Reminder: Project Proposals

• Project proposals due NOON on Monday 9/26
• Two to three pages consisting of
  – Problem
  – Novelty
  – Idea
  – Hypothesis
  – Methodology
  – Plan
• All the details are in the project handout
Agenda for Today’s Class

• Brief background on hybrid main memories
• Project example from Fall 2010
• Project pitches and feedback
• Q & A
Main Memory in Today’s Systems

CPU

DRAM

HDD/SSD
Main Memory in Today’s Systems

CPU

Main memory

DRAM

HDD/SSD
DRAM

• Pros
  – Low latency
  – Low cost

• Cons
  – Low capacity
  – High power

• Some new and important applications require HUGE capacity (in the terabytes)
Main Memory in Today’s Systems

CPU

Main memory

DRAM

HDD/SSD
Hybrid Memory (Future Systems)

Hybrid main memory

- DRAM (cache)
- New memories (high capacity)

HDD/SSD
Row Buffer Locality-Aware Hybrid Memory Caching Policies

Justin Meza
HanBin Yoon
Rachata Ausavarungnirun
Rachael Harding
Onur Mutlu
Motivation

• Two conflicting trends:
  1. ITRS predicts the end of DRAM scalability
  2. Workloads continue to demand more memory

• Want future memories to have
  – Large capacity
  – High performance
  – Energy efficient

• Need scalable DRAM alternatives
Motivation

• Emerging memories can offer more scalability
• Phase change memory (PCM)
  – Projected to be 3–12× denser than DRAM
• However, cannot simply replace DRAM
  – Longer access latencies (4–12× DRAM)
  – Higher access energies (2–40× DRAM)
• Use DRAM as a cache to large PCM memory

[Mohan, HPTS ’09; Lee+, ISCA ’09]
Phase Change Memory (PCM)

• Data stored in form of resistance
  – High current melts cell material
  – Rate of cooling determines stored resistance
  – Low current used to read cell contents
## Projected PCM Characteristics (~2013)

<table>
<thead>
<tr>
<th></th>
<th>32 nm</th>
<th>DRAM</th>
<th>PCM</th>
<th>Relative to DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell size</td>
<td>6 F²</td>
<td>0.5–2 F²</td>
<td>3–12× denser</td>
<td></td>
</tr>
<tr>
<td>Read latency</td>
<td>60 ns</td>
<td>300–800 ns</td>
<td>6–13× slower</td>
<td></td>
</tr>
<tr>
<td>Write latency</td>
<td>60 ns</td>
<td>1400 ns</td>
<td>24× slower</td>
<td></td>
</tr>
<tr>
<td>Read energy</td>
<td>1.2 pJ/bit</td>
<td>2.5 pJ/bit</td>
<td>2× more energy</td>
<td></td>
</tr>
<tr>
<td>Write energy</td>
<td>0.39 pJ/bit</td>
<td>16.8 pJ/bit</td>
<td>40× more energy</td>
<td></td>
</tr>
<tr>
<td>Durability</td>
<td>N/A</td>
<td>10⁶–10⁸ writes</td>
<td>Limited lifetime</td>
<td></td>
</tr>
</tbody>
</table>

[Mohan, HPTS ’09; Lee+, ISCA ’09]
Row Buffers and Locality

• Memory array organized in columns and rows
• Row buffers store contents of accessed row
• Row buffers are important for mem. devices
  – Device slower than bus: need to buffer data
  – Fast accesses for data with spatial locality
  – DRAM: Destructive reads
  – PCM: Writes are costly: want to coalesce
Row Buffers and Locality

Row decoder

Row address

Memory cell array

ROW DATA

Row buffer

LOAD X

LOAD X+1

Row buffer hits!
Key Idea

• Since DRAM and PCM both use row buffers,
  – Row buffer hit latency same in DRAM and PCM
  – Row buffer miss latency small in DRAM
  – Row buffer miss latency large in PCM

• Cache data in DRAM which
  – Frequently row buffer misses
  – Is reused many times

→ because miss penalty is smaller in DRAM
Hybrid Memory Architecture

- CPU
- Memory Controller
- DRAM Cache (Low density)
- PCM (High density)

Memory channel
Hybrid Memory Architecture
Hybrid Memory Architecture

- CPU
- Memory Controller
- DRAM Cache (Low density)
- PCM (High density)
- Tag store: 2 KB rows
Hybrid Memory Architecture

Tag store: X \( \rightarrow \) DRAM

- CPU
  - Memory Controller
  - LOAD X

- DRAM Cache (Low density)
- PCM (High density)
Hybrid Memory Architecture

How does data get migrated to DRAM?

