Course Evaluations

- https://course-evaluations.berkeley.edu
- Fill out by Friday, August 11th
- Extra credit if more than 50% of class fills out
    - The higher the response rate - the more extra credit!

CS61C: Great Ideas in Computer Architecture (aka Machine Structures)

Lecture 25: Summary, What's Next, and Farewell :(

Instructors: Rosalie Fang, Charles Hong, Jero Wang

Announcements

- Project 4 due Tuesday 8/8
- Homework 7 due Wednesday 8/9
- Last deadline for assignments: 8/11
- Almost there!!!

Final Exam Logistics

- Final on Thursday 8/10, 3-6 PM, logistics on Ed
- Tuesday 3:30-5 PM in Pimentel: Review of pre-midterm topics
  - C, FP, RISC-V, CALL, SDS
- Tuesday 5-8 PM in Cory 540AB: Summer 2022 final walkthrough
- Wednesday 2-5 PM in the Woz: Review of post-midterm topics
  - Datapath, Parallelism, Caches, VM
- Charles additional OH this week: Tuesday 2-3 PM (Soda 341B), Wednesday 5-7 PM (Soda 411)

Agenda

- Summary
- What’s Next

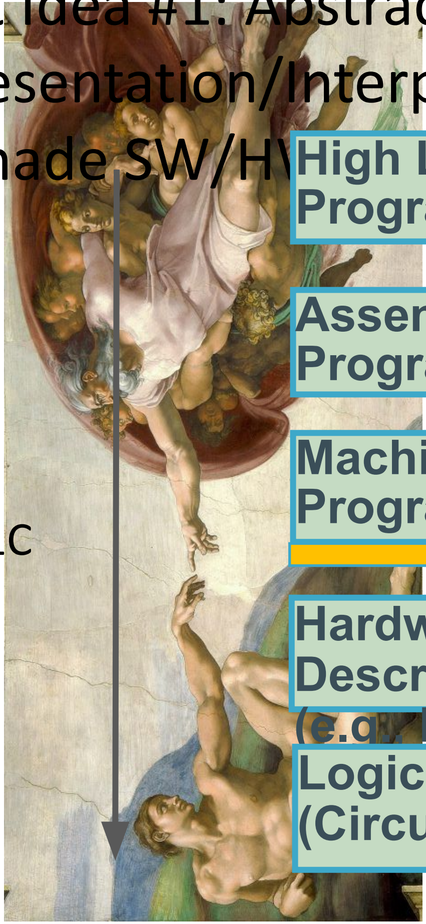
Summary:
What did we learn this summer?

6 Great Ideas of Computer Architecture

1. Abstraction (Layers of Representation/Interpretation)
2. Moore’s Law
3. Principle of Locality/Memory Hierarchy
4. Parallelism
5. Performance Measurement & Improvement
6. Dependability via redundancy

Great Idea #1: Abstraction (Layers of Representation/Interpretation)

We made SW/HW



CS61C

High Level Language Program (e.g., C)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compiler

Assembly Language Program (e.g., RISC-V)

```
lw    x3, 0(x10)
lw    x4, 4(x10)
sw    x4, 0(x10)
sw    x3, 4(x10)
```

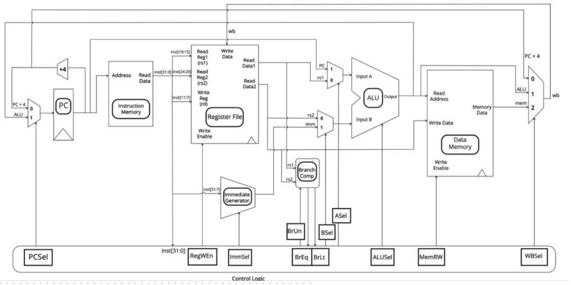
Anything can be represented as a number, i.e., data or instructions

Assembler

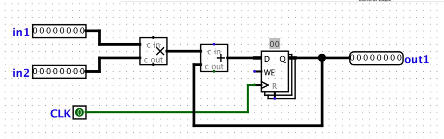
Machine Language Program (RISC-V)

```
1000 1101 1110 0010 0000 0000 0000 0000
1000 1110 0001 0000 0000 0000 0000 0100
1010 1110 0001 0010 0000 0000 0000 0000
1010 1101 1110 0010 0000 0000 0000 0100
```

Hardware Architecture Description (e.g., block diagrams)

