Memory Systems and Memory-Centric Computing Systems
Lecture 2, Topic 1: Memory Trends and Basics

Prof. Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
10 July 2018
HiPEAC ACACES Summer School 2018
What Will You Learn in This Course?

- Memory Systems and Memory-Centric Computing Systems
  - July 9-13, 2018

- Topic 1: Main Memory Trends and Basics

- Topic 2: Memory Reliability & Security: RowHammer and Beyond

- Topic 3: In-memory Computation

- Topic 4: Low-Latency and Low-Energy Memory

- Topic 5 (unlikely): Enabling and Exploiting Non-Volatile Memory

- Topic 6 (unlikely): Flash Memory and SSD Scaling

- Major Overview Reading:
Multiple Banks (Interleaving) and Channels

- Multiple banks
  - Enable concurrent DRAM accesses
  - Bits in address determine which bank an address resides in
- Multiple independent channels serve the same purpose
  - But they are even better because they have separate data buses
  - Increased bus bandwidth

- Enabling more concurrency requires reducing
  - Bank conflicts
  - Channel conflicts
- How to select/randomize bank/channel indices in address?
  - Lower order bits have more entropy
  - Randomizing hash functions (XOR of different address bits)
How Multiple Banks Help

Before: No Overlapping
Assuming accesses to different DRAM rows

After: Overlapped Accesses
Assuming no bank conflicts
Address Mapping (Single Channel)

- Single-channel system with 8-byte memory bus
  - 2GB memory, 8 banks, 16K rows & 2K columns per bank

- Row interleaving
  - Consecutive rows of memory in consecutive banks
  - Accesses to consecutive cache blocks serviced in a pipelined manner

- Cache block interleaving
  - Consecutive cache block addresses in consecutive banks
  - 64 byte cache blocks
  - Accesses to consecutive cache blocks can be serviced in parallel
Bank Mapping Randomization

- DRAM controller can randomize the address mapping to banks so that bank conflicts are less likely

Reading:
Address Mapping (Multiple Channels)

- Where are consecutive cache blocks?

<table>
<thead>
<tr>
<th>C</th>
<th>Row (14 bits)</th>
<th>High Column</th>
<th>Bank (3 bits)</th>
<th>Low Col.</th>
<th>Byte in bus (3 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 bits</td>
<td>3 bits</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Row (14 bits)</td>
<td>High Column</td>
<td>Bank (3 bits)</td>
<td>Low Col.</td>
<td>Byte in bus (3 bits)</td>
</tr>
<tr>
<td></td>
<td>8 bits</td>
<td>3 bits</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Row (14 bits)</td>
<td>High Column</td>
<td>C</td>
<td>Low Col.</td>
<td>Byte in bus (3 bits)</td>
</tr>
<tr>
<td></td>
<td>8 bits</td>
<td>3 bits</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Row (14 bits)</td>
<td>High Column</td>
<td>Bank (3 bits)</td>
<td>C</td>
<td>Byte in bus (3 bits)</td>
</tr>
<tr>
<td></td>
<td>8 bits</td>
<td>3 bits</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Row (14 bits)</td>
<td>High Column</td>
<td>Bank (3 bits)</td>
<td>Low Col.</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>8 bits</td>
<td>3 bits</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interaction with Virtual → Physical Mapping

- **Operating System influences where an address maps to in DRAM**

- **Operating system can influence which bank/channel/rank a virtual page is mapped to.**

- **It can perform **page coloring** to**
  - Minimize bank conflicts
  - Minimize inter-application interference [Muralidhara+ MICRO’11]
  - Minimize latency in the network [Das+ HPCA’13]
Memory Channel Partitioning

- Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda,
"Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning"
Proceedings of the 44th International Symposium on Microarchitecture (MICRO), Porto Alegre, Brazil, December 2011. Slides (pptx)

Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning

Sai Prashanth Muralidhara  
Pennsylvania State University  
smuralid@cse.psu.edu

Lavanya Subramanian  
Carnegie Mellon University  
lsubrama@ece.cmu.edu

Onur Mutlu  
Carnegie Mellon University  
onur@cmu.edu

Mahmut Kandemir  
Pennsylvania State University  
kandemir@cse.psu.edu

Thomas Moscibroda  
Microsoft Research Asia  
moscitho@microsoft.com
Application-to-Core Mapping

