comp.arch Notes Documentation

Alper YAZAR and CONTRIBUTORS

May 04, 2019

Contents

Welcome to notes on computer architecture![1]

You are on the home page of set of public notes. This is a collection of mostly personal notes. Notes are (will be) mixture of:

- Lecture notes on courses offered in universities or other institutions.
- Processor history
- Paradigms like reconfigurable computing

but **NOT** about:

- Programming
- VHDL/Verilog
- Details of FPGA based designs

My main purpose is to collect my notes in one place. I prefer to share them publicly because I believe that this will lead creation of more up-to-date and more correct resource. Therefore, I would like to hear your feedback, your comments and corrections. If you have anything to say, you can contact_page. Moreover, it would be great if you contribute by creating new pages or fixing errors. Just check contributing_page page.

Lastly, I would like to kindly remind that all materials (including text) are licensed and copyrighted. License is **CC-BY-SA-4.0** if otherwise stated. You are free to share and adapt materials even for commercial purposes if you follow license conditions. Just check license_page and copyright_page page. As stated in Section 5 of CC-BY-SA-4.0 legal code, all materials (independent from chosen license) are provided **AS-IS** and **NO WARRANTY**. Check disclaimer_page page if you need.

I hope you enjoy.

Sincerely,

Alper YAZAR

[1] **comp.arch** is a Usenet group on computer architecture. I choose this name as the short code.

CHAPTER 1

Content

1.1 Lecture Notes

Notes on some computer architecture related lectures are listed here.

1.1.1 Design of Digital Circuits - Spring 2018

Table 1: Course Information

Name	Design of Digital Circuits
Official Web Page	Design of Digital Circuits Spring 2018 (252-0028-00L)
University	ETH Zürich
Instructor	Prof. Onur MUTLU

Books

- [?]
- [?]

Lectures

Lecture 01 - Introduction and Basics

Info

- Video Link
- Lecture Slides

Reading

- [**?**]
- Chapter 1 in [**?**]
- Chapter 1-2 in [**?**]
- Binary Numbers (*Done*)
- [**?**] (*Done*)
- [**?**]
- [**?**]

Reading Notes

Binary Numbers

Basic stuff..

Todo: Expand notes on number representation, possibly on separate page.

Lecture Minutes

- **29:00** Importance of principled design
- **40:00** "architecture based upon principle not upon precedent" Precedent means examples from past. Design should have some principles and shouldn't build blindly on past or wide spread approaches. For example, let's consider Spectre and Meltdown. In those cases, precedent is building a fast processor. But designer might have considered security principle to avoid this kind of flaws.
- **44:00** Although technology changes over time, fundamental techniques and principles may remain similar.
- **48:00** Mini talk about *systolic array* although it is kind of out of context.
- **01:03:00** Electrons → Transistors → Logic Gates → Combinational Logic Circuits → Sequential Logic Circuits (Storage Elements and Memory) → … → Cores → Caches → Interconnect → Memories → …
- **01:13:00** The transformation hierarchy is shown below. How can we talk with electrons? This is a fundamental question.
- **01:15:00** *algorithm* is defined.

  See *ISA* and *microarchitecture* definitions.
- **01:24:00** VAX architecture, 3D array checking at hardware (?)

  Todo: What are they? 3D array checking?

