Rethinking Memory System Design (for Data-Intensive Computing)

Onur Mutlu
onur@cmu.edu
http://users.ece.cmu.edu/~omutlu/
September 23, 2015
Stanford University
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**
The Main Memory System

- 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
- How Can We Do Better?
- 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
- How Can We Do Better?
- 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.

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)
x86 CPU

DRAM Module

loop:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp loop

https://github.com/CMU-SAFARI/rowhammer
x86 CPU

DRAM Module

**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)

RowHammer Characterization Results

1. Most Modules Are at Risk
2. Errors vs. Vintage
3. Error = Charge Loss
4. Adjacency: Aggressor & Victim
5. Sensitivity Studies
6. Other Results in Paper
7. Solution Space

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

Abstract. Memory isolation is a key property of a reliable and secure computing system — an access to one memory address should not have unintended side effects on data stored in other addresses. However, as DRAM process technology

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

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.
Rowhammer

It’s like breaking into an apartment by repeatedly slaming a neighbor’s door until the vibrations open the door you were after
Recap: The DRAM Scaling Problem

DRAM Process Scaling Challenges

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

THE MEMORY FORUM 2014

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 DRAM and controllers more intelligent
  - New interfaces, new architectures: system-DRAM 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: Fix DRAM

- Overcome DRAM shortcomings with
  - System-DRAM co-design
  - Novel DRAM architectures, interface, 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: Fix DRAM

- 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


Yoon, Meza et al., "Row Buffer Locality Aware Caching Policies for Hybrid Memories," ICCD 2012 Best Paper Award.

Hardware/software manage data allocation and movement to achieve the best of multiple technologies.
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]
An Orthogonal Issue: Memory Interference

Cores’ interfere with each other when accessing shared main memory.
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

Some Promising Directions

- **New memory architectures**
  - Rethinking DRAM and flash memory
  - A lot of hope in fixing DRAM

- **Enabling emerging NVM technologies**
  - Hybrid memory systems
  - Single-level memory and storage
  - A lot of hope in hybrid memory systems and single-level stores

- **System-level memory/storage QoS**
  - A lot of hope in designing a predictable system
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- How Can We Do Better?
- Summary
Rethinking DRAM

- In-Memory Computation
- Refresh
- Reliability
- Latency
- Bandwidth
- Energy
- Memory Compression
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
   - **PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture** (Ahn et al., ISCA 2015)
   - **A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing** (Ahn et al., ISCA 2015)
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

CPU → L1 → L2 → L3 → MC → Memory

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

Step 1: Activate row

Step 2: Read
Transfer byte onto bus

8 Kbits

Data Bus

Row Buffer (8 Kbits)

DRAM array
RowClone: In-DRAM Row Copy

Step 1: Activate row A
Transfer row

Step 2: Activate row B
Transfer row

0.01% area cost

8 Kbits

DRAM array

Row Buffer (8 Kbits)

8 bits

Data Bus
Generalized RowClone

Inter Subarray Copy
(Use Inter-Bank Copy Twice)

Inter Bank Copy
(Pipelined Internal RD/WR)

Intra Subarray Copy (2 ACTs)

0.01% area cost
RowClone: Latency and Energy Savings

RowClone: Application Performance

![Bar chart showing IPC Improvement and Energy Reduction for various applications: bootup, compile, forkbench, mcached, mysql, and shell.](chart.png)
RowClone: Multi-Core Performance

Normalized Weighted Speedup

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 Search

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

Final State

\[ AB + BC + AC \]

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

In-DRAM Bulk Bitwise AND/OR Operation

- **BULKAND A, B → 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
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)
Rethinking DRAM

- In-Memory Computation
- 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

---

Going Forward (for DRAM and Flash)

- **How to find out weak memory cells/rows**

- **Low-cost system-level tolerance of memory errors**
  - Luo+，“Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost,” DSN 2014.

- **Tolerating cell-to-cell interference at the system level**
  - Kim+，“Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors,” ISCA 2014.
  - Cai+，“Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation,” ICCD 2013.
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)

More Information [ISCA’13, SIGMETRICS’14]

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

Samira Khan†*
samirakhan@cmu.edu

Donghyuk Lee†
dongyuk1@cmu.edu

Yoongu Kim†
yoongukim@cmu.edu

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

Chris Wilkerson*
chris.wilkerson@intel.com

Onur Mutlu†
onur@cmu.edu

†Carnegie Mellon University    *Intel Labs

SAFARI
Online Profiling of DRAM In the Field

1. Initially protect DRAM with ECC
2. Periodically test parts of DRAM
3. Adjust refresh rate and reduce ECC

Optimize DRAM and mitigate errors online without disturbing the system and applications
Rethinking DRAM

- In-Memory Computation
- 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 Latency?

DRAM Latency = Subarray Latency + I/O Latency

Dominant

DRAM Chip

channel

subarray

I/O

Subarray

I/O
Why is the Subarray So Slow?

