Lecture 1:

Why Parallelism?

Parallel Computer Architecture and Programming
CMU 15-418, Spring 2013

Hi!

Kayvon

Hongyi

Alex

Manish

Parag

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 pthreads in 15-213**

DEMO 1

(15-418 Spring 2013'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, communication was communicating 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 parallel pieces**
  2. **Assigning work to processors**
  3. **Orchestrating communication/synchronization between the processors so that it does not limit speedup**

- **Abstractions 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 HW?
  - Because the characteristics of the machine really matter (recall speed of communication issues in class 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 10 processors a good result?

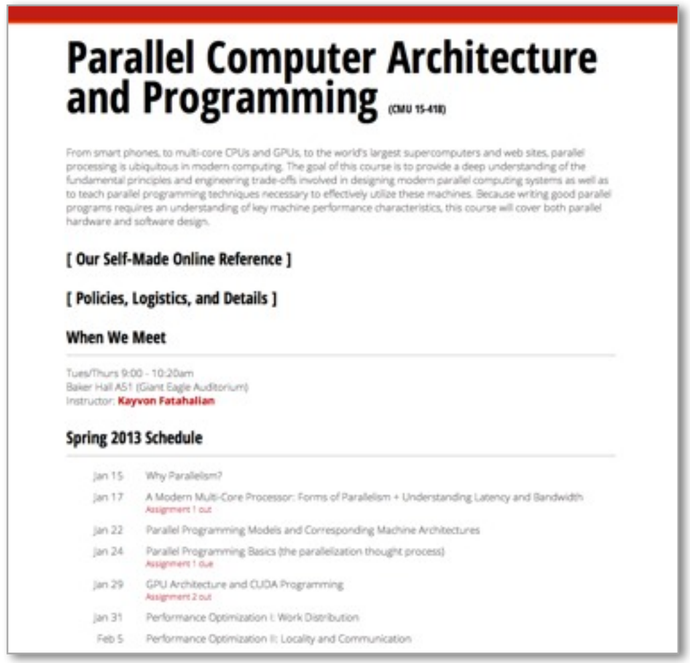
- Programmer's perspective: make use of provided machine capabilities

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

Course logistics

Getting started

- Create an account on the course web site
  - http://www.cs.cmu.edu/~15418



- Sign up for the course on Piazza
  - http://piazza.com/cmu/spring2013/15418/home

Amazing web site made by TAs Alex and Manish

- Textbook
  - There is no course textbook, but please see web site for suggested references
  - We will be generating a course reference together as part of the class!

Assignments

- Four programming assignments
  - First assignment is done individually, the rest may be done in pairs
  - Each uses a different parallel programming environment

Assignment 1: ISPC programming on Intel quad-core CPU

Assignment 2: CUDA programming on NVIDIA GPUs

Assignment 3: OpenMP and MPI programming on a supercomputing cluster

Assignment 4: Create an elastic web server that scales with load

Last year's biggest student complaint:

"Lectures are fun, but we don't have good reference material to review and learn on our own after class."

(a.k.a. "We fell asleep in class and now we're screwed.")

Well…

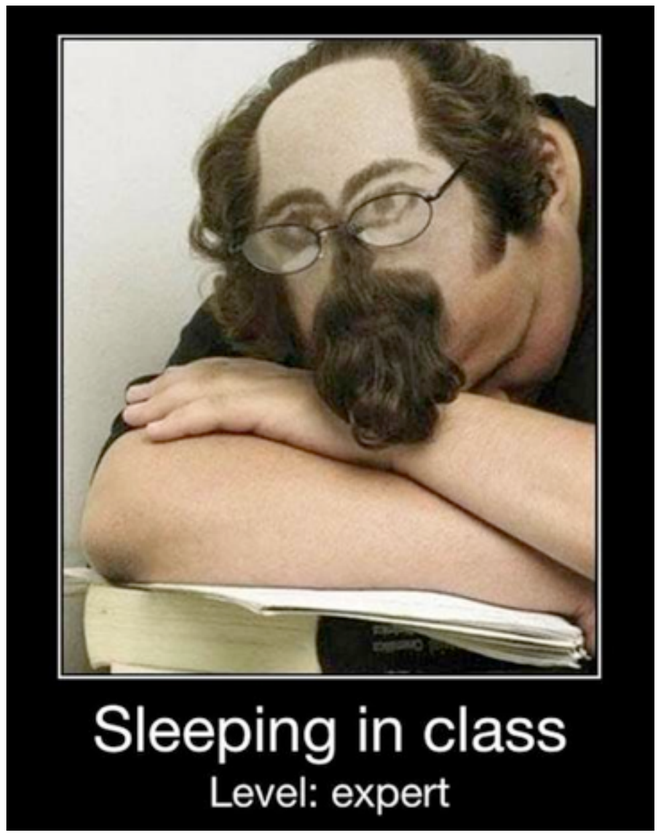


Image credit: www.hilarious-pictures.com

This year we are writing our own reference for 418!

