Lecture 1:

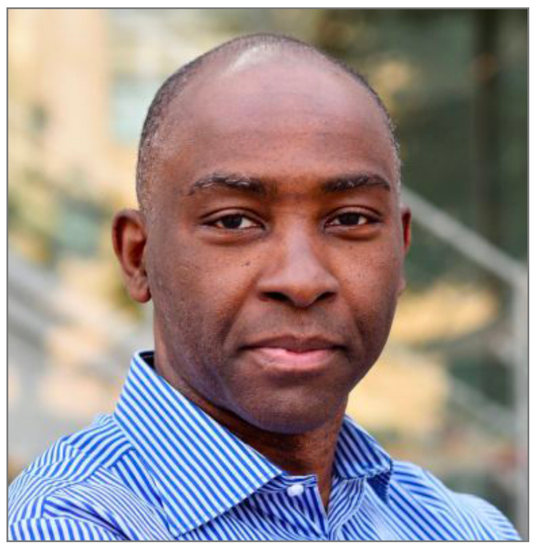
Why Parallelism?

Why Efficiency?

Parallel Computing
Stanford CS149, Fall 2020

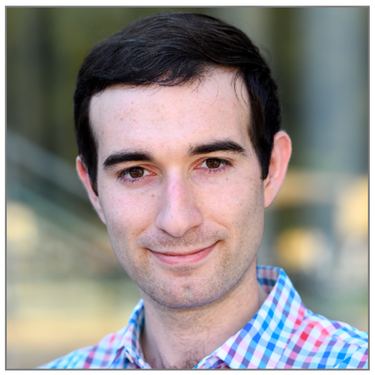
Hi!

Prof. Kayvon

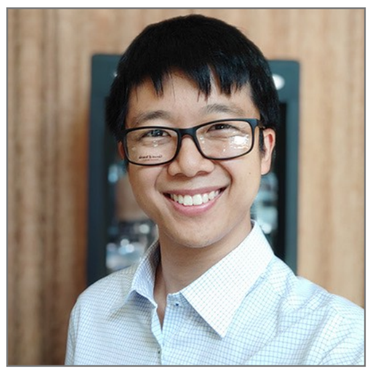


Prof. Olukotun

Fait



David



Matthew

One common definition

A parallel computer is a collection of processing elements that cooperate to solve problems quickly

We care about performance *
We care about efficiency

We're going to use multiple processors to get it

*** Note: different motivation from "concurrent programming" using threads like in CS110**

DEMO 1
(CS149 Fall 2020's first parallel program)

Speedup

One major motivation of using parallel processing: achieve a speedup

For a given problem:

$$\text{speedup(using P processors)} = \frac{\text{execution time (using 1 processor)}}{\text{execution time (using P processors)}}$$

Class observations from demo 1

- **Communication limited the maximum speedup achieved**
  - In the demo, the communication was telling each other the partial sums

- **Minimizing the cost of communication improved speedup**
  - Moved students ("processors") closer together (or let them shout)

DEMO 2

(scaling up to four “processors”)

Class observations from demo 2

- Imbalance in work assignment limited speedup
  - Some students ("processors") ran out work to do (went idle), while others were still working on their assigned task

- Improving the distribution of work improved speedup

DEMO 3

(massively parallel execution)

Class observations from demo 3

- The problem I just gave you has a significant amount of communication compared to computation
- Communication costs can dominate a parallel computation, severely limiting speedup

Course theme 1:

Designing and writing parallel programs ... <u>that scale</u>!

- **Parallel thinking**
  1. **Decomposing work into pieces that can safely be performed in parallel**
  2. **Assigning work to processors**
  3. **Managing communication/synchronization between the processors so that it does not limit speedup**

- **Abstractions/mechanisms for performing the above tasks**
  - **Writing code in popular parallel programming languages**

Course theme 2:

Parallel computer hardware implementation: how parallel computers work

- Mechanisms used to implement abstractions efficiently
  - Performance characteristics of implementations
  - Design trade-offs: performance vs. convenience vs. cost

- Why do I need to know about hardware?
  - Because the characteristics of the machine really matter (recall speed of communication issues in earlier demos)
  - Because you care about efficiency and performance (you are writing parallel programs after all!)

Course theme 3:

Thinking about efficiency

- **FAST != EFFICIENT**

- **Just because your program runs faster on a parallel computer, it does not mean it is using the hardware efficiently**
  - **Is 2x speedup on computer with 10 processors a good result?**

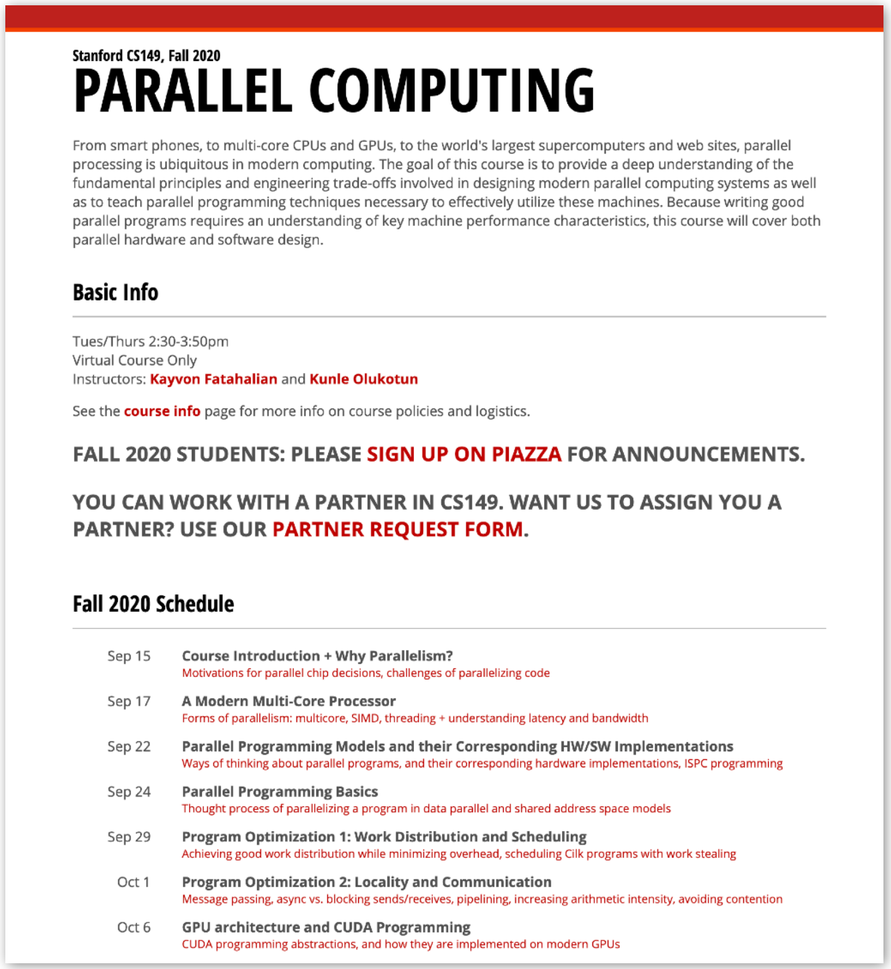
- **Programmer's perspective: make use of provided machine capabilities**

- **HW designer's perspective: choosing the right capabilities to put in system (performance/cost, cost = silicon area?, power?, etc.)**

Course logistics

Getting started

- The course web site
  - http://cs149.stanford.edu
- Sign up for the course on Piazza
  - https://piazza.com/stanford/fall2020/cs149/home
- Fill out our partner request form
  - If you want us to match you with a partner
- Textbook
  - There is no course textbook (the internet is plenty good these days), but please see course web site for suggested references



Stanford CS149, Fall 2020

PARALLEL COMPUTING

From smart phones, to multi-core CPUs and GPUs, to the world's largest supercomputers and web sites, parallel processing is ubiquitous in modern computing. The goal of this course is to provide a deep understanding of the fundamental principles and engineering trade-offs involved in designing modern parallel computing systems as well as to teach parallel programming techniques necessary to effectively utilize these machines. Because writing good parallel programs requires an understanding of key machine performance characteristics, this course will cover both parallel hardware and software design.

Basic Info

Tues/Thurs 2:30-3:50pm
Virtual Course Only
Instructors: Kayvon Fatahalian and Kunle Olukotun

See the course info page for more info on course policies and logistics.

FALL 2020 STUDENTS: PLEASE SIGN UP ON PIAZZA FOR ANNOUNCEMENTS.

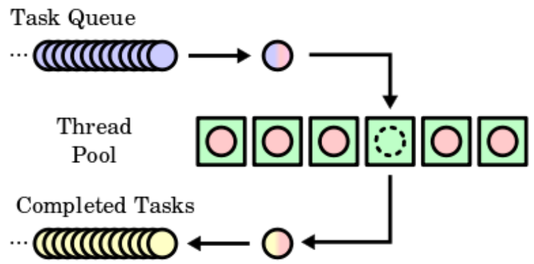
YOU CAN WORK WITH A PARTNER IN CS149. WANT US TO ASSIGN YOU A PARTNER? USE OUR PARTNER REQUEST FORM.

Fall 2020 Schedule

Sep 15	**Course Introduction + Why Parallelism?** Motivations for parallel chip decisions, challenges of parallelizing code
Sep 17	**A Modern Multi-Core Processor** Forms of parallelism: multicore, SIMD, threading + understanding latency and bandwidth
Sep 22	**Parallel Programming Models and their Corresponding HW/SW Implementations** Ways of thinking about parallel programs, and their corresponding hardware implementations, ISPC programming
Sep 24	**Parallel Programming Basics** Thought process of parallelizing a program in data parallel and shared address space models
Sep 29	**Program Optimization 1: Work Distribution and Scheduling** Achieving good work distribution while minimizing overhead, scheduling Cilk programs with work stealing
Oct 1	**Program Optimization 2: Locality and Communication** Message passing, async vs. blocking sends/receives, pipelining, increasing arithmetic intensity, avoiding contention
Oct 6	**GPU architecture and CUDA Programming** CUDA programming abstractions, and how they are implemented on modern GPUs

Four programming assignments

Assignment 1: ISPC programming on multi-core CPUs



Assignment 2: scheduler for a task graph

Assignment 3: Writing a renderer in CUDA on NVIDIA GPUs

Assignment 4: parallel large graph algorithms on a multi-core CPU

Optional assignment 5: (will boost exam grade)

Plus a few optional extra credit challenges… ;-)

