Improving DRAM Performance by Parallelizing Refreshes with Accesses

Kevin Chang

Donghyuk Lee, Zeshan Chishti, Alaa Alameldeen, Chris Wilkerson, Yoongu Kim, Onur Mutlu
Executive Summary

• DRAM refresh interferes with memory accesses
  – Degrades system performance and energy efficiency
  – Becomes exacerbated as DRAM density increases

• Goal: Serve memory accesses in parallel with refreshes to reduce refresh interference on demand requests

• Our mechanisms:
  – 1. Enable more parallelization between refreshes and accesses across different banks with new per-bank refresh scheduling algorithms
  – 2. Enable serving accesses concurrently with refreshes in the same bank by exploiting DRAM subarrays

• Improve system performance and energy efficiency for a wide variety of different workloads and DRAM densities
  – 20.2% and 9.0% for 8-core systems using 32Gb DRAM
  – Very close to the ideal scheme without refreshes
Outline

• Motivation and Key Ideas
• DRAM and Refresh Background
• Our Mechanisms
• Results
Refresh Penalty

Refresh interferes with memory accesses

Refresh delays requests by 100s of ns
Existing Refresh Modes

All-bank refresh in commodity DRAM (DDRx)

Bank 7

Bank 1

Bank 0

Refresh

Time

Bank 7

Bank 1

Bank 0

Time

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
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
Outline

• Motivation and Key Ideas
• DRAM and Refresh Background
• Our Mechanisms
• Results
• Banks can serve multiple requests in parallel
DRAM Refresh Frequency

- DRAM standard requires memory controllers to send periodic refreshes to DRAM

\[ t_{\text{RefLatency}} (t_{\text{RFC}}) \]: Varies based on DRAM chip density (e.g., 350ns)

\[ t_{\text{RefPeriod}} (t_{\text{REFI}}) \]: Remains constant

Read/Write: roughly 50ns
Increasing Performance Impact

• DRAM is unavailable to serve requests for \( \frac{t_{RefLatency}}{t_{RefPeriod}} \) of time

• 6.7% for today’s 4Gb DRAM

• Unavailability increases with higher density due to higher \( t_{RefLatency} \)
  – 23% / 41% for future 32Gb / 64Gb DRAM
All-Bank vs. Per-Bank Refresh

All-Bank Refresh: Employed in commodity DRAM (DDRx, LPDDRx)

- Bank 0
  - Read
  - Refresh
  - Bank 1
  - Read
  - Refresh
  - Timeline

Per-Bank Refresh: In mobile DRAM (LPDDRx)

- Bank 0
  - Refresh
  - Read
  - Bank 1
  - Read
  - Refresh
  - Timeline

Staggered across banks to limit power

Can serve memory accesses in parallel with refreshes across banks
Shortcomings of Per-Bank Refresh

• 1) Per-bank refreshes are strictly scheduled in round-robin order (as fixed by DRAM’s internal logic)

• 2) A refreshing bank cannot serve memory accesses

Goal: Enable more parallelization between refreshes and accesses using practical mechanisms
Outline

• Motivation and Key Ideas
• DRAM and Refresh Background
• Our Mechanisms
  – 1. Dynamic Access-Refresh Parallelization (DARP)
  – 2. Subarray Access-Refresh Parallelization (SARP)
• Results
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
1) Out-of-Order Per-Bank Refresh

- **Dynamic scheduling policy** that prioritizes refreshes to idle banks
- **Memory controllers** decide which bank to refresh
1) Out-of-Order Per-Bank Refresh

Baseline: Round robin

- Request queue (Bank 0)
- Request queue (Bank 1)

Timeline:

Bank 1

- Refesh
- Read

Bank 0

- Refresh
- Read

Saved cycles

Reduces refresh penalty on demand requests by refreshing idle banks first in a flexible order
Outline

• Motivation and Key Ideas
• DRAM and Refresh Background
• Our Mechanisms
  – 1. Dynamic Access-Refresh Parallelization (DARP)
    • 1) Out-of-Order Per-Bank Refresh
    • 2) Write-Refresh Parallelization
  – 2. Subarray Access-Refresh Parallelization (SARP)
• Results
Refresh Interference on Upcoming Requests