(And we've designed a whole web site to help you do it)

- **A team of four students (+ a TA editor) will write up an explanation of each lecture for the class**
  - **"How would you explain the lecture's main ideas to your classmates?"**
  - **What do <u>you</u> think your friends should know?**

- **Everyone in the class is going to help you!**
  - **Everyone must post at least two good comments on lecture slides each week.**
  - **Once a team posts their explanation, everyone can help make it better by posting comments and suggesting improvements**
  - **If your explanation stinks, your classmates won't be too happy with you come exam time**

- **The quality of your explanation and your posted comments determine your participation grade**

Final project

- 5-week self-selected final project
- May be completed in pairs

- Announcing: the second annual 418 parallelism competition!
  - Held during the final exam slot
  - Non-CMU judges from Intel, NVIDIA, etc.
  - Expect non-trivial prizes… (e.g., high-end GPUs, solid state disks)

Grades

38% Programming assignments (4)

25% Exams (2)

25% Final project

12% Participation (5% comments + 7% lecture explanation)

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

Why parallelism?

Why parallel processing?

- **The answer 10 years ago**
  - To realize performance improvements that exceeded what CPU performance improvements could provide
  - Because if you just waited until next year, your application would run faster on a new CPU
- **Implication: working to parallelize your code was often not worth the time**
  - Software developer does nothing: CPU performance doubles ~ every 18 months. Woot!

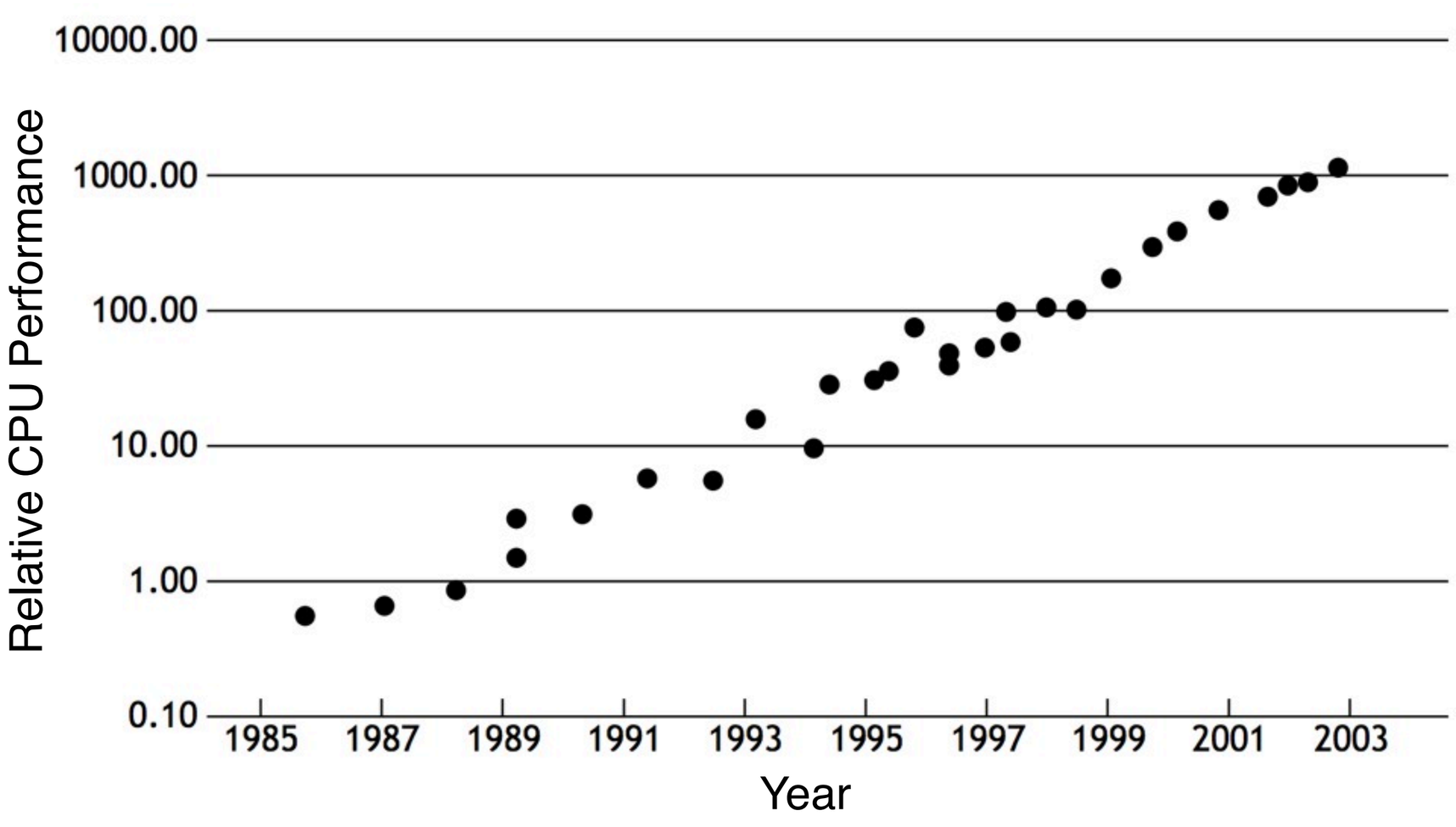


Image credit: Olukutun and Hammond, ACM Queue 2005

Until 10 years ago: two significant reasons for processor performance improvement

1. Increasing clock frequency

2. Exploiting instruction level parallelism (superscalar execution)

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 are meant to be executed in program order. BUT <u>independent</u> instructions can be executed simultaneously by a processor without changing program correctness