→ Caching Policy
Methodology

• Simulated our system configurations
  – Collected program traces using a tool called Pin
  – Fed instruction trace information to a timing simulator modeling an OoO core and DDR3 memory
    – Migrated data at the row (2 KB) granularity
• Collected memory traces from a standard computer architecture benchmark suite
  – SPEC CPU2006
• Used an in-house simulator written in C#
Conventional Caching

• Data is migrated when first accessed
• Simple, used for many caches
Conventional Caching

- Data is migrated when first accessed
- Simple, used for many caches

How does conventional caching perform in a hybrid main memory?

Bus contention
Conventional Caching

![Bar chart comparing IPC normalized to All DRAM for Conventional Caching and No Caching (All PCM) across various benchmarks. The chart shows a comparison of performance for different applications under conventional caching and no caching scenarios.]
Conventional Caching

No Caching (All PCM) vs Conventional Caching

Beneficial for some benchmarks
Conventional Caching

Performance degrades due to bus contention
Conventional Caching

Many row buffer hits: don’t need to migrate data
Conventional Caching

Want to identify data which misses in row buffer and is reused
Problems with Conventional Caching

• Performs useless migrations
  – Migrates data which are not reused
  – Migrates data which hit in the row buffer

• Causes bus contention and DRAM pollution
  – Want to cache rows which are reused
  – Want to cache rows which miss in row buffer
Problems with Conventional Caching

• Performs useless migrations
  – Migrates data which are not reused
  – Migrates data which hit in the row buffer

• Causes bus contention and DRAM pollution
  – Want to cache rows which are reused
  – Want to cache rows which miss in row buffer
A Reuse-Aware Policy

• Keep track of the number of accesses to a row
• Cache row in DRAM when accesses ≥ A
  – Reset accesses every Q cycles
• Similar to CHOP [Jiang+, HPCA ’10]
  – Cached “hot” (reused) pages in on-chip DRAM
  – To reduce off-chip bandwidth requirements
• We call this policy A-COUNT
A Reuse-Aware Policy

IPC Normalized to All DRAM

- No Caching (All PCM)
- Conventional Caching
- A-COUNT.4

Graph showing IPC normalized to All DRAM for different benchmarks.
A Reuse-Aware Policy

IPC Normalized to All DRAM

<table>
<thead>
<tr>
<th></th>
<th>No Caching (All PCM)</th>
<th>Conventional</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcf</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>milc</td>
<td>0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>cactusADM</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>leslie3d</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>soplex</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>sjeng</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>GemsFDTD</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>libquantum</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>ibm</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>omnetpp</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>astar</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>xalancbmk</td>
<td>0.9</td>
<td>0.8</td>
</tr>
<tr>
<td>gmean</td>
<td>0.9</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Performs fewer migrations: reduces channel contention
A Reuse-Aware Policy

Too few migrations: too many accesses go to PCM

IPC Normalized to All DRAM

mcf, milc, cactusADM, leslie3d, soplex, sjeng, GemsFDTD, libquantum, ibm, omnetpp, astar, xalancbmk, gmean
A Reuse-Aware Policy

Rows with many hits still needlessly migrated

IPC Normalized to All DRAM

mcf  milc  cactusADM  leslie3d  soplex  sjeng  GemsFDTD  libquantum  ibm  omnetpp  astar  xalancbmk  gmean

No Caching (All PCM)  Conventional Caching
Problems with Reuse-Aware Policy

• Agnostic of DRAM/PCM access latencies
  – May keep data which row buffer misses in PCM
  – Missed opportunity: could save cycles in DRAM
Problems with Reuse-Aware Policy

• Agnostic of DRAM/PCM access latencies

Data with frequent row buffer hits

<table>
<thead>
<tr>
<th>Time</th>
<th>PCM</th>
<th>DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
</tr>
</tbody>
</table>

Saved cycles if placed in DRAM

Data with frequent row buffer misses

<table>
<thead>
<tr>
<th>Time</th>
<th>PCM</th>
<th>DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
</tr>
</tbody>
</table>

Saved cycles if placed in DRAM
Row Buffer Locality-Aware Policy

- Cache rows which benefit from being in DRAM
  - I.e., those with frequent row buffer misses
- Keep track of number of misses to a row
- Cache row in DRAM when misses ≥ M
  - Reset misses every Q cycles
- We call this policy M-COUNT
Row Buffer Locality-Aware Policy

