Rethinking Memory System Design
Business as Usual in the Next Decade?

Onur Mutlu
onur.mutlu@inf.ethz.ch
http://users.ece.cmu.edu/~omutlu/
October 6, 2016
RSP 2016 Keynote (Pittsburgh)
Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor.

Main memory system must scale (in size, technology, efficiency, cost, and management algorithms) to maintain performance growth and technology scaling benefits.
Memory System: A *Shared Resource View*
State of the Main Memory System

- Recent technology, architecture, and application trends
  - lead to new requirements
  - exacerbate old requirements

- DRAM and memory controllers, as we know them today, are (will be) unlikely to satisfy all requirements

- Some emerging non-volatile memory technologies (e.g., PCM) enable new opportunities: memory+storage merging

- We need to rethink the main memory system
  - to fix DRAM issues and enable emerging technologies
  - to satisfy all requirements
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary
Major Trends Affecting Main Memory (I)

- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending
Major Trends Affecting Main Memory (II)

- Need for main memory capacity, bandwidth, QoS increasing
  - Multi-core: increasing number of cores/agents
  - Data-intensive applications: increasing demand/hunger for data
  - Consolidation: cloud computing, GPUs, mobile, heterogeneity

- Main memory energy/power is a key system design concern

- DRAM technology scaling is ending
Example: The Memory Capacity Gap

- Memory capacity per core expected to drop by 30% every two years
- Trends worse for memory bandwidth per core!
Major Trends Affecting Main Memory (III)

- Need for main memory capacity, bandwidth, QoS increasing

- Main memory energy/power is a key system design concern
  - ~40-50% energy spent in off-chip memory hierarchy [Lefurgy, IEEE Computer 2003]
  - DRAM consumes power even when not used (periodic refresh)

- DRAM technology scaling is ending
Major Trends Affecting Main Memory (IV)

- Need for main memory capacity, bandwidth, QoS increasing

- Main memory energy/power is a key system design concern

- DRAM technology scaling is ending
  - ITRS projects DRAM will not scale easily below X nm
  - Scaling has provided many benefits:
    - higher capacity (density), lower cost, lower energy
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary
The DRAM Scaling Problem

- DRAM stores charge in a capacitor (charge-based memory)
  - Capacitor must be large enough for reliable sensing
  - Access transistor should be large enough for low leakage and high retention time
  - Scaling beyond 40-35nm (2013) is challenging [ITRS, 2009]

- DRAM capacity, cost, and energy/power hard to scale
Repeatedly opening and closing a row enough times within a refresh interval induces disturbance errors in adjacent rows in most real DRAM chips you can buy today.

*An Example of the DRAM Scaling Problem*

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors, (Kim et al., ISCA 2014)
Most DRAM Modules Are at Risk

A company

86%
(37/43)

Up to
$1.0 \times 10^7$
errors

B company

83%
(45/54)

Up to
$2.7 \times 10^6$
errors

C company

88%
(28/32)

Up to
$3.3 \times 10^5$
errors

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors, (Kim et al., ISCA 2014)
**loop:**
- `mov (X), %eax`
- `mov (Y), %ebx`
- `clflush (X)`
- `clflush (Y)`
- `mfence`
- `jmp loop`

https://github.com/CMU-SAFARI/rowhammer
```assembly
loop:
    mov (X), %eax
    mov (Y), %ebx
    clflush (X)
    clflush (Y)
    mfence
    jmp loop
```

https://github.com/CMU-SAFARI/rowhammer
loop:
    mov (X), %eax
    mov (Y), %ebx
    clflush (X)
    clflush (Y)
    mfence
    jmp loop

https://github.com/CMU-SAFARI/rowhammer
loop:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp loop

https://github.com/CMU-SAFARI/rowhammer
A real reliability & security issue

In a more controlled environment, we can induce as many as ten million disturbance errors

Errors vs. Vintage

All modules from 2012–2013 are vulnerable
Experimental DRAM Testing Infrastructure

An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems (Qureshi et al., DSN 2015)
Experimental Infrastructure (DRAM)

One Can Take Over an Otherwise-Secure System

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Project Zero

News and updates from the Project Zero team at Google

Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn, 2015)
RowHammer Security Attack Example

- “Rowhammer” is a problem with some recent DRAM devices in which repeatedly accessing a row of memory can cause bit flips in adjacent rows (Kim et al., ISCA 2014).
  - *Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors* (Kim et al., ISCA 2014)

- We tested a selection of laptops and found that a subset of them exhibited the problem.

- We built two working privilege escalation exploits that use this effect.
  - *Exploiting the DRAM rowhammer bug to gain kernel privileges* (Seaborn, 2015)

- One exploit uses rowhammer-induced bit flips to gain kernel privileges on x86-64 Linux when run as an unprivileged userland process.

- When run on a machine vulnerable to the rowhammer problem, the process was able to induce bit flips in page table entries (PTEs).

- It was able to use this to gain write access to its own page table, and hence gain read-write access to all of physical memory.
It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after.
Apple’s Patch for RowHammer

- https://support.apple.com/en-gb/HT204934

Available for: OS X Mountain Lion v10.8.5, OS X Mavericks v10.9.5

Impact: A malicious application may induce memory corruption to escalate privileges

Description: A disturbance error, also known as Rowhammer, exists with some DDR3 RAM that could have led to memory corruption. This issue was mitigated by increasing memory refresh rates.

CVE-ID

CVE-2015-3693: Mark Seaborn and Thomas Dullien of Google, working from original research by Yoongu Kim et al (2014)

HP and Lenovo released similar patches
Challenge and Opportunity

Reliability
(and Security)
More Intelligent Memory Controllers

Online Detection and Fixing of DRAM Errors
Large-Scale Failure Analysis of DRAM Chips

- Analysis and modeling of memory errors found in all of Facebook’s server fleet

- Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field" Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Rio de Janeiro, Brazil, June 2015. [Slides (pptx) (pdf)] [DRAM Error Model]
DRAM Reliability Reducing

Intuition: quadratic increase in capacity
Aside: Flash Error Analysis in the Field

- First large-scale field study of flash memory errors


A Large-Scale Study of Flash Memory Failures in the Field

Justin Meza
Carnegie Mellon University
meza@cmu.edu

Qiang Wu
Facebook, Inc.
qwu@fb.com

Sanjeev Kumar
Facebook, Inc.
skumar@fb.com

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu
Recap: The DRAM Scaling Problem

DRAM Process Scaling Challenges

- Refresh
  - Difficult to build high-aspect ratio cell capacitors decreasing cell capacitance

Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling

Uksong Kang, Hak-soo Yu, Churoo Park, Hongzhong Zheng, John Halbert, Kuljit Bains, SeongJin Jang, and Joo Sun Choi

Samsung Electronics, Hwasung, Korea / Samsung Electronics, San Jose / Intel
How Do We Solve The Problem?

