The Challenges and Opportunities of Processing-in-Memory:

A performance point of view

Xian-He Sun
Illinois Institute of Technology
sun@iit.edu

Pilot Talk at NSF PIMT Workshop
January 7, 2021

PIM: Challenges and Opportunities

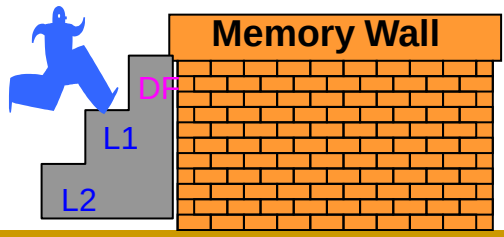
- Big Data
- *High Performance and Could Computing*
- Intelligent Computing (AI and Deep Learning)

(Computing) Performance

Data Access Performance

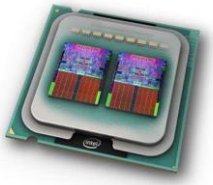
Integrated Compute-Data Performance

(PIM)



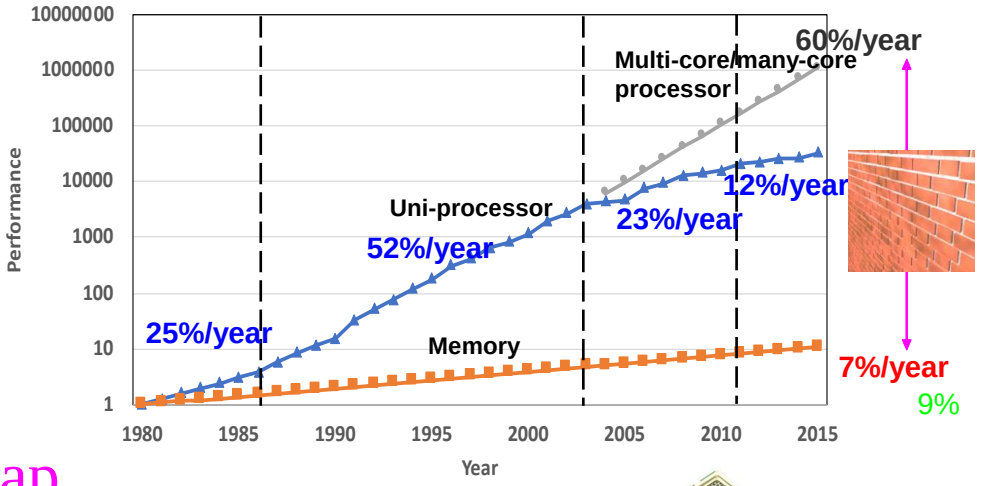
Why Bottleneck? The Memory-wall Problem

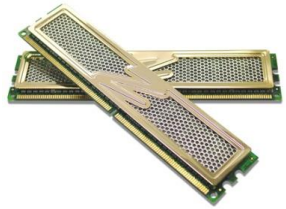
- Processor performance increases rapidly
  - Uni-processor: ~52% until 2004
  - Aggregate multi-core/many-core processor performance even higher since 2004
- Memory: ~7% per year
  - Storage: ~6% per year
- Processor-memory speed gap keeps increasing



Source: Intel

Leiserson, Charles E., et al. "There's plenty of room at the Top: What will drive computer performance after Moore's law?." *Science* 368.6495 (2020).





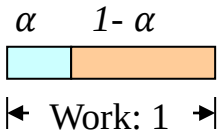
Source: OCZ

Memory-bounded speedup (1990), Memory wall problem (1994)

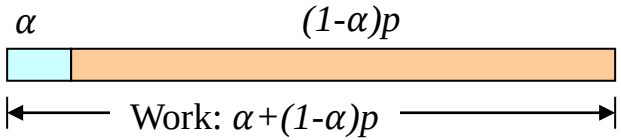
X.-H. Sun, and L. Ni, "Another View of Parallel Speedup," Proc. of ACM-IEEE Supercomputing'90, NY, Nov. 1990

The Three Laws

- Tacit assumption of Amdahl's law
  - Problem size is fixed
  - Speedup emphasizes on time reduction

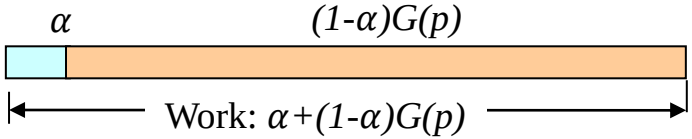


- Gustafson's Law, 1988
  - Fixed-time speedup model



$$Speedup_{fixed-time} = \frac{Sequential\ Time\ of\ Solving\ Scaled\ Workload}{Parallel\ Time\ of\ Solving\ Scaled\ Workload}$$
$$= \alpha + (1-\alpha)p$$

- Sun and Ni's law, 1990
  - Memory-bounded speedup model



$$Speedup_{memory-bounded} = \frac{Sequential\ Time\ of\ Solving\ Scaled\ Workload}{Parallel\ Time\ of\ Solving\ Scaled\ Workload}$$
$$= \frac{\alpha + (1-\alpha)G(p)}{\alpha + (1-\alpha)\,G(p)/p}$$

X.-H. Sun, and L. Ni, "Scalable Problems and Memory-Bounded Speedup," Journal of Parallel and Distributed Computing, Vol. 19, pp.27-37, Sept. 1993.

Implication of Memory-Bounded Model

- **W = G(M)** shows the trade-off between computing & memory
  - W, the work in floating point operation
  - M, the memory requirement
  - G, the data reuse rate
  - It is application/algorithm dependent

- **W = G(M)** unifies the models
  - G(p) = 1, Amdahl's law
  - G(p) = p, Gustafson's law

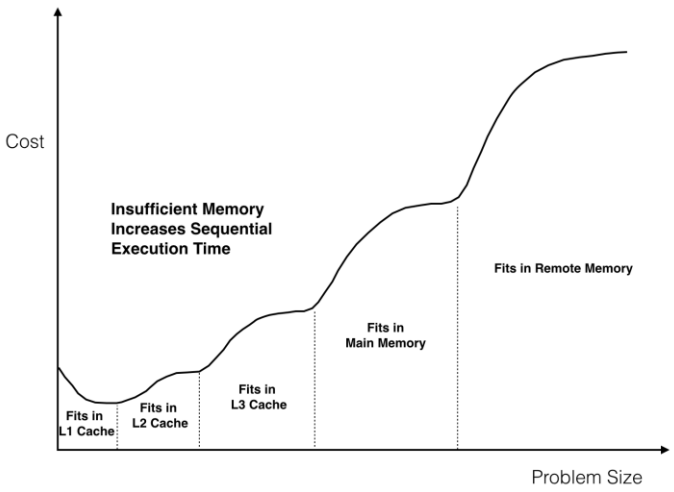
- Reveal memory is the performance bottleneck
  - In parallel processing, scalability, as well as sequential processing,
  - The ***Memory-Wall problem*** (1994)

X.H. Sun, and J. Gustafson, "Toward A Better Parallel Performance Metric," Parallel Computing, Vol. 17, pp.1093-1109, Dec. 1991.

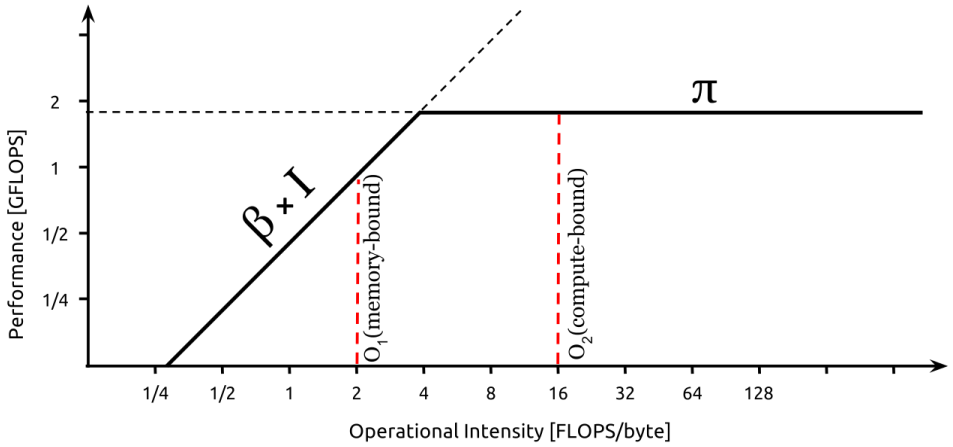
X.-H. Sun, and D. Rover, "Scalability of Parallel Algorithm-Machine Combinations," IEEE Trans. on Parallel and Distributed Systems, Vol.5, pp.599-613, June 1994.

