

Lecture 1:

Why Parallelism? Why Efficiency?

**Parallel Computer Architecture and Programming
CMU 15-418/15-618, Fall 2018**

Hi!



Prof. Mowry



Aayush



Frank



Ting



Prof. Railing



Haoran



Hao

What will you be doing in this course?

Assignments

- **Four programming assignments**

- **First assignment is done individually, the rest will be done in pairs**
- **Each uses a different parallel programming environment**



Assignment 1: ISPC programming on Intel quad-core CPU (and Xeon Phi)



Assignment 2: CUDA programming on NVIDIA GPUs



Assignment 3: Parallel Programming via a Shared-Address Space Model



Assignment 4: Parallel Programming via a Message Passing Model

If you are on the Wait List

- We will hand out Assignment 1 later this week
- Our algorithm for filling the K remaining slots in the class:
 - the first K students on the Wait List who hand in Assignment 1 and receive an A on it are enrolled in the class

Exams

- **We will have two midterm-style exams**
 - **Each covers roughly half of the course material**
 - **Closed-book, closed-notes**
- **No final exam**
 - **We use the final exam slot for our project poster session**

Final project

- 6-week self-selected final project
 - Performed in groups (by default, 2 people per group)
 - Start thinking about your project ideas TODAY!
 - Poster session during the final exam slot
-
- Check out last semester's projects:

<http://15418.courses.cs.cmu.edu/fall2017/article/10>

Participation Grade: Take-Home Quizzes

- **We will have 2-4 take-home quizzes**
 - **Goal: help you prepare for the exams**
 - **You must complete the quiz on your own**
 - **We will grade your work to give you feedback, but only a participation grade will go into the gradebook**

Participation Grade: In-Class Mini-Quizzes

- **In most lectures, we will have a simple in-class (online) quiz**
- **The quizzes should be easy**
 - **the goal is just to demonstrate that you are paying attention in class**
- **They also give us feedback on what the class is understanding**
- **Grace budget:**
 - **Full credit for this portion of the participation grade if you complete 70% of the in-class quizzes**

Grades

40% Programming assignments (4)

30% Exams (2)

25% Final project

5% Participation (take-home and in-class quizzes)

Each student (or group) gets up to five late days on programming assignments (see syllabus for details)

Getting started

- **Pay attention to Piazza posts**

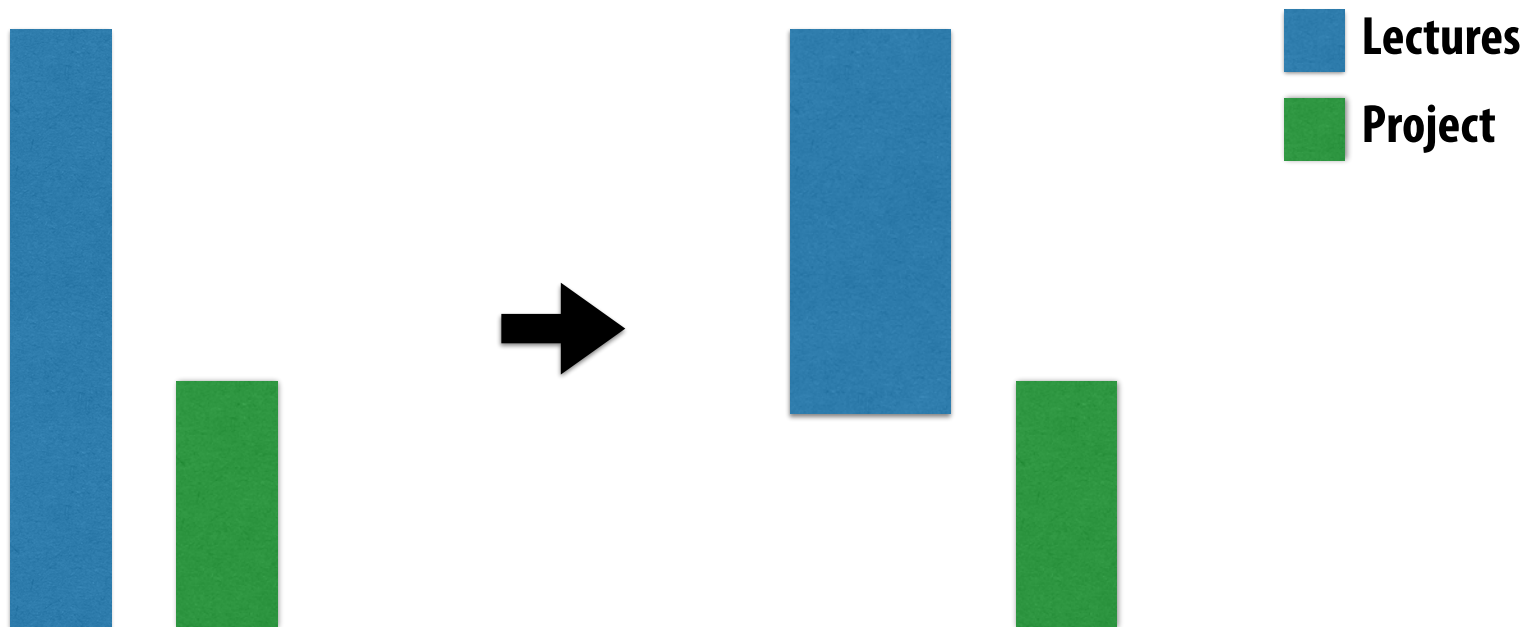
- <http://piazza.com/cmu/fall2018/1541815618>

