EECS 570

Lecture 1

Parallel Computer Arch

Intel Paragon XP/S

Winter 2025

Prof. Satish Narayanasamy

http://www.eecs.umich.edu/courses/eecs570/

Slides developed in part by Profs. Austin, Adve, Dreslinski, Falsafi, Martin, Manerkar, Narayanasamy, Nowatzyk, Peh, and Wenisch of CMU, EPFL, MIT, UPenn, U-M, UIUC

EECS 570 Class Info

- Instructor: Prof. Satish Narayanasamy <nsatish@umich.edu>
- Research interests:
  - Intersection of computer architecture, systems, PL
- Current projects
  - Accelerators for genomics
  - Trusted hardware for improving security:
    - Confidential computing
    - Ransomware defense

EECS 570 Class Info

- GSIs:
  - Parin Senta psenta@umich.edu

- Class info:
  - URL: https://www.eecs.umich.edu/courses/eecs570/
  - Canvas for reading quizzes and reporting grades
  - Piazza for technical discussions and project coordination

- Enrollment:
  - **Will try to accommodate everyone**

Meeting Times

- Lecture:
  - MW 3:00-4:20pm, 1017 DOW
- Discussion
  - F 3:30-4:20pm, G906 COOL
  - Used for talking about programming assignments and projects
  - Also, for discussing project milestones
- Office Hours:
  - Prof. Satish: Fridays 1:30-2:30 4741 BBB
  - GSI: TBD (will be posted on course webpage)
- Q&A:
  - Use Piazza for all technical questions
  - Office hours for course policy feedback
  - Use email *very* sparingly

Who Should Take 570?

- Graduate Students (and seniors interested in research)
  - Computer architects to be
  - Computer system designers
  - Those interested in computer systems

- Required Background
  - Computer Architecture (e.g., EECS 370)
    - 470 recommended, but not required
  - C/C++ Programming

Grading (tentative)

- 2 Programming Assignments: 5% (PA1) and 10% (PA2)
- Reading Quizzes for (almost) every lecture: 10%
- Midterm Exam: 25%
- Final Exam: 25%
- Final Research Project: 25%
- **This course is a lot of work. Do NOT underestimate it.**
- The course grade is heavily weighted towards the end of the semester

Grading (Contd.)

- Group studies are encouraged
- Group discussions are encouraged
- However,…
  - All programming assignments must be results of individual work
  - All reading quizzes must be done individually, questions/ answers should not be posted publicly

There is no tolerance for academic dishonesty. Please refer to the University Policy on cheating and plagiarism. Discussion and group studies are encouraged, but all submitted material must be the student's individual work (or in case of the project, individual group work).

Some Advice on Reading

- Learning to read scientific articles is a skill you will learn in this course
  - Has significant amount of reading

- If you carefully read every paper from start to finish, it will take a very long time
- Learn to skim past details that are not critical to the paper's overall message

Reading Quizzes

- Quizzes due **before class every Monday**
- You must take an online quiz for *every* assigned paper
- There will be 2 multiple choice questions
  - The questions are chosen randomly from a list
  - You only have 5 minutes
    - Not enough time to find the answer if you haven't read the paper
  - You only get one attempt
- Some of the questions may be reused on the midterm/final
- 6 lowest quiz grades (of about 30) will be dropped over the course of the semester (e.g., can skip some if you fall ill)
  - Retakes/retries/reschedules will not be given for any reason
    - The quiz drops are intended to cover such cases

Reading Quizzes

Quizzes do not make the difference between you passing and failing

- They are only worth 10% of your grade
- Don't worry too much if you get some quiz questions wrong

The quiz questions have been used in 4 (possibly more) past semesters

- They did not cause people to fail in those semesters

Final Project

- Original research on a topic related to the course
  - Goal: conduct and present original research in computer architecture
  - 25% of overall grade
  - Done in groups of 4-5 (exceptions *may* be made for PhD students)
  - **Please don't join a final project group and then drop the course; it makes things difficult for the other group members**
- Available infrastructure
  - gem5 multiprocessor simulator
  - GPGPUsim
  - Pin
  - Xeon Phi accelerators
- Timeline (and further info) will be posted on course website

Late Assignment Policy

- No late submissions for PA1, PA2, or final project
- If you have extenuating circumstances (e.g., admitted to hospital), email us and we'll work something out
  - Having too much coursework to complete to be able to meet the deadline is not an extenuating circumstance
- You will have ample time to work on the assignments, so please start early!

Regrade Policy

- Applies only to PA1, PA2, and midterm
- Regrade requests must be submitted in writing within one week from the day the assignment/midterm is handed back
- Regrade requests must specify clearly what the grading issue is
  - There should be a clear technical grading mistake for a regrade to be justified
  - Do not use regrades to just try and squeeze out more points

Announcements

No discussion this Friday.

Online quizzes (Canvas) available, stay on top of the readings and take the first 2 quizzes soon.

Please sign up for Piazza via Canvas

Readings

Before next Monday

- David Wood and Mark Hill. "Cost-Effective Parallel Computing," *IEEE Computer*, 1995.
- Mark Hill et al. "21st Century Computer Architecture." CCC White Paper, 2012.

Parallel Computer Architecture

The Multicore Revolution
Why did it happen?

