Lecture 1:

Why Parallelism?

Parallel Computer Architecture and Programming
CMU 15-418/15-618, Spring 2015

Tunes

Leela James

"Long Time Coming"

(A Change is Gonna Come)

"I'd spent all winter break waiting to write some parallel code, and when I got back in front of a machine I was so jacked I ended up just spawning pthreads all night long."

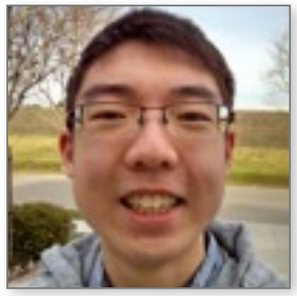
- Leela James, on the inspiration for "Long Time Coming"

Hi!

Prof. Kayvon

Arjun

Will



Cary

Xiaofan

Vivek

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 2015'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 pieces that can safely be performed in parallel**
  2. **Assigning work to processors**
  3. **Orchestrating 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 HW?
  - 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 10 processors a good result?

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

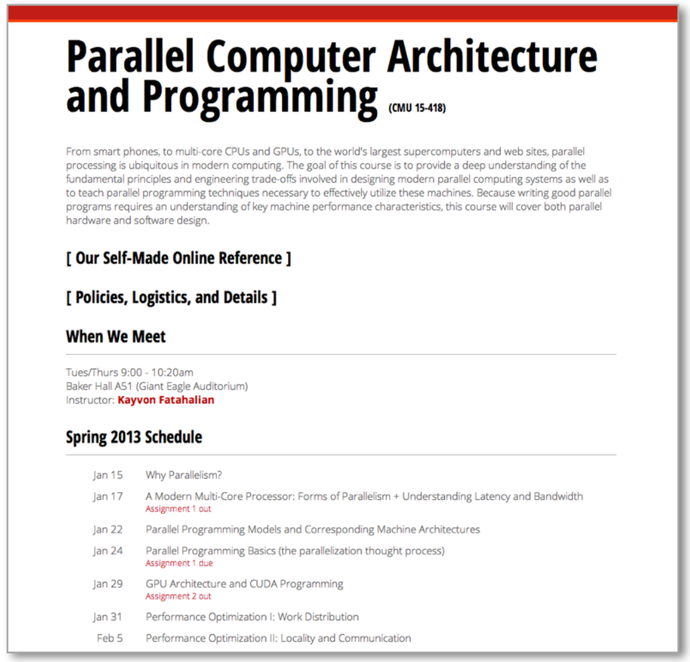
- HW designer's perspective: choosing the right capabilities to put in system (performance/cost, cost = silicon area?, power?, etc.)

Course logistics

Getting started

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

- Sign up for the course on Piazza
  - http://piazza.com/cmu/spring2015/15418/home



Parallel Computer Architecture and Programming (CMU 15-418)

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.

[Our Self-Made Online Reference]

[Policies, Logistics, and Details]

When We Meet

Tues/Thurs 9:00 - 10:20am
Baker Hall A51 (Giant Eagle Auditorium)
Instructor: Kayvon Fatahalian

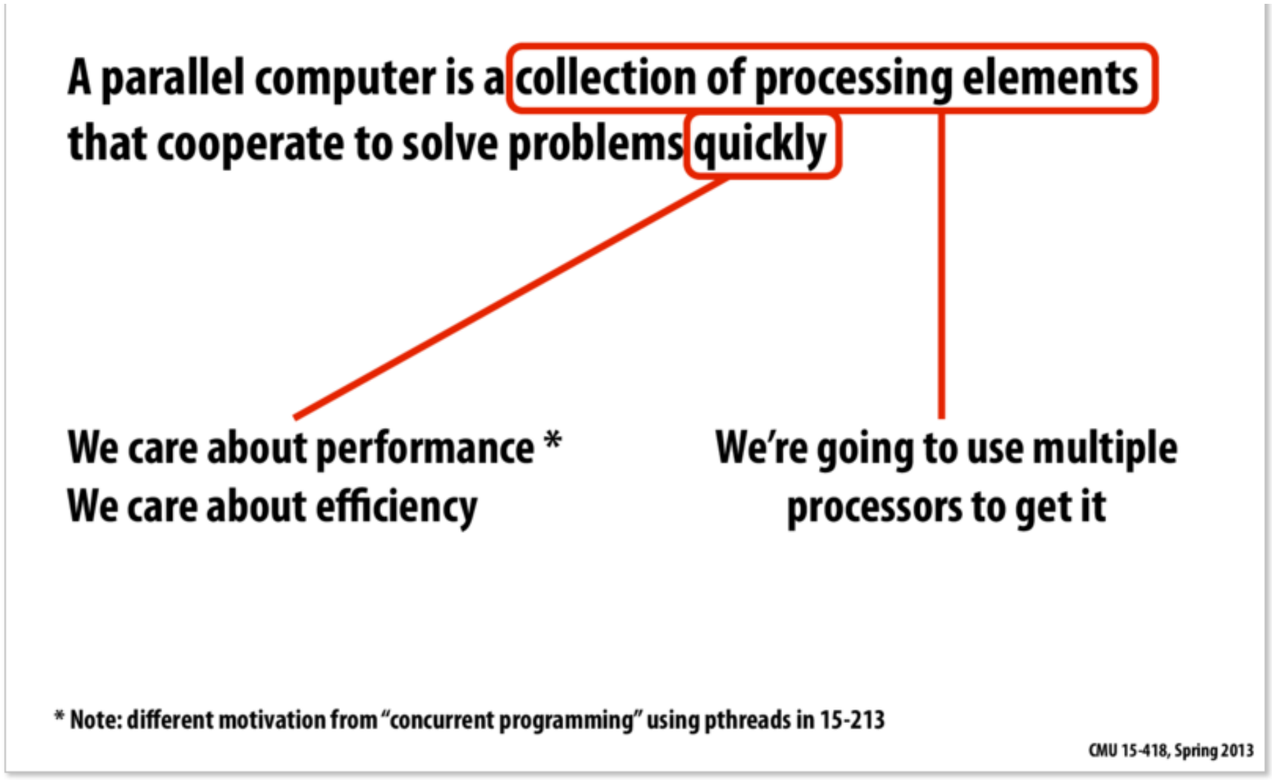
Spring 2013 Schedule

Jan 15	Why Parallelism?
Jan 17	A Modern Multi-Core Processor: Forms of Parallelism + Understanding Latency and Bandwidth Assignment 1 out
Jan 22	Parallel Programming Models and Corresponding Machine Architectures
Jan 24	Parallel Programming Basics (the parallelization thought process) Assignment 1 due
Jan 29	GPU Architecture and CUDA Programming Assignment 2 out
Jan 31	Performance Optimization I: Work Distribution
Feb 5	Performance Optimization II: Locality and Communication

