18-447: Computer Architecture
Lecture 26: Memory Controllers and Memory Scheduling

Prof. Onur Mutlu
Carnegie Mellon University
Spring 2013, 4/8/2013
Homework 6

- Due April 19 (Friday)
- Topics: Virtual memory and cache interaction, main memory, memory scheduling

- Strong suggestion:
  - Please complete this before the exam to prepare for the exam

- Reminder:
  - Homeworks are mainly for your benefit and learning (and preparation for the exam).
  - They are not meant to be a large part of your grade
Homework 4 Grade Distribution

HW 4 Score Distribution

Score

Number of Students

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

0 5 10 15 20 25 30
Lab 6: Memory Hierarchy

- Due April 22 (Monday)
- Cycle-level modeling of L2 cache and DRAM-based main memory

- Extra credit: Prefetching
  - Design your own hardware prefetcher to improve system performance

- HW 6 and Lab 6 are synergistic – work on them together
Lab 4 Grade Distribution

Number of Students

Score
Lab 4 Extra Credit

7. Albert Wang (amwang): 345175250 cycles
6. Eddie Sears (esears): 224904650 cycles
5. Andrew Pfeifer (apfeifer): 104665950 cycles
4. Andrew Mort (amort): 104645150 cycle
3. Martin Gao (yiang): 104639250 cycles
2. Gun Charnmanee (gcharnma): 104634750 cycles
1. Xiao Bo Zhao (xiaoboz): 104630750 cycles
Heads Up: Midterm II Next Week

- April 17

- Similar format as Midterm I

- Suggestion: Do Homework 6 to prepare for the Midterm
Last Lecture

- Enabling multiple accesses in parallel
  - Non-blocking caches/memories
  - Multiporting (virtual and physical), multiple copies, banking

- Main memory
  - Interleaving
  - DRAM subsystem (5D nature), bottom up and top down
  - Address mapping
Today

- Memory Controllers
  - With a focus on DRAM

- Memory Access Scheduling
- Memory Interference (and Techniques to Manage It)
Guest on Wednesday

- Dr. William Strecker
- Architect of VAX
- SVP Corporate Strategy & Tech. and CTO, DEC
- CMU Alum (BS’66, MS’67, PhD’71)
- IEEE Wallace McDowell Award Recipient (1985)
  - For being principal designer of the VAX architecture and for contributions to local area networks, high-performance interconnects, caches, and memory hierarchies

Recommended Reading

Gordon Bell and William D. Strecker, “What Have We Learned from the PDP-11 - What We Have Learned from VAX and Alpha,” 25 Years of ISCA, Retrospectives and Reprints, 1998.
Course Feedback Analysis

- Course pace
  - Fast, but OK/appropriate/necessary: 4
  - Fast/complex: 2
  - Good/I like it: 2
  - OK/Not too bad: 2

- “Fast, but this pushes me to work more efficiently”
- “The material is intuitive after some explanation”
Course Feedback Analysis

- Lectures
  - Useful, informative, helpful, thorough, interesting, great: 6
  - Long or too long: 4

- Pace of lectures
  - Fast but appropriate: 3
  - Good/fine/OK: 6
  - Slow but good: 1

  - “Slides and notes are very helpful”
  - “Sometimes there is repetition”
Course Feedback Analysis

- Homeworks
  - Interesting and long: 2
  - OK/fair: 6
  - Pretty long and add more work: 2
  - “Balances with the difficulty of the labs”
  - “Too many homeworks”
  - “Questions are sometimes vague”
Course Feedback Analysis

- Labs
  - Great!: 1
  - Fun: 1
  - Good/fair: 3
  - Tedious but interesting: 2
  - Long: 1
  - Harsh grading: 1
  - Lab 1 sucked... no way of verifying the solution: 1

- “Challenging, often repetitive, but definitely help understanding”
- “Second lab seemed unstructured”, “Lab 2 seemed poorly documented – in memory interface”
- “Feels good once done”
Course Feedback Analysis

- **Satisfaction with Material**
  - Right level: 7
  - Interesting and new: 1
  - Give more microprocessor examples: 1
  - Overwhelming lectures: 1

- **Material as Expected?**
  - Yes: 5
  - Yes!: 1
  - Yes, and more: 1
  - Yes, quite interesting: 1
  - Yes and no; kind of heavy on the theory: 1
  - Less fun than expected: 1
Course Feedback Analysis