If you want to make your computer faster, there are only two options:

~~1. increase clock frequency~~

2. execute two or more things in parallel

~~Instruction-Level Parallelism (ILP)~~

Programmer specified explicit parallelism

The ILP Wall

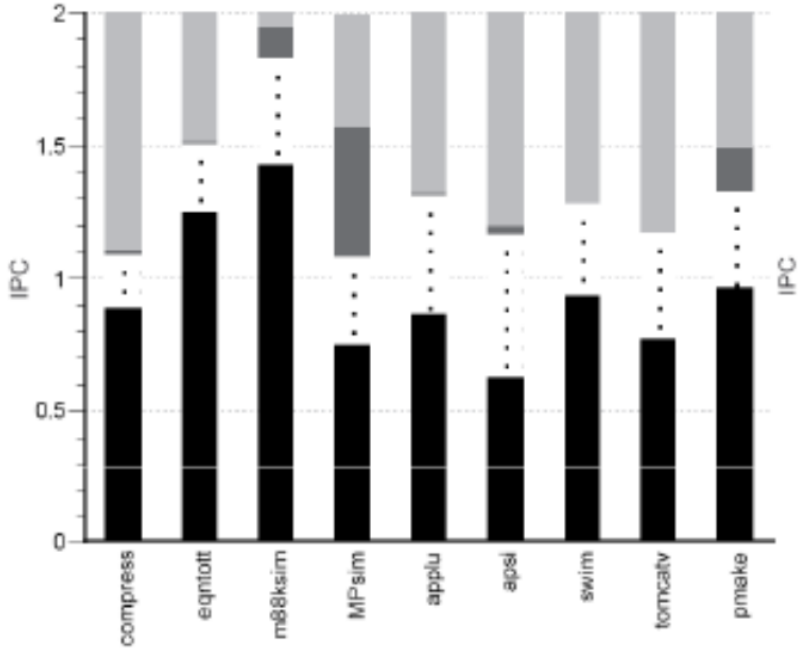


Figure 4. IPC Breakdown for a single 2-issue

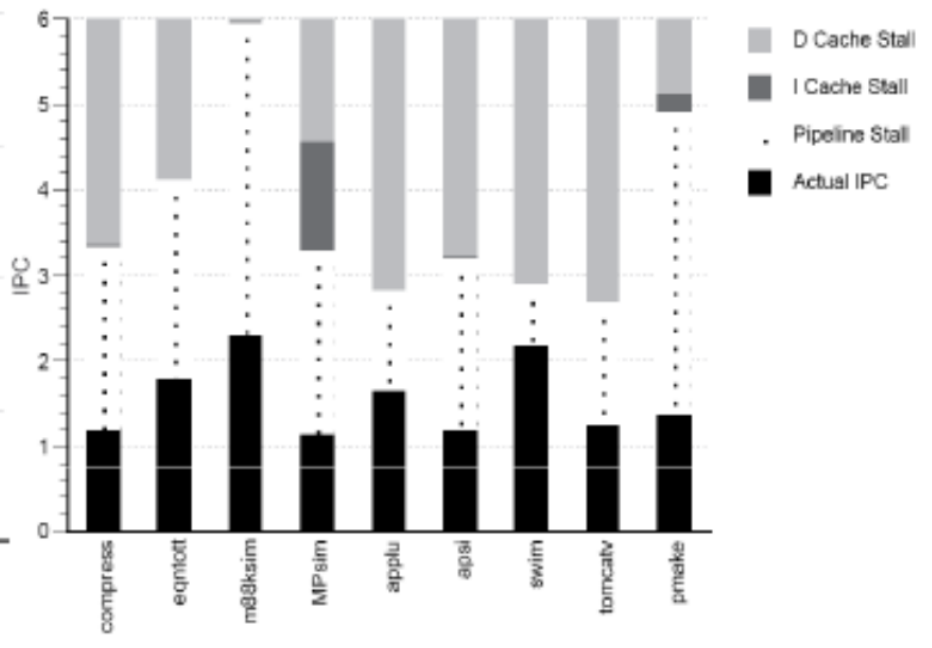
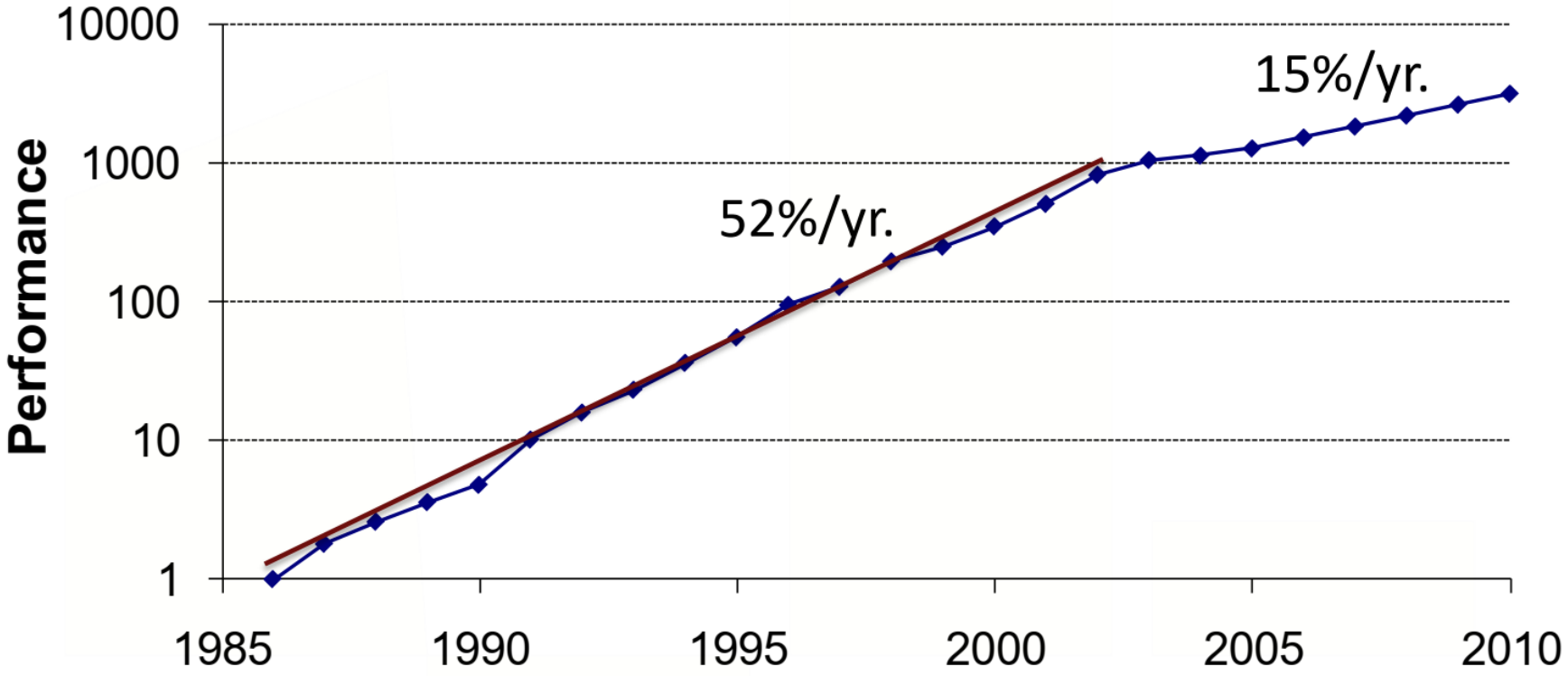