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

Commenting and contributing to lectures

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



Previous | Next --- Slide 2 of 36 Back to Lecture Thumbnails

kayvonf 12 months ago
Edit Delete
Like Archive

Question: In 15-213's web proxy assignment you gained experience writing concurrent programs using pthreads. Think about your motivation for programming with threads in that assignment. How was it different from the motivation to create multi-threaded programs in this class? (e.g., consider Assignment 1, Program 1)

Hint: What is the difference between *concurrent* execution and *parallel* execution?

jpaulson 12 months ago
Edit Delete
Like Archive

Threads are about latency (responding quickly); parallel execution is about minimizing total time. These two metrics are totally independent.

Edit: A previous version of this comment said "work" instead of "time" (because I forgot "work" was a technical term at CMU), prompting some of the comments below.

gbarboza 12 months ago
Edit Delete
Unlike Archive

I've always liked the way **these slides** explain it; concurrency is about splitting a program up into tasks that can communicate and synchronize with each other, whereas parallelism is about making use of multiple processing units to decrease the time it takes for a program to run.

Liked by 3 people!

briandecost 12 months ago
Edit Delete
Unlike Archive

The thing is that there's an overhead to splitting up data or tasks to take advantage of multiple processing units -- it's a tradeoff. The parallel implementation is actually more total work (in terms of total instructions executed), but your task gets done quicker (if you did a good job writing your code). Though I guess you might save energy by not having a bunch of cores idling while one core crunches away at a serial task...

Liked by 2 people!

Xiao 12 months ago
Edit Delete
Unlike Archive

To further elaborate on concurrency: it is about doing things simultaneously, and includes not only the division of a single program. Concurrent execution was important before multi-core processors even existed. I suppose you could call scheduling multiple tasks on a single CPU "false" concurrency, as from the CPU's perspective they are not concurrent, but nonetheless to the users they looked simultaneous and that is important. Often times, the user prefers progress on all tasks rather than ultimate throughput (assuming single CPU). This goes back to the proxy example mentioned by professor Kayvon. Even if our proxy was running on a single-core machine, the concurrency would still be very useful as we do not wish to starve any single request.

Liked by 4 people!

Participation requirement (comments)

- Submit one <u>well-thought-out</u> comment per lecture (only two comments per week)

- Answer a TA's question when randomly prompted:
  - My TAs will be randomly seeding the site with questions (and asking specific students to respond!)

What I am looking for in comments

- Try to explain the slide (as if you were trying to teach your classmate while studying for an exam)
  - "Kayvon 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 hardware components in the XBox One
- Respectfully respond to another student's comment or question
  - "@segfault21, are you sure that is correct? I thought that 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
  - "@funkysenior15s point is that the overhead of communication…"

Quizzes

- Every two-weeks ON THURSDAY we will have a take-home quiz
  - You must complete on your own
  - Due 11:59pm on Thursday
  - We will grade your work to give you feedback, but only a participation grade will go into the gradebook

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 large multi-core machine

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

OR

This year's new mystery assignment: TBD

Final project

- 6-week self-selected final project
- May be completed in pairs
- Start thinking about your project ideas TODAY!

- Announcing: the FOURTH annual 418 parallelism competition!
  - Held during the final exam slot
  - Non-CMU judges… (previous years: from Intel, Apple, NVIDIA)
  - Expect non-trivial prizes… (e.g., high-end GPUs, drones, iPads, solid state disks) and most importantly fame, glory, and respect from your peers.

Grades

39% Programming assignments (4)

28% Exams (2)

28% Final project

5% Participation (quizzes and lecture comments)

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 10-15 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!