Memory-Wall vs Memory-Bound

- Memory hierarchy is introduced to solve the memory-wall problem
- The bound of fast memory
- (sequential/parallel) Performance (speed) varies with problem size



Cost variation cross layers



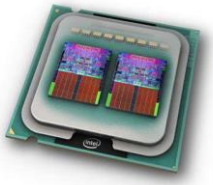
The Berkeley roofline Model

Memory bound analysis

X.-H. Sun, and J. Zhu, "Performance Prediction: A Case Study Using a Shared-Virtual-Memory Machine," IEEE Parallel & Distributed Technology, pp. 36-49, Winter 1996.

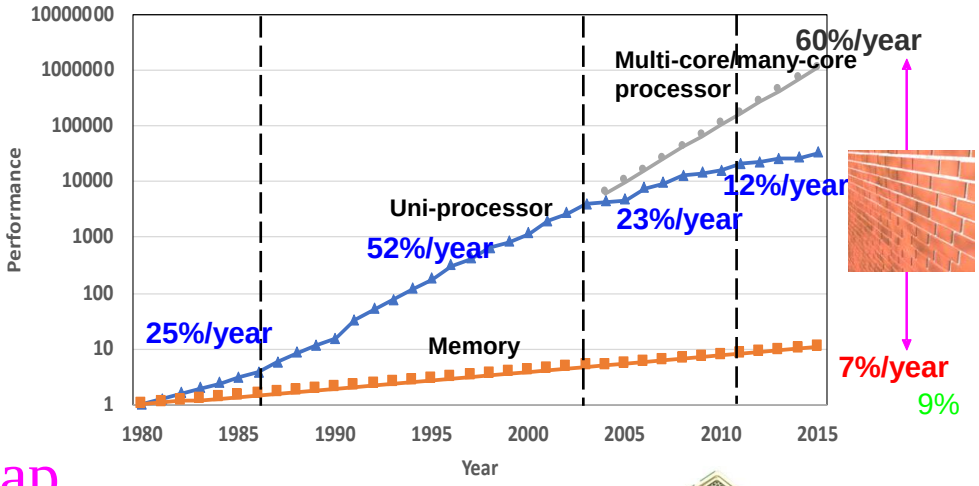
Why Bottleneck? The Memory-wall Problem

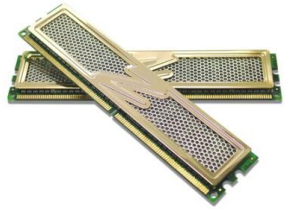
- Processor performance increases rapidly
  - Uni-processor: ~52% until 2004
  - Aggregate multi-core/many-core processor performance even higher since 2004
- Memory: ~7% per year
  - Storage: ~6% per year
- Processor-memory speed gap keeps increasing



Source: Intel

Leiserson, Charles E., et al. "There's plenty of room at the Top: What will drive computer performance after Moore's law?." *Science* 368.6495 (2020).





Source: OCZ

Multicore technology (2004), Big Data Initiative (2012)

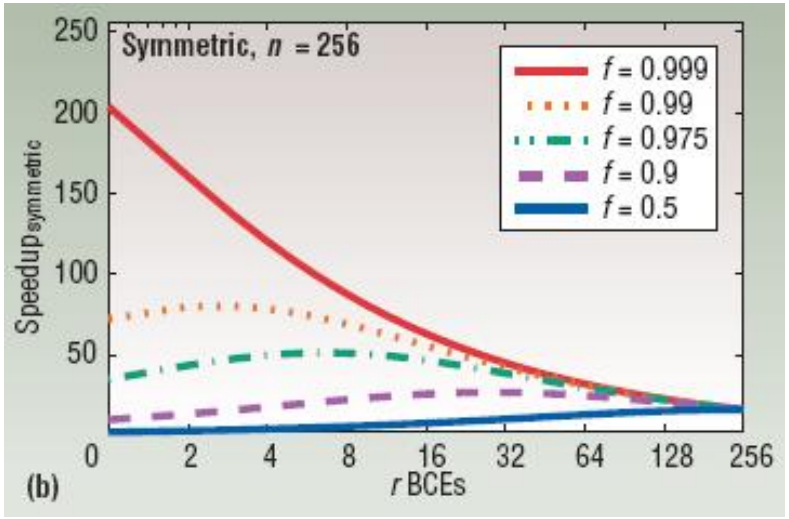
Amdahl's Law for Multicore (Hill&Marty08)

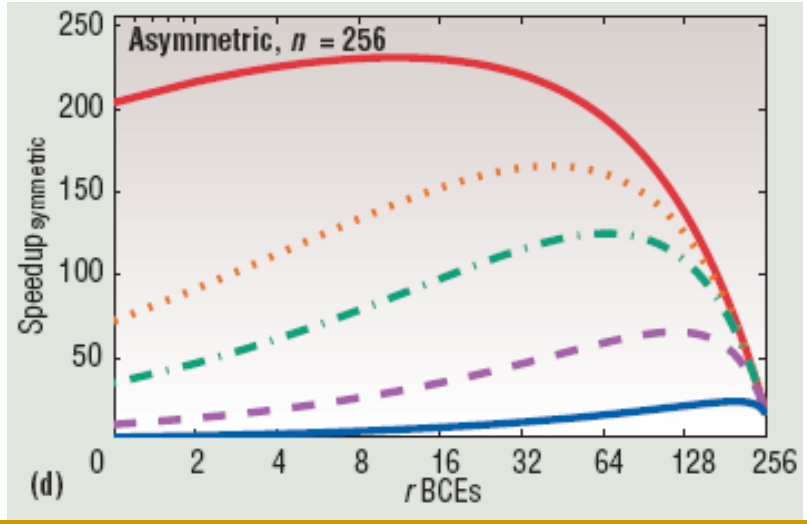
- Speedup of symmetric architecture

$$Speedup_{symmetric}(f,n,r) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f \cdot r}{perf(r) \cdot n}}$$

- Speedup of asymmetric architecture

$$Speedup_{asymmetric}(f,n,r) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{perf(r) + n - r}}$$





Scaled Speedup under Memory-wall

- Assuming perfect parallel, data access time, w_c , is the constraint

$$\frac{w_c}{perf(r)} + \frac{w_p}{perf(r)} = \frac{w_c}{perf(r)} + \frac{w_p'}{m \cdot perf(r)} \quad => \quad w_p' = m w_p$$

- Fixed-time speedup

$$\frac{\frac{w_c}{perf(r)} + \frac{w_p'}{m \cdot perf(r)}}{\frac{w_c}{perf(r)} + \frac{w_p}{perf(r)}} = \frac{w_c + m \cdot w_p}{w_c + w_p} = (1 - f') + mf'$$

$$f' = \frac{w_p}{w_c + w_p}$$

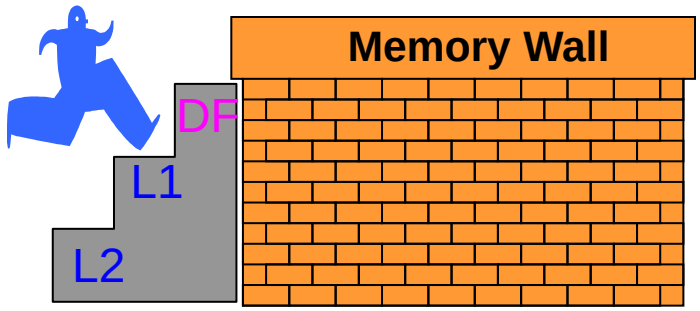
- Memory-bounded speedup
  - With $g(m) = 0.38m^{3/2}$ memory-bounded speedup is bigger than fixed-time speedup
  - *g(m)* equals one, memory-bounded is the as fixed-size, *g(m)* equals *m*, then memory-bound is the same as fixed-time

X.-H. Sun, Y. Chen. "Reevaluating Amdahl's Law in the Multicore Era," Journal of Parallel and Distributed Computing, Vol. 70, No. 2, pp183-188, 2010. (conf. 08)