- Reetuparna Das, Rachata Ausavarungnirun, Onur Mutlu, Akhilesh Kumar, and Mani Azimi,
"Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems"
Slides (pptx)
More on Reducing Bank Conflicts

- Read Sections 1 through 4 of:

Figure 1. DRAM bank organization
Subarray Level Parallelism


A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM

Yoongu Kim    Vivek Seshadri    Donghyuk Lee    Jamie Liu    Onur Mutlu
Carnegie Mellon University
DRAM Refresh (I)

- DRAM capacitor charge leaks over time
- The memory controller needs to read each row periodically to restore the charge
  - Activate + precharge each row every N ms
  - Typical N = 64 ms
- Implications 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 we can reduce the effect of refresh on performance?
  - Can we reduce the number of refreshes?
Downsides of DRAM 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 density scaling**

Liu et al., "**RAIDR: Retention-aware Intelligent DRAM Refresh**," ISCA 2012.
More on DRAM Refresh

- Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh"
  Slides (pdf)
DRAM Retention Analysis


An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms

Jamie Liu*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
jamiel@alumni.cmu.edu

Ben Jaiyen*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
bjaiyen@alumni.cmu.edu

Yoongu Kim
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
yoonguk@ece.cmu.edu

Chris Wilkerson
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95054
chris.wilkerson@intel.com

Onur Mutlu
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
onur@cmu.edu
Data Retention in Memory [Liu et al., ISCA 2013]

- Data Retention Time Profile of DRAM looks like this:

- Location dependent
- Stored value pattern dependent
- Time dependent
Reducing Performance Impact of DRAM Refresh by Parallelizing Refreshes with Accesses

Kevin Kai-Wei Chang  Donghyuk Lee  Zeshan Chishti†
Alaa R. Alameldeen†  Chris Wilkerson†  Yoongu Kim  Onur Mutlu
Carnegie Mellon University  †Intel Labs
Memory Controllers
Long latency memories have similar characteristics that need to be controlled.

The following discussion will use DRAM as an example, but many scheduling and control 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
- These other technologies can place other demands on the controller
Flash Memory (SSD) Controllers

- Similar to DRAM memory controllers, except:
  - They are flash memory specific
  - They do much more: error correction, garbage collection, page remapping, ...

Another View of the SSD Controller

Fig. 1. (a) SSD system architecture, showing controller (Ctrl) and chips. (b) Detailed view of connections between controller components and chips.


Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

This paper reviews the most recent advances in solid-state drive (SSD) error characterization, mitigation, and data recovery techniques to improve both SSD’s reliability and lifetime.

By Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu

https://arxiv.org/pdf/1706.08642
On Modern SSD Controllers (II)

- Arash Tavakkol, Juan Gomez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu,

"MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices"


[Slides (pptx) (pdf)]
[Source Code]

---

MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices

Arash Tavakkol†, Juan Gómez-Luna†, Mohammad Sadrosadati†, Saugata Ghose‡, Onur Mutlu††

†ETH Zürich ‡Carnegie Mellon University
On Modern SSD Controllers (III)

- Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan G. Luna and Onur Mutlu,

"FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives"


[Slides (pptx) (pdf)] [Lightning Talk Slides (pptx) (pdf)]
[Lightning Talk Video]

---

**FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives**

Arash Tavakkol† Mohammad Sadrosadati† Saugata Ghose‡ Jeremie S. Kim‡‡ Yixin Luo‡ Yaohua Wang‡§ Nika Mansouri Ghiasi† Lois Orosa‡* Juan Gómez-Luna† Onur Mutlu‡‡

†ETH Zürich ‡Carnegie Mellon University ‡‡NUDT *Unicamp
DRAM Types

- DRAM has different types with different interfaces optimized for different purposes
  - Commodity: DDR, DDR2, DDR3, DDR4, ...
  - Low power (for mobile): LPDDR1, ..., LPDDR5, ...
  - High bandwidth (for graphics): GDDR2, ..., GDDR5, ...
  - Low latency: eDRAM, RLDRAM, ...
  - 3D stacked: WIO, HBM, HMC, ...
  - ...