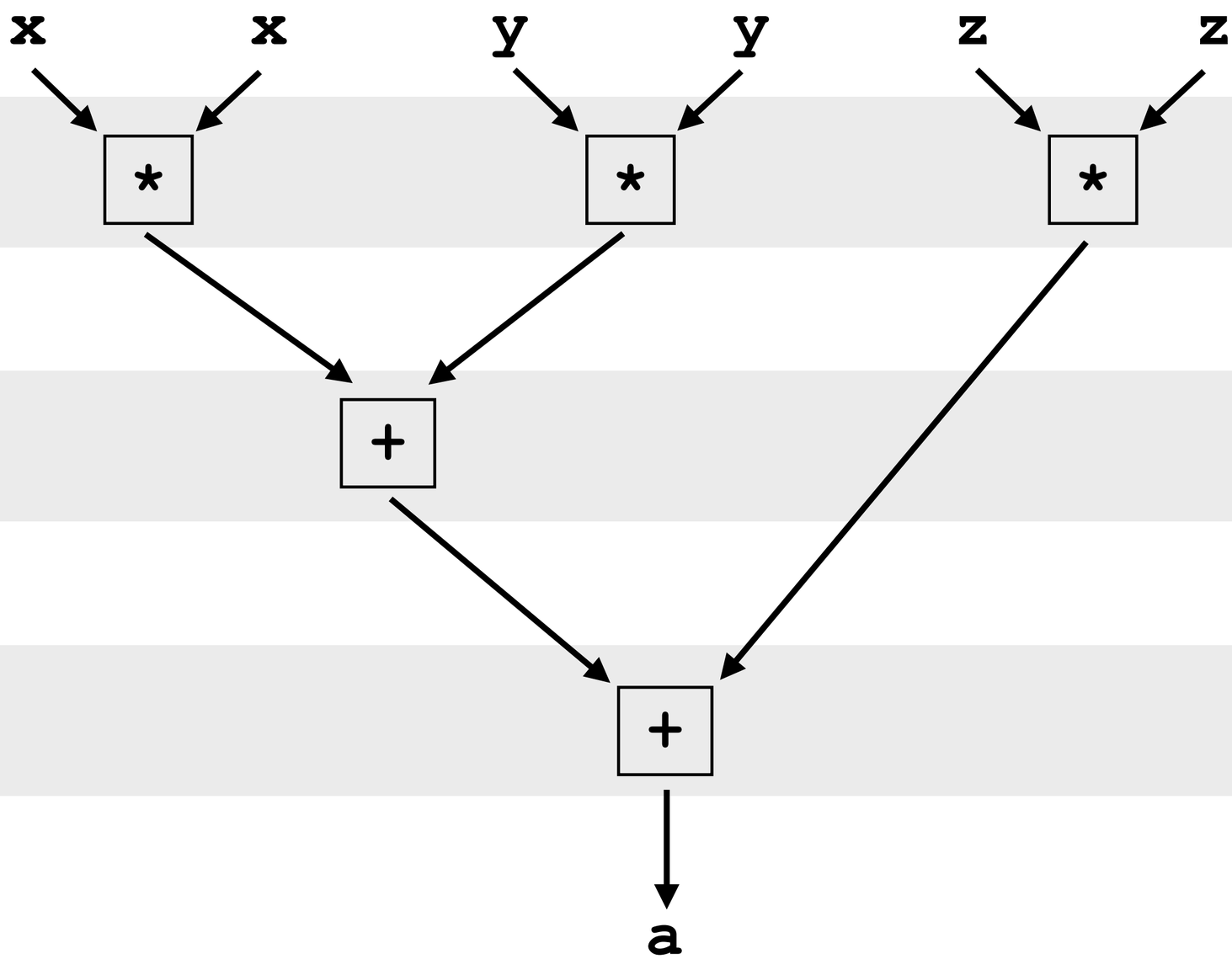
  - Superscalar execution: processor detects independent instructions in an instruction sequence and executes them in parallel

```
bf bc 44 40 00          mov    $0x4044bc,%edi
b8 01 00 00 00          mov    $0x1,%eax
41 bd 03 00 00 00       mov    $0x3,%r13d
f2 0f 59 44 24 18       mulsd  0x18(%rsp),%xmm0
e8 a5 f8 ff ff          callq  400ec8 <printf@plt>
4c 89 e6                mov    %r12,%rsi
48 89 df                mov    %rbx,%rdi
e8 02 fc ff ff          callq  401230 <_ZL12verifyResultiPfS_
4c 89 74 24 08          mov    %r14,0x8(%rsp)
0f 31                   rdtsc
41 89 c7                mov    %eax,%r15d
89 14 24                mov    %edx,(%rsp)
e8 90 fc ff ff          callq  4012d0 <_ZN10CycleTimer14secon
8b 04 24                mov    (%rsp),%eax
48 c1 e0 20             shl    $0x20,%rax
49 09 c7                or     %rax,%r15
0f 88 ed 01 00 00       js     40183d <main+0x37d>
f2 49 0f 2a cf          cvtsi2sd %r15,%xmm1
f2 0f 59 c8             mulsd  %xmm0,%xmm1
48 89 da                mov    %rbx,%rdx
48 89 ee                mov    %rbp,%rsi
```