Figure 5. IPC Breakdown for the 6-issue processor.

Olukotun et al ASPLOS 96

- 6-issue has higher IPC than 2-issue, but not by 3x
  - Memory (I & D) and dependence (pipeline) stalls limit IPC

Single-thread performance

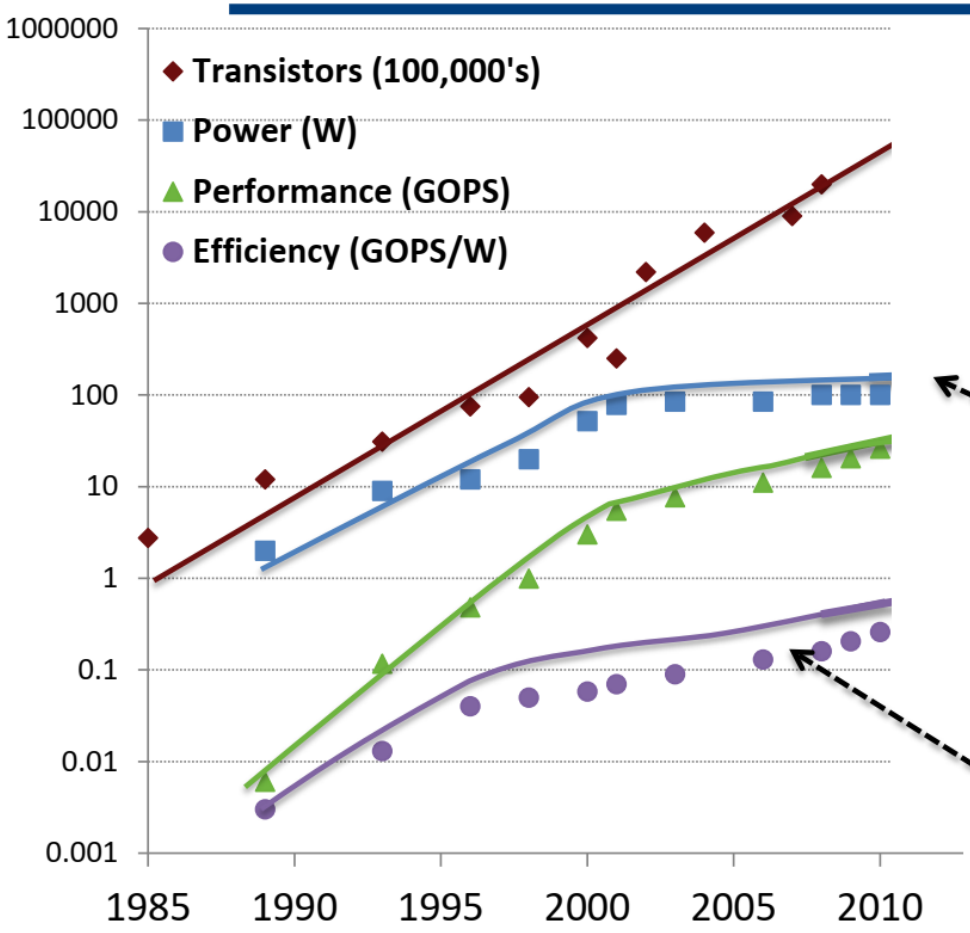


Source: Hennessy & Patterson, *Computer Architecture: A Quantitative Approach, 4th ed.*

Conclusion: Can't scale MHz or issue width to keep selling chips

Hence, multicore!