- **Fix it**: Make memory and controllers more intelligent
  - New interfaces, architectures: system-mem codesign

- **Eliminate or minimize it**: Replace or (more likely) augment DRAM with a different technology
  - New technologies and system-wide rethinking of memory & storage

- **Embrace it**: Design heterogeneous memories (none of which are perfect) and map data intelligently across them
  - New models for data management and maybe usage

---

Solutions (to memory scaling) require software/hardware/device cooperation
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
  - Reduce energy
  - Improve latency and bandwidth
  - Reduce waste (capacity, bandwidth, latency)
  - Enable computation close to data
Solution 1: New Memory Architectures

- 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," DSN 2014.
- Seshadri+, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses," MICRO 2015.

Avoid DRAM:
Solution 2: Emerging Memory Technologies

- Some emerging resistive memory technologies seem more scalable than DRAM (and they are non-volatile)
- Example: Phase Change Memory
  - Expected to scale to 9nm (2022 [ITRS])
  - Expected to be denser than DRAM: can store multiple bits/cell
- But, emerging technologies have shortcomings as well
  - Can they be enabled to replace/augment/surpass DRAM?

- Zhao+, “FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems,” MICRO 2014.
Solution 3: Hybrid Memory Systems

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

Yoon, Meza et al., “Row Buffer Locality Aware Caching Policies for Hybrid Memories,” ICCD 2012 Best Paper Award.
Exploiting Memory Error Tolerance with Hybrid Memory Systems

- Vulnerable data
- Tolerant data
- Reliable memory
- Low-cost memory

On Microsoft’s Web Search workload:
Reduces server hardware cost by 4.7%
Achieves single server availability target of 99.90%

Heterogeneous-Reliability Memory [DSN 2014]
Challenge and Opportunity

Providing the Best of Multiple Metrics
Departing From “Business as Usual”

Heterogeneous Memory Systems

Configurable Memory Systems
An Orthogonal Issue: Memory Interference

Cores’ interfere with each other when accessing shared main memory
An Orthogonal Issue: Memory Interference

- **Problem:** Memory interference between cores is uncontrolled
  - unfairness, starvation, low performance
  - uncontrollable, unpredictable, vulnerable system

- **Solution:** QoS-Aware Memory Systems
  - Hardware designed to provide a configurable fairness substrate
    - Application-aware memory scheduling, partitioning, throttling
  - Software designed to configure the resources to satisfy different QoS goals

- QoS-aware memory systems can provide predictable performance and higher efficiency
Goal: Predictable Performance in Complex Systems

- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs

How to allocate resources to heterogeneous agents to mitigate interference and provide predictable performance?
Strong Memory Service Guarantees

- **Goal:** Satisfy performance/SLA requirements in the presence of shared main memory, heterogeneous agents, and hybrid memory/storage

- **Approach:**
  - Develop techniques/models to accurately estimate the performance loss of an application/agent in the presence of resource sharing
  - Develop mechanisms (hardware and software) to enable the resource partitioning/prioritization needed to achieve the required performance levels for all applications
  - All the while providing high system performance

Challenge and Opportunity

Strong Memory Service Guarantees
Departing From “Business as Usual”

Predictable Memory Management

Programmable Memory Systems
Some Promising Directions

- **New memory architectures**
  - Rethinking memory’s role and functions
  - Memory-centric system design

- **Enabling and exploiting emerging NVM technologies**
  - Hybrid memory systems
  - Single-level memory and storage

- **System-level memory/storage QoS**
  - Predictable systems with configurable QoS
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary
Rethinking Memory Architecture

- Compute-capable memory
- Refresh
- Reliability
- Latency
- Bandwidth
- Energy
- Memory Compression
Why In-Memory Computation Today?

- Push from Technology
  - DRAM Scaling at jeopardy
  - Controllers close to DRAM

- Pull from Systems and Applications
  - Data access is a major system and application bottleneck
  - Systems are energy limited
  - Data movement much more energy-hungry than computation

Dally, HiPEAC 2015
Two Approaches to In-Memory Processing

1. Minimally change DRAM to enable simple yet powerful computation primitives
   - RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
   - Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)

2. Exploit the control logic in 3D-stacked memory to enable more comprehensive computation near memory
   - A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing (Ahn et al., ISCA 2015)
Bulk Copy and Initialization

*memmove & memcpy*: 5% cycles in Google’s datacenter [Kanev+ ISCA’15]

- Forking
- Zero initialization (e.g., security)
- Checkpointing
- VM Cloning
- Deduplication
- Page Migration
- Many more
Today’s Memory: Bulk Data Copy

1) High latency
2) High bandwidth utilization
3) Cache pollution
4) Unwanted data movement

1046ns, 3.6uJ  (for 4KB page copy via DMA)
Future: RowClone (In-Memory Copy)

1) Low latency
2) Low bandwidth utilization
3) No cache pollution
4) No unwanted data movement

1906ns, 306uJ
DRAM Subarray Operation (load one byte)

Step 1: Activate row

Row Buffer (4 Kbytes)

Step 2: Read transfer byte onto bus

DRAM subarray

Transfer row

4 Kbytes

Data Bus
RowClone: In-DRAM Row Copy

- **Step 1:** Activate row A
- **Step 2:** Activate row B

- **DRAM subarray**
- **Row Buffer (4 Kbytes)**
- **Data Bus**
- **Transfer row**
- **4 Kbytes**
- **8 bits**
Generalized RowClone

0.01% area cost

Inter Subarray Copy
(Use Inter-Bank Copy Twice)

Inter Bank Copy
(Pipelined Internal RD/WR)

Intra Subarray Copy
(2 ACTs)
RowClone: Latency and Energy Savings

RowClone: Application Performance

% Compared to Baseline

- IPC Improvement
- Energy Reduction

- bootup
- compile
- forkbench
- mcached
- mysql
- shell
RowClone: Multi-Core Performance

![Normalized Weighted Speedup Graph](image)

- Baseline
- RowClone

50 Workloads (4-core)
End-to-End System Design

- **Application**
- **Operating System**
- **ISA**
- **Microarchitecture**
- **DRAM (RowClone)**

How to communicate occurrences of bulk copy/initialization across layers?

