Computer Architecture
Lecture 16b: Emerging Memory Technologies

Prof. Onur Mutlu
ETH Zürich
Fall 2019
15 November 2019
Emerging Memory Technologies
Limits of Charge Memory

- Difficult charge placement and control
  - Flash: floating gate charge
  - DRAM: capacitor charge, transistor leakage

- Reliable sensing becomes difficult as charge storage unit size reduces
Solution 1: New Memory Architectures

- Overcome memory shortcomings with
  - Memory-centric system design
  - Novel memory architectures, interfaces, functions
  - Better waste management (efficient utilization)

- Key issues to tackle
  - Enable reliability at low cost → high capacity
  - Reduce energy
  - Reduce latency
  - Improve bandwidth
  - Reduce waste (capacity, bandwidth, latency)
  - Enable computation close to data
Solution 1: New Memory Architectures

- Liu+，“An Experimental Study of Data Retention Behavior in Modern DRAM Devices,” ISCA 2013.
- Seshadri+, "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," MICRO 2013.
- Luo+, "Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost," DNS 2014.
- Seshadri+, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-uniform Strided Accesses," MICRO 2015.
- Khan+, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM," IEEE CAL 2016.
- Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Dense," DATE 2017.
- Khan+, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," MICRO 2017.
Solution 2: Emerging Memory Technologies

- Some emerging **resistive** memory technologies seem more scalable than DRAM (and they are non-volatile)
  
  - Example: Phase Change Memory
    - Data stored by changing phase of material
    - Data read by detecting material’s resistance
    - Expected to scale to 9nm (2022 [ITRS 2009])
    - Prototyped at 20nm (Raoux+, IBM JRD 2008)
    - Expected to be denser than DRAM: can store multiple bits/cell

- But, emerging technologies have (many) shortcomings
  - Can they be enabled to replace/augment/surpass DRAM?
Solution 2: Emerging Memory Technologies

- Zhao+, “FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems,” MICRO 2014.
Charge vs. Resistive Memories

- Charge Memory (e.g., DRAM, Flash)
  - Write data by capturing charge Q
  - Read data by detecting voltage V

- Resistive Memory (e.g., PCM, STT-MRAM, memristors)
  - Write data by pulsing current dQ/dt
  - Read data by detecting resistance R
Promising Resistive Memory Technologies

- **PCM**
  - Inject current to change *material phase*
  - Resistance determined by phase

- **STT-MRAM**
  - Inject current to change *magnet polarity*
  - Resistance determined by polarity

- **Memristors/RRAM/ReRAM**
  - Inject current to change *atomic structure*
  - Resistance determined by atom distance
What is Phase Change Memory?

- Phase change material (chalcogenide glass) exists in two states:
  - Amorphous: Low optical reflexivity and high electrical resistivity
  - Crystalline: High optical reflexivity and low electrical resistivity

PCM is resistive memory: High resistance (0), Low resistance (1)
PCM cell can be switched between states reliably and quickly
How Does PCM Work?

- **Write**: change phase via current injection
  - **SET**: sustained current to heat cell above $T_{\text{cryst}}$
  - **RESET**: cell heated above $T_{\text{melt}}$ and quenched
- **Read**: detect phase via material resistance
  - amorphous/crystalline

![Diagram showing temperature over time for SET and RESET processes with corresponding resistance values.](Photo Courtesy: Bipin Rajendran, IBM  Slide Courtesy: Moinuddin Qureshi, IBM)
Opportunity: PCM Advantages

- **Scales better than DRAM, Flash**
  - Requires current pulses, which scale linearly with feature size
  - Expected to scale to 9nm (2022 [ITRS])
  - Prototyped at 20nm (Raoux+, IBM JRD 2008)

- **Can be denser than DRAM**
  - Can store multiple bits per cell due to large resistance range
  - Prototypes with 2 bits/cell in ISSCC’ 08, 4 bits/cell by 2012

- **Non-volatile**
  - Retain data for >10 years at 85C

- **No refresh needed, low idle power**
PCM Resistance → Value

Cell value:

1   0

Cell resistance
Multi-Level Cell PCM

- Multi-level cell: more than 1 bit per cell
  - Further increases density by 2 to 4x [Lee+, ISCA'09]

- But MLC-PCM also has drawbacks
  - Higher latency and energy than single-level cell PCM
MLC-PCM Resistance $\rightarrow$ Value

Cell value:

11 10 01 00

Bit 1  Bit 0

Cell resistance
MLC-PCM Resistance $\rightarrow$ Value

*Less margin between values*

$\rightarrow$ need more precise sensing/modification of cell contents

$\rightarrow$ higher latency/energy ($\sim$2x for reads and 4x for writes)
Phase Change Memory Properties

- Surveyed prototypes from 2003-2008 (ITRS, IEDM, VLSI, ISSCC)
- Derived PCM parameters for F=90nm