- Underlying microarchitecture is fundamentally the same
- A flexible memory controller can support various DRAM types
- This complicates the memory controller
  - Difficult to support all types (and upgrades)
## DRAM Types (circa 2015)

<table>
<thead>
<tr>
<th>Segment</th>
<th>DRAM Standards &amp; Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commodity</td>
<td>DDR3 (2007) [14]; DDR4 (2012) [18]</td>
</tr>
<tr>
<td>Performance</td>
<td>eDRAM [28], [32]; RLDRAM3 (2011) [29]</td>
</tr>
</tbody>
</table>

Table 1. Landscape of DRAM-based 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 for high performance + QoS**
  - Reordering, row-buffer, bank, rank, bus management

- **Manage power consumption and thermals in DRAM**
  - Turn on/off DRAM chips, manage power modes
A Modern DRAM Controller (I)
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 $\rightarrow$ 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
Review: DRAM Bank Operation

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

Row address 0

Columns

Row decoder

Rows

Column address 85

Row 1

Column mux

Data

CONFLICT!
A scheduling policy is a request 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?
  - Will it stall the processor?
- Interference caused to other cores
- ...
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 and act accordingly
# 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>
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)

- Tradeoff: State transitions incur latency during which the chip cannot be accessed
Difficulty of DRAM Control
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 **manage power** consumption

- Need to **optimize performance & QoS** (in the presence of constraints)
  - Reordering is not simple
  - Fairness and QoS needs complicates the scheduling problem
Many DRAM Timing Constraints


<table>
<thead>
<tr>
<th>Latency</th>
<th>Symbol</th>
<th>DRAM cycles</th>
<th>Latency</th>
<th>Symbol</th>
<th>DRAM cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precharge</td>
<td>$t_{RP}$</td>
<td>11</td>
<td>Activate to read/write</td>
<td>$t_{RCD}$</td>
<td>11</td>
</tr>
<tr>
<td>Read column address strobe</td>
<td>$CL$</td>
<td>11</td>
<td>Write column address strobe</td>
<td>$CW_L$</td>
<td>8</td>
</tr>
<tr>
<td>Additive</td>
<td>$AL$</td>
<td>0</td>
<td>Activate to activate</td>
<td>$t_{RC}$</td>
<td>39</td>
</tr>
<tr>
<td>Activate to precharge</td>
<td>$t_{RAS}$</td>
<td>28</td>
<td>Read to precharge</td>
<td>$t_{RTP}$</td>
<td>6</td>
</tr>
<tr>
<td>Burst length</td>
<td>$t_{BL}$</td>
<td>4</td>
<td>Column address strobe to column address strobe</td>
<td>$t_{CCD}$</td>
<td>4</td>
</tr>
<tr>
<td>Activate to activate (different bank)</td>
<td>$t_{RRD}$</td>
<td>6</td>
<td>Four activate windows</td>
<td>$t_{FAW}$</td>
<td>24</td>
</tr>
<tr>
<td>Write to read</td>
<td>$t_{WTR}$</td>
<td>6</td>
<td>Write recovery</td>
<td>$t_{WR}$</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 4. DDR3 1600 DRAM timing specifications
More on DRAM Operation


![Figure 5. Three Phases of DRAM Access](image)

<table>
<thead>
<tr>
<th>Table 2. Timing Constraints (DDR3-1066) [43]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase</td>
</tr>
</tbody>
</table>
| 1    | ACT $\rightarrow$ READ  
ACT $\rightarrow$ WRITE | $t_{RCD}$ | 15ns |
|      | ACT $\rightarrow$ PRE | $t_{RAS}$ | 37.5ns |
| 2    | READ $\rightarrow$ data  
WRITE $\rightarrow$ data | $t_{CL}$  
$t_{CWL}$ | 15ns  
11.25ns |
|      | data burst | $t_{BL}$ | 7.5ns |
| 3    | PRE $\rightarrow$ ACT | $t_{RP}$ | 15ns |
| 1 & 3 | ACT $\rightarrow$ ACT | ($t_{RAS} + t_{RP}$) | 52.5ns |
Why So Many Timing Constraints? (I)

Figure 4. DRAM bank operation: Steps involved in serving a memory request [17] \((V_{PP} > V_{DD})\)