How to ensure cache coherence?

How to maximize latency and energy savings?

How to handle data reuse?
Goal: Ultra-Efficient Processing Near Data

Memory similar to a “conventional” accelerator
Enabling In-Memory X

- What is a flexible and scalable memory interface?
- What is the right partitioning of computation capability?
- What is the right low-cost memory substrate?
- What memory technologies are the best enablers?
- How do we rethink/ease X algorithms/applications?
Enabling In-Memory Computation

<table>
<thead>
<tr>
<th>DRAM Support</th>
<th>Cache Coherence</th>
<th>Virtual Memory Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>RowClone (MICRO 2013)</td>
<td>Dirty-Block Index (ISCA 2014)</td>
<td>Page Overlays (ISCA 2015)</td>
</tr>
<tr>
<td>In-DRAM Gather Scatter (MICRO 2015)</td>
<td>Non-contiguous Cache lines</td>
<td>Gathered Pages</td>
</tr>
<tr>
<td>In-DRAM Bitwise Operations (IEEE CAL 2015)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In-DRAM AND/OR: Triple Row Activation

\[ \frac{1}{2}V_{DD} + \delta \]

A
B
C

Final State
\[ AB + BC + AC \]

\[ C(A + B) + \sim C(AB) \]

In-DRAM Bulk Bitwise AND/OR Operation

- **BULKAND A, B \(\rightarrow\) C**
- Semantics: Perform a bitwise AND of two rows A and B and store the result in row C
- R0 – reserved zero row, R1 – reserved one row
- D1, D2, D3 – Designated rows for triple activation

1. RowClone A into D1
2. RowClone B into D2
3. RowClone R0 into D3
4. ACTIVATE D1,D2,D3
5. RowClone Result into C
In-DRAM AND/OR Results

- 20X improvement in AND/OR throughput vs. Intel AVX
- 50.5X reduction in memory energy consumption
- At least 30% performance improvement in range queries

Going Forward

- A bulk computation model in memory

- New memory & software interfaces to enable bulk in-memory computation

- New programming models, algorithms, compilers, and system designs that can take advantage of the model
Challenge and Opportunity

Primitives and Interfaces for Computation in Memory
Departing From “Business as Usual”

Memory No Longer a Dumb Device
Two Approaches to In-Memory Processing

1. **Minimally change DRAM** to enable simple yet powerful computation primitives
   - RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
   - Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)

2. **Exploit the control logic in 3D-stacked memory** to enable more comprehensive computation near memory
   - A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing (Ahn et al., ISCA 2015)
Key Bottlenecks in Graph Processing

```cpp
for (v: graph.vertices) {
    for (w: v.successors) {
        w.next_rank += weight * v.rank;
    }
}
```

1. Frequent random memory accesses
2. Little amount of computation
Challenges in Scalable Graph Processing

- **Challenge 1**: How to provide *high memory bandwidth* to computation units in a practical way?
  - Processing-in-memory based on 3D-stacked DRAM

- **Challenge 2**: How to design computation units that *efficiently exploit large memory bandwidth*?
  - Specialized in-order cores called *Tesseract* cores
    - Latency-tolerant programming model
    - Graph-processing-specific prefetching schemes
Tesseract System for Graph Processing

Host Processor

Memory-Mapped Accelerator Interface
Noncacheable, Physically Addressed

Crossbar Network

In-Order Core

LP
PF Buffer
MTP
Message Queue

DRAM Controller

NI
Tesseract System for Graph Processing

Host Processor

Memory-Mapped Accelerator Interface (Noncacheable, Physically Addressed)

Communication via Remote Function Calls

In-Order Core

Message Queue
Tesseract System for Graph Processing

Host Processor

Memory-Mapped Accelerator Interface
Noncacheable, Physically Addressed

Crossbar Network

Prefetching

DRAM Controller

Message Queue

LP
PF Buffer
MTP

SAFARI
### Evaluated Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Cores</th>
<th>Frequency</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR3-OoO (with FDP)</td>
<td>8</td>
<td>4GHz</td>
<td>102.4GB/s</td>
</tr>
<tr>
<td>HMC-OoO (with FDP)</td>
<td>8</td>
<td>4GHz</td>
<td>640GB/s</td>
</tr>
<tr>
<td>HMC-MC</td>
<td>128 In-Order 2GHz</td>
<td>640GB/s</td>
<td></td>
</tr>
<tr>
<td>Tesseract</td>
<td>32 Cores</td>
<td>8TB/s</td>
<td></td>
</tr>
</tbody>
</table>

(with FDP) denotes the use of Fused Data Processing (FDP) technology.
Workloads

- Five graph processing algorithms
  - Average teenage follower
  - Conductance
  - PageRank
  - Single-source shortest path
  - Vertex cover

- Three real-world large graphs
  - ljournal-2008 (social network)
  - enwiki-2003 (Wikipedia)
  - indochina-0024 (web graph)
  - 4~7M vertices, 79~194M edges
Tesseract Graph Processing Performance

Speedup

<table>
<thead>
<tr>
<th>DDR3-OoO</th>
<th>HMC-OoO</th>
<th>HMC-MC</th>
<th>Tesseract</th>
<th>Tesseract-LP</th>
<th>Tesseract-LP-MTP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+56%</td>
<td>+25%</td>
<td>9.0x</td>
<td>11.6x</td>
<td>13.8x</td>
</tr>
</tbody>
</table>
Tesseract Graph Processing Performance

Memory Bandwidth Consumption

- DDR3-OoO: 80GB/s
- HMC-OoO: 190GB/s
- HMC-MC: 243GB/s
- Tesseract: 1.3TB/s
- Tesseract-LP: 2.2TB/s
- Tesseract-LP-MTP: 2.9TB/s