IPC Normalized to All DRAM

- No Caching (All PCM)
- Conventional Caching
- A-COUNT.4
- M-COUNT.2
Row Buffer Locality-Aware Policy

Recognizes rows with many hits and does not migrate them
Row Buffer Locality-Aware Policy

Lots of data with just enough misses to get cached but little reuse after being cached → need to also track reuse
Combined Reuse/Locality Approach

• Cache rows with **reuse** and which frequently **miss in the row buffer**
  – Use A-COUNT as predictor of future reuse and
  – M-COUNT as predictor of future row buffer misses
• Cache row if **accesses ≥ A** and **misses ≥ M**
• We call this policy **AM-COUNT**
Combined Reuse/Locality Approach

Normalized to All DRAM

No Caching (All PCM)  Conv. Caching  A-COUNT.4
M-COUNT.2  AM-COUNT.4.2
Combined Reuse/Locality Approach

- No Caching (All PCM)
- Conventional Caching
- M-COUNT.2
- AM-COUNT.4.2

Reduces useless migrations
Combined Reuse/Locality Approach

And data with little reuse kept out of DRAM
Dynamic Reuse/Locality Approach

• Previously mentioned policies require profiling
  – To determine the best $A$ and $M$ thresholds
• We propose a **dynamic threshold** policy
  – Performs a cost-benefit analysis every $Q$ cycles
  – Simple hill-climbing algorithm to maximize benefit
  – (Side note: we simplify the problem slightly by just finding the best $A$ threshold, because we observe that $M = 2$ performs the best for a given $A$.)
Cost-Benefit Analysis

• Each quantum, we measure the *first-order* costs and benefits of the current $A$ threshold
  – Cost = cycles of bus contention due to migrations
  – Benefit = cycles saved at the banks by servicing a request in DRAM versus PCM

• Cost = Migrations $\times t_{\text{migration}}$

• Benefit = \( \text{Reads}_{\text{DRAM}} \times (t_{\text{read,PCM}} - t_{\text{read,DRAM}}) \)
  + \( \text{Writes}_{\text{DRAM}} \times (t_{\text{write,PCM}} - t_{\text{write,DRAM}}) \)
Cost-Benefit Maximization Algorithm

Each quantum (10 million cycles):

1. $Net = Benefit - Cost$  // net benefit
2. if $Net < 0$ then  // too many migrations?
   3. $A++$  // increase threshold
3. else  // last $A$ beneficial
   4. if $Net > PreviousNet$ then  // increasing benefit?
      5. $A++$  // try next $A$
   6. else  // decreasing benefit
      7. $A--$  // too strict, reduce
   8. end
9. end
10. end
11. $PreviousNet = Net$
Dynamic Policy Performance

- No Caching (All PCM)
- Conventional Caching
- Best Static
- Dynamic

IPC Normalized to All DRAM

- mcf
- milc
- cactusADM
- leslie3d
- soplex
- sjeng
- GemsFDTD
- libquantum
- ibm
- omnetpp
- astar
- xalancbmk
- gmean
Dynamic Policy Performance

29% improvement over All PCM, Within 18% of All DRAM
Evaluation Methodology/Metrics

- 16-core system
- Averaged across 100 randomly-generated workloads of varying working set size
  - LARGE = working set size > main memory size

- Weighted speedup (performance) = \[ \sum \frac{IPC_{\text{together}}}{IPC_{\text{alone}}} \]

- Maximum slowdown (fairness) = max \[ \frac{IPC_{\text{alone}}}{IPC_{\text{together}}} \]
16-core Performance & Fairness

![Graphs showing performance and fairness results for 16-core systems.](image)

**Graph (a):** Weighted speedup for Conventional Caching, A-COUNT, AM-COUNT, and DAM-COUNT.

**Graph (b):** Maximum slowdown for Conventional Caching, A-COUNT, AM-COUNT, and DAM-COUNT.

**Graph (c):** Harmonic speedup for Conventional Caching, A-COUNT, AM-COUNT, and DAM-COUNT.

**Graph (d):** Power consumption for Conventional Caching, A-COUNT, AM-COUNT, and DAM-COUNT.
16-core Performance & Fairness

Figure 7 shows the steps taken by a memory controller to support promotion policies. When a memory request is issued from the CPU, the DRAM memory controller's tag store is indexed with the row address to see if the requested row resides in DRAM or PCM. The request is then placed in the appropriate request queue, where it will stay until scheduled.