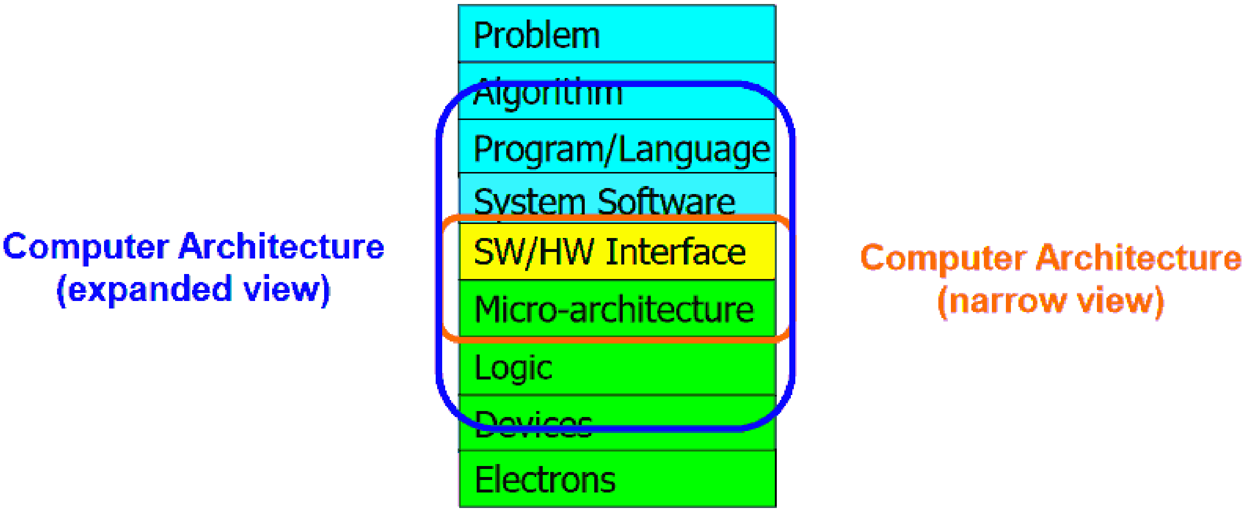

Fig. 1: *Taken directly from the lecture notes. © belongs to Prof. Mutlu*

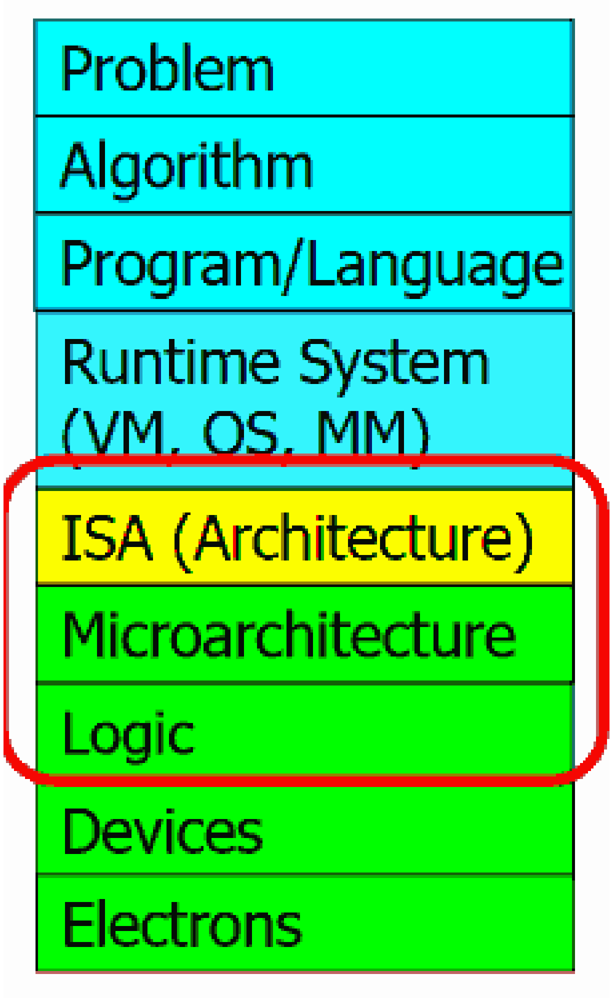

Fig. 2: *Taken directly from the lecture notes. © belongs to Prof. Mutlu*

Glossary

algorithm Algorithm is step-by-step procedure that is guaranteed to terminate where each step is precisely stated and can be carried out by a computer

ISA **Instruction Set Architecture**. It is a contract between Hardware and Software. It is programmers assumption about hardware.

microarchitecture An implementation of the *ISA*.

systolic array It looks like a type of parallel processing architecture. It is used by Google Tensor. It is a memory bandwidth efficient architecture. https://www.geeksforgeeks.org/parallel-processing-systolic-arrays/

> **Todo:** I may not have fully understood.

Section author: Alper Yazar

Lecture 02 - Mysteries in Comp Arch

Info

- Video Link
- Lecture Slides

Reading

- [?]
- [?]
- [?]
- [?]
- [?]
- [?]
- [?]
- [?]
- [?]

Reading Notes

Lecture Minutes

- **05:17** User-centric view of “transformations”. Entire stack should be optimized for the user. But there are many users with different expectations from a computer. Also some users write OS and some of them write algorithms. User also interacts with computer with different levels.