Memory Bandwidth (TB/s)

0 0.5 1 1.5 2 2.5 3 3.5

Memory Bandwidth Consumption

- DDR3-OoO
- HMC-OoO
- HMC-MC
- Tesseract
- Tesseract-LP
- Tesseract-LP-MTP
Memory Energy Consumption (Normalized)

- HMC-OoO: -87%
- Tesseract with Prefetching

- Memory Layers
- Logic Layers
- Cores
Challenge and Opportunity

Memory
Bandwidth
and
Energy
Departing From “Business as Usual”

Memory No Longer a Dumb Device

Dynamic Management of Margins in Memory
Rethinking Memory Architecture

- Compute Capable Memory
- Refresh
- Reliability
- Latency
- Bandwidth
- Energy
- Memory Compression
DRAM Refresh

- DRAM capacitor charge leaks over time

- The memory controller needs to refresh each row periodically to restore charge
  - Activate each row every $N$ ms
  - Typical $N = 64$ ms

- Downsides of refresh
  - **Energy consumption**: Each refresh consumes energy
  - **Performance degradation**: DRAM rank/bank unavailable while refreshed
  - **QoS/predictability impact**: (Long) pause times during refresh
  - **Refresh rate limits DRAM capacity scaling**
Refresh Overhead: Performance

Refresh Overhead: Energy

Retention Time Profile of DRAM

64-128ms

>256ms

128-256ms
**RAIDR: Eliminating Unnecessary Refreshes**

- **Observation:** Most DRAM rows can be refreshed much less often without losing data [Kim+, EDL’09][Liu+ ISCA’13]

- **Key idea:** Refresh rows containing weak cells more frequently, other rows less frequently

  1. **Profiling:** Profile retention time of all rows
  2. **Binning:** Store rows into bins by retention time in memory controller
    - Efficient storage with Bloom Filters (only 1.25KB for 32GB memory)
  3. **Refreshing:** Memory controller refreshes rows in different bins at different rates

- **Results:** 8-core, 32GB, SPEC, TPC-C, TPC-H
  - 74.6% refresh reduction @ 1.25KB storage
  - ~16%/20% DRAM dynamic/idle power reduction
  - ~9% performance improvement
  - Benefits increase with DRAM capacity

Experimental DRAM Testing Infrastructure

- An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)
- The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)
- Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)
- Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)
Experimental Infrastructure (DRAM)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study

Samira Khan†*
samirakhan@cmu.edu

Donghyuk Lee†
donghyuk1@cmu.edu

Yoongu Kim†
yoongukim@cmu.edu

Alaa R. Alameldeen*
aliaa.r.alameldeen@intel.com

Chris Wilkerson*
chris.wilkerson@intel.com

Onur Mutlu†
onur@cmu.edu

†Carnegie Mellon University  *Intel Labs
Online Profiling of DRAM In the Field

Initially protect DRAM with ECC

1

Periodically test parts of DRAM

2

Test
Test
Test

Adjust refresh rate and reduce ECC

3

Optimize DRAM and mitigate errors online without disturbing the system and applications
Challenge and Opportunity

Minimizing Refresh (and Other Technology Taxes)
Departing From “Business as Usual”

Online Detection and Management of Memory Errors

(Online Avoidance of Technology Taxes)
Rethinking Memory Architecture

- In-Memory Computation
- Refresh
- Reliability
- Latency
- Bandwidth
- Energy
- Memory Compression

Many More Challenges and Opportunities
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary
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])
    - 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?
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
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
Phase Change Memory: Pros and Cons

- **Pros over DRAM**
  - Better technology scaling (capacity and cost)
  - Non volatility
  - 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
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\(\times\) DRAM

**Latency**
- 50ns Rd, 150ns Wr
- 4\(\times\), 12\(\times\) DRAM

**Endurance**
- 1E+08 writes
- 1E-08\(\times\) DRAM

**Energy**
- 40\(\mu\)A Rd, 150\(\mu\)A 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

**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?
Challenge and Opportunity

Enabling an Emerging Technology to Replace DRAM
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.
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

Hybrid vs. All-PCM/DRAM [ICCD’12]

31% better performance than all PCM, within 29% of all DRAM performance

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
  - Non volatility
  - Low idle power (no refresh)

- Cons
  - Higher write latency
  - Higher write energy
  - 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

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
The traditional two-level storage model is a bottleneck with NVM

- **Volatile** data in memory $\rightarrow$ a **load/store** interface
- **Persistent** data in storage $\rightarrow$ 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
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

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

The Persistent Memory Manager (PMM)

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

0.044, 0.009

~24X, ~5X

Energy Benefits of a Single-Level Store

Challenge and Opportunity

Combined Memory & Storage
Departing From “Business as Usual”

A Unified Interface to All Data
One Challenge

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

- Two extremes
  - Programmer transparent: Let the system handle everything
  - Programmer only: Let the programmer handle everything
  - Many alternatives in-between...
CHALLENGE: CRASH CONSISTENCY

