ADVANCED MEMORY SYSTEMS

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

- Announcement
  - Homework 5 will be released tonight (the last one 😊)

- This lecture
  - Memory addressing/scheduling
  - DRAM refresh
  - Emerging technologies
Recall: DRAM Control Tasks

- Refresh management
  - Periodically replenish the DRAM cells (burst vs. distributed)

- Address mapping
  - Distribute the requests to destination banks (load balancing)

- Request scheduling
  - Generate a sequence of commands for memory requests
    - Reduce overheads by eliminating unnecessary commands

- Power management
  - Keep the power consumption under a cap

- Error detection/correction
  - Detect and recover corrupted data
Address Mapping

- A memory request

<table>
<thead>
<tr>
<th>Type</th>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
</table>

- Address is used to find the location in memory
  - Channel, rank, bank, row, and column IDs

- Example physical address format

<table>
<thead>
<tr>
<th>Row ID</th>
<th>Channel ID</th>
<th>Rank ID</th>
<th>Bank ID</th>
<th>Column ID</th>
</tr>
</thead>
</table>

- A 4GB channel, 2 ranks, 4 banks/rank, 8KB page
Address Mapping

- A memory request

- Address is used to find the location in memory
  - Channel, rank, bank, row, and column IDs

- Example physical address format

<table>
<thead>
<tr>
<th>Type</th>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row ID</th>
<th>Channel ID</th>
<th>Rank ID</th>
<th>Bank ID</th>
<th>Column ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>13</td>
</tr>
</tbody>
</table>

- A 4GB channel, 2 ranks, 4 banks/rank, 8KB page
Example Problem

- Start with empty row buffers, find the total number of commands if all the request are served in order
  - Address = row(12):channel(0):rank(1):bank(3):column(16)

<table>
<thead>
<tr>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
</tr>
<tr>
<td>20000001</td>
</tr>
<tr>
<td>40000100</td>
</tr>
<tr>
<td>60000010</td>
</tr>
<tr>
<td>40000101</td>
</tr>
</tbody>
</table>
Example Problem

- Start with empty row buffers, find the total number of commands if all the request are served in order
  - Address= row(12):channel(0):rank(1):bank(3):column(16)

<table>
<thead>
<tr>
<th>addr</th>
<th>rank</th>
<th>bank</th>
<th>row</th>
<th>column</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
<td>0</td>
<td>0</td>
<td>000</td>
<td>0010</td>
</tr>
<tr>
<td>20000001</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40000100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60000100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40000101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example Problem

- Start with empty row buffers, find the total number of commands if all the request are served in order
  - Address= row(12):channel(0):rank(1):bank(3):column(16)

<table>
<thead>
<tr>
<th>addr</th>
<th>rank</th>
<th>bank</th>
<th>row</th>
<th>column</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
<td>0</td>
<td>0</td>
<td>000</td>
<td>0010</td>
</tr>
<tr>
<td>20000001</td>
<td>0</td>
<td>0</td>
<td>200</td>
<td>0001</td>
</tr>
<tr>
<td>40000100</td>
<td>0</td>
<td>0</td>
<td>400</td>
<td>0100</td>
</tr>
<tr>
<td>60000010</td>
<td>0</td>
<td>0</td>
<td>600</td>
<td>0010</td>
</tr>
<tr>
<td>40000101</td>
<td>0</td>
<td>0</td>
<td>400</td>
<td>0101</td>
</tr>
</tbody>
</table>
Example Problem

- Start with empty row buffers, find the total number of commands if all the request are served in order

  - Address = row(12):channel(0):rank(1):bank(3):column(16)

<table>
<thead>
<tr>
<th>addr</th>
<th>rank</th>
<th>bank</th>
<th>row</th>
<th>column</th>
<th>commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
<td>0</td>
<td>0</td>
<td>000</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>20000001</td>
<td>0</td>
<td>0</td>
<td>200</td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>40000100</td>
<td>0</td>
<td>0</td>
<td>400</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>60000010</td>
<td>0</td>
<td>0</td>
<td>600</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>40000101</td>
<td>0</td>
<td>0</td>
<td>400</td>
<td>0101</td>
<td></td>
</tr>
</tbody>
</table>
Example Problem

- Start with empty row buffers, find the total number of commands if all the request are served in order

  Address = row(12):channel(0):rank(1):bank(3):column(16)

<table>
<thead>
<tr>
<th>addr</th>
<th>rank</th>
<th>bank</th>
<th>row</th>
<th>column</th>
<th>commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
<td>0</td>
<td>0</td>
<td>000</td>
<td>0010</td>
<td>ACT</td>
</tr>
<tr>
<td>20000001</td>
<td>0</td>
<td>0</td>
<td>200</td>
<td>0001</td>
<td>PRE</td>
</tr>
<tr>
<td>40000100</td>
<td>0</td>
<td>0</td>
<td>400</td>
<td>0100</td>
<td>PRE</td>
</tr>
<tr>
<td>60000010</td>
<td>0</td>
<td>0</td>
<td>600</td>
<td>0010</td>
<td>PRE</td>
</tr>
<tr>
<td>40000101</td>
<td>0</td>
<td>0</td>
<td>400</td>
<td>0101</td>
<td>PRE</td>
</tr>
</tbody>
</table>
Example Problem

