About Me

- Postdoctoral Researcher at SAFARI
  - Computer architecture
  - Algorithm-architecture co-design
  - Processing in memory (Non-volatile memory)
  - Processing in storage
  - Bioinformatics
  - Machine learning acceleration

- Group Associate at ETH Future Computing Laboratory
  - Cooperate with industry

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 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.
Intel Optane Persistent Memory (2019)

- Non-volatile main memory
- Based on 3D-XPoint Technology
PCM as Main Memory: Idea in 2009


One of the 13 computer architecture papers of 2009 selected as Top Picks by IEEE Micro.
Selected as a CACM Research Highlight.

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
PCM as Main Memory: Idea in 2009

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 reflectivity and high electrical resistivity
  - Crystalline: High optical reflectivity 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 of PCM operation](image)

- **SET (cryst)**: Low resistance
  - $10^3 - 10^4 \Omega$
- **RESET (amorph)**: High resistance
  - $10^6 - 10^7 \Omega$

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: 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:

- **Bit 1**: 11
- **Bit 0**

Cell resistance
MLC-PCM Resistance → Value

*Less margin between values*

→ need more precise sensing/modification of cell contents
→ higher latency/energy (~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>
<thead>
<tr>
<th>Parameter</th>
<th>Horri$^6$</th>
<th>Ahn$^{12}$</th>
<th>Bedeschi$^{13}$</th>
<th>Oh$^{14}$</th>
<th>Pellizer$^{15}$</th>
<th>Chen$^5$</th>
<th>Kang$^{16}$</th>
<th>Bedeschi$^9$</th>
<th>Lee$^{10}$</th>
<th>Lee$^2$</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>GST</td>
<td>GST</td>
<td>GST</td>
<td>GST, N-d</td>
<td>GST, N-d</td>
</tr>
<tr>
<td>Cell size ($\mu$m$^2$)</td>
<td>**</td>
<td>0.290</td>
<td>0.290</td>
<td>**</td>
<td>0.097</td>
<td>60 nm$^2$</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, $F^2$</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 ($\mu$A)</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>**</td>
<td>3.0</td>
<td>1.0</td>
<td>1.8</td>
<td>1.6</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
<tr>
<td>Read power ($\mu$W)</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 energy (pJ)</td>
<td>**</td>
<td>**</td>
<td>2.0</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</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 ($\mu$A)</td>
<td>200</td>
<td>**</td>
<td>300</td>
<td>200</td>
<td>**</td>
<td>55</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</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>90</td>
</tr>
<tr>
<td>Set power ($\mu$W)</td>
<td>**</td>
<td>**</td>
<td>300</td>
<td>**</td>
<td>**</td>
<td>34.4</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</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>**</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 ($\mu$A)</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>600</td>
<td>400</td>
<td>90</td>
<td>600</td>
<td>600</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>Reset voltage (V)</td>
<td>**</td>
<td>**</td>
<td>2.7</td>
<td>**</td>
<td>**</td>
<td>1.6</td>
<td>**</td>
<td>1.6</td>
<td>**</td>
<td>1.6</td>
</tr>
<tr>
<td>Reset power ($\mu$W)</td>
<td>**</td>
<td>**</td>
<td>1620</td>
<td>**</td>
<td>**</td>
<td>80.4</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</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>**</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>$10^6$</td>
<td>$10^5$</td>
<td>$10^5$</td>
<td>$10^8$</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.
Where Can PCM Fit in the System?

- Caches
- Main Memory
- Storage (SSD/HDD)
Phase Change Memory Properties: Latency

- Latency comparable to, but slower than DRAM

- Read Latency
  - 50ns: \(4\times\) DRAM, \(10^{-3}\times\) NAND Flash

- Write Latency
  - 150ns: \(12\times\) DRAM

- Write Bandwidth
  - 5-10 MB/s: \(0.1\times\) DRAM, \(1\times\) NAND Flash

Phase Change Memory Properties

- **Dynamic Energy**
  - 40 uA Rd, 150 uA Wr
  - \(2\text{-}43\times\) DRAM, \(1\times\) NAND Flash