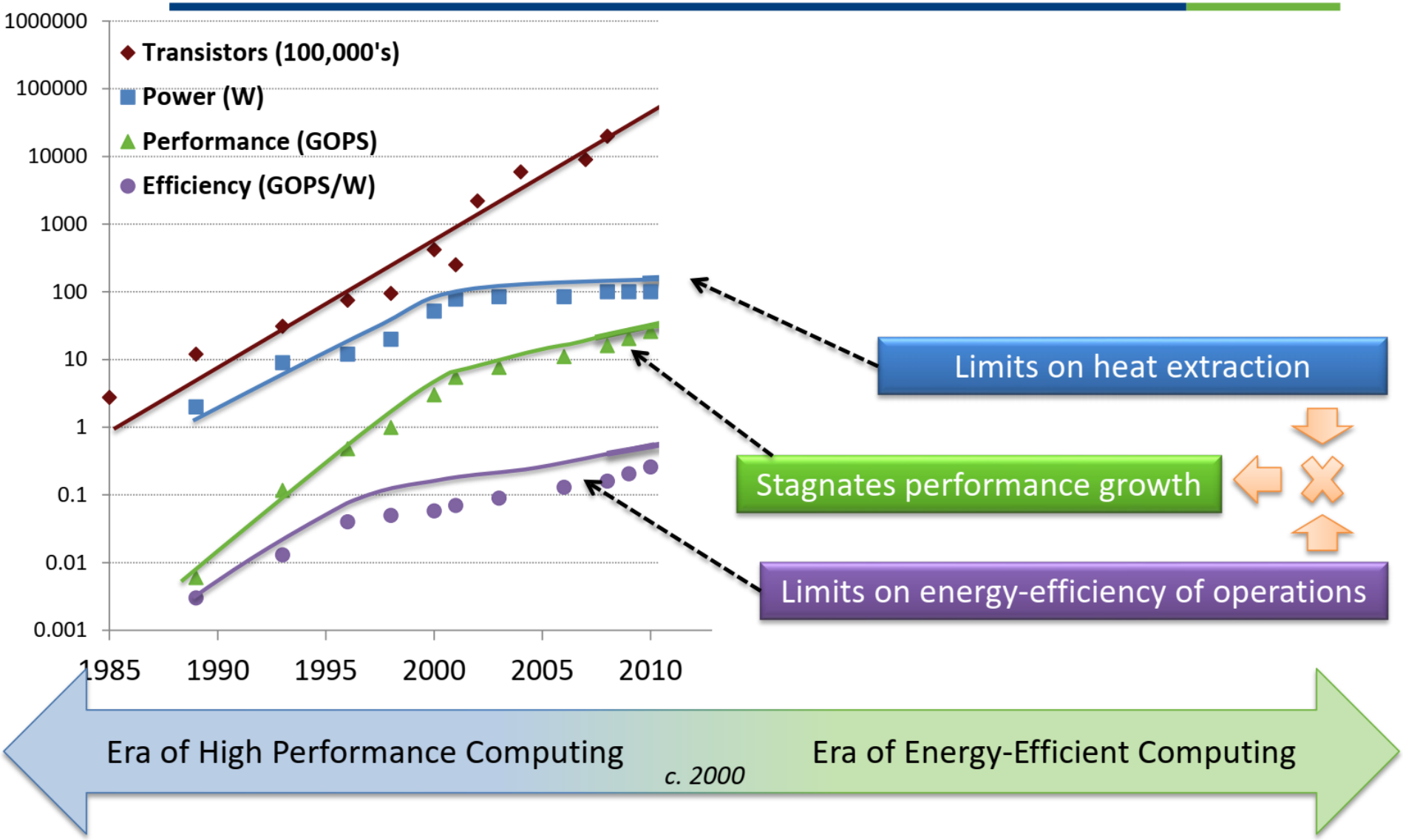
The Power Wall



Limits on heat extraction

Limits on energy-efficiency of operations

The Power Wall

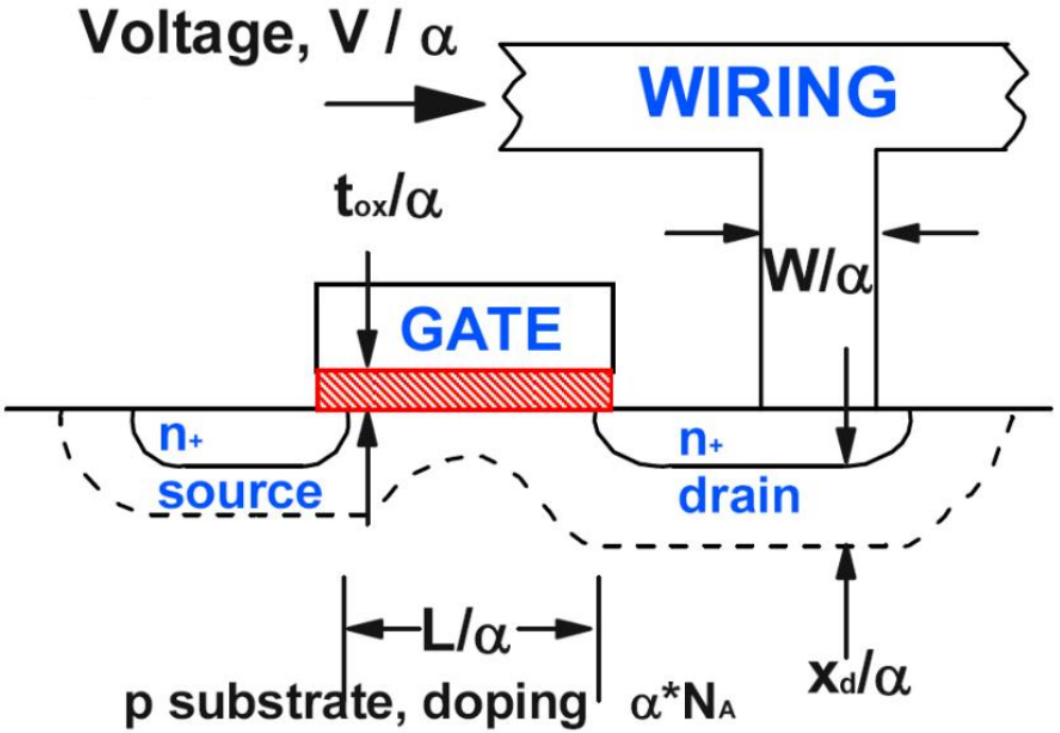


Classic CMOS Dennard Scaling: the Science behind Moore's Law

Source: Future of Computing Performance: Game Over or Next Level?, National Academy Press, 2011

Scaling:	
Voltage:	V/α
Oxide:	t_{ox}/α
Wire width:	W/α
Gate width:	L/α
Diffusion:	x_d/α
Substrate:	αN_A