If a request is scheduled in PCM, the address of the row containing the data is used to index the stats store and a value corresponding to the number of row accesses is incremented if the request is a read and increased by a value $W$ if the request is a write (cf. Section 4.3); if the row content is not present in the row buffer, a value corresponding to the number of row buffer misses is also incremented.

After the requested memory block from PCM (whose access is on the critical path) is sent back to the CPU, the promotion policy is invoked. If the number of row accesses is greater than a threshold $A$ and the number of row buffer misses is greater than a threshold $M$, then the row is copied to the write buffer in the DRAM memory controller (i.e., promoted). A replacement policy in DRAM is used to determine a row to replace. Note that if the row was not modified while in DRAM, it does not need to be written back to PCM, otherwise, only the dirty contents of the row are sent to PCM to be written back.

Updating the stats store and invoking the promotion policy are not on the critical path, because they can be performed in parallel with accessing and transferring the critical word from PCM. If needed, the stats store's associative comparison logic can take multiple cycles to access, as the time taken to read data from PCM is on the order of hundreds of CPU cycles.

6 Experimental results

6.1 16-core results

We first analyze the performance and fairness of our technique, DAM-COUNT, on a 16-core system compared to three promotion policies: conventional caching, A-COUNT, and AM-COUNT. Note that for A- and AM-COUNT results, it was not feasible to find the best static $A$ threshold for the large number of workloads surveyed. We instead show results for $A = 4$ and $M = 2$, which were found to be effective over a wide range of workloads. We show the A-COUNT results for reference to illustrate the limitations of a promotion policy that only considers row reuse (and not row buffer locality).

Data were collected for the initial 100 million cycles of a simulation run following a 10 million cycle warm-up period. The applications we use have relatively small working set sizes at reasonable simulation lengths and to ensure that we study the problem of data placement in hybrid memory systems properly, we set the DRAM size such that the working sets do not reside mainly in the DRAM cache. Longer simulations showed results consistent with shorter ones.

More contention → more benefit
16-core Performance & Fairness

Figure 7 shows the steps taken by a memory controller to support promotion policies. When a memory request is issued from the CPU, the DRAM memory controller's tag store is indexed with the row address to see if the requested row resides in DRAM or PCM. The request is then placed in the appropriate request queue, where it will stay until scheduled.

If a request is scheduled in PCM, the address of the row containing the data is used to index the stats store and a value corresponding to the number of row accesses is incremented if the request is a read and increased by a value \( W \) if the request is a write (cf. Section 4.3); if the row content is not present in the row buffer, a value corresponding to the number of row buffer misses is also incremented.

After the requested memory block from PCM (whose access is on the critical path) is sent back to the CPU, the promotion policy is invoked. If the number of row accesses is greater than a threshold \( A \) and the number of row buffer misses is greater than a threshold \( M \), then the row is copied to the write buffer in the DRAM memory controller (i.e., promoted). A replacement policy in DRAM is used to determine a row to replace. Note that if the row was not modified while in DRAM, it does not need to be written back to PCM, otherwise, only the dirty contents of the row are sent to PCM to be written back.

Updating the stats store and invoking the promotion policy are not on the critical path, because they can be performed in parallel with accessing and transferring the critical word from PCM. If needed, the stats store's associative comparison logic can take multiple cycles to access, as the time taken to read data from PCM is on the order of hundreds of CPU cycles.

6 Experimental results

6.1 16-core results

We first analyze the performance and fairness of our technique, DAM-COUNT, on a 16-core system compared to three promotion policies: conventional caching, A-COUNT, and AM-COUNT. Note that for A- and AM-COUNT results, it was not feasible to find the best static \( A \) threshold for the large number of workloads surveyed. We instead show results for \( A = 4 \) and \( M = 2 \), which were found to be effective over a wide range of workloads. We show the A-COUNT results for reference to illustrate the limitations of a promotion policy that only considers row reuse (and not row buffer locality).

Data were collected for the initial 100 million cycles of a simulation run following a 10 million cycle warm-up period. The applications we use have relatively small working set sizes at reasonable simulation lengths and to ensure that we study the problem of data placement in hybrid memory systems properly, we set the DRAM size such that the working sets do not reside mainly in the DRAM cache. Longer simulations showed results consistent with shorter ones.

