighbors (KNN)	✓	✓	PIM + Host
	Linear Regression	✓		PIM
Unsupervised Learning	K-means	✓	✓	PIM
Neural Network	VGG-13	✓		PIM + Host
	VGG-16	✓		PIM + Host
	VGG-19	✓		PIM + Host

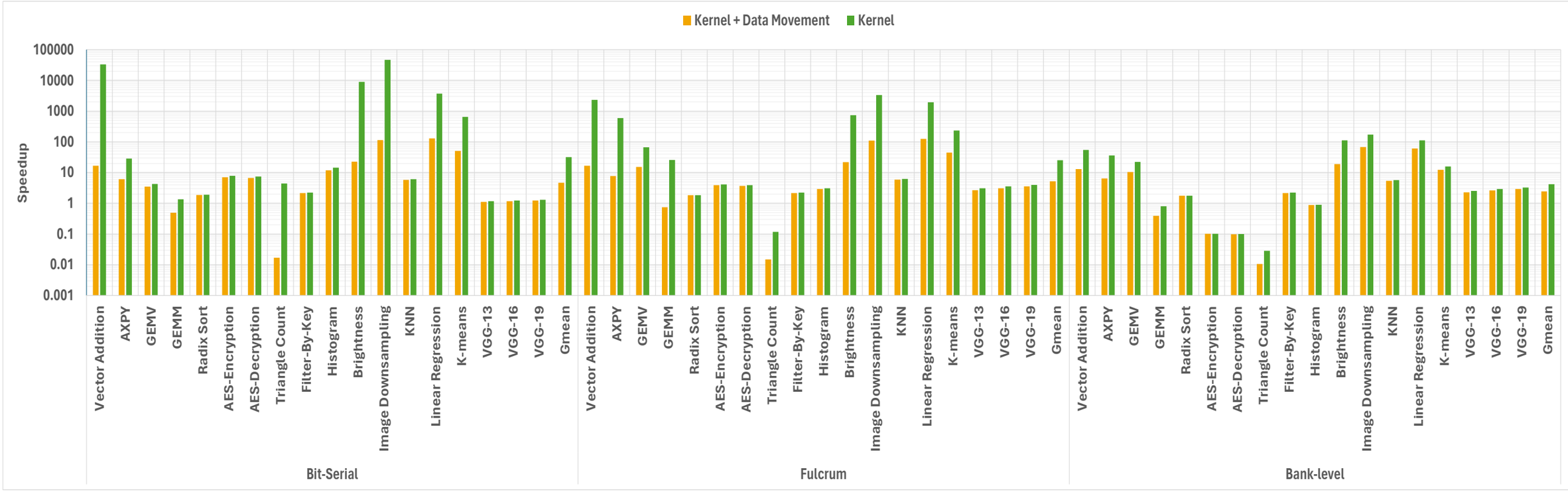
PIM API

- Designed an API for writing PIM programs that is portable across PIM architectures

```
void axpy(uint64_t vectorLength, const vector<int> &X, const vector<int> &Y, int A)
        {
  unsigned bitsPerElement = sizeof(int) * 8;
  // Allocate device memory
  PimObjId objX = pimAlloc(PIM_ALLOC_AUTO, vectorLength, bitsPerElement, PIM_INT32)
      ;
  PimObjId objY = pimAllocAssociated(bitsPerElement, objX, PIM_INT32);
  assert((objX != -1) && (objY != -1));
  // Copy inputs, perform operations, copy back results
  pimCopyHostToDevice(X.data(), objX);
  pimCopyHostToDevice(Y.data(), objY);
  pimScaledAdd(objX, objY, objY, A)
  pimCopyDeviceToHost(objY, Y.data());
  // Free allocated memory
  pimFree(objX);
  pimFree(objY);
}
```