<table>
<thead>
<tr>
<th>Category</th>
<th>RowCmd↔RowCmd</th>
<th>RowCmd↔ColCmd</th>
<th>ColCmd↔ColCmd</th>
<th>ColCmd→DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>tRC</td>
<td>tRAS</td>
<td>tRP</td>
<td>tCCD</td>
</tr>
<tr>
<td>Commands</td>
<td>A→A</td>
<td>A→P</td>
<td>P→A</td>
<td>R(W)→R(W)</td>
</tr>
<tr>
<td>Scope</td>
<td>Bank</td>
<td>Bank</td>
<td>Bank</td>
<td>Channel</td>
</tr>
<tr>
<td>Value (ns)</td>
<td>~50</td>
<td>~35</td>
<td>13-15</td>
<td>5-7.5</td>
</tr>
</tbody>
</table>

A: ACTIVATE—P: PRECHARGE—R: READ—W: WRITE

* Goes into effect after the last write data, not from the WRITE command
† Not explicitly specified by the JEDEC DDR3 standard [18]. Defined as a function of other timing constraints.

Table 1. Summary of DDR3-SDRAM timing constraints (derived from Micron’s 2Gb DDR3-SDRAM datasheet [33])

Why So Many Timing Constraints? (II)

1. Activation
2. Sensing & Amplification
3. Precharging

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</td>
<td>tRCD</td>
<td>15ns</td>
</tr>
<tr>
<td></td>
<td>ACT → WRITE</td>
<td>trCD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACT → PRE</td>
<td>trAS</td>
<td>37.5ns</td>
</tr>
<tr>
<td>2</td>
<td>READ → data</td>
<td>tCL</td>
<td>15ns</td>
</tr>
<tr>
<td></td>
<td>WRITE → data</td>
<td>tCWL</td>
<td>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</td>
<td>52.5ns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(tRAS + tRP)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Charge Flow Between the Cell Capacitor ($C_C$), Bitline Parasitic Capacitor ($C_B$), and the Sense-Amplifier ($C_B \approx 3.5C_C$ [39]).

DRAM Controller Design Is Becoming More Difficult

- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs
- Many timing constraints for various memory types
- Many goals at the same time: performance, fairness, QoS, energy efficiency, ...
Reality and Dream

- Reality: It difficult to optimize all these different constraints while maximizing performance, QoS, energy-efficiency, ...

- Dream: Wouldn’t it be nice if the DRAM controller automatically found a good scheduling policy on its own?
Self-Optimizing DRAM Controllers

- Problem: DRAM controllers difficult to design → It is difficult for human designers to design a policy that can adapt itself very well to different workloads and different system conditions.

- Idea: Design a memory controller that adapts its scheduling policy decisions to workload behavior and system conditions using machine learning.

- Observation: Reinforcement learning maps nicely to memory control.

- Design: Memory controller is a reinforcement learning agent that dynamically and continuously learns and employs the best scheduling policy.

Goal: Learn to choose actions to maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + ... \ (0 \leq \gamma < 1)$
Self-Optimizing DRAM Controllers

- Dynamically adapt the memory scheduling policy via interaction with the system at runtime
  - Associate system states and actions (commands) with long term reward values: each action at a given state leads to a learned reward
  - Schedule command with highest estimated long-term reward value in each state
  - Continuously update reward values for \(<\text{state}, \text{action}>\) pairs based on feedback from system
Self-Optimizing DRAM Controllers

Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana,
"Self Optimizing Memory Controllers: A Reinforcement Learning Approach"

Figure 4: High-level overview of an RL-based scheduler.
States, Actions, Rewards

❖ Reward function

• +1 for scheduling Read and Write commands
• 0 at all other times

Goal is to maximize long-term data bus utilization

❖ State attributes

• Number of reads, writes, and load misses in transaction queue
• Number of pending writes and ROB heads waiting for referenced row
• Request’s relative ROB order

❖ Actions

• Activate
• Write
• Read - load miss
• Read - store miss
• Precharge - pending
• Precharge - preemptive
• NOP
Performance Results

**Figure 7:** Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers.

**Figure 15:** Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth.
Self Optimizing DRAM Controllers

- Advantages
  + Adapts the scheduling policy dynamically to changing workload behavior and to maximize a long-term target
  + Reduces the designer’s burden in finding a good scheduling policy. Designer specifies:
    1) What system variables might be useful
    2) What target to optimize, but not how to optimize it