We find least frequently used to perform the best, however, the performance of least recently used follows very closely.

Dynamic policy can adjust to different workloads.
Versus All PCM and All DRAM

• Compared to an **All PCM** main memory
  – 17% performance improvement
  – 21% fairness improvement

• Compared to an **All DRAM** main memory
  – Within 21% of performance
  – Within 53% of fairness
Robustness to System Configuration

Different numbers of cores.

Figure 10 shows the weighted speedup of our DAM-COUNT policy for 200 randomly-generated workloads on 2-, 4-, 8-, and 16-core systems, normalized to conventional caching. Results are sorted in terms of increasing normalized weighted speedup.

Our technique achieves more performance benefit as the number of cores increases, yet there are a few workloads where our technique does not perform as well as the conventional caching baseline (on 2- and 4-core systems). This is because for some workloads composed of a large proportion of SMALL benchmarks, the working set both fits entirely in the DRAM cache and has high reuse. In such exceptional cases, all data can be promoted without tracking row reuse or row buffer locality information.

DRAM cache size.

Figure 11 shows the performance of conventional caching and DAM-COUNT for DRAM cache sizes from 64 MB to 512 MB averaged across 200 randomly-generated workloads consisting of 100% LARGE benchmarks, to exercise the DRAM cache. There are two things to note. First, even when a larger portion of the working set of workloads fits in the cache (e.g., 512 MB on Figure 11), DAM-COUNT outperforms conventional caching. This is because, compared to conventional caching, DAM-COUNT reduces the amount of channel contention and also accesses data from PCM, enabling channel-level parallelism.

With 256 MB of DRAM, we are able to achieve within 21% of the weighted speedup, 53% of the maximum slowdown, and 31% of the harmonic speedup of a system with an unlimited amount of DRAM. Compared to a system with an all PCM main memory, we improve weighted speedup by 17%, reduce maximum slowdown by 21%, and improve harmonic speedup by 27%.
Implementation/Hardware Cost

• Requires a tag store in memory controller
  – We currently assume 36 KB of storage per 16 MB of DRAM
  – We are investigating ways to mitigate this overhead

• Requires a statistics store
  – To keep track of accesses and misses
Conclusions

- DRAM scalability is nearing its limit
  - Emerging memories (e.g. PCM) offer scalability
  - Problem: must address high latency and energy

- We propose a dynamic, row buffer locality-aware caching policy for hybrid memories
  - Cache rows which miss frequently in row buffer
  - Cache rows which are reused many times

- 17/21% perf/fairness improvement vs. all PCM
- Within 21/53% perf/fairness of all DRAM system
Thank you! Questions?
Backup Slides
Related Work

![Bar chart showing weighted speedup for different techniques: DIP, Probabilistic, Probabilistic+RBL, DAM-COUNT.](image)

- **DIP**: Generally performs worse than other techniques.
- **Probabilistic**: Shows moderate performance.
- **Probabilistic+RBL**: Improves upon the probabilistic technique.
- **DAM-COUNT**: Exhibits the highest weighted speedup, indicating superior performance.

6.2 Robustness to architectural configurations

- **Different numbers of cores**: Figure 10 illustrates the weighted speedup of our DAM-COUNT policy for 200 randomly-generated workloads on 2-, 4-, 8-, and 16-core systems, normalized to conventional caching. Results are sorted in terms of increasing normalized weighted speedup.

- **Our technique** achieves more performance benefit as the number of cores increases, yet there are a few workloads where our technique does not perform as well as the conventional caching baseline (on 2- and 4-core systems). This is because for some workloads composed of a large proportion of SMALL benchmarks, the working set both fits entirely in the DRAM cache and has high reuse. In such exceptional cases, all data can be promoted without tracking row reuse or row buffer locality information.

- **DRAM cache size**: Figure 11 shows the performance of conventional caching and DAM-COUNT for DRAM cache sizes from 64 MB to 512 MB averaged across 200 randomly-generated workloads consisting of 100% LARGE benchmarks, to exercise the DRAM cache. There are two things to note:

  - Even when a larger portion of the working set of workloads fits in the cache (e.g., 512 MB on Figure 11), DAM-COUNT outperforms conventional caching. This is because, compared to conventional caching, DAM-COUNT reduces the amount of channel contention and also accesses data from PCM, enabling channel-level parallelism.