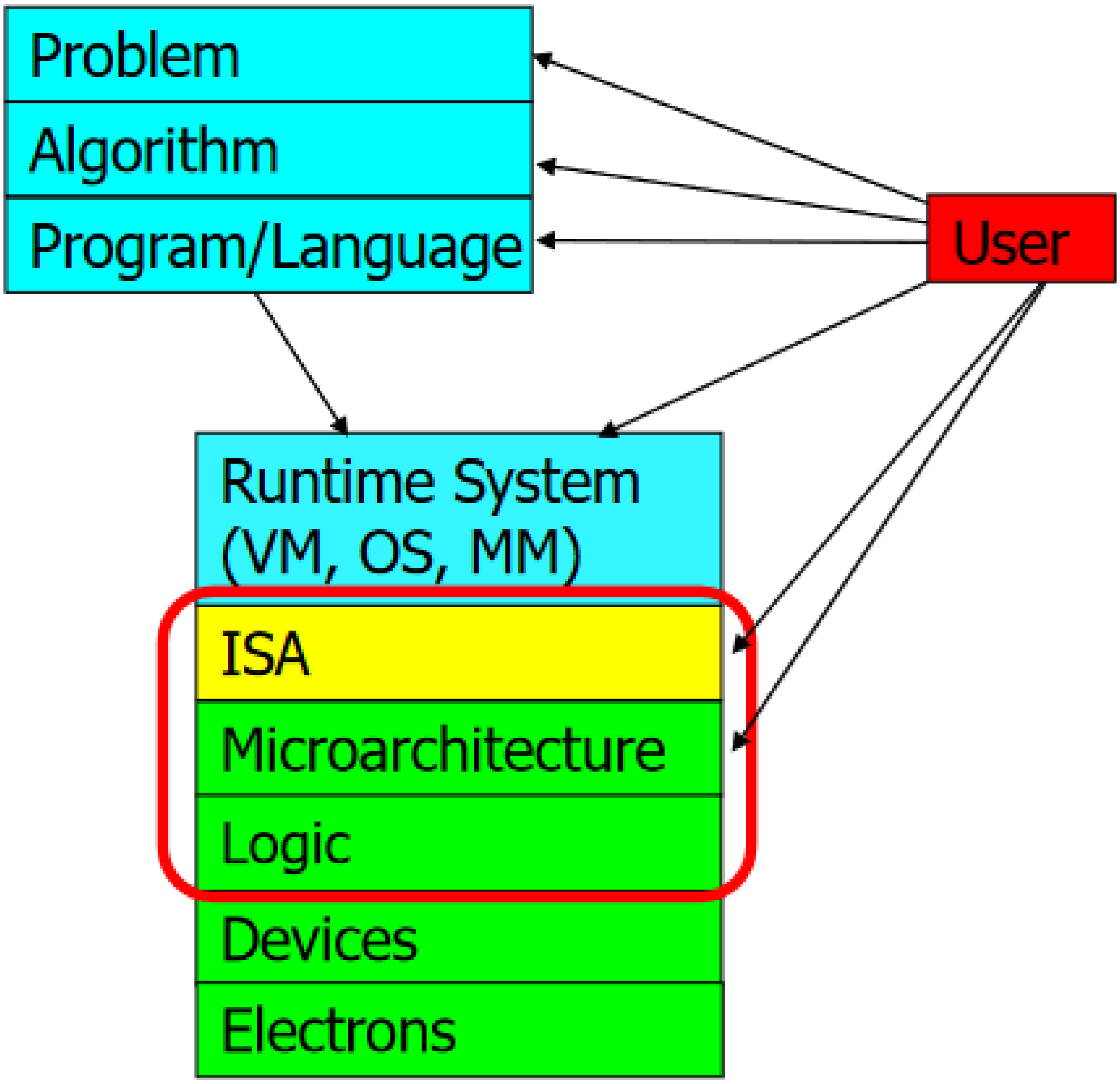

Fig. 3: *Taken directly from the lecture notes. © belongs to Prof. Mutlu*

- **07:15** *abstraction* is defined. Programmer doesn't need to know details of processor because it is abstracted by many levels. Abstraction improves productivity. Then why worry about details?
- **13:00** If you know details of a processor, you can write better programs. As long as everything is OK, no need to dive in details. But what if program runs slow? not correct results? energy hungary? security? efficiency? From hardware designer perspective, if designer doesn't know what a programmer expects from a processor he/she design a processor which is very hard to program. Designer needs to know software perspective to implement right primitives.
- **15:00** As an example, **IBM Cell Broadband Engine** was the first heterogenous multi-core processor. Beautiful design but missing cache coherency in hardware. So, it was very hard to program.
- **16:40** Another example: **Transmeta**. They tried to compete with Intel. They wanted to design x86 processor. They wrote a software translation layer which takes x86 code and translates to a code for the internal microarchitecture (*VLIW* type) that is simpler than x86. Why? Because implementing an efficient x86 ISA microarchitecture isn't a trivial thing especially for a small, startup company. They chose to design very fast internal microarchitecture but they needed an efficient translation layer. Notice that translation layer is a software. But the bottleneck was that layer. In the end, they failed racing. Read [**?**] and [**?**].
- **23:10** Course goals: 1) How processor works underneath. 2) Making design and optimizations.
- **24:10** Mysteries
- **25:10 Meltdown** and **Spectre**. Someone can steal secret data even though no software bugs, hardware behaves according to specifications. But why this happens? These are hardware security vulnerabilities that affects almost all modern processors. *speculative execution* is the main cause (I mean main "trigger"). You predict code and do something. If you realize that prediction was wrong, you undo. For example you may prefer it in an *if* statement if accessing the control variable takes many cycles. You assume a result, go on then when access is completed you check your assumption. But processor operates this mechanism such that it obeys programmers assumption about processor like sequential execution and ISA. Programmers aren't interested in microarchitecture. In these flaws, processor leaks data due to this mechanism. With help of caching, attacker may guess processor state about speculative execution. Processor brings data which should not be accessed if there is no speculative execution into cache due to speculative execution. Attacker can inspect the contents of the cache to "sniff" secret data (by measuring how long it takes to access the data). It is also possible to execute another program by abusing flaws. They are sophisticated attacks.
- **38:30** Processor cache is a *side channel*.
- **41:30** *branch prediction* is mentioned.
- **42:45** Read: [**?**]
- **43:00** If you don't "cross layers", you can't detect and fix the problem in case of Meltdown and Spectre. You can't create an another attack.
- **45:45**
- **46:30**
- **48:30** Design goal of a system determines the design mindset and evaluation metrics. Design goal of cutting edge processors is a reason of Meltdown and Spectre. But security is not main design goal.
- **50:30** Other course goals: Think Critically, Think Broadly.
- **51:10** Videos about Meltdown and Spectra: https://www.youtube.com/watch?v=syAdX44pokE https://www.youtube.com/watch?v=mgAN4w7LH2o
- **52:50 RowHammer** DRAM Disturbance Error. It is a story of how a simple hardware failure mechanism can create a widespread system security vulnerability. Reading a row in DRAM disturb adjacent rows. It is also

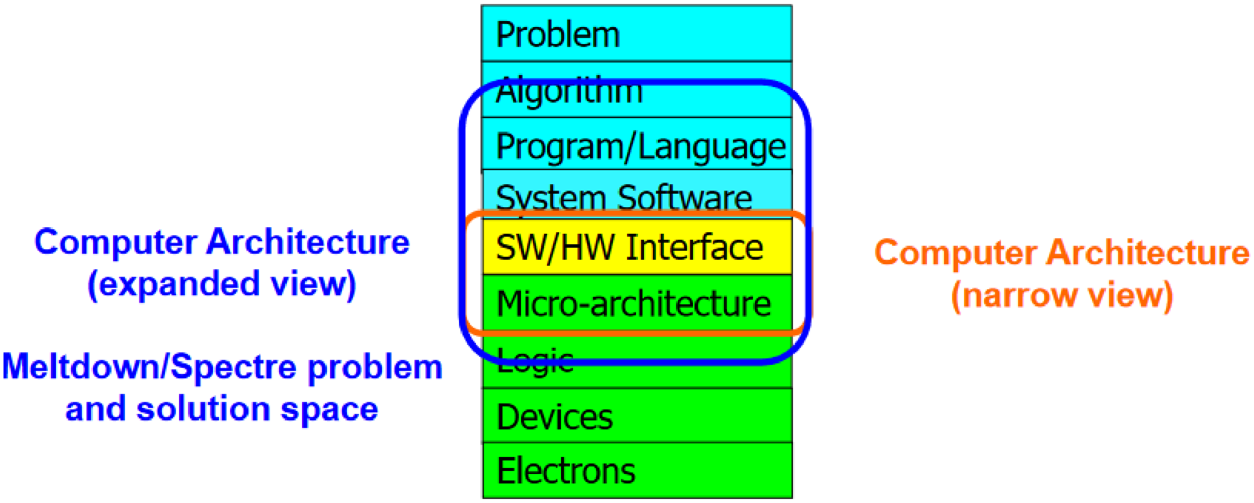
Problem
Algorithm
Program/Language
System Software
SW/HW Interface
Micro-architecture
Logic
Devices
Electrons
Computer Architecture (expanded view)
Meltdown/Spectre problem and solution space
Computer Architecture (narrow view)

Fig. 4: *Taken directly from the lecture notes. © belongs to Prof. Mutlu*

possible in SSDs or HDDs, etc. When you want to read a row in memory, you apply high voltage to it. After reading, you apply low voltage. If you repeat this in most DRAM chips you get errors in adjacent rows. Notice that corruption is triggered by only reading adjacent rows. You hammer row by reading.