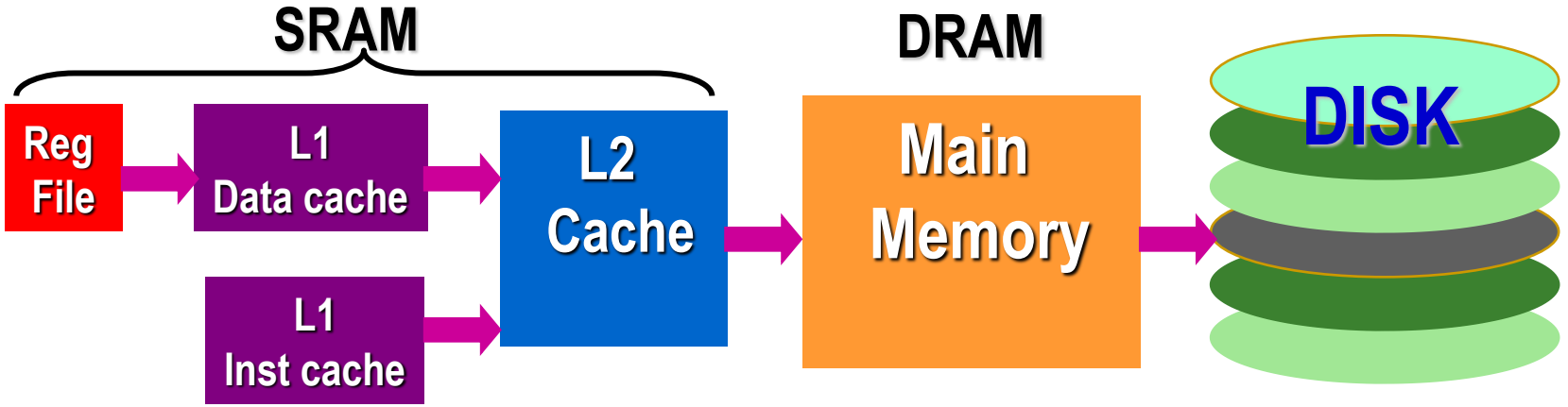
Memory-wall Effect

- Result: Multicore is scalable, but under the assumption
  - Data access time is fixed and does not increase with the amount of work and the number of cores
- **Implication:** Data access is the bottleneck needs attention

- **Conclusion**
  - The multicore result can be extended to **any** (computing) accelerator
  - Data access is the performance bottleneck of
    - Sequential processing
    - Multicore/Accelerator
    - Parallel processing
    - Scalability
- **Question**
  - How to reduce data access delay?



Memory-wall Solution: Memory Hierarchy

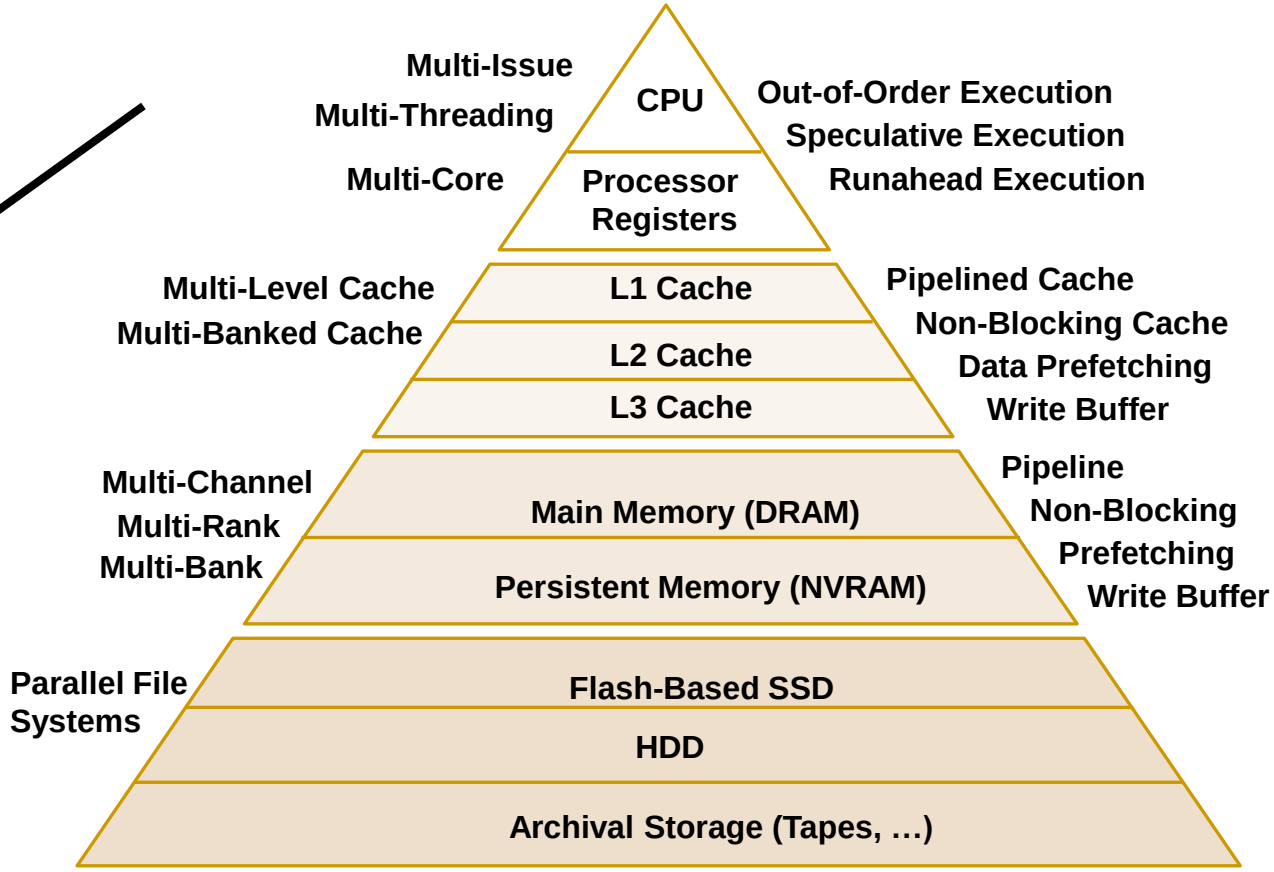


Data Locality/Concurrency

Advanced Solution: Deep Hierarchy & Concurrency

Assumptions

- Memory Hierarchy: Locality
- Concurrence: Data access pattern
  - Data stream



Deep Memory-Storage Hierarchy with Concurrence

Advanced Solution: ASIC from CPU side

- GPU, DSP, AI Chip
  - GPU is a chip tailored to graphics processing, DSP is for signal processing, and AI chip is designed to do AI tasks
- Limited solution
  - Assume data are on the chip
- Limited application
  - *Computation Accelerator*
  - Please *recall* our memory-bound results for multicore

New Solution: PIM chip

- PIM
  - Processing in memory (also called processor in memory) is the integration of a processor with RAM on a single chip.
  - NDP (Near-memory Data Processing)
  - ISP (In-Storage Processing)
- Computer power is weak
  - A full kitchen needs a refrigerator
- Limited application
  - *Data movement reducer*
  - A helper/mitigator

How to use it?

Basic Idea: Separate CPU & Memory Under von Neumann

Memory Stall Time (MST)

$$\text{CPU.time} = \text{IC} \times (\text{CPI}_{\text{exe}} + \textit{Memory stall time}) \times \text{Cycle.time}$$

- Let reducing **MST** be the **final goal** of memory systems
  - Reduce data access delay
  - Separate the concern of memory systems
- The measurement of memory system performance
  - AMAT (Average Memory Access Time)

    AMAT = Hit time + MR × AMP
  - C-AMAT (Concurrent-AMAT)
  - APC (Access Per memory active Cycle) = 1/C-AMAT

D. Wang & X.-H. Sun, "APC: A Novel Memory Metric and Measurement Methodology for Modern Memory System," in *IEEE Transactions on Computers*, vol. 63, no. 7, pp. 1626-1639, July 2014

Reduce Memory Stall Time

The Traditional AMAT model

$$CPU.time = IC \times (CPI_{exe} + f_{mem} \times AMAT) \times Cycle.time$$

Memory stall time

The New C-AMAT model

$$CPU.time = IC \times (CPI_{exe} + f_{mem} \times C\text{-}AMAT \times (1 - overlapRatio_{c\text{-}m})) \times Cycle.time$$

