P&S Ramulator
Designing and Evaluating Memory Systems and Modern Software Workloads with Ramulator

Hasan Hassan
Prof. Onur Mutlu
ETH Zürich
Spring 2022
9 March 2022
You will learn in detail how modern memory systems operate.

You will design new DRAM and memory controller mechanisms for improving overall system performance, energy consumption, and reliability.

You will simulate and understand the memory system behavior of modern workloads such as machine learning, graph analytics, genome analysis.
P&S Ramulator: Key Takeaways

- This P&S is aimed at improving your
  - **Knowledge** in Computer Architecture and Memory Systems
  - **Technical skills** in simulating memory systems
  - **Critical thinking and analysis**
  - **Interaction** with a nice group of researchers
  - **Familiarity with key research directions**
  - **Technical presentation** of your project
Learn how state-of-the-art memory controllers operate, design new DRAM and memory controller mechanisms, and evaluate your mechanisms using simulation.
Prerequisites of the Course

- Digital Design and Computer Architecture (or equivalent course)

- A good knowledge in C/C++ programming language

- Interest in making things efficient and solving problems

- Interest in understanding software development and hardware design, and their interaction
Onur Mutlu
- Full Professor @ ETH Zurich ITET (INFK), since September 2015
- Strecker Professor @ Carnegie Mellon University ECE/CS, 2009-2016, 2016-...
- PhD from UT-Austin, worked at Google, VMware, Microsoft Research, Intel, AMD
- [https://people.inf.ethz.ch/omutlu/](https://people.inf.ethz.ch/omutlu/)
- omutlu@gmail.com (Best way to reach me)
- [https://people.inf.ethz.ch/omutlu/projects.htm](https://people.inf.ethz.ch/omutlu/projects.htm)

Research and Teaching in:
- Computer architecture, computer systems, hardware security, bioinformatics
- Memory and storage systems
- Hardware security, safety, predictability
- Fault tolerance
- Hardware/software cooperation
- Architectures for bioinformatics, health, medicine
- ...
Course Info: Who Are We? (II)

- **Lead Supervisor:**
  - Hasan Hassan

- **Supervisors:**
  - Geraldo de Oliveira
  - Giray Yaglikci
  - Ataberk Olgun
  - Haocong Luo
  - Jeremie Kim
  - Minesh Patel

- **Get to know us and our research**
  - [https://safari.ethz.ch/group-members](https://safari.ethz.ch/group-members)
Onur Mutlu’s SAFARI Research Group

Computer architecture, HW/SW, systems, bioinformatics, security, memory

https://safari.ethz.ch/safari-newsletter-april-2020/

Think BIG, Aim HIGH!

https://safari.ethz.ch
Current Research Focus Areas

**Research Focus: Computer architecture, HW/SW, bioinformatics**

- Memory and storage (DRAM, flash, emerging), interconnects
- Heterogeneous & parallel systems, GPUs, systems for data analytics
- System/architecture interaction, new execution models, new interfaces
- Energy efficiency, fault tolerance, hardware security, performance
- Genome sequence analysis & assembly algorithms and architectures
- Biologically inspired systems & system design for bio/medicine

**Hybrid Main Memory**

**Heterogeneous Processors and Accelerators**

**Persistent Memory/Storage**

**Broad research spanning apps, systems, logic with architecture at the center**

**Graphics and Vision Processing**
Course Info: How About You?

- Let us know your background, interests
- Why did you join this P&S?
- Please submit HW0
Course Requirements and Expectations

- Attendance required for all meetings

- Study the learning materials

- Each student will carry out a hands-on project
  - Build, implement, code, and design with close engagement from the supervisors

- Participation
  - Ask questions, contribute thoughts/ideas
  - Read relevant papers

We will help in all projects!
If your work is really good, you may get it published!
Course Website

- [https://safari.ethz.ch/projects_and_seminars/doku.php?id=ramulator](https://safari.ethz.ch/projects_and_seminars/doku.php?id=ramulator)

- Useful information about the course

- Check your email frequently for announcements
Meeting 1

Learning materials:

- The github version of Ramulator: https://github.com/CMU-SAFARI/ramulator
- Three examples of new ideas enabled by Ramulator based evaluation:
Meeting 2 (TBD)