- **57:30** It is interesting that this flaw is observed chips manufactured after ~2010. More recent chips have higher error rates. This is because physical decrease in distance between rows, similar to Moore's Law.
- **58:15** DRAM cells are too close to each other. Actually they are not completely electrically isolated from each other. Another term is cell-to-cell coupling. Some cells are more vulnerable than other cells. DRAM memory store information as charge. You can drain charge with this attack.
- **1:01:30** Some implementation details about RowHammer, assembly codes.
- **1:03:30** Read [?] and [?]
- **1:04:30** [?] gained root access on a Linux system by exploiting RowHammer.
- **1:09:10** [?]
- **1:09:30** [?] Hammer and root millions of Androids
- **1:10:00** How can we fix? Better DRAM chips (cost), refresh frequently (power, performance), sophisticated error correction (cost, power), access counter (a method of detecting attack but cost, power, complexity)
- **1:14:00** https://support.apple.com/en-gb/HT204934 is Apple solution to problem. They refresh whole memory more frequently but it will increase power consumption.
- **1:15:25 PARA**: Probabilistic Adjacent Row Activation is a cheaper solution to RowHammer problem. Key idea: After closing a row, with some probability (probably low value like 0.005) you refresh its neighbors.
- **1:17:45** RowHammer also shows that one should be able to "cross layers" to understand, fix or repeat the problem.
- **1:19:35** These are example of *byzantine failures*. Read [?]
- **1:24:05** [?]

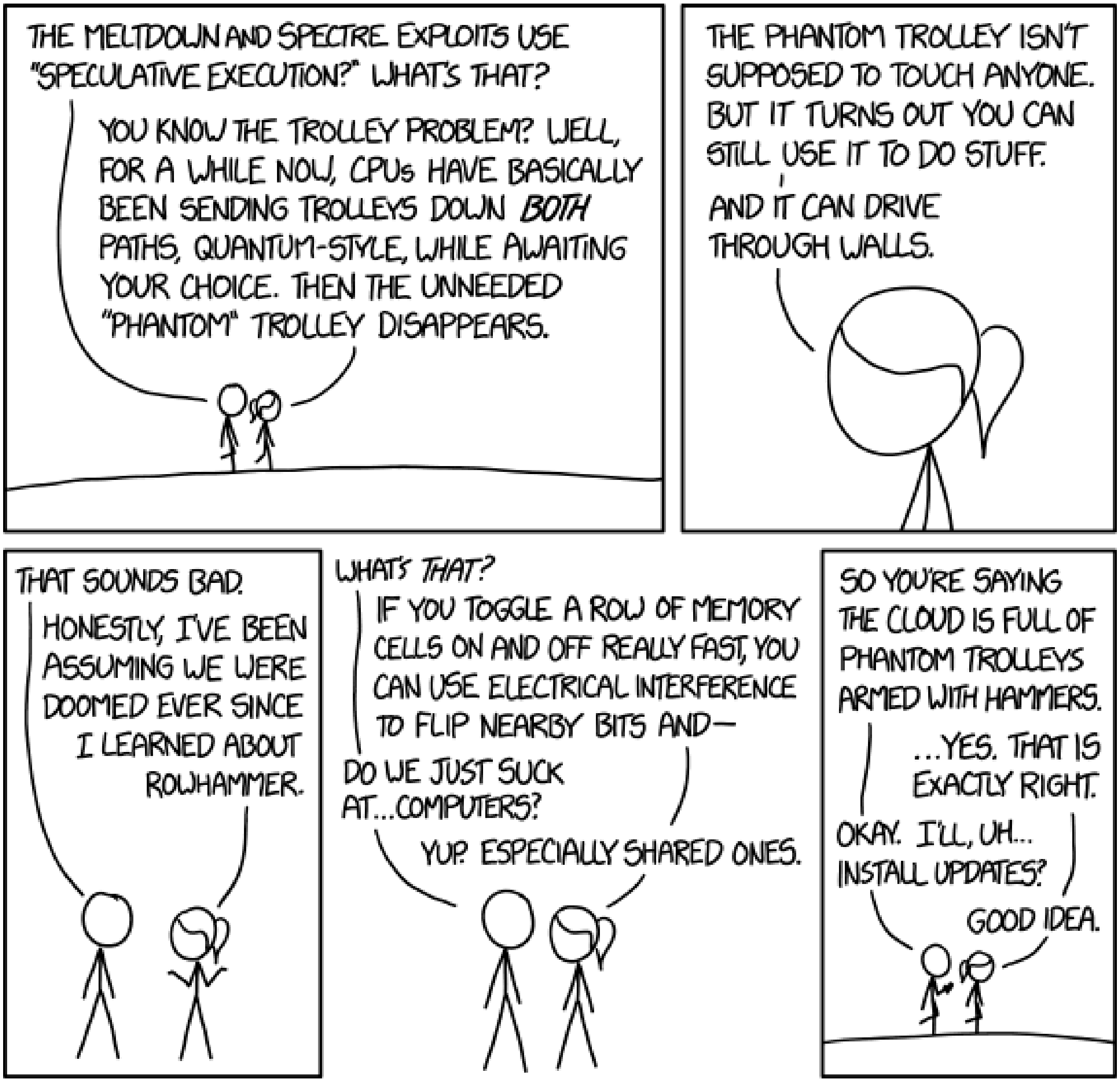
THE MELTDOWN AND SPECTRE EXPLOITS USE "SPECULATIVE EXECUTION?" WHAT'S THAT?
YOU KNOW THE TROLLEY PROBLEM? WELL, FOR A WHILE NOW, CPUs HAVE BASICALLY BEEN SENDING TROLLEYS DOWN *BOTH* PATHS, QUANTUM-STYLE, WHILE AWAITING YOUR CHOICE. THEN THE UNNEEDED "PHANTOM" TROLLEY DISAPPEARS.
THE PHANTOM TROLLEY ISN'T SUPPOSED TO TOUCH ANYONE. BUT IT TURNS OUT YOU CAN STILL USE IT TO DO STUFF.
AND IT CAN DRIVE THROUGH WALLS.
THAT SOUNDS BAD.
HONESTLY, I'VE BEEN ASSUMING WE WERE DOOMED EVER SINCE I LEARNED ABOUT ROWHAMMER.
WHAT'S *THAT?*
IF YOU TOGGLE A ROW OF MEMORY CELLS ON AND OFF REALLY FAST, YOU CAN USE ELECTRICAL INTERFERENCE TO FLIP NEARBY BITS AND—
DO WE JUST SUCK AT...COMPUTERS?
YUP. ESPECIALLY SHARED ONES.
SO YOU'RE SAYING THE CLOUD IS FULL OF PHANTOM TROLLEYS ARMED WITH HAMMERS.
...YES. THAT IS EXACTLY RIGHT.
OKAY. I'LL, UH... INSTALL UPDATES?
GOOD IDEA.