Results:	
Higher Density:	$\sim\alpha^2$
Higher Speed:	$\sim\alpha$
Power/ckt:	$1/\alpha^2$
Power Density:	~Constant



$$P = C V^2 f$$

R. H. Dennard *et al.*, *IEEE J. Solid State Circuits*, (1974).

Post-classic CMOS Dennard Scaling

Post Dennard CMOS Scaling Rule

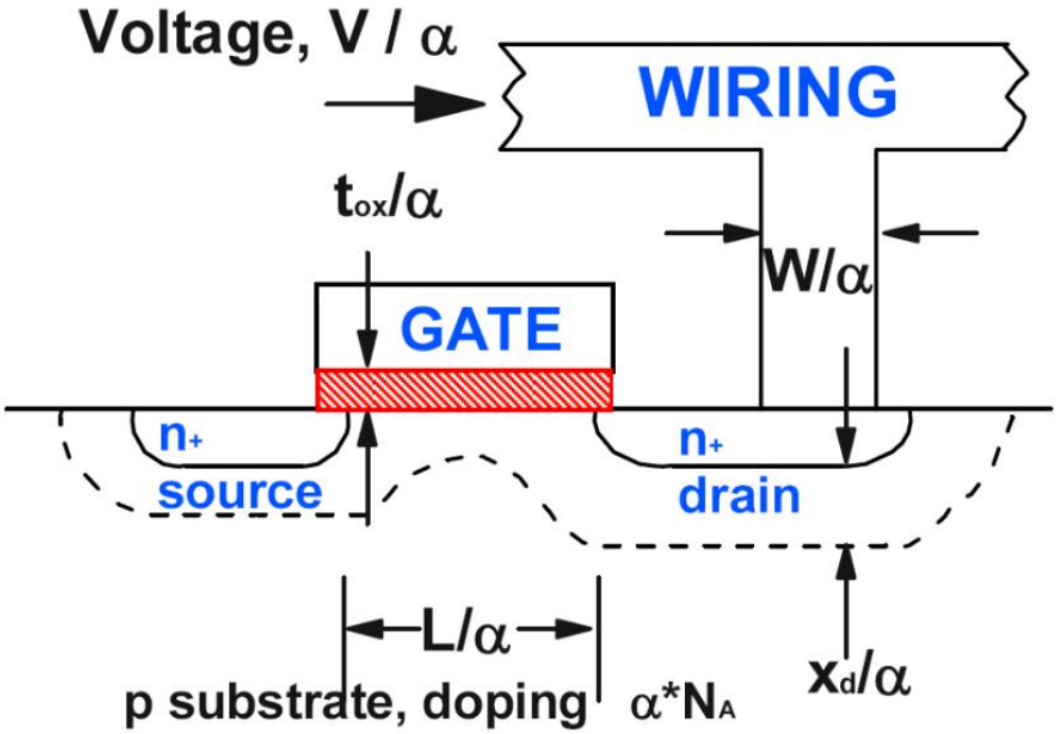
TODO:
Chips w/ higher power (no), smaller (☹), dark silicon (☺), or other (?)

Scaling:

Voltage:	~~V/α~~	V
Oxide:	t_{ox}/α	
Wire width:	W/α	
Gate width:	L/α	
Diffusion:	x_d/α	
Substrate:	αN_A	

Results:

Higher Density:	$\sim\alpha^2$	
Higher Speed:	$\sim\alpha$	
~~Power/ckt:~~	~~$1/\alpha^2$~~	~~1~~
Power Density:	~~$\sim$Constant~~	α^2



$$P = C V^2 f$$

R. H. Dennard *et al.*, *IEEE J. Solid State Circuits*, (1974).

Leakage Killed Dennard Scaling

Leakage:

- Exponential in inverse of V_{th}
- Exponential in temperature
- Linear in device count

To switch well

- must keep $V_{dd}/V_{th} > 3$

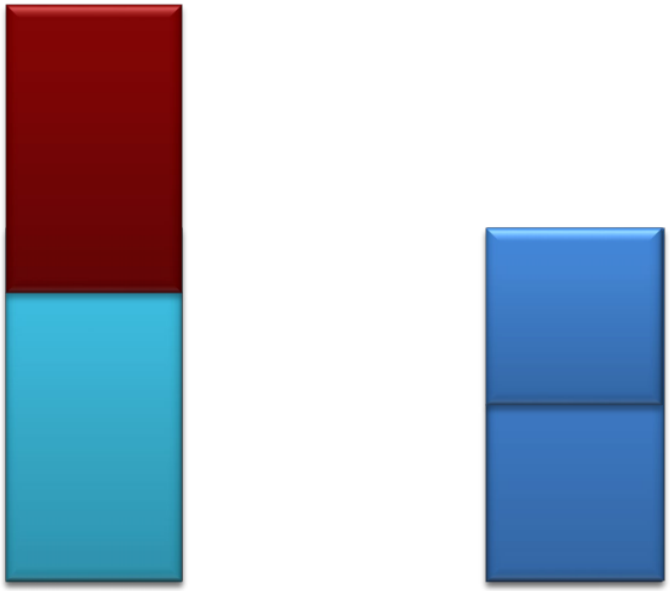
→V_{dd} can't go down

Multicore: Solution to Power-constrained design?

Power = CV^2F $\quad F \propto V$

Scale clock frequency to 80%

Now add a second core



Performance Power

Same power budget, but 1.6x performance!

But:

- Must parallelize application
- Remember Amdahl's Law!

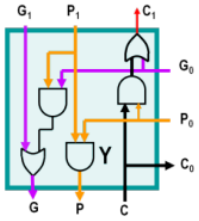
What Is a Parallel Computer?

"A collection of processing elements that communicate and cooperate to solve large problems fast."