- **Textbook**

- **There is no course textbook, but please see web site for suggested references**

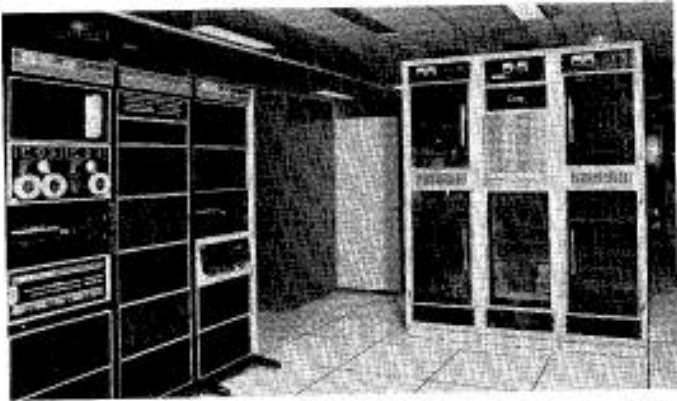
Regarding the class meeting times

- We meet 3 days a week (MWF) for the first 2/3 of the semester
- Same content as 2 days a week over a full semester, but two major advantages this way:
 - you are better prepared to do an interesting project
 - more time to focus on your project



A Brief History of Parallel Computing

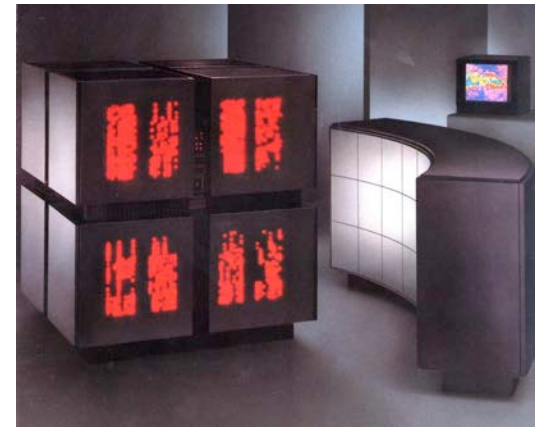
- Initial Focus (starting in 1970s): “Supercomputers” for Scientific Computing



C.mmp at CMU (1971)
16 PDP-11 processors



Cray XMP (circa 1984)
4 vector processors



Thinking Machines CM-2 (circa 1987)
65,536 1-bit processors +
2048 floating-point co-processors



SGI UV 1000cc-NUMA (today)
4096 processor cores

← **Blacklight at the Pittsburgh
Supercomputer Center**

A Brief History of Parallel Computing

- Initial Focus (starting in 1970s): “Supercomputers” for Scientific Computing
- Another Driving Application (starting in early ‘90s): **Databases**



