Readings for Next Week

- Combinational Logic chapters from both books
  - Patt and Patel, Chapter 3
  - Harris and Harris, Chapter 2

- Check course website for all future readings
  - Required
  - Recommended
  - Mentioned
How Do Problems Get Solved by Electrons?
Recall: The Transformation Hierarchy

<table>
<thead>
<tr>
<th>Problem</th>
<th>Computer Architecture (expanded view)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>SW/HW Interface</td>
</tr>
<tr>
<td>Program/Language</td>
<td>Micro-architecture</td>
</tr>
<tr>
<td>System Software</td>
<td>Logic</td>
</tr>
<tr>
<td>SW/HW Interface</td>
<td>Devices</td>
</tr>
<tr>
<td>Micro-architecture</td>
<td>Electrons</td>
</tr>
</tbody>
</table>

Computer Architecture (narrow view)
Two goals of this course (especially the second half) are

- to understand how a processor works underneath the software layer and how decisions made in hardware affect the software/programmer
- to enable you to be comfortable in making design and optimization decisions that cross the boundaries of different layers and system components
Some Example “Mysteries”, Continued
Four Mysteries: Familiar with Any?

- Meltdown & Spectre (2017-2018)
- Rowhammer (2012-2014)
- Memory Performance Attacks (2006-2007)
- Memories Forget: Refresh (2011-2012)
Mystery No Longer!

Source: J. Masters, Redhat, FOSDEM 2018 keynote talk.
Mystery No Longer!

It’s like breaking into an apartment by repeatedly slamming a neighbor’s door until the vibrations open the door you were after.
Recall: Takeaway

Breaking the abstraction layers (between components and transformation hierarchy levels)

and knowing what is underneath enables you to understand and solve problems
Future of Memory Security & Reliability

- Onur Mutlu,
  "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser"
  [Slides (pptx) (pdf)]

The RowHammer Problem
and Other Issues We May Face as Memory Becomes Denser

Onur Mutlu
ETH Zürich
onur.mutlu@inf.ethz.ch
https://people.inf.ethz.ch/omutlu

Another Example “Mystery”
Mystery #3:
Memory Performance Attacks
Multi-Core Systems

*Die photo credit: AMD Barcelona
A Trend: Many Cores on Chip

- **Simpler and lower power** than a single large core
- Parallel processing on single chip $\rightarrow$ faster, new applications

- **AMD Barcelona**
  - 4 cores
- **Intel Core i7**
  - 8 cores
- **IBM Cell BE**
  - 8+1 cores
- **IBM POWER7**
  - 8 cores
- **Sun Niagara II**
  - 8 cores
- **Nvidia Fermi**
  - 448 “cores”
- **Intel SCC**
  - 48 cores, networked
- **Tilera TILE Gx**
  - 100 cores, networked
Many Cores on Chip

- What we want:
  - N times the system performance with N times the cores

- What do we get today?
Unexpected Slowdowns in Multi-Core

Three Questions

- Can you figure out why the applications slow down if you do not know the underlying system and how it works?

- Can you figure out why there is a disparity in slowdowns if you do not know how the system executes the programs?

- Can you fix the problem without knowing what is happening “underneath”? 
Three Questions

- Why is there any slowdown?

- Why is there a disparity in slowdowns?

- How can we solve the problem if we do not want that disparity?
  - What do we want (the system to provide)?
Why Is This Important?

- We want to execute applications in parallel in multi-core systems → consolidate more and more
  - Cloud computing
  - Mobile phones

- We want to mix different types of applications together
  - those requiring QoS guarantees (e.g., video, pedestrian detection)
  - those that are important but less so
  - those that are less important

- We want the system to be controllable and high performance
Why the Disparity in Slowdowns?

CORE 1

L2 CACHE

CORE 2

L2 CACHE

INTERCONNECT

DRAM MEMORY CONTROLLER

DRAM Bank 0

DRAM Bank 1

DRAM Bank 2

DRAM Bank 3

Multi-Core Chip

Shared DRAM Memory System
Why the Disparity in Slowdowns?

Multi-Core Chip

Shared DRAM Memory System

unfairness

DRAM MEMORY CONTROLLER

DRAM Bank 0

DRAM Bank 1

DRAM Bank 2

DRAM Bank 3

L2 CACHE

L2 CACHE

INTERCONNECT

Why the Disparity in Slowdowns?

Why the Disparity in Slowdowns?
Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

This view of a bank is an abstraction.