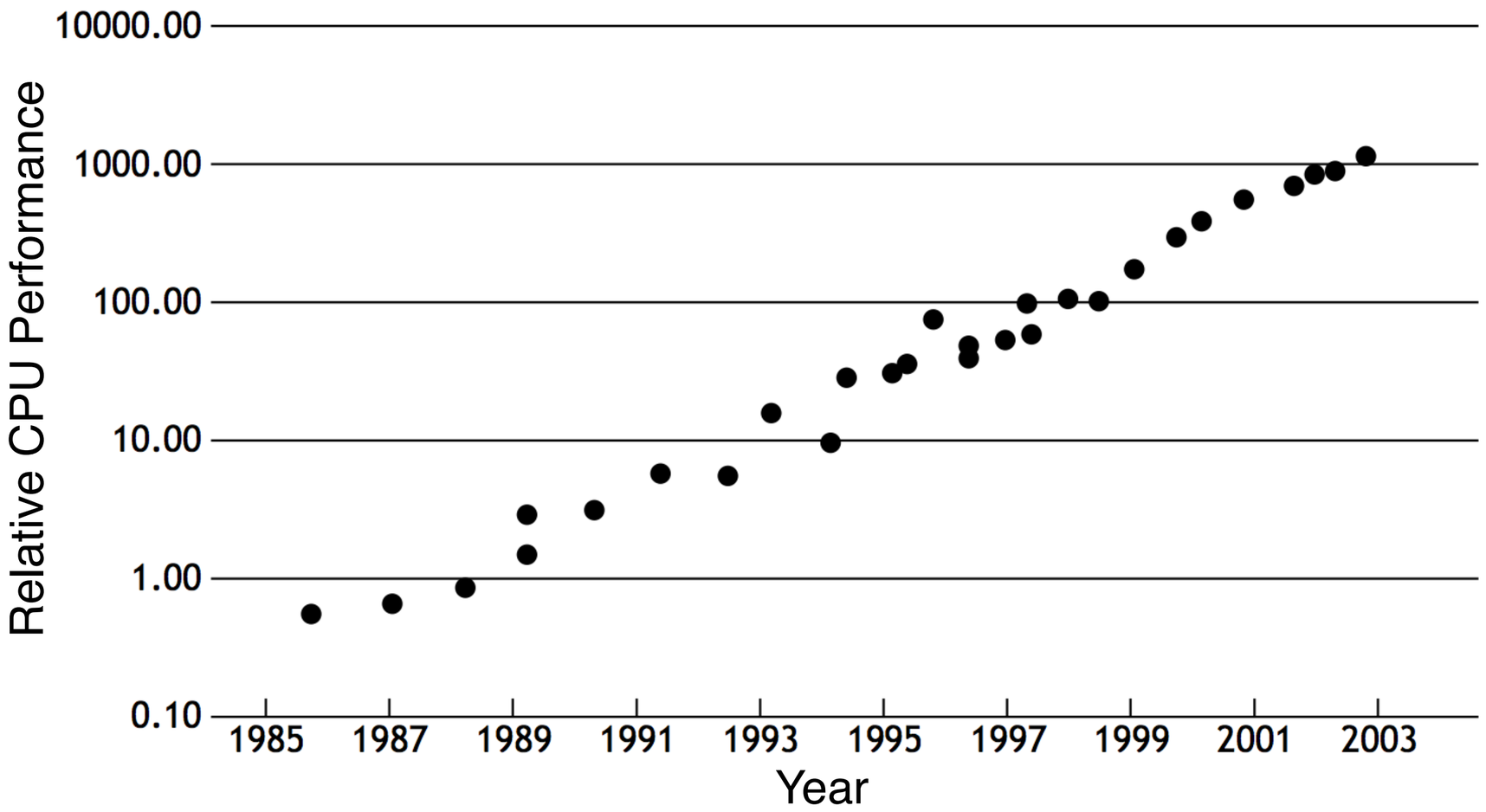


Image credit: Olukutun and Hammond, ACM Queue 2005

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)

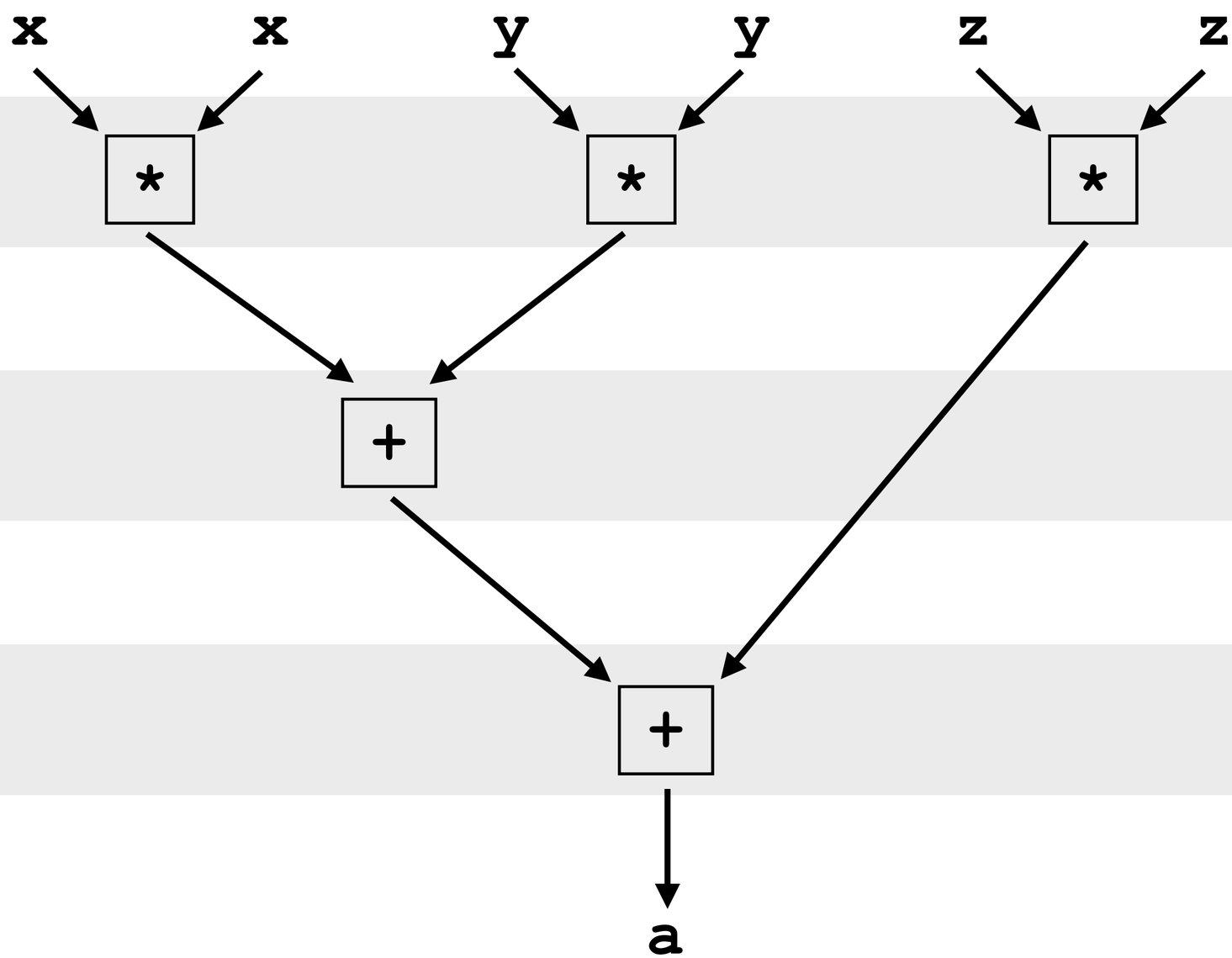
- Processors did in fact leverage parallel execution to make programs run faster, it was just invisible to the programmer