- **Workload**
  - Just right: 5 (till now, at least, says one of you...)
  - As much as expected: 1
  - Slightly heavy (but doable): 1
  - Heavy: 1 (can be 18 hours a week)
  - Too heavy: 1
  - Maybe not have HW and Lab due the same week: 1

  - “Hard to keep up as readings, etc are all due at the same time”
  - “The lab 2 bonus felt a bit like a sick joke! (with three other classes rated for 30hrs a week)”
Course Feedback Analysis

What would you change?
- Shorter lectures, longer breaks, shorter but more lectures
- Shorter homeworks
- Shorter explanations on board
- More documentation on lab starter code
- Tell us how much the avg/median time spent on each lab is
- System Verilog
- Checkpoint in the middle for long, complex labs
- What about doing labs as continued evolution of a MIPS processor in Verilog?
- Stagger due dates of assignments so we don’t frantically move between them
- Limited group discussions on concepts
Course Feedback Analysis

- Other comments
  - Materials on course website are good...
  - It would be nice if there was more feedback about how my previous designs could be improved so they do not negatively affect future designs.
  - Lectures are leading the labs by 2-3 weeks
  - Overall, I am enjoying this course and feel that I am learning a lot.
  - There is just a lot of work but I feel that this is the only way to learn and completely understand the material.
  - The TAs are awesome!
DRAM Refresh
DRAM Refresh

- DRAM capacitor charge leaks over time
- The memory controller needs to refresh each row periodically to restore charge
  - Read and close 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
DRAM Refresh: Performance

- Implications of refresh on performance
  -- DRAM bank unavailable while refreshed
  -- Long pause times: If we refresh all rows in burst, every 64ms the DRAM will be unavailable until refresh ends

- **Burst refresh**: All rows refreshed immediately after one another

- **Distributed refresh**: Each row refreshed at a different time, at regular intervals
Distributed refresh eliminates long pause times

How else can we reduce the effect of refresh on performance/QoS?

Does distributed refresh reduce refresh impact on energy?

Can we reduce the number of refreshes?
A batch of rows are periodically refreshed via the auto-refresh command.
Refresh Overhead: Performance

Refresh Overhead: Energy

Problem with Conventional Refresh

- Today: Every row is refreshed at the same rate

- Observation: Most rows can be refreshed much less often without losing data [Kim+, EDL’09]

- Problem: No support in DRAM for different refresh rates per row
Retention Time of DRAM Rows

- **Observation:** Only very few rows need to be refreshed at the worst-case rate.

- Can we exploit this to reduce refresh operations at low cost?
Reducing DRAM Refresh Operations

- **Idea:** Identify the retention time of different rows and refresh each row at the frequency it needs to be refreshed.

- **(Cost-conscious) Idea:** Bin the rows according to their minimum retention times and refresh rows in each bin at the refresh rate specified for the bin.
  - e.g., a bin for 64-128ms, another for 128-256ms, ...

- **Observation:** Only very few rows need to be refreshed very frequently [64-128ms] → Have only a few bins → Low HW overhead to achieve large reductions in refresh operations.

1. Profiling: Profile the retention time of all DRAM rows can be done at DRAM design time or dynamically.

2. Binning: Store rows into bins by retention time. Use Bloom Filters for efficient and scalable storage.

3. Refreshing: Memory controller refreshes rows in different bins at different rates. Probe Bloom Filters to determine refresh rate of a row.

RAIDR: Mechanism

- 64-128ms
- >256ms
- 128-256ms

1.25KB storage in controller for 32GB DRAM memory

Bins at different rates

→ probe Bloom Filters to determine refresh rate of a row
1. Profiling

To profile a row:

1. Write data to the row
2. Prevent it from being refreshed
3. Measure time before data corruption

<table>
<thead>
<tr>
<th>Time</th>
<th>Row 1</th>
<th>Row 2</th>
<th>Row 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially</td>
<td>111111111...</td>
<td>111111111...</td>
<td>111111111...</td>
</tr>
<tr>
<td>After 64 ms</td>
<td>111111111...</td>
<td>111111111...</td>
<td>111111111...</td>
</tr>
<tr>
<td>After 128 ms</td>
<td>110111111... (64–128ms)</td>
<td>111111111...</td>
<td>111111111...</td>
</tr>
<tr>
<td>After 256 ms</td>
<td>111110111... (128–256ms)</td>
<td>111111111...</td>
<td>(&gt;256ms)</td>
</tr>
</tbody>
</table>
2. Binning

- How to efficiently and scalably store rows into retention time bins?
- Use Hardware Bloom Filters [Bloom, CACM 1970]

