Some New Ideas in Memory System Design for Data-Intensive Computing

Onur Mutlu
onur@cmu.edu
September 4, 2014
ISTC-CC Retreat

Carnegie Mellon
Some New Ideas (This Year)

- **Specialization**
  - Heterogeneous Reliability Memory [DSN 2014]
  - Heterogeneous Block Architecture [ICCD 2014]

- **Persistent Memory**
  - Loose Ordering Consistency for Persistent Memory [ICCD 2014]
  - Transparent Consistency for Persistent/Hybrid Memory [in progress]

- **Memory Reliability/Security**
  - **Row Hammer Problem in DRAM [ISCA 2014]**
  - Neighbor-Cell Assisted Error Correction in Flash [SIGMETRICS 2014]
  - Error Mitigation for Intermittent DRAM Failures [SIGMETRICS 2014]

- **Memory Performance**
  - The Dirty-Block Index [ISCA 2014]
  - DRAM Refresh-Access Parallelization [HPCA 2014]
  - The Blacklisting Memory Scheduler [ICCD 2014]
  - Exploiting Read-Write Disparity in Caches [HPCA 2014]
Memory Reliability Trends

- Memory is becoming less reliable as its density increases with technology scaling
  - Reduced retention times
  - Increased vulnerability to disturbance
  - New error types (e.g., due to inter-cell interference)
  - ...

- Maintaining reliability is expensive in terms of
  - Energy
  - Performance
  - Cost (TCO)
DRAM Scaling

- **DRAM technology scaling has provided many benefits**
  - Higher capacity
  - Lower cost
  - Reasonable energy

- **DRAM scaling is becoming difficult due to reduced reliability**
  - ITRS projects **DRAM will not scale easily below X nm**
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
The DRAM Scaling Problem

- DRAM scaling has become a real problem the system should be concerned about
  - And, maybe embrace
Flipping Bits in Memory Without Accessing Them

DRAM Disturbance Errors

Yoongu Kim
Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, Onur Mutlu

Carnegie Mellon SAFARI intel
Repeatedly opening and closing a row induces *disturbance errors* in adjacent rows in *most real DRAM chips* [Kim+ ISCA 2014]
Quick Summary

• **We expose the existence and prevalence of disturbance errors in DRAM chips of today**
  – 110 of 129 modules are vulnerable
  – Affects modules of 2010 vintage or later

• **We characterize the cause and symptoms**
  – Toggling a row accelerates charge leakage in adjacent rows: *row-to-row coupling*

• **We prevent errors using a system-level approach**
  – Each time a row is closed, we refresh the charge stored in its adjacent rows with a low probability

Experimental Infrastructure (DRAM)

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

x86 CPU

DRAM Module

\texttt{loop:}
\begin{verbatim}
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp \texttt{loop}
\end{verbatim}
loop:
    mov (X), %eax
    mov (Y), %ebx
    clflush (X)
    clflush (Y)
    mfence
    jmp loop
**loop:**

```
mov (X), %eax
mov (Y), %ebx
clflush (X)
clflush (Y)
mfence
jmp loop
```
loop:
    mov (X), %eax
    mov (Y), %ebx
    clflush (X)
    clflush (Y)
    mfence
    jmp loop
Observed Errors in Real Systems

<table>
<thead>
<tr>
<th>CPU Architecture</th>
<th>Errors</th>
<th>Access-Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Haswell (2013)</td>
<td>22.9K</td>
<td>12.3M/sec</td>
</tr>
<tr>
<td>Intel Ivy Bridge (2012)</td>
<td>20.7K</td>
<td>11.7M/sec</td>
</tr>
<tr>
<td>Intel Sandy Bridge (2011)</td>
<td>16.1K</td>
<td>11.6M/sec</td>
</tr>
<tr>
<td>AMD Piledriver (2012)</td>
<td>59</td>
<td>6.1M/sec</td>
</tr>
</tbody>
</table>

- In a more controlled environment, we can induce as many as ten million disturbance errors.
- A real reliability & security issue.