System crash can result in permanent data corruption in NVM
CURRENT SOLUTIONS

Explicit interfaces to manage consistency

– NV-Heaps [ASPLOS’11], BPFS [SOSP’09], Mnemosyne [ASPLOS’11]

```
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
OUR APPROACH: ThyNVM

Goal:
Software transparent consistency in persistent memory systems
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
More About ThyNVM

Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary
Principles (So Far)

- **Better interfaces between layers of the system stack**
  - Expose more information judiciously across the system stack
  - Design more flexible and efficient interfaces

- **Better-than-worst-case design**
  - Do not optimize for the worst case
  - Worst case should not determine the common case

- **Heterogeneity in design (specialization, asymmetry)**
  - Enables a more efficient design (No one size fits all)

- **These principles are coupled**
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary
## Summary

<table>
<thead>
<tr>
<th>Business as Usual</th>
<th>Opportunity</th>
</tr>
</thead>
<tbody>
<tr>
<td>RowHammer</td>
<td>Memory controller anticipates and fixes errors</td>
</tr>
<tr>
<td>Fixed, frequent refreshes</td>
<td>Heterogeneous refresh rate across memory</td>
</tr>
<tr>
<td>Fixed, high latency</td>
<td>Heterogeneous latency in time and space</td>
</tr>
<tr>
<td>Slow page copy &amp; initialization</td>
<td>Exploit internal connectivity in memory to move data</td>
</tr>
<tr>
<td>Fixed reliability mechanisms</td>
<td>Heterogeneous reliability across time and space</td>
</tr>
<tr>
<td>Memory as a dumb device</td>
<td>Memory as an accelerator and autonomous agent</td>
</tr>
<tr>
<td>DRAM-only main memory</td>
<td>Emerging memory technologies and hybrid memories</td>
</tr>
<tr>
<td>Two-level data storage model</td>
<td>Unified interface to and management of all data</td>
</tr>
<tr>
<td>Large timing and error margins</td>
<td>Online adaptation of timing and error margins</td>
</tr>
<tr>
<td>Poor performance guarantees</td>
<td>Strong service guarantees and configurable QoS</td>
</tr>
<tr>
<td>Fixed policies in controllers</td>
<td>Configurable and programmable memory controllers</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Summary

- Memory problems are a critical bottleneck for system performance, efficiency, and usability

- New memory architectures
  - Compute capable and autonomous memory

- Enabling emerging NVM technologies
  - Persistent and hybrid memory

- System-level memory/storage QoS
  - Predictable systems with configurable QoS

- Many opportunities and challenges that will change the systems and software we design
Acknowledgments

- **My current and past students and postdocs**
  - Rachata Ausavarungnirun, Abhishek Bhowmick, Amirali Boroumand, Rui Cai, Yu Cai, Kevin Chang, Saugata Ghose, Kevin Hsieh, Tyler Huberty, Ben Jaiyen, Samira Khan, Jeremie Kim, Yoongu Kim, Yang Li, Jamie Liu, Lavanya Subramanian, Donghyuk Lee, Yixin Luo, Justin Meza, Gennady Pekhimenko, Vivek Seshadri, Lavanya Subramanian, Nandita Vijaykumar, HanBin Yoon, Jishen Zhao, ...

- **My collaborators**
  - Can Alkan, Chita Das, Phil Gibbons, Sriram Govindan, Norm Jouppi, Mahmut Kandemir, Mike Kozuch, Konrad Lai, Ken Mai, Todd Mowry, Yale Patt, Moinuddin Qureshi, Partha Ranganathan, Bikash Sharma, Kushagra Vaid, Chris Wilkerson, ...
Funding Acknowledgments

- NSF
- GSRC
- SRC
- CyLab
- AMD, Google, Facebook, HP Labs, Huawei, IBM, Intel, Microsoft, Nvidia, Oracle, Qualcomm, Rambus, Samsung, Seagate, VMware
Open Source Tools

- **Rowhammer**
  - https://github.com/CMU-SAFARI/rowhammer

- **Ramulator – Fast and Extensible DRAM Simulator**
  - https://github.com/CMU-SAFARI/ramulator

- **MemSim**
  - https://github.com/CMU-SAFARI/memsim

- **NOCulator**
  - https://github.com/CMU-SAFARI/NOCulator

- **DRAM Error Model**
  - http://www.ece.cmu.edu/~safari/tools/memerr/index.html

- **Other open-source software from my group**
  - https://github.com/CMU-SAFARI/
  - http://www.ece.cmu.edu/~safari/tools.html
Referenced Papers

- All are available at
  http://users.ece.cmu.edu/~omutlu/projects.htm
  http://scholar.google.com/citations?user=7XyGUGkAAAAJ&hl=en

- A detailed accompanying overview paper
  
  - Onur Mutlu and Lavanya Subramanian,
    "Research Problems and Opportunities in Memory Systems"
    Invited Article in Supercomputing Frontiers and Innovations (SUPERFRI), 2015.
Related Videos and Course Materials

- Parallel Computer Architecture Course Materials (Lecture Videos)
- Memory Systems Short Course Materials (Lecture Video on Main Memory and DRAM Basics)
Thank you.

onur.mutlu@inf.ethz.ch
http://users.ece.cmu.edu/~omutlu/
Rethinking Memory System Design
Business as Usual in the Next Decade?

Onur Mutlu
onur.mutlu@inf.ethz.ch
http://users.ece.cmu.edu/~omutlu/

October 6, 2016
RSP 2016 Keynote (Pittsburgh)
Backup Slides
NAND Flash Memory Scaling
Another Talk: NAND Flash Scaling Challenges

Onur Mutlu,
"Error Analysis and Management for MLC NAND Flash Memory"
*Technical talk at Flash Memory Summit 2014 (FMS),* Santa Clara, CA, August 2014. [Slides (ppt) (pdf)]


Meza+, “A Large-Scale Study of Flash Memory Errors in the Field,” SIGMETRICS 2015.
Experimental Infrastructure (Flash)

Problem: MLC NAND flash memory reliability/endurance is a key challenge for satisfying future storage systems’ requirements

Our Goals: (1) Build reliable error models for NAND flash memory via experimental characterization, (2) Develop efficient techniques to improve reliability and endurance

This talk provides a “flash” summary of our recent results published in the past 3 years:

- Experimental error and threshold voltage characterization [DATE’12&13]
- Retention-aware error management [ICCD’12]
- Program interference analysis and read reference V prediction [ICCD’13]
- Neighbor-assisted error correction [SIGMETRICS’14]
Ramulator: A Fast and Extensible DRAM Simulator

[IEEE Comp Arch Letters’15]
Ramulator Motivation

- DRAM and Memory Controller landscape is changing
- Many new and upcoming standards
- Many new controller designs
- A fast and easy-to-extend simulator is very much needed

<table>
<thead>
<tr>
<th>Segment</th>
<th>DRAM Standards &amp; Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commodity</td>
<td>DDR3 (2007) [14]; DDR4 (2012) [18]</td>
</tr>
<tr>
<td>Performance</td>
<td>eDRAM [28], [32]; RLDRAM3 (2011) [29]</td>
</tr>
</tbody>
</table>

Table 1. Landscape of DRAM-based memory
Ramulator

- Provides out-of-the-box support for many DRAM standards:
  - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, plus new proposals (SALP, AL-DRAM, TLDRAAM, RowClone, and SARP)
- \(~2.5\times\) faster than fastest open-source simulator
- Modular and extensible to different standards

<table>
<thead>
<tr>
<th>Simulator (clang -O3)</th>
<th>Cycles (10^6)</th>
<th>Runtime (sec.)</th>
<th>Req/sec (10^3)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random</td>
<td>Stream</td>
<td>Random</td>
<td>Stream</td>
</tr>
<tr>
<td>Ramulator</td>
<td>652</td>
<td>411</td>
<td>752</td>
<td>249</td>
</tr>
<tr>
<td>DRAMSim2</td>
<td>645</td>
<td>413</td>
<td>2,030</td>
<td>876</td>
</tr>
<tr>
<td>USIMM</td>
<td>661</td>
<td>409</td>
<td>1,880</td>
<td>750</td>
</tr>
<tr>
<td>DrSim</td>
<td>647</td>
<td>406</td>
<td>18,109</td>
<td>12,984</td>
</tr>
<tr>
<td>NVMain</td>
<td>666</td>
<td>413</td>
<td>6,881</td>
<td>5,023</td>
</tr>
</tbody>
</table>

Table 3. Comparison of five simulators using two traces
Case Study: Comparison of DRAM Standards

<table>
<thead>
<tr>
<th>Standard</th>
<th>Rate (MT/s)</th>
<th>Timing (CL-RCD-RP)</th>
<th>Data-Bus (Width x Chan.)</th>
<th>Rank-per-Chan</th>
<th>BW (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR3</td>
<td>1,600</td>
<td>11-11-11</td>
<td>64-bit x 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>DDR4</td>
<td>2,400</td>
<td>16-16-16</td>
<td>64-bit x 1</td>
<td>1</td>
<td>17.9</td>
</tr>
<tr>
<td>SALP†</td>
<td>1,600</td>
<td>11-11-11</td>
<td>64-bit x 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>LPDDR3</td>
<td>1,600</td>
<td>12-15-15</td>
<td>64-bit x 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>LPDDR4</td>
<td>2,400</td>
<td>22-22-22</td>
<td>32-bit x 2*</td>
<td>1</td>
<td>17.9</td>
</tr>
<tr>
<td>GDDR5 [12]</td>
<td>6,000</td>
<td>18-18-18</td>
<td>64-bit x 1</td>
<td>1</td>
<td>44.7</td>
</tr>
<tr>
<td>HBM</td>
<td>1,000</td>
<td>7-7-7</td>
<td>128-bit x 8*</td>
<td>1</td>
<td>119.2</td>
</tr>
<tr>
<td>WIO</td>
<td>266</td>
<td>7-7-7</td>
<td>128-bit x 4*</td>
<td>1</td>
<td>15.9</td>
</tr>
<tr>
<td>WIO2</td>
<td>1,066</td>
<td>9-10-10</td>
<td>128-bit x 8*</td>
<td>1</td>
<td>127.2</td>
</tr>
</tbody>
</table>

Across 22 workloads, simple CPU model
Ramulator Paper and Source Code


- Source code is released under the liberal MIT License
  - https://github.com/CMU-SAFARI/ramulator
DRAM Infrastructure
Experimental DRAM Testing Infrastructure

An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems (Qureshi et al., DSN 2015)
Experimental Infrastructure (DRAM)

ThyNVM
CHALLENGE: CRASH CONSISTENCY

Persistent Memory System

System crash can result in permanent data corruption in NVM
CURRENT SOLUTIONS

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