Fig. 5: *https://xkcd.com/1938/ © belongs to XKCD*

Glossary

abstraction A higher level only needs to know about the interface to the lower level, not how the lower level is implemented.

branch prediction A way of "guessing" mentioned in *speculative execution*.

byzantine failures They are characterized by undetected erroneous computation (opposite of fail fast (with an error or no result)). They are "sneaky" errors. "erroneous" can be "malicious" because one can abuse it. They are very difficult to detect and confine. It is a real problem in distributed systems (may be like blockcahin :)). Avoidance is the best solution. [**?**]

side channel A channel through which someone sophisticated can extract information.

speculative execution Doing something before you know what it is needed. If you realize that it is not needed after some time, you revert. In overall, this saves time and improves performance.

VLIW Very Long Instruction World. A type of architecture.

Section author: Alper Yazar

Lecture 03 - Introduction to the Labs and FPGAs

I skip this lecture for now. I will try to complete it if time permits. It is about course labs and basics of FPGAs.

Todo: Try to write this lecture.

Lecture 04 - Mysteries in Comp Arch and Basics

Info

- Video Link
- Lecture Slides

Reading

- Chapter 2 in [**?**]
- Chapter 3 in [**?**]
- [**?**]
- [**?**]
- [**?**]
- [**?**]
- [**?**]
- [**?**]
- [**?**]
- [**?**]

- [?]
- [?] (Couldn't notice during lecture, I added as reference since it is stated on course page)

Reading Notes

Lecture Minutes

- **04:30** Another mystery is **Memory Performance Attacks**
- **10:00** *throughput computing* is mentioned.
- **10:55** We want N time the system performance with N times the cores. (*super linear scaling*)
- **12:00** [?] is mentioned. They run MATLAB and GCC on different cores and measured slow down comparing standalone runs. MATLAB slows down 7% if GCC is running. But GCC slows down 204% if MATLAB runs on the other core. MATLAB somewhat slows down GCC. Even if in OS MATLAB is set to low priority and GCC is set to high priority, results don't change. The problem is not related with OS priorities. MATLAB "hogs" memory. In a fair system, this isn't expected. This is neither fair nor controllable system.
- **17:00** If you don't "cross layers", you can't understand the issue.
- **18:30** This is important because multi-core computing is everywhere: cloud, mobile, etc. Also there are different applications with different QoS requirements: video processing, pedestrian detection, etc.
- **20:00** Although each core has different cache, DRAM memory controller is shared between cores. It looks like memory controller prioritize request of MATLAB.
- **21:30** Deeper DRAM structure. A DRAM bank consists of 2D array of cells. Dimensions: Columns and Rows. There are also other components. To access row, you need to bring it to the row buffer. When you need to bring a row to the row buffer, you activate that row by applying high voltage. After data is in the buffer, you select a column depending on your address input.
- **25:30** If you access Row:0, Column:0 and then access Row:0 Column:1, you don't need the activate Row:0 again because Column:1 is also brought to the buffer when Row:0 is transferred to the buffer for Column:0. This is true for all Columns with the same Row number. If you access Row:1 after some access to Row:0. This takes some time. Because memory first writes back (precharge) content of buffer to Row:0, then opens Row:1.
- **29:40** Memory controller may prioritize memory access with the same row number (row-hit first), then serve older request (oldest-first). This is a commonly used scheduling policy called FR-FCFS. See: [?] The goal is to maximize DRAM throughput. But for multiple programs with different characteristics, this may be bad.
- **31:00** Row-hit first rule unfairly prioritizes apps with high row buffer locality. In other words, applications that keep on accessing the same row (i.e. *streaming application*) have an advantage. Secondly, oldest-first unfairly prioritizes memory intensive applications. Memory intensive applications get higher chance to access memory. Oldest-first looks fair. But suppose that app A generates 1 million requests before app B's only one request. Now app B which is not a memory intensive application has to wait all request of app A. If applications are equal, it may be fair.
- **33:11** DRAM controller is vulnerable to denial of service (DoS) attacks.
- **35:10** An example case for the memory controller. DoS scenario is illustrated.
- **36:30** One potential solution is having multiple row buffers in DRAM but it's cost is high. Also this doesn't solve the fairness problem.
- **37:50** Another potential method for priority assignment is that cores may indicate priorities of requests. How do you trust the core?
- **39:30** At which layer this problem should be solved? Hardware looks more meaningful than software layers.