Internally, a bank consists of many cells (transistors & capacitors) and other structures that enable access to cells.
DRAM Controllers

- A row-conflict memory access takes significantly longer than a row-hit access

- Current controllers take advantage of this fact

- Commonly used scheduling policy (FR-FCFS) [Rixner 2000]*
  1. Row-hit first: Service row-hit memory accesses first
  2. Oldest-first: Then service older accesses first

- This scheduling policy aims to maximize DRAM throughput

The Problem

- Multiple applications share the DRAM controller
- DRAM controllers designed to maximize DRAM data throughput

- DRAM scheduling policies are unfair to some applications
  - Row-hit first: unfairly prioritizes apps with high row buffer locality
    - Threads that keep on accessing the same row
  - Oldest-first: unfairly prioritizes memory-intensive applications

- DRAM controller vulnerable to denial of service attacks
  - Can write programs to exploit unfairness
A Memory Performance Hog

// initialize large arrays A, B
for (j=0; j<N; j++) {
  index = rand();  \textcolor{red}{\textbf{random}}
  A[index] = B[index]; \textcolor{blue}{\textbf{streaming}} \textcolor{blue}{\textbf{(in sequence)}}
  ...
}

\textbf{STREAM}
- Sequential memory access
- Very high row buffer locality (96\% hit rate)
- Memory intensive

\textbf{RANDOM}
- Random memory access
- Very low row buffer locality (3\% hit rate)
- Similarly memory intensive

What Does the Memory Hog Do?

| T0: Row 0 | T0: Row 6 | T0: Row101 | T0: Row106 |

Memory Request Buffer

Row size: 8KB, request size: 64B

128 (8KB/64B) requests of STREAM serviced before a single request of RANDOM

Now That We Know What Happens Underneath

- How would you solve the problem?

- What is the right place to solve the problem?
  - Programmer?
  - System software?
  - Compiler?
  - Hardware (Memory controller)?
  - Hardware (DRAM)?
  - Circuits?

- Two other goals of this course:
  - Enable you to think critically
  - Enable you to think broadly
For the Really Interested...


Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems

Thomas Moscibroda  Onur Mutlu
Microsoft Research
{moscitho, onur}@microsoft.com
Really Interested? ... Further Readings


- Onur Mutlu and Thomas Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems". Proceedings of the 35th International Symposium on Computer Architecture (ISCA) [Slides (ppt)].

Takeaway

Breaking the abstraction layers (between components and transformation hierarchy levels)

and knowing what is underneath enables you to understand and solve problems
Another Example “Mystery”
Mystery #4: DRAM Refresh
DRAM in the System

*Die photo credit: AMD Barcelona*
A DRAM cell consists of a capacitor and an access transistor.

It stores data in terms of charge status of the capacitor.

A DRAM chip consists of (10s of 1000s of) rows of such cells.
DRAM Refresh

- DRAM capacitor charge leaks over time

- The memory controller needs to refresh each row periodically to restore charge
  - Activate 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**
First, Some Analysis

- Imagine a system with **1 ExaByte DRAM** (2^60 bytes)
- Assume a row size of **8 KiloBytes** (2^13 bytes)

- How many rows are there?
- How many refreshes happen in 64ms?
- What is the total power consumption of DRAM refresh?
- What is the total energy consumption of DRAM refresh during a day?

- A good exercise... Optional homework...
- Brownie points from me if you do it...
Refresh Overhead: Performance

Refresh Overhead: Energy

How Do We Solve the Problem?

- **Observation:** All DRAM rows are refreshed every 64ms.

- **Critical thinking:** Do we need to refresh all rows every 64ms?

- **What if we knew what happened underneath and exposed that information to upper layers?**
Underneath: Retention Time Profile of DRAM

64-128ms

>256ms

128-256ms

Aside: Why Do We Have Such a Profile?

- Answer: Manufacturing is not perfect
- Not all DRAM cells are exactly the same
- Some are more leaky than others
- This is called **Manufacturing Process Variation**
Opportunity: Taking Advantage of This Profile

- Assume we know the retention time of each row exactly
- What can we do with this information?
- Who do we expose this information to?
- How much information do we expose?
  - Affects hardware/software overhead, power, verification complexity, cost
- How do we determine this profile information?
  - Also, who determines it?
Observation: Overwhelming majority of DRAM rows can be refreshed much less often without losing data.

Key Idea of RAIDR: Refresh weak rows more frequently, all other rows less frequently.