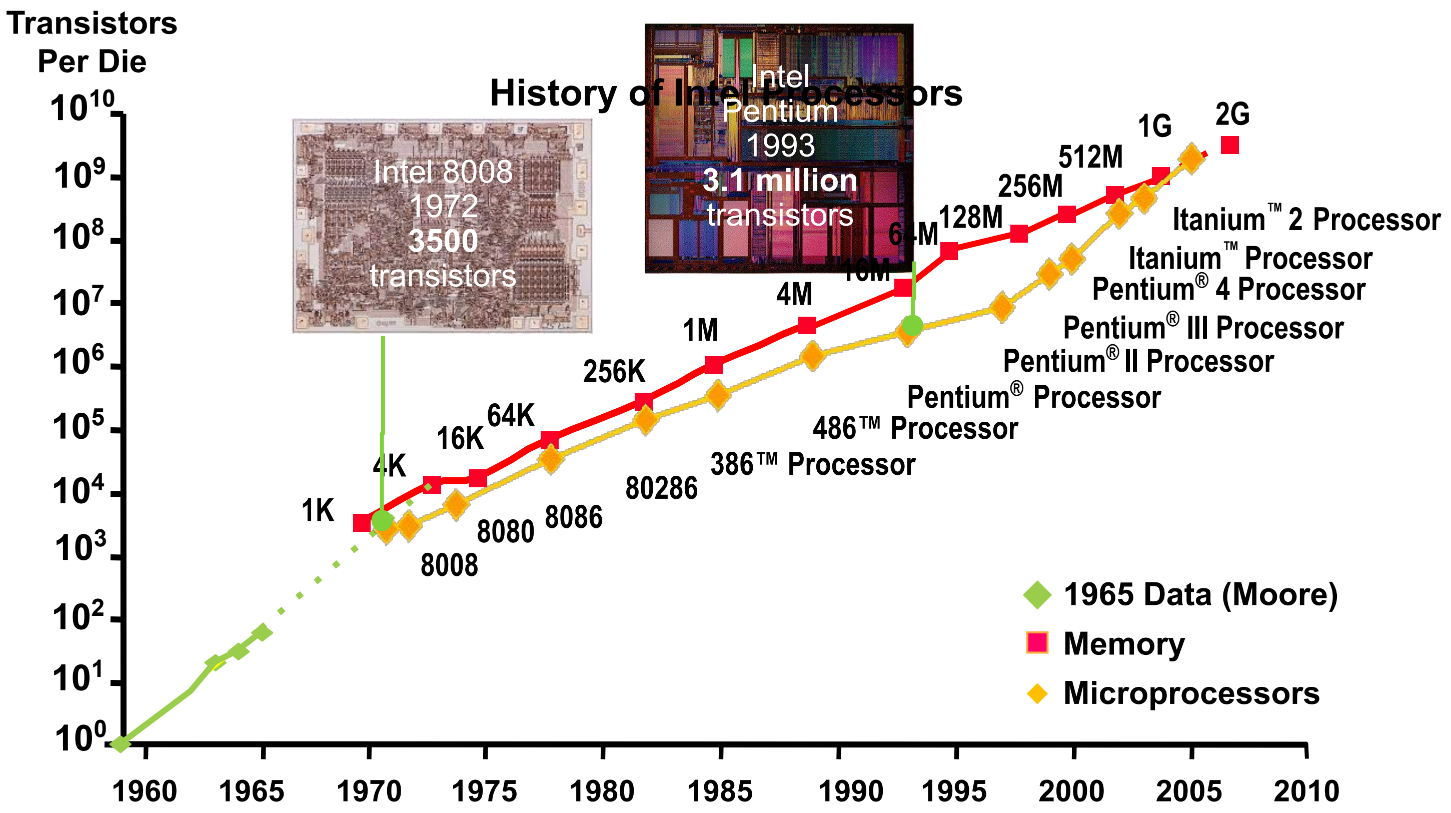


Logic Circuit Description (Circuit Schematic Diagrams)



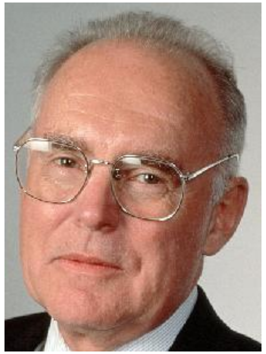
Great Idea #2: Moore's Law

i8008 (1972)
i8080 (1974)
i8086 (1078)
i80286 (1982)
i80386 (386)
i80486 (486)
Pentium (1993)
Pentium II (1997)
Pentium III (1999)
Pentium 4 (2000)
Itanium (2001)
Itanium 2 (2002)



Great Idea #2: Moore's Law

- Moore's law: idea that # of transistors (a base hardware component of integrated circuits) doubles every 2 years
- We briefly touched on how transistors work and are used to build logic gates
  - Mostly out of scope for 61C: learn more in EE 105 (Microelectronic Devices & Circuits), EE 141 (Digital Integrated Circuits), EECS 151 (Digital Design and ICs)
- We then learned how logic gates can be used to build more complex logic, all the way up to a processor datapath
- Unfortunately, slowing down… (?)