Example with 64–128ms bin:

```
0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
```

Hash function 1

Hash function 2

Hash function 3

Insert Row 1

Bloom Filter

- [Bloom, CACM 1970]
- Probabilistic data structure that compactly represents set membership (presence or absence of element in a set)

- Non-approximate set membership: Use 1 bit per element to indicate absence/presence of each element from an element space of N elements

- Approximate set membership: use a much smaller number of bits and indicate each element’s presence/absence with a subset of those bits
  - Some elements map to the bits also mapped to other elements

- Operations: 1) insert, 2) test, 3) remove all elements

Bloom Filter Operation Example

Example with 64–128ms bin:

Bloom Filter Operation Example

Example with 64-128ms bin:

```
0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0
```

Hash function 1

Hash function 2

Hash function 3

Row 1 present? Yes
Bloom Filter Operation Example

Example with 64-128ms bin:

\[
\begin{array}{ccccccccccc}
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

- Hash function 1
- Hash function 2
- Hash function 3

Row 2 present? No
Bloom Filter Operation Example

Example with 64-128ms bin:

```
0 0 1 0 1 1 0 0 0 1 0 0 1 0 1 0
```

Hash function 1

Hash function 2

Hash function 3

Insert Row 4
Bloom Filter Operation Example

Example with 64-128ms bin:

```
0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 1 1 0
```

```
1 & 1 & 1 = 1
```

Hash function 1

Hash function 2

Hash function 3

Row 5 present? Yes (false positive)
Benefits of Bloom Filters as Bins

- **False positives:** a row may be declared present in the Bloom filter even if it was never inserted
  - **Not a problem:** Refresh some rows more frequently than needed

- **No false negatives:** rows are never refreshed less frequently than needed (no correctness problems)

- **Scalable:** a Bloom filter never overflows (unlike a fixed-size table)

- **Efficient:** No need to store info on a per-row basis; simple hardware → 1.25 KB for 2 filters for 32 GB DRAM system
Use of Bloom Filters in Hardware

- Useful when you can tolerate false positives in set membership tests

- See the following recent examples for clear descriptions of how Bloom Filters are used
3. Refreshing (RAIDR Refresh Controller)

Choose a refresh candidate row

Determine which bin the row is in

Determine if refreshing is needed
3. Refreshing (RAIDR Refresh Controller)

Refresh control is in DRAM in today’s auto-refresh systems

RAIDR can be implemented in either the controller or DRAM
RAIDR in Memory Controller: Option 1

Overhead of RAIDR in DRAM controller:
1.25 KB Bloom Filters, 3 counters, additional commands issued for per-row refresh (all accounted for in evaluations)
Overhead of RAIDR in DRAM chip:
Per-chip overhead: 20B Bloom Filters, 1 counter (4 Gbit chip)
Total overhead: 1.25KB Bloom Filters, 64 counters (32 GB DRAM)
RAIDR: Results and Takeaways

- System: 32GB DRAM, 8-core; SPEC, TPC-C, TPC-H workloads
- RAIDR hardware cost: 1.25 kB (2 Bloom filters)
- Refresh reduction: 74.6%
- Dynamic DRAM energy reduction: 16%
- Idle DRAM power reduction: 20%
- Performance improvement: 9%
- Benefits increase as DRAM scales in density
DRAM Refresh: More Questions

- What else can you do to reduce the impact of refresh?
- What else can you do if you know the retention times of rows?
- How can you accurately measure the retention time of DRAM rows?

Recommended reading:
Memory Controllers
DRAM versus Other Types of Memories

- Long latency memories have similar characteristics that need to be controlled.

- The following discussion will use DRAM as an example, but many issues are similar in the design of controllers for other types of memories
  - Flash memory
  - Other emerging memory technologies
    - Phase Change Memory
    - Spin-Transfer Torque Magnetic Memory
DRAM Controller: Functions

- Ensure correct operation of DRAM (refresh and timing)

- Service DRAM requests while obeying timing constraints of DRAM chips
  - Constraints: resource conflicts (bank, bus, channel), minimum write-to-read delays
  - Translate requests to DRAM command sequences

- Buffer and schedule requests to improve performance
  - Reordering, row-buffer, bank, rank, bus management

- Manage power consumption and thermals in DRAM
  - Turn on/off DRAM chips, manage power modes
DRAM Controller: Where to Place

- In chipset
  + More flexibility to plug different DRAM types into the system
  + Less power density in the CPU chip