Written assignments

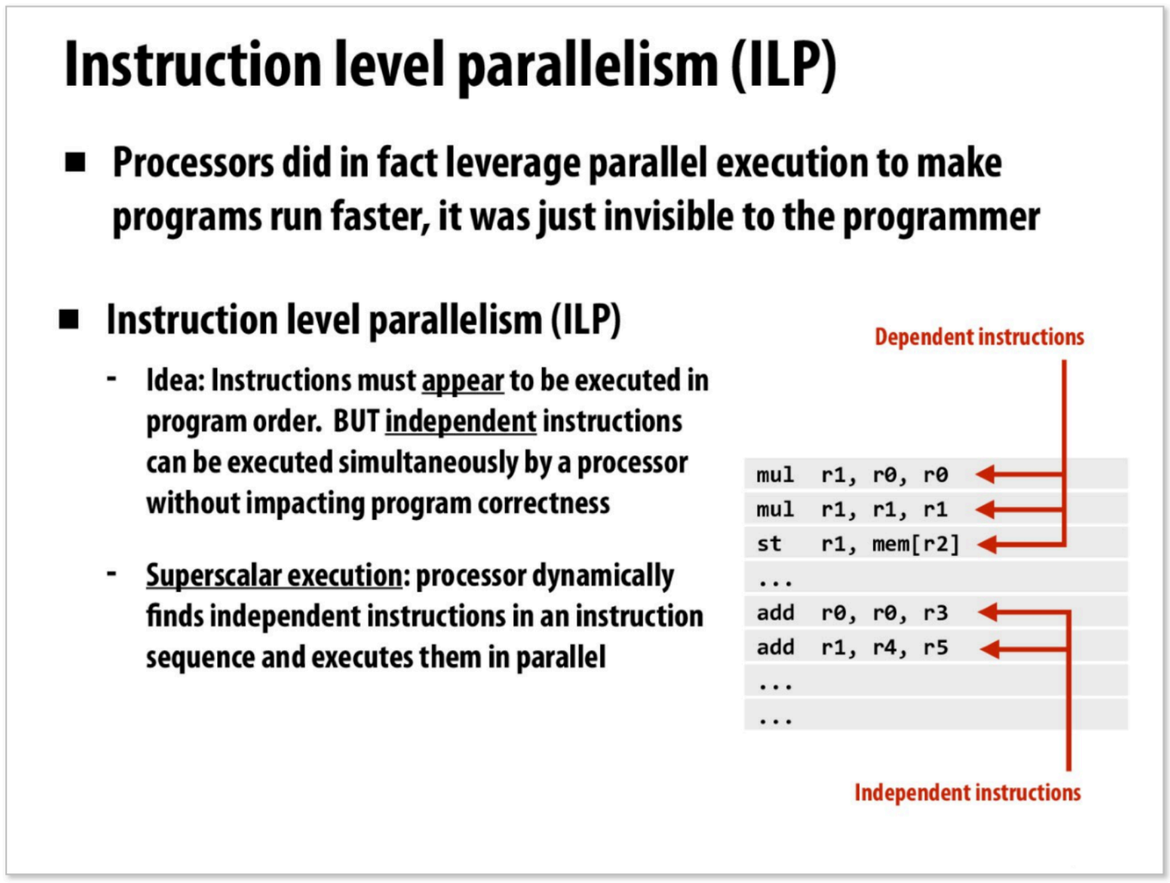
- Approximately every two-weeks we will have a take-home written assignment

- Written assignments contain modified versions of previous exam questions, so consider them practice for the exam

Commenting and contributing to lectures

We have no textbook for this class and so the lecture slides are the primary course reference

Why Parallelism? Why Efficiency?



Previous | **Next** --- Slide 30 of 48

Back to **Lecture Thumbnails**

The website supports commenting on a per-slide basis

rrastogi

It is computationally expensive for the processor to determine dependencies between instructions. The following PPT (slides 9/10) provides an example of how the number of checks grows with the number of instructions that are simultaneously dispatched: **http://www.cs.cmu.edu/afs/cs/academic/class/15740-f15/www/lectures/11-superscalar-pipelining.pdf**

This additional cost is likely one of the predominant reasons that ILP has plateaued at 4 simultaneous instructions. To circumvent this issue, architects have tried to force the compiler to solve the dependency issue using VLIW (very long instruction word). To summarize VLIW, if a processor contains 5 independent execution units, the compiler will have 5 operations in the "very long instruction word" that the processor will map to the 5 execution units: **https://en.wikipedia.org/wiki/Very_long_instruction_word**. This way dependency checking is the responsibility of software and not hardware.

I am not sure if VLIW has helped significantly pushed the four simultaneous instruction threshold though. If somebody knows, please share.

kayvonf

Question: The key phrase on this slide is that a processor must execute instructions in a manner "appears" as if they were executed in program order. **This is a *key idea* in this class.**

What is program order?

And what does it mean for the results of a program's execution *to appear* as if instructions were executed in program order?

And finally... Why is the program order guarantee a useful one? (What if the results of execution were inconsistent with the results that would be obtained if the instructions were executed in program order?)

void

> *And what does it mean for the results of a program's execution to appear as if instructions were executed in program order?*

A programmer might write something like the code below.

```
x = a + b
print(x)
y = c + d
print(y)
```

Participation (comments)

- You are encouraged to submit one <u>well-thought-out</u> comment per lecture (only two comments per week)

- Why do we write?
  - Because writing is a way many good architects and systems designers force themselves to think (explaining clearly and thinking clearly are highly correlated!)

- But take it seriously, this is your participation grade.

What we are looking for in comments

- Try to explain the slide (as if you were trying to teach your classmate while studying for an exam)
  - "The instructor said this, but if you think about it this way instead it makes much more sense…"
- Explain what is confusing to you:
  - "What I'm totally confused by here was…"
- Challenge classmates with a question
  - For example, make up a question you think might be on an exam.
- Provide a link to an alternate explanation
  - "This site has a really good description of how multi-threading works…"
- Mention real-world examples
  - For example, describe all the parallel hardware components in the XBox One
- Constructively respond to another student's comment or question
  - "@segfault21, are you sure that is correct? I thought that Prof. Kayvon said…"
- It is OKAY (and even encouraged) to address the same topic (or repeat someone else's summary, explanation or idea) in your own words
  - "@funkysenior19's point is that the overhead of communication…"

Grades

42% Programming assignments (4)

25% Written Assignments (5)

28% Exam

5% Asynchronous participation (comments)

Reminder: we can match you with a partner! See Piazza for our partner request form!

Expectations

- We plan to be flexible with students this quarter
  - If you need exceptions to policies, just come ask us, we understand the quarter might be very unpredictable

But…

- We expect all students in the class to work hard
  - We expect working and baseline performance handins of all programming assignments, and a passing exam score, to achieve a passing grade

Why parallelism?

Some historical context: why <u>not</u> parallel processing?

- Single-threaded CPU performance doubling ~ every 18 months
- Implication: working to parallelize your code was often not worth the time
  - Software developer does nothing, code gets faster next year. Woot!

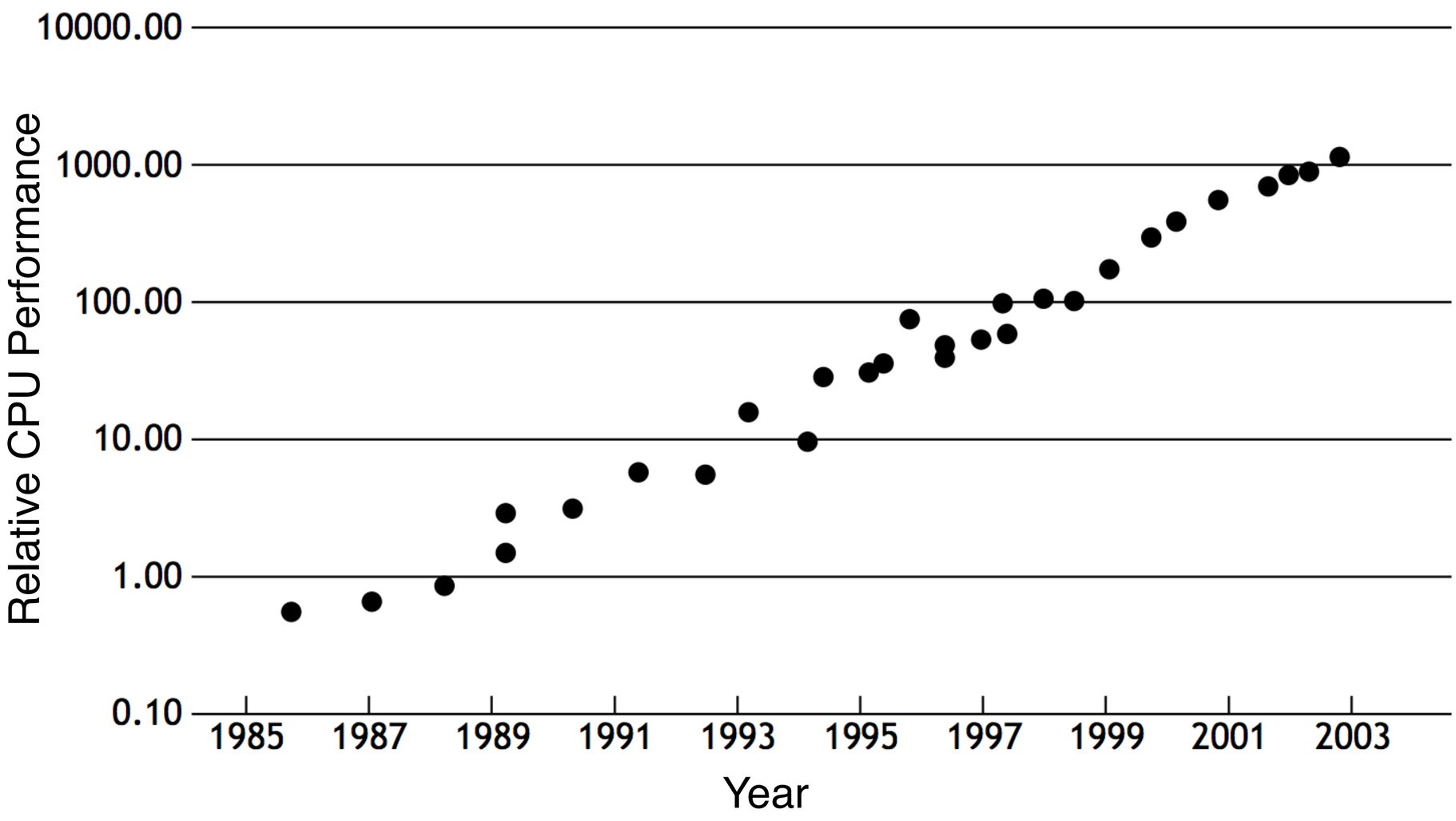
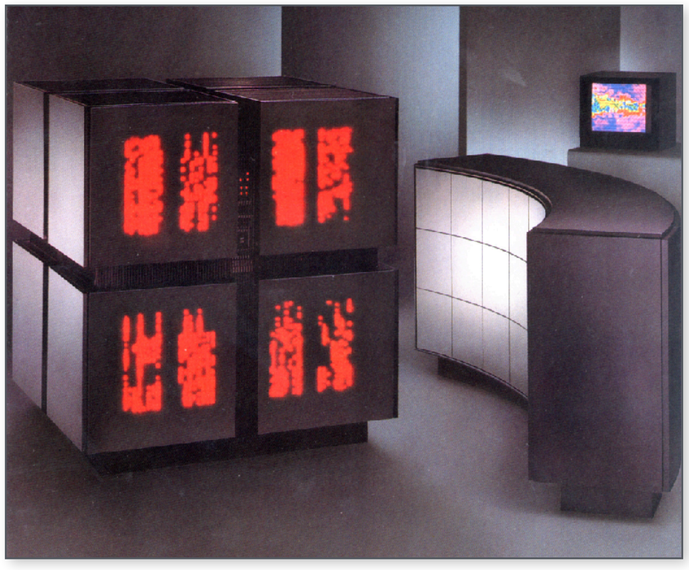