- **Endurance**
  - Writes induce phase change at 650C
  - Contacts degrade from thermal expansion/contraction
  - \(10^8\) writes per cell
  - \(10^{-8}\times\) DRAM, \(10^3\times\) NAND Flash

- **Cell Size**
  - \(9\text{-}12F^2\) using BJT, single-level cells
  - \(1.5\times\) DRAM, \(2\text{-}3\times\) 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 - 12$F^2$ using BJT
- 1.5× DRAM

### Latency
- 50ns Rd, 150ns Wr
- 4×, 12× DRAM

### Energy
- 40μA Rd, 150μA Wr
- 2×, 43× 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

- 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?
PCM As Main Memory


One of the 13 computer architecture papers of 2009 selected as Top Picks by IEEE Micro.
Selected as a CACM Research Highlight.

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)

Intel Optane Memory (Idea Realized in 2019)

- Non-volatile main memory
- Based on 3D-XPoint Technology
More on PCM Based Main Memory


Best (student) presentation award.

Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories

HANBIN YOON* and JUSTIN MEZA, Carnegie Mellon University
NAVEEN MURALIMANOHAR, Hewlett-Packard Labs
NORMAN P. JOUPPI**, Google Inc.
ONUR MUTLU, Carnegie Mellon University
Some PCM Bits Take Longer to Read...

(a) Sensing time is longer for higher cell resistances.

(b) One bit is determined before the other.

Fig. 3. MLC PCM cell read operation [Qureshi et al. 2010b].
Observation 1: Read Asymmetry

- The read latency/energy of Bit 1 is lower than that of Bit 0

- This is due to how MLC-PCM cells are read
Observation 1: Read Asymmetry

Simplified example

Capacitor filled with reference voltage
MLC-PCM cell with unknown resistance
Observation 1: Read Asymmetry

Simplified example
Observation 1: Read Asymmetry

Simplified example
Observation 1: Read Asymmetry
Observation 1: Read Asymmetry
Observation 1: Read Asymmetry

Initial voltage (fully charged capacitor)
Observation 1: Read Asymmetry

PCM cell connected → draining capacitor

Voltage

Time
Observation 1: Read Asymmetry

Capacitor drained → data value known (01)
Observation 1: Read Asymmetry

- In existing devices
  - Both MLC bits are read at the same time
  - Must wait *maximum time* to read both bits

- However, *we can infer information about Bit 1 before this time*
Observation 1: Read Asymmetry
Observation 1: Read Asymmetry
Observation 1: Read Asymmetry

Time to determine Bit 1's value
Observation 1: Read Asymmetry

Time to determine Bit 0's value
Some PCM Bits Take Longer to Write…

Efficient Data Mapping and Buffering Techniques for MLC PCM

(a) All possible cell state transitions.

(b) Cell state transitions when modifying only the MSB or the LSB.

Fig. 4. MLC PCM cell write latencies [Joshi et al. 2011; Nirschl et al. 2007; Happ et al. 2006].
More on PCM Latencies and Exploiting Them

Presented at the 10th HiPEAC Conference, Amsterdam, Netherlands, January 2015. [Slides (ppt) (pdf)]

Best (student) presentation award.
STT-RAM as 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 \(\rightarrow\) 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)
Hybrid Main Memory
A More Viable Approach: Hybrid Memory Systems

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

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

SAFARI
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
Hybrid vs. All-PCM/DRAM [ICCD’12]

More on Hybrid Memory Data Placement

- HanBin Yoon, Justin Meza, Rachata Ausavarungrun, 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)
Best paper award (in Computer Systems and Applications track).