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

Long Bitline → Faster

Short Bitline → Smaller

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

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

**Normalized DRAM Area**

- 0
- 1
- 2
- 3
- 4

**Latency (ns)**

- 0
- 10
- 20
- 30
- 40
- 50
- 60
- 70

- **GOAL**

**Commodity DRAM**
- Long Bitline

**Fancy DRAM**
- Short Bitline

**512 cells/bitline**
Approximating the Best of Both Worlds

Long Bitline
Small Area
High Latency

Our Proposal

Short Bitline
Large Area
Low Latency

Need Isolation
Add Isolation Transistors

Fast
Approximating the Best of Both Worlds

Long Bitline Tiered-Latency DRAM | Short Bitline

**Small Area**
- Low Latency
- Small Area

**Large Area**
- Low Latency

**High Latency**
- Small Area

Small area using long bitline

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

• DRAM Latency ($t_{RC}$) • DRAM Power

- Latency
  - Commodity DRAM
  - Near TL-DRAM: -56%
  - Far TL-DRAM: +23%
  - (52.5ns)

- Power
  - Commodity DRAM
  - Near TL-DRAM: -51%
  - Far TL-DRAM: +49%

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

- **Near Segment**: 32 cells/bitline
- **Far Segment**: 512 cells/bitline
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

What Else Causes the Long DRAM 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

SAFARI

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)

---

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:30</td>
<td>Reserved</td>
</tr>
<tr>
<td>29:24</td>
<td><strong>Tras</strong>: 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.</td>
</tr>
<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>
<tr>
<td>23:21</td>
<td>Reserved</td>
</tr>
<tr>
<td>20:16</td>
<td><strong>Trp</strong>: 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.</td>
</tr>
</tbody>
</table>
AL-DRAM improves performance on a real system.
AL-DRAM provides higher performance for multi-programmed & multi-threaded workloads
Rethinking DRAM

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

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- How Can We Do Better?
- Summary
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])
  - 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
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

<table>
<thead>
<tr>
<th>Density</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 - 12F² using BJT</td>
<td>50ns Rd, 150ns Wr</td>
</tr>
<tr>
<td>1.5× DRAM</td>
<td>4×, 12× DRAM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Endurance</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E+08 writes</td>
<td>40μA Rd, 150μA Wr</td>
</tr>
<tr>
<td>1E-08× DRAM</td>
<td>2×, 43× DRAM</td>
</tr>
</tbody>
</table>
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?
Solution 3: 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.
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

Energy Benefits of a Single-Level Store

Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- How Can We Do Better?
- Summary
Principles (So Far)

- Better cooperation between devices and the system
  - Expose more information about devices to upper layers
  - More flexible 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
- How Can We Do Better?
- Summary
Summary: Memory Scaling

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

- New memory architectures
  - A lot of hope in fixing DRAM

- Enabling emerging NVM technologies
  - A lot of hope in hybrid memory systems and single-level stores

- System-level memory/storage QoS
  - A lot of hope in designing a predictable system

- Three principles are essential for scaling
  - Software/hardware/device cooperation
  - Better-than-worst-case design
  - Heterogeneity (specialization, asymmetry)
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 at CMU
  - Greg Ganger, Phil Gibbons, Mor Harchol-Balter, James Hoe, Mike Kozuch, Ken Mai, Todd Mowry, ...

- My collaborators elsewhere
  - Can Alkan, Chita Das, Sriram Govindan, Norm Jouppi, Mahmut Kandemir, Konrad Lai, 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](https://github.com/CMU-SAFARI/rowhammer)

- **Ramulator**
  - [https://github.com/CMU-SAFARI/ramulator](https://github.com/CMU-SAFARI/ramulator)

- **MemSim**
  - [https://github.com/CMU-SAFARI/memsim](https://github.com/CMU-SAFARI/memsim)

- **NOCulator**
  - [https://github.com/CMU-SAFARI/NOCulator](https://github.com/CMU-SAFARI/NOCulator)

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

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


- Graduate Computer Architecture Course Materials (Lecture Videos)

- Parallel Computer Architecture Course Materials (Lecture Videos)

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

onur@cmu.edu
http://users.ece.cmu.edu/~omutlu/
Rethinking Memory System Design (for Data-Intensive Computing)

Onur Mutlu
onur@cmu.edu
http://users.ece.cmu.edu/~omutlu/
September 23, 2015
Stanford University
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)

Luo+，“WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management,” MSST 2015.
Meza+, “A Large-Scale Study of Flash Memory Errors in the Field,” SIGMETRICS 2015.
Experimental Infrastructure (Flash)

HAPS-52 Mother Board

USB Daughter Board

USB Jack

Virtex-V FPGA (NAND Controller)

USB Jack

virtex-II Pro (USB controller)

3x-nm NAND Flash

NAND Daughter Board

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, TLDRAM, RowClone, and SARP)
- ~2.5X 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

Figure 2. Performance comparison of DRAM standards
Ramulator Paper and Source Code


- Source code is released under the liberal MIT License
  - [https://github.com/CMU-SAFARI/ramulator](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)