## Table 1. Technology survey.

<table>
<thead>
<tr>
<th>Parameter*</th>
<th>Horri6</th>
<th>Ahn12</th>
<th>Bedeschi13</th>
<th>Oh14</th>
<th>Pellizer15</th>
<th>Chen5</th>
<th>Kang16</th>
<th>Bedeschi9</th>
<th>Lee10</th>
<th>Lee2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process, F(nm)</td>
<td>**</td>
<td>120</td>
<td>180</td>
<td>120</td>
<td>90</td>
<td>**</td>
<td>100</td>
<td>90</td>
<td>90</td>
<td>90</td>
</tr>
<tr>
<td>Array size (Mbytes)</td>
<td>**</td>
<td>64</td>
<td>8</td>
<td>64</td>
<td>**</td>
<td>**</td>
<td>256</td>
<td>256</td>
<td>512</td>
<td>**</td>
</tr>
<tr>
<td>Material</td>
<td>GST, N-d</td>
<td>GST, N-d</td>
<td>GST</td>
<td>GST</td>
<td>GST</td>
<td>GS, N-d</td>
<td>GST</td>
<td>GST</td>
<td>GST</td>
<td>GST, N-d</td>
</tr>
<tr>
<td>Cell size (µm²)</td>
<td>**</td>
<td>0.290</td>
<td>0.290</td>
<td>**</td>
<td>0.097</td>
<td>60 nm²</td>
<td>0.166</td>
<td>0.097</td>
<td>0.047</td>
<td>0.065 to 0.097</td>
</tr>
<tr>
<td>Cell size, F²</td>
<td>**</td>
<td>20.1</td>
<td>9.0</td>
<td>**</td>
<td>12.0</td>
<td>**</td>
<td>16.6</td>
<td>12.0</td>
<td>5.8</td>
<td>9.0 to 12.0</td>
</tr>
<tr>
<td>Access device</td>
<td>**</td>
<td>**</td>
<td>BJT</td>
<td>FET</td>
<td>BJT</td>
<td>**</td>
<td>FET</td>
<td>BJT</td>
<td>Diode</td>
<td>BJT</td>
</tr>
<tr>
<td>Read time (ns)</td>
<td>**</td>
<td>70</td>
<td>48</td>
<td>68</td>
<td>**</td>
<td>**</td>
<td>62</td>
<td>**</td>
<td>55</td>
<td>48</td>
</tr>
<tr>
<td>Read current (µA)</td>
<td>**</td>
<td>**</td>
<td>40</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
<tr>
<td>Read voltage (V)</td>
<td>**</td>
<td>3.0</td>
<td>1.0</td>
<td>1.8</td>
<td>1.6</td>
<td>**</td>
<td>1.8</td>
<td>**</td>
<td>1.8</td>
<td>1.0</td>
</tr>
<tr>
<td>Read power (µW)</td>
<td>**</td>
<td>**</td>
<td>40</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>40</td>
</tr>
<tr>
<td>Read energy (pJ)</td>
<td>**</td>
<td>**</td>
<td>2.0</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>2.0</td>
</tr>
<tr>
<td>Set time (ns)</td>
<td>100</td>
<td>150</td>
<td>150</td>
<td>180</td>
<td>**</td>
<td>80</td>
<td>300</td>
<td>**</td>
<td>400</td>
<td>150</td>
</tr>
<tr>
<td>Set current (µA)</td>
<td>200</td>
<td>**</td>
<td>300</td>
<td>200</td>
<td>**</td>
<td>55</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>150</td>
</tr>
<tr>
<td>Set voltage (V)</td>
<td>**</td>
<td>**</td>
<td>2.0</td>
<td>**</td>
<td>**</td>
<td>1.25</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>1.2</td>
</tr>
<tr>
<td>Set power (µW)</td>
<td>**</td>
<td>**</td>
<td>300</td>
<td>**</td>
<td>**</td>
<td>34.4</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>90</td>
</tr>
<tr>
<td>Set energy (pJ)</td>
<td>**</td>
<td>**</td>
<td>45</td>
<td>**</td>
<td>**</td>
<td>2.8</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>13.5</td>
</tr>
<tr>
<td>Reset time (ns)</td>
<td>50</td>
<td>10</td>
<td>40</td>
<td>10</td>
<td>**</td>
<td>60</td>
<td>50</td>
<td>**</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>Reset current (µA)</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>400</td>
<td>90</td>
<td>600</td>
<td>300</td>
<td>600</td>
<td>300</td>
</tr>
<tr>
<td>Reset voltage (V)</td>
<td>**</td>
<td>**</td>
<td>2.7</td>
<td>**</td>
<td>1.8</td>
<td>1.6</td>
<td>**</td>
<td>1.6</td>
<td>**</td>
<td>1.6</td>
</tr>
<tr>
<td>Reset power (µW)</td>
<td>**</td>
<td>**</td>
<td>1620</td>
<td>**</td>
<td>**</td>
<td>80.4</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>480</td>
</tr>
<tr>
<td>Reset energy (pJ)</td>
<td>**</td>
<td>**</td>
<td>64.8</td>
<td>**</td>
<td>**</td>
<td>4.8</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>19.2</td>
</tr>
<tr>
<td>Write endurance (MLC)</td>
<td>$10^7$</td>
<td>$10^9$</td>
<td>$10^8$</td>
<td>**</td>
<td>$10^8$</td>
<td>$10^4$</td>
<td>**</td>
<td>$10^5$</td>
<td>$10^5$</td>
<td>$10^8$</td>
</tr>
</tbody>
</table>