```
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
OUR APPROACH: ThyNVM

Goal:
Software transparent consistency in persistent memory systems
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
ThyNVM: Transparent Hybrid NVM

- **Problem**: How do you provide consistency and prevent data corruption in NVM upon a system crash?

- **Goal**: Provide **efficient programmer-transparent crash consistency** in hybrid NVM
  - **Transparency**: no library APIs or explicit interfaces to access NVM; just loads and stores
    - Easier to support legacy code and hypervisors
    - No programmer effort to adopt persistent memory
  - **Efficiency**: use hybrid DRAM/NVM for high performance

---

Idea 1: Transparent periodic checkpointing of data

- Need to overlap checkpointing and execution

Idea 2: Differentiated checkpointing schemes for different types of updates

- Page Writeback: for sequential accesses (use DRAM)
- Address Remapping: for random accesses (use NVM/DRAM)

Idea 3: Coordination/switching between checkpointing schemes for high performance
# Checkpointing Tradeoffs in Hybrid Memory

<table>
<thead>
<tr>
<th>Location of working copy</th>
<th>Checkpointing granularity</th>
</tr>
</thead>
</table>
| DRAM (based on writeback) | **1** *Inefficient*  
× Large metadata overhead  
× Long checkpointing latency |
| NVM (based on remapping)  | **2** *Partially efficient*  
✓ Small metadata overhead  
× Long checkpointing latency |
|                           | **3** *Partially efficient*  
× Large metadata overhead  
✓ **Short checkpointing latency**  
✓ Fast remapping |
|                           | **4** *Inefficient*  
✓ Small metadata overhead  
✓ Short checkpointing latency  
× **Slow remapping**  
(on the critical path) |
ThyNVM: Dual-Scheme Checkpointing

- **Idea:** Combine two types of checkpointing schemes to adapt to different types of access patterns

- Sparse updates with low spatial locality $\rightarrow$ address remapping $\rightarrow$ block granularity checkpointing $\rightarrow$ working copy stored in NVM (for short ckpt latency)

- Dense updates with high spatial locality $\rightarrow$ page writeback $\rightarrow$ page granularity checkpointing (small metadata) $\rightarrow$ working copy stored in DRAM for fast buffering; written back to NVM during ckpt.

- Can switch between schemes when one is on critical path
ThyNVM Performance (I)

- In-memory storage workloads

8.8%/29.9% higher throughput than journaling/shadow paging with a hash table based key-value store

ThyNVM Performance (II)

- Legacy compute-intensive workloads

- Within 3.4% of Ideal DRAM,
- 2.7% higher performance than Ideal NVM.