Row Buffer Locality Aware Caching Policies for Hybrid Memories

HanBin Yoon, Justin Meza, Rachata Ausavarungrun, Rachael A. Harding and Onur Mutlu
Carnegie Mellon University
{hanbinyoon,meza,rachata,onor}@cmu.edu, rhardin@mit.edu
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 → Mem. B

After migration:

requests to Page 1 → Mem. A

Migrating one page reduces stall time by \( T \)

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
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>ALL</th>
<th>FREQ</th>
<th>RBLA</th>
<th>UH-MEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{RCD}$: x3.0</td>
<td>8%</td>
<td>6%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_{WR}$: x3.0</td>
<td>13%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x4.0</td>
<td>13%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x4.5</td>
<td>14%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x6.0</td>
<td>13%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x7.5</td>
<td>13%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x12</td>
<td>8%</td>
<td>6%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x16</td>
<td>13%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x20</td>
<td>13%</td>
<td></td>
<td></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)]

Utility-Based Hybrid Memory Management

<table>
<thead>
<tr>
<th>Yang Li†</th>
<th>Saugata Ghose†</th>
<th>Jongmoo Choi‡</th>
<th>Jin Sun†</th>
<th>Hui Wang*</th>
<th>Onur Mutlu†‡</th>
</tr>
</thead>
</table>
†Carnegie Mellon University
‡Dankook University
*Beihang University
†ETH Zürich
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

-6%

Reduced channel contention and improved spatial locality

TIMBER Energy Efficiency

Fewer migrations reduce transmitted data and channel contention.

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

![Graphs showing performance comparison between Banshee, Alloy, TDC, and Unison across different cache latencies and bandwidths.]
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\textsuperscript{1}  Christopher J. Hughes\textsuperscript{2}  Nadathur Satish\textsuperscript{2}  Onur Mutlu\textsuperscript{3}  Srinivas Devadas\textsuperscript{1}

\textsuperscript{1}MIT  \textsuperscript{2}Intel Labs  \textsuperscript{3}ETH Zürich
Other Opportunities with Emerging Memory Technologies
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
Recall: Processing Using Memory
In-Memory Bulk Bitwise Operations

- We can support in-DRAM COPY, ZERO, AND, OR, NOT, MAJ
- At low cost
- Using analog computation capability of DRAM
  - Idea: activating multiple rows performs computation
- 30-60X performance and energy improvement

- New memory technologies enable even more opportunities
  - Memristors, resistive RAM, phase change mem, STT-MRAM, ...
  - Can operate on data with minimal movement
In-DRAM Bulk Bitwise AND/OR

- Vivek Seshadri, Kevin Hsieh, Amirali Boroumand, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
  "Fast Bulk Bitwise AND and OR in DRAM"

Fast Bulk Bitwise AND and OR in DRAM

Vivek Seshadri*, Kevin Hsieh*, Amirali Boroumand*, Donghyuk Lee*, Michael A. Kozuch†, Onur Mutlu*, Phillip B. Gibbons†, Todd C. Mowry*

*Carnegie Mellon University†Intel Pittsburgh
Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

Proceedings of the 50th International Symposium on Microarchitecture (MICRO), Boston, MA, USA, October 2017.
[Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]
In-DRAM Bulk Bitwise Execution Paradigm


In-DRAM Bulk Bitwise Execution Engine

Vivek Seshadri  
Microsoft Research India  
visesha@microsoft.com

Onur Mutlu  
ETH Zürich  
onur.mutlu@inf.ethz.ch
SIMDRAM Framework for in-DRAM Computing


[2-page Extended Abstract]
[Short Talk Slides (pptx) (pdf)]
[Talk Slides (pptx) (pdf)]
[Short Talk Video (5 mins)]
[Full Talk Video (27 mins)]

SIMDRAM: A Framework for Bit-Serial SIMD Processing using DRAM

*Nastaran Hajinazar$^{1,2}$ Nika Mansouri Ghiasi$^1$
Geraldo F. Oliveira$^1$ Minesh Patel$^1$
Sven Gregorio$^1$ Mohammed Alser$^1$
João Dinis Ferreira$^1$ Saugata Ghose$^3$

$^1$ETH Zürich $^2$Simon Fraser University $^3$University of Illinois at Urbana–Champaign
Lecture on RowClone & Processing using DRAM
Key Idea and Applications

- **Low-cost Inter-linked subarrays (LISA)**
  - Fast bulk data movement between subarrays
  - Wide datapath via isolation transistors: 0.8% DRAM chip area