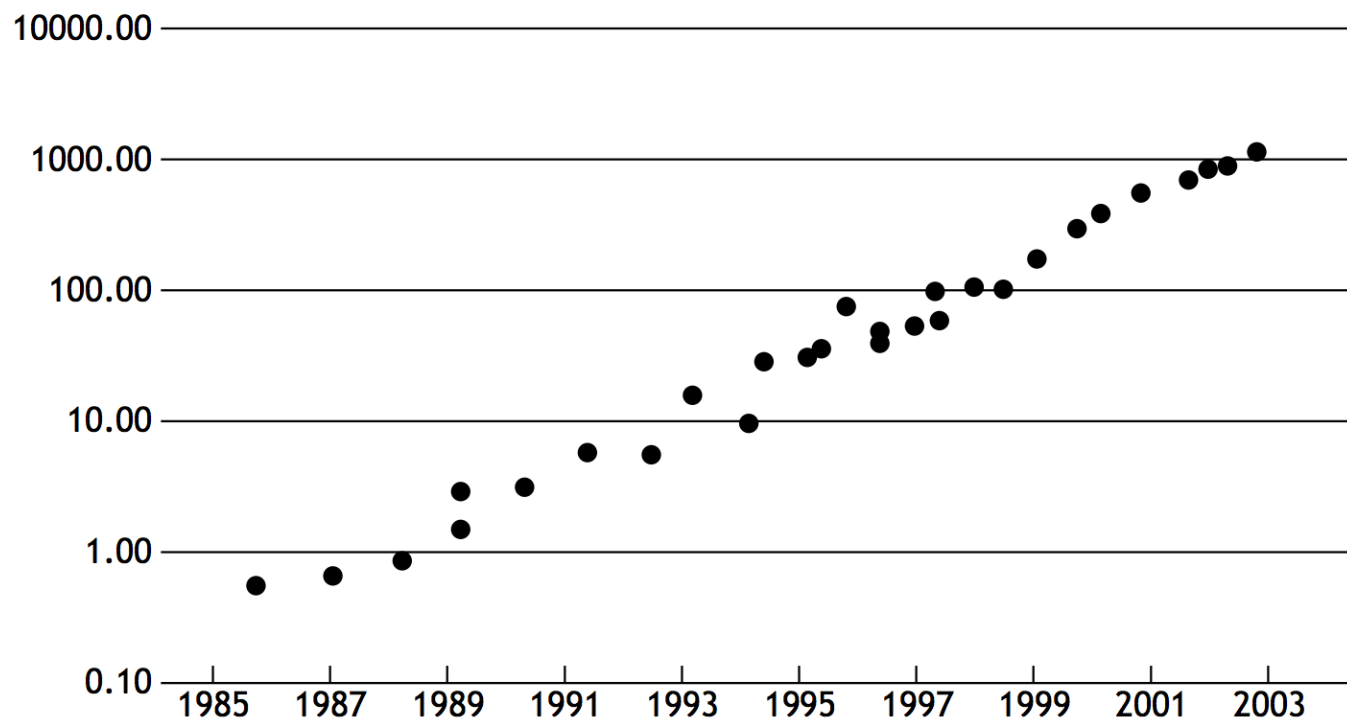
Sun Enterprise 10000 (circa **1997**)
16 UltraSPARC-II processors



Oracle Supercluster M6-32 (**today**)
32 SPARC M2 processors

Setting Some Context

- Before we continue our multiprocessor story, let's pause to consider:
 - Q: what had been happening with single-processor performance?
- A: since forever, they had been getting **exponentially faster**
 - Why?



A Brief History of Processor Performance

■ Wider data paths

- 4 bit → 8 bit → 16 bit → 32 bit → 64 bit

■ More efficient pipelining

- e.g., 3.5 Cycles Per Instruction (CPI) → 1.1 CPI

■ Exploiting instruction-level parallelism (ILP)