- We will announce the projects and will give you some description about them

- You will have a week to submit your project preferences

- The supervisors would like to help you with selecting a project that matches your interests, skills, and background

- It is important that you study the learning materials before our next meeting!
Tentative Weekly Schedule

- Week 1 – Logistics & Intro to Simulating Memory Systems using Ramulator
- Week 2 – Tutorial on Using Ramulator | Available Projects
- Week 3 – BlockHammer [HPCA’21] | PU
- Week 4 – CLR-DRAM [ISCA’20] | PU
- Week 5 – CODIC [ISCA’21] | PU
- Week 6 – SIMDARM [ASPLOS’17] | PU
- Week 7 – DAMOV [IEEE Access’21] | PU
- Week 8 – Syncron [HPCA’21] | PU
- Week 9+ - PU

- PU = Project Updates

- How about meeting every Thursday at 18:00?
Performance Assessment

We expect you to:

- **Learn** how DRAM operates and how to analyze performance of memory systems using simulation

- **Achieve the goals** of your project

- **Deliver** your **code and results** with sufficient documentation

- Prepare a **final presentation** and present your work to SAFARI
An Introduction to Simulating Memory Systems with 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: A Fast and Extensible DRAM Simulator

- Provides out-of-the-box support for many DRAM standards:
  - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, HMC, and academic proposals (SALP, AL-DRAM, TLDRAM, RowClone, and SARP)
  - Models timing of non-volatile memories (PCM, STT-MRAM)
- Supports multiple scheduling and row buffer management policies
- Modular and extensible to different standards
- Can be paired with other simulators, e.g., gem5 and DRAMPower
- Written in C++11
- ~2.5X faster than fastest open-source simulator

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Cycles (10^6)</th>
<th>Runime (sec)</th>
<th>Reg/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>NVMain</td>
<td>666</td>
<td>413</td>
<td>6,881</td>
<td>5,023</td>
</tr>
</tbody>
</table>
### 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
Saugata Ghose, Tianshi Li, Nastaran Hajinazar, Damla Senol Cali, and Onur Mutlu,
"Demystifying Workload–DRAM Interactions: An Experimental Study"
[Preliminary arXiv Version]
[Abstract]
[Slides (pptx) (pdf)]
Simulator Architecture
Design Objective: Extensibility

- Treats *extensibility* as a first-class citizen

**Observation:** DRAM can be abstracted as a hierarchy of *state machines*

- Provides a *standard-agnostic* state machine
  - Paired with any standard at compile time
DRAM Organization
DRAM Operations

Memory Bus

Precharge

CPU

DRAM Cell

DRAM Row

Sense Amplifier
DRAM Latency

Retention Time: The interval during which the data is retained correctly in the DRAM cell without accessing it.
The ‘DRAM’ class: a template for building a hierarchy of state machines (i.e., nodes)

```cpp
// DRAM.h
template <typename T>
class DRAM {
    DRAM<T>* parent;
    vector<DRAM<T>*> children;
    T::Level level;
    int index;
    // more code...
};

// DDR3.h/cpp
class DDR3 {
    enum class Level {
        Channel, Rank, Bank, Row, Column, MAX
    };
};
```

```
DRAM<DDR3> Instance
- level = DDR3::Level::Channel
- index = 0

DRAM<DDR3>
- Rank
  - 0

DRAM<DDR3>
- Rank
  - 1

DRAM<DDR3>
- Rank
  - 2

DRAM<DDR3>
- Bank
  - 0

... DRAM<DDR3>
- Bank
  - 7
```
**DRAM Node States**

- **status**: may change when the node receives one of the DDR3 commands.
- **next**: a lookup table specifying the earliest time the node can receive each command (for honoring DDR3 timing parameters).

```cpp
// DRAM.h
template <typename T>
class DRAM {
  // states (queried/updated by functions below)
  T::Status status;
  long horizon[T::Command::MAX];
  map<int, T::Status> leaf_status; // for bank only
  // functions (recursively traverses down tree)
  T::Command decode(T::Command cmd, int addr[]);
  bool check(T::Command cmd, int addr[], long now);
  void update(T::Command cmd, int addr[], long now);
};

// DDR3.h/cpp
class DDR3 {
  enum class Status {Open, Closed, ... , MAX};
  enum class Command {ACT, PRE, RD, WR, ... , MAX};
};
```