Gordon Moore
Intel Cofounder
B.S. Cal 1950!

Great Idea #2: Moore’s Law???

“Moore's Law isn't possible anymore.”

Jensen Huang, CEO Nvidia

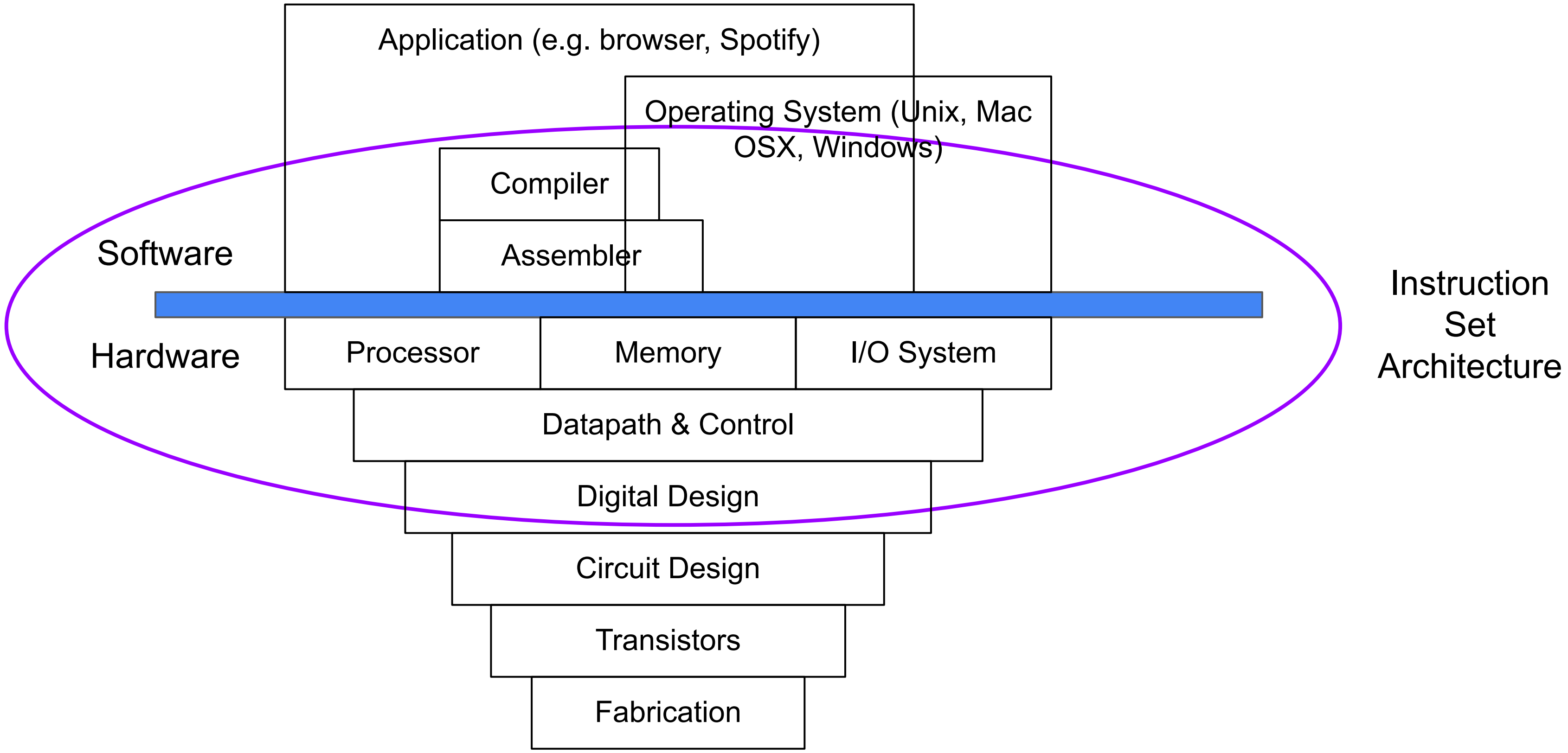
“It’s [Moore’s Law] not dead. It’s not slowing down. It’s not even sick.”