- “superscalar” processing: e.g., issue up to 4 instructions/cycle

■ Faster clock rates

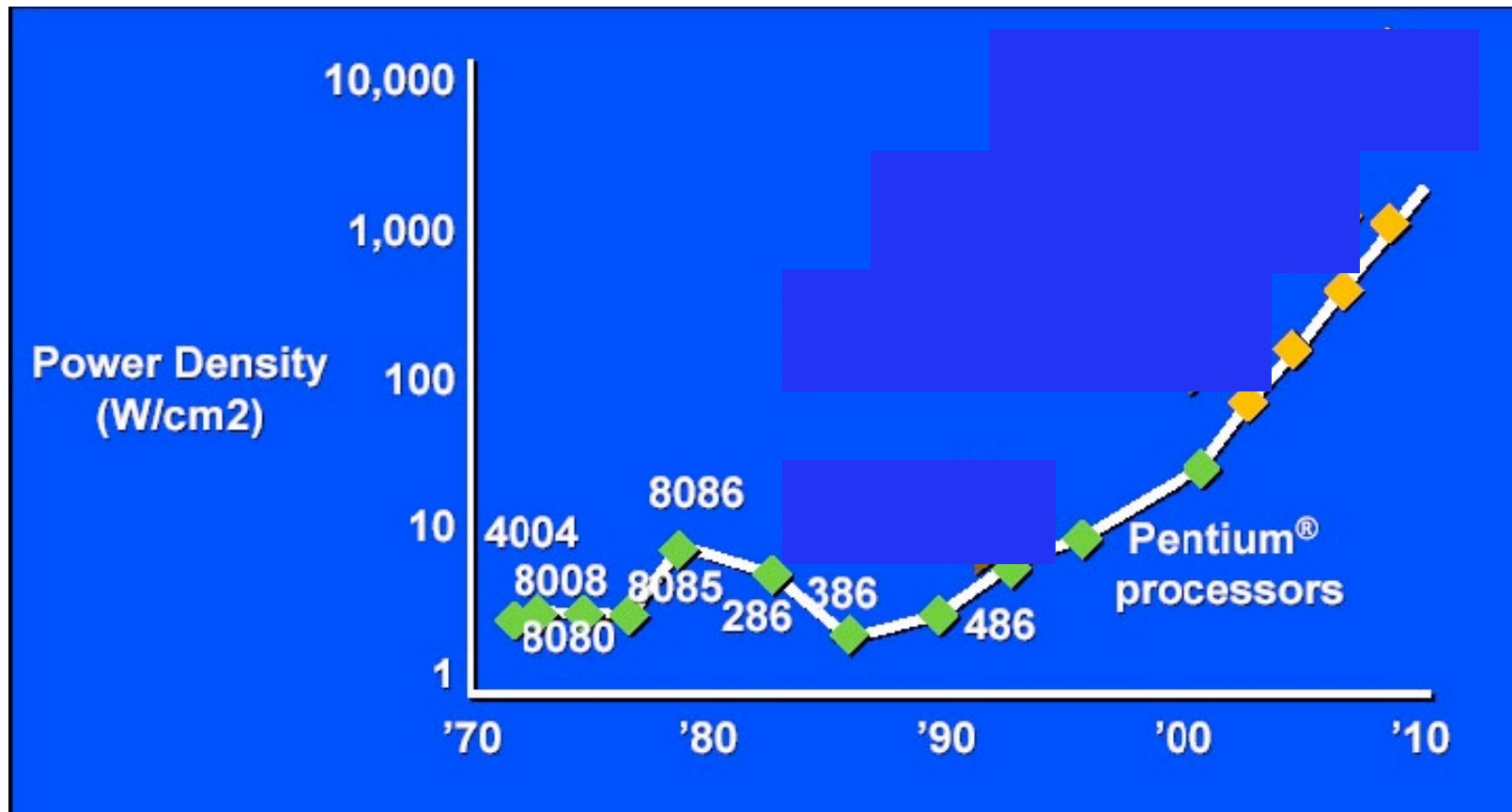
- e.g., 10 MHz → 200 MHz → 3 GHz

■ During the 80s and 90s: large exponential performance gains

- and then...

A Brief History of Parallel Computing

- Initial Focus (starting in 1970s): “Supercomputers” for Scientific Computing
- Another Driving Application (starting in early '90s): Databases
- Inflection point in 2004: Intel hits the **Power Density Wall**



Pat Gelsinger, ISSCC 2001

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.