- **40:20** Optional readings for further: [**?**], [**?**], [**?**]
- **40:40** Another mystery: **DRAM Refresh**
- **41:20** For a modern datacenter, more than 60% of cost is for memory.
- **42:00** A DRAM cell consists of a capacitor and an access transistor. Data is stored in terms of charge. When wordline is activated, capacitor is connected to the bitline. Transistor works as a switch.

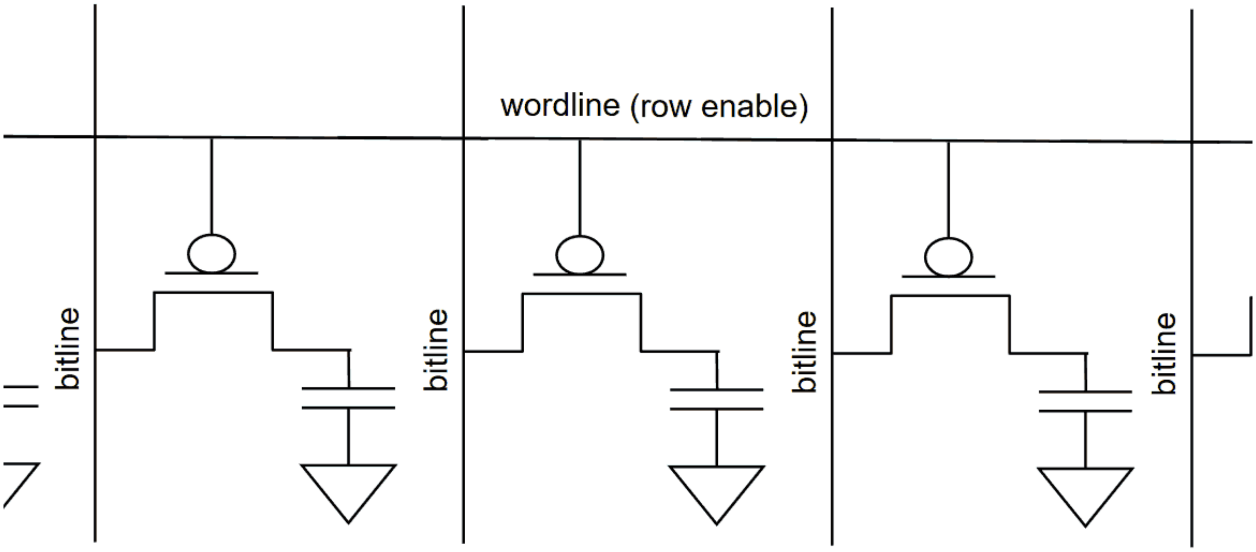

Fig. 6: Cells and a row *Taken directly from the lecture notes. © belongs to Prof. Mutlu*

- **43:21** Capacitor charge leaks over time. Memory controller should refresh each row periodically. Each row is activated (applied high voltage) every N s. A typical N value is 64 ms. Each refresh consumes energy. A DRAM rank/bank is unavailable during refresh. Long pause times during refresh affects QoS and predictability badly. Refres rate limits DRAM capacity scaling.
- **45:20** Consider 1 ExaByte (2^60 bytes) DRAM. Assume row size of 8 KiloBytes (2^13 bytes). There are 2^47 rows. Consider refresh period as 64 ms.

Todo: Complete the exercise. It is also homework.

- **47:30** From [**?**]

Today, max capacity of single DRAM chip is about 8 Gb. Notice that DRAM module consists of several chips. Time spent in refreshing is lost time. It is an overhead. What is refresh granularity? Not row by row but bank by bank. In the past, whole RAM should be suspended for refresh affecting QoS badly. What about increasing the number of cells in a single row, i.e. decreasing number of rows for a fixed size? Does it help to this problem? If you enlarge a row, you affect latency. Ideally, you want achieve a square slice array. Also power is affected badly because even if you need a small group of slice in a row, you have to active whole row. This will lead power inefficiency.

Todo: How latency is affected badly when a row becomes larger? I don't get it. May row buffer wait slowest