Philip Wong, VP of Research, TSMC

“Today we are predicting that we will maintain or even go faster than Moore's Law for the next decade.”

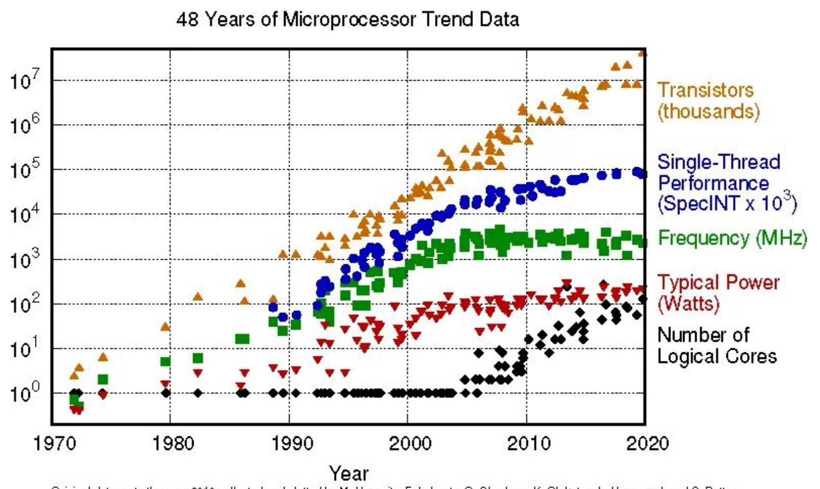
Intel Chief Executive Pat Gelsinger

Old School Machine Structures (still applicable!)



Slowdown of single-threaded performance

- Even if Moore's Law is still alive, scaling single-threaded performance is definitely getting harder
- Breakdown of Dennard scaling: increasing the number of transistors in a given area of chip now increases power utilization 😞



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

New School Machine Structures

Software

Harness Parallelism & Achieve High Performance!!!

Parallel Requests
Assigned to computer
e.g., Search "Cats"

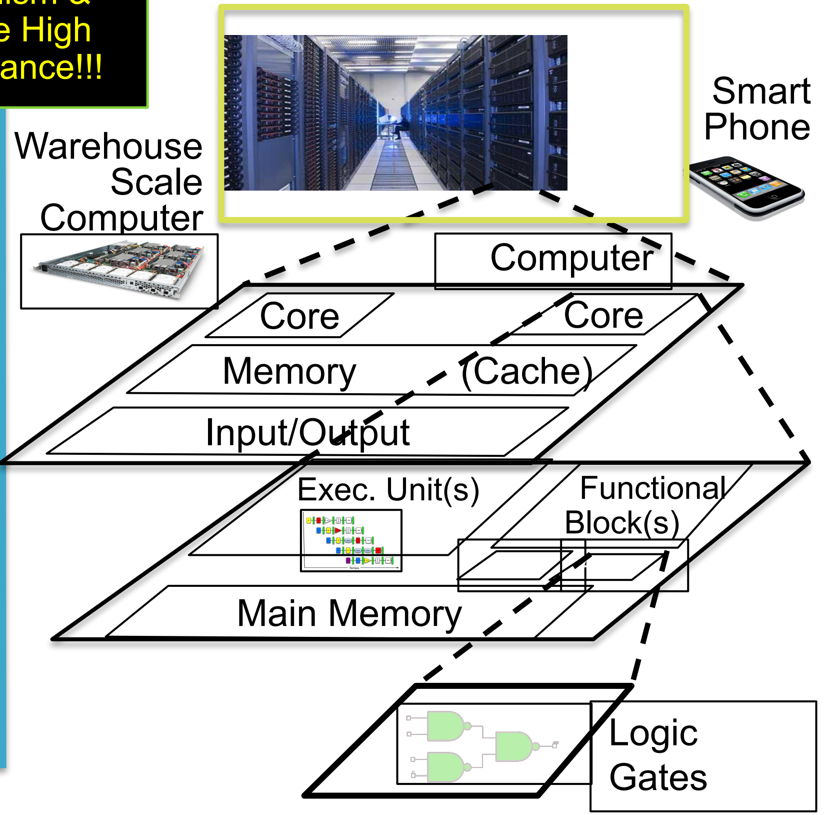
Parallel Threads
Assigned to core e.g., Lookup, Ads

Parallel Instructions
>1 instruction @ one time
e.g., 5 pipelined instructions

Parallel Data
>1 data item @ one time
e.g., Add of 4 pairs of words

Hardware descriptions
All gates work in parallel at same time

Hardware



Great Idea #4: Parallelism

- With slowdown in single-core scaling, new school computer architects needed other methods to scale performance
- Parallelism! In many forms:
    - Data-level parallelism (in this class: SIMD instructions)
    - Thread-level parallelism (OpenMP)
    - Process-level parallelism (Open MPI)
    - We even touched on request-level parallelism in warehouse-scale computers.