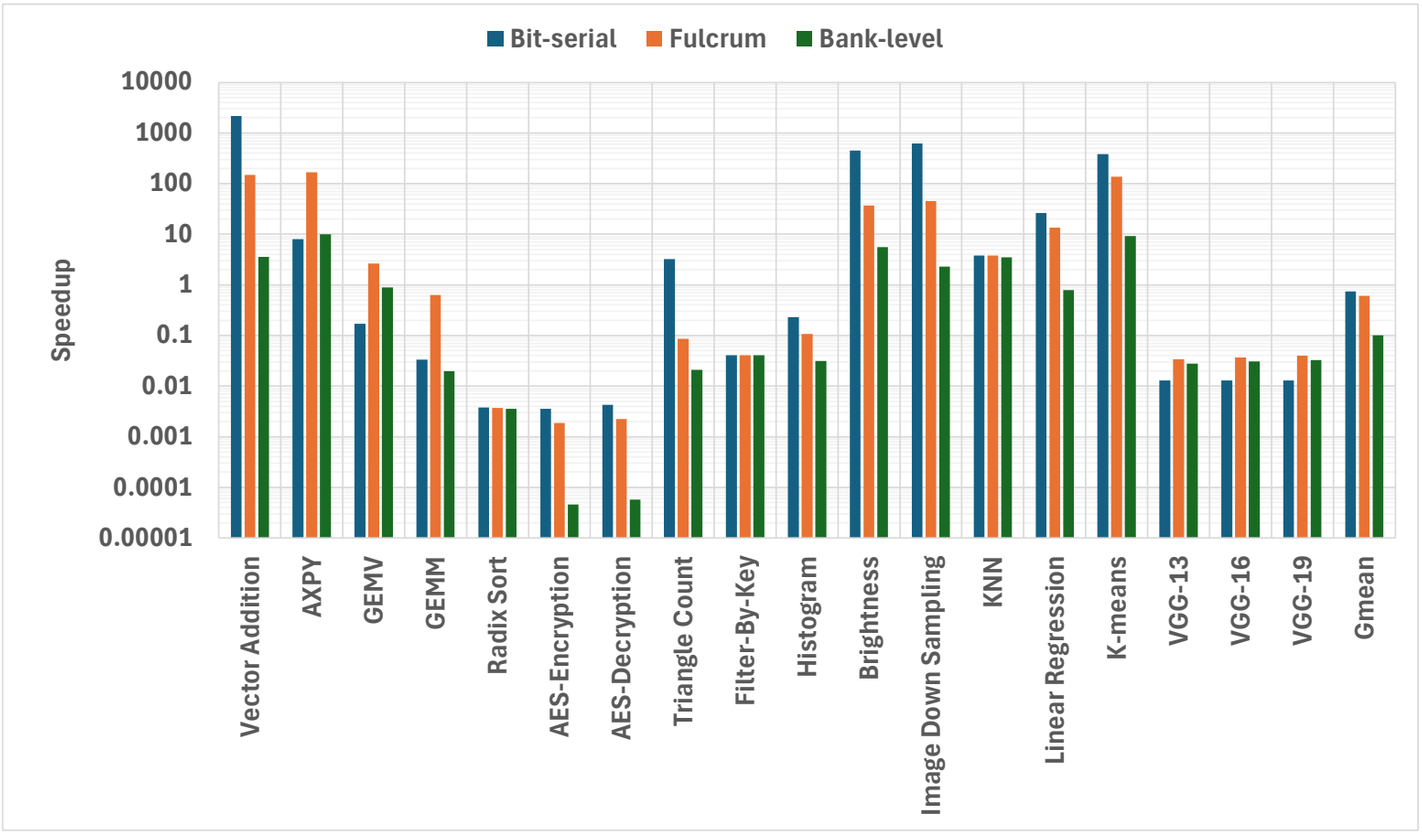
PIM architecture comparison – vs CPU

- Fulcrum is scalar, bank-level is 128-bit SIMD
- Assumes data must be copied into PIM memory—significant overhead
- Bit-serial fastest on Boolean, addition
- Fulcrum fastest on everything else
- Bank-level limited by narrow bank interface