- Disadvantages and Limitations
  -- Black box: designer much less likely to implement what she cannot easily reason about
  -- How to specify different reward functions that can achieve different objectives? (e.g., fairness, QoS)
  -- Hardware complexity?
More on Self-Optimizing DRAM Controllers

Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana, *Self Optimizing Memory Controllers: A Reinforcement Learning Approach* 

Self-Optimizing Memory Controllers: A Reinforcement Learning Approach

Engin İpek\(^1,2\) Onur Mutlu\(^2\) José F. Martínez\(^1\) Rich Caruana\(^1\)

\(^1\)Cornell University, Ithaca, NY 14850 USA  
\(^2\)Microsoft Research, Redmond, WA 98052 USA
Simulating Memory
Ramulator: A Fast and Extensible DRAM Simulator

[IEEE Comp Arch Letters’15]
Ramulator Motivation

- DRAM and Memory Controller landscape is changing
- Many new and upcoming standards
- Many new controller designs
- A fast and easy-to-extend simulator is very much needed

<table>
<thead>
<tr>
<th>Segment</th>
<th>DRAM Standards &amp; Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commodity</td>
<td>DDR3 (2007) [14]; DDR4 (2012) [18]</td>
</tr>
<tr>
<td>Performance</td>
<td>eDRAM [28], [32]; RLDRAM3 (2011) [29]</td>
</tr>
</tbody>
</table>

Table 1. Landscape of DRAM-based memory
Ramulator

- Provides out-of-the-box support for many DRAM standards:
  - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, plus new proposals (SALP, AL-DRAM, TLDRA M, RowClone, and SARP)
- ~2.5X faster than fastest open-source simulator
- Modular and extensible to different standards

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Cycles (10^6)</th>
<th>Runtime (sec.)</th>
<th>Req/sec (10^3)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random</td>
<td>Stream</td>
<td>Random</td>
<td>Stream</td>
</tr>
<tr>
<td>Ramulator</td>
<td>652</td>
<td>411</td>
<td>752</td>
<td>249</td>
</tr>
<tr>
<td>DRAMSim2</td>
<td>645</td>
<td>413</td>
<td>2,030</td>
<td>876</td>
</tr>
<tr>
<td>USIMM</td>
<td>661</td>
<td>409</td>
<td>1,880</td>
<td>750</td>
</tr>
<tr>
<td>DrSim</td>
<td>647</td>
<td>406</td>
<td>18,109</td>
<td>12,984</td>
</tr>
<tr>
<td>NVMMain</td>
<td>666</td>
<td>413</td>
<td>6,881</td>
<td>5,023</td>
</tr>
</tbody>
</table>

Table 3. Comparison of five simulators using two traces
Case Study: Comparison of DRAM Standards

<table>
<thead>
<tr>
<th>Standard</th>
<th>Rate (MT/s)</th>
<th>Timing (CL-RCD-RP)</th>
<th>Data-Bus (Width × Chan.)</th>
<th>Rank-per-Chan</th>
<th>BW (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR3</td>
<td>1,600</td>
<td>11-11-11</td>
<td>64-bit × 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>DDR4</td>
<td>2,400</td>
<td>16-16-16</td>
<td>64-bit × 1</td>
<td>1</td>
<td>17.9</td>
</tr>
<tr>
<td>SALP†</td>
<td>1,600</td>
<td>11-11-11</td>
<td>64-bit × 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>LPDDR3</td>
<td>1,600</td>
<td>12-15-15</td>
<td>64-bit × 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>LPDDR4</td>
<td>2,400</td>
<td>22-22-22</td>
<td>32-bit × 2*</td>
<td>1</td>
<td>17.9</td>
</tr>
<tr>
<td>GDDR5 [12]</td>
<td>6,000</td>
<td>18-18-18</td>
<td>64-bit × 1</td>
<td>1</td>
<td>44.7</td>
</tr>
<tr>
<td>HBM</td>
<td>1,000</td>
<td>7-7-7</td>
<td>128-bit × 8*</td>
<td>1</td>
<td>119.2</td>
</tr>
<tr>
<td>WIO</td>
<td>266</td>
<td>7-7-7</td>
<td>128-bit × 4*</td>
<td>1</td>
<td>15.9</td>
</tr>
<tr>
<td>WIO2</td>
<td>1,066</td>
<td>9-10-10</td>
<td>128-bit × 8*</td>
<td>1</td>
<td>127.2</td>
</tr>
</tbody>
</table>