Image credit: Olukutun and Hammond, ACM Queue 2005

Why parallel processing? (80's, 90's, early 2000's)

The answer until ~15 years ago: to realize performance improvements that exceeded what CPU performance improvements could provide

For supercomputing applications



Thinking Machines (CM2) (1987)

65,536 1-bit processors + 2,048 32 bit FP processors

SGI Origin 2000 — 128 CPUs (1996)

Photo shows ASIC Blue Mountain supercomputer at Los Alamos (48 Origin 2000's)

For database applications

Sun Enterprise 10000 (circa 1997)

64 UltraSPARC-II processors

Until ~15 years ago: two significant reasons for processor performance improvement

1. Exploiting instruction-level parallelism (superscalar execution)
2. Increasing CPU clock frequency

What is a computer program?

```
int main(int argc, char** argv) {

    int x = 1;

    for (int i=0; i<10; i++) {
       x = x + x;
    }

    printf(“%d\n”, x);

    return 0;
}
```

Review: what is a program?

From a processor's perspective, a program is a sequence of instructions.

```
_main:
100000f10:    pushq    %rbp
100000f11:    movq %rsp, %rbp
100000f14:    subq $32, %rsp
100000f18:    movl $0, -4(%rbp)
100000f1f:    movl %edi, -8(%rbp)
100000f22:    movq %rsi, -16(%rbp)
100000f26:    movl $1, -20(%rbp)
100000f2d:    movl $0, -24(%rbp)
100000f34:    cmpl $10, -24(%rbp)
100000f38:    jge  23 <_main+0x45>
100000f3e:    movl -20(%rbp), %eax
100000f41:    addl -20(%rbp), %eax
100000f44:    movl %eax, -20(%rbp)
100000f47:    movl -24(%rbp), %eax
100000f4a:    addl $1, %eax
100000f4d:    movl %eax, -24(%rbp)
100000f50:    jmp  -33 <_main+0x24>
100000f55:    leaq 58(%rip), %rdi
100000f5c:    movl -20(%rbp), %esi
100000f5f:    movb $0, %al
100000f61:    callq    14
100000f66:    xorl %esi, %esi
100000f68:    movl %eax, -28(%rbp)
100000f6b:    movl %esi, %eax
100000f6d:    addq $32, %rsp
100000f71:    popq %rbp
100000f72:    retq
```

Review: what does a processor do?

It runs programs!

Processor executes instruction referenced by the program counter (PC)

(executing the instruction will modify machine state: contents of registers, memory, CPU state, etc.)

Move to next instruction …

Then execute it…

And so on…

PC

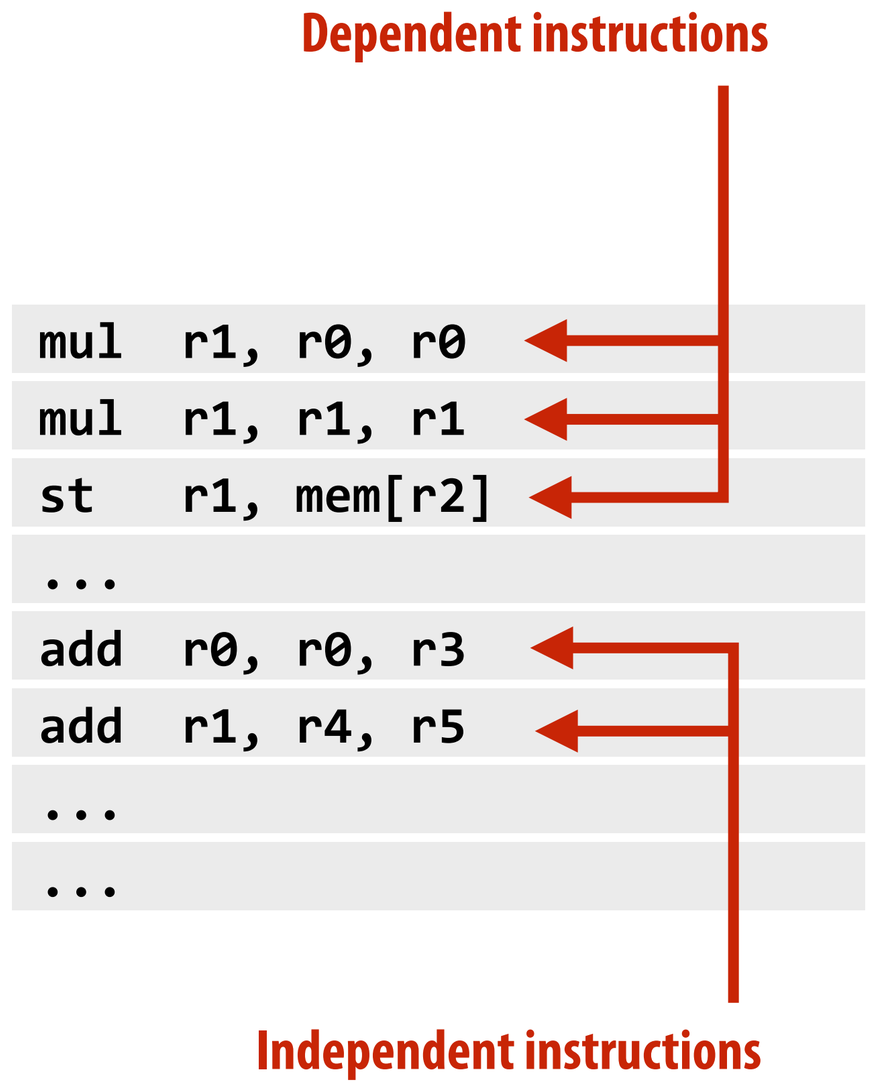
```
_main:
100000f10:    pushq    %rbp
100000f11:    movq %rsp, %rbp
100000f14:    subq $32, %rsp
100000f18:    movl $0, -4(%rbp)
100000f1f:    movl %edi, -8(%rbp)
100000f22:    movq %rsi, -16(%rbp)
100000f26:    movl $1, -20(%rbp)
100000f2d:    movl $0, -24(%rbp)
100000f34:    cmpl $10, -24(%rbp)
100000f38:    jge  23 <_main+0x45>
100000f3e:    movl -20(%rbp), %eax
100000f41:    addl -20(%rbp), %eax
100000f44:    movl %eax, -20(%rbp)
100000f47:    movl -24(%rbp), %eax
100000f4a:    addl $1, %eax
100000f4d:    movl %eax, -24(%rbp)
100000f50:    jmp  -33 <_main+0x24>
100000f55:    leaq 58(%rip), %rdi
100000f5c:    movl -20(%rbp), %esi
100000f5f:    movb $0, %al
100000f61:    callq    14
100000f66:    xorl %esi, %esi
100000f68:    movl %eax, -28(%rbp)
100000f6b:    movl %esi, %eax
100000f6d:    addq $32, %rsp
100000f71:    popq %rbp
100000f72:    retq
```

Instruction level parallelism (ILP)

- Processors did in fact leverage parallel execution to make programs run faster, it was just invisible to the programmer

- Instruction level parallelism (ILP)
  - Idea: Instructions must <u>appear</u> to be executed in program order. BUT <u>independent</u> instructions can be executed simultaneously by a processor without impacting program correctness

  - <u>Superscalar execution</u>: processor dynamically finds independent instructions in an instruction sequence and executes them in parallel



ILP example

$$a = x*x + y*y + z*z$$

Consider the following program:

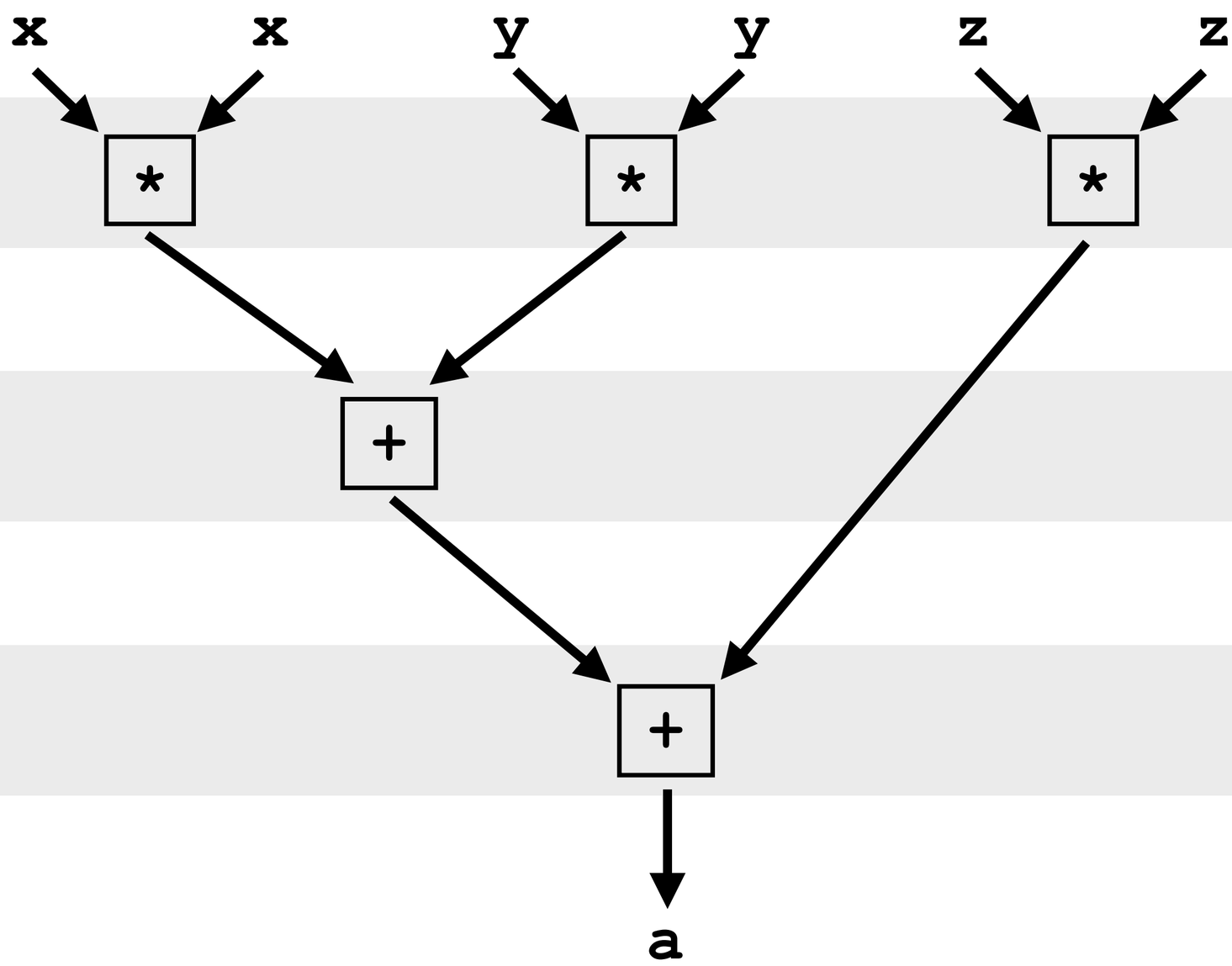
```
// assume r0=x, r1=y, r2=z

mul r0, r0, r0
mul r1, r1, r1
mul r2, r2, r2
add r0, r0, r1
add r3, r0, r2

// now r3 stores value of program variable ‘a’
```

This program has five instructions, so it will take five clocks to execute, correct?