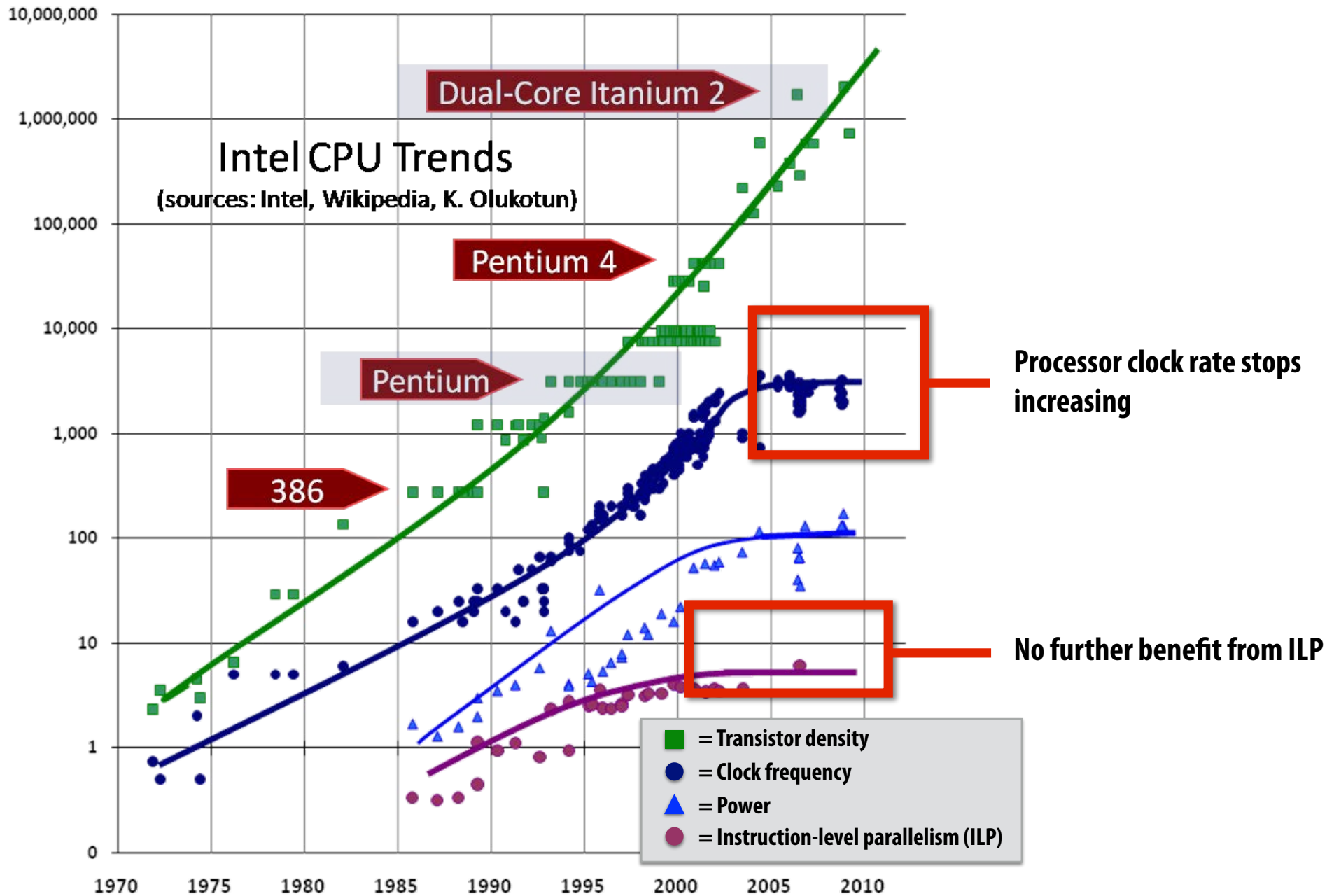
Then two weeks ago, Intel, the world's largest chip maker, publicly acknowledged that it had hit a "thermal wall" on its microprocessor line. As a result, the company is changing its product strategy and disbanding one of its most advanced design groups. Intel also said that it would abandon two advanced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

ILP tapped out + end of frequency scaling



Programmer's Perspective on Performance

Question: **How do you make your program run faster?**

Answer before 2004:

- **Just wait 6 months, and buy a new machine!**
- *(Or if you're really obsessed, you can learn about parallelism.)*

Answer after 2004:

- **You need to write parallel software.**

Parallel Machines Today

Examples from Apple's product line:



Mac Pro

12 Intel Xeon E5 cores



iPad Retina

2 Swift cores



iPhone 6s

2 A9 cores



iMac

12 Intel Xeon E5 cores



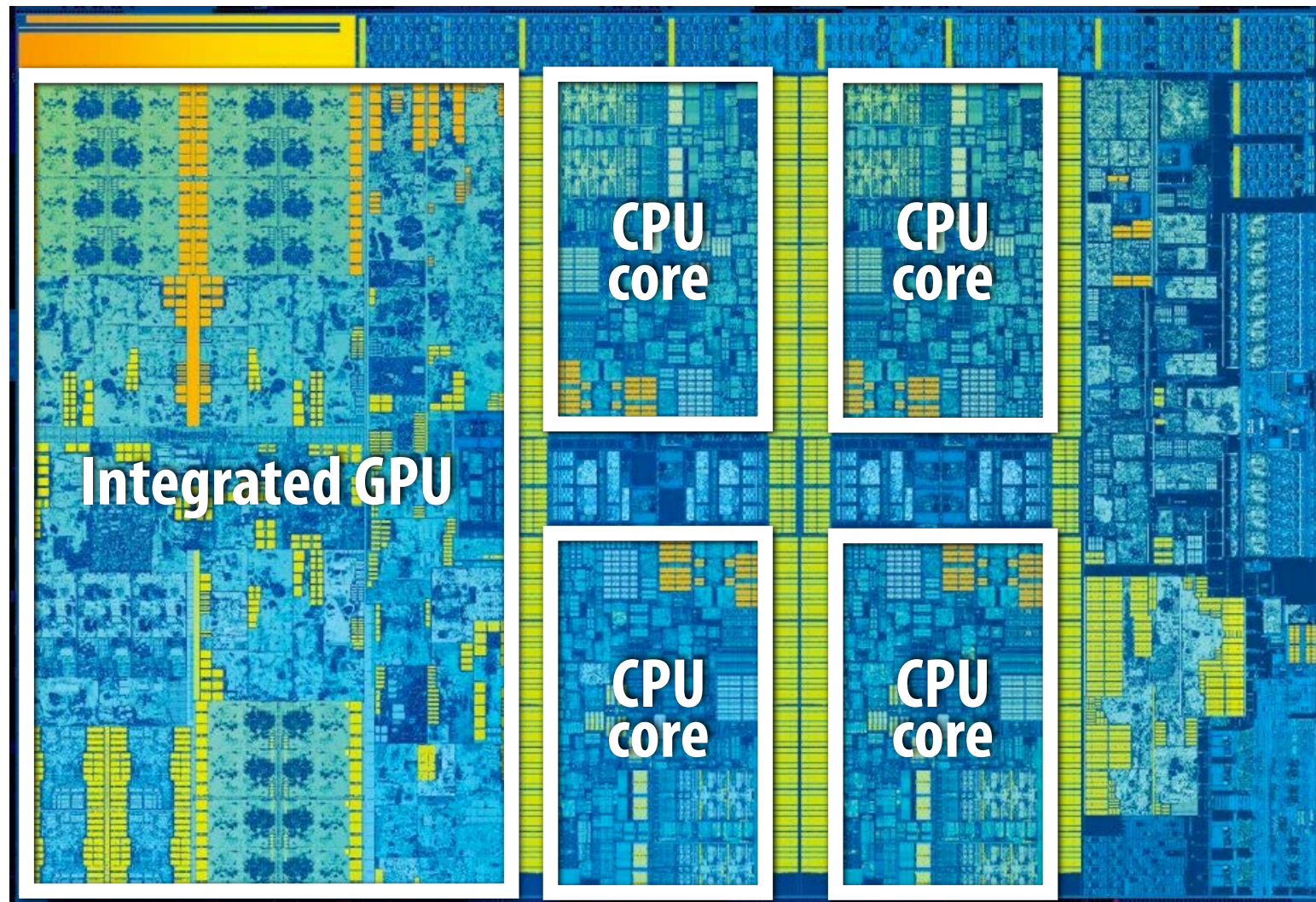
MacBook Pro Retina 15"

4 Intel Core i7 cores

(images from apple.com)