- On CPU chip
  + Reduced latency for main memory access
  + Higher bandwidth between cores and controller
    - More information can be communicated (e.g. request’s importance in the processing core)
DRAM Controller (II)
A Modern DRAM Controller
DRAM Scheduling Policies (I)

- **FCFS** (first come first served)
  - Oldest request first

- **FR-FCFS** (first ready, first come first served)
  1. Row-hit first
  2. Oldest first

  Goal: Maximize row buffer hit rate \(ightarrow\) maximize DRAM throughput

- Actually, scheduling is done at the **command level**
  - Column commands (read/write) prioritized over row commands (activate/precharge)
  - Within each group, older commands prioritized over younger ones
DRAM Scheduling Policies (II)

- A scheduling policy is essentially a prioritization order

- Prioritization can be based on
  - Request age
  - Row buffer hit/miss status
  - Request type (prefetch, read, write)
  - Requestor type (load miss or store miss)
  - Request criticality
    - Oldest miss in the core?
    - How many instructions in core are dependent on it?
Row Buffer Management Policies

- Open row
  - Keep the row open after an access
    + Next access might need the same row → row hit
    -- Next access might need a different row → row conflict, wasted energy

- Closed row
  - Close the row after an access (if no other requests already in the request buffer need the same row)
    + Next access might need a different row → avoid a row conflict
    -- Next access might need the same row → extra activate latency

- Adaptive policies
  - Predict whether or not the next access to the bank will be to the same row
### Open vs. Closed Row Policies

<table>
<thead>
<tr>
<th>Policy</th>
<th>First access</th>
<th>Next access</th>
<th>Commands needed for next access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open row</td>
<td>Row 0</td>
<td>Row 0 (row hit)</td>
<td>Read</td>
</tr>
<tr>
<td>Open row</td>
<td>Row 0</td>
<td>Row 1 (row conflict)</td>
<td>Precharge + Activate Row 1 + Read</td>
</tr>
<tr>
<td>Closed row</td>
<td>Row 0</td>
<td>Row 0 – access in request buffer (row hit)</td>
<td>Read</td>
</tr>
<tr>
<td>Closed row</td>
<td>Row 0</td>
<td>Row 0 – access not in request buffer (row closed)</td>
<td>Activate Row 0 + Read + Precharge</td>
</tr>
<tr>
<td>Closed row</td>
<td>Row 0</td>
<td>Row 1 (row closed)</td>
<td>Activate Row 1 + Read + Precharge</td>
</tr>
</tbody>
</table>
Why are DRAM Controllers Difficult to Design?

- Need to obey **DRAM timing constraints** for correctness
  - There are many (50+) timing constraints in DRAM
  - $t_{WTR}$: Minimum number of cycles to wait before issuing a read command after a write command is issued
  - $t_{RC}$: Minimum number of cycles between the issuing of two consecutive activate commands to the same bank
  - ...

- Need to keep track of many resources to prevent conflicts
  - Channels, banks, ranks, data bus, address bus, row buffers

- Need to handle **DRAM refresh**

- Need to optimize for performance (in the presence of constraints)
  - Reordering is not simple
  - Predicting the future?
Many DRAM Timing Constraints

More on DRAM Operation


![Diagram of DRAM Access Phases](image)

**Table 2. Timing Constraints (DDR3-1066) [43]**

<table>
<thead>
<tr>
<th>Phase</th>
<th>Commands</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACT → READ, ACT → WRITE</td>
<td>tRCD</td>
<td>15ns</td>
</tr>
<tr>
<td></td>
<td>ACT → PRE</td>
<td>tRAS</td>
<td>37.5ns</td>
</tr>
<tr>
<td>2</td>
<td>READ → data, WRITE → data</td>
<td>tCL, tCWL</td>
<td>15ns, 11.25ns</td>
</tr>
<tr>
<td></td>
<td>data burst</td>
<td>tBL</td>
<td>7.5ns</td>
</tr>
<tr>
<td>3</td>
<td>PRE → ACT</td>
<td>tRP</td>
<td>15ns</td>
</tr>
<tr>
<td>1 &amp; 3</td>
<td>ACT → ACT</td>
<td>tRC (tRAS+tRP)</td>
<td>52.5ns</td>
</tr>
</tbody>
</table>

Figure 5. Three Phases of DRAM Access
DRAM Power Management

- DRAM chips have power modes
- Idea: *When not accessing a chip power it down*

- Power states
  - Active (highest power)
  - All banks idle
  - Power-down
  - Self-refresh (lowest power)

- State transitions incur latency during which the chip cannot be accessed