* BJT: bipolar junction transistor; FET: field-effect transistor; GST: Ge$_2$Sb$_2$Te$_5$; MLC: multilevel cells; N-d: nitrogen doped.

** This information is not available in the publication cited.
Phase Change Memory Properties: Latency

- Latency comparable to, but slower than DRAM

Read Latency
- 50ns: 4x DRAM, $10^{-3}$x NAND Flash

Write Latency
- 150ns: 12x DRAM

Write Bandwidth
- 5-10 MB/s: 0.1x DRAM, 1x NAND Flash

Phase Change Memory Properties

- **Dynamic Energy**
  - 40 uA Rd, 150 uA Wr
  - **2-43x DRAM, 1x NAND Flash**

- **Endurance**
  - Writes induce phase change at 650°C
  - Contacts degrade from thermal expansion/contraction
  - 10^8 writes per cell
  - **10^-8x DRAM, 10^3x NAND Flash**

- **Cell Size**
  - 9-12F^2 using BJT, single-level cells
  - **1.5x DRAM, 2-3x NAND** (will scale with feature size, MLC)
Phase Change Memory: Pros and Cons

 Pros over DRAM
- Better technology scaling (capacity and cost)
- Non volatile → Persistent
- Low idle power (no refresh)

 Cons
- Higher latencies: ~4-15x DRAM (especially write)
- Higher active energy: ~2-50x DRAM (especially write)
- Lower endurance (a cell dies after ~$10^8$ writes)
- Reliability issues (resistance drift)

 Challenges in enabling PCM as DRAM replacement/helper:
- Mitigate PCM shortcomings
- Find the right way to place PCM in the system
PCM-based Main Memory (I)

- How should PCM-based (main) memory be organized?

- **Hybrid PCM+DRAM** [Qureshi+ ISCA’09, Dhiman+ DAC’09]:
  - How to partition/migrate data between PCM and DRAM
PCM-based Main Memory (II)

- How should PCM-based (main) memory be organized?

- **Pure PCM main memory** [Lee et al., ISCA’09, Top Picks’10]:
  - How to redesign entire hierarchy (and cores) to overcome PCM shortcomings
An Initial Study: Replace DRAM with PCM

  - Surveyed prototypes from 2003-2008 (e.g. IEDM, VLSI, ISSCC)
  - Derived “average” PCM parameters for F=90nm

**Density**
- $9 - 12F^2$ using BJT
- $1.5 \times$ DRAM

**Latency**
- $50\text{ns} \text{ Rd}, 150\text{ns} \text{ Wr}$
- $4\times, 12\times$ DRAM

**Endurance**
- $1E+08$ writes
- $1E-08 \times$ DRAM

**Energy**
- $40\mu\text{A} \text{ Rd}, 150\mu\text{A} \text{ Wr}$
- $2\times, 43\times$ DRAM
Results: Naïve Replacement of DRAM with PCM

- Replace DRAM with PCM in a 4-core, 4MB L2 system
- PCM organized the same as DRAM: row buffers, banks, peripherals
- 1.6x delay, 2.2x energy, 500-hour average lifetime

Architecting PCM to Mitigate Shortcomings

- Idea 1: Use multiple narrow row buffers in each PCM chip
  → Reduces array reads/writes → better endurance, latency, energy

- Idea 2: Write into array at cache block or word granularity
  → Reduces unnecessary wear
Results: Architected PCM as Main Memory

- 1.2x delay, 1.0x energy, 5.6-year average lifetime
- Scaling improves energy, endurance, density

![Bar chart showing PCM Performance](image)

- Caveat 1: Worst-case lifetime is much shorter (no guarantees)
- Caveat 2: Intensive applications see large performance and energy hits
- Caveat 3: Optimistic PCM parameters?
Required Reading: PCM As Main Memory


Architecting Phase Change Memory as a Scalable DRAM Alternative

Benjamin C. Lee†  Engin Ipek†  Onur Mutlu‡  Doug Burger‡

†Computer Architecture Group
Microsoft Research
Redmond, WA
{blee, ipek, dburger}@microsoft.com