Memory stall time

- Reducing **MST** becomes reducing C-AMAT

Reduce C-AMAT

- C-AMAT is Recursive

$AMAT = HitCycle_1 + MR_1 \times AMP_1$

Where $AMP_1 = (HitCycle_2 + MR_2 \times AMP_2)$

Where

$$C\text{-}AMAT_1 = \frac{H_1}{C_{H_1}} + MR_1 \times \kappa_1 \times C\text{-}AMAT_2$$

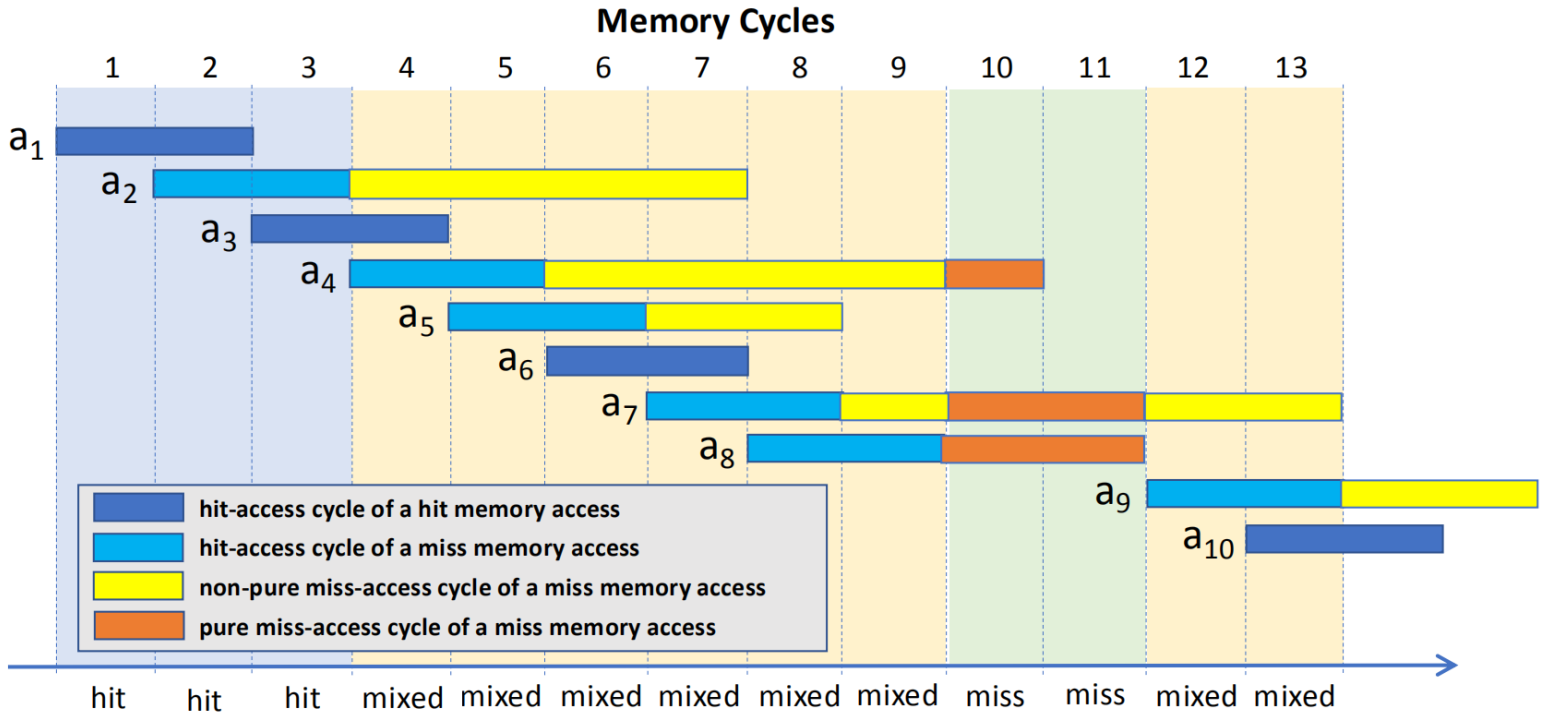
$$C - AMAT_2 = \frac{H_2}{C_{H_2}} + MR_2 \times \kappa_2 \times C - AMAT_3 \qquad \kappa_1 = \frac{pMR_1}{MR_1} \times \frac{pAMP_1}{AMP_1} \times \frac{C_{m_1}}{C_{M_1}}$$

- H is hit time
- MR is the miss ratio
- C_H is the hit concurrency
- κ is the overlapping ratio (pure miss cycles over miss cycles)
- A pure miss cycle is a miss cycle with no hit

With Clear Physical Meaning

C-AMAT : Four Types Cycle Analysis

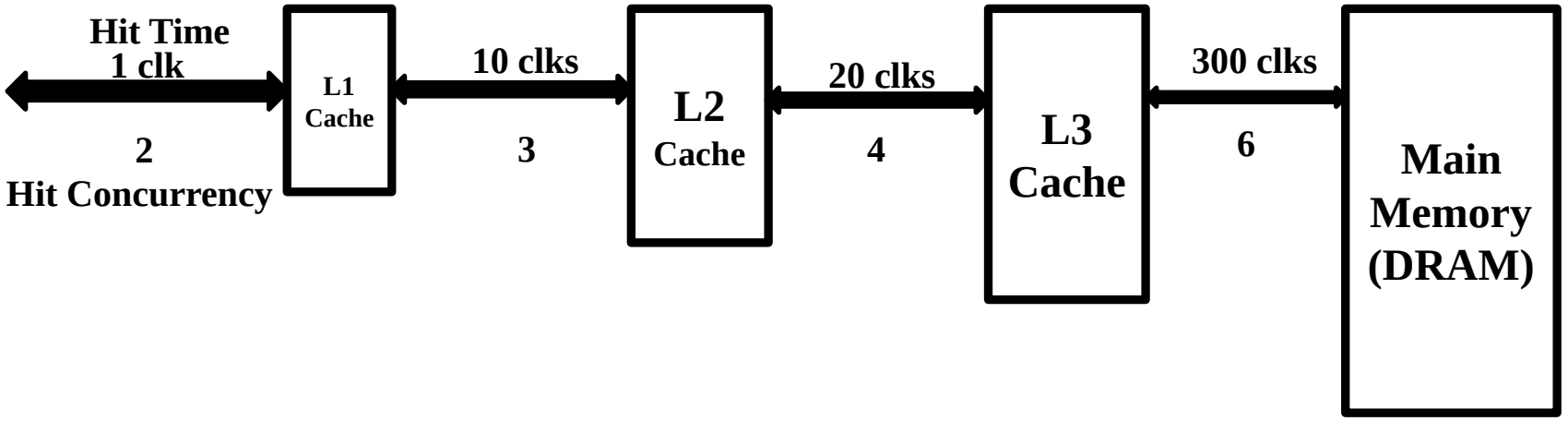
- Data (memory) centric analysis: memory cycles
- Memory cycles can see the overlapping



c	1	2	2	3	3	4	5	4	3	3	2	2	3
$c^{(h)}$	1	2	2	2	2	2	2	2	1	0	0	1	2
$c^{(m)}$	0	0	0	1	1	2	3	2	2	3	2	1	1

J. Liu, P. Espina, & X.-H. Sun, "A Study on Modeling and Optimization of Memory Systems," JCST, vol. 35, no. 1, January 2021