- **LISA is a versatile substrate** → new applications
  - **Fast bulk data copy**: Copy latency 1.363ms → 0.148ms (9.2x)
    → 66% speedup, -55% DRAM energy
  - **In-DRAM caching**: Hot data access latency 48.7ns → 21.5ns (2.2x)
    → 5% speedup
  - **Fast precharge**: Precharge latency 13.1ns → 5.0ns (2.6x)
    → 8% speedup
In-DRAM NOT Operation

Figure 5: Bitwise NOT using a dual contact capacitor


Computer Architecture - Lecture 7: Processing using Memory II (Fall 2021)
630 views • Streamed live on Oct 21, 2021
Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories

Shuangchen Li, Cong Xu, Qiaosha Zou, Jishen Zhao, Yu Lu, and Yuan Xie

University of California, Santa Barbara, Hewlett Packard Labs
University of California, Santa Cruz, Qualcomm Inc., Huawei Technologies Inc.

{shuangchenli, yuanxie}@ece.ucsb.edu
Pinatubo: RowClone and Bitwise Ops in PCM

Figure 2: Overview: (a) Computing-centric approach, moving tons of data to CPU and write back. (b) The proposed Pinatubo architecture, performs $n$-row bitwise operations inside NVM in one step.
New: In-Memory
Crossbar Array Operations
In-Memory Crossbar Array Operations

- Some emerging NVM technologies have crossbar array structure
  - Memristors, resistive RAM, phase change mem, STT-MRAM, ...

- Crossbar arrays can be used to perform dot product operations using “analog computation capability”
  - Can operate on multiple pieces of data using Kirchoff’s laws
    - Bitline current is a sum of products of wordline V x (1 / cell R)
    - Computation is in analog domain inside the crossbar array

- Need peripheral circuitry for D->A and A->D conversion of inputs and outputs
In-Memory Crossbar Computation

Fig. 1. (a) Using a bitline to perform an analog sum of products operation. (b) A memristor crossbar used as a vector-matrix multiplier.
In-Memory Crossbar Computation

\[
\left( \begin{array}{cccc}
i_1 & i_2 & i_3 & i_4 \\
\end{array} \right) \rightarrow \left( \begin{array}{cccc}
o_1 & o_2 & o_3 & o_4 \\
\end{array} \right)
\]

\[
I = \frac{1}{R_{11}} V_1 + \frac{1}{R_{21}} V_2 + \frac{1}{R_{31}} V_3 + \frac{1}{R_{41}} V_4
\]
Required Peripheral Circuitry

DAC: Digital to Analog
ADC: Analog to Digital
S&H: Sample and Hold

Shift and add: used to summarize the final output
An Example of 2D Convolution

**Structure information**
- Input: 5*5 (blue)
- Kernel (filter): 3*3 (grey)
- Output: 5*5 (green)

**Computation information**
- Stride: 1
- Padding: 1 (white)

**Output Dim**
\[
\text{Output Dim} = \frac{\text{Input} + 2 \times \text{Padding} - \text{Kernel}}{\text{Stride}} + 1
\]
Mapping Computation onto the Crossbar

A convolution operation in neural network application

An NVM-based PIM array
An Overview of NVM-Based PIM System

NVM-based PIM array:
- core processing unit for vector-matrix multiplication

Non-linear function array:
- processing unit for non-linear functions (e.g., ReLU operations in neural networks)

Multiplier array:
- handles element-wise operations
NVM-based PIM used in Genome Analysis

- Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina, Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, and Onur Mutlu, "GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping". Proceedings of the 55th International Symposium on Microarchitecture (MICRO), Chicago, IL, USA, October 2022.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Lecture Video (25 minutes)]
  [arXiv version]

GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping

Haiyu Mao¹  Mohammed Alser¹  Mohammad Sadrosadati¹  Can Firtina¹  Akanksha Baranwal¹
Damla Senol Cali²  Aditya Manglik¹  Nour Almadhoun Alserr¹  Onur Mutlu¹