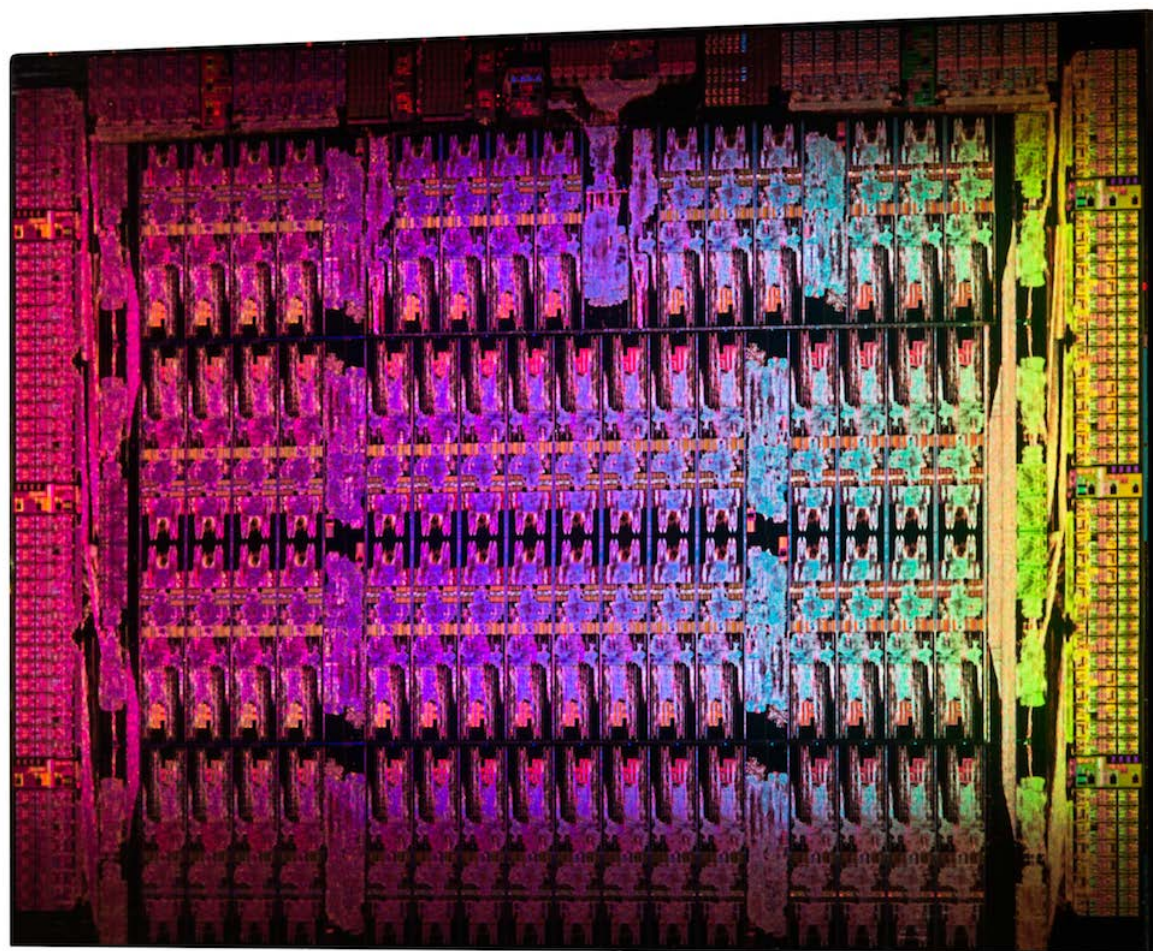
Intel Skylake (2015) (aka "6th generation Core i7")

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



Intel Xeon Phi 7120A “coprocessor”

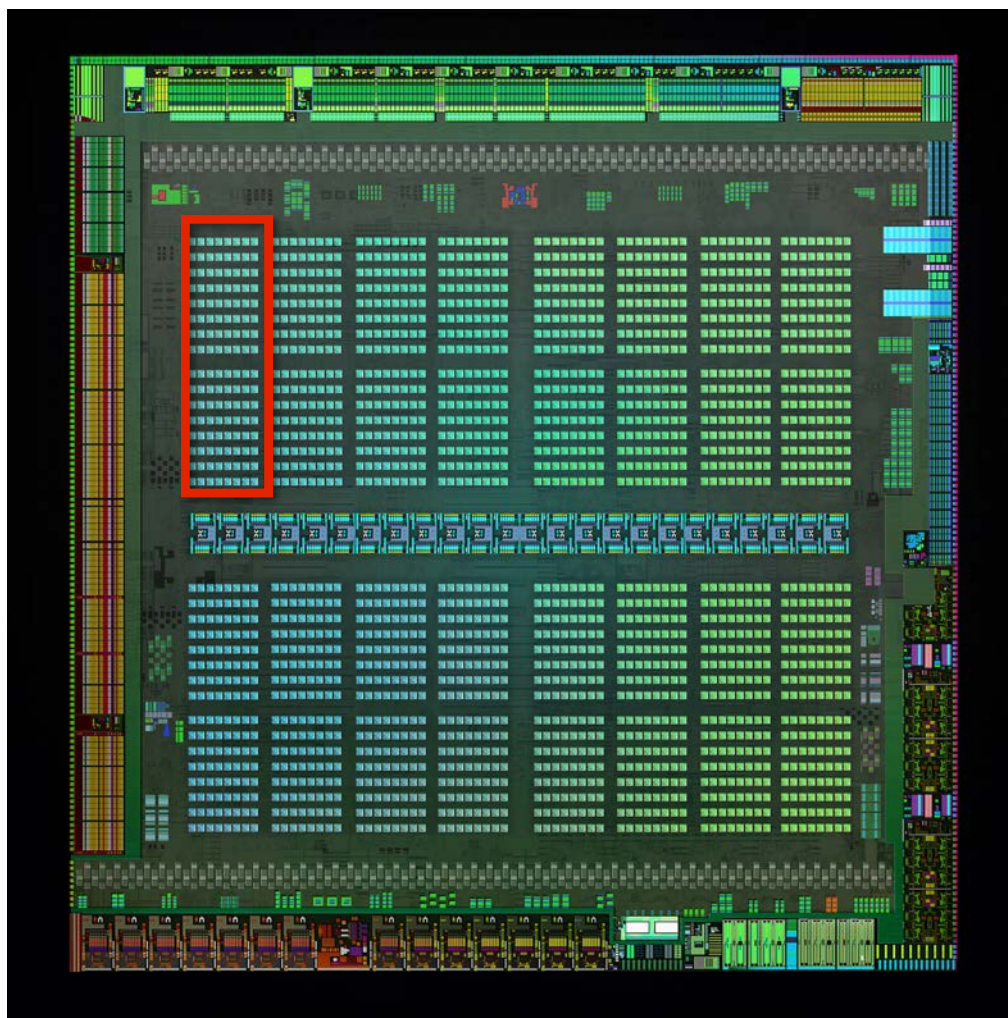
- 61 “simple” x86 cores (1.3 Ghz, derived from Pentium)
- Targeted as an accelerator for supercomputing applications