C-AMAT is Recursive: Data Access Time



- **Concurrent Average Memory Access Time (C-AMAT)**

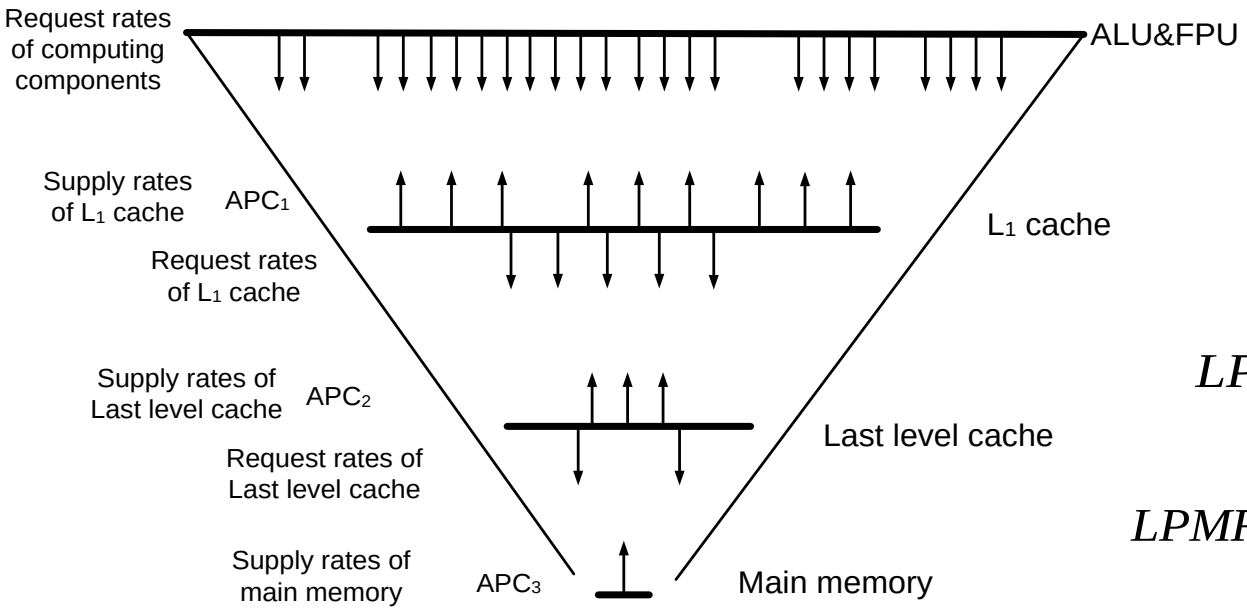
$$= \frac{H_1}{C_{H_1}} + MR_1 \times \kappa_1 \times \left(\frac{H_2}{C_{H_2}} + MR_2 \times \kappa_2 \times \left(\frac{H_3}{C_{H_3}} + MR_3 \times \kappa_3 \times \frac{H_{Mem}}{C_{H_{Mem}}} \right) \right)$$

- **Example**
  - *Miss Rate: L1=10%, L2=5%, L3=1%*
  - *κ: L1=0.56, L2=0.6, L3=0.8*
  - *C-AMAT≈0.696*

pMR, pAMP, AMP, C_M, C_m: *L1=7%, 10, 10, 5, 4*
L2=3%, 60, 40, 9, 6
L3=0.8%, 400, 300, 16, 12

Optimization: Layered Performance Matching

LPM with C-AMAT



$$LPMR_1 = \frac{IPC_{exe} \times f_{mem}}{APC_1}$$

$$LPMR_2 = \frac{IPC_{exe} \times f_{mem} \times MR_1}{APC_2}$$

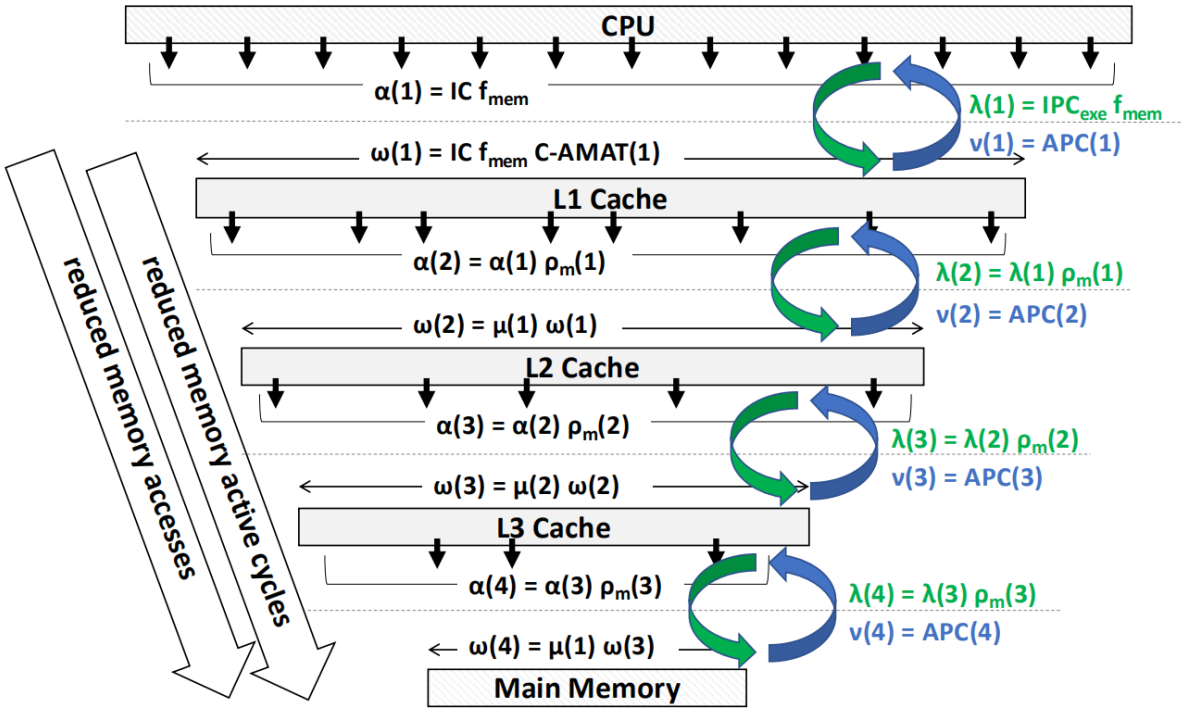
$$LPMR_3 = \frac{IPC_{exe} \times f_{mem} \times MR_1 \times MR_2}{APC_3}$$

- Match the data request and supply at each layer
- C-AMAT can increase supply with effective concurrency and locality
- Transfer a complex global problem into **simpler local match problems**

Y. Liu, X.-H. Sun. "LPM: A Systematic Methodology for Concurrent Data Access Pattern Optimization from a Matching Perspective," IEEE TPDS, vol. 30, no. 11, pp. 2478-2493, 1 Nov. 2019

Layered Performance Matching (LPM)

- The *Matching* ratio values of request and supply at each layer are given and the matching process is well designed & analyzed

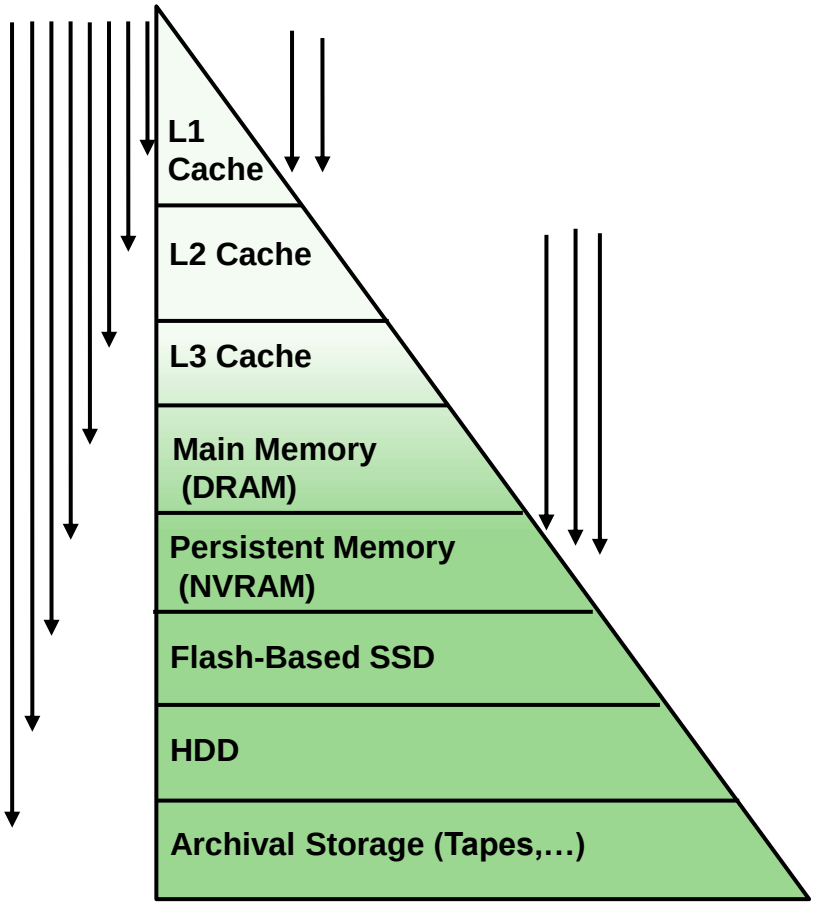


Simulatable → Measurable → Controllable → Optimizable

J. Liu, P. Espina, & X.-H. Sun, "A Study on Modeling and Optimization of Memory Systems," JCST, vol. 35, no. 1, January 2021