Can we do better?

ILP example

```
a = x*x + y*y + z*z
```



ILP = 3

ILP = 1

ILP = 1

Superscalar execution

$$a = x*x + y*y + z*z$$

```
// assume r0=x, r1=y, r2=z

1.  mul r0, r0, r0
2.  mul r1, r1, r1
3.  mul r2, r2, r2
4.  add r0, r0, r1
5.  add r3, r0, r2

// r3 stores value of variable ‘a’
```

Superscalar execution: processor automatically finds independent instructions in an instruction sequence and executes them in parallel on multiple execution units!

In this example: instructions 1, 2, and 3 can be executed in parallel (on a superscalar processor that determines that the lack of dependencies exists)

But instruction 4 must come after instructions 1 and 2

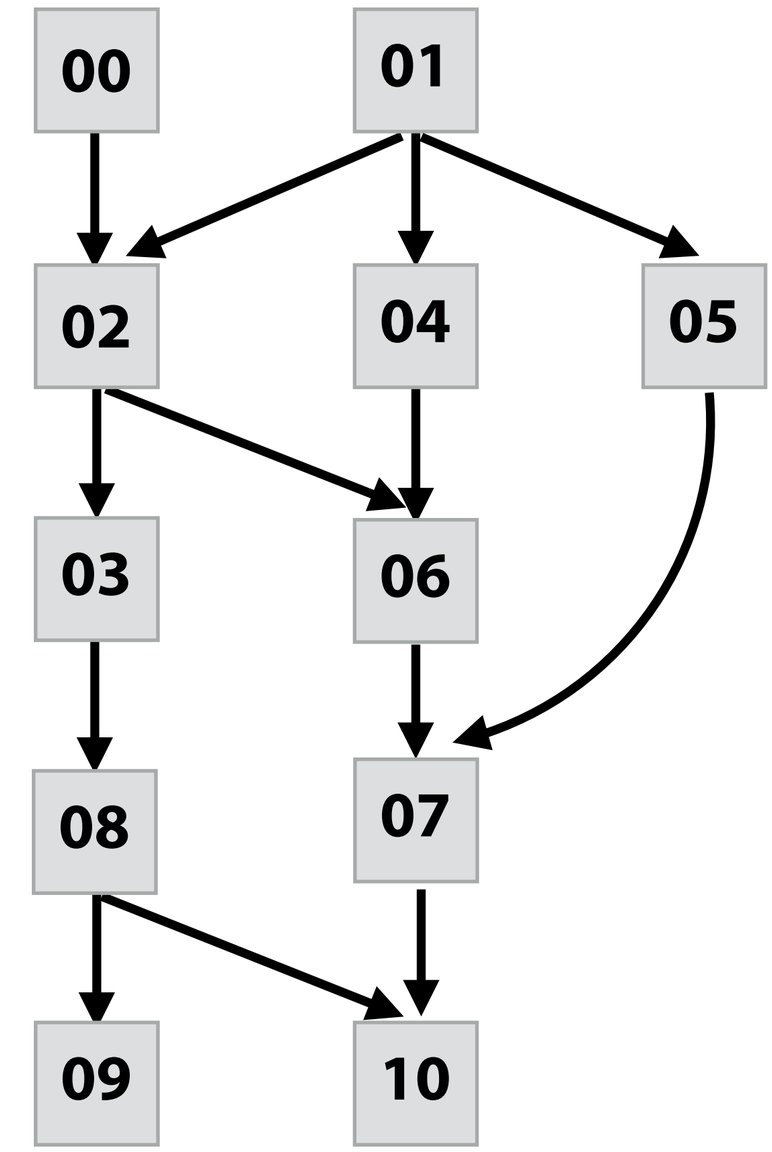
And instruction 5 must come after instruction 4

A more complex example

Program (sequence of instructions)

PC	Instruction	
00	a = 2	
01	b = 4	
		value during execution
02	tmp2 = a + b	// 6
03	tmp3 = tmp2 + a	// 8
04	tmp4 = b + b	// 8
05	tmp5 = b * b	// 16
06	tmp6 = tmp2 + tmp4	// 14
07	tmp7 = tmp5 + tmp6	// 30
08	if (tmp3 > 7)	
09	print tmp3	
	else	
10	print tmp7	

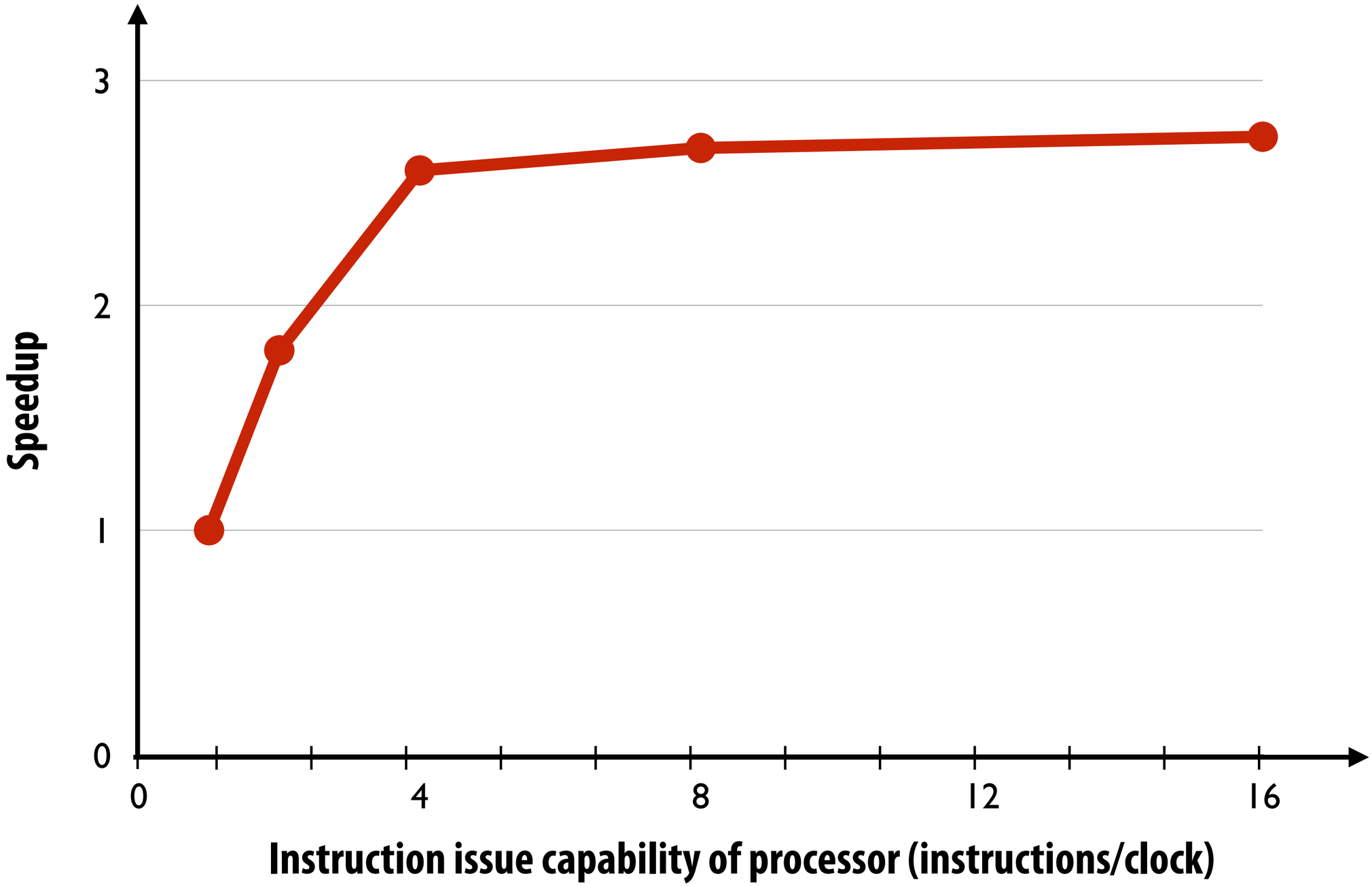
Instruction dependency graph



What does it mean for a superscalar processor to "respect program order"?