- Find the total number of commands using the following address mapping scheme

  - Address = bank(3):rank(1):channel(0):row(12):column(16)

<table>
<thead>
<tr>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
</tr>
<tr>
<td>20000001</td>
</tr>
<tr>
<td>40000100</td>
</tr>
<tr>
<td>60000010</td>
</tr>
<tr>
<td>40000101</td>
</tr>
</tbody>
</table>
Example Problem

- Find the total number of commands using the following address mapping scheme

  Address = bank(3):rank(1):channel(0):row(12):column(16)

<table>
<thead>
<tr>
<th>addr</th>
<th>rank</th>
<th>bank</th>
<th>row</th>
<th>column</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
<td>0</td>
<td>0</td>
<td>000</td>
<td>0010</td>
</tr>
<tr>
<td>20000001</td>
<td>0</td>
<td>1</td>
<td>000</td>
<td>0001</td>
</tr>
<tr>
<td>40000010</td>
<td>0</td>
<td>2</td>
<td>000</td>
<td>0100</td>
</tr>
<tr>
<td>60000010</td>
<td>0</td>
<td>3</td>
<td>000</td>
<td>0010</td>
</tr>
<tr>
<td>40000101</td>
<td>0</td>
<td>2</td>
<td>000</td>
<td>0101</td>
</tr>
</tbody>
</table>
Example Problem

- Find the total number of commands using the following address mapping scheme

  Address = bank(3):rank(1):channel(0):row(12):column(16)

<table>
<thead>
<tr>
<th>addr</th>
<th>rank</th>
<th>bank</th>
<th>row</th>
<th>column</th>
<th>commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
<td>0</td>
<td>0</td>
<td>000</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>20000001</td>
<td>0</td>
<td>1</td>
<td>000</td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>40000100</td>
<td>0</td>
<td>2</td>
<td>000</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>60000010</td>
<td>0</td>
<td>3</td>
<td>000</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>40000101</td>
<td>0</td>
<td>2</td>
<td>000</td>
<td>0101</td>
<td></td>
</tr>
</tbody>
</table>
Example Problem

Find the total number of commands using the following address mapping scheme:

- Address = bank(3):rank(1):channel(0):row(12):column(16)

<table>
<thead>
<tr>
<th>addr</th>
<th>rank</th>
<th>bank</th>
<th>row</th>
<th>column</th>
<th>commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
<td>0</td>
<td>0</td>
<td>000</td>
<td>0010</td>
<td>ACT</td>
</tr>
<tr>
<td>20000001</td>
<td>0</td>
<td>1</td>
<td>000</td>
<td>0001</td>
<td>RD</td>
</tr>
<tr>
<td>40000100</td>
<td>0</td>
<td>2</td>
<td>000</td>
<td>0100</td>
<td>ACT</td>
</tr>
<tr>
<td>60000010</td>
<td>0</td>
<td>3</td>
<td>000</td>
<td>0010</td>
<td>RD</td>
</tr>
<tr>
<td>40000101</td>
<td>0</td>
<td>2</td>
<td>000</td>
<td>0101</td>
<td>RD</td>
</tr>
</tbody>
</table>
Command Scheduling

- **Write buffering**
  - Writes can wait until reads are done

- **Controller queues DRAM commands**
  - Usually into per-bank queues
  - Allows easily reordering ops. meant for same bank

- **Common policies**
  - First-Come-First-Served (FCFS)
  - First-Ready First-Come-First-Served (FR-FCFS)
Command Scheduling

- First-Come-First-Served
  - Oldest request first
- First-Ready First-Come-First-Served
  - Prioritize column changes over row changes
  - Skip over older conflicting requests
  - Find row hits (on queued requests)
    - Find oldest
    - If no conflicts with in-progress request \( \rightarrow \) good
    - Otherwise (if conflicts), try next oldest
FCFS vs. FR-FCFS

- $\text{READ(B0,R0,C0)} \ \text{READ(B0,R1,C0)} \ \text{READ(B0,R0,C1)}$
- FCFS
FCFS vs. FR-FCFS

- READ(B0,R0,C0) READ(B0,R1,C0) READ(B0,R0,C1)

- FCFS

Cmd: ACT READ PRE ACT READ PRE ACT READ

Addr: R0 C0 B0 R1 C0 B1 R0 C1
FCFS vs. FR-FCFS

- READ(B0,R0,C0) READ(B0,R1,C0) READ(B0,R0,C1)
  - FCFS

  ![Diagram of FCFS](image)