Gene Amdahl

Computer Pioneer

So does this mean we can improve performance without limits?

Unfortunately, and unsurprisingly, no. Even with the appropriate hardware and software support, we're limited by **Amdahl's law**.

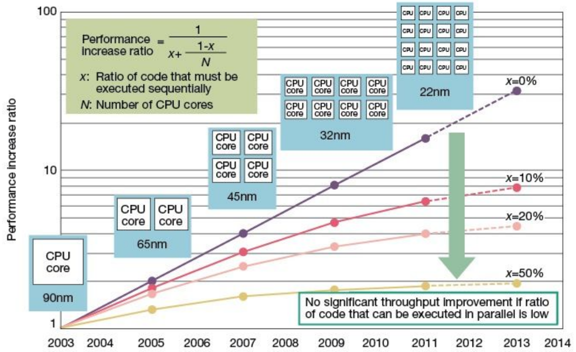
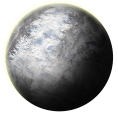
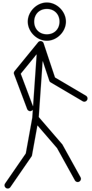


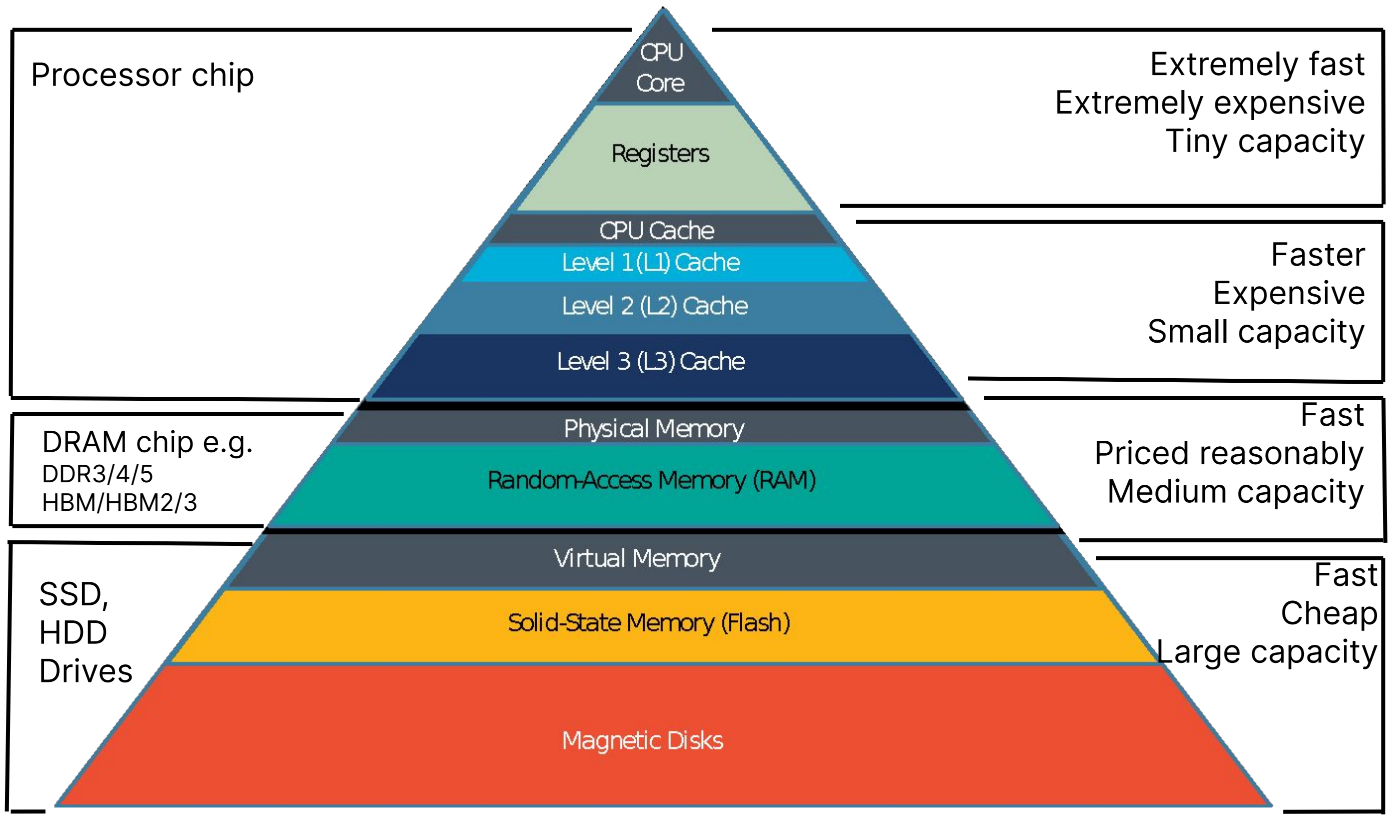
Fig 3 Amdahl's Law an Obstacle to Improved Performance Performance will not rise in the same proportion as the increase in CPU cores. Performance gains are limited by the ratio of software processing that must be executed sequentially. Amdahl's Law is a major obstacle in boosting multicore microprocessor performance. Diagram assumes no overhead in parallel processing. Years shown for design rules based on Intel planned and actual technology. Core count assumed to double for each rule generation.