Diminishing returns of superscalar execution

Most available ILP is exploited by a processor capable of issuing four instructions per clock (Little performance benefit from building a processor that can issue more)



Source: Culler & Singh (data from Johnson 1991)

ILP tapped out + end of frequency scaling

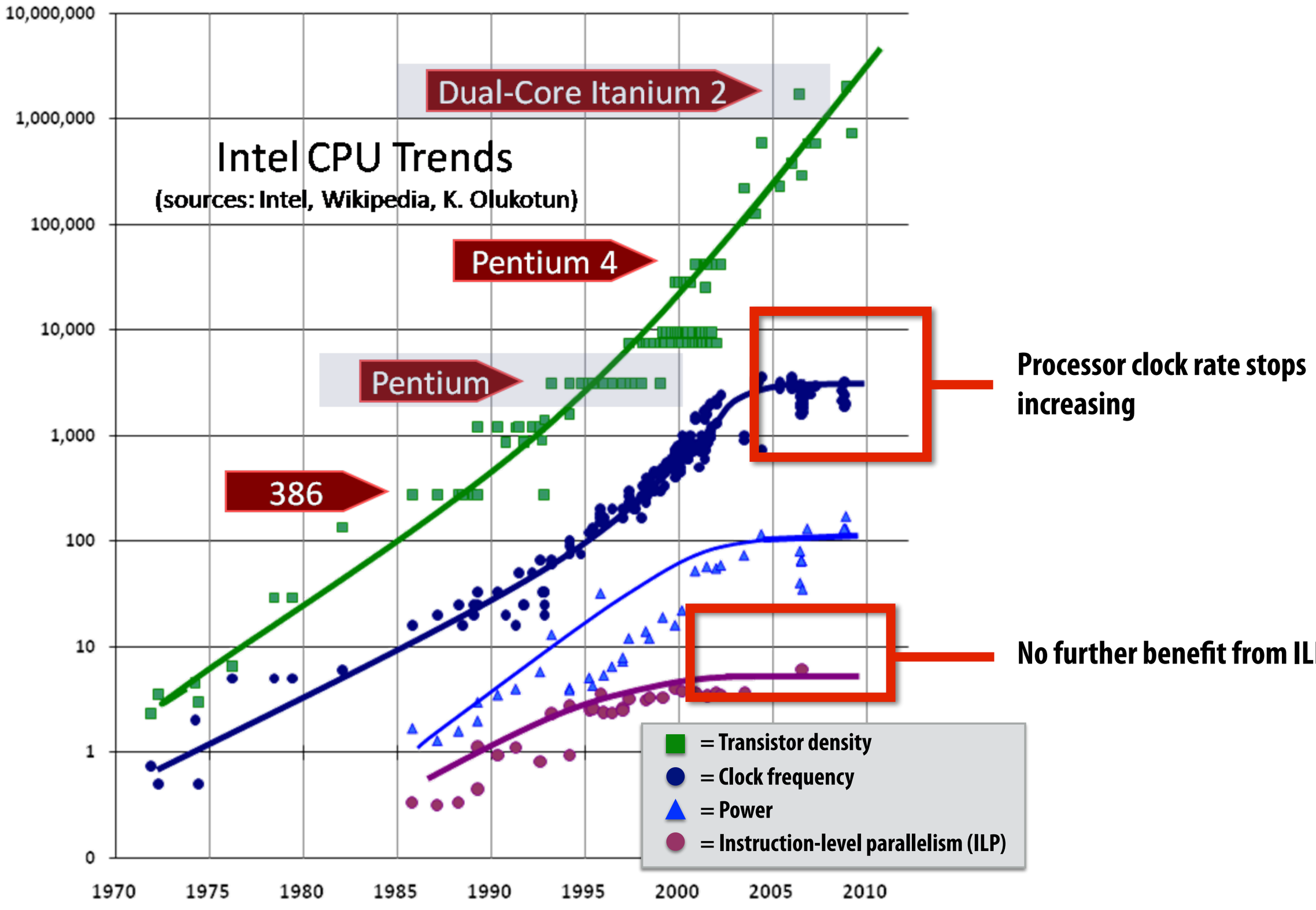


Image credit: "The free Lunch is Over" by Herb Sutter, Dr. Dobbs 2005

The "power wall"

Power consumed by a transistor:

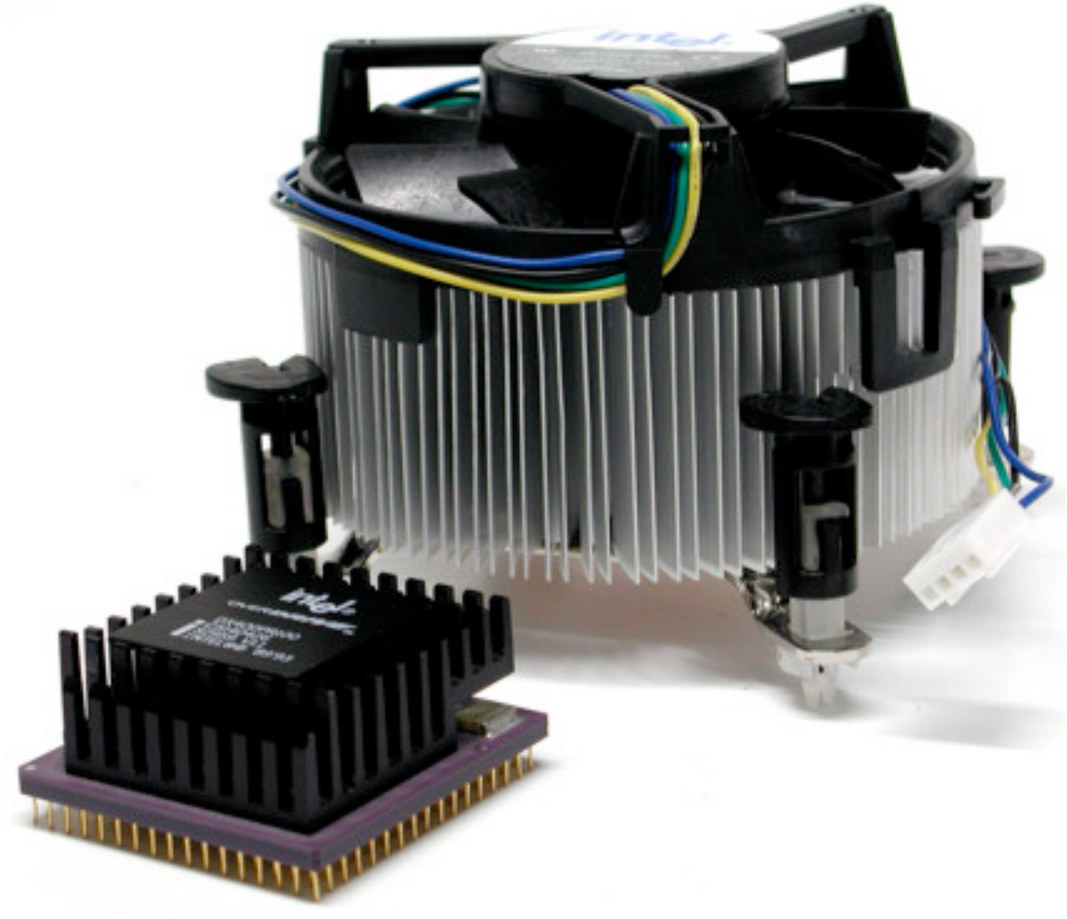
Dynamic power $\propto$ capacitive load $\times$ voltage^2 $\times$ frequency

Static power: transistors burn power even when inactive due to leakage

High power = high heat

Power is a critical design constraint in modern processors

	TDP
Intel Core i7 (in this laptop):	45W
Intel Core i7 2700K (fast desktop CPU):	95W
NVIDIA Titan V GPU	250W
Mobile phone processor	1/2 - 2W
World's fastest supercomputer	megawatts
Standard microwave oven	700W



Source: Intel, NVIDIA, Wikipedia, Top500.org

Power draw as a function of frequency

Dynamic power $\propto$ capacitive load $\times$ voltage^2 $\times$ frequency

Static power: transistors burn power even when inactive due to leakage

Maximum allowed frequency determined by processor's core voltage

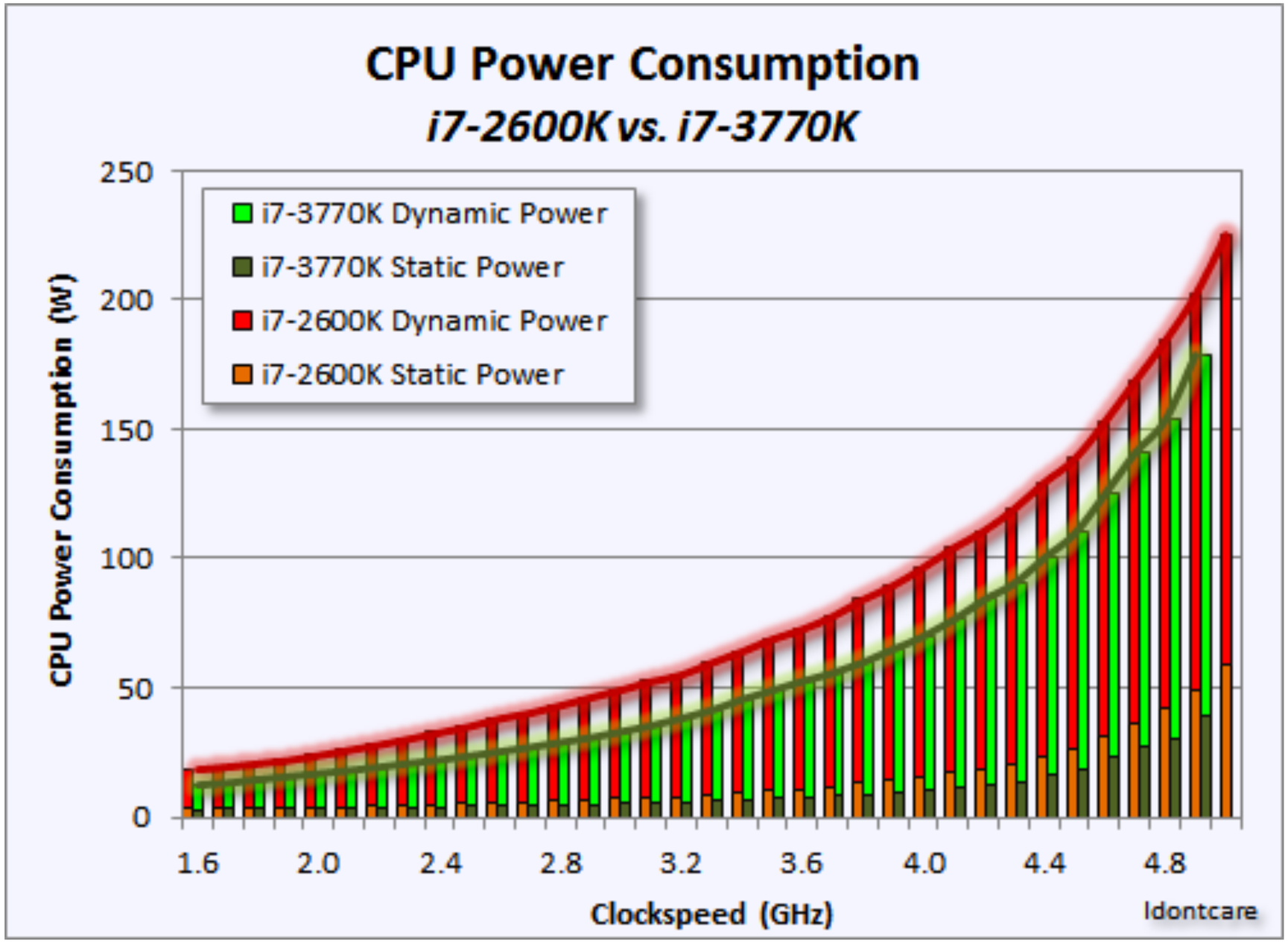


Image credit: "Idontcare": posted at: http://forums.anandtech.com/showthread.php?t=2281195

Single-core performance scaling

The rate of single-instruction stream performance scaling has decreased (almost to zero)

1. Frequency scaling limited by power

2. ILP scaling tapped out

Architects are now building faster processors by adding more execution units that run in parallel

(Or units that are specialized for a specific task (like graphics, or audio/video playback)

Software must be written to be parallel to see performance gains. No more free lunch for software developers!