ILP example

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

ILP = 3

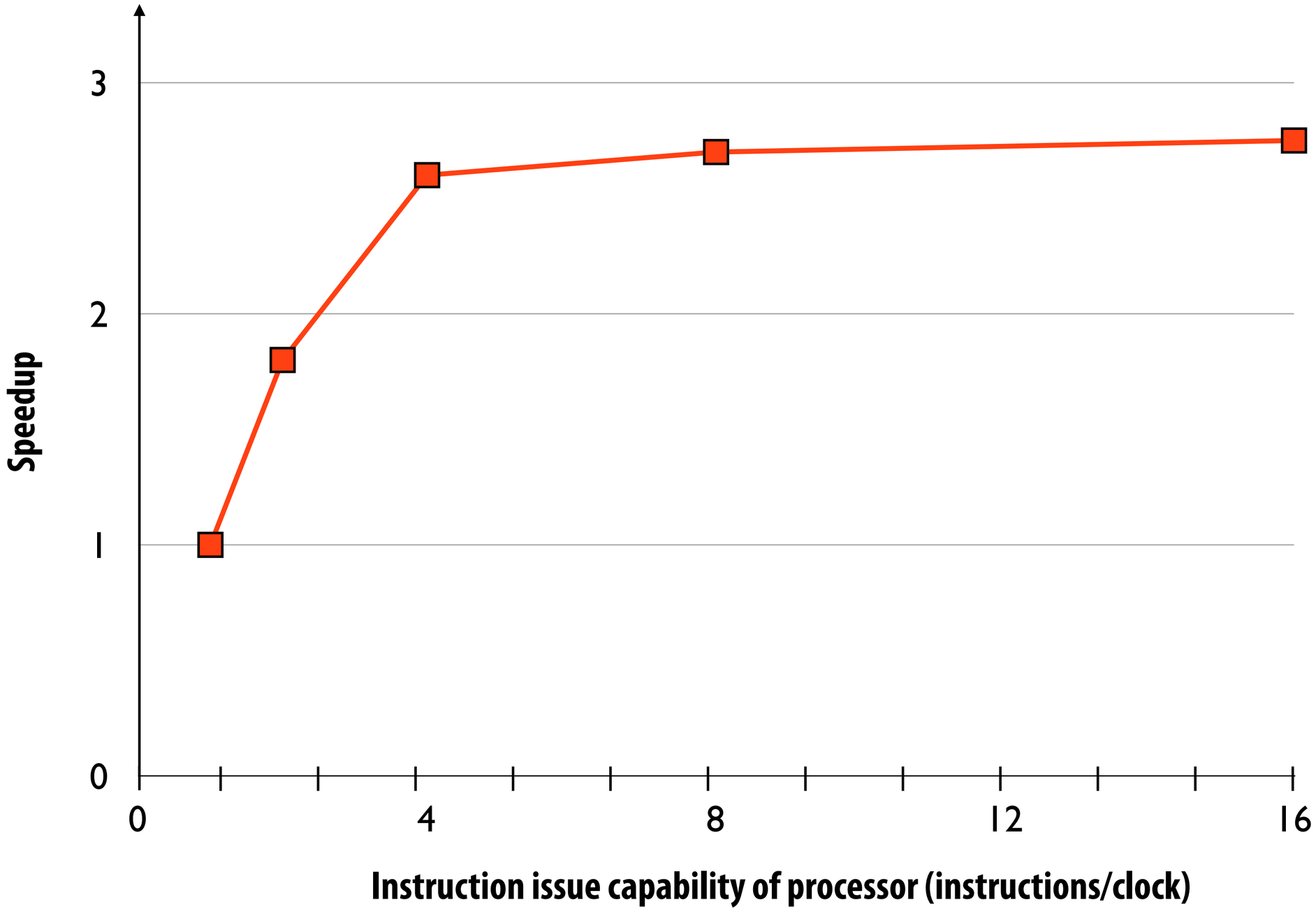


ILP = 1

ILP = 1

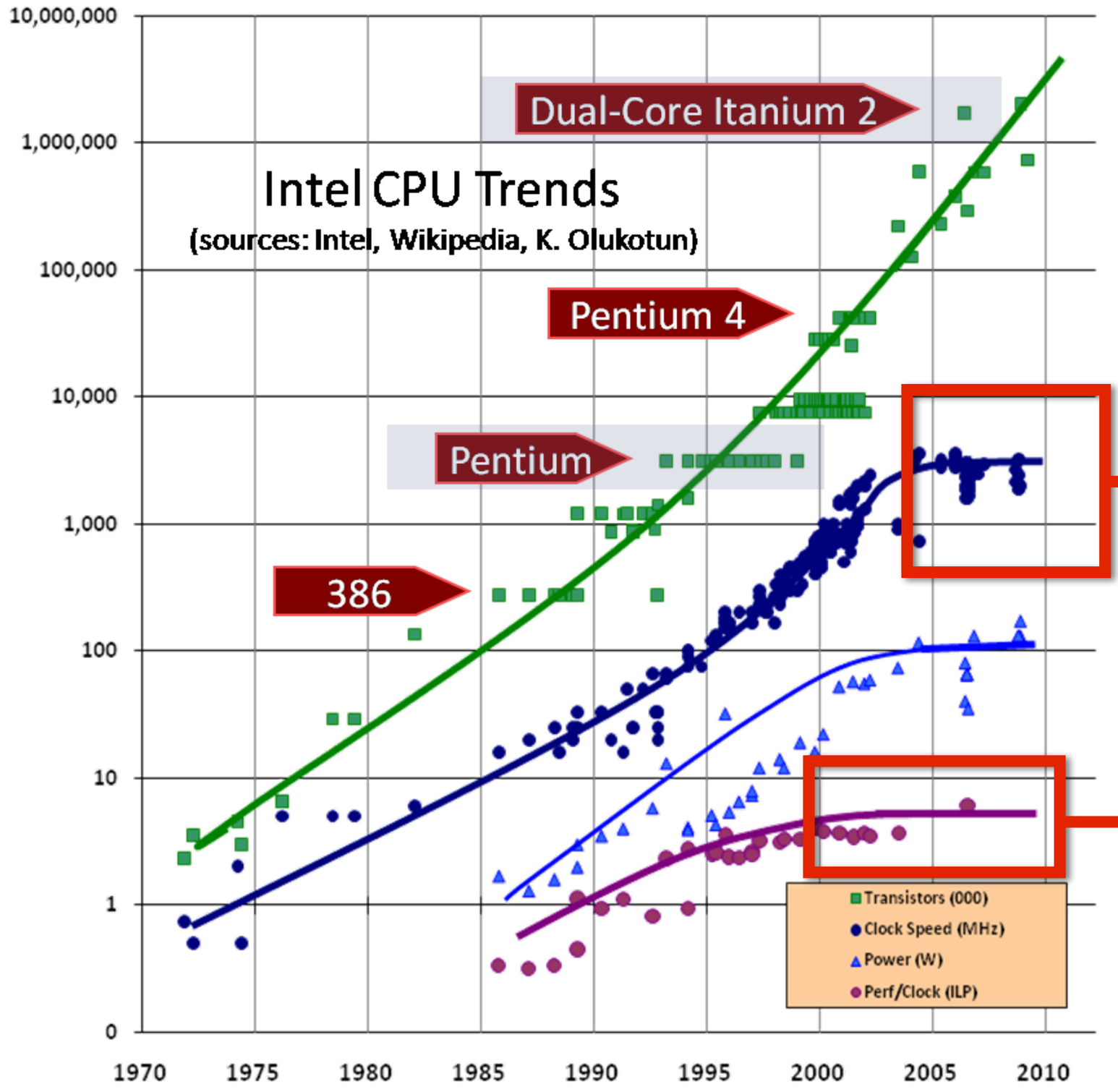
Diminishing returns of superscalar execution

Most available ILP exploited by processor capable of issuing four instructions per clock



Source: Culler & Singh (data from Johnson 1991)

ILP tapped out + end of frequency scaling



Processor clock rate stops increasing

No further benefit from ILP

Image credit: Herb Sutter

The "power wall"

Per transistor:

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

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 GTX 680 GPU	195W
Mobile phone processor	1/2 - 2W
World's fastest supercomputer	megawatts
Standard microwave oven	700W