Across 22 workloads, simple CPU model

Figure 2. Performance comparison of DRAM standards
Ramulator Paper and Source Code


- Source code is released under the liberal MIT License
  - [https://github.com/CMU-SAFARI/ramulator](https://github.com/CMU-SAFARI/ramulator)

Ramulator: A Fast and Extensible DRAM Simulator

Yoongu Kim¹, Weikun Yang¹,², Onur Mutlu¹
¹Carnegie Mellon University ²Peking University
Optional Assignment

- **Review the Ramulator paper**
  - Email me your review ([omutlu@gmail.com](mailto:omutlu@gmail.com))

- **Download and run Ramulator**
  - Compare DDR3, DDR4, SALP, HBM for the libquantum benchmark (provided in Ramulator repository)
  - Email me your report ([omutlu@gmail.com](mailto:omutlu@gmail.com))

- This may help you get into memory systems research quickly
Topics We Will Not Cover
No Time, Unfortunately, for:

- Memory Interference and QoS
- Predictable Performance
  - QoS-aware Memory Controllers
- Emerging Memory Technologies and Hybrid Memories
- Cache Management
- Interconnects

You can find many materials on these at my online lectures
  - https://people.inf.ethz.ch/omutlu/teaching.html
Some More Suggested Readings
Some Key Readings on DRAM (I)

- **DRAM Organization and Operation**


Some Key Readings on DRAM (II)

- **DRAM Refresh**
Reading on Simulating Main Memory

- How to evaluate future main memory systems?
- An open-source simulator and its brief description

Some Key Readings on Memory Control 1

  https://people.inf.ethz.ch/omutlu/pub/parbs_isca08.pdf


https://people.inf.ethz.ch/omutlu/pub/rlmc_isca08.pdf


Lee et al., “Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM,” PACT 2015.
More Readings

- To come as we cover the future topics

- Search for “DRAM” or “Memory” in:
  - [https://people.inf.ethz.ch/omutlu/projects.htm](https://people.inf.ethz.ch/omutlu/projects.htm)
Memory Systems and Memory-Centric Computing Systems

Lecture 2, Topic 1: Memory Trends and Basics

Prof. Onur Mutlu

omutlu@gmail.com

https://people.inf.ethz.ch/omutlu

10 July 2018

HiPEAC ACACES Summer School 2018
Backup Slides
Inside A DRAM Chip
DRAM Module and Chip
Goals

• Cost
• Latency
• Bandwidth
• Parallelism
• Power
• Energy
• Reliability
• ...
DRAM Chip
Sense Amplifier

enable

Inverter
Sense Amplifier – Two Stable States

Logical “1”

Logical “0”
Sense Amplifier Operation

\[ V_T > V_B \]
DRAM Cell – Capacitor

Empty State

Logical “0”

1. Small – Cannot drive circuits

2. Reading destroys the state

Fully Charged State

Logical “1”
Capacitor to Sense Amplifier

\[ V_{DD} \]

\[ 0 \]

\[ 1 \]

\[ V_{DD} \]

\[ 0 \]

\[ 1 \]
DRAM Cell Operation

\[ \frac{1}{2} V_{DD} + \delta \]

\[ \frac{1}{2} V_{DD} \]
DRAM Subarray – Building Block for DRAM Chip

Row Decoder

Cell Array

Array of Sense Amplifiers (Row Buffer) 8Kb

Cell Array
DRAM Bank

Address

Row Decoder

Cell Array

Array of Sense Amplifiers (8Kb)

Cell Array

Array of Sense Amplifiers

Cell Array

Bank I/O (64b)

Address

Data
DRAM Chip

Shared internal bus

Memory channel - 8bits
DRAM Operation

1. ACTIVATE Row
2. READ/WRITE Column
3. PRECHARGE
Evaluating New Ideas for New (Memory) Architectures
Potential Evaluation Methods

- How do we assess an idea will improve a target metric X?