RAIDR: Eliminating Unnecessary DRAM Refreshes

Liu, Jaiyen, Veras, Mutlu, RAIDR: Retention-Aware Intelligent DRAM Refresh ISCA 2012.
RAIDR: Mechanism

1. Profiling: Identify the retention time of all DRAM rows

64-128ms

>256ms

1.25KB storage in controller for 32GB DRAM memory

→ check the bins to determine refresh rate of a row

RAIDR: Results and Takeaways

- System: 32GB DRAM, 8-core; Various 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
Reading for the Really Interested


RAIDR: Retention-Aware Intelligent DRAM Refresh

Jamie Liu    Ben Jaiyen    Richard Veras    Onur Mutlu
Carnegie Mellon University
{jamiel,bjaiyen,rveras,onur}@cmu.edu
Really Interested? … Further Readings

- Onur Mutlu,
  "Memory Scaling: A Systems Architecture Perspective"
  Technical talk at MemCon 2013 (MEMCON), Santa Clara, CA, August 2013.
  Slides (pptx) (pdf) Video

- Kevin Chang, Donghyuk Lee, Zeshan Chishti, Alaa Alameldeen, Chris Wilkerson, Yoongu Kim, and Onur Mutlu,
  "Improving DRAM Performance by Parallelizing Refreshes with Accesses"
Takeaway I

Breaking the abstraction layers (between components and transformation hierarchy levels) and knowing what is underneath enables you to understand and solve problems.
Cooperation between multiple components and layers can enable more effective solutions and systems.
Bloom Filters
Approximate Set Membership

- Suppose you want to quickly find out:
  - whether an element belongs to a set

- And, you can tolerate mistakes of the sort:
  - The element is actually not in the set, but you are incorrectly told that it is → false positive

- But, you cannot tolerate mistakes of the sort:
  - The element is actually in the set, but you are incorrectly told that it is not → false negative

- Example task: You want to quickly identify all Mobile Phone Model X owners among all possible people in the world
  - Perhaps you want to give them free replacement phones
Example Task

- World population
  - ~8 billion (and growing)
  - 1 bit per person to indicate Model X owner or not
  - $2^{33}$ bits needed to represent the entire set accurately
    - 8 Gigabits $\rightarrow$ large storage cost, slow access

- Mobile Phone Model X owner population
  - Say 1 million (and growing)

- Can we represent the Model X owner set approximately, using a much smaller number of bits?
  - Record the ID’s of owners in a much smaller Bloom Filter
Example Task II

- **DRAM row population**
  - ~8 billion (and growing)
  - 1 bit per row to indicate Refresh-often or not
  - \(2^{33}\) bits needed to represent the entire set accurately
    - 8 Gigabits → large storage cost, slow access

- **Refresh-often population**
  - Say 1 million

- Can we represent Refresh-often set approximately, using a much smaller number of bits?
  - **Record the ID’s of Refresh-often rows in a much smaller Bloom Filter**
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 other elements also map to

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

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 0
```

Hash function 1

Hash function 2

Hash function 3

Insert Row 1

Bloom Filter Operation Example

Example with 64-128ms bin:

1 & 1 & 1 = 1
0 0 1 1 0 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:

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

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:

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Hash function 1
Hash function 2
Hash function 3

Row 5 present? Yes (false positive)
Bloom Filters

Space/Time Trade-offs in Hash Coding with Allowable Errors

In such applications, it is envisaged that overall performance could be improved by using a smaller core resident hash area in conjunction with the new methods and, when necessary, by using some secondary and perhaps time-consuming test to “catch” the small fraction of errors associated with the new methods. An example is discussed which illustrates possible areas of application for the new methods.

BURTON H. BLOOM

In this paper trade-offs among certain computational factors in hash coding are analyzed. The paradigm problem considered is that of testing a series of messages one-by-one for membership in a given set of messages. Two new hash-coding methods are examined and compared with a particular conventional hash-coding method. The computational factors considered are the size of the hash area (space), the time required to identify a message as a nonmember of the given set (reject time), and an allowable error frequency.

Bloom Filters: Pros and Cons

- Advantages
  + Enables **storage-efficient** representation of set membership
  + Insertion and testing for set membership (presence) are **fast**
  + **No false negatives**: If Bloom Filter says an element is not present in the set, the element must not have been inserted
  + Enables **tradeoffs** between **time & storage efficiency & false positive rate** (via sizing and hashing)

- Disadvantages
  -- **False positives**: An element may be deemed to be present in the set by the Bloom Filter but it may never have been inserted
    Not the right data structure when you cannot tolerate false positives

Benefits of Bloom Filters as Refresh Rate 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
Recap: Four Mysteries

- Meltdown & Spectre (2017-2018)

- Rowhammer (2012-2014)

- Memory Performance Attacks (2006-2007)

- Memories Forget: Refresh (2011-2012)
Takeaways
Some Takeaways

- It is an exciting time to be understanding and designing computing platforms

- Many challenging and exciting problems in platform design
  - That noone has tackled (or thought about) before
  - That can have huge impact on the world’s future

- Driven by huge hunger for data and its analysis ("Big Data"), new applications, ever-greater realism, ...
  - We can easily collect more data than we can analyze/understand

- Driven by significant difficulties in keeping up with that hunger at the technology layer
  - Three walls: Energy, reliability, complexity
Design of Digital Circuits
Lecture 4: Mysteries in Comp Arch and Basics

Prof. Onur Mutlu
ETH Zurich
Spring 2018
2 March 2018