‡Computer Architecture Laboratory
Carnegie Mellon University
Pittsburgh, PA
onur@cmu.edu
More on PCM As Main Memory (II)


Phase-Change Technology and the Future of Main Memory
STT-MRAM as Main Memory

- Magnetic Tunnel Junction (MTJ) device
  - Reference layer: Fixed magnetic orientation
  - Free layer: Parallel or anti-parallel

- Magnetic orientation of the free layer determines logical state of device
  - High vs. low resistance

- Write: Push large current through MTJ to change orientation of free layer
- Read: Sense current flow

STT-MRAM: Pros and Cons

- **Pros over DRAM**
  - Better technology scaling (capacity and cost)
  - Non volatile → Persistent
  - Low idle power (no refresh)

- **Cons**
  - Higher write latency
  - Higher write energy
  - Poor density (currently)
  - Reliability?

- **Another level of freedom**
  - Can trade off non-volatility for lower write latency/energy (by reducing the size of the MTJ)
Architected STT-MRAM as Main Memory

- 4-core, 4GB main memory, multiprogrammed workloads
- ~6% performance loss, ~60% energy savings vs. DRAM

More on STT-MRAM as Main Memory

- Emre Kultursay, Mahmut Kandemir, Anand Sivasubramaniam, and Onur Mutlu

"Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative"

Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Austin, TX, April 2013. Slides (pptx) (pdf)
A More Viable Approach: Hybrid Memory Systems

Yoon+, “Row Buffer Locality Aware Caching Policies for Hybrid Memories,” ICCD 2012 Best Paper Award.

Hardware/software manage data allocation and movement to achieve the best of multiple technologies
Challenge and Opportunity

Providing the Best of Multiple Metrics with Multiple Memory Technologies
Challenge and Opportunity

Heterogeneous, Configurable, Programmable Memory Systems
Hybrid Memory Systems: Issues

- Cache vs. Main Memory
- Granularity of Data Move/Manage-ment: Fine or Coarse
- Hardware vs. Software vs. HW/SW Cooperative
- When to migrate data?
- How to design a scalable and efficient large cache?
- ...
One Option: DRAM as a Cache for PCM

- PCM is main memory; DRAM caches memory rows/blocks
  - Benefits: Reduced latency on DRAM cache hit; write filtering
- Memory controller hardware manages the DRAM cache
  - Benefit: Eliminates system software overhead

Three issues:
- What data should be placed in DRAM versus kept in PCM?
- What is the granularity of data movement?
- How to design a low-cost hardware-managed DRAM cache?

Two idea directions:
- Locality-aware data placement [Yoon+, ICCD 2012]
- Cheap tag stores and dynamic granularity [Meza+, IEEE CAL 2012]
**DRAM as a Cache for PCM**

- **Goal:** Achieve the best of both DRAM and PCM/NVM
  - Minimize amount of DRAM w/o sacrificing performance, endurance
  - DRAM as cache to tolerate PCM latency and write bandwidth
  - PCM as main memory to provide large capacity at good cost and power

---

Write Filtering Techniques

- Lazy Write: Pages from disk installed only in DRAM, not PCM
- Partial Writes: Only dirty lines from DRAM page written back
- Page Bypass: Discard pages with poor reuse on DRAM eviction

Results: DRAM as PCM Cache (I)

- Simulation of 16-core system, 8GB DRAM main-memory at 320 cycles, HDD (2 ms) with Flash (32 us) with Flash hit-rate of 99%
- Assumption: PCM 4x denser, 4x slower than DRAM
- DRAM block size = PCM page size (4kB)

Results: DRAM as PCM Cache (II)

- PCM-DRAM Hybrid performs similarly to similar-size DRAM
- Significant energy savings with PCM-DRAM Hybrid
- Average lifetime: 9.7 years (no guarantees)

More on DRAM-PCM Hybrid Memory

- **Scalable High-Performance Main Memory System Using Phase-Change Memory Technology.**
  Moinuddin K. Qureshi, Viji Srinivasan, and Jude A. Rivers
  *Appears in the International Symposium on Computer Architecture (ISCA) 2009.*

Scalable High Performance Main Memory System Using Phase-Change Memory Technology

Moinuddin K. Qureshi  Vijayalakshmi Srinivasan  Jude A. Rivers

IBM Research
T. J. Watson Research Center, Yorktown Heights NY 10598
{mkquresh, viji, jarivers}@us.ibm.com
Data Placement in Hybrid Memory

- Memory A is fast, but small
- Load should be balanced on both channels?
- Page migrations have performance and energy overhead

Which memory do we place each page in, to maximize system performance?