Deep Memory-Storage Hierarchy: a general match

- Do we need to use all layers every time?
  NO
- Flexible tier selection with no inclusive
- Concurrent accesses now can concurrently access on different tiers
- Tier: memory device with different performance
- Layer: memory hierarchy with data inclusiveness
- A general match in Deep Memory-Storage Hierarchy (DMSH)



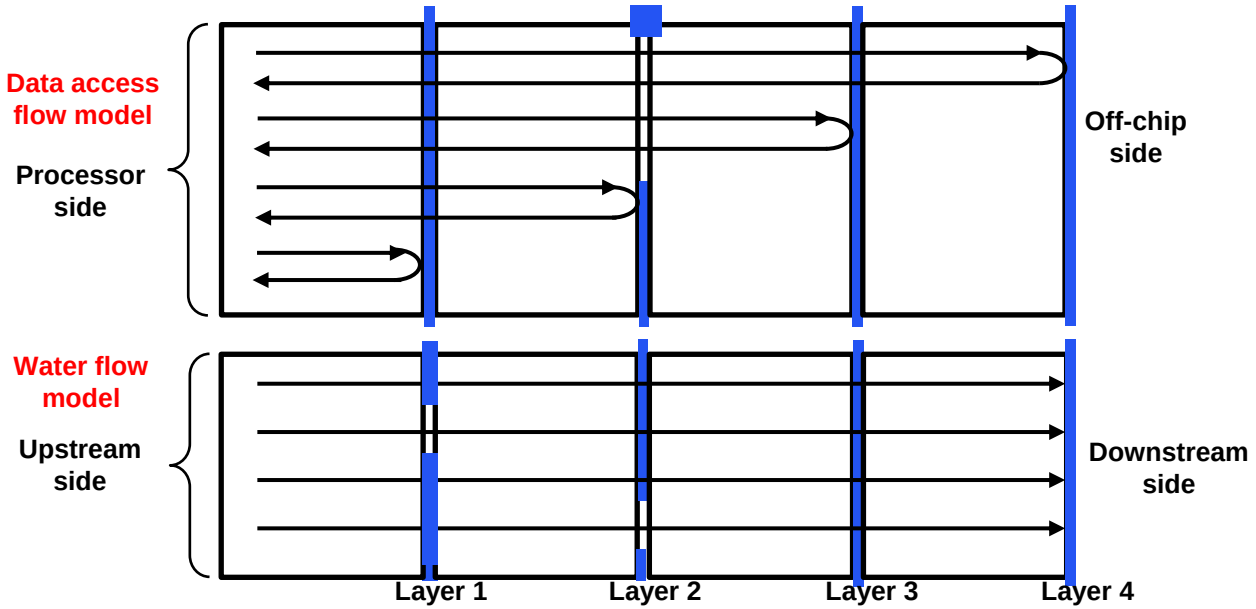
Persistent memory blurs memory & storage

DMSH with Concurrence

Y. Liu and X.-H. Sun, "CaL: Extending Data Locality to Consider Concurrency for Performance Optimization," IEEE Transactions on Big Data, vol. 5, no. 2, pp. 273-288, June 2018

Memory Sluice Gate Theory

***Sluice Gate Theorem**: If a memory system can match an application's data access requirement for any matching parameter $T_1 > 0$, then this memory system has removed the memory wall effect for this application.*



X.-H. Sun and Y.-H. Liu, "Utilizing Concurrency Data Access: A New Theory," in Proc. of LCPC2016, Sept. 2016, New York, USA

Next Step: Include PIM into the Picture

Memory Stall Time (MST)

$$\text{CPU.time} = \text{IC}_{\text{exe}} \times (\text{CPI}_{\text{exe}} + \textit{Memory stall time}) \times \text{Cycle.time} + \text{IC}_{\text{pim}} \times \text{CPI}_{\text{pim}} \times \text{Cycle.time}_{\text{pim}}$$

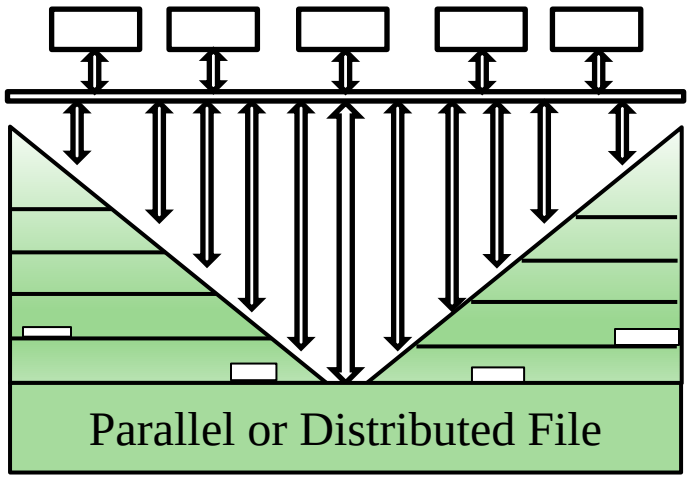
- Add PIM into the performance formulation
- PIM is a way to reduce request and is a trade-off of computing and MST
- Result: data movement cost decides where to do the computing

Data Flow Sluice Gate Control

- Two classes of computing devises, powerful CPU (multicore, GPU, XPU. Etc.) and less powerful PIM (NDP, ISP， etc.)
- Sluice gates decide which data are processed on PIM
- (rest) Data flows from memory in a rhythmic, concurrent matching fashion, passing through sluice gates (layers) before reach a CPU, then return to memory
- A general structure:
  - Fin-in, fin-out, branch,
  - More than one PIM/NDP/ISP and more than one CPU/GPU/XPU
  - Staged execution
  - Storage is the last layer of the data movement hierarchy

Data Flow/move under von Neumann

- Memory hierarchy with PIM (von Neumann)
- *Optimize 【compute + data access】 via Sluice Gate theory*
- Data flow from memory to CPU with minimum MST and conduct processing in memory when necessary
- $Dataflow_v$



$Dataflow_v$: Workload Offloading in PIM

- **CoPIM:** a Concurrency-aware PIM workload offloading architecture
- **PEI:** a locality aware (**LLC miss**) offloading approach to decide where the PIM operations should be executed
- GraphPIM : utilizing the fact that atomic functionalities cause inefficient memory system, but are suitable for PIM

CPU + PIM core performance differences

Offloading too much or too little will hurt the overall performance

a New metric: to define the code indeed induce memory stall

Most of these partitioning strategies aim to move highly 'data-intensive' portions of the application to PIM logic units

A. Baoroumnd et al., "CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators," in ISCA, 2019, pp. 629–642

A New Metric considering Concurrency

- **Concurrent-AMAT (C-AMAT)** is the metric used in **LPM**
- Based on **C-AMAT**, memory stall is due to **pure miss**, where pure miss is miss which contains at least one pure miss cycle and pure miss cycle is a miss cycle which does not overlap with any hit access
- **Pure miss cycle** provide a better way than miss to determine offloading

Based on the insights from the C-AMAT model, we use the pure miss cycle rate of LLC(θ) to describe the cache efficiency of a loop code block

$$\theta = \frac{\#\ of\ LLC\ Pure\ Miss\ Cycles}{\#\ of\ Total\ CPU\ Cycles}.$$

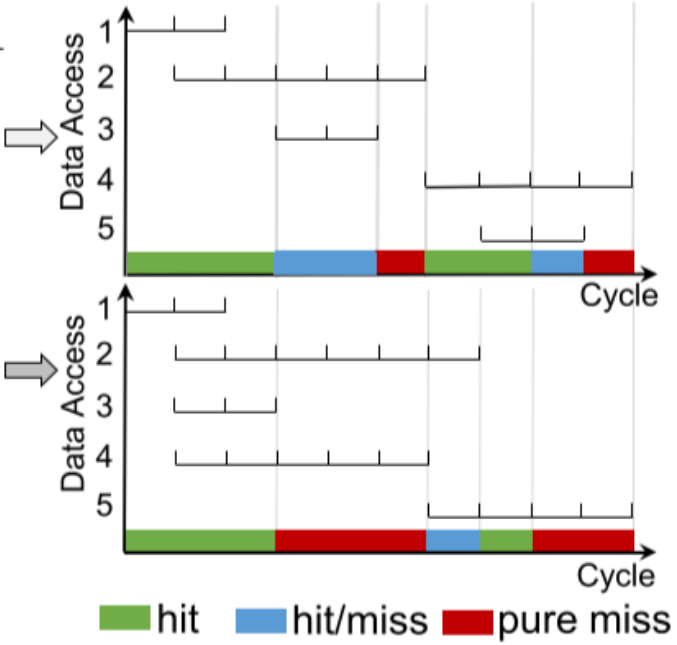
L. Yan, et al., "CoPIM: A Currency-Aware PIM Workload Offloading Architecture," submitted for publication