Great Idea #3: Principles of Locality / Memory Hierarchy

- We started out by having to deal with memory for the first time in C
- Then registers in RISC-V
- Then what goes in between: caches!
  - We discussed types of caches, multilevel caches, cache coherence
- Then added virtual memory
  - Provides isolation, protection, and space for programs
  - Harness the slow but large storage provided by SSD/HDD.

10^9 ns	Tape / Optical Robot		2,000 years
10^6 ns	Disk		2 years
100 ns	Main Memory		1.5 hours
10 ns	On-board cache	This campus	15 minutes
2 ns	On-chip cache	This classroom	2 minutes
< 1 ns	Registers		< 1 minute

Great Idea #3: Principles of Locality / Memory Hierarchy

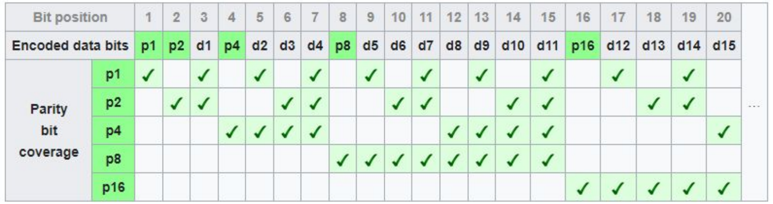


Great Idea #5: Performance Measurement & Improvement

- Most of the knowledge in 61C can somehow be applied to make code run faster
  - Assembly, pipelining, parallelism, caches, …
- We learned principles that can guide performance measurement/ improvement
  - Amdahl’s Law
  - Spatial and temporal locality
  - Analyzing cache misses and cache performance
  - etc.
- And put some of them into practice!
  - e.g. HW5, Project 4

Great Idea #6: Dependability via Redundancy

- Failures happen! (cosmic rays…)
- Redundancy is everywhere to protect your data
  - In datacenters, storage, memory, etc. in the form of ECC, RAID
- If you found ideas like ECC interesting, CS theory classes might be a good fit (CS 17x such as CS 170 - algorithms)



Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✓		✓		✓		✓		✓		✓		✓		✓		✓		✓		
	p2		✓	✓			✓	✓			✓	✓			✓	✓			✓	✓		...
	p4				✓	✓	✓	✓					✓	✓	✓	✓					✓	
	p8								✓	✓	✓	✓	✓	✓	✓	✓						
	p16																✓	✓	✓	✓	✓	

What’s Next?

Administrivia: Become active!

- **If you did well in CS10 or 61[ABC] (B or above) and want to be on staff?**
  - Usual path: AI (Academic Intern) → Tutor/Reader (UCS1) → TA (UCS2)
  - Application forms: https://eecs.berkeley.edu/resources/gsis/prospective
    - EECS 101 on Ed will have course-specific info
  - We strongly encourage anyone who gets an B or above in the class to follow this path!