- Memory A is fast, but small
- Load should be balanced on both channels?
- Page migrations have performance and energy overhead
Data Placement Between DRAM and PCM

- Idea: Characterize data access patterns and guide data placement in hybrid memory

- Streaming accesses: As fast in PCM as in DRAM

- Random accesses: Much faster in DRAM

- Idea: Place random access data with some reuse in DRAM; streaming data in PCM

Key Observation & Idea

• Row buffers exist in both DRAM and PCM
  – Row hit latency similar in DRAM & PCM [Lee+ ISCA’09]
  – Row miss latency small in DRAM, large in PCM

• Place data in DRAM which
  – is likely to miss in the row buffer (low row buffer locality) → miss penalty is smaller in DRAM
    AND
  – is reused many times → cache only the data worth the movement cost and DRAM space
More on Hybrid Memory Data Placement

- HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael Harding, and Onur Mutlu,
  "Row Buffer Locality Aware Caching Policies for Hybrid Memories"
  *Proceedings of the 30th IEEE International Conference on Computer Design (ICCD)*, Montreal, Quebec, Canada, September 2012. Slides (pptx) (pdf)
Weaknesses of Existing Solutions

- They are all heuristics that consider only a limited part of memory access behavior
- Do not directly capture the overall system performance impact of data placement decisions

Example: None capture memory-level parallelism (MLP)
- Number of concurrent memory requests from the same application when a page is accessed
- Affects how much page migration helps performance
Importance of Memory-Level Parallelism

Before migration:
- requests to Page 1
  - time
  - Migrating one page reduces stall time by $T$

After migration:
- requests to Page 1
  - Mem. A

Before migration:
- requests to Page 2
  - Mem. B
- requests to Page 3
  - Mem. B

After migration:
- requests to Page 2
  - Mem. A
- requests to Page 3
  - Mem. B

**Time**
- **Must migrate two pages**
  - to reduce stall time by $T$: migrating one page alone does not help

Page migration decisions **need to consider MLP**
Our Goal [CLUSTER 2017]

A **generalized** mechanism that

1. Directly estimates the **performance benefit** of migrating a page between any two types of memory

2. Places **only** the **performance-critical data** in the fast memory
Utility-Based Hybrid Memory Management

- A memory manager that works for *any* hybrid memory
  - e.g., DRAM-NVM, DRAM-RLDRAM

- **Key Idea**
  - For each page, use *comprehensive* characteristics to calculate estimated *utility* (i.e., performance impact) of migrating page from one memory to the other in the system
  - Migrate only pages with the highest *utility* (i.e., pages that improve system performance the most when migrated)

Key Mechanisms of UH-MEM

- For each page, estimate **utility** using a **performance model**
  - **Application stall time reduction**
    How much would migrating a page benefit the performance of the application that the page belongs to?
  - **Application performance sensitivity**
    How much does the improvement of a single application’s performance increase the overall system performance?
    \[ Utility = \Delta StallTime_i \times Sensitivity_i \]

- **Migrate** only pages whose utility exceed the migration threshold from slow memory to fast memory

- Periodically **adjust migration threshold**
Results: System Performance

UH-MEM improves system performance over the best state-of-the-art hybrid memory manager.
Results: Sensitivity to Slow Memory Latency

- We vary $t_{RCD}$ and $t_{WR}$ of the slow memory

<table>
<thead>
<tr>
<th>Slow Memory Latency Multiplier</th>
<th>Weighted Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{RCD}: x3.0</td>
<td>13%</td>
</tr>
<tr>
<td>t_{RCD}: x4.0</td>
<td>13%</td>
</tr>
<tr>
<td>t_{RCD}: x4.5</td>
<td>8%</td>
</tr>
<tr>
<td>t_{WR}: x3.0</td>
<td>6%</td>
</tr>
<tr>
<td>t_{WR}: x4.0</td>
<td>14%</td>
</tr>
<tr>
<td>t_{WR}: x12</td>
<td>13%</td>
</tr>
<tr>
<td>t_{WR}: x16</td>
<td>13%</td>
</tr>
<tr>
<td>t_{WR}: x20</td>
<td></td>
</tr>
</tbody>
</table>

UH-MEM improves system performance for a wide variety of hybrid memory systems
More on UH-MEM

- Yang Li, Saugata Ghose, Jongmoo Choi, Jin Sun, Hui Wang, and Onur Mutlu,

"Utility-Based Hybrid Memory Management"


[Slides (pptx) (pdf)]
Challenge and Opportunity

Enabling an Emerging Technology to Augment DRAM

Managing Hybrid Memories
Another Challenge

Designing Effective Large (DRAM) Caches
One Problem with Large DRAM Caches

- A large DRAM cache requires a large metadata (tag + block-based information) store
- How do we design an efficient DRAM cache?
Idea 1: Tags in Memory

- Store tags in the same row as data in DRAM
  - Store metadata in same row as their data
  - Data and metadata can be accessed together

- Benefit: No on-chip tag storage overhead

- Downsides:
  - Cache hit determined only after a DRAM access
  - Cache hit requires two DRAM accesses
Idea 2: Cache Tags in SRAM