Currently named ‘next’
DRAM Functions

The memory controller relies on three recursive functions to serve a memory request:

- **decode()**: returns the prerequisite command (e.g., ACT for a closed bank)
- **check()**: returns whether or not the DRAM is ready to accept a given command (i.e., timing violation check)
- **update()**: updates the node state based on the issued command
decode(): Determining the **Prerequisite**

```cpp
// DRAM.h
template <typename T>
class DRAM {
    T::Command decode(T::Command cmd, int addr[]) {
        if (prereq[level][cmd]) {
            // consult lookup-table to decode command
            T::Command p = prereq[level][cmd](this);
            if (p != T::Command::MAX)
                return p; // decoded successfully
        }
        if (children.size() == 0) // lowest-level
            return cmd; // decoded successfully
        // use addr[] to identify target child...
        // invoke decode() at the target child...
    }
};

// DDR3.h/cpp
class DDR3 {
    // declare 2D lookup-table of lambdas
    function<Command(DRAM<DDR3>*)> prereq[Level::MAX][Command::MAX];
    // populate an entry in the table
    prereq[Level::Rank][Command::REF] = [] (DRAM<DDR3>* node) -> Command {
        for (auto bank : node->children)
            if (bank->status == Status::Open)
                return Command::PREA;
        return Command::REF;
    };
    // populate other entries...
};
```
check(): Satisfying the DRAM Timing

- Verifies whether \( \text{next}[\text{cmd}] \leq \text{now} \) for every node affected by \text{cmd}
update(): Transitioning

Defined in DDR3.cpp

```cpp
template <typename T>
void DRAM<T>::update(typename T::Command cmd, const int* addr, long clk)
{
    cur_clk = clk;
    update_state(cmd, addr);
    update_timing(cmd, addr, clk);
}
```

Defined in DDR3.cpp

```cpp
template <typename T>
void DRAM<T>::update_state(typename T::Command cmd, const int* addr)
{
    int child_id = addr[int(level)+1];
    if (lambda[int(cmd)])
        lambda[int(cmd)](his, child_id); // update this level
    if (level == spec->scope[int(cmd)] || !children.size())
        return; // stop recursion: updated all levels
    // recursively update my child
    children[child_id]->update_state(cmd, addr);
}
```

```cpp
void DRAM<T>::update_state_(typename T::Command cmd, const int* addr)
{
    // I am not a target node: I am merely one of its siblings
    if (id != addr[int(level)]) {
        for (auto& t : timing[int(cmd)])
            if (t.lt.sibling) // not an applicable timing parameter
                continue;
        assert (t.dist == 1);
        long future = clk + t.val;
        next[int(t.cmd)] = max(next[int(t.cmd)], future); // update future
    }
    return; // stop recursion: only target nodes should be recursed
}
```
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-pim
    - ZSim+Ramulator: a framework for design space exploration of general-purpose *Processing-in-Memory (PIM)* architectures
Conclusion

- Provides out-of-the-box support for many DRAM standards:
  - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, HMC, and academic proposals (SALP, AL-DRAM, TLDREAM, RowClone, and SARP)
  - Models timing of non-volatile memories (PCM, STT-MRAM)
- Supports multiple scheduling and row buffer management policies
- Modular and extensible to different standards
- Can be paired with other simulators, e.g., gem5 and DRAMPower
- Written in C++11
- ~2.5X faster than fastest open-source simulator

<table>
<thead>
<tr>
<th>Simulator (clang -O3)</th>
<th>Cycles (10^6)</th>
<th>Runtime (sec.)</th>
<th>Reg/sec (10^8)</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>NVMain</td>
<td>666</td>
<td>413</td>
<td>6,881</td>
<td>5,023</td>
</tr>
</tbody>
</table>
P&S Ramulator
Designing and Evaluating Memory Systems and Modern Software Workloads with Ramulator

Hasan Hassan
Prof. Onur Mutlu
ETH Zürich
Spring 2022
9 March 2022