Security Implications

• Breach of memory protection
  – OS page (4KB) fits inside DRAM row (8KB)
  – Adjacent DRAM row \( \Rightarrow \) Different OS page

• Vulnerability: disturbance attack
  – By accessing its own page, a program could corrupt pages belonging to another program

• We constructed a proof-of-concept
  – Using only user-level instructions
Errors vs. Vintage

First Appearance

All modules from 2012–2013 are vulnerable
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
Several Potential Solutions

- Make better DRAM chips
  - Cost

- Refresh frequently
  - Power, Performance

- Sophisticated ECC
  - Cost, Power

- Access counters
  - Cost, Power, Complexity
Our Solution

• **PARA: Probabilistic Adjacent Row Activation**

• **Key Idea**
  – After closing a row, we activate (i.e., refresh) one of its neighbors with a low probability: \( p = 0.005 \)

• **Reliability Guarantee**
  – When \( p=0.005 \), errors in one year: \( 9.4 \times 10^{-14} \)
  – By adjusting the value of \( p \), we can provide an arbitrarily strong protection against errors
Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu,
"Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors"
Lightning Session Slides (pptx) (pdf) Source Code and Data
Some New Ideas (This Year)

- **Specialization**
  - Heterogeneous Reliability Memory [DSN 2014]
  - Heterogeneous Block Architecture [ICCD 2014]

- **Persistent Memory**
  - Loose Ordering Consistency for Persistent Memory [ICCD 2014]
  - Transparent Consistency for Persistent/Hybrid Memory [in progress]

- **Memory Reliability/Security**
  - Row Hammer Problem in DRAM [ISCA 2014]
  - Neighbor-Cell Assisted Error Correction in Flash [SIGMETRICS 2014]
  - Error Mitigation for Intermittent DRAM Failures [SIGMETRICS 2014]

- **Memory Performance**
  - The Dirty-Block Index [ISCA 2014]
  - DRAM Refresh-Access Parallelization [HPCA 2014]
  - The Blacklisting Memory Scheduler [ICCD 2014]
  - Exploiting Read-Write Disparity in Caches [HPCA 2014]
Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory

Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, Onur Mutlu
Executive Summary

• **Problem:** Reliable memory hardware increases cost

• **Our Goal:** Reduce datacenter cost; meet availability target

• **Observation:** Data-intensive applications’ data exhibit a diverse spectrum of tolerance to memory errors
  - Across applications and within an application
  - We characterized 3 modern data-intensive applications

• **Our Proposal:** Heterogeneous-reliability memory (HRM)
  - Store error-tolerant data in less-reliable lower-cost memory
  - Store error-vulnerable data in more-reliable memory

• **Major results:**
  - Reduce server hardware cost by 4.7 %
  - Achieve single server availability target of 99.90 %
Motivation

Characterizing application memory error tolerance

Key observations
- Observation 1: Memory error tolerance varies across applications and within an application
- Observation 2: Data can be recovered by software

Heterogeneous-Reliability Memory (HRM)

Evaluation
Outline

• Motivation
• Characterizing application memory error tolerance
• Key observations
  - Observation 1: Memory error tolerance varies across applications and within an application
  - Observation 2: Data can be recovered by software
• Heterogeneous-Reliability Memory (HRM)
• Evaluation
Server Memory Cost is High

• Server hardware cost dominates datacenter Total Cost of Ownership (TCO) [Barroso ‘09]

• As server memory capacity grows, memory cost becomes the most important component of server hardware costs [Kozyrakis ‘10]

128GB Memory cost
~$140 (per 16GB) \times 8
= ~$1120 *

2 CPUs cost
~$500 (per CPU) \times 2
= ~$1000 *

* Numbers in the year of 2014
Memory Reliability is Important

System/app crash

System/app hang or slowdown

Silent data corruption or incorrect app output

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

If you'd like to know more, you can search online later for this error: HAL INITIALIZATION FAILED
Memory testing cost can be a significant fraction of memory cost as memory capacity grows.
**Existing Error Mitigation Techniques (II)**