- **Problem**: A refresh may collide with an upcoming request in the near future

![Diagram showing refresh interference]

- Bank 1: Read
- Bank 0: Refresh
- Read delayed by refresh
DRAM Write Draining

- **Observations:**

  1) **Bus-turnaround latency** when transitioning from writes to reads or vice versa
     - To mitigate **bus-turnaround latency**, writes are typically drained to DRAM in a batch during a period of time
  
  2) **Writes are not latency-critical**
2) Write-Refresh Parallelization

- Proactively schedules refreshes when banks are serving write batches

Baseline

Timeline

Bank 1

Read

Bank 0

Refresh

Read

Turnaround

Bank 1

Read

Write

Write

Write

Timeline

Bank 0

Refresh

Read

Saved cycles

Avoids stalling latency-critical read requests by refreshing with non-latency-critical writes

1. Postpone refresh

2. Refresh during writes
Outline

• Motivation and Key Ideas
• DRAM and Refresh Background
• Our Mechanisms
  – 1. Dynamic Access-Refresh Parallelization (DARP)
  – 2. Subarray Access-Refresh Parallelization (SARP)
• Results
Our Second Approach: SARP

Observations:
1. A bank is further divided into **subarrays**
   - Each has its own **row buffer** to perform refresh operations

2. Some **subarrays** and bank I/O remain completely **idle** during refresh
Our Second Approach: SARP

- **Subarray Access-Refresh Parallelization (SARP):**
  - Parallelizes refreshes and accesses **within a bank**
Our Second Approach: SARP

- **Subarray Access-Refresh Parallelization (SARP):**
  - Parallelizes refreshes and accesses within a bank

**Bank 7**

**Bank 1**

**Bank 0**

**Timeline**

**Refresh**

**Read**

**Very modest DRAM modifications: 0.71% die area overhead**
Outline

• Motivation and Key Ideas
• DRAM and Refresh Background
• Our Mechanisms
• Results
Methodology

Simulator configurations

- 8-core processor
- Memory Controller
- DDR3 Rank
- Bank 7
- Bank 1
- Bank 0

L1 $: 32KB
L2 $: 512KB/core

- **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)
Consistent reduction on energy consumption
Other Results and Discussion in the Paper

- Detailed multi-core results and analysis
- Result breakdown based on memory intensity
- Sensitivity results on number of cores, subarray counts, refresh interval length, and DRAM parameters
- Comparisons to DDR4 fine granularity refresh
Executive Summary

• DRAM refresh interferes with memory accesses
  – Degrades system performance and energy efficiency
  – Becomes exacerbated as DRAM density increases

• Goal: Serve memory accesses in parallel with refreshes to reduce refresh interference on demand requests

• Our mechanisms:
  – 1. Enable more parallelization between refreshes and accesses across different banks with new per-bank refresh scheduling algorithms
  – 2. Enable serving accesses concurrently with refreshes in the same bank by exploiting DRAM subarrays

• Improve system performance and energy efficiency for a wide variety of different workloads and DRAM densities
  – 20.2% and 9.0% for 8-core systems using 32Gb DRAM
  – Very close to the ideal scheme without refreshes
Improving DRAM Performance by Parallelizing Refreshes with Accesses

Kevin Chang

Donghyuk Lee, Zeshan Chishti, Alaa Alameldeen, Chris Wilkerson, Yoongu Kim, Onur Mutlu

SAFARI  Carnegie Mellon  Intel
Backup
Comparison to Concurrent Work

• Zhang et al., HPCA’14

• Ideas:
  – 1) Sub-rank refresh → refreshes a subset of banks within a rank
  – 2) Subarray refresh → refreshes one subarray at a time
  – 3) Dynamic sub-rank refresh scheduling policies

• Similarities:
  – 1) Leverage idle subarrays to serve accesses
  – 2) Schedule refreshes to idle banks first

• Differences:
  – 1) Exploit write draining periods to hide refresh latency
  – 2) We provide detailed analysis on existing per-bank refresh in mobile DRAM
  – 3) Concrete description on our scheduling algorithm
Performance Impact of Refreshes

- Refresh penalty exacerbates as density grows

![Graph showing technology feature trend with unavailability (%) on the y-axis and gigabits (Gb) per DRAM chip on the x-axis. The graph includes current and future data points and potential range. The graph indicates that unavailability increases as density grows.]