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



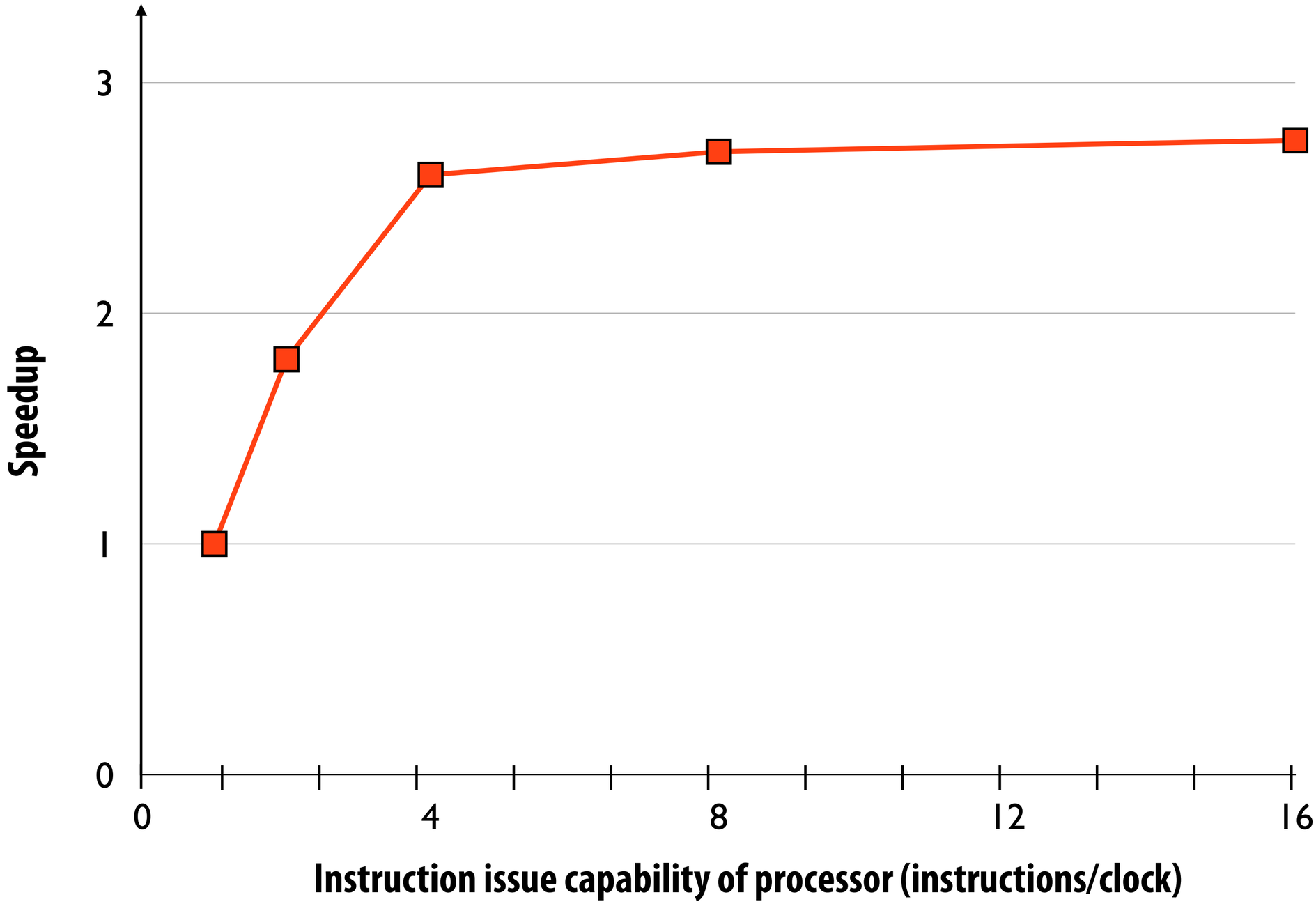
ILP = 3

ILP = 1

ILP = 1

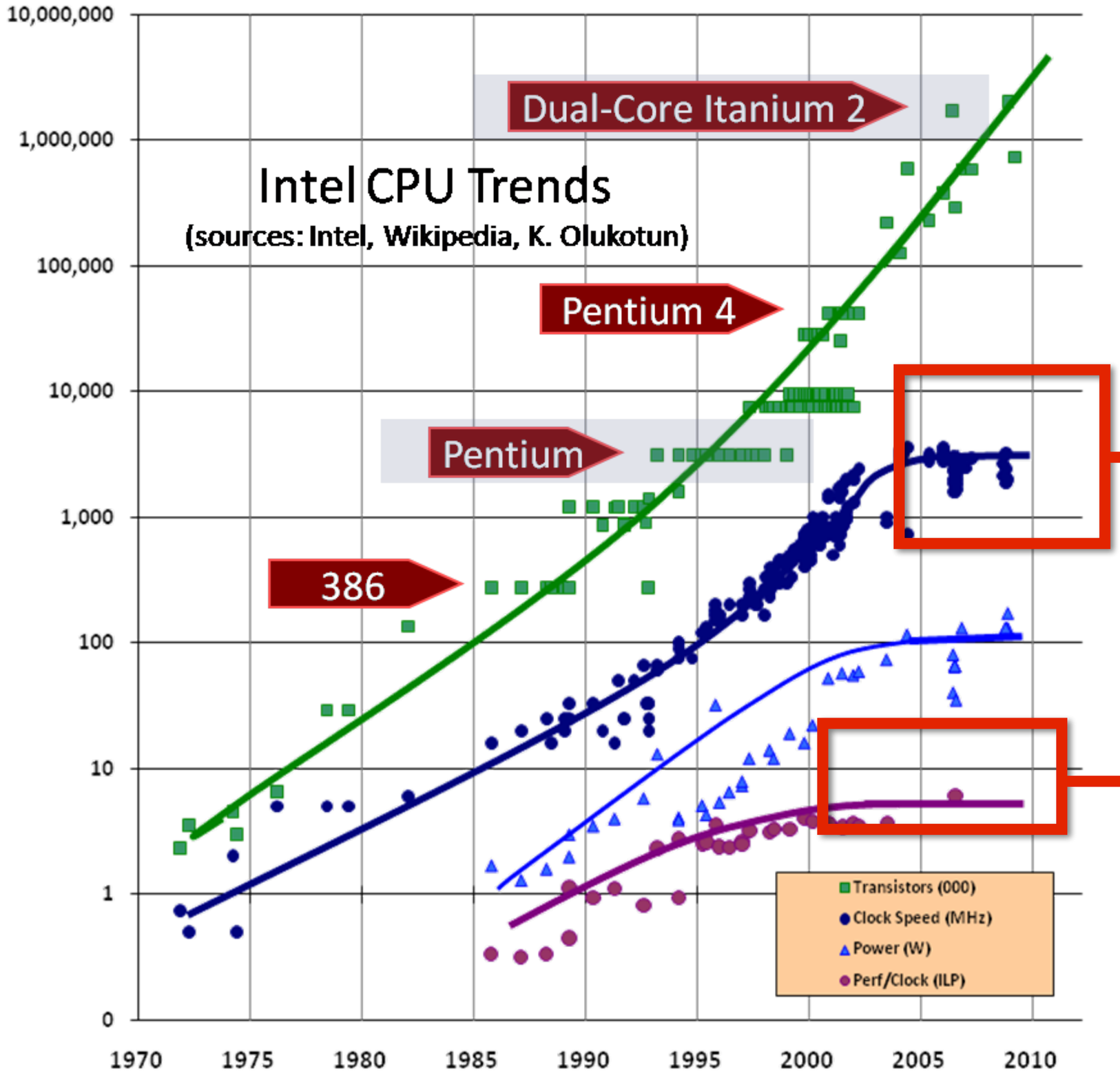
Diminishing returns of superscalar execution