Source: Intel, NVIDIA, Wikipedia, Top500.org

Core voltage increases with frequency

Pentium M CPU

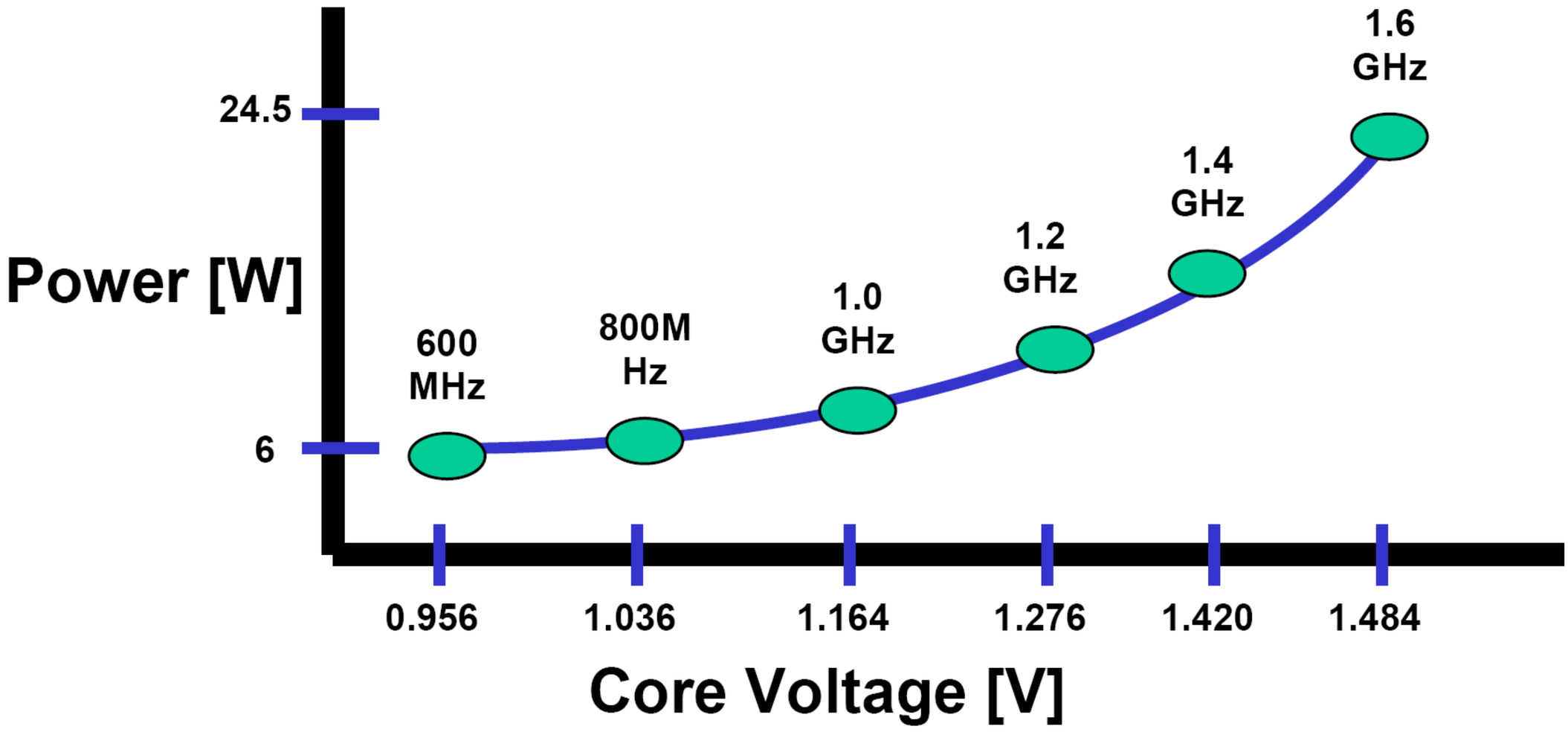


Image credit: Intel

Single-core performance scaling

The rate of single-instruction stream performance scaling has decreased (essentially to 0)

1. Frequency scaling limited by power

2. ILP scaling tapped out

Architects are now building faster processors by adding processing cores

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

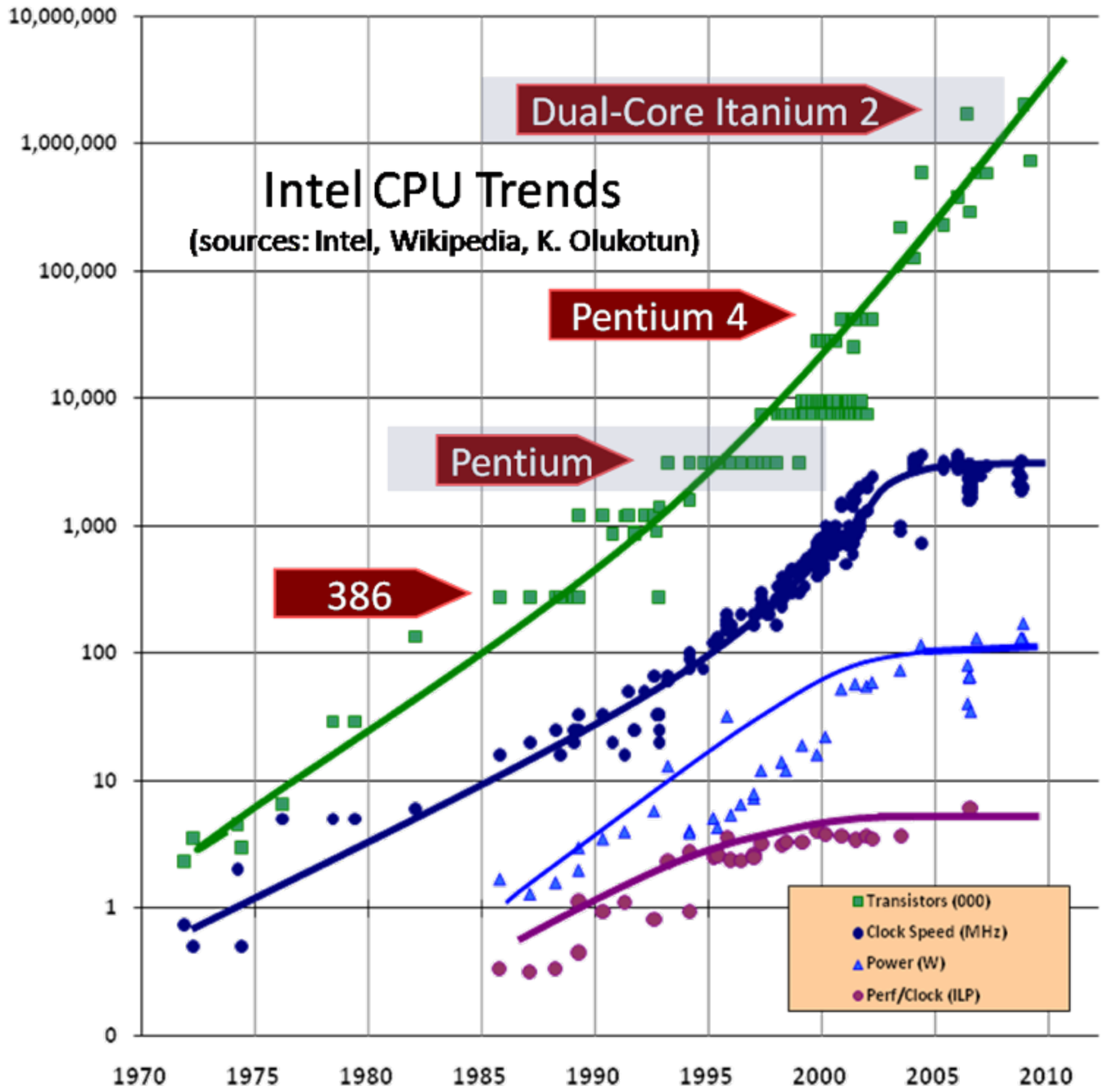


Image credit: Herb Sutter

Why parallelism?

