ILLINOIS TECH

High Performance Computing with Emerging Memory Architectures

Rujia Wang

2022/03/23

About me

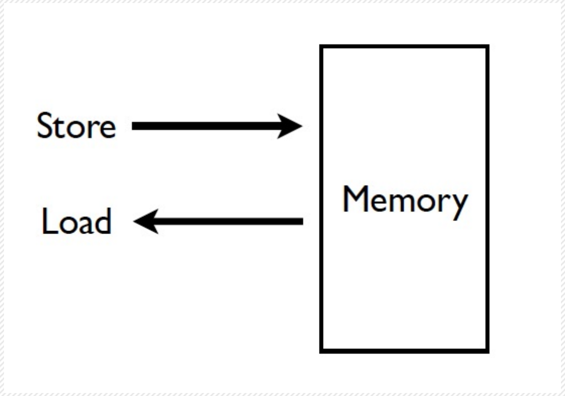
Rujia Wang - Assistant Professor @ Illinois Tech CS

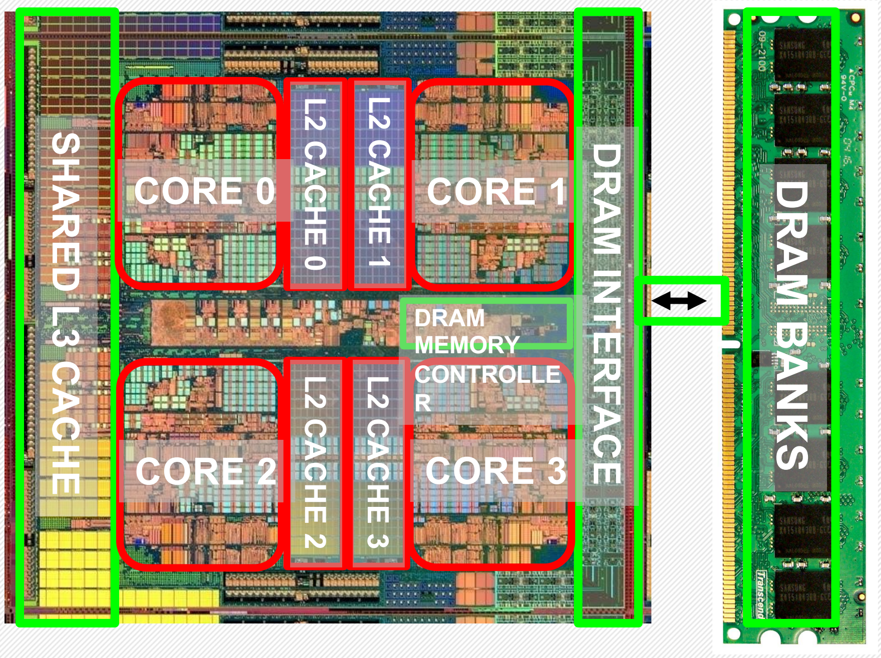
- https://rujiawang.github.io/
- rwang67@iit.edu

Research interest:

- Computer architecture
- Memory systems
- Emerging memory technologies
- Architectural support for security and privacy

An overview on current memory system





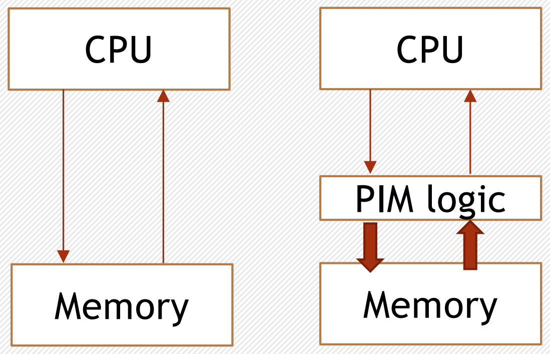
Challenges of current memory system

- **Performance**
  - Memory technology(e.g., DRAM) scaling is much more difficult than the processor
  - Memory latency has not been changed much
  - Memory bandwidth becomes the major performance bottleneck
- **Energy Consumption**
  - Frequent accessing the memory could cause high energy consumption
  - A memory access consumes ~1000x the energy of a complex ALU addition[Dally, HiPEAC 2015]
- **Heterogeneity**
  - Heterogeneity exists across multi-tier memory system
  - Data placement and management could significantly impact the overall performance

Process-in-memory(PIM) paradigm

- Instead of processing in the CPU, we can process tasks in the memory instead
  - -> process in memory
  - -> incorporate logic/cores in memory banks or on DIMMs