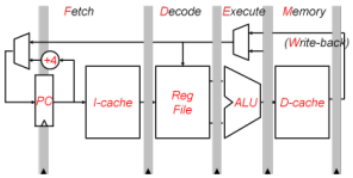
Almasi & Gottlieb, 1989

Spectrum of Parallelism

Bit-level

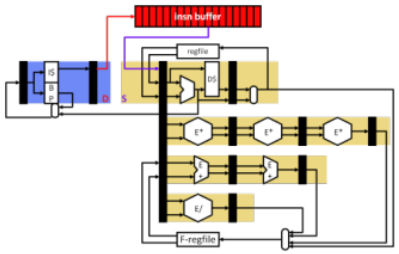


Pipelining



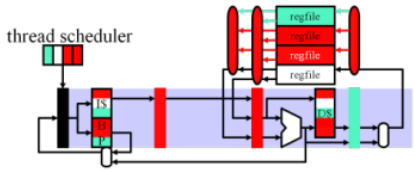
EECS 370

ILP



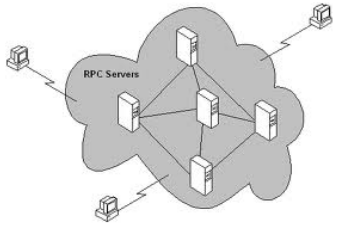
EECS 470

Multithreading
Multiprocessing



EECS 570

Distributed



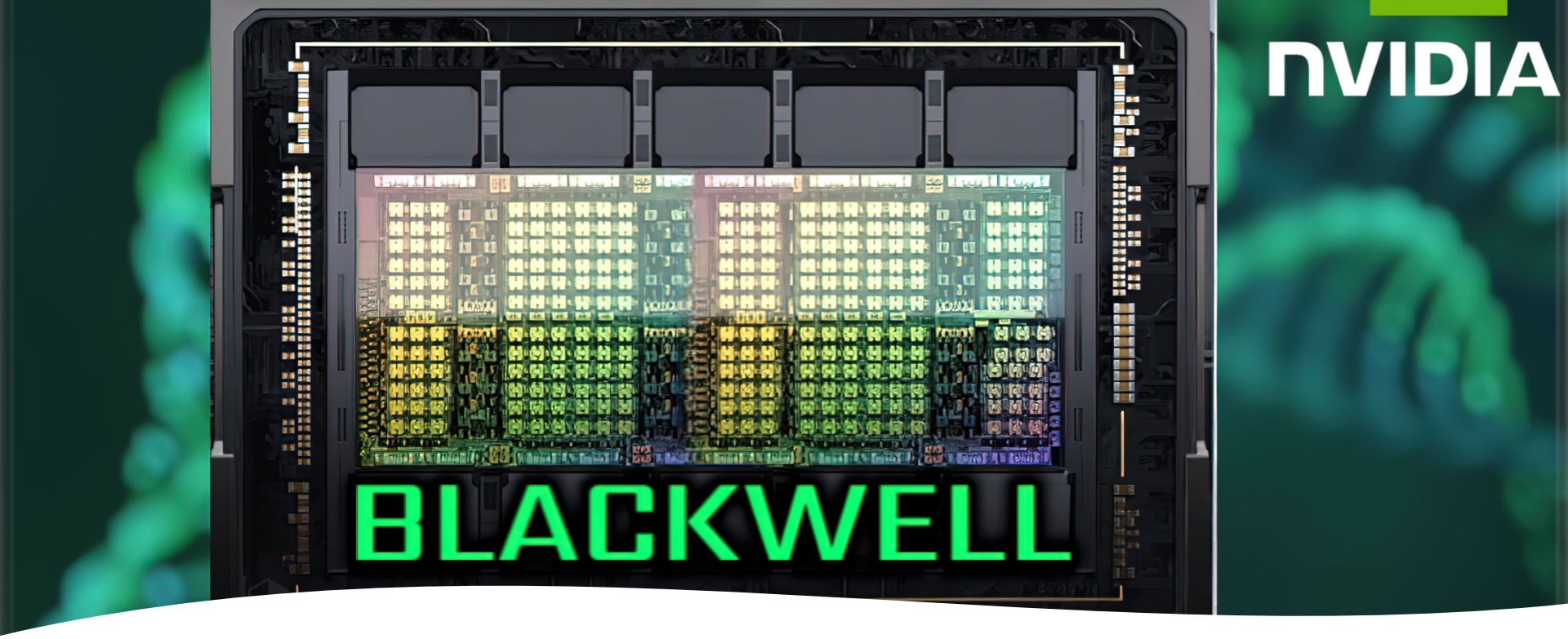
EECS 591

Why multiprocessing?

- Desire for performance
- Techniques from 370/470 difficult to scale further

Why Parallelism Now?

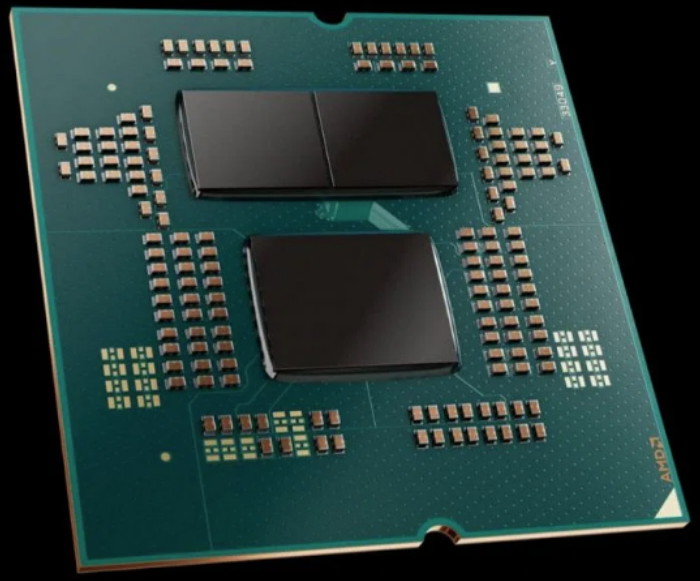
- All major processor vendors are producing multicore chips
  - Most machines today are already parallel machines
  - GPU and accelerators are driving advancements in AI
  - All programmers will be parallel programmers?
- New software model
  - Want a new feature? Hide the "cost" by speeding up the code first
  - All programmers will be performance programmers?
- Some may eventually be hidden in libraries, compilers, and high-level languages
  - But a lot of work is needed to get there
- Big open questions:
  - How should the chips, languages, OS be designed to make it easier for us to develop efficient parallel programs?