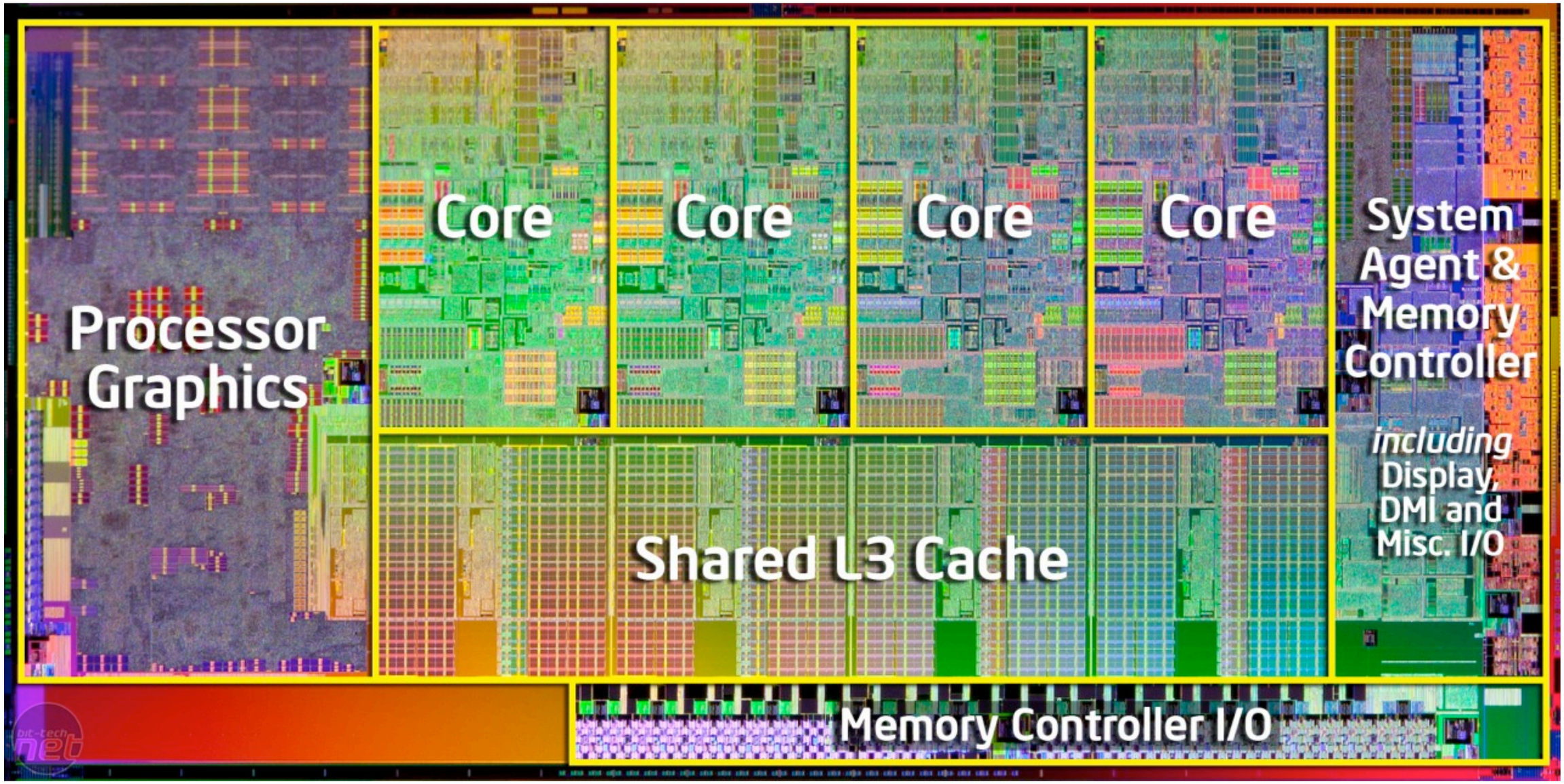
- **The answer 10 years ago**
  - To realize performance improvements that exceeded what CPU performance improvements could provide
    (specifically, in the early 2000's, what clock frequency scaling could provide)
  - Because if you just waited until next year, your code would run faster on the next generation CPU