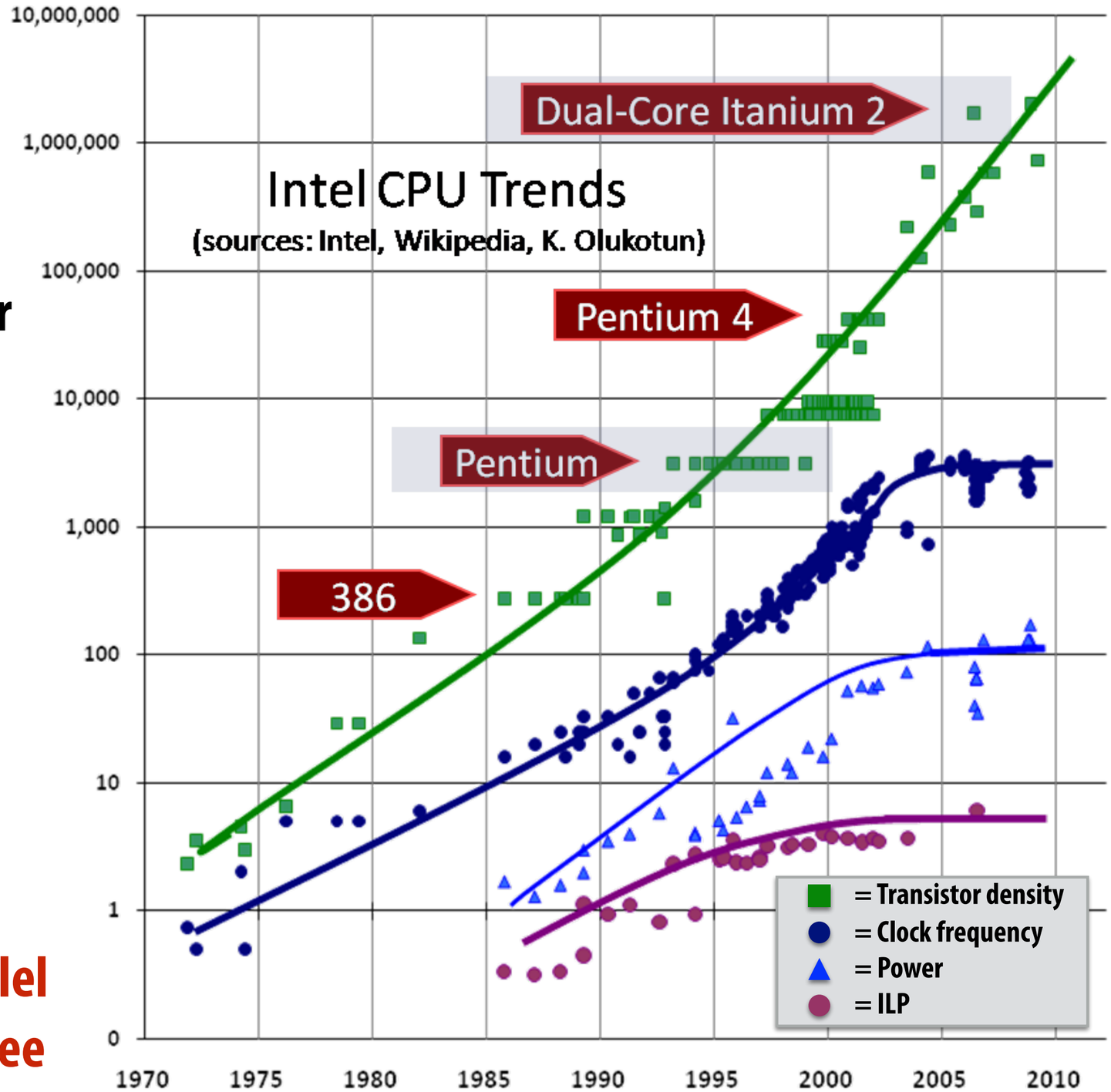


Image credit: "The free Lunch is Over" by Herb Sutter, Dr. Dobbs 2005

From the New York Times

Intel's Big Shift After Hitting Technical Wall

The warning came first from a group of hobbyists that tests the speeds of computer chips. This year, the group discovered that the Intel Corporation's newest microprocessor was running slower and hotter than its predecessor.

What they had stumbled upon was a major threat to Intel's longstanding approach to dominating the semiconductor industry - relentlessly raising the clock speed of its chips.

Then two weeks ago, Intel, the world's largest chip maker, publicly acknowledged that it had hit a "thermal wall" on its microprocessor line. As a result, the company is changing its product strategy and disbanding one of its most advanced design groups. Intel also said that it would abandon two advanced chip development projects, code-named Tejas and Jayhawk.

Now, Intel is embarked on a course already adopted by some of its major rivals: obtaining more computing power by stamping multiple processors on a single chip rather than straining to increase the speed of a single processor.

…

John Markoff, New York Times, May 17, 2004

Recap: why parallelism?

- **The answer up until ~15 years ago**
  - To realize performance improvements that exceeded what CPU performance improvements could provide

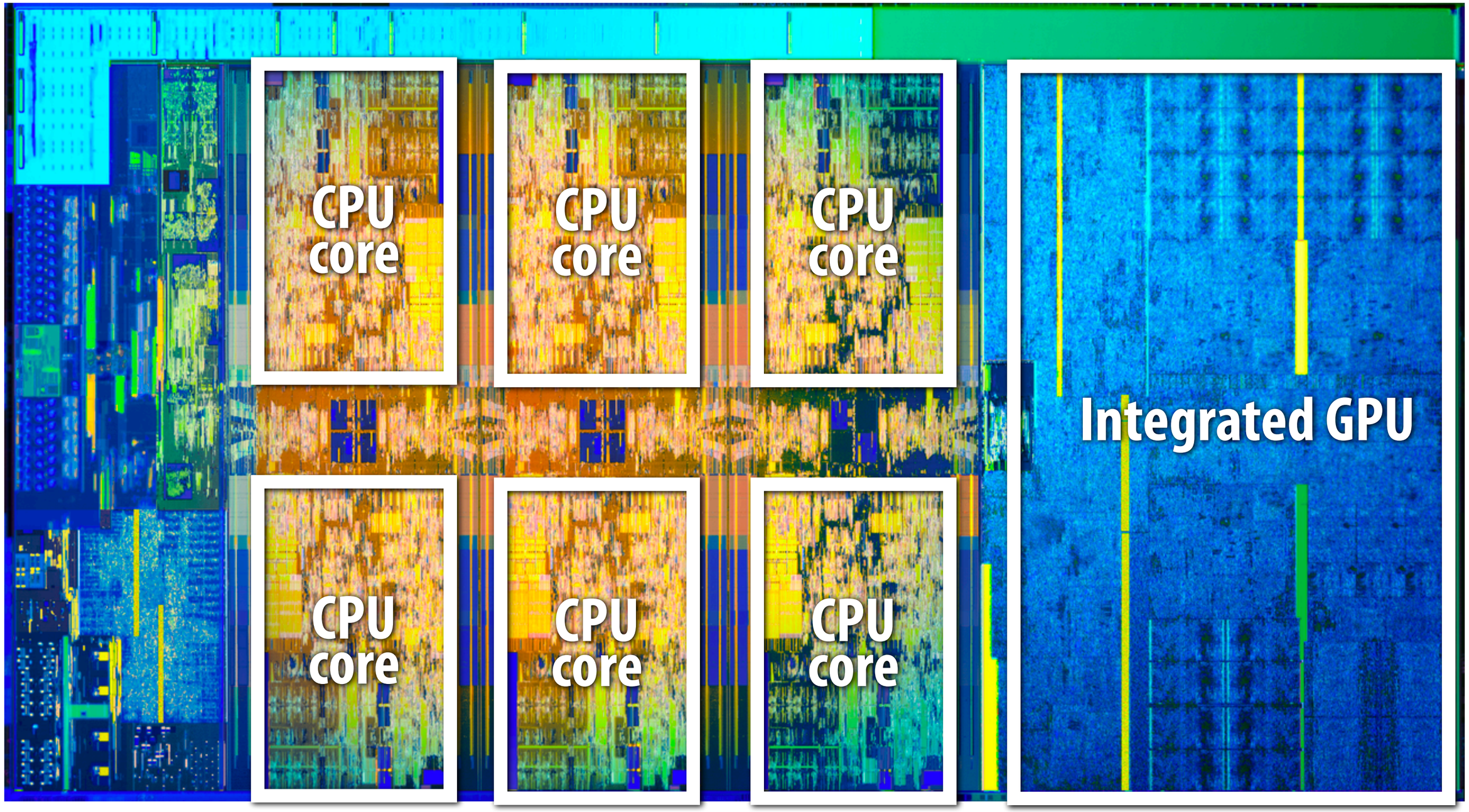
    (specifically, in the early 2000's, what clock frequency scaling could provide)
  - Because if you just waited until next year, your code would run faster on the next generation CPU

- **The answer today:**
  - Because it is <u>the primary way</u> to achieve significantly higher application performance for the foreseeable future *

* We'll revisit this comment later in the heterogeneous processing lecture

Intel Coffee Lake (2017) (aka "8th generation Core i7")

Six-core CPU + multi-core GPU integrated on one chip



One thing you will learn in this course

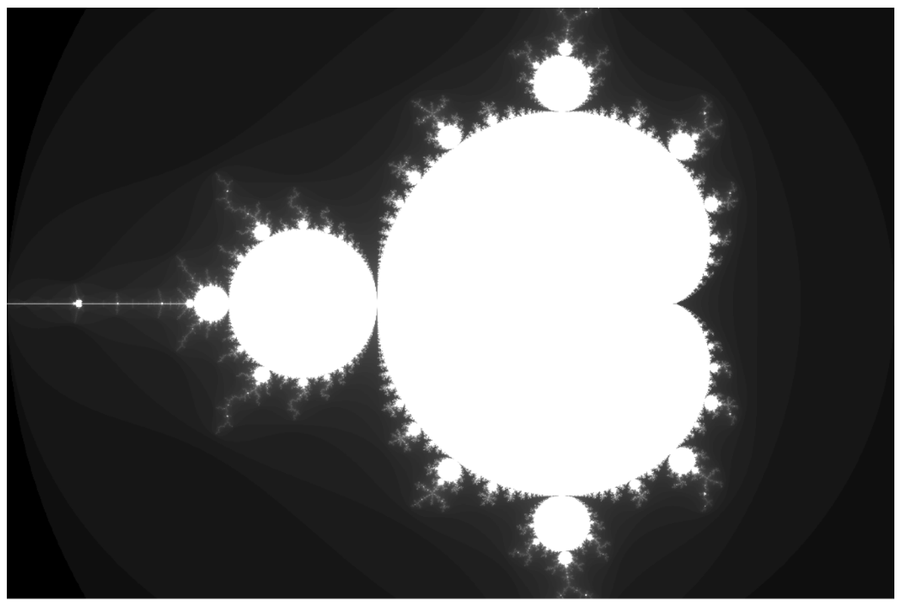
- How to write code that efficiently uses the resources in a modern multi-core CPU
- Example: assignment 1 (coming up!)
  - My laptop: quad-core Intel Core i7 CPU
    - Four CPU cores
    - AVX SIMD vector instructions + hyper-threading

We'll talk about these terms next time!