¹ETH Zürich  ²Bionano Genomics
NVM-based PIM used in Genome Analysis

- Taha Shahroodi, Gagandeep Singh, Mahdi Zahedi, Haiyu Mao, Joel Lindegger, Can Firtina, Stephan Wong, Onur Mutlu, and Said Hamdioui,
  "Swordfish: A Framework for Evaluating Deep Neural Network-based Basecalling using Computation-In-Memory with Non-Ideal Memristors"
  Proceedings of the 56th International Symposium on Microarchitecture (MICRO), Toronto, ON, Canada, November 2023.
  [Slides (pptx) (pdf)]
  [arXiv version]

Swordfish: A Framework for Evaluating Deep Neural Network-based Basecalling using Computation-In-Memory with Non-Ideal Memristors

Taha Shahroodi\textsuperscript{1} Gagandeep Singh\textsuperscript{2,3} Mahdi Zahedi\textsuperscript{1} Haiyu Mao\textsuperscript{3} Joel Lindegger\textsuperscript{3} Can Firtina\textsuperscript{3}
Stephan Wong\textsuperscript{1} Onur Mutlu\textsuperscript{3} Said Hamdioui\textsuperscript{1}

\textsuperscript{1}TU Delft \quad \textsuperscript{2}AMD Research \quad \textsuperscript{3}ETH Zürich
Example Readings on NVM-Based PIM


Example 2D Systolic Array Computation

Multiply two 3x3 matrices (inputs)
- Keep the final result in PE accumulators

\[
\begin{bmatrix}
    c_{00} & c_{01} & c_{02} \\
    c_{10} & c_{11} & c_{12} \\
    c_{20} & c_{21} & c_{22}
\end{bmatrix}
\times
\begin{bmatrix}
    a_{00} & a_{01} & a_{02} \\
    a_{10} & a_{11} & a_{12} \\
    a_{20} & a_{21} & a_{22}
\end{bmatrix}
= \begin{bmatrix}
    b_{00} & b_{01} & b_{02} \\
    b_{10} & b_{11} & b_{12} \\
    b_{20} & b_{21} & b_{22}
\end{bmatrix}
\]

P = M
Q = N
R = R + M*N

Figure 1: A systolic array processing element

Digital Design & Computer Arch. - Lecture 19: VLIW, Systolic Arrays, DAE (ETH Zürich, Spring 2021)

https://www.youtube.com/watch?v=UtLy4Yagdys&list=PL5Q2soXY2Zi_uej3aY39YB5pfW4SJ7LIN&index=21
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

FILE

I/O

Ld/St

VOLATILE

FAST

BYTE ADDR

NONVOLATILE

SLOW

BLOCK ADDR
Non-volatile memories combine characteristics of memory and storage
Two-Level Memory/Storage Model

- 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
}
```

Figure 2: Sample program with access to file-based (left) and object-based (right) persistent data.
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
Efficient Data Mapping among Heterogeneous Devices

Columns in a column store that are scanned through only infrequently → place on Flash

Less Locality

More Locality

Temporary

Persistent
Efficient Data Mapping among Heterogeneous Devices

Applications or system software can provide hints for data placement

Columns in a column store that are scanned through only infrequently → place on Flash

Frequently-updated index for a Content Delivery Network (CDN) → place in DRAM
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

- **HDD 2-level**
- **NVM 2-level**
- **Persistent Memory**

Normalized Execution Time

- **User CPU**
- **User Memory**
- **Syscall CPU**
- **Syscall I/O**

- **Normalized Execution Time**
  - HDD 2-level: ~24X
  - NVM 2-level: ~5X
  - Persistent Memory: 0.009

Energy Benefits of a Single-Level Store

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
Intel Optane Persistent Memory (2019)

- Non-volatile main memory
- Based on 3D-XPoint Technology
UPMEM Processing-in-DRAM Engine (2019)

- Processing in DRAM Engine
- Includes **standard DIMM modules**, with a **large number of DPU processors** combined with DRAM chips.

- Replaces **standard** DIMMs
  - DDR4 R-DIMM modules
    - 8GB+128 DPUs (16 PIM chips)
    - Standard 2x-nm DRAM process
  - **Large amounts of** compute & memory bandwidth

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

1. Link to next
2. Link to prev

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
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 **TM_hashtable_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 \textbf{TMhashtable\_update}(TMARCGDECL hashtable\_t* ht, void *key, void*data){
  list\_t* chain = \textbf{get\_chain}(ht, key);
  pair\_t* pair;
  pair\_t updatePair;
  updatePair.first = key;
  pair = (pair\_t*) \textbf{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;
}
```