A New Metric considering Concurrency

- An example

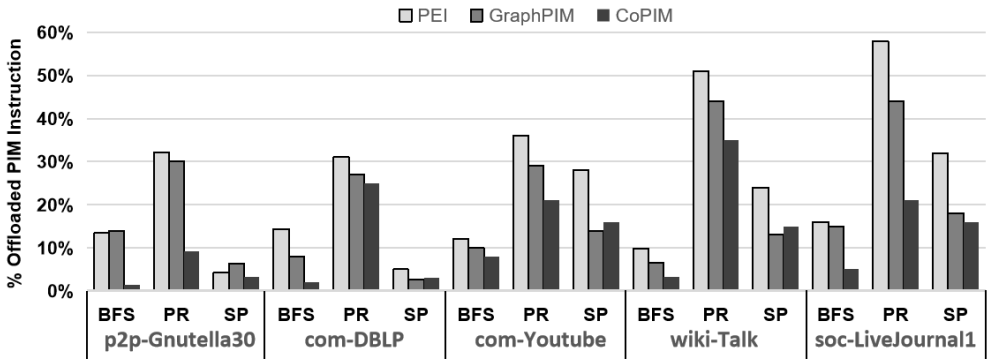
```
Bellman-Ford Shortest Path(G, w, v)
1:  for i =1 to |G.V|-1 do:
2:      if v is source then v.d = 0
3:      else v.d = infinity
4:      v' = null
5:  for i =1 to |G.V|-1 do
6:      for each edge(u,v) with weight w in edges:
7:          if v.d >u.d + w(u,v)
8:              v.d = u.d + w
9:              v' = u
10: for each edge(u,v) with weight w in edges:
11:     if v.d > u.d + w(u,v)
12:         return FALSE
13: return TRUE
```

(a)



(b)

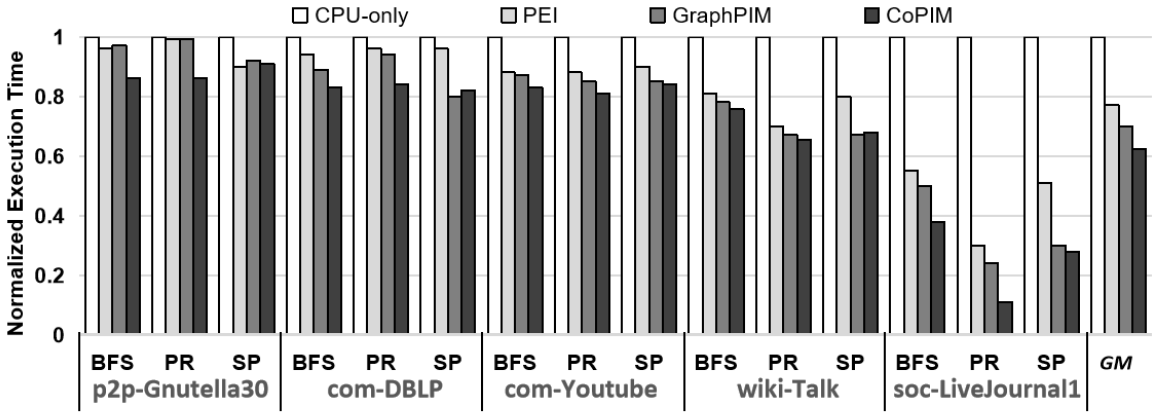
Performance Comparison



Percentage of offloaded instructions into memory

Speedup by **19.5%** than **PEI** with **51.1%** fewer offloaded instructions

speedup by **11.4%** than GraphPIM with **33.0%** fewer offloaded instructions



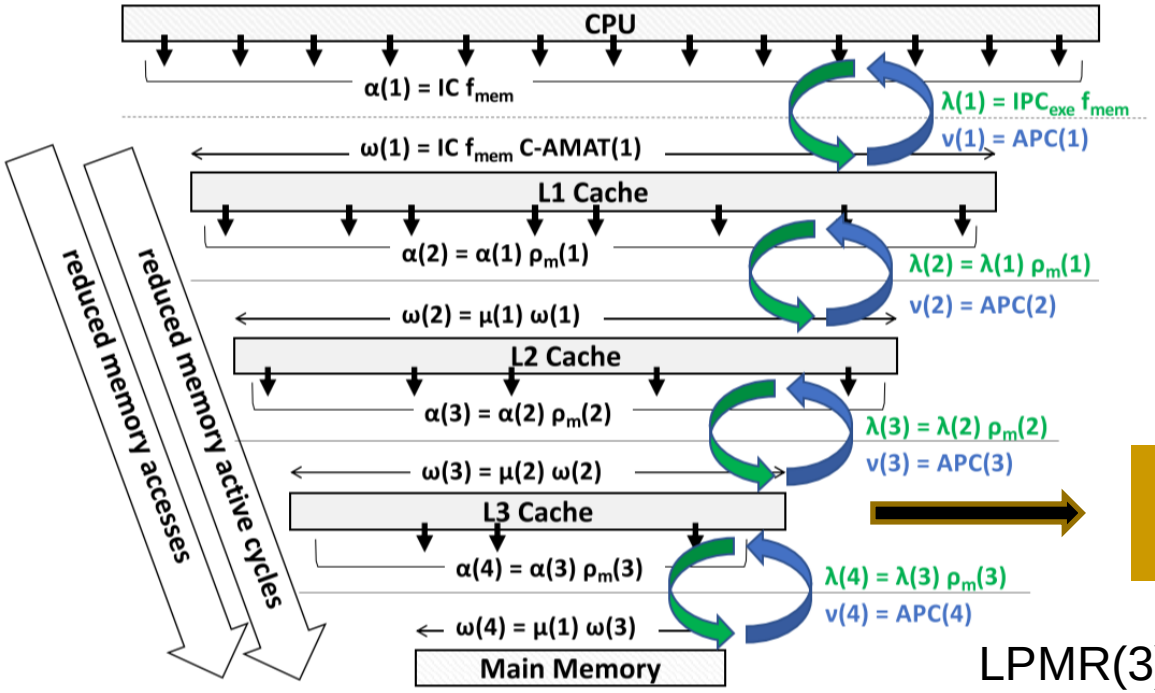
Normalized performance evaluation using graphs of different sizes, GM: geometric mean.

Direct LPM Matching

- LPMR(l) is the matching ratio at cache level l. Let λ(l) be the request rate at cache level l, and let ν(l) be the supply rate at cache level l.