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



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

The "power wall"

Per 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 GTX 780 GPU	250W
Mobile phone processor	1/2 - 2W
World's fastest supercomputer	megawatts
Standard microwave oven	700W

Source: Intel, NVIDIA, Wikipedia, Top500.org

Required core voltage increases with frequency

Pentium M CPU

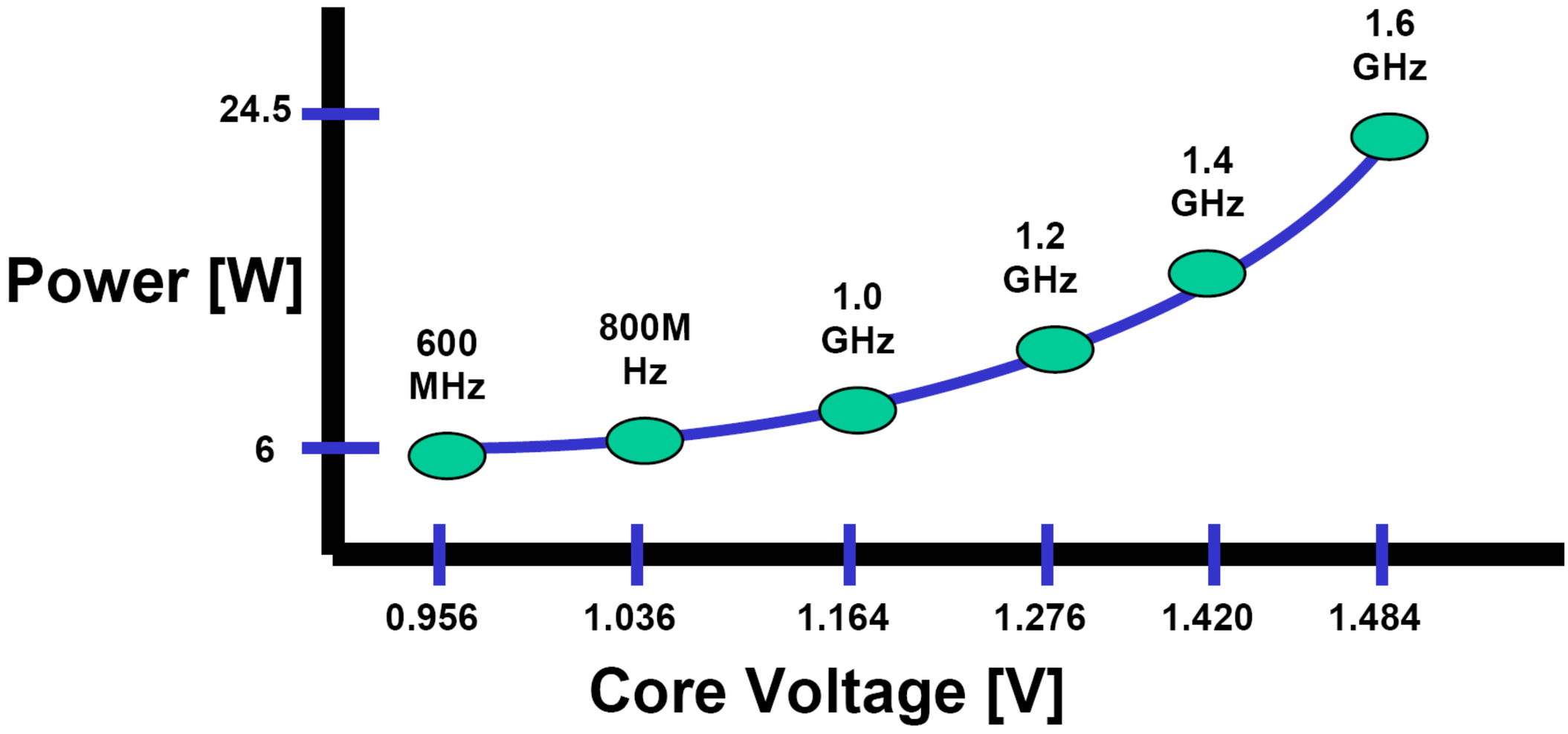


Image credit: Intel

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