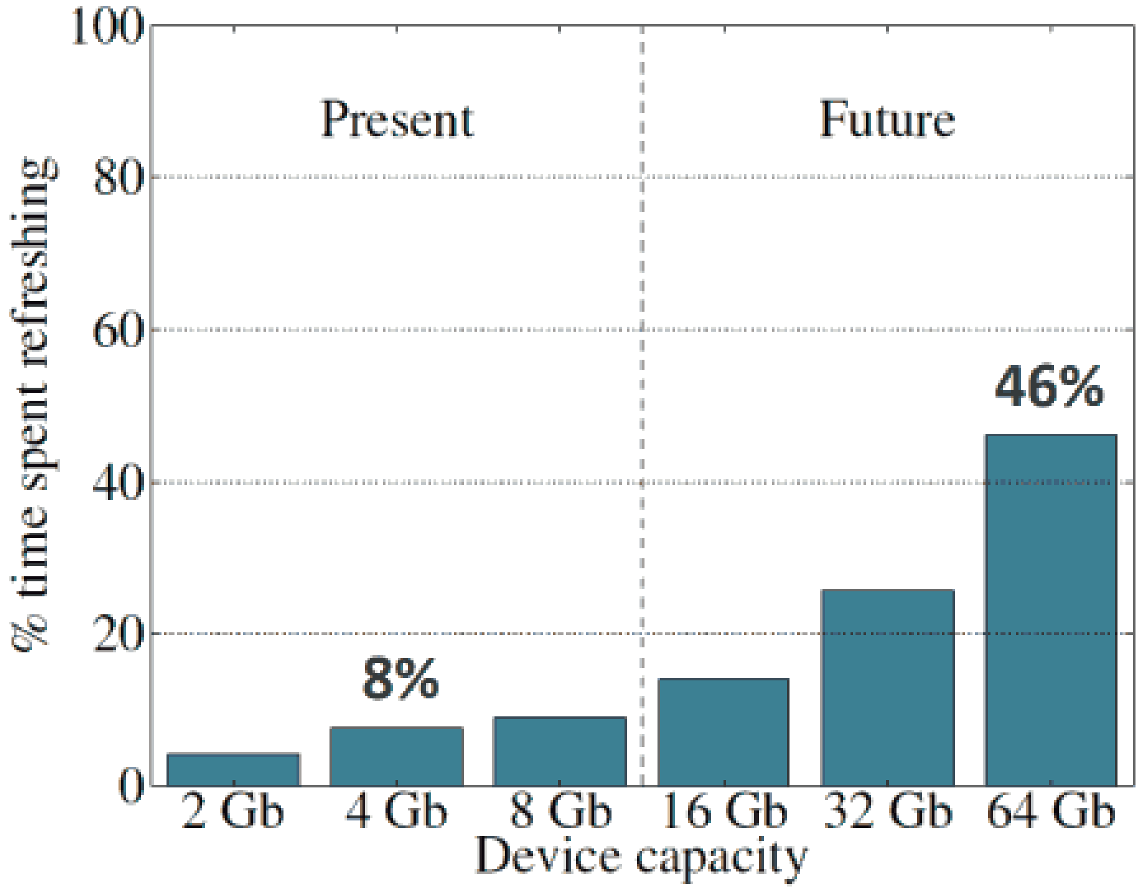

Fig. 7: *Taken directly from the lecture notes. © belongs to Prof. Mutlu*

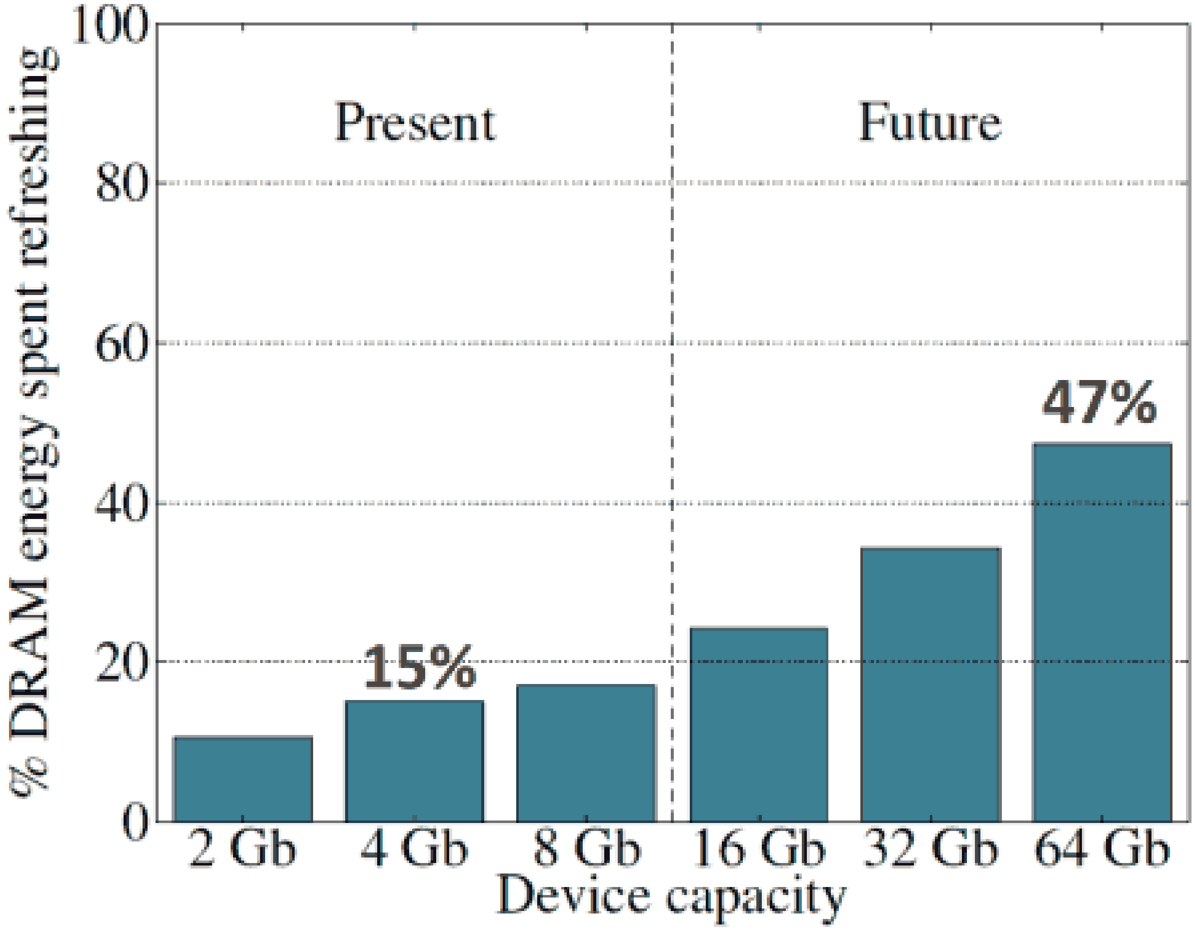

Fig. 8: *Taken directly from the lecture notes. © belongs to Prof. Mutlu*

cell? Like "critical path" concept? All bitlines should be stabilized before reading?

It is clear that there is a scalability problem.