- A variety of evaluation methods are available:
  - Theoretical proof
  - Analytical modeling/estimation
  - Simulation (at varying degrees of abstraction and accuracy)
  - Prototyping with a real system (e.g., FPGAs)
  - Real implementation
The Difficulty in Architectural Evaluation

- The answer is usually workload dependent
  - E.g., think caching
  - E.g., think pipelining
  - E.g., think any idea we talked about (RAIDR, Mem. Sched., ...)

- Workloads change

- System has many design choices and parameters
  - Architect needs to decide many ideas and many parameters for a design
  - Not easy to evaluate all possible combinations!

- System parameters may change
Simulation: The Field of Dreams
Dreaming and Reality

- An architect is in part a dreamer, a creator
- Simulation is a key tool of the architect
- Simulation enables
  - The exploration of many dreams
  - A reality check of the dreams
  - Deciding which dream is better
- Simulation also enables
  - The ability to fool yourself with false dreams
Why High-Level Simulation?

- **Problem:** RTL simulation is intractable for design space exploration → too time consuming to design and evaluate
  - Especially over a large number of workloads
  - Especially if you want to predict the performance of a good chunk of a workload on a particular design
  - Especially if you want to consider many design choices
    - Cache size, associativity, block size, algorithms
    - Memory control and scheduling algorithms
    - In-order vs. out-of-order execution
    - Reservation station sizes, ld/st queue size, register file size, ...
    - ...

- **Goal:** Explore design choices quickly to see their impact on the workloads we are designing the platform for
Different Goals in Simulation

- **Explore the design space quickly** and see what you want to
  - potentially implement in a next-generation platform
  - propose as the next big idea to advance the state of the art
  - the goal is mainly to see relative effects of design decisions

- **Match the behavior of an existing system** so that you can
  - debug and verify it at cycle-level accuracy
  - propose small tweaks to the design that can make a difference in performance or energy
  - the goal is very high accuracy

- **Other goals in-between:**
  - **Refine the explored design space** without going into a full detailed, cycle-accurate design
  - **Gain confidence in your design decisions** made by higher-level design space exploration
Tradeoffs in Simulation

- Three metrics to evaluate a simulator
  - Speed
  - Flexibility
  - Accuracy

- Speed: How fast the simulator runs (xIPS, xCPS, slowdown)
- Flexibility: How quickly one can modify the simulator to evaluate different algorithms and design choices?
- Accuracy: How accurate the performance (energy) numbers the simulator generates are vs. a real design (Simulation error)

- The relative importance of these metrics varies depending on where you are in the design process (what your goal is)
Trading Off Speed, Flexibility, Accuracy

- **Speed & flexibility affect:**
  - How quickly you can make design tradeoffs

- **Accuracy affects:**
  - How good your design tradeoffs *may* end up being
  - How fast you can build your simulator (simulator design time)

- **Flexibility also affects:**
  - How much human effort you need to spend modifying the simulator

- You can **trade off between the three to achieve design exploration and decision goals**
High-Level Simulation

Key Idea: Raise the abstraction level of modeling to give up some accuracy to enable speed & flexibility (and quick simulator design)

Advantage
+ Can still make the right tradeoffs, and can do it quickly
  + All you need is modeling the key high-level factors, you can omit corner case conditions
  + All you need is to get the “relative trends” accurately, not exact performance numbers

Disadvantage
-- Opens up the possibility of potentially wrong decisions
  -- How do you ensure you get the “relative trends” accurately?
Simulation as Progressive Refinement

- High-level models (Abstract, C)
- ...
- Medium-level models (Less abstract)
- ...
- Low-level models (RTL with everything modeled)
- ...
- Real design

As you refine (go down the above list)
  - Abstraction level reduces
  - Accuracy (hopefully) increases (not necessarily, if not careful)
  - Flexibility reduces; Speed likely reduces except for real design
  - You can loop back and fix higher-level models
Making The Best of Architecture

- A good architect is comfortable at all levels of refinement
  - Including the extremes

- A good architect knows when to use what type of simulation
  - And, more generally, what type of evaluation method

- Recall: A variety of evaluation methods are available:
  - Theoretical proof
  - Analytical modeling
  - Simulation (at varying degrees of abstraction and accuracy)
  - Prototyping with a real system (e.g., FPGAs)
  - Real implementation
Ramulator: A Fast and Extensible DRAM Simulator