- FR-FCFS

  ![Diagram of FR-FCFS](image)
**FCFS vs. FR-FCFS**

- **READ(B0,R0,C0) READ(B0,R1,C0) READ(B0,R0,C1)**
  - **FCFS**

  ![FCFS diagram]

  - **FR-FCFS**

  ![FR-FCFS diagram]

  **Savings**
Row Buffer Management Policies

- **Open-page policy**
  - After access, keep page in DRAM row buffer
  - If access to different page, must close old one first
    - Good if lots of locality

- **Close-page policy**
  - After access, immediately close page in DRAM row buffer
  - If access to different page, old one already closed
    - Good if no locality (random access)
DRAM Refresh Management

- DRAM requires the cells’ contents to be read and written periodically.
DRAM Refresh Management

- DRAM requires the cells’ contents to be read and written periodically
  - **Burst refresh**: refresh all of the cells each time
    - Simple control mechanism
DRAM Refresh Management

- DRAM requires the cells’ contents to be read and written periodically
  - **Burst refresh**: refresh all of the cells each time
    - Simple control mechanism
  - **Distributed refresh**: a group of cells are refreshed
    - Avoid blocking memory for a long time
DRAM Refresh Management

- DRAM requires the cells’ contents to be read and written periodically
  - **Burst refresh**: refresh all of the cells each time
    - Simple control mechanism
  - **Distributed refresh**: a group of cells are refreshed
    - Avoid blocking memory for a long time
- Recently accessed rows need not to be refreshed
  - **Smart refresh**
Error Detection/Correction

- Data in memory may be corrupted
  - Many reasons: leakage, alpha particles, hard errors

- Can errors be detected?
  - Error detection codes: additional parity bits

- Can errors be corrected?
  - Error correction codes: ECC bits are added to data

- Single-Error Correction, Double-Error Detection
  - Commonly used in memory systems
An additional DRAM chip is used for storing SECDED ECC bits for error correction.

Hamming Code (72,64)
Emerging Technologies
DRAM Cell Structure

- One-transistor, one-capacitor
  - Realizing the capacitor is challenging

- 1T-1C DRAM
- Charge based sensing
- Volatile
One-transistor, one-capacitor

- Realizing the capacitor is challenging

- 1T-1C DRAM
- Charge based sensing
- Volatile
Memory Scaling in Jeopardy

Scaling of semiconductor memories greatly challenged beyond 20nm

Example: DRAM
Memory Scaling in Jeopardy

Scaling of semiconductor memories greatly challenged beyond 20nm

Example: DRAM

![Diagram showing aspect ratio (A/R) of storage nodes and trend over technology nodes. The aspect ratio is significantly rising, indicating challenges beyond 20nm tech nodes. The source is cited as S.J. Hong (Hynix), IEDM 2010. The diagram also indicates that A/R should be less than 10 for feasible scaling.]
Why DRAM Slow?

- **Logic VLSI Process**: optimized for better transistor performance
- **DRAM VLSI Process**: optimized for low cost and low leakage

How to reduce distance?
3D Die-Stacking

- Different devices are stacked on top of each other
- Layers are connected by through-silicon vias (TSVs)

Why?
- Communication between devices bottlenecked by limited I/O pins
- Integrating heterogeneous elements on a single wafer is expensive and suboptimal
3D Stacked Memory

- **Hybrid Memory Cube (HMC)**
  - A logic layer at the bottom

- **High Bandwidth Memory (HBM)**
  - Silicon interposer at the bottom
Emerging Non Volatile Memory
Resistive Memory Technologies

- **Key concept:** replace DRAM cell capacitor with a programmable resistor

- 1T-1C DRAM
  - Charge based sensing
  - Volatile

- 1T-1R STT-MRAM, PCM, RRAM
  - Resistance based sensing
  - Non-volatile
Leading Contenders

**STT-MRAM**
- Limited to single-level cell
- 3D un-stackable
+ High endurance ($\sim 10^{15}$)
+ $\sim 4\text{ns}$ switching time
+ $\sim 50\mu\text{W}$ switching power

[Halupka, et al. ISSCC’10]

**PCM-RAM**
+ Multi-level cell capable
+ $4F^2$ 3D-stackable cell
- Endurance: $\sim 10^9$ writes
- $\sim 100\text{ns}$ switching time
- $\sim 300\mu\text{W}$ switching power

[Pronin. EETime’13]

**R-RAM**
+ Multi-level cell capable
+ $4F^2$ 3D-stackable cell
- Endurance: $10^6 \sim 10^{12}$ writes
+ $\sim 5\text{ns}$ switching time
+ $\sim 50\mu\text{W}$ switching power

[Henderson. InfoTracks’11]

[ITRS’13]
Positioning of Resistive Memories

- **RRAM**
- **PCM**
- **STT**
- **SRAM**
- **DRAM**
- **FLASH**
- **HDD**

- **Higher Speed**
- **Lower Cost**
- **Higher Endurance**

**Capacity**