$$LPMR(l) = \frac{\lambda(l)}{\nu(l)}$$

- understanding of Layered Performance Matching: the use of LPMR(i)



target of PIM architecture

LPMR(3) > T 则访存特性差

$$\frac{\Delta}{\mu(1)\cdot\kappa(1)}\cdot\prod_{i=1}^{l-1}\mu(i)$$

Δ is the key identification of PIM offloading

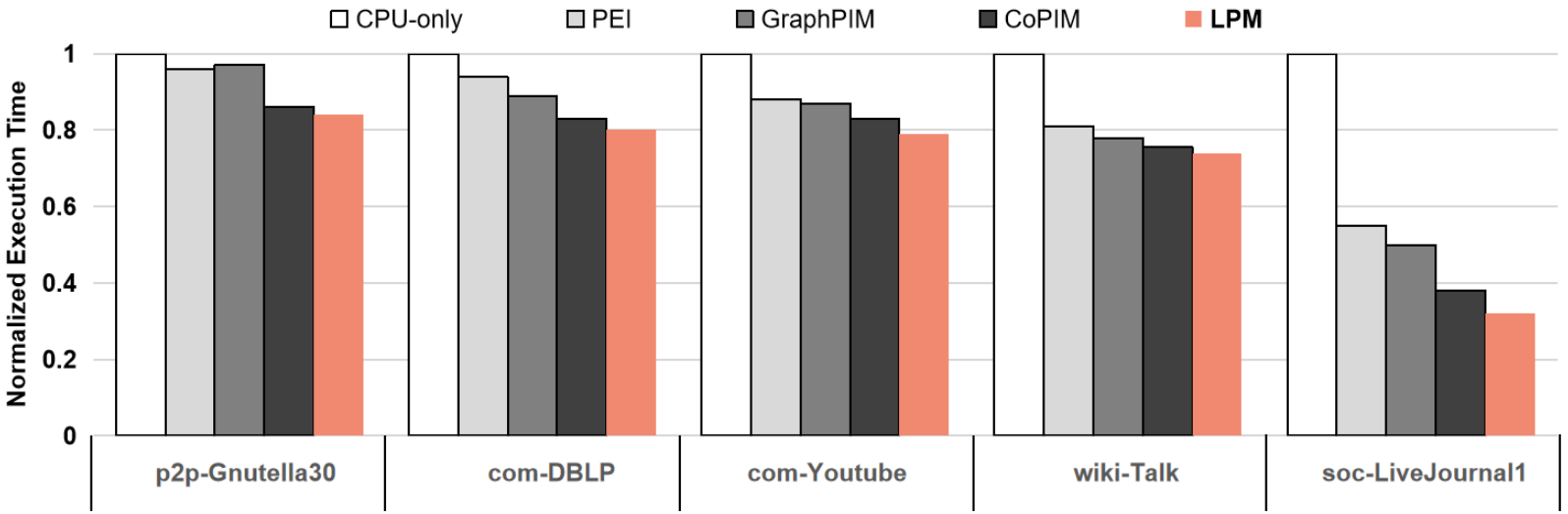
Experimental Results

- Execution time:

  datasets: p2p-Gnutella30.....soc-LiveJournal1

  application: BFS（Breadth-First search）

  Δ：60%

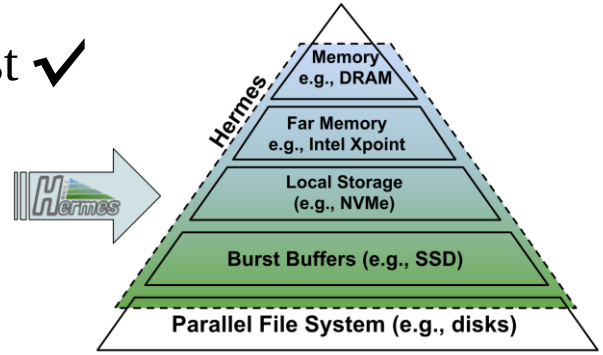


Note: LPM shows better offloading efficiency than other offloading strategies under the BFS application.

LPM Matching: I/O-level implementation

- PIM assumes data is already in memory
- Storage is the last level of the memory hierarchy (DMSH)√
- Start at where the data is
- Advantage
  - Can be implemented and verified
- ***Challenges***
  - Data management ✓
  - Network impact √
  - Passing operation demands with data request ✓

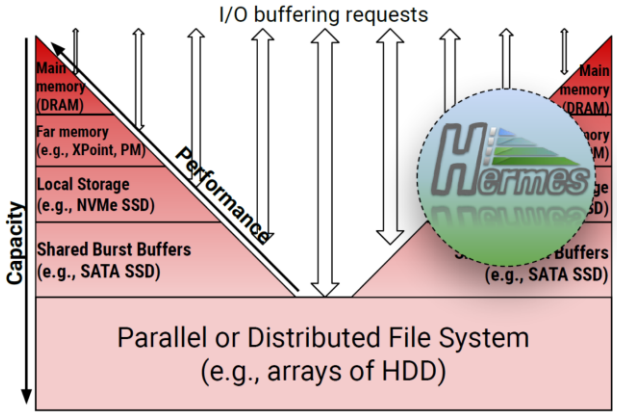
Let us do it ***(on going CSSI framework)***

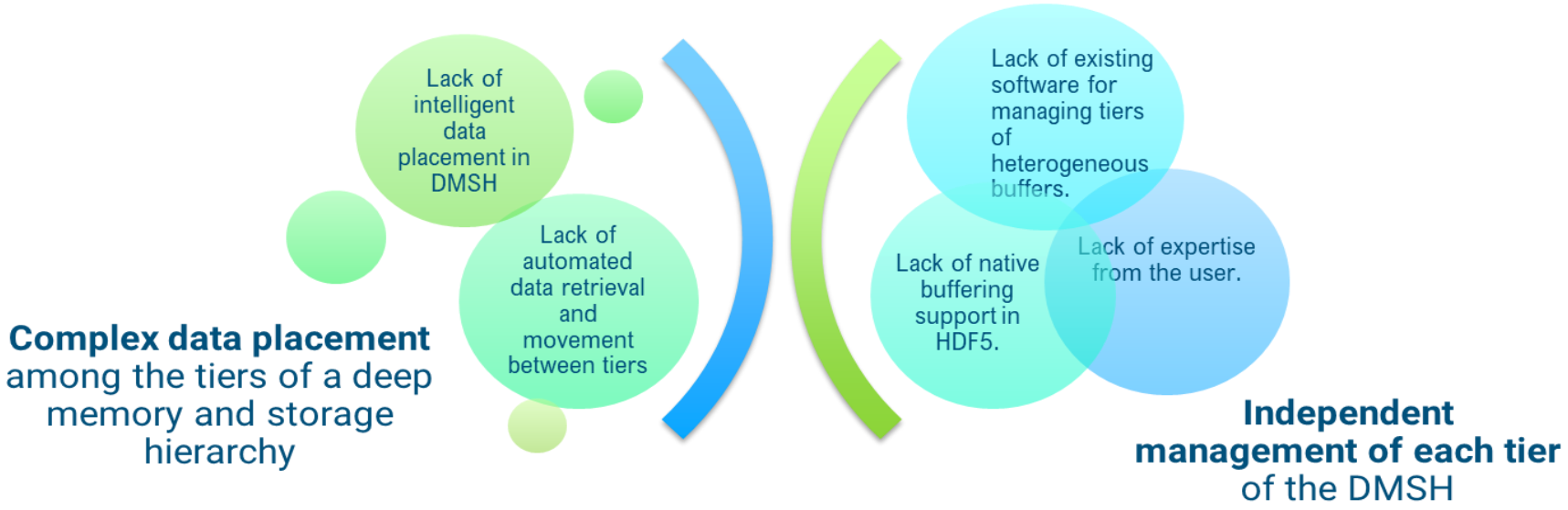


Anthony Kougkas, Hariharan Devarajan, and Xian-He Sun. "Hermes: a heterogeneous-aware multi-tiered distributed I/O buffering system," ACM, HPDC18, Tempe, Arizona, USA, June 2018

Hermes: A Multi-tiered I/O Buffering System

- Application-aware multi-tier matching
- Start at the log file
- An example of memory/storage integration
- An implementation of the $Dataflow_v$ concept





Take Home Messages

- What are the challenges of PIM (from performance)?
  - PIM is data movement reducer for the memory-wall problem
  - It is inherently tight to the complex memory system performance, including storage
- What are the opportunities (from performance)?
  - Many new applications are data intensive and data driven
  - An integrated model, $\boldsymbol{Dataflow_v}$, is developed where PIM is a pivoting factor (an I/O implementation)
- How to do it?
  - Theoretical methodology and practical experience
  - There is a merging of memory and storage, memory and processor
- Many things remain open
  - From language to system to …

A. Kougkas, H. Devarajan, K. Bateman, J. Cernuda, N. Rajesh and X.-H. Sun, "ChronoLog: A Distributed Shared Tiered Log Store with Time-based Data Ordering," Proceedings of MSST 2020, Oct. 2020,

Conclusion

- PIM should be utilized with the consideration of memory system and multicore/CPU
- Opportunity is plenty, as well as challenges
  - $Dataflow_v$
- The potential is high

Thank you
Any questions?

The Challenges and Opportunities of Processing-in-Memory:
A performance point of view

The SCS laboratory at the Illinois Institute of Technology & Collaborators

sun@iit.edu

www.cs.iit.edu/~scs

We would like to thank our sponsors the

National Science Foundation