- **Error detection and correction increases system cost**

<table>
<thead>
<tr>
<th>Technique</th>
<th>Detection</th>
<th>Correction</th>
<th>Added capacity</th>
<th>Added logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoECC</td>
<td>N/A</td>
<td>N/A</td>
<td>0.00%</td>
<td>No</td>
</tr>
<tr>
<td>Parity</td>
<td>1 bit</td>
<td>N/A</td>
<td>1.56%</td>
<td>Low</td>
</tr>
<tr>
<td>SEC-DED</td>
<td>2 bit</td>
<td>1 bit</td>
<td>12.5%</td>
<td>Low</td>
</tr>
<tr>
<td>Chipkill</td>
<td>2 chip</td>
<td>1 chip</td>
<td>12.5%</td>
<td>High</td>
</tr>
</tbody>
</table>

Stronger error protection techniques have higher cost.
Shortcomings of Existing Approaches

• *Uniformly* improve memory reliability
  - **Observation 1:** Memory error tolerance varies across applications and with an application

• *Rely only on hardware-level techniques*
  - **Observation 2:** Once a memory error is detected, most corrupted data can be recovered by software

**Goal:** Design a new cost-efficient memory system that flexibly matches memory reliability with application memory error tolerance
Outline

• Motivation
• Characterizing application memory error tolerance
• Key observations
  - Observation 1: Memory error tolerance varies across applications and within an application
  - Observation 2: Data can be recovered by software
• Heterogeneous-Reliability Memory (HRM)
• Evaluation
Characterization Goal

Quantify application memory error tolerance

Memory Error

- Store
  - Masked by Overwrite
    - Masked by Logic
      - Correct Result
    - Incorrect Response
      - Incorrect Result
  - Consumed by Application
    - System/App Crash
      - Return x;
or
      - *x;

- Load
  - Correct Result

Memory Error Outcomes

x = ...000...

if (x != 0)
...

corrupted
x = ...110...
Characterization Methodology

• **3 modern data-intensive applications**

<table>
<thead>
<tr>
<th>Application</th>
<th>WebSearch</th>
<th>Memcached</th>
<th>GraphLab</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory footprint</td>
<td>46 GB</td>
<td>35 GB</td>
<td>4 GB</td>
</tr>
</tbody>
</table>

• **3 dominant memory regions**
  - Heap – dynamically allocated data
  - Stack – function parameters and local variables
  - Private – private heap managed by user

• **Injected a total of 23,718 memory errors using software debuggers (WinDbg and GDB)**

• **Examined correctness for over 4 billion queries**
Outline

• Motivation
• Characterizing application memory error tolerance
• Key observations
  - **Observation 1**: Memory error tolerance varies across applications and within an application
  - **Observation 2**: Data can be recovered by software
• Heterogeneous-Reliability Memory (HRM)
• Evaluation
Observation 1: Memory Error Tolerance Varies Across Applications

System/Application Crash

Showing results for single-bit soft errors
Results for other memory error types can be found in the paper with similar conclusion
Observation 1: Memory Error Tolerance Varies Across Applications

Incorrect Responses

WebSearch
Memcached
GraphLab

Showing results for single-bit soft errors
Results for other memory error types can be found in the paper

>10^5× difference
Observation 1: Memory Error Tolerance Varies Across Applications and **Within an Application**

**System/Application Crash**

- **Private**: Probability of Crash (%)
- **Heap**: Probability of Crash (%)
- **Stack**: Probability of Crash (%)

Showing results for WebSearch
Results for other workloads can be found in the paper
Observation 1: Memory Error Tolerance Varies Across Applications and **Within an Application**

Incorrect Responses

- Private
- Heap
- Stack

All averaged at a very low rate

Showing results for WebSearch

Results for other workloads can be found in the paper
• Motivation

• Characterizing application memory error tolerance

• Key observations
  - Observation 1: Memory error tolerance varies across applications and within an application
  - Observation 2: Data can be recovered by software