New Memory Architectures

- Compute Capable Memory
- Refresh
- Reliability
- **Latency**
- Bandwidth
- Energy
- Memory Compression
DRAM Latency
New Memory Architectures

- Compute Capable Memory
- Refresh
- Reliability
- Latency
- Bandwidth
- Energy
- Memory Compression
DRAM latency continues to be a critical bottleneck, especially for response time-sensitive workloads.
What Causes the Long Memory Latency?

- **Conservative timing margins!**
- DRAM timing parameters are set to cover the worst case

- Worst-case temperatures
  - 85 degrees vs. common-case
  - to enable a wide range of operating conditions

- Worst-case devices
  - DRAM cell with smallest charge across any acceptable device
  - to tolerate process variation at acceptable yield

- This leads to large timing margins for the common case
Adaptive-Latency DRAM [HPCA 2015]

- **Idea:** Optimize DRAM timing for the common case
  - Current temperature
  - Current DRAM module

- Why would this reduce latency?
  - A DRAM cell can store much more charge in the common case (low temperature, strong cell) than in the worst case
  - More charge in a DRAM cell
    - Faster sensing, charge restoration, precharging
    - Faster access (read, write, refresh, ...)

AL-DRAM

• **Key idea**
  – Optimize DRAM timing parameters online

• **Two components**
  – DRAM manufacturer provides multiple sets of reliable DRAM timing parameters at different temperatures for each DIMM
  – System monitors DRAM temperature & uses appropriate DRAM timing parameters

Experimental DRAM Testing Infrastructure

An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems (Qureshi et al., DSN 2015)
Latency Reduction Summary of 115 DIMMs

• *Latency reduction for read & write (55°C)*
  – Read Latency: **32.7%**
  – Write Latency: **55.1%**

• *Latency reduction for each timing parameter (55°C)*
  – Sensing: **17.3%**
  – Restore: **37.3%** (read), **54.8%** (write)
  – Precharge: **35.2%**
AL-DRAM: Real System Evaluation

- **System**
  - **CPU**: AMD 4386 (8 Cores, 3.1GHz, 8MB LLC)

---

### DDR3 DRAM Timing 0

Reset: 0F05_0505h. See 2.9.3 [DCT Configuration Registers].

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:30</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Tras: row active strobe.** Read-write. BIOS: See 2.9.7.5 [SPD ROM-Based Configuration]. Specifies the minimum time in memory clock cycles from an activate command to a precharge command, both to the same chip select bank.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>07h-00h</td>
<td>Reserved</td>
</tr>
<tr>
<td>2Ah-08h</td>
<td>&lt;Tras&gt; clocks</td>
</tr>
<tr>
<td>3Fh-2Bh</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Trp: row precharge time.** Read-write. BIOS: See 2.9.7.5 [SPD ROM-Based Configuration]. Specifies the minimum time in memory clock cycles from a precharge command to an activate command or auto refresh command, both to the same bank.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20:16</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
AL-DRAM: Single-Core Evaluation

AL-DRAM improves performance on a real system
AL-DRAM: Multi-Core Evaluation

Performance Improvement

<table>
<thead>
<tr>
<th>Workload</th>
<th>Single Core</th>
<th>Multi Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>gups</td>
<td>14.0%</td>
<td>10.4%</td>
</tr>
<tr>
<td>s.cluster</td>
<td>2.9%</td>
<td></td>
</tr>
<tr>
<td>all-35-workload</td>
<td>14.0%</td>
<td>10.4%</td>
</tr>
<tr>
<td>non-intensive</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>intensive</td>
<td>0%</td>
<td></td>
</tr>
</tbody>
</table>

AL-DRAM provides higher performance for multi-programmed & multi-threaded workloads
ChargeCache
ChargeCache: Executive Summary

• **Goal**: Reduce average DRAM access latency with no modification to the existing DRAM chips

• **Observations**:
  1) A highly-charged DRAM row can be accessed with low latency
  2) A row’s charge is restored when the row is accessed
  3) A recently-accessed row is likely to be accessed again:
     **Row Level Temporal Locality (RLTL)**

• **Key Idea**: Track recently-accessed DRAM rows and use lower timing parameters if such rows are accessed again

• **ChargeCache**:
  – Low cost & no modifications to the DRAM
  – Higher performance (8.6-10.6% on average for 8-core)
  – Lower DRAM energy (7.9% on average)
Accessing Highly-charged Rows

- Ready to Access
- Ready to Precharge

Charge vs. Time

Cell

Sense-Amplifier

Data 1

Data 0

Sensing

Restore

Precharge

ACT

R/W

PRE

tRCD

tRAS

SAFARI
Observation 1

A highly-charged DRAM row can be accessed with low latency

- tRCD: 44%
- tRAS: 37%

How does a row become highly-charged?
How Does a Row Become Highly-Charged?

DRAM cells lose charge over time

Two ways of restoring a row’s charge:

• Refresh Operation
• Access
Observation 2

A row’s charge is restored when the row is accessed

How likely is a recently-accessed row to be accessed again?
Row Level Temporal Locality (RLTL)

A recently-accessed DRAM row is likely to be accessed again.

• $t$-RLTL: Fraction of rows that are accessed within time $t$ after their previous access

88ms—RLTL for eight-core workloads
Key Idea

Track recently-accessed DRAM rows and use lower timing parameters if such rows are accessed again.
ChargeCache Overview

Requests: A D A

ChargeCache Hit: Use Default Timings
Area and Power Overhead

• Modeled with CACTI

• Area
  – \( \sim 5\text{KB} \) for 128-entry ChargeCache
  – 0.24\% of a 4MB Last Level Cache (LLC) area

• Power Consumption
  – 0.15 mW on average (static + dynamic)
  – 0.23\% of the 4MB LLC power consumption
Methodology

• Simulator
  – DRAM Simulator (Ramulator [Kim+, CAL’15])
    https://github.com/CMU-SAFARI/ramulator

• Workloads
  – 22 single-core workloads
    • SPEC CPU2006, TPC, STREAM
  – 20 multi-programmed 8-core workloads
    • By randomly choosing from single-core workloads
  – Execute at least 1 billion representative instructions per core (Pinpoints)

• System Parameters
  – 1/8 core system with 4MB LLC
  – Default tRCD/tRAS of 11/28 cycles
ChargeCache improves single-core performance
Eight-core Performance

- NUAT: 2.5%
- ChargeCache: 9%
- ChargeCache + NUAT
- LL-DRAM (Upperbound): 13%

ChargeCache significantly improves multi-core performance
ChargeCache reduces DRAM energy

DRAM Energy Savings

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-core</td>
<td>5%</td>
<td></td>
</tr>
<tr>
<td>Eight-core</td>
<td>10%</td>
<td>15%</td>
</tr>
</tbody>
</table>
More on ChargeCache


- Source code will be released as part of Ramulator (May 2016) https://github.com/CMU-SAFARI/ramulator
Tiered Latency DRAM
What Causes the Long Latency?

```
DRAM Latency = Subarray Latency + I/O Latency
```

Dominant Subarray I/O
Why is the Subarray So Slow?