Taking advantage of Cal Opportunities

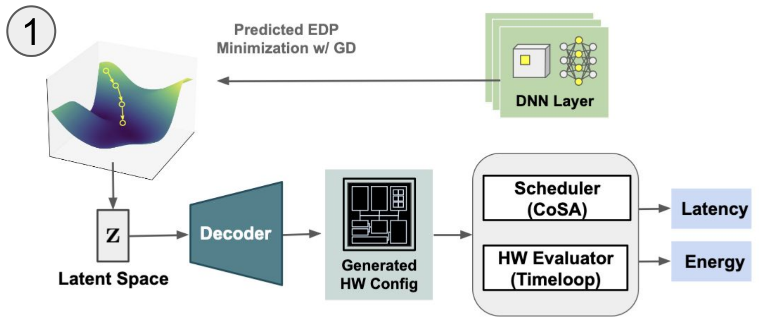
- **Why are we one of the top universities in the world?**
  - Research, research, research!
- **Techniques**
  - Find out what you like, take relevant classes, do lots of web research (read published papers), hit OH of Prof, be a go-getter!
- **http://research.berkeley.edu/**

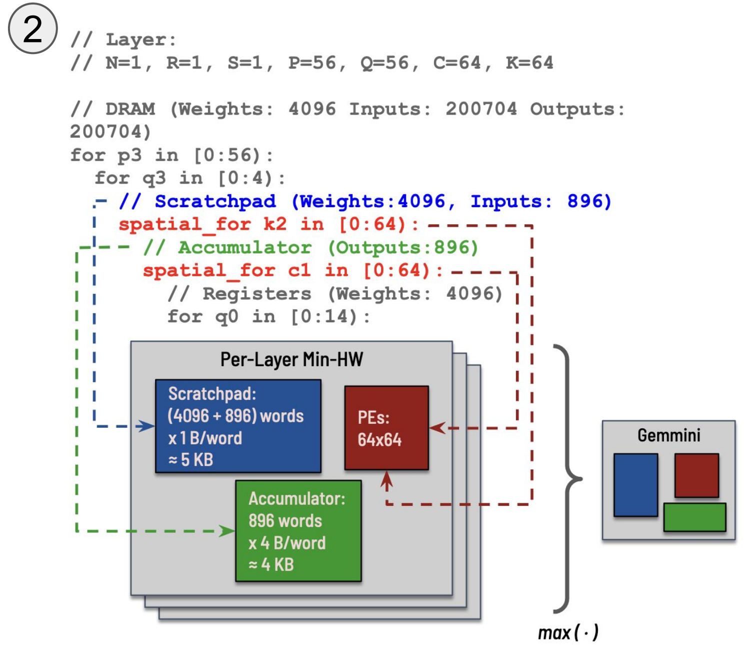
Research Opportunities in Computer Architecture and Systems

- **Domain-specific hardware/systems (big field right now)**
  - How do we run important applications (datacenter workloads, AI, robotics) as efficiently as possible?
    - Design and co-design of domain-specific languages (DSLs) and domain-specific accelerators (DSAs)
    - Techniques for translating, verifying, automatically generating DSLs and DSAs
    - Or make us all obsolete with quantum computing…
- **Fundamentals can still be improved upon!**
  - How do we make branch predictors, caches, parallel systems work better for today's important applications?

For example: Charles's Research Interests

- How can we automate the hardware design process - specifically, performance prediction and design space exploration?
- These are optimization problems - apply techniques from machine learning and optimization literature
- Idea 1 (ISPASS'22): Map hardware parameters to latent space using variational autoencoder and speed up search algorithms
- Idea 2 (MICRO'23): Collapse hardware DSE and software scheduling into one problem. Generate solutions to combined hard problem by running gradient descent on a differentiable performance model.




2
Per-Layer Min-HW
Scratchpad: (4096 + 896) words x 1 B/word ≈ 5 KB
PEs: 64x64
Accumulator: 896 words x 4 B/word ≈ 4 KB
Gemmini
max(·)

```
// Layer:
// N=1, R=1, S=1, P=56, Q=56, C=64, K=64

// DRAM (Weights: 4096 Inputs: 200704 Outputs:
200704)
for p3 in [0:56):
  for q3 in [0:4):
   // Scratchpad (Weights:4096, Inputs: 896)
    spatial_for k2 in [0:64):
     // Accumulator (Outputs:896)
      spatial_for c1 in [0:64):
        // Registers (Weights: 4096)
        for q0 in [0:14):
```

Classes

What was your favorite project in 61C?

proj1 - snek 0%

proj2 - 61Classify 0%

proj3 - 61CPU 0%

proj4 - 61kaChow 0%

None of the above 0%

Opportunities Next Semester

- CS152 (Computer Architecture and Engineering)
  - If you liked CPU design, this is a great follow-on course!
- CS161 (Computer Security)
  - If you liked memory management and C and are interested in exploiting badly written code
- CS162 (OS and System Programming)
  - If you liked the rest, this is a great follow-on course!
- CS170 (Algorithms)
  - If you didn’t like 61C much and would prefer more mathy/proofy