• Heterogeneous-Reliability Memory (HRM)

• Evaluation
Observation 2: Data Can be Recovered by Software Implicitly and Explicitly

- **Implicitly recoverable** – application intrinsically has a clean copy of the data on disk
- **Explicitly recoverable** – application can create a copy of the data at a low cost (if it has very low write frequency)

### WebSearch Recoverability

<table>
<thead>
<tr>
<th></th>
<th>Implicitly recoverable</th>
<th>Explicitly recoverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private</td>
<td>88%</td>
<td>63%</td>
</tr>
<tr>
<td>Heap</td>
<td>59%</td>
<td>28%</td>
</tr>
<tr>
<td>Stack</td>
<td>1%</td>
<td>16%</td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Motivation
• Characterizing application memory error tolerance
• Key observations
  - Observation 1: Memory error tolerance varies across applications and within an application
  - Observation 2: Data can be recovered by software
• Heterogeneous-Reliability Memory (HRM)
• Evaluation
Exploiting Memory Error Tolerance

**Vulnerable data**
- ECC protected
- Well-tested chips

**Tolerant data**
- NoECC or Parity
- Less-tested chips

**Reliable memory**

**Low-cost memory**

**Heterogeneous-Reliability Memory**
Par+R: Parity Detection + Software Recovery

**Implicit Recovery**

- Memory
  - Memory Error
  - Copy
  - Intrinsics
    - Copy

**Explicit Recovery**

- Memory
  - Memory Error
  - Copy
  - Write
  - Write non-intensive
    - Write intensive
Heterogeneous-Reliability Memory

Step 1: Characterize and classify application memory error tolerance

Step 2: Map application data to the HRM system enabled by SW/HW cooperative solutions

Vulnerable

Reliable
Parity memory + software recovery (Par+R)

Tolerant

Unreliable

Reliable memory

Low-cost memory
Outline

• Motivation
• Characterizing application memory error tolerance
• Key observations
  - Observation 1: Memory error tolerance varies across applications and within an application
  - Observation 2: Data can be recovered by software
• Heterogeneous-Reliability Memory (HRM)
• Evaluation
## Evaluated Systems

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Mapping</th>
<th>Pros and Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Private (36 GB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Heap (9 GB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stack (60 MB)</td>
<td></td>
</tr>
<tr>
<td><strong>Typical Server</strong></td>
<td>ECC</td>
<td>Reliable but expensive</td>
</tr>
<tr>
<td><strong>Consumer PC</strong></td>
<td>NoECC</td>
<td>Low-cost but unreliable</td>
</tr>
<tr>
<td><strong>HRM</strong></td>
<td>Par+R</td>
<td>Parity only</td>
</tr>
<tr>
<td><strong>Less-Tested (L)</strong></td>
<td>NoECC</td>
<td>Least expensive and reliable</td>
</tr>
<tr>
<td><strong>HRM/L</strong></td>
<td>ECC</td>
<td>Low-cost and reliable HRM</td>
</tr>
</tbody>
</table>

Baseline systems

HRM systems
## Design Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM/server HW cost [Kozyrakis ‘10]</td>
<td>30%</td>
</tr>
<tr>
<td>NoECC memory cost savings</td>
<td>11.1%</td>
</tr>
<tr>
<td>Parity memory cost savings</td>
<td>9.7%</td>
</tr>
<tr>
<td>Less-tested memory cost savings</td>
<td>18%±12%</td>
</tr>
<tr>
<td>Crash recovery time</td>
<td>10 mins</td>
</tr>
<tr>
<td>Par+R flush threshold</td>
<td>5 mins</td>
</tr>
<tr>
<td>Errors/server/month [Schroeder ‘09]</td>
<td>2000</td>
</tr>
<tr>
<td>Target single server availability</td>
<td>99.90%</td>
</tr>
</tbody>
</table>
Evaluation Metrics

• Cost
  - Memory cost savings
  - Server HW cost savings
    (both compared with Typical Server)