NVIDIA Maxwell GTX 980 GPU (2014)

Sixteen major processing blocks

(but much, much more parallelism available... details coming next class)

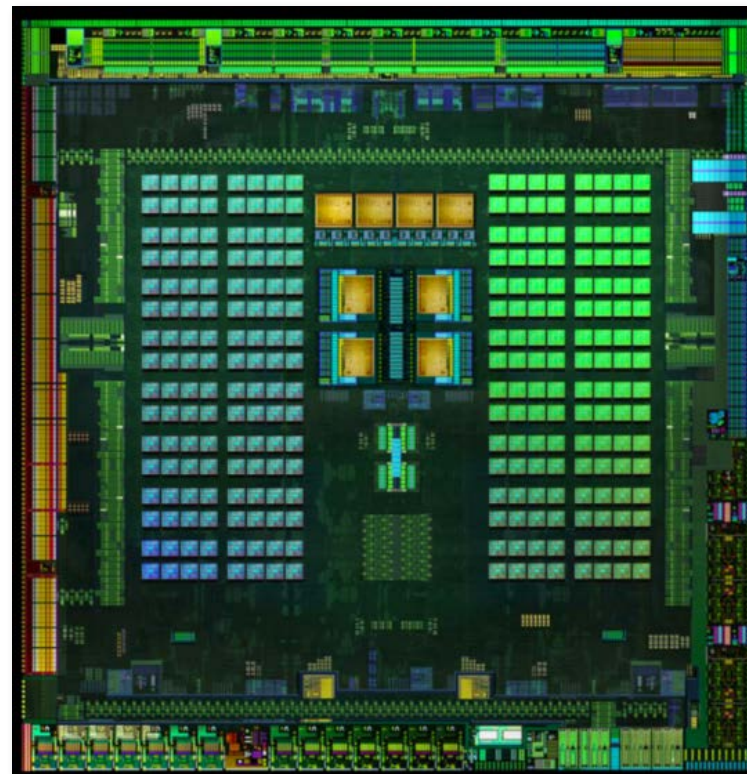


Mobile parallel processing

Power constraints heavily influence design of mobile systems



**Apple A9: (in iPhone 6s)
Dual-core CPU + GPU + image processor
and more on one chip**



**NVIDIA Tegra K1:
Quad-core ARM A57 CPU + 4 ARM A53 CPUs +
NVIDIA GPU + image processor...**

Supercomputing

- Today: clusters of multi-core CPUs + GPUs
- Oak Ridge National Laboratory: Titan (#2 supercomputer in world)
 - 18,688 x 16 core AMD CPUs + 18,688 NVIDIA K20X GPUs



What is a parallel computer?

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

DEMO 1

(This semester'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 improves speedup**
 - Moving 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 ... that scale!

■ 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.)**

Fundamental Shift in CPU Design Philosophy

Before 2004:

- within the chip area budget, maximize **performance**
 - increasingly aggressive speculative execution for ILP

After 2004:

- **area *within* the chip matters** (limits # of cores/chip):
 - maximize **performance per area**
- **power consumption** is critical (battery life, data centers)
 - maximize **performance per Watt**
- upshot: major focus on **efficiency** of cores

Summary

- **Today, single-thread performance is improving very slowly**
 - **To run programs significantly faster, programs must utilize multiple processing elements**
 - **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!**
- **Welcome to 15-418!**