- Recall Idea 1: Store all metadata in DRAM
  - To reduce metadata storage overhead

- Idea 2: **Cache in on-chip SRAM frequently-accessed metadata**
  - Cache only a small amount to keep SRAM size small
Idea 3: Dynamic Data Transfer Granularity

- Some applications benefit from caching more data
  - They have good spatial locality
- Others do not
  - Large granularity wastes bandwidth and reduces cache utilization

Idea 3: Simple dynamic caching granularity policy

- Cost-benefit analysis to determine best DRAM cache block size
- Group main memory into sets of rows
- Different sampled row sets follow different fixed caching granularities
- The rest of main memory follows the best granularity
  - Cost–benefit analysis: access latency versus number of cachings
  - Performed every quantum
TIMBER Performance

Reduced channel contention and improved spatial locality

TIMBER Energy Efficiency

Fewer migrations reduce transmitted data and channel contention

On Large DRAM Cache Design

- Justin Meza, Jichuan Chang, HanBin Yoon, Onur Mutlu, and Parthasarathy Ranganathan,
  "Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management"
  IEEE Computer Architecture Letters (CAL), February 2012.

Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management

Justin Meza*  Jichuan Chang†  HanBin Yoon*  Onur Mutlu*  Parthasarathy Ranganathan†
* Carnegie Mellon University  † Hewlett-Packard Labs
{meza,hanbinyoon,onur}@cmu.edu  {jichuan.chang,partha.ranganathan}@hp.com
DRAM Caches: Many Recent Options

Table 1: Summary of Operational Characteristics of Different State-of-the-Art DRAM Cache Designs – We assume perfect way prediction for Unison Cache. Latency is relative to the access time of the off-package DRAM (see Section 6 for baseline latencies). We use different colors to indicate the high (dark red), medium (white), and low (light green) overhead of a characteristic.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>DRAM Cache Hit</th>
<th>DRAM Cache Miss</th>
<th>Replacement Traffic</th>
<th>Replacement Decision</th>
<th>Large Page Caching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unison [32]</td>
<td>In-package traffic: 128 B (data + tag read and update) Latency: ~1x</td>
<td>In-package traffic: 96 B (spec. data + tag read) Latency: ~2x</td>
<td>On every miss Footprint size [31]</td>
<td>Hardware managed, set-associative, LRU</td>
<td>Yes</td>
</tr>
<tr>
<td>Alloy [50]</td>
<td>In-package traffic: 96 B (data + tag read) Latency: ~1x</td>
<td>In-package traffic: 96 B (spec. data + tag read) Latency: ~2x</td>
<td>On some misses Cacheline size (64 B)</td>
<td>Hardware managed, direct-mapped, stochastic [20]</td>
<td>Yes</td>
</tr>
<tr>
<td>TDC [38]</td>
<td>In-package traffic: 64 B Latency: ~1x TLB coherence</td>
<td>In-package traffic: 0 B Latency: ~1x TLB coherence</td>
<td>On every miss Footprint size [28]</td>
<td>Hardware managed, fully-associative, FIFO</td>
<td>No</td>
</tr>
<tr>
<td>HMA [44]</td>
<td>In-package traffic: 64 B Latency: ~1x</td>
<td>In-package traffic: 0 B Latency: ~1x</td>
<td>Software managed, high replacement cost</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Banshee (This work)</td>
<td>In-package traffic: 64 B Latency: ~1x</td>
<td>In-package traffic: 0 B Latency: ~1x</td>
<td>Only for hot pages Page size (4 KB)</td>
<td>Hardware managed, set-associative, frequency based</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Banshee [MICRO 2017]

- Tracks presence in cache using TLB and Page Table
  - No tag store needed for DRAM cache
  - Enabled by a new lightweight lazy TLB coherence protocol

- New bandwidth-aware frequency-based replacement policy
More on Banshee

- Xiangyao Yu, Christopher J. Hughes, Nadathur Satish, Onur Mutlu, and Srinivas Devadas,

"**Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation**"

*Proceedings of the 50th International Symposium on Microarchitecture (MICRO)*, Boston, MA, USA, October 2017.

