

**Lecture 1:**

# **Why Parallelism? Why Efficiency?**

---

**Parallel Computer Architecture and Programming  
CMU 15-418/15-618, Spring 2017**

# Tunes

## Leela James

**“Long Time Coming”**

**(A Change is Gonna Come)**

*“I’d heard a bit about parallelism in 213. Then I mastered the idea of span in 210.  
And so I was just itching to start tuning code for some Skylake cores.”*

*- Leela James, on the inspiration for “Long Time Coming”*

# Hi!



**Prof. Kayvon**



**Prof. Bryant**



**Alex**



**Teguh**



**Yicheng**



**Anant**



**Ravi**



**Junhong**



**Tao**



**Riya**

# One common definition

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

A diagram illustrating the definition of a parallel computer. The main text is "A parallel computer is a collection of processing elements that cooperate to solve problems quickly". The phrase "collection of processing elements" is enclosed in a red rounded rectangle, and "quickly" is enclosed in a smaller red rounded rectangle. A red line extends from the top of the "quickly" box down to the text "We're going to use multiple processors to get it". Another red line extends from the bottom of the "collection of processing elements" box down to the text "We care about performance \* We care about efficiency".

**We care about performance \***  
**We care about efficiency**

**We're going to use multiple  
processors to get it**

# DEMO 1

**(15-418/618 Spring 2017's first parallel program)**

# Speedup

**One major motivation of using parallel processing: achieve a speedup**

**For a given problem:**

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

# 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 ... 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  $\neq$  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

## ■ 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/spring2017/15418618/home>

## ■ Textbook

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

### 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 AS1 (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 <a href="#">Assignment 1 out</a>
Jan 22	Parallel Programming Models and Corresponding Machine Architectures
Jan 24	Parallel Programming Basics (the parallelization thought process) <a href="#">Assignment 1 due</a>
Jan 29	GPU Architecture and CUDA Programming <