• Reliability
  - Crashes/server/month
  - Single server availability
  - # incorrect/million queries
Improving Server HW Cost Savings

Reducing the use of memory error mitigation techniques in part of memory space can save noticeable amount of server HW cost.
HRM systems are flexible to adjust and can achieve availability target.
HRM systems can achieve acceptable correctness.
Evaluation Results

Bigger area means better tradeoff
Other Results and Findings

- **Characterization of applications’ reactions to memory errors**
  - Finding: Quick-to-crash vs. periodically incorrect behavior

- **Characterization of most common types of memory errors including single-bit soft/hard errors, multi-bit hard errors**
  - Finding: More severe errors mainly decrease correctness

- **Characterization of how errors are masked**
  - Finding: Some memory regions are safer than others

- **Discussion about heterogeneous reliability design dimensions, techniques, and their benefits and tradeoffs**
Conclusion

• **Our Goal:** Reduce datacenter *cost*; meet *availability* target

• **Characterized** application-level memory error tolerance of 3 modern data-intensive workloads

• **Proposed** Heterogeneous-Reliability Memory (HRM)
  - Store error-tolerant data in less-reliable lower-cost memory
  - Store error-vulnerable data in more-reliable memory

• **Evaluated** example HRM systems
  - Reduce server hardware *cost* by 4.7 %
  - Achieve single-server *availability* target 99.90 %
Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, and Onur Mutlu,
"Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost via Heterogeneous-Reliability Memory"
Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Atlanta, GA, June 2014. Slides (pptx) (pdf) Coverage on ZDNet
Some New Ideas (This Year)

- **Specialization**
  - Heterogeneous Reliability Memory [DSN 2014]
  - Heterogeneous Block Architecture [ICCD 2014]

- **Persistent Memory**
  - Loose Ordering Consistency for Persistent Memory [ICCD 2014]
  - Transparent Consistency for Persistent/Hybrid Memory [in progress]

- **Memory Reliability/Security**
  - Row Hammer Problem in DRAM [ISCA 2014]
  - Neighbor-Cell Assisted Error Correction in Flash [SIGMETRICS 2014]
  - Error Mitigation for Intermittent DRAM Failures [SIGMETRICS 2014]

- **Memory Performance**
  - The Dirty-Block Index [ISCA 2014]
  - DRAM Refresh-Access Parallelization [HPCA 2014]
  - The Blacklisting Memory Scheduler [ICCD 2014]
  - Exploiting Read-Write Disparity in Caches [HPCA 2014]
Some New Ideas in Memory System Design for Data-Intensive Computing

Onur Mutlu
onur@cmu.edu
September 4, 2014
ISTC-CC Retreat

Carnegie Mellon
Backup Slides
The Dirty-Block Index
The Dirty-Block Index
ISCA 2014

Vivek Seshadri
Abhishek Bhowmick • Onur Mutlu
Phillip B. Gibbons • Michael A. Kozuch • Todd C. Mowry

SAFARI  
Carnegie Mellon

intel®
Mismatch: Representation and Query

Sorted by Title

Get all the books written by author X
Mismatch: Representation and Query

Breadth First Search
List all edges adjacent to vertex ‘a’
## Mismatch: Representation and Query

List all dirty blocks of DRAM row R.

Is block X dirty?

<table>
<thead>
<tr>
<th>Dirty Bit</th>
<th>Tag</th>
<th>Dirty Bit</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td></td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>
## Dirty-Block Index

### Cache Tag Store

<table>
<thead>
<tr>
<th>Tag</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Tag</td>
</tr>
<tr>
<td>Tag</td>
<td>Tag</td>
</tr>
</tbody>
</table>

### DBI

- **List all dirty blocks of DRAM row R.**
- **Is block X dirty?**
Application: DRAM-Aware Writeback

Virtual Write Queue [ISCA 2010], DRAM-Aware Writeback [TR-HPS-2010-2]