- With 256 MB of DRAM, we are able to achieve within 21% of the weighted speedup, 53% of the maximum slowdown, and 31% of the harmonic speedup of a system with an unlimited amount of DRAM. Compared to a system with an all PCM main memory, we improve weighted speedup by 17%, reduce maximum slowdown by 21%, and improve harmonic speedup by 27%.
PCM Latency

Figure 9: Versus DRAM/PCM.

Figure 10: Number of cores.

Figure 11: Effects of DRAM size.

Figure 12: Effects of PCM latency.

Figure 13: Related techniques.

6.2 Robustness to architectural configurations

Different numbers of cores.

Figure 10 shows the weighted speedup of our DAM-COUNT policy for 200 randomly-generated workloads on 2-, 4-, 8-, and 16-core systems, normalized to conventional caching. Results are sorted in terms of increasing normalized weighted speedup.

Our technique achieves more performance benefit as the number of cores increases, yet there are a few workloads where our technique does not perform as well as the conventional caching baseline (on 2- and 4-core systems). This is because for some workloads composed of a large proportion of SMALL benchmarks, the working set both fits entirely in the DRAM cache and has high reuse. In such exceptional cases, all data can be promoted without tracking row reuse or row buffer locality information.

DRAM cache size.

Figure 11 shows the performance of conventional caching and DAM-COUNT for DRAM cache sizes from 64 MB to 512 MB averaged across 200 randomly-generated workloads consisting of 100% LARGE benchmarks, to exercise the DRAM cache. There are two things to note. First, even when a larger portion of the working set of workloads fits in the cache (e.g., 512 MB on Figure 11), DAM-COUNT outperforms conventional caching. This is because, compared to conventional caching, DAM-COUNT reduces the amount of channel contention and also accesses data from PCM, enabling channel-level parallelism.

With 256 MB of DRAM, we are able to achieve within 21% of the weighted speedup, 53% of the maximum slowdown, and 31% of the harmonic speedup of a system with an unlimited amount of DRAM. Compared to a system with an all PCM main memory, we improve weighted speedup by 17%, reduce maximum slowdown by 21%, and improve harmonic speedup by 27%.
workloads. For the all PCM and all DRAM systems, we model infinite memory capacity to fit the entire working sets of the workloads. Data are mapped to two ranks, totaling sixteen banks, across two memory controllers. In the all PCM system, benchmarks always pay the high cost (in terms of latency and energy) for accessing PCM on a row buffer miss. On the other hand, workloads benefit from the lower latencies and energy consumptions for accessing DRAM on a row buffer miss in the all DRAM system. A hybrid memory system employing a naïve caching scheme—such as conventional caching—can have worse performance and fairness compared to an all PCM system. This is due to the high amount of channel contention introduced and the inefficient use of the DRAM cache. The same system with the DAM-COUNT policy makes efficient use of the small DRAM cache by placing in it only the rows that exhibit high reuse and frequently miss in the row buffer.

With 256 MB of DRAM, we are able to achieve within 21% of the weighted speedup, 53% of the maximum slowdown, and 31% of the harmonic speedup of a system with an unlimited amount of DRAM. Compared to a system with an all PCM main memory, we improve weighted speedup by 17%, reduce maximum slowdown by 21%, and improve harmonic speedup by 27%.

6.2 Robustness to architectural configurations

Different numbers of cores. Figure 10 shows the weighted speedup of our DAM-COUNT policy for 200 randomly-generated workloads on 2-, 4-, 8-, and 16-core systems, normalized to conventional caching. Results are sorted in terms of increasing normalized weighted speedup. Our technique achieves more performance benefit as the number of cores increases, yet there are a few workloads where our technique does not perform as well as the conventional caching baseline (on 2- and 4-core systems). This is because for some workloads composed of a large proportion of SMALL benchmarks, the working set both fits entirely in the DRAM cache and has high reuse. In such exceptional cases, all data can be promoted without tracking row reuse or row buffer locality information.

DRAM cache size. Figure 11 shows the performance of conventional caching and DAM-COUNT for DRAM cache sizes from 64 MB to 512 MB averaged across 200 randomly-generated workloads consisting of 100% LARGE benchmarks, to exercise the DRAM cache. There are two things to note. First, even when a larger portion of the working set of workloads fits in the cache (e.g., 512 MB on Figure 11), DAM-COUNT outperforms conventional caching. This is because, compared to conventional caching, DAM-COUNT reduces the amount of channel contention and also accesses data from PCM, enabling channel-level parallelism.
Versus All DRAM and All PCM

**Figure 9:** Versus DRAM/PCM.

**Figure 10:** Number of cores.

**Figure 11:** Effects of DRAM size.

**Figure 12:** Effects of PCM latency.

**Figure 13:** Related techniques.

workloads. For the all PCM and all DRAM systems, we model infinite memory capacity to fit the entire working sets of the workloads. Data are mapped to two ranks, totaling sixteen banks, across two memory controllers.