- Long bitline
  - Amortizes sense amplifier cost → Small area
  - Large bitline capacitance → High latency & power
Trade-Off: Area (Die Size) vs. Latency

Long Bitline

Short Bitline

Faster

Smaller

Trade-Off: Area vs. Latency
Trade-Off: Area (Die Size) vs. Latency

- **Cheaper**
- **Faster**

- **Normalized DRAM Area**
- **Latency (ns)**

- **GOAL**

- Fancy DRAM Short Bitline
- Commodity DRAM Long Bitline

- 512 cells/bitline
Approximating the Best of Both Worlds

Long Bitline
- Small Area
- High Latency

Our Proposal
- Add Isolation Transistors

Short Bitline
- Large Area
- Low Latency

Need
Isolation

Fast
Approximating the Best of Both Worlds

Long Bitline Tiered-Latency DRAM

Small Area

High Latency

Small Area

Low Latency

Large Area

Low Latency

Small area using long bitline

Low Latency
Commodity DRAM vs. TL-DRAM [HPCA 2013]

- DRAM Latency (tRC)
- DRAM Power

- DRAM Area Overhead

~3%: mainly due to the isolation transistors
Trade-Off: Area (Die-Area) vs. Latency

- Cheaper
- Normalized DRAM Area
- Latency (ns)

Near Segment
Far Segment

GOAL

Faster

512 cells/bitline

32
64
128
256
Leveraging Tiered-Latency DRAM

• TL-DRAM is a *substrate* that can be leveraged by the hardware and/or software

• Many potential uses

1. Use near segment as hardware-managed *inclusive* cache to far segment
2. Use near segment as hardware-managed *exclusive* cache to far segment
3. Profile-based page mapping by operating system
4. Simply replace DRAM with TL-DRAM

Using near segment as a cache improves performance and reduces power consumption

Architecture-Aware DRM
Virtualized Cluster

Distributed Resource Management (DRM) policies
Conventional DRM Policies

Based on operating-system-level metrics e.g., CPU utilization, memory capacity demand
Microarchitecture-level Interference

- VMs within a host compete for:
  - Shared cache capacity
  - Shared memory bandwidth

Can operating-system-level metrics capture the microarchitecture-level resource interference?
## Microarchitecture Unawareness

<table>
<thead>
<tr>
<th>VM</th>
<th>Operating-system-level metrics</th>
<th></th>
<th>Microarchitecture-level metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU Utilization</td>
<td>Memory Capacity</td>
<td>LLC Hit Ratio</td>
</tr>
<tr>
<td>App</td>
<td>92%</td>
<td>369 MB</td>
<td>2%</td>
</tr>
<tr>
<td>App</td>
<td>93%</td>
<td>348 MB</td>
<td>98%</td>
</tr>
</tbody>
</table>

### Microarchitecture-level metrics

- **LLC Hit Ratio**: 2% for VM App, 98% for another VM App
- **Memory Bandwidth**: 2267 MB/s for VM App, 1 MB/s for another VM App

### Diagram

- **CPU**: Stream and gromacs
- **Memory Capacity**: VM App
- **Host**: Core0, Core1, LLC, DRAM

---

*SAFARI*
Impact on Performance

IPC (Harmonic Mean)

Conventional DRM

CPU

Memory Capacity

VM

App

Core0

Core1

Host

DRAM

LLC

SWAP

App

STREAM

gromacs

SAFARI
Impact on Performance

We need microarchitecture-level interference awareness in DRM!
A-DRM: Architecture-aware DRM

• **Goal**: Take into account microarchitecture-level shared resource interference
  – Shared cache capacity
  – Shared memory bandwidth

• **Key Idea**:
  – Monitor and detect microarchitecture-level shared resource interference
  – Balance microarchitecture-level resource usage across cluster to minimize memory interference while maximizing system performance
A-DRM: Architecture-aware DRM

**Hosts**

- OS+Hypervisor
  - VM
    - App
  - CPU/Memory Capacity
  - Architectural Resources

**Profiler**

**Controller**

  - Profiling Engine
  - Architecture-aware Interference Detector
  - Architecture-aware Distributed Resource Management (Policy)
  - Migration Engine
More on Architecture-Aware DRM

- Hui Wang, Canturk Isci, Lavanya Subramanian, Jongmoo Choi, Depei Qian, and Onur Mutlu,

*A-DRM: Architecture-aware Distributed Resource Management of Virtualized Clusters*

Proceedings of the
11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE), Istanbul, Turkey, March 2015.
[Slides (pptx) (pdf)]

A-DRM: Architecture-aware Distributed Resource Management of Virtualized Clusters

Hui Wang†*, Canturk Isci‡, Lavanya Subramanian*, Jongmoo Choi‡*, Depei Qian†, Onur Mutlu*

†Beihang University, ‡IBM Thomas J. Watson Research Center, *Carnegie Mellon University, †Dankook University

{hui.wang, depeiq}@buaa.edu.cn, canturk@us.ibm.com, {lsubrama, onur}@cmu.edu, choijm@dankook.ac.kr
Other Research
Current Research Focus Areas

Research Focus: Computer architecture, HW/SW, bioinformatics

- Memory, memory, memory, storage, interconnects
- Parallel architectures, heterogeneous architectures, GP-GPUs
- System/architecture interaction, new execution models
- Energy efficiency, fault tolerance, hardware security
- Genome sequence analysis & assembly algorithms and architectures

Broad research spanning apps, systems, logic with architecture at the center.