*ITRS Roadmap, 2011
Temporal Flexibility

- DRAM standard allows a few refresh commands to be issued early or late

**Timeline**

1. Delayed by 1 refresh command
2. Ahead by 1 refresh command
Refresh

- $t_{Retention} = 32\, ms$
- $t_{Refresh\, Period} = 3.9\, \mu s$
- Fixed number of refresh commands to refresh entire DRAM: $N = 8192$

$t_{Refresh\, Window} = N \times t_{Refresh\, Period} = 31.948\, ms < t_{Retention}$

$t_{Retention} > t_{Refresh\, Window} + t_{Delay}$
Unfairness \( (\text{Maximum Slowdown} = \max_i \frac{IPC_i^{\text{alone}}}{IPC_i^{\text{shared}}}) \)

Our mechanisms do not unfairly slow down specific applications to gain performance.
Power Overhead

Power overhead to parallelize a refresh operation and accesses over a four-activate window:

\[
PowerOverhead_{t_{FAW}} = \frac{(4 \times I_{ACT} + I_{REF})}{4 \times I_{ACT}}
\]

Extend both \( t_{FAW} \) and \( t_{RRD} \) timing parameters:

\[
t_{FAW_{SARP}} = t_{FAW} \times PowerOverhead_{t_{FAW}}
\]

\[
t_{RRD_{SARP}} = t_{RRD} \times PowerOverhead_{t_{FAW}}
\]
Refresh Interval (7.8μs)

GeoMean Weighted Speedup

<table>
<thead>
<tr>
<th>DRAM Chip Density</th>
<th>REFab</th>
<th>REFpb</th>
<th>(D+S)ARP</th>
<th>Ideal</th>
</tr>
</thead>
<tbody>
<tr>
<td>8Gb</td>
<td>3.3%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16Gb</td>
<td>5.3%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32Gb</td>
<td>9.1%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Die Area Overhead

- Rambus DRAM model with 55nm

- **SARP area overhead**: 0.71% in a 2Gb DRAM chip
System Performance

![Graph showing GeoMean Weighted Speedup vs DRAM Chip Density for different DRAM sizes (8Gb, 16Gb, 32Gb). The graph compares various techniques: REFab, Elastic, REFpb, DARP, SARP, (D+S)ARP, and Ideal.]
Effect of Memory Intensity

Compared to REFab

Compared to REFpb

WS Improvement (%)
DDR4 FGR

Normalized WS

0 0.2 0.4 0.6 0.8 1 1.2

8Gb 16Gb 32Gb

DRAM Density

REFab
FGR 2x
FGR 4x
AR
DSARP
Performance Breakdown

- Out-of-order refresh improves performance by 3.2%/3.9%/3.0% over 8/16/32Gb DRAM

- Write-refresh parallelization provides additional benefits of 4.3%/5.8%/5.2%
## tFAW Sweep

<table>
<thead>
<tr>
<th>tFAW/tRRD</th>
<th>5/1</th>
<th>10/2</th>
<th>15/3</th>
<th>20/4</th>
<th>25/5</th>
<th>30/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS Gain (%)</td>
<td>14.0</td>
<td>13.9</td>
<td>13.5</td>
<td>12.4</td>
<td>11.9</td>
<td>10.3</td>
</tr>
</tbody>
</table>

Baseline
Pathological latency = 3.5 * tRefLatency_AllBank
Our Second Approach: SARP

- **Subarray Access-Refresh Parallelization (SARP):**
  - Parallelizes refreshes and accesses within a bank
- **Problem:** Shared address path for refreshes and accesses
- **Solution:** Decouple the shared address path
Our Second Approach: SARP

- **Subarray Access-Refresh Parallelization (SARP):**
  - Parallelizes refreshes and accesses within a bank

- **Problem:** Shared address path for refreshes and accesses

- **Solution:** Decouple the shared address path