4nm, 104 billion transistors

Dual-Die: Two GB100 dies in a single package, connected via a 10 TB/s NV-High Bandwidth Interface (NV-HBI)

NVIDIA's NVLink interconnect, capable of scaling up to 576 GPUs



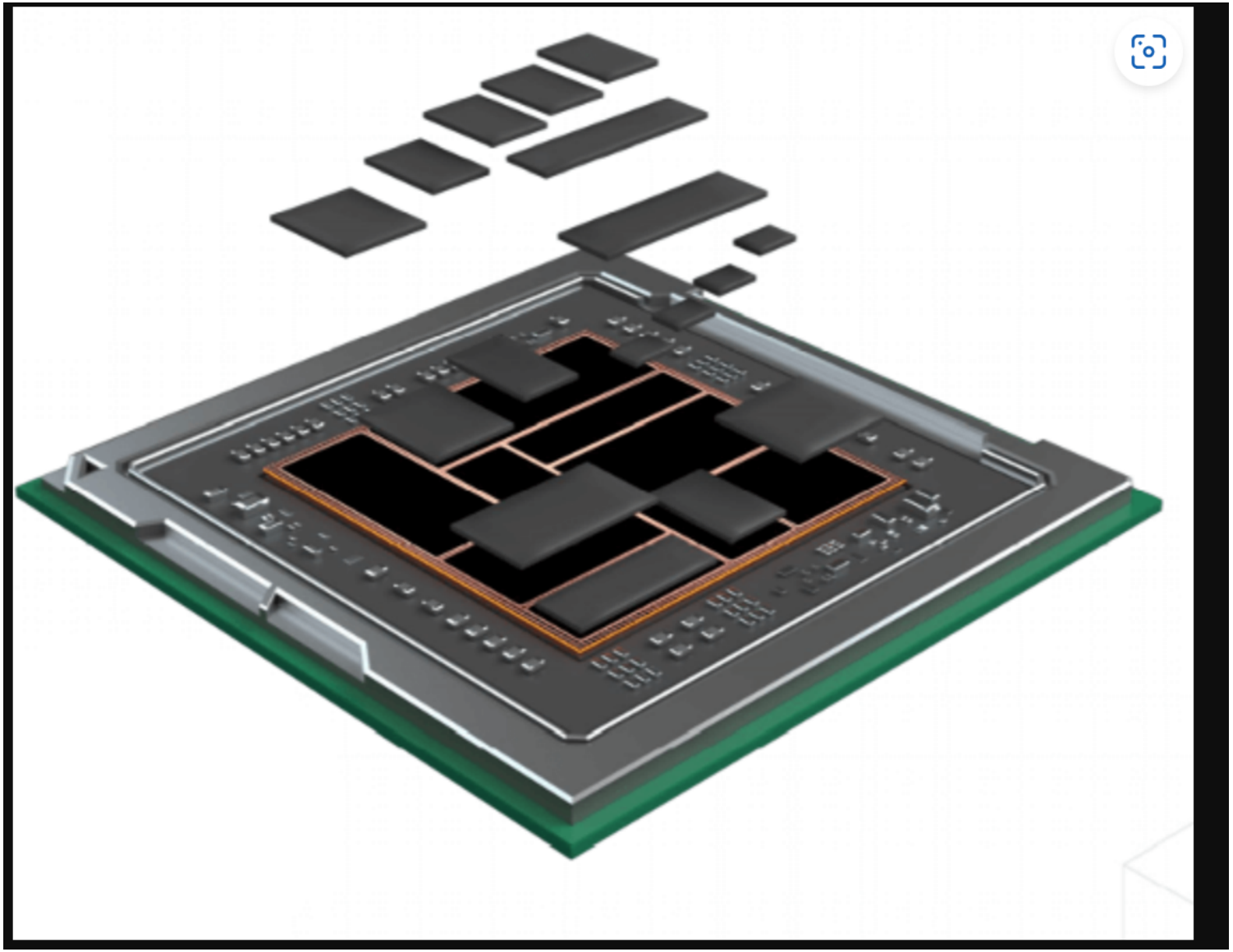
Announcing at Computex 2024

AMD Ryzen™ 9 9950X

The best for gamers and creators

16 Cores 32 Threads	Up to 5.7 GHz Boost	80MB L2+L3 Cache	170W TDP

Chiplets



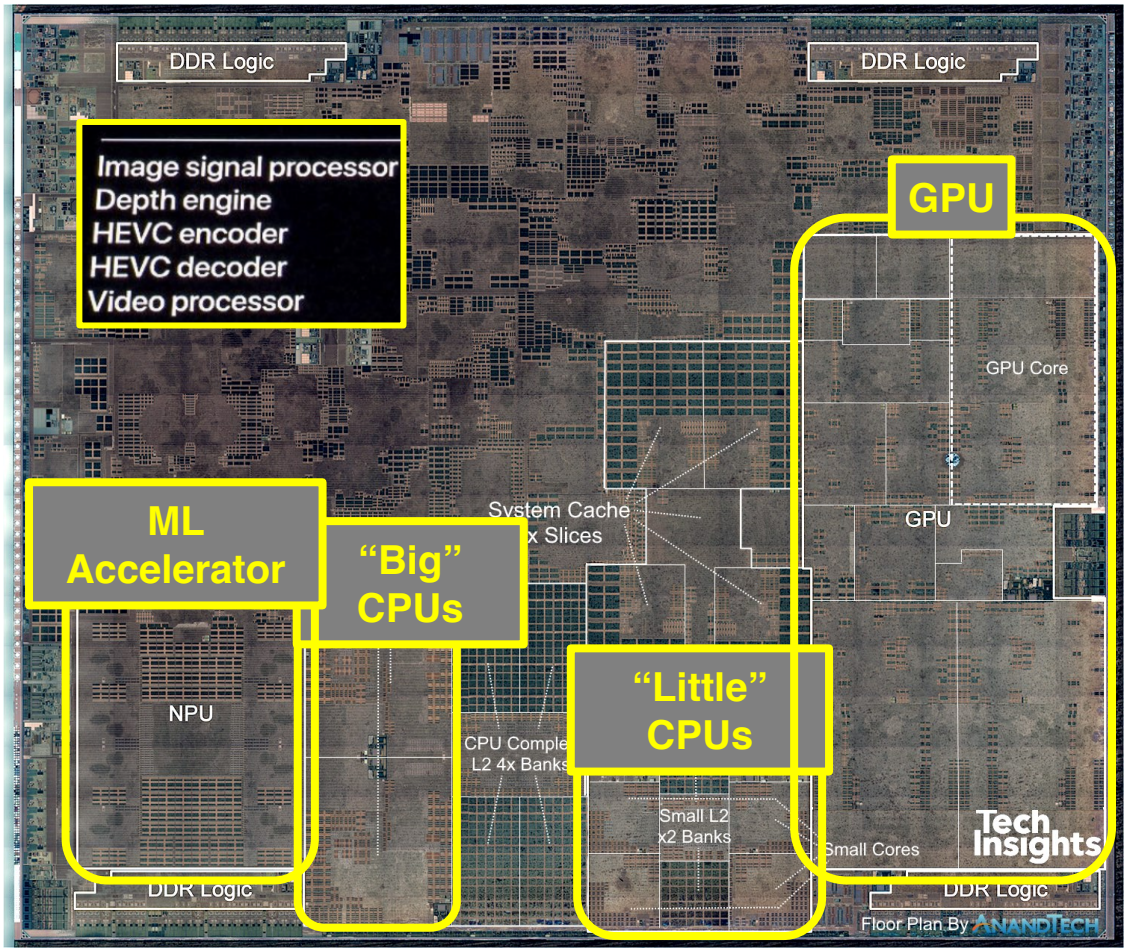
Multiprocessors Are Here To Stay

- Moore's law is making the multiprocessor a commodity part
  - 1B transistors on a chip, what to do with all of them?
  - Not enough ILP to justify a huge uniprocessor
  - Really big caches? t_{hit} increases, diminishing $\%_{miss}$ returns

- **Chip multiprocessors (CMPs)**
  - Every computing device (even your cell phone) is now a multiprocessor

Accelerator-Level Parallelism [Reddi and Hill 2019]

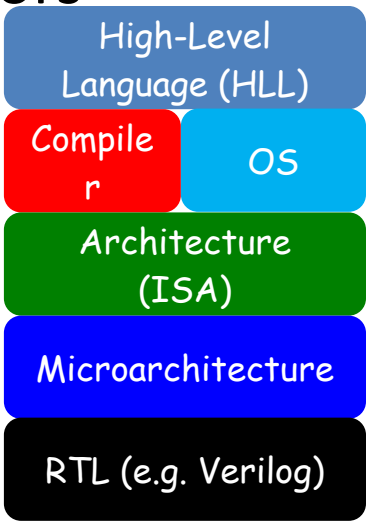
- Accelerators are specialized processing elements for different types of tasks
  - Machine learning, crypto, graphics (GPUs), etc



- Apple A12 System-on-Chip (SoC)
  - More than 40 accelerators [Wang and Shao 2019]

Heterogeneous Parallelism Across the Stack

- Parallelism has percolated up to high-level languages and compilers
  - Java, C/C++11, C#, etc. all have threads
- OS modified to best utilize parallel hardware
- New software toolchains for hardware accelerators
  - e.g. TensorFlow, PyTorch, TVM
- New landscape!



Course Outline

- Unit I – Parallel Programming Models
  - Message passing, shared memory, data-parallel (CUDA)
  - Synchronization, Locks, Lock-free structures
  - Transactional Memory
- Unit II –Coherency and Consistency
  - Snooping bus-based systems