[IEEE Comp Arch Letters’15]
Ramulator Motivation

- DRAM and Memory Controller landscape is changing
- Many new and upcoming standards
- Many new controller designs
- A fast and easy-to-extend simulator is very much needed

<table>
<thead>
<tr>
<th>Segment</th>
<th>DRAM Standards &amp; Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commodity</td>
<td>DDR3 (2007) [14]; DDR4 (2012) [18]</td>
</tr>
<tr>
<td>Performance</td>
<td>eDRAM [28], [32]; RLDRAAM3 (2011) [29]</td>
</tr>
</tbody>
</table>

Table 1. Landscape of DRAM-based memory
Ramulator

- Provides out-of-the-box support for many DRAM standards:
  - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, plus new proposals (SALP, AL-DRAM, TLDRAK, RowClone, and SARP)
- ~2.5X faster than fastest open-source simulator
- Modular and extensible to different standards

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Cycles (10^6)</th>
<th>Runtime (sec.)</th>
<th>Req/sec (10^3)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random</td>
<td>Stream</td>
<td>Random</td>
<td>Stream</td>
</tr>
<tr>
<td>Ramulator</td>
<td>652</td>
<td>411</td>
<td>752</td>
<td>249</td>
</tr>
<tr>
<td>DRAMSIm2</td>
<td>645</td>
<td>413</td>
<td>2,030</td>
<td>876</td>
</tr>
<tr>
<td>USIMM</td>
<td>661</td>
<td>409</td>
<td>1,880</td>
<td>750</td>
</tr>
<tr>
<td>DrSim</td>
<td>647</td>
<td>406</td>
<td>18,109</td>
<td>12,984</td>
</tr>
<tr>
<td>NVMMain</td>
<td>666</td>
<td>413</td>
<td>6,881</td>
<td>5,023</td>
</tr>
</tbody>
</table>

Table 3. Comparison of five simulators using two traces
## Case Study: Comparison of DRAM Standards

<table>
<thead>
<tr>
<th>Standard</th>
<th>Rate (MT/s)</th>
<th>Timing (CL-RCD-RP)</th>
<th>Data-Bus (Width×Chan.)</th>
<th>Rank-per-Chan</th>
<th>BW (GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR3</td>
<td>1,600</td>
<td>11-11-11</td>
<td>64-bit × 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>DDR4</td>
<td>2,400</td>
<td>16-16-16</td>
<td>64-bit × 1</td>
<td>1</td>
<td>17.9</td>
</tr>
<tr>
<td>SALP†</td>
<td>1,600</td>
<td>11-11-11</td>
<td>64-bit × 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>LPDDR3</td>
<td>1,600</td>
<td>12-15-15</td>
<td>64-bit × 1</td>
<td>1</td>
<td>11.9</td>
</tr>
<tr>
<td>LPDDR4</td>
<td>2,400</td>
<td>22-22-22</td>
<td>32-bit × 2*</td>
<td>1</td>
<td>17.9</td>
</tr>
<tr>
<td>GDDR5 [12]</td>
<td>6,000</td>
<td>18-18-18</td>
<td>64-bit × 1</td>
<td>1</td>
<td>44.7</td>
</tr>
<tr>
<td>HBM</td>
<td>1,000</td>
<td>7-7-7</td>
<td>128-bit × 8*</td>
<td>1</td>
<td>119.2</td>
</tr>
<tr>
<td>WIO</td>
<td>266</td>
<td>7-7-7</td>
<td>128-bit × 4*</td>
<td>1</td>
<td>15.9</td>
</tr>
<tr>
<td>WIO2</td>
<td>1,066</td>
<td>9-10-10</td>
<td>128-bit × 8*</td>
<td>1</td>
<td>127.2</td>
</tr>
</tbody>
</table>

### Figure 2. Performance comparison of DRAM standards

Across 22 workloads, simple CPU model
Ramulator Paper and Source Code


- Source code is released under the liberal MIT License
  - https://github.com/CMU-SAFARI/ramulator
Optional Assignment

- Review the Ramulator paper
  - Email me your review

- Download and run Ramulator
  - Compare DDR3, DDR4, SALP, HBM for the libquantum benchmark (provided in Ramulator repository)
  - Email me your report

- This may help you get into memory systems research quickly
End of Backup Slides