  - Baseline: single-threaded C program compiled with -O3
  - Parallelized program that uses all parallel execution resources on this CPU…

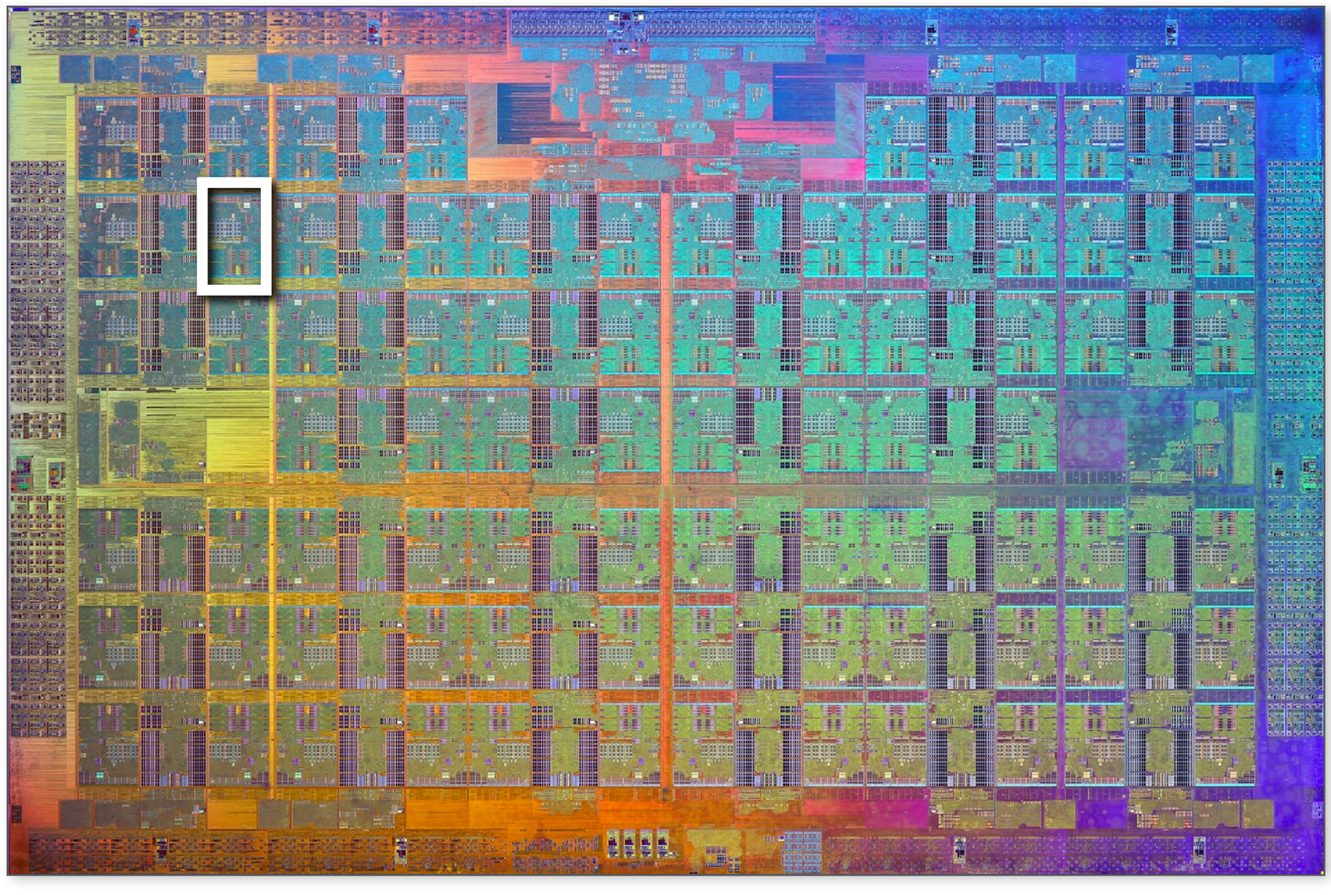
~32x faster!

And ~41x faster on another example.



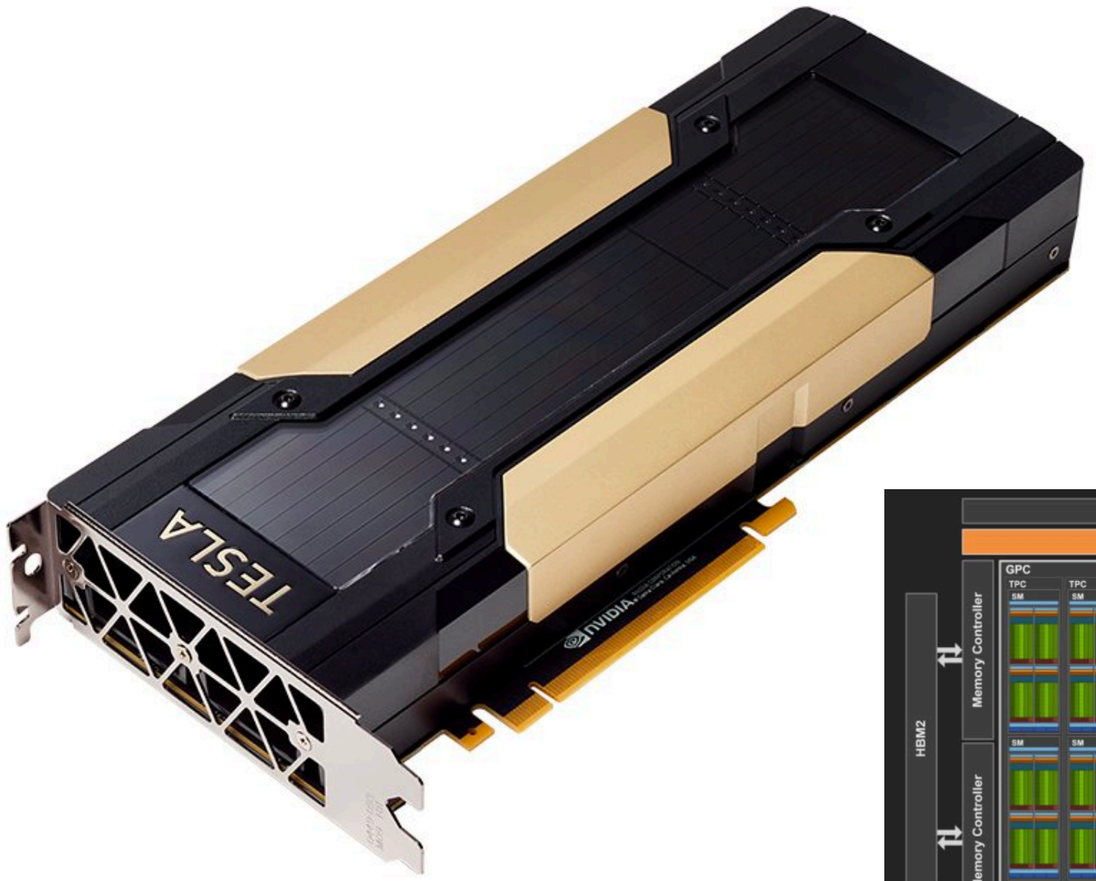
Intel Xeon Phi 7290 (2016)

72 cores (1.5 Ghz)



NVIDIA Tesla V100 GPU (2017)

5,376 fp32 units grouped into 84 major processing blocks



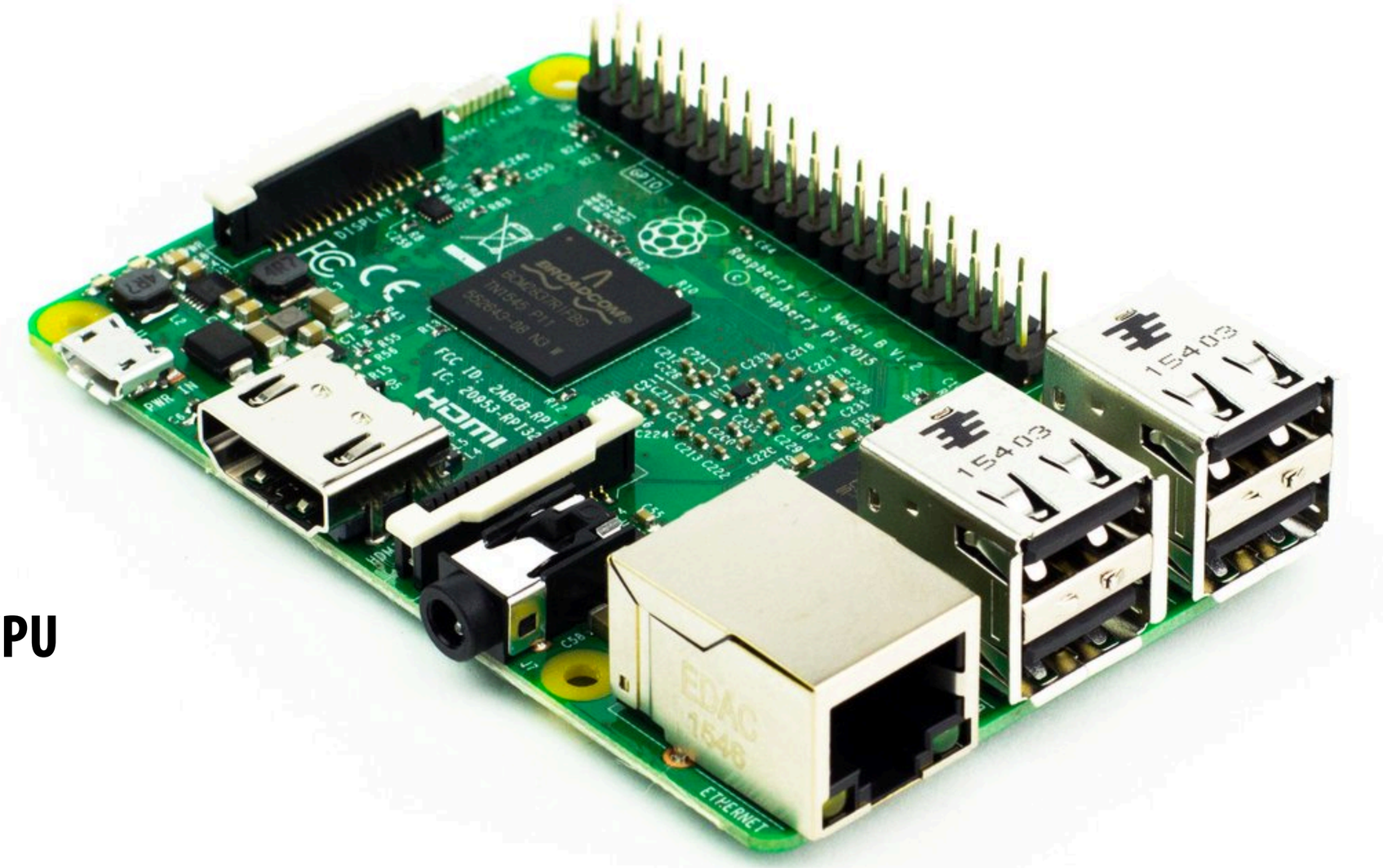
Supercomputing

- Today: combinations of multi-core CPUs + GPUs
- Oak Ridge National Laboratory: Summit (currently #2 supercomputer in world)
  - 9,216 x 22-core IBM Power9 CPUs + 27,648 NVIDIA Volta GPUs