- **51:15** Do we have to refresh every row every 64 ms? What if all memory isn't allocated? Today, this information from OS (page table) doesn't get into the memory controller.
- **53:00** It looks like very small portion of rows should be refreshed at 64 ms. Refresh time of most rows is greater than 256 ms. Why? Because of imperfections during manufacturing. This is *Manufacturing Process Variation* As process' nanometer decreases, imperfections increases. This is also true for processor speed variation between processors.
- **55:30 Cold boot attack** is mentioned as side note.
- **56:30** Assume we know the retention time of each row exactly? How can we use this information? At which layer?
- **1:00:45** We are refreshing all rows at every 64 ms although very small portion needs that much frequent refresh. Most of them can be refreshed at 256 ms. One proposal is **RAIDR**: Refresh ony weak rows more frequently.
- **1:01:00** Read: [**?**] For RAIDR approach 1) you profile DRAM and identify rows. 2) you should store row profile information in an efficient and scalable way (Hint: Bloom Filters). For this case 1.25 KB storage is sufficient for 32GB DRAM memory. 3) refresh accordingly.
- **1:30:10** 74.6/ refresh reduction and some power reduction is achieved.
- **1:05:20** Really interested? [**?**], [**?**]
- **1:05:40** Cooperation between multiple layers leads to more effective system.
- **1:08:08** Talk about *bloom filters* started. *Approximate Set Membership*. You can tolerate false positive (although a member is not in the send, you find that it belongs to the set). You can not tolerate false negative (element in the set but you think that it isn't). You want to control false positive rate but don't want any false negative.
- **1:16:00** Start of Bloom Filter example on board with chalk. Nice example, worth watching.
- **1:28:20** Original idea: [**?**] It is a probabilistic data structure that compactly represents set membership. Notice that solution to RowHammer problem also includes a probabilistic solution.

Glossary

bloom filters Developed late 1960s. It is mentioned last ~20 minutes of the lecture. See the notes.

Todo: I may write a separate page about Bloom filters.

streaming application Application that reaches memory regions with high locality. For example accessing x[i] while i is increasing like i++ is a streaming application.

super linear scaling The case when N cores improves the performance N times

throughput computing

Todo: I don't know yet.

Section author: Alper Yazar

Lecture 05 - Combinational Logic

Info

- Video Link
- Lecture Slides

Reading

- Chapter 2 in [?]
- Chapter 4 until 4.3 and 4.5 in [?]
- Chapter 3 until 3.3 in [?]
- [?]
- [?]
- [?]

Reading Notes

Lecture Minutes

- **02:00** Video assignment: Onur Mutlu - Future Computing Architectures - ETH Zurich Inaugural Lecture (May 15, 2017)
- **04:00** Bloom filter recall, it was mentioned *there*.
- **14:00** Bloom filter advantages and disadvantages. Bloom filters are also used by search engines.
- **16:45** Bloom filters never overflow unlike fixed-size tables. You can insert new elements without increasing storage space. However this will increase false positive rate likely. It is scalable.
- **20:00** Takeaways: This field has probably many challenging and exciting problems that no one has attacked. It is driven by huge hunger for data and its analysis ("Big data"). We can easily collect more data than we can analyze/understand. Field is driven by significant difficulties. Three walls: **energy, reliability, complexity**.
- **22:30** *Dream and they will come*.
- **24:00** A memory access consumes ~1000x the energy of a complex addition. Communication Dominates Arithmetic (Daily, HiPEAC 2015).
- **35:30** Three key components of a computer: computation, communication and storage (memory).
- **39:45** Differences between microprocessors, FPGAs, ASICs.
- **42:30** Introduction to transistors. Today processors have transistors at billion scale.
- **43:40** MOS Transistor: Conductors (Metal), Insulators (Oxide) Semiconductors. It is controlled switch. N and P types are mentioned.
- **51:00** Logic gates.
- **51:34** CMOS technology = nMOS + pMOS. C stands for complementary.
- **52:50** Gender neutrality in Turkish :). "Haklısınız hocam :)"
- **54:00** Operation of a simple CMOS inverter.

- **1:00:11** CMOS NAND gate.
- **1:05:30** NAND gate is "logically complete". You can build any logic just only using NAND gates.

Note: Note that AND gate doesn't have this property. You can't invert using AND. NAND and NOR are functionally complete.

- **1:06:00** CMOS AND gate. AND gate is more complicated than NAND in terms of number of transistors.
- **1:07:00** General CMOS structure has PMOS transistors connected to VCC and the output. It works as pull-up network. NMOS transistors are connected to GND and form a pull-down network.
- **1:08:40** Common logic gates.
- **1:10:45** *Moore's Law* is mentioned.
- **1:13:44** *Dennard scaling* is mentioned.
- **1:15:00** What is the limit on transistor size? We don't know!
- **1:15:45** Heat is a real problem in computer design.
- **1:17:45** *functional specification* and *timing specification* are defined.
- **1:18:15** Combinational logic is memoryless. It doesn't remember anything. Some books call Combinatorial Logic. Sequential Logic has memory.
- **1:19:00** *functional specification* and examples. First example: full 1-bit adder.
- **1:21:50** Simple equations: NOT / AND / OR.
- **1:22:20** [**?**] is mentioned.
- **1:23:07** Axioms of Boolean Algebra
- **1:24:45** Duality in Boolean Algebra. All the axioms come in dual form. This form can be derived by replacing: every AND with OR, every OR with AND, every 1 to 0, every 0 to 1 BUT don't change any literals or play with complements.
- **1:25:50** Some useful Boolean Algebra laws.

Glossary

Dennard scaling From Wikipedia: "Originally formulated for MOSFETs, it states, roughly, that as transistors get smaller, their power density stays constant, so that the power use stays in proportion with area; both voltage and current scale (downward) with length" and "Therefore, in every technology generation the transistor density doubles, the circuit becomes 40% faster, and power consumption (with twice the number of transistors) stays the same." See: [**?**]

functional specification Describes relationship between inputs and outputs

Moore's Law See: [**?**] Moore is one of founders of Intel. He says that component on a chip counts double every other year. Some says that interval is 18 months. Key idea is a fixed interval. Size of transistor is shrinking so we can put more on a chip. Also cost of per circuit varies. We can say that rule was still valid until 2016. (I don't mean that this is invalid since 2016. That was the data shown given on the slides)

timing specification Describes the delay between inputs changing and outputs responding

Section author: Alper Yazar

CHAPTER 2

Pages

- disclaimer_page
- contact_page
- contributing_page
- contributors_page
- copyright_page
- license_page
- todos_page
- Glossary Index
  - Miscellaneous Terms
- bibliography_page
- search

CHAPTER 3

Project Statistics

Index