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

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


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
    \[
    \text{Speedup}_{\text{fixed-time}} = \frac{\text{Sequential Time of Solving Scaled Workload}}{\text{Parallel Time of Solving Scaled Workload}} = \alpha + (1 - \alpha)p
    \]

- Sun and Ni’s law, 1990
  - Memory-bounded speedup model
    \[
    \text{Speedup}_{\text{memory-bounded}} = \frac{\text{Sequential Time of Solving Scaled Workload}}{\text{Parallel Time of Solving Scaled Workload}} = \frac{\alpha + (1 - \alpha)G(p)}{\alpha + (1 - \alpha)G(p)/p}
    \]

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)

---


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

The Berkeley roofline Model

Cost variation cross layers

Memory bound analysis

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


**Multicore technology (2004), Big Data Initiative (2012)**
Amdahl’s Law for Multicore (Hill&Marty08)

- Speedup of symmetric architecture
  \[
  \text{Speedup}_{\text{symmetric}}(f, n, r) = \frac{1}{1 - f \frac{1}{\text{perf}(r)} + f \frac{r}{\text{perf}(r) \cdot n}}
  \]

- Speedup of asymmetric architecture
  \[
  \text{Speedup}_{\text{asymmetric}}(f, n, r) = \frac{1}{1 - f \frac{1}{\text{perf}(r)} + f \frac{r}{\text{perf}(r) + n - r}}
  \]
Scaled Speedup under Memory-wall

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

\[
\frac{w_c}{\text{perf}(r)} + \frac{w_p}{\text{perf}(r)} = \frac{w_c}{\text{perf}(r)} + \frac{w_p'}{m \cdot \text{perf}(r)} \quad \Rightarrow \quad w_p' = mw_p
\]

- Fixed-time speedup

\[
\frac{w_c}{\text{perf}(r)} + \frac{w_p'}{m \cdot \text{perf}(r)} = \frac{w_c + m \cdot w_p}{w_c + w_p} = (1 - f') + mf' \quad \Rightarrow \quad f' = \frac{w_p}{w_c + w_p}
\]

- 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 same as fixed-size, \( g(m) \) equals \( m \), then memory-bound is the same as fixed-time

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

Reg File → L1 Data cache → L2 Cache → Main Memory → DISK

SRAM → DRAM

Data Locality/Concurrency
Assumptions

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

Deep Memory-Storage Hierarchy with Concurrency

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

Memory Stall Time (MST)

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

- 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)
    \[ \text{AMAT} = \text{Hit time} + \text{MR} \times \text{AMP} \]
  - C-AMAT (Concurrent-AMAT)
  - APC (Access Per memory active Cycle) = \( 1/\text{C-AMAT} \)

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

\[ CPU\_time = IC \times (CPI\_exe + f_{mem} \times C-AMAT \times (1 - overlapRatio_{c-m})) \times Cycle\_time \]

Memory stall time

- Reducing MST becomes reducing C-AMAT
Reduce C-AMAT

C-AMAT is Recursive

Where

\[ C_{-AMAT_1} = \frac{H_1}{C_{H_1}} + MR_1 \times \kappa_1 \times C_{-AMAT_2} \]

\[ C_{-AMAT_2} = \frac{H_2}{C_{H_2}} + MR_2 \times \kappa_2 \times C_{-AMAT_3} \]

\[ \kappa_1 = \frac{pMR_1}{MR_1} \times \frac{pAMP_1}{AMP_1} \times \frac{C_{m_1}}{C_{M_1}} \]

- H is hit time
- MR is the miss ratio
- \( C_{H} \) is the hit concurrency
- \( \kappa \) 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

![Diagram of Memory Cycles]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>c</strong></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td><strong>c^{(h)}</strong></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>c^{(m)}</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
C-AMAT is Recursive: Data Access Time

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

\[
\text{C-AMAT} \approx \frac{H_1}{C_{H_1}} + \text{MR}_1 \times \kappa_1 \times \left( \frac{H_2}{C_{H_2}} + \text{MR}_2 \times \kappa_2 \times \left( \frac{H_3}{C_{H_3}} + \text{MR}_3 \times \kappa_3 \times \frac{H_{Mem}}{C_{H_{Mem}}} \right) \right)
\]

- **Example**
  - Miss Rate: L1=10%, L2=5%, L3=1%
  - MR, AMP, C, C_m: L1=7%, 10, 10, 5, 4
  - \(\kappa\): L1=0.56, L2=0.6, L3=0.8
  - C-AMAT≈0.696

---


©copy right 2020 Xian-He Sun
Optimization: Layered Performance Matching

LPM with C-AMAT

- 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

\[
LPMR_1 = \frac{IPC_{exe} \times f_{mem}}{APC_1} \\
LPMR_2 = \frac{IPC_{exe} \times f_{mem} \times MR_1}{APC_2} \\
LPMR_3 = \frac{IPC_{exe} \times f_{mem} \times MR_1 \times MR_2}{APC_3}
\]

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

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


©copy right 2020 Xian-He Sun
**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.

Next Step: Include PIM into the Picture

Memory Stall Time (MST)

\[
\text{CPU.time} = IC_{\text{exe}} \times (CPI_{\text{exe}} + \text{Memory stall time}) \times \text{Cycle.time} \\
+ IC_{\text{pim}} \times CPI_{\text{pim}} \times \text{Cycle.time}_{\text{pim}}
\]

- 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 devices, 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 \( \text{compute} + \text{data access} \) via Sluice Gate theory
- Data flow from memory to CPU with minimum MST and conduct processing in memory when necessary
- Dataflow

"Parallel or Distributed File"
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 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

\[
\theta = \frac{\text{# of LLC Pure Miss Cycles}}{\text{# of Total CPU Cycles}}
\]

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)

PIMSim, an open-source PIM simulator based on Gem5

(b)
Performance Comparison

Percentage of offloaded instructions into memory

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

Normal 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)

\[ \Delta \] is the key identification of PIM offloading

\[ \frac{\Delta}{\mu(1) \cdot \kappa(1) \cdot \prod_{i=1}^{l-1} \mu(i)} \]
Experimental Results

- Execution time:
  - datasets: p2p-Gnutella30.....soc-LiveJournal1
  - application: BFS (Breadth-First search)
  - $\Delta$: 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)


©copy right 2020 Xian-He Sun
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\textsubscript{\nu} 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, *Dataflow* \(\nu\), 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 …

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

We would like to thank our sponsors the
National Science Foundation

sun@iit.edu
www.cs.iit.edu/~scs