Mobile parallel processing

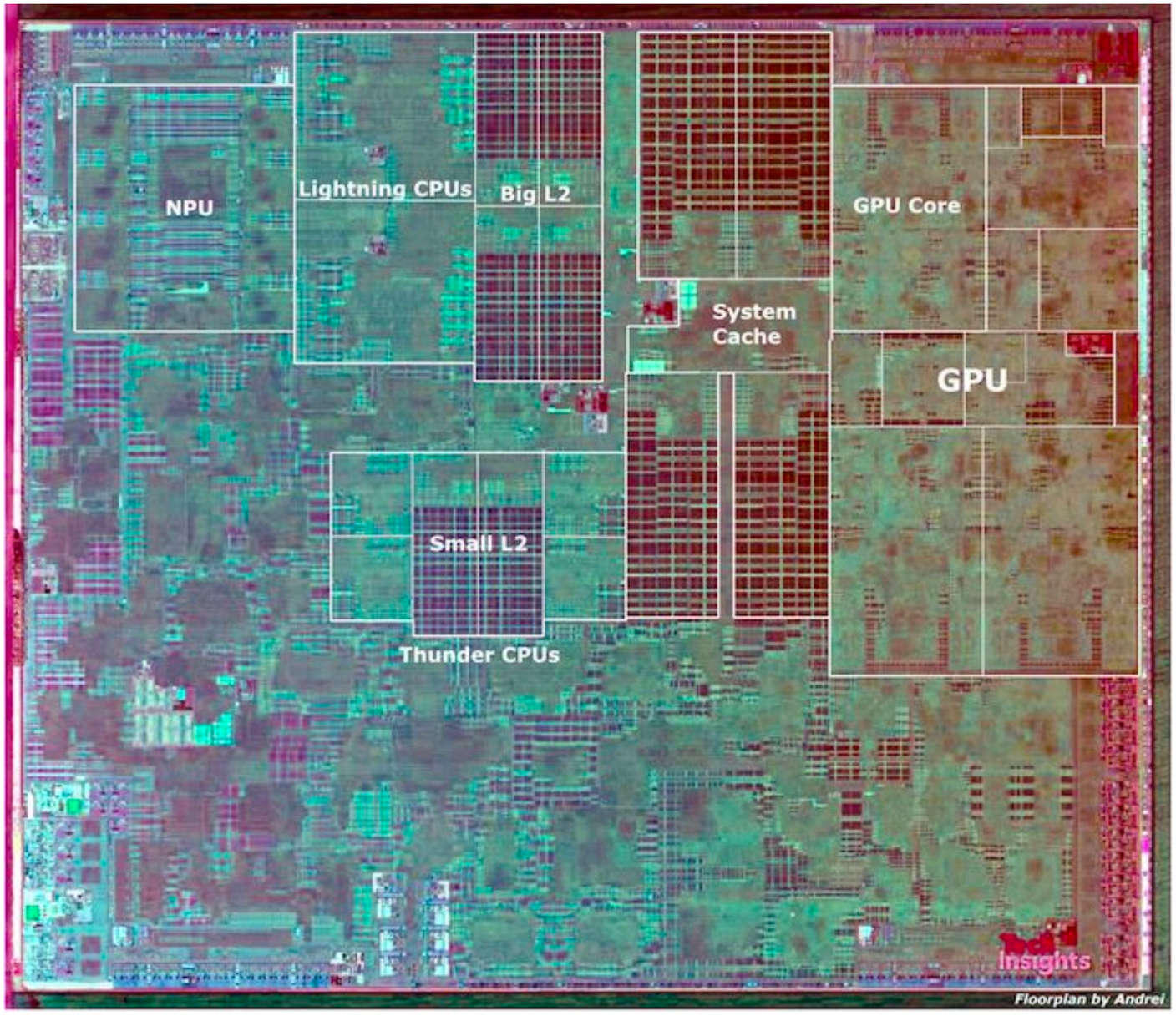
Raspberry Pi 3

Quad-core ARM A53 CPU



Mobile parallel processing

Power constraints heavily influence the design of mobile systems



Apple A13 Bionic (in iPhone 11)

2 "big" CPU cores +
4 "small" CPU cores +

Apple-designed multi-core GPU +
Image processor +
Neural Engine for DNN acceleration +
Motion processor

Image Credit: Anandtech / TechInsights Inc.

Parallel + specialized HW

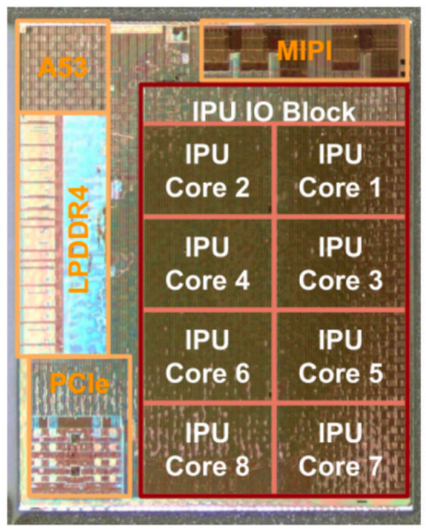
- Achieving high efficiency will be a key theme in this class

- We will discuss how modern systems are not only parallel, but also specialize processing units to achieve high levels of power efficiency

Another recent smartphone

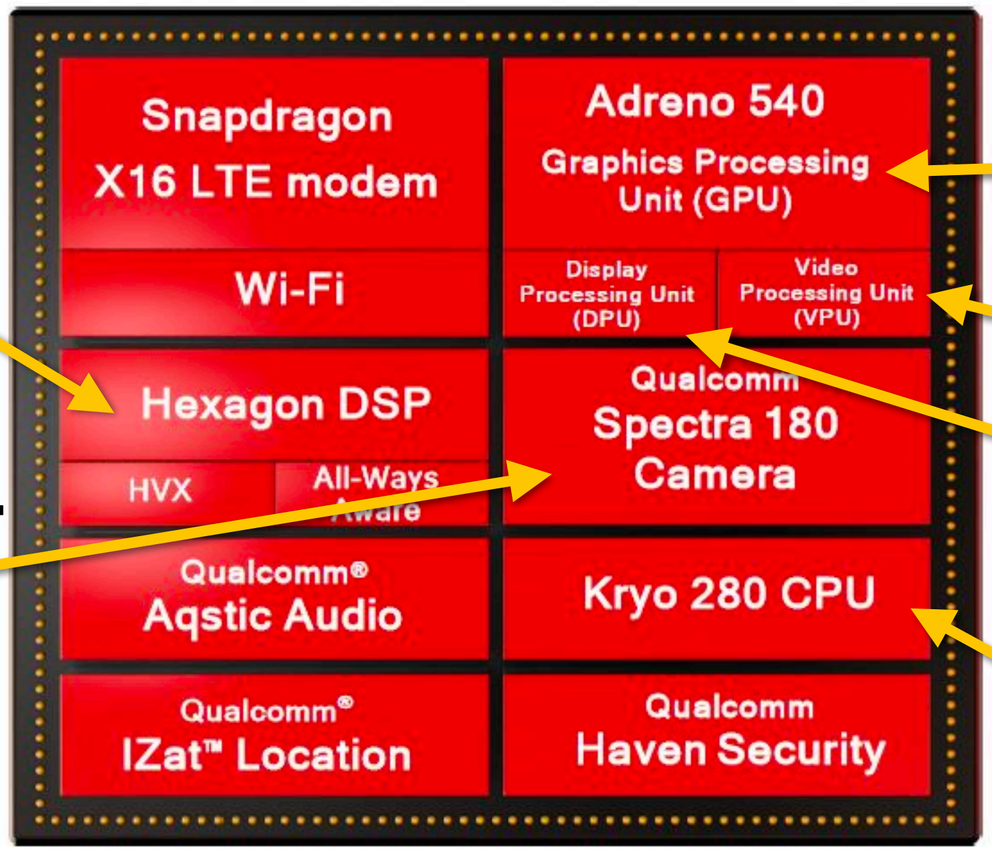
Google Pixel 2 Phone:
Qualcomm Snapdragon 835 SoC + Google Visual Pixel Core

Visual Pixel Core
Programmable image processor and DNN accelerator



"Hexagon" Programmable DSP
data-parallel multi-media processing

Image Signal Processor
ASIC for processing camera sensor pixels



Multi-core GPU
(3D graphics, OpenCL data-parallel compute)

Video encode/decode ASIC

Display engine
(compresses pixels for transfer to high-res screen)

Multi-core ARM CPU
4 "big cores" + 4 "little cores"

Datacenter-scale applications

Google TPU pods

TPU = Tensor Processing Unit: specialized processor for ML computations

Image Credit: TechInsights Inc.

Summary

- Today, single-thread-of-control performance is improving very slowly
  - To run programs significantly faster, programs must utilize multiple processing elements or specialized processing
  - Which means you need to know how to write parallel code

- Writing parallel programs can be challenging
  - Requires problem partitioning, communication, synchronization
  - Knowledge of machine characteristics is important

- I suspect you will find that modern computers have tremendously more processing power than you might realize, if you just use it!

Reminders

- This is going to be an unpredictable quarter
- Everyone students and staff are going to need to help each other out
- We expect that students are going to require different support based on circumstances

Prof. Kayvon

- But that doesn't mean expectations aren't high
- Let's work together to have a great experience
- Welcome to CS149!

Prof. Olukotun

- And we remind you to take your mask wearing seriously when around others.