ADVANCED MEMORY CONTROLLERS

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview: 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>
<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>
<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>
<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>
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)
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

- Two common policies
  - First-Come-First-Served (FCFS)
    - In order request scheduling
  - First-Ready First-Come-First-Served (FR-FCFS)
    - Out of order request scheduling
Command Scheduling

- First-Come-First-Served
  - Oldest request first

- First-Ready First-Come-First-Served
  - Prioritize column accesses 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

- Single bank memory
  - READ(B0,R0,C0) READ(B0,R1,C0) READ(B0,R0,C1)

- FCFS

<table>
<thead>
<tr>
<th>Cmd</th>
<th>ACT</th>
<th>READ</th>
<th>PRE</th>
<th>ACT</th>
<th>READ</th>
<th>PRE</th>
<th>ACT</th>
<th>READ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr</td>
<td>R0</td>
<td>C0</td>
<td>B0</td>
<td>R1</td>
<td>C0</td>
<td>B1</td>
<td>R0</td>
<td>C1</td>
</tr>
</tbody>
</table>

- FR-FCFS

<table>
<thead>
<tr>
<th>Cmd</th>
<th>ACT</th>
<th>READ</th>
<th>READ</th>
<th>PRE</th>
<th>ACT</th>
<th>READ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr</td>
<td>R0</td>
<td>C0</td>
<td>C1</td>
<td>B0</td>
<td>R1</td>
<td>C0</td>
</tr>
</tbody>
</table>

Savings
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.
Limitations to Existing Memory Controllers

- Modern memory controllers are performance-critical and complex

- Multiple performance objectives
- Application-specific optimizations
- Patches and in-field updates

Diagram:
- Core 1, Core 2, Core 3, Core 4
- Shared Cache
- Memory Controller
- Bank 1, Bank 2, Bank 3, Bank 4
- Address Mapping
- Power Management
- Command Scheduling
- QoS Maintenance
- Refresh Management
Programmable Memory Controllers

Programmability can make a memory controller higher-performance and more flexible.

- Multiple performance objectives
- Application-specific optimizations
- Patches and in-field updates

![Diagram showing a memory controller with cores, banks, and shared cache]

Programmable Framework
Key idea: Judicious division of labor between specialized hardware and firmware

- Request and transaction processing in firmware
- Configurable timing validation in hardware
Request Processing

- A RISC ISA for operating on memory requests

<table>
<thead>
<tr>
<th>Memory Request</th>
<th>Metadata</th>
<th>Address</th>
</tr>
</thead>
</table>

- Processor
- Memory
- Application Hints
- Control Flow
- Address Mapping

- ALU
Queue management with instruction flags
- R flag enqueues a request
- T flag dequeues a transaction

An instruction can be annotated with both R and T flags if needed
Implementation

- Two five-stage pipelines and one configurable timing validation circuit