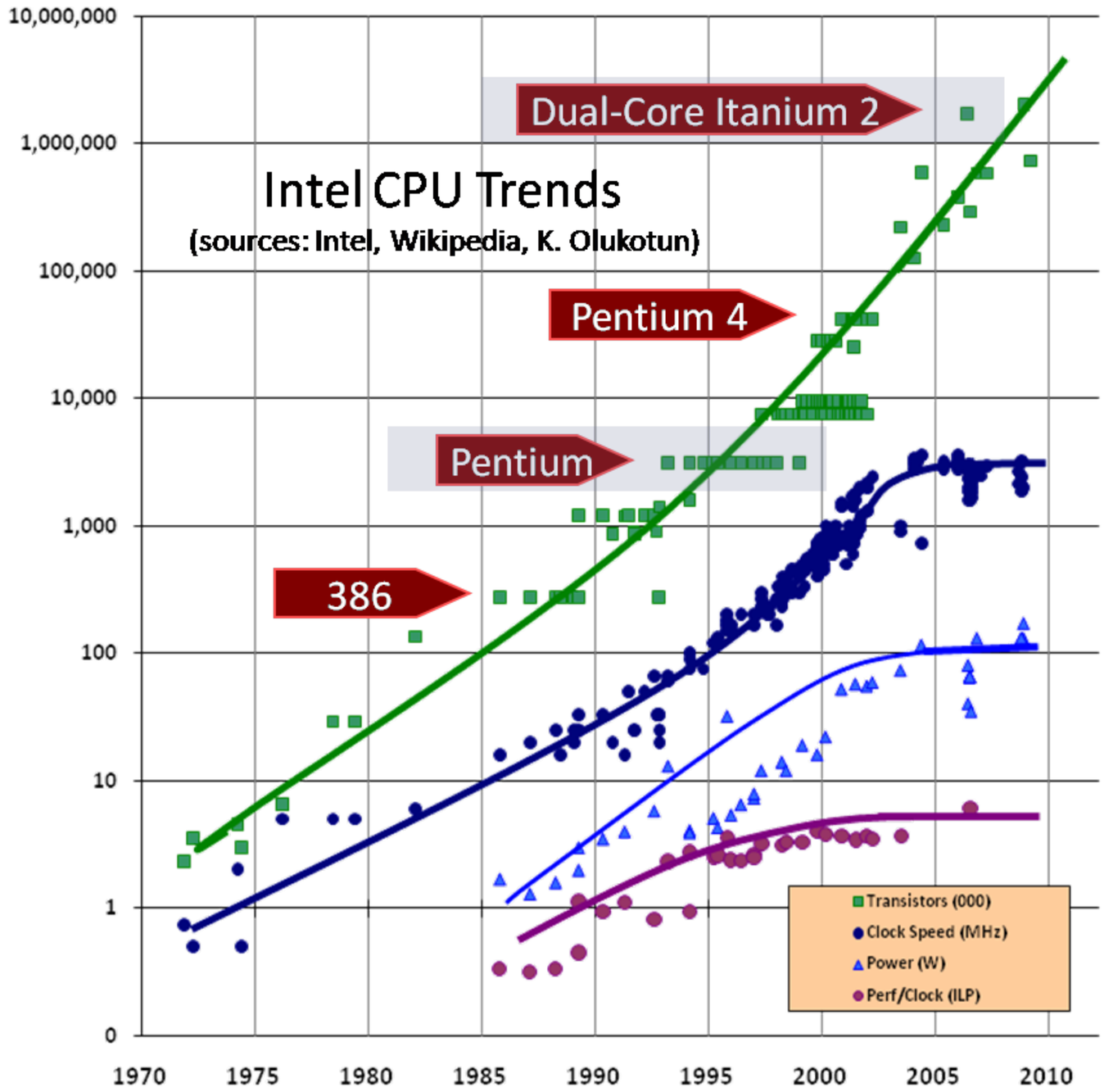


Image credit: "The free Lunch is Over" by Herb Sutter, Dr. Dobbs 2005

Why parallelism?

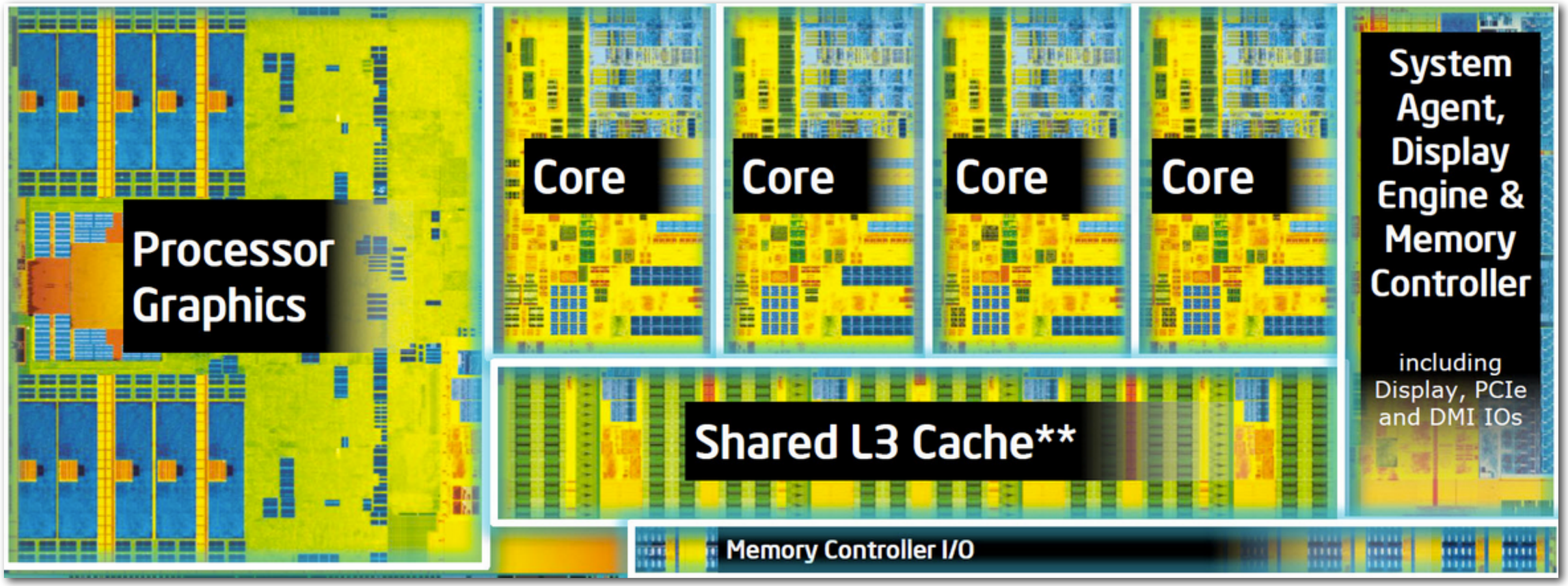
- **The answer 10 years ago**
  - To realize performance improvements that exceeded what CPU performance improvements could provide
    (specifically, in the early 2000's, what clock frequency scaling could provide)
  - Because if you just waited until next year, your code would run faster on the next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