1. Buffer writes and flush them in a burst

2. Row buffer hits are faster and more efficient than row misses
Application: DRAM-Aware Writeback

Virtual Write Queue [ISCA 2010], DRAM-Aware Writeback [TR-HPS-2010-2]

Dirty Block

Proactively write back all other dirty blocks from the same DRAM row

Significantly increases the DRAM write row hit rate

Get all dirty blocks of DRAM row ‘R’
Shortcoming of Block-Oriented Organization

Get all dirty blocks of DRAM row ‘R’

Set of blocks co-located in DRAM
~8KB = 128 cache blocks

Is block 1 of Row R dirty?
Is block 2 of Row R dirty?
Is block 3 of Row R dirty?
...
Is block 128 of Row R dirty?
The Dirty-Block Index (DBI)
The Dirty-Block Index

1 DRAM-Aware Writeback w/ DBI

Virtual Write Queue [ISCA 2010], DRAM-Aware Writeback [TR-HPS-2010-2]

Dirty Block

Proactively write back dirty blocks from the same DRAM row

DBI achieves the benefit of DRAM-aware writeback without increasing contention for the tag store!

Look up the cache only for these blocks
Many Optimizations

1. DRAM-aware writeback
2. Bypassing cache lookups
3. Reducing ECC overhead
4. Efficient cache flushing
5. Load balancing memory accesses
6. Bulk DMA
7. Efficient write scheduling

...
Many Optimizations

1. DRAM-aware writeback
2. Bypassing cache lookups
3. Reducing ECC overhead

31% performance over baseline
6% over best previous mechanism
8% cache area reduction
More Information …

- Vivek Seshadri, Abhishek Bhowmick, Onur Mutlu, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry,
  "The Dirty-Block Index"
  Lightning Session Slides (pptx) (pdf)
Refresh-Access Parallelization
Refresh Penalty

Refresh interferes with memory accesses

Refresh delays requests by 100s of ns
Existing Refresh Modes

All-bank refresh in commodity DRAM (DDRx)

*Per-bank refresh allows accesses to other banks while a bank is refreshing*
Shortcomings of Per-Bank Refresh

• **Problem 1**: Refreshes to different banks are scheduled in a strict round-robin order
  – The static ordering is hardwired into DRAM chips
  – Refreshes busy banks with many queued requests when other banks are idle

• **Key idea**: Schedule per-bank refreshes to idle banks opportunistically in a dynamic order
Our First Approach: DARP

• Dynamic Access-Refresh Parallelization (DARP)
  – An improved scheduling policy for per-bank refreshes
  – Exploits refresh scheduling flexibility in DDR DRAM

• Component 1: Out-of-order per-bank refresh
  – Avoids poor static scheduling decisions
  – Dynamically issues per-bank refreshes to idle banks

• Component 2: Write-Refresh Parallelization
  – Avoids refresh interference on latency-critical reads
  – Parallelizes refreshes with a batch of writes
Shortcomings of Per-Bank Refresh

- Problem 2: Banks that are being refreshed cannot concurrently serve memory requests
Shortcomings of Per-Bank Refresh

• **Problem 2:** Refreshing banks cannot concurrently serve memory requests

• **Key idea:** Exploit **subarrays** within a bank to parallelize refreshes and accesses across **subarrays**

![Diagram showing subarrays and refreshes](image)
Methodology

100 workloads: SPEC CPU2006, STREAM, TPC-C/H, random access

System performance metric: Weighted speedup
Comparison Points

• **All-bank refresh** [DDR3, LPDDR3, ...]

• **Per-bank refresh** [LPDDR3]

• **Elastic refresh** [Stuecheli et al., MICRO ‘10]:
  – Postpones refreshes by a time delay based on the predicted rank idle time to avoid interference on memory requests
  – Proposed to schedule all-bank refreshes without exploiting per-bank refreshes
  – Cannot parallelize refreshes and accesses within a rank

• **Ideal (no refresh)**
2. Consistent system performance improvement across DRAM densities (within 0.9%, 1.2%, and 3.8% of ideal)
Energy Efficiency

Consistent reduction on energy consumption