In the all PCM system, benchmarks always pay the high cost (in terms of latency and energy) for accessing PCM on a row buffer miss. On the other hand, workloads benefit from the lower latencies and energy consumptions for accessing DRAM on a row buffer miss in the all DRAM system. A hybrid memory system employing a naïve caching scheme—such as conventional caching—can have worse performance and fairness compared to an all PCM system. This is due to the high amount of channel contention introduced and the inefficient use of the DRAM cache. The same system with the DAM-COUNT policy makes efficient use of the small DRAM cache by placing in it only the rows that exhibit high reuse and frequently miss in the row buffer.

With 256 MB of DRAM, we are able to achieve within 21% of the weighted speedup, 53% of the maximum slowdown, and 31% of the harmonic speedup of a system with an unlimited amount of DRAM. Compared to a system with an all PCM main memory, we improve weighted speedup by 17%, reduce maximum slowdown by 21%, and improve harmonic speedup by 27%.

**6.2 Robustness to architectural configurations**

Different numbers of cores. Figure 10 shows the weighted speedup of our DAM-COUNT policy for 200 randomly-generated workloads on 2-, 4-, 8-, and 16-core systems, normalized to conventional caching. Results are sorted in terms of increasing normalized weighted speedup.

Our technique achieves more performance benefit as the number of cores increases, yet there are a few workloads where our technique does not perform as well as the conventional caching baseline (on 2- and 4-core systems). This is because for some workloads composed of a large proportion of SMALL benchmarks, the working set both fits entirely in the DRAM cache and has high reuse. In such exceptional cases, all data can be promoted without tracking row reuse or row buffer locality information.

**DRAM cache size.** Figure 11 shows the performance of conventional caching and DAM-COUNT for DRAM cache sizes from 64 MB to 512 MB averaged across 200 randomly-generated workloads consisting of 100% LARGE benchmarks, to exercise the DRAM cache. There are two things to note. First, even when a larger portion of the working set of workloads fits in the cache (e.g., 512 MB on Figure 11), DAM-COUNT outperforms conventional caching. This is because, compared to conventional caching, DAM-COUNT reduces the amount of channel contention and also accesses data from PCM, enabling channel-level parallelism.
Performance vs. Statistics Store Size

(8 ways, LRU)

- 512-entry (0.2 KB)
- 1024-entry (0.4 KB)
- 2048-entry (0.8 KB)
- 4096-entry (1.6 KB)
- ∞-entry

IPC Normalized to All DRAM
Performance vs. Statistics Store Size

(8 ways, LRU)

- 512-entry (0.2 KB)
- 1024-entry (0.4 KB)
- 2048-entry (0.8 KB)
- 4096-entry (1.6 KB)
- ∞-entry

Within ~1% of infinite storage with 200 B of storage
All DRAM 8 Banks

![Bar Chart](chart.png)

- IPC Normalized to All DRAM with 8 Banks
- No Caching (All PCM), Conventional Caching, Best Static, Dynamic
All DRAM 16 Banks

IPC Normalized to All DRAM with 16 Banks

- No Caching (All PCM)
- Conventional Caching
- Best Static
- Dynamic
Simulation Parameters