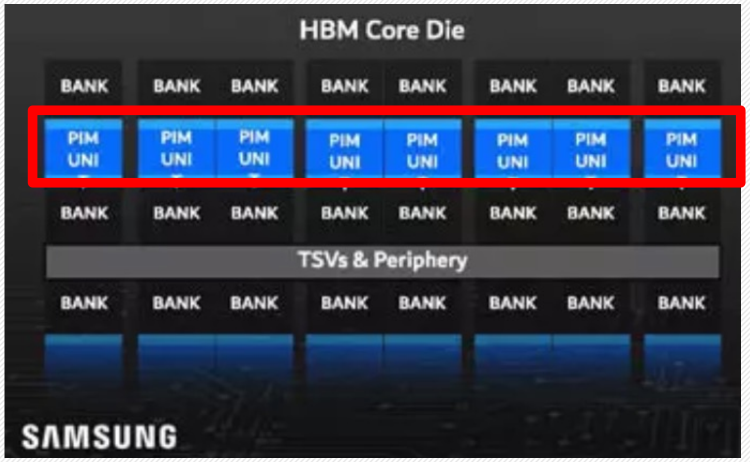
- Benefits of PIM paradigm:
  - compute closer to data
  - less data movement on bus
  - mitigates memory bandwidth and latency bottleneck
  - improves the energy efficiency



Von Neumann Arch. v.s. PIM Arch.

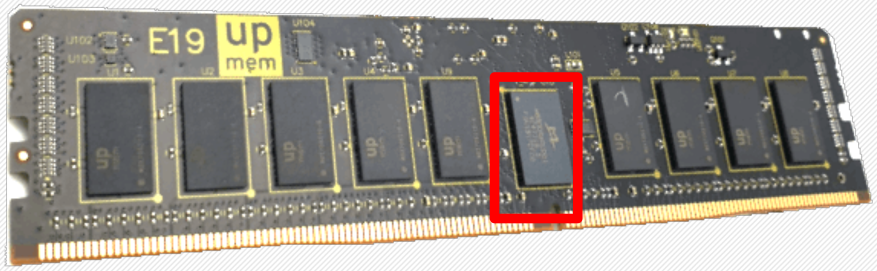
Process-in-memory(PIM) prototypes

- **Samsung HBM-PIM, 2021**



128 PIM units per HBM @ 300MHz
Each PIM unit has 16 MUL/ADD FPUs,
~1TB/s compute bandwidth [ISCA'21]

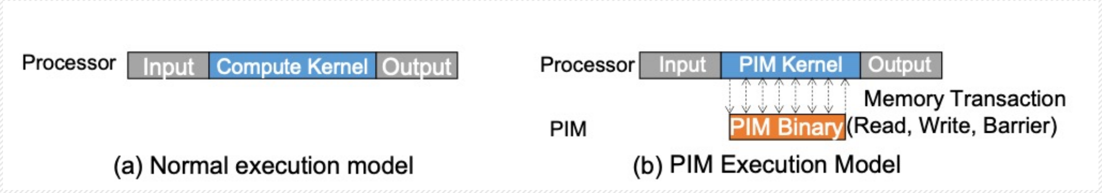
- **UPMEM DIMM, 2020**



128 PIM DPUs per DIMM @450MHz
The DPU is a 24 threads, 32b RISC processor,
each compute at 1GB/s BW (2.56TB/s for 2560 DPUs)
[HOTCHIPS'19]

PIM Computation Model

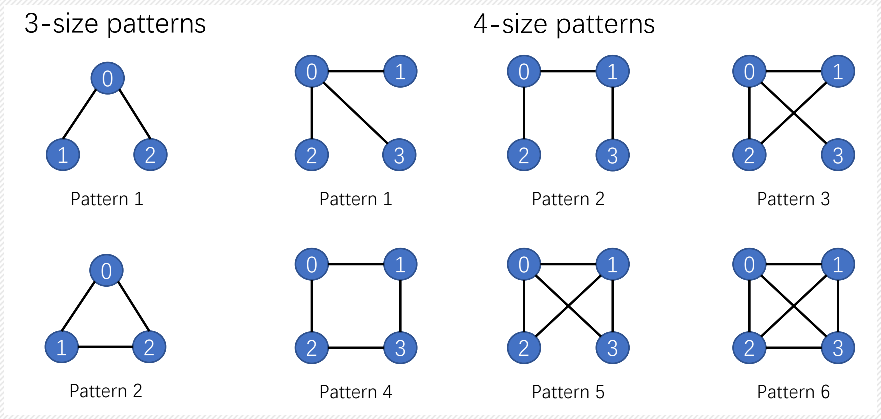
- Use PIM as a co-processor
  - Offloading tasks from host CPU to the PIM cores
  - Similar to the CPU-GPU computation model, but requires NO data movement on external bus/links



- Questions to be answered..
  - What/when to offload?
  - Offload granularity?
  - Best strategy to layout data?
  - Best strategy to schedule workloads on PIM cores?