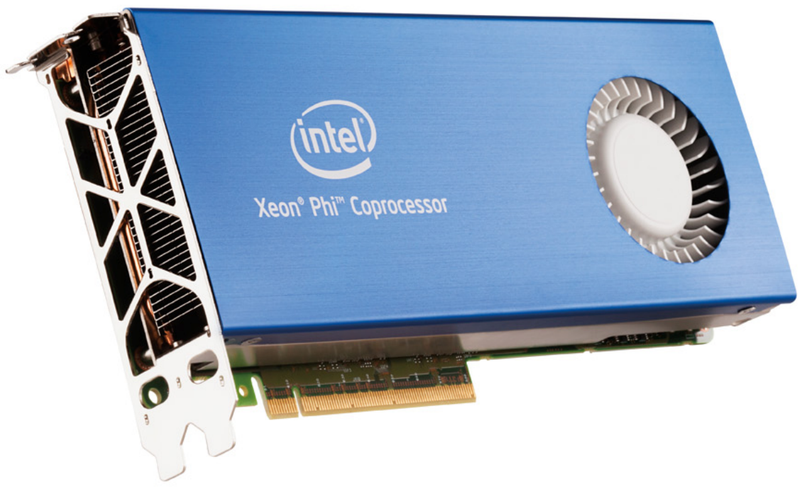
Intel Haswell (2013)

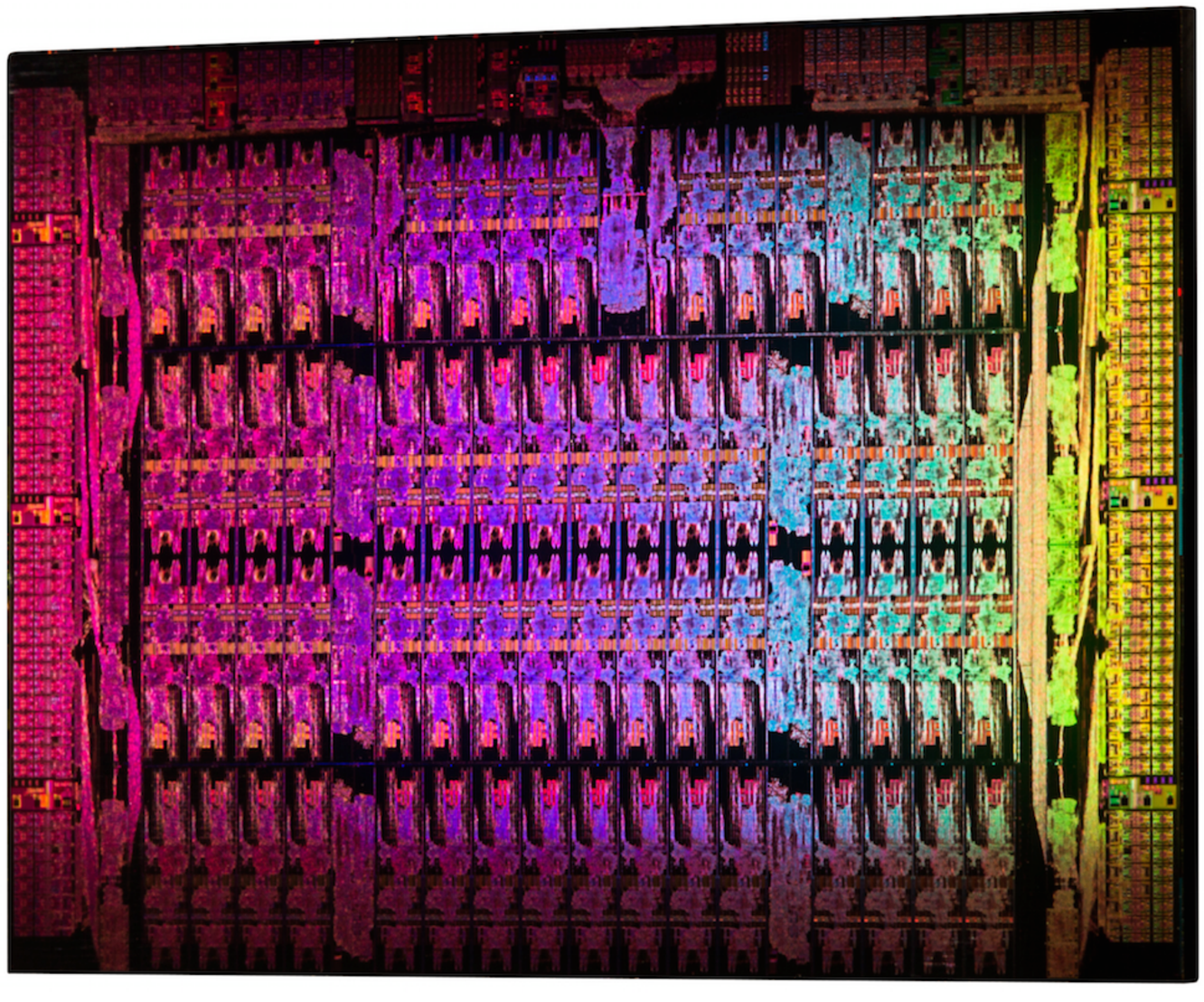
- Quad-core CPU + multi-core GPU integrated on one chip



Intel Xeon Phi "coprocessor"