<table>
<thead>
<tr>
<th>System</th>
<th>1–16 cores; 1 on-chip memory controller; 2 memory channels; 16/512 MB DRAM/PCM per core.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction window</td>
<td>128-entry out-of-order issue instruction window.</td>
</tr>
<tr>
<td>Fetch/execute/commit</td>
<td>3 instructions per cycle per core; maximum of 1 memory operation per cycle.</td>
</tr>
<tr>
<td>L1/L2 cache</td>
<td>32/512 KB per core; 4/8-way set associative; 128/128 B block size.</td>
</tr>
<tr>
<td>Memory controller</td>
<td>DDR3 800 MHz; 64-entry FR-FCFS request buffer per channel; 64-entry write buffer per channel.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIMMs</th>
<th>8 banks; 2 KB row buffer per bank; open row policy; 0.0016 pJ/b/bank/cycle static energy.</th>
<th>Row buffer</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (ns, cycles)</td>
<td>80, 400</td>
<td>Latency (ns, cycles)</td>
<td>40, 200</td>
<td>40, 200</td>
</tr>
<tr>
<td>Dynamic energy (pJ/b)</td>
<td>0.93</td>
<td>Dynamic energy (pJ/b)</td>
<td>1.02</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory devices</th>
<th>DRAM</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (ns, cycles)</td>
<td>80, 400</td>
<td>Latency (ns, cycles)</td>
<td>128, 640</td>
</tr>
<tr>
<td>Energy (pJ/b)</td>
<td>1.17</td>
<td>Energy (pJ/b)</td>
<td>2.47</td>
</tr>
</tbody>
</table>
Overview

• DRAM is reaching its scalability limits
  — Yet, memory capacity requirements are increasing

• Emerging memory devices offer scalability
  — Phase-change, resistive, ferroelectric, etc.
  — But, have worse latency/energy than DRAM

• We propose a scalable hybrid memory arch.
  — Use DRAM as a cache to phase change memory
  — Cache data based on row buffer locality and reuse
Methodology

• Core model
  – 3-wide issue with 128-entry instruction window
  – 32 KB L1 D-cache per core
  – 512 KB shared L2 cache per core

• Memory model
  – 16 MB DRAM / 512 MB PCM per core
    • Scaled based on workload trace size and access patterns to be smaller than working set
  – DDR3 800 MHz, single channel, 8 banks per device
  – Row buffer hit: 40 ns
  – Row buffer miss: 80 ns (DRAM); 128, 368 ns (PCM)
  – Migrate data at 2 KB row granularity
Outline

• Overview
• Motivation/Background
• Methodology
• Caching Policies
• Multicore Evaluation
• Conclusions
16-core Performance & Fairness

We find that a 16-way 32-set LRU-replacement stats store \( \sim 3.3 \text{ KB} \) with LRU replacement achieves performance within 4% of an unlimited-sized stats store.

Enabling promotion policies.

Figure 7 shows the steps taken by a memory controller to support promotion policies. When a memory request is issued from the CPU, the DRAM memory controller's tag store is indexed with the row address to see if the requested row resides in DRAM or PCM. The request is then placed in the appropriate request queue, where it will stay until scheduled.

If a request is scheduled in PCM, the address of the row containing the data is used to index the stats store and a value corresponding to the number of row accesses is incremented if the request is a read and increased by a value \( W \) if the request is a write (cf. Section 4.3); if the row content is not present in the row buffer, a value corresponding to the number of row buffer misses is also incremented.

After the requested memory block from PCM (whose access is on the critical path) is sent back to the CPU, the promotion policy is invoked. If the number of row accesses is greater than a threshold \( A \) and the number of row buffer misses is greater than a threshold \( M \), then the row is copied to the write buffer in the DRAM memory controller (i.e., promoted). A replacement policy in DRAM is used to determine a row to replace. Note that if the row was not modified while in DRAM, it does not need to be written back to PCM, otherwise, only the dirty contents of the row are sent to PCM to be written back.

Updating the stats store and invoking the promotion policy are not on the critical path, because they can be performed in parallel with accessing and transferring the critical word from PCM. If needed, the stats store's associative comparison logic can take multiple cycles to access, as the time taken to read data from PCM is on the order of hundreds of CPU cycles.

6 Experimental results

6.1 16-core results

We first analyze the performance and fairness of our technique, DAM-COUNT, on a 16-core system compared to three promotion policies: conventional caching, A-COUNT, and AM-COUNT. Note that for A- and AM-COUNT results, it was not feasible to find the best static \( A \) threshold for the large number of workloads surveyed. We instead show results for \( A = 4 \) and \( M = 2 \), which were found to be effective over a wide range of workloads. We show the A-COUNT results for reference to illustrate the limitations of a promotion policy that only considers row reuse (and not row buffer locality).

Data were collected for the initial 100 million cycles of a simulation run following a 10 million cycle warm-up period. The applications we use have relatively small working set sizes at reasonable simulation lengths and to ensure that we study the problem of data placement in hybrid memory systems properly, we set the DRAM size such that the working sets do not reside mainly in the DRAM cache. Longer simulations showed results consistent with shorter ones.

We find least frequently used to perform the best, however, the performance of least recently used follows very closely.