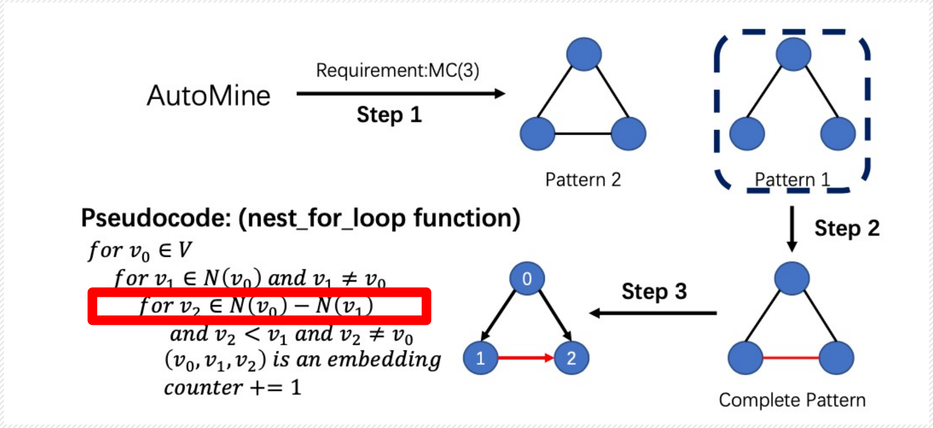
Use PIM to accelerate data mining workloads

- **Graph pattern mining** (GPMI) needs to generate patterns and do pattern matching according to the requirements of the application.
- Frequently used in data mining domains like bioinformatics, chemical reactions, social networks, etc.
- E.g., **Motif counting** (MC) is to identify all motifs (patterns) with *k* vertices and count the embeddings of each of the patterns.



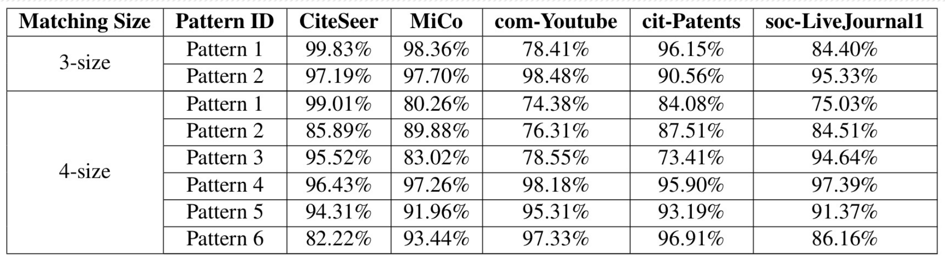
Use PIM to accelerate data mining workloads

- **Core computation kernel:**
  - Intensive vertex neighbor list Intersection/Subtraction
- Very time consuming and memory intensive, may be a good candidate for PIM acceleration



Pseudocode: (nest_for_loop function)