PIM architecture comparison-GPU

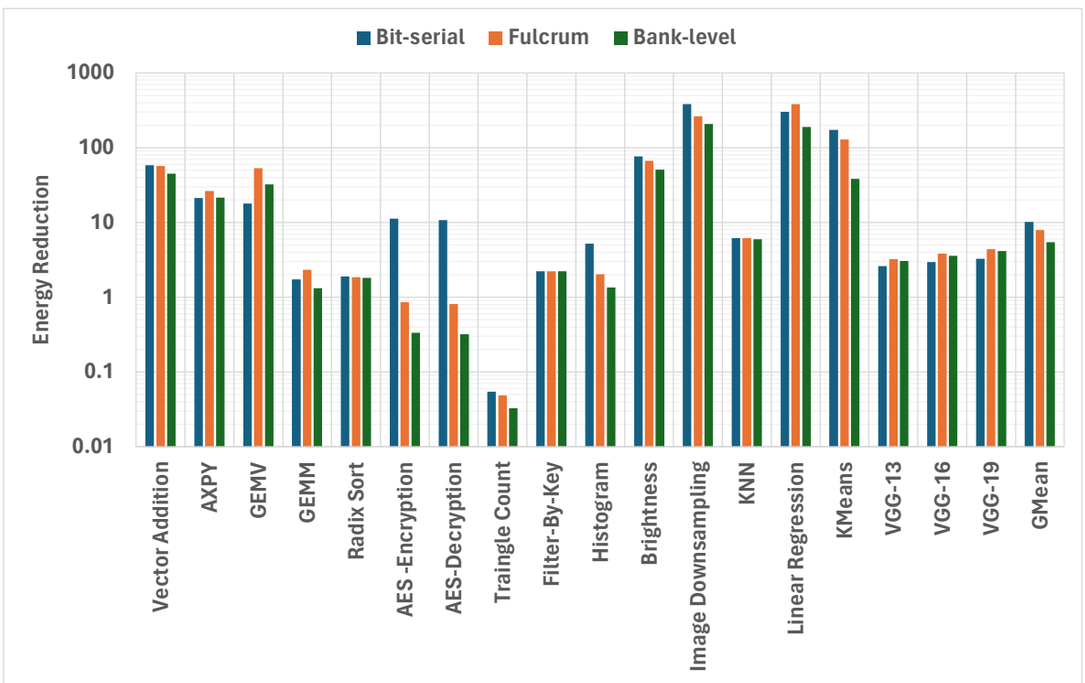
- DDR-based PIM has difficulty competing with A100 GPU
- Takes many ranks to match bandwidth of GPU with 4 stacks of HBM
- Note GEMV result here is flawed – data too small