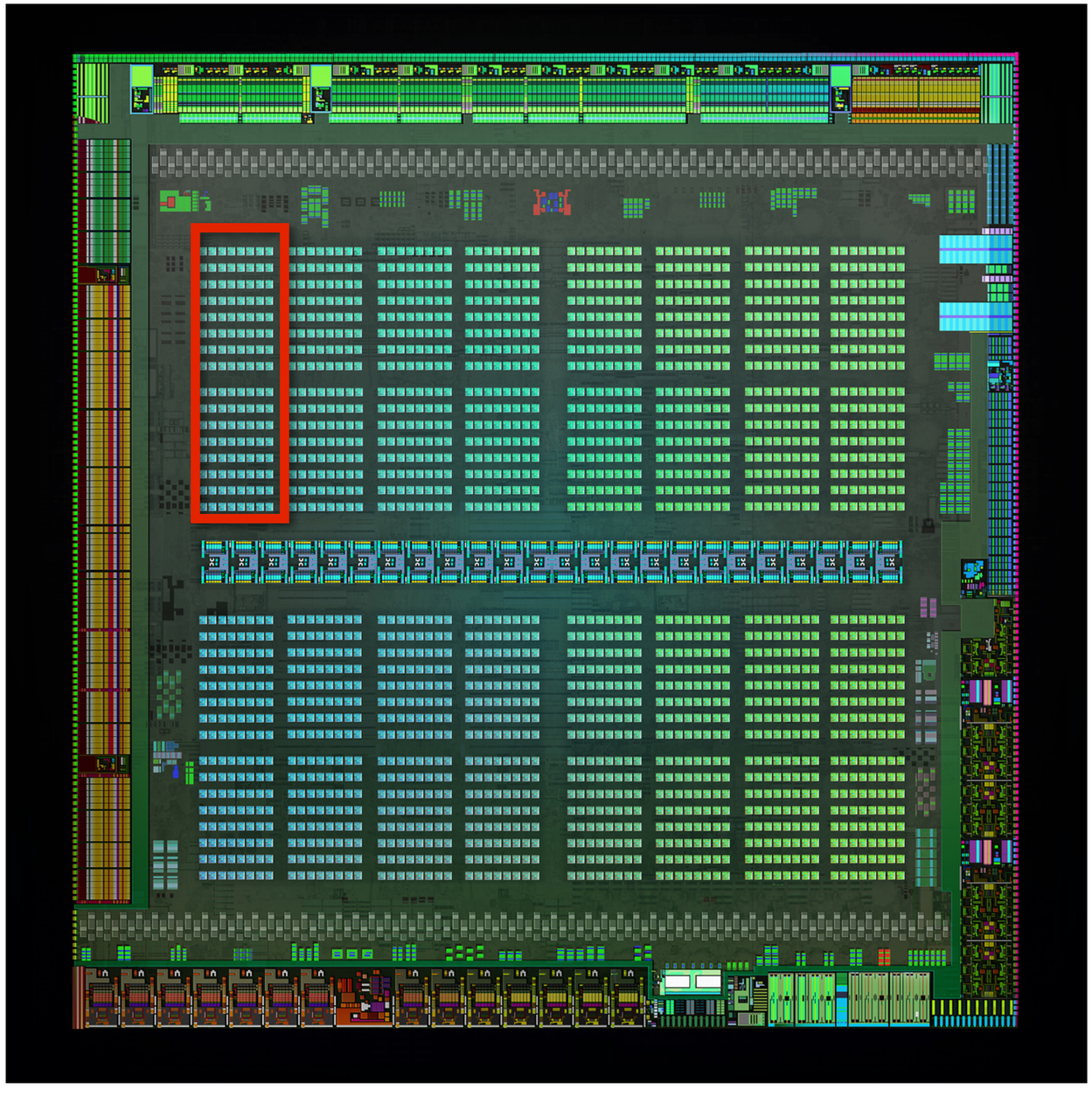
- 61 "simple" x86 cores (1.2 Ghz, derived from Pentium)
- Targeted as an accelerator for supercomputing applications





NVIDIA Maxwell GTX 980 GPU (2014)

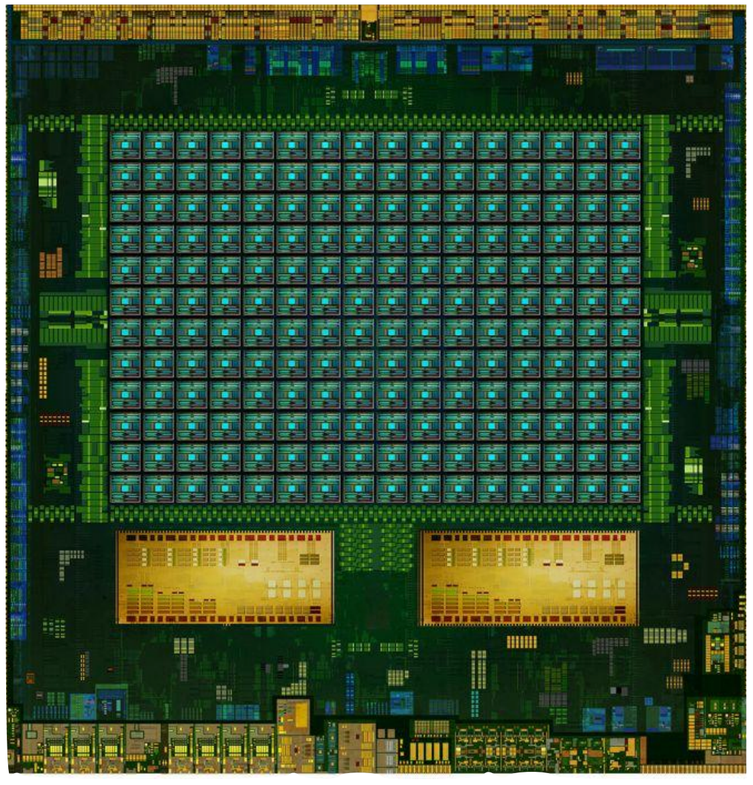
- **Sixteen major processing blocks**
  (but much, much more parallelism to use… details next class)



Mobile parallel processing

- Power constraints heavily influence design of mobile systems

Apple A7: (in iPhone 5s and modern iPad)
Dual-core CPU + GPU + image processor
and more on one chip



NVIDIA Tegra K1:
Quad core CPU + GPU + image processor...

Supercomputing

- Today: clusters of CPUs + GPUs
- Pittsburgh Supercomputing Center: Blacklight
  - 512 eight-core Intel Xeon processors (4,096 total cores)

Summary

- Today, single-thread-of-control performance is improving very slowly
  - To run 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
  - 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!