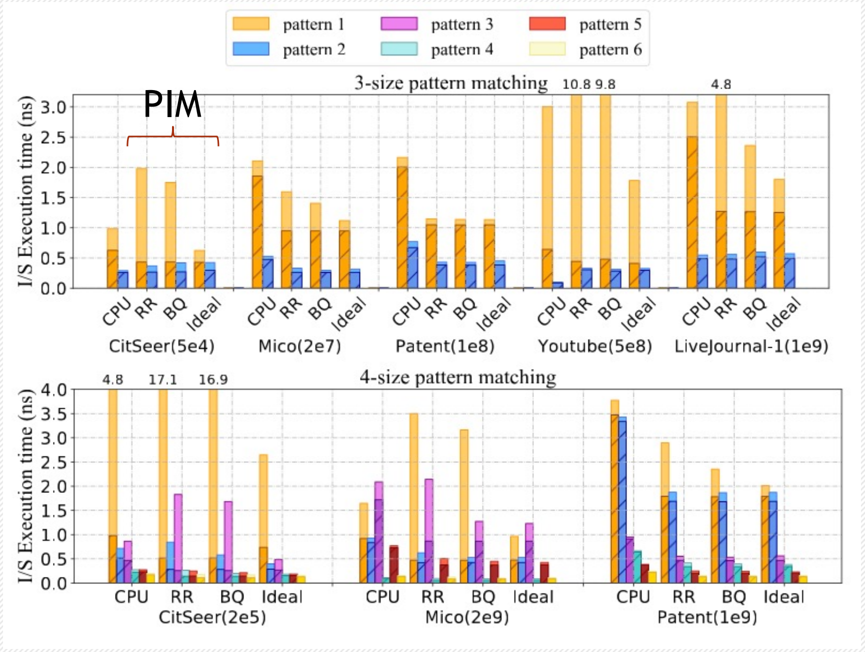
```
$for\ v_0 \in V$
    $for\ v_1 \in N(v_0)\ and\ v_1 \neq v_0$
        $for\ v_2 \in N(v_0) - N(v_1)$
            $and\ v_2 < v_1\ and\ v_2 \neq v_0$
            $(v_0, v_1, v_2)\ is\ an\ embedding$
            $counter\ += 1$
```



Matching Size	Pattern ID	CiteSeer	MiCo	com-Youtube	cit-Patents	soc-LiveJournal1
3-size	Pattern 1	99.83%	98.36%	78.41%	96.15%	84.40%
	Pattern 2	97.19%	97.70%	98.48%	90.56%	95.33%
4-size	Pattern 1	99.01%	80.26%	74.38%	84.08%	75.03%
	Pattern 2	85.89%	89.88%	76.31%	87.51%	84.51%
	Pattern 3	95.52%	83.02%	78.55%	73.41%	94.64%
	Pattern 4	96.43%	97.26%	98.18%	95.90%	97.39%
	Pattern 5	94.31%	91.96%	95.31%	93.19%	91.37%
	Pattern 6	82.22%	93.44%	97.33%	96.91%	86.16%

Challenges

- Simply offload the I/S computation kernel in GPMI cannot fully utilize the advantages of PIM hardware
- The PIM execution time is close to or longer than CPU execution time with the same core number
- Issues that we identify:
  - Load imbalance
  - Suboptimal data mapping
  - Locality
  - PIM internal heterogeneity



Key techniques

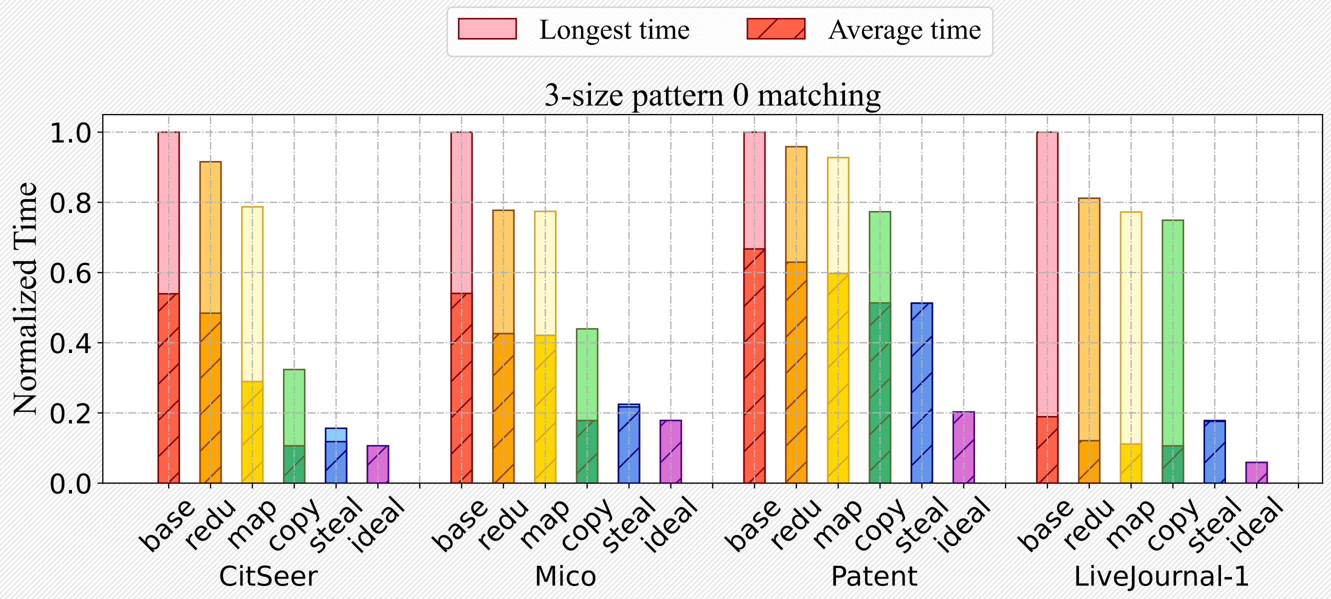
Reduce unnecessary data accesses from remote banks