---

**Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation**

Xiangyao Yu\(^1\)  Christopher J. Hughes\(^2\)  Nadathur Satish\(^2\)  Onur Mutlu\(^3\)  Srinivas Devadas\(^1\)

\(^1\)MIT  \(^2\)Intel Labs  \(^3\)ETH Zürich
Other Opportunities with Emerging Technologies

- **Merging of memory and storage**
  - e.g., a single interface to manage all data

- **New applications**
  - e.g., ultra-fast checkpoint and restore

- **More robust system design**
  - e.g., reducing data loss

- **Processing tightly-coupled with memory**
  - e.g., enabling efficient search and filtering
TWO-LEVEL STORAGE MODEL

CPU

MEMORY

DRAM

STORAGE

Ld/St

FILE I/O

VOLATILE

FAST

BYTE ADDR

NONVOLATILE

SLOW

BLOCK ADDR
Non-volatile memories combine characteristics of memory and storage
The traditional two-level storage model is a bottleneck with NVM

- **Volatile** data in memory → a **load/store** interface
- **Persistent** data in storage → a **file system** interface
- Problem: Operating system (OS) and file system (FS) code to locate, translate, buffer data become performance and energy bottlenecks with fast NVM stores.
Unified Memory and Storage with NVM

- **Goal:** Unify memory and storage management in a single unit to eliminate wasted work to locate, transfer, and translate data
  - Improves both energy and performance
  - Simplifies programming model as well

**Unified Memory/Storage**

- **Persistent Memory Manager**
- **Processor and caches**
- **Load/Store**
- **Feedback**
- **Persistent (e.g., Phase-Change) Memory**

PERSISTENT MEMORY

Provides an opportunity to manipulate persistent data directly
The Persistent Memory Manager (PMM)

PMM uses access and hint information to allocate, locate, migrate and access data in the heterogeneous array of devices.

```c
int main(void) {
    // data in file.dat is persistent
    FILE myData = "file.dat";
    myData = new int[64];
}

void updateValue(int n, int value) {
    FILE myData = "file.dat";
    myData[n] = value; // value is persistent
}
```
The Persistent Memory Manager (PMM)

- Exposes a load/store interface to access persistent data
  - Applications can directly access persistent memory → no conversion, translation, location overhead for persistent data

- Manages data placement, location, persistence, security
  - To get the best of multiple forms of storage

- Manages metadata storage and retrieval
  - This can lead to overheads that need to be managed

- Exposes hooks and interfaces for system software
  - To enable better data placement and management decisions

A persistent memory exposes a large, persistent address space
- But it may use many different devices to satisfy this goal
- From fast, low-capacity volatile DRAM to slow, high-capacity non-volatile HDD or Flash
- And other NVM devices in between

Performance and energy can benefit from good placement of data among these devices
- Utilizing the strengths of each device and avoiding their weaknesses, if possible
- For example, consider two important application characteristics: locality and persistence
Efficient Data Mapping among Heterogeneous Devices
Columns in a column store that are scanned through only infrequently → place on Flash
Applications or system software can provide hints for data placement.
Evaluated Systems

- **HDD Baseline**
  - Traditional system with volatile DRAM memory and persistent HDD storage
  - Overheads of operating system and file system code and buffering

- **NVM Baseline (NB)**
  - Same as HDD Baseline, but HDD is replaced with NVM
  - Still has OS/FS overheads of the two-level storage model

- **Persistent Memory (PM)**
  - Uses only NVM (no DRAM) to ensure full-system persistence
  - All data accessed using loads and stores
  - Does not waste time on system calls
  - Data is manipulated directly on the NVM device
Performance Benefits of a Single-Level Store

Energy Benefits of a Single-Level Store

~16X

0.065 ~5X 0.013

HDD 2-level NVM 2-level Persistent Memory

On Persistent Memory Benefits & Challenges

- Justin Meza, Yixin Luo, Samira Khan, Jishen Zhao, Yuan Xie, and Onur Mutlu,

"A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory"

Proceedings of the 5th Workshop on Energy-Efficient Design (WEED), Tel-Aviv, Israel, June 2013. Slides (pptx) Slides (pdf)
Challenge and Opportunity

Combined Memory & Storage
A Unified Interface to All Data
One Key Challenge in Persistent Memory

- How to ensure consistency of system/data if all memory is persistent?

- Two extremes
  - Programmer transparent: Let the system handle it
  - Programmer only: Let the programmer handle it

- Many alternatives in-between...
CRASH CONSISTENCY PROBLEM

Add a node to a linked list

2. Link to prev
1. Link to next

System crash can result in inconsistent memory state
CURRENT SOLUTIONS

Explicit interfaces to manage consistency
– NV-Heaps [ASPLOS’11], BPFS [SOSP’09], Mnemosyne [ASPLOS’11]

```c
AtomicBegin {
    Insert a new node;
} AtomicEnd;
```

Limits adoption of NVM
Have to rewrite code with clear partition between volatile and non-volatile data

Burden on the programmers
CURRENT SOLUTIONS

Explicit interfaces to manage consistency
- NV-Heaps [ASPLOS’11], BPFS [SOSP’09], Mnemosyne [ASPLOS’11]