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

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

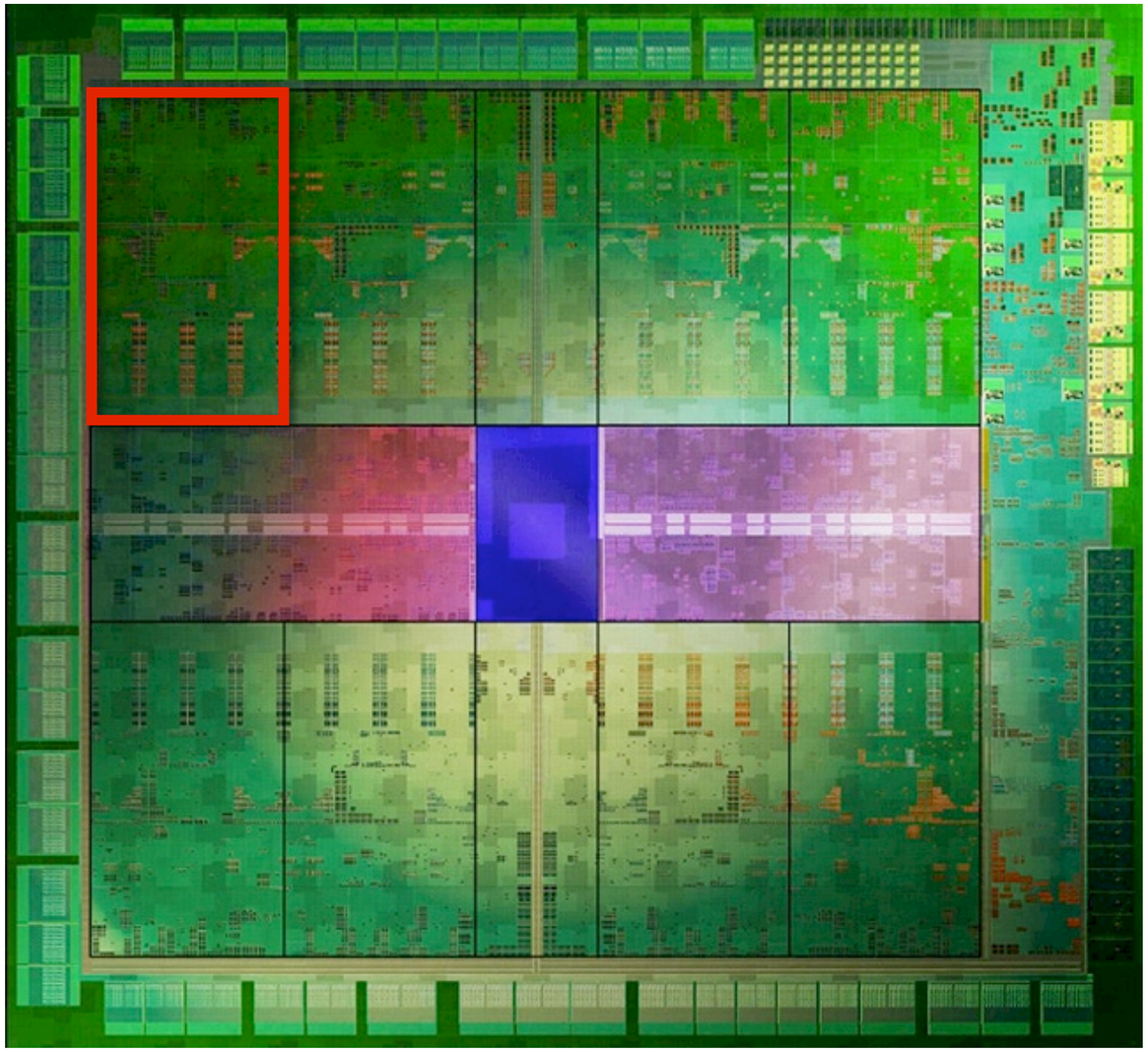
Intel Sandy Bridge (2011)

- Quad-core CPU + integrated multi-core GPU



NVIDIA Kepler GPU (2012)

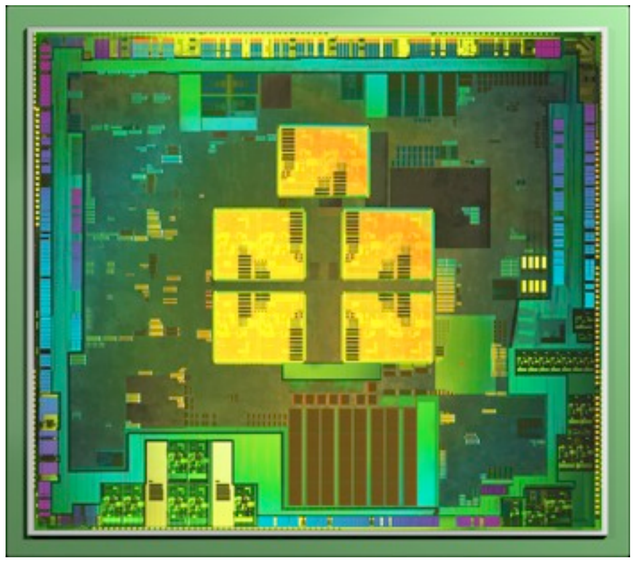
- Eight major processing blocks (more on this next lecture)



Mobile parallel processing

- Power constraints heavily influence design of mobile systems

Apple A5: (in iPhone 4s and iPad 2)
Dual-core CPU + GPU + image processor
and more on one chip



NVIDIA Tegra:
Quad core CPU + GPU + image processor...

Supercomputing

- Today: clusters of CPUs + GPUs
- Pittsburgh Supercomputing Center: Blacklight
  - 512 eight-core Intel Xeon processors (4096 total cores)

Summary

- Single-thread of control performance scaling has ended
  - To run faster, you will need to use multiple processing elements
  - Which means you need to know how to write parallel code

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

- Welcome to 15-418!