  - Directory-based distributed shared memory
  - Memory Consistency Models
- Unit III – Interconnection Networks
  - On-chip, off-chip networks
  - Chiplets (new)
- Unity IV – Accelerators (new)
  - GPUs
  - ML Accelerators
  - Processing-in-Memory

Parallel Programming Intro

Motivation for MP Systems

- Classical reason for multiprocessing:

  More performance by using multiple processors in parallel

  - Divide computation among processors and allow them to work concurrently
  - Assumption 1: There is parallelism in the application
  - Assumption 2: We can exploit this parallelism

Finding Parallelism

1. Functional parallelism
   - Car: {engine, brakes, entertain, nav, …}
   - Game: {physics, logic, UI, render, …}
   - Signal processing: {transform, filter, scaling, …}
2. Data parallelism
   - Vector, matrix, db table, pixels, …
3. Request parallelism
   - Web, shared database, telephony, …

Computational Complexity of (Sequential) Algorithms

- Model: Each step takes a unit time
- Determine the time (/space) required by the algorithm as a function of input size

Sequential Sorting Example

- Given an array of size n

- MergeSort takes O(n log n) time
- BubbleSort takes $O(n^2)$ time

- But, a BubbleSort implementation can sometimes be faster than a MergeSort implementation
- Why?

Sequential Sorting Example

- Given an array of size n
- MergeSort takes O(n log n) time
- BubbleSort takes $O(n^2)$ time
- But, a BubbleSort implementation can sometimes be faster than a MergeSort implementation
- The model is still useful
  - Indicates the scalability of the algorithm for large inputs
  - Lets us prove things like a sorting algorithm requires at least O(n log n) comparisons

We need a similar model for parallel algorithms