Example Code

update a node in a persistent hash table

```c
void hashtable_update(hashtable_t* ht,
                      void *key, void *data)
{
    list_t* chain = get_chain(ht, key);
    pair_t* pair;
    pair_t updatePair;
    updatePair.first = key;
    pair = (pair_t*) list_find(chain, &updatePair);
    pair->second = data;
}
```
void TMhashtable_update(TMARCDECL hashtable_t* ht, void *key, void*data){
    list_t* chain = get_chain(ht, key);
    pair_t* pair;
    pair_t updatePair;
    updatePair.first = key;
    pair = (pair_t*) TMLIST_FIND(chain, &updatePair);
    pair->second = data;
}
Manual declaration of persistent components

```c
void TMhashtable_update(TMARCGDECL hashtable_t* ht, void *key, void*data){
    list_t* chain = get_chain(ht, key);
    pair_t* pair; 
    pair_t updatePair;
    updatePair.first = key;
    pair = (pair_t*) TMLIST_FIND(chain, &updatePair);
    pair->second = data;
}
```
void TMhashtable_update(TMARCGDECL hashtable_t* ht, void *key, void*data){
    list_t* chain = get_chain(ht, key);
    pair_t* pair;
    pair_t updatePair;
    updatePair.first = key;
    pair = (pair_t*) TMLIST_FIND (chain, &updatePair);
    pair->second = data;
}
CURRENT SOLUTIONS

Manual declaration of persistent components

```c
void TMhashtable_update(TMARGDECL hashtable_t* ht, void *key, void*data){
    list_t* chain = get_chain(ht, key);
    pair_t* pair;
    pair_t updatePair;
    updatePair.first = key;
    pair = (pair_t*)TMLIST_FIND(chain, &updatePair);
    pair->second = data;
}
```

Need a new implementation

Third party code can be inconsistent
CURRENT SOLUTIONS

Manual declaration of persistent components

```c
void TM_hashtable_update(TMARCDECL hashtable_t* ht, void *key, void*data){
    list_t* chain = get_chain(ht, key);
    pair_t* pair;
    pair_t updatePair;
    updatePair.first = key;
    pair = (pair_t*)TMLIST_FIND(chain, &updatePair);
    pair->second = data;
}
```

Need a new implementation

Prohibited Operation

Third party code can be inconsistent

Burden on the programmers
OUR APPROACH: ThyNVM

Goal:
Software transparent consistency in persistent memory systems

Key Idea:
Periodically checkpoint state; recover to previous checkpt on crash
ThyNVM: Summary

A new hardware-based checkpointing mechanism

- **Checkpoints** at *multiple granularities* to reduce both checkpointing latency and metadata overhead

- **Overlaps** checkpointing and execution to reduce checkpointing latency

- **Adapts** to *DRAM and NVM* characteristics

Performs within **4.9%** of an *idealized DRAM* with zero cost consistency
2. OVERLAPPING CHECKPOINTING AND EXECUTION
More About ThyNVM

- Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu, and Onur Mutlu,

"ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems"

Proceedings of the 48th International Symposium on Microarchitecture (MICRO), Waikiki, Hawaii, USA, December 2015.

[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]

[Source Code]

ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems

Jinglei Ren*† Jishen Zhao‡ Samira Khan†† Jongmoo Choi††† Yongwei Wu* Onur Mutlu†

†Carnegie Mellon University   *Tsinghua University
‡University of California, Santa Cruz   †University of Virginia   ††Dankook University
Another Key Challenge in Persistent Memory

Programming Ease to Exploit Persistence
Tools/Libraries to Help Programmers

- Himanshu Chauhan, Irina Calciu, Vijay Chidambaram, Eric Schkufza, Onur Mutlu, and Pratap Subrahmanyanam, "NVMove: Helping Programmers Move to Byte-Based Persistence"
  
  *Proceedings of the 4th Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW), Savannah, GA, USA, November 2016. [Slides (pptx) (pdf)]

NVMove: Helping Programmers Move to Byte-Based Persistence

Himanshu Chauhan *
UT Austin

Irina Calciu
VMware Research Group

Vijay Chidambaram
UT Austin

Eric Schkufza
VMware Research Group

Onur Mutlu
ETH Zürich

Pratap Subrahmanyanam
VMware
The Future of Emerging Technologies is Bright

- Regardless of challenges
  - in underlying technology and overlying problems/requirements

Can enable:
- Orders of magnitude improvements
- New applications and computing systems

Yet, we have to
- Think across the stack
- Design enabling systems
If In Doubt, Refer to Flash Memory

- A very “doubtful” emerging technology
  - for at least two decades

Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

By Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu

Abstract: NAND flash memory is ubiquitous in everyday life today because its capacity has continuously increased and its cost has simultaneously decreased. This versatility and cost-effectiveness make NAND flash a preferred choice for solid-state drives (SSDs) and other applications, despite its inherent unreliability. In this paper, we present an overview of error characterization, mitigation, and recovery techniques for NAND flash memories.

Keywords: Data storage systems; error recovery; fault tolerance; flash memory; reliability; solid-state drives

https://arxiv.org/pdf/1706.08642