Locality-aware address **mapping** to leverage near-bank acceleration

Critical data **copy** near PIM cores

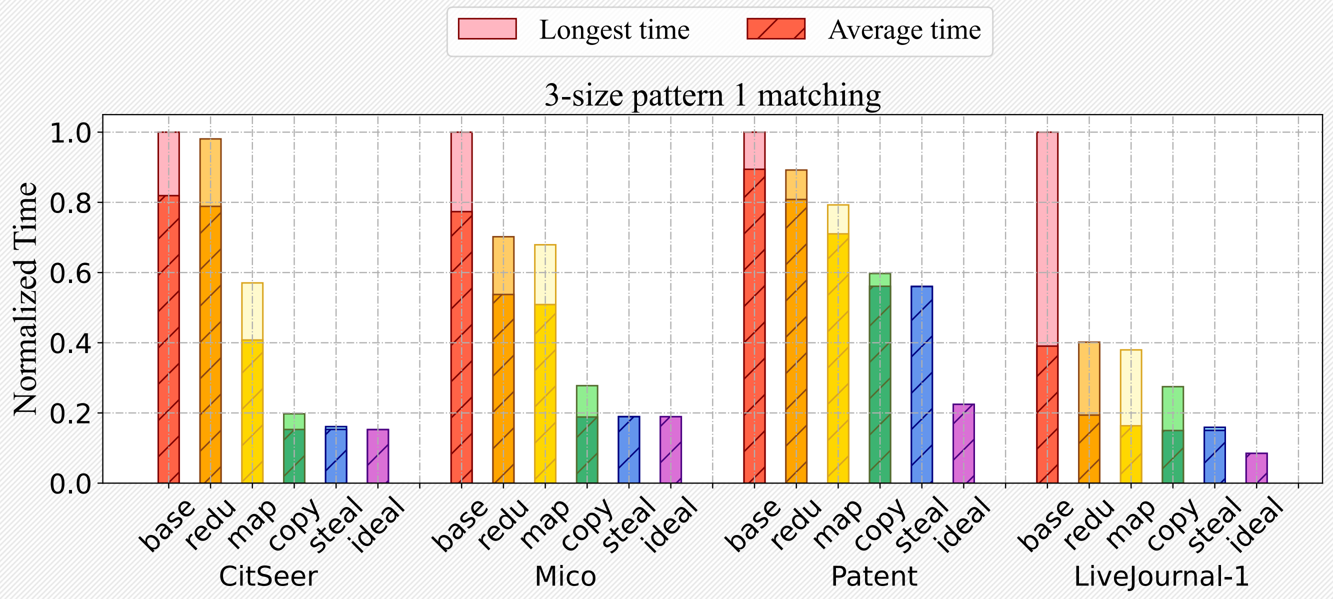
Lightweight workload **stealing** to balance computation across cores

PIM architectural aware GPMI acceleration



1.8x-5.5x speedup compared to baseline

PIM architectural aware GPMI acceleration



1.8x-6.7x speedup compared to baseline

Some takeaways

- PIM has the potential to accelerate many memory-intensive workloads
- PIM cores can outperform CPU and GPU by leveraging the internal memory bandwidth with parallel computing
- Accelerating irregular workloads (e.g., graph applications) requires full-stack co-designs
- There is a lack of a holistic management framework for PIM-assisted systems due to the changing workloads and the missing abstraction of PIM hardware
- Current in collaboration with IIT CS (Kyle Hale, Xian-He Sun), ECE (Ken Choi) and other universities faculties to work on these challenges

Open collaborative research topics

PIM for other data-intensive workloads

- HPC and scientific computing workloads
- Graph neural network

PIM at a scale

- Integration with HPC systems and components
- runtime resource management, e.g., inter-PIM communication
- Programming interface

Thank you – Questions?

Rujia Wang - rwang67@iit.edu - https://rujiawang.github.io/