Opportunities Next Semester

- EE16A, EE16B (lower-div EE courses)
  - Fundamentals of signal processing, learning, control, and circuit design while introducing key linear-algebraic concepts motivated by application contexts.
- EE12X (signals, systems, comm, networks and optimization)
  - If you like making the computer solve interesting problems and make things work!
- EE105, 14X series (Circuits)
  - If you like to know what's below the gates
- EE13X (Devices)
  - …To know how to make IC and devices for computing
- EECS151 – Intro to Digital Systems and Circuits
  - EECS151LA – ASIC lab, EECS151LB – FPGA lab
  - Build RISC-V processors, peripherals, run on an FPGA or create chip layout

EECS 151

- Lab-based class: FPGA or ASIC (or both!)
    - FPGA: Field Programmable Gate Array, will be hands-on based (working with an FPGA board)
    - ASIC: Application-Specific Integrated Circuit, will be simulated
    - Both different ways in building and designing chips to be shipped for production!
- Focuses on datapath/SDS sections of course
    - A lot of things we've abstracted away in 61C become clearer at this level!
    - Delays of hardware components
    - Various gates and how they're created
    - FSMs
    - More datapath
- Introduces Verilog

CS 152

- Branch prediction, remember her?
  - It's an entire research field, and there are many different methods by which we can improve execution!
  - Bit-wise manipulation + tracking, mostly
- One-instruction executed per cycle max (even during pipelining)
  - No longer the case with instruction-level parallelism!
  - ILP: multiple instructions get executed per clock cycle!
  - Architecture that supports ILP include
    - Superscalar architecture
    - VLIW: very long instruction words (152)
- Vectorised architecture, SIMD which is Intel-supported
  - What about RISC-V vectorised architecture?
  - Currently undergoing research at Berkeley!
  - BAR: Berkeley Architecture Research
  - (Vectorising radix-based sorting algorithms?)

EE 149, 143

- Introduction to Embedded Systems, Microfabrication Technology
- EE 149
  - Embedded system: special purpose system (CPU, memory, I/O) designed to perform singular or few specific functions
  - Basic analysis, control, and systems simulation
  - How systems interface with real world usage
  - Embedded platforms ⇒ OS things
  - Distributed embedded systems
- EE 143
  - Microfabrication: how all of this stuff on chips/hardware gets actuated (typically on the micro-scale and on silicon chips)
  - "Thermal oxidation, ion implantation, impurity diffusion, film deposition, expitaxy, lithography, etching, contacts and interconnections, and process integration issues"
  - Transistors!

CS 161

- Computer Security
- Primary focus: Hardware, Software, Theory
- Part 1: How to hack programs, and how to prevent your programs from being hacked
- Part 2: How the internet works (because the internet is terribly insecure)
- Ends up affecting all layers of the tech stack, so it's fairly fast-paced
- Very important to take, but it's also not urgent

CS 162

- Operating Systems
- Primary focus: Hardware, Software
- Expands on the concepts of the OS and VM
- Uses C a lot, so it helps to be comfortable with explicit memory management
- There's also a lot of multiprocess programming
    - Locks, deadlocks, timers, etc.
- The main project sequence goes through writing a simple OS, and adding various bits of functionality to it.

CS 164

- Compilers
- Primary focus: Software, Theory
- The "C" part of CALL
- Has a lot of connections to linguistics and parsing
- The main project sequence goes through writing a compiler from (a variant of) Python to RISC-V, step by step
    - Lexer: Translate code into sequences of tokens (letters->words)
    - Parser: Translate tokens to meaningful statements (words->sentences)
    - Code Generation: Translate meaningful statements into the new language (sentences->new language)

200-level classes

- Many graduate-level classes also allow undergraduates to enroll (after graduate enrollment)
- Often go deeper into concepts covered in upper-division classes
    - EECS 251B: More digital circuits
    - CS 267: Parallel computing
    - CS 263: Compilers
    - CS 294: Various topics (based on the semester)
- Many of these classes are rarely taught, so take them when you can!
- Primary difference from undergraduate classes: Most graduate courses have no final, but instead have a final project, where you design or analyze an emerging research topic.

Questions?

Penultimate slides: Thanks to the Staff!

Penultimate slides: Standing on the shoulder of giants

Thanks to past instructors: Dan Garcia, Justin Yokota, Lisa Yan, Caroline Liu, Peyrin Kao, Nick Weaver, Cece McMahon, John Wawrzynek…

Thank **you**!

Course Evals:

https://course-evaluations.berkeley.edu/