Need a new implementation
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;
}
CURRENT SOLUTIONS

Manual declaration of persistent components

```c
void TM_hashtable_update(TMARCGDECL hashtable_t* ht, void *key, void* data) {
    list_t* chain = get_chain(ht, key);
    pair_t* 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 checkpoint 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

Epoch 0

Epoch 1

Epoch 2

Running

Checkpointing

Running

Checkpointing

Running

Checkpointing

Running

Checkpointing

Running

Checkpointing
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)]
Consistency Support for Persistent Memory

- Youyou Lu, Jiwu Shu, Long Sun, and Onur Mutlu, "Loose-Ordering Consistency for Persistent Memory"
  Proceedings of the 32nd IEEE International Conference on Computer Design (ICCD), Seoul, South Korea, October 2014.
  [Slides (pptx) (pdf)]
  [Erratum]

Loose-Ordering Consistency for Persistent Memory

Youyou Lu †, Jiwu Shu † §, Long Sun † and Onur Mutlu ‡
†Department of Computer Science and Technology, Tsinghua University, Beijing, China
§State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
‡Computer Architecture Laboratory, Carnegie Mellon University, Pittsburgh, PA, USA
luyy09@mails.tsinghua.edu.cn, shujw@tsinghua.edu.cn, sun-l12@mails.tsinghua.edu.cn, onur@cmu.edu
Another Key Challenge in Persistent Memory

Security and Data Privacy Issues
Security and Privacy Issues of NVM

- Endurance problems → Wearout attacks
- Hybrid memories → Performance attacks
- Data not erased after power-off → Privacy breaches
Conclusion
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 continuously decreased. This article examines the current error rates in modern NAND flash memories and presents approaches to improve reliability and fault tolerance.

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

https://arxiv.org/pdf/1706.08642
Many Research & Design Opportunities

- Enabling completely persistent memory
- Computation in/using NVM based memories
- Hybrid memory systems
- Security and privacy issues in persistent memory
- Reliability and endurance related problems
- Virtual memory systems for NVM (e.g., virtual block interface)
GenPIP

In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping

Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina, Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, Onur Mutlu
Overview: Genome Analysis

- **Genome analysis**: Enables us to determine the order of the DNA sequence in an organism’s genome
  - Plays an important role in
    - Personalized medicine
    - Outbreak tracing
    - Understanding of evolution
    - ...

- Modern genome sequencing machines extract smaller randomized fragments of the original DNA sequence, known as *reads*
  - **Oxford Nanopore Technologies (ONT)**:
    A widely-used sequencing technology
    - Portable sequencing devices
    - High-throughput
    - Cheap

[forbes.com]
Overview: Two Limitations

Multiple steps in genome analysis

1. Large data movement between multiple steps
2. A lot of wasted computation done on data that is later discovered to be useless
Overview: GenPIP

- **GenPIP**: A fast and energy-efficient in-memory acceleration system for the Genome analysis PIPeline via tight integration of genome analysis steps.

- GenPIP has two key techniques:
  - Chunk-based pipeline (CP)
    - Provides fine-grained collaboration of genome analysis steps
  - Early rejection (ER)
    - Timely stops the execution on useless data by predicting which reads will not be useful

- GenPIP outperforms state-of-the-art software & hardware solutions using CPU, GPU, and optimistic PIM by 41.6×, 8.4×, and 1.4×, respectively.
Outline

- Background and Motivation
  - GenPIP: Tight Integration of Genome Analysis Steps
    - Chunk-based Pipeline (CP)
    - Early Rejection (ER)
- GenPIP Implementation
- Evaluation
- Conclusion
Genome Analysis Pipeline

1. Basecalling
- Compute
- Storage

2. Read Quality Control
- Compute
- Storage