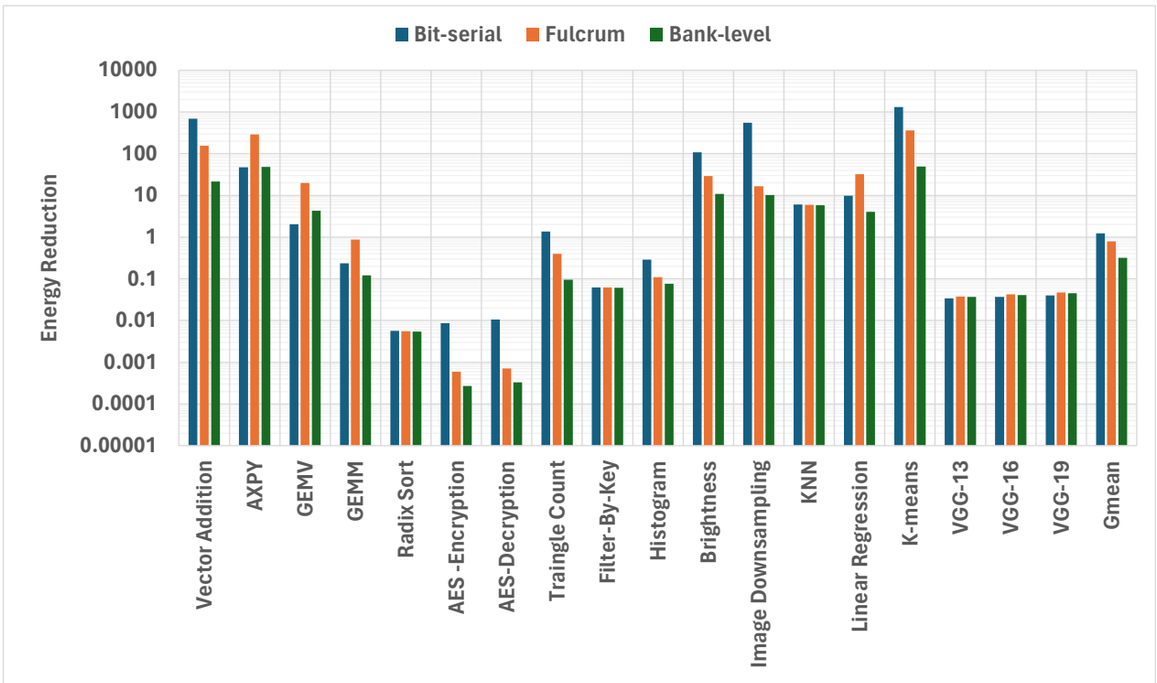


32 ranks

But PIM has significant energy efficiency benefits

- Energy efficiency win for most benchmarks vs. CPU
- Also beats GPU on subset of benchmarks





Observations and Open issues

- Software ecosystem and portability are key to success
  - Good programming model and libraries were key to GPU successes
- Not yet clear which of these PIM architectures is most promising
  - Want a design that combines benefits of each

- Exploring how to integrate with gem5 for full-system simulation
- System integration is hard
  - How to communicate with CPU? (Eg, how does a PIM program run?)
  - Memory allocation (PIM requires specific data layout, esp. for subarray-level)
  - Memory consistency (SC, TSO, etc. – what is contract with PIM programmer?)
  - Cache coherence
  - Multi-tenancy
  - Etc.
- Debugging support is also needed for practical adoption

Concluding thoughts

- A “tools-first” research agenda is rewarding
  - Enables fun research and the tools have external impact
- To have external impact, tools need to be highly flexible and easy to use
  - Other researchers will use them in ways you never expected
- PIM shows promise to accelerate key compute tasks while saving energy
  - Data analytics and GEMV as potential early killer apps

UVA team effort!

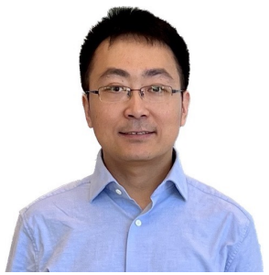

Alif Ahmed

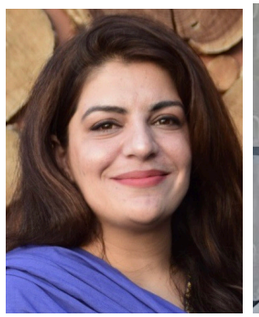

Hugo Abbot

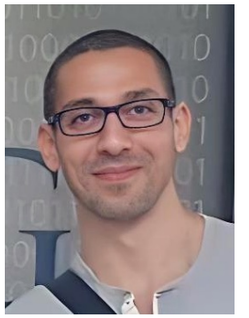

Kyle Durrer

Ethan Ermovick

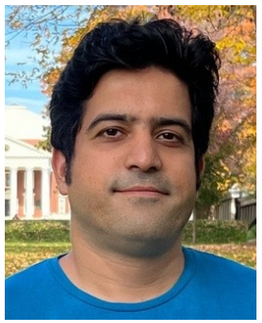

Kumaresh Nandagopal


Deyuan Guo


Beenish Gul


Abdullah Mughrabi


Marzieh Lenjani

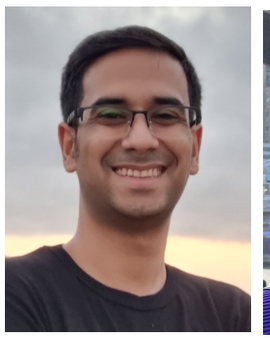

Morteza Baradaran

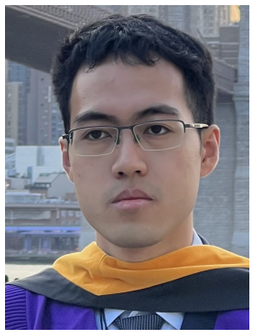

Sergiu Mosanu


Akhil Shekar

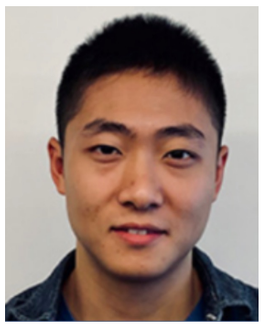

Farzana Siddique


Khyati Kiyawat


Mohammadhosein Gholamrezaei


Zhenxing Xi


Mircea Stan


Lingxi Wu


Ashish Venkat