3. Read Mapping
- Compute
- Storage

Reference genome:
- T A T G G A C T T T A G C A A A A C

Mapped
- ATGGAC

Unmapped
- GC G T T T C

Low-quality:
- GC G T T T C

Store mapping results
Limitation 1: Large Data Movement

- Using a human dataset in [NC’19] as an example:

![Diagram showing data movement]

- Large data movement between genome analysis steps

Limitation 2: Wasted Computation

- Using a human dataset in [NC’19] as an example:

A considerable amount of computation on **useless data** due to
  - Low-quality reads
  - Unmapped reads

---

State-of-the-art Works

- NVM-based PIM is an efficient technique to reduce data movement by processing data using or near memory

Raw Signals -> Basecalling -> Reads -> Read quality control -> High-quality reads -> Read mapping -> Mapped reads

- NVM-based PIM for dot-product operation [Helix, PACT’20]
- Reduce the data movement in a single genome analysis step
- Exacerbate the data movement overhead between analysis steps

No prior work tackles data movement between analysis steps and reduces useless computation
Goal and Opportunities

**Goal:** Efficiently accelerate the entire genome analysis pipeline while minimizing data movement and useless computation.

- We perform a study to quantify potential performance benefits.
  - Results are normalized to the performance of GPU.

![Normalized Speedup Chart](chart.png)

- **NVM-based PIM accelerators** for separate basecalling and read mapping: 2.7x normalized speedup.
- **No data movement** between the accelerators of analysis steps: 6.1x normalized speedup.
- **No data movement and no useless reads** (ideal case): 9x normalized speedup.
Outline

- Background and Motivation

- GenPIP: Tight Integration of Genome Analysis Steps
  - Chunk-based Pipeline (CP)
  - Early Rejection (ER)

- GenPIP Implementation

- Evaluation

- Conclusion
GenPIP

- *First holistic in-memory accelerator for the genome analysis pipeline*, including basecalling, read quality control, and read mapping steps

- **GenPIP** has two key techniques
  - **Chunk-based Pipeline (CP)**
    - Enables fine-grained pipelining of genome analysis steps
    - Processes reads at *chunk* granularity (i.e., a subsequence; 300 bases)
  - **Early Rejection (ER)**
Chunk-based Pipeline (CP)

- **CP increases parallelism** by overlapping the execution of different steps at chunk granularity.
- **CP reduces intermediate data** by computing on data as soon as data is generated.
- **CP provides opportunities for ER** by analyzing a read at chunk granularity.

A read consists of four chunks: **C1, C2, C3, C4**.
GenPIP

- First holistic in-memory accelerator for the genome analysis pipeline, including basecalling, read quality control, and read mapping steps

- GenPIP has two key techniques
  
  - Chunk-based Pipeline (CP)
    - Enables fine-grained collaboration of genome analysis steps by processing reads at chunk granularity (i.e., a subsequence of a read, e.g., 300 bases)

  - Early Rejection (ER)
    - Stops the execution on useless reads as early as possible by using a small number of chunks to predict the usefulness of a read
Early Rejection (ER)

- **Predict and eliminate** low-quality and unmapped reads from the genome analysis pipeline as early as possible

Diagram:

- Basecall a small number of chunks
- Check the average quality of these chunks
- Pass: Basecall more chunks
- Fail: Stop analysis
- Map basecalled chunks so far
- Fail: Basecall the remaining chunks
- Pass: Execute the remaining computation in read mapping

**Early-Rejection based on chunk quality scores (ER-QSR):**
- Predict **low-quality** reads using chunk quality scores

**Early-Rejection based on chunk mapping scores (ER-CMR):**
- Predict **unmapped reads** using chunk mapping scores
Implementation of CP and ER

CP and ER can be applied on different systems, e.g., CPU, GPU, and PIM

We implement CP and ER using PIM since PIM is more efficient to reduce the data movement between genome analysis steps

We also apply CP and ER on CPU and GPU baselines and observe speedup and energy savings
Outline

- Background and Motivation
- GenPIP: Tight Integration of Genome Analysis Steps
  - Chunk-based Pipeline (CP)
  - Early Rejection (ER)
- GenPIP Implementation
- Evaluation
- Conclusion
GenPIP Implementation

Raw signals from the sequencing machine

- In-memory Basecaller [Helix, PACT’20]
- Basecalling Module
- PIM-CQS
  - PIM chunk quality score calculation
- Base quality score

- eDRAM
- Signal chunk
- Basecalled chunk
- Chunk mapping score

- Average Calculator
- Quality score
- Chunk

- ER Controller
- GenPIP Controller
- ER

- Read Mapping Controller
- Read mapping result
- Read Mapping Module
- In-memory Read Mapping [PARC, ASPDAC’20] + Our design

- Chunk
- Chunk
- To storage

Tightly integrating the genome analysis steps
- Reduces data movement
- Eliminates useless computation
Outline

- Background and Motivation
- GenPIP: Tight Integration of Genome Analysis Steps
  - Chunk-based Pipeline (CP)
  - Early Rejection (ER)
- GenPIP Implementation
- Evaluation
- Conclusion
Evaluation Methodology

- **Performance, Area and Power Analysis:**
  - Simulation via Verilog HDL, NVSim [TCAD’12], and CACTI 6.5 [MICRO’07]
  - See methodology in the paper for more

- **Baselines:**
  - **CPU** (Intel Xeon Gold 5118 CPU)
  - **GPU** (NVIDIA GeForce RTX 2080 Ti GPU)
  - Optimistic integration of two PIM accelerators (Helix [PACT’20] and PARC [ASP-DAC’20])
    - Assumes no data movement between steps
    - Assumes intermediate data causes no overhead

- **Datasets:**
  - **E. coli** ([http://lab.loman.net/2016/07/30/nano_pore- r9- data- release/](http://lab.loman.net/2016/07/30/nano_pore- r9- data- release/))
  - **Human** ([https://www.ebi.ac.uk/ena/browser/view/PRJEB30620](https://www.ebi.ac.uk/ena/browser/view/PRJEB30620))
GenPIP provides 41.6x, 8.4x, and 1.4x speedup over CPU, GPU, and optimistic PIM.

Both CP and ER are critical to the speedup.
Key Results – Energy Efficiency

GenPIP provides 32.8x, 20.8x, and 1.37x energy savings over CPU, GPU, and optimistic PIM.

ER is especially critical to the energy efficiency.
More in the Paper

GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping

Haiyu Mao¹ Mohammed Alser¹ Mohammad Sadrosadati¹ Can Firtina¹ Akanksha Baranwal¹
Damla Senol Cali² Aditya Manglik¹ Nour Almadhoun Alserr¹ Onur Mutlu¹

¹ETH Zürich  ²Bionano Genomics

- Timely early rejection implementation
- In-memory seeding accelerator

More in the Paper

- Details of **CP and ER**
- Detailed **GenPIP** implementation
  - GenPIP controller
  - Early rejection implementation
  - In-memory seeding accelerator
- Results of **applying CP and ER in CPU and GPU**
- **Sensitivity analysis** on the number of sampled chunks used for ER
- **Area and power** analysis
Outline

- Background and Motivation
- GenPIP: Tight Integration of Genome Analysis Steps
  - Chunk-based Pipeline (CP)
  - Early Rejection (ER)
- GenPIP Implementation
- Evaluation
- Conclusion
Conclusion

- **Problem**: The genome analysis pipeline has large data movement between genome analysis steps and a significant amount of wasted computation on useless data.

- **Goal**: Tightly integrate genome analysis steps to reduce the data movement between steps and eliminate computation on useless data.

- **GenPIP**: The first in-memory genome analysis accelerator that tightly integrates genome analysis steps.

- GenPIP has two key techniques:
  - A chunk-based pipeline
  - A new early-rejection technique

- GenPIP outperforms state-of-the-art software & hardware solutions using CPU, GPU, and optimistic PIM by 41.6×, 8.4×, and 1.4×, respectively.
GenPIP
In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping

Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina, Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, Onur Mutlu

SAFARI
ETH Zürich
Computer Architecture

Lecture 15: Emerging Memory Technologies

Dr. Haiyu Mao
Prof. Onur Mutlu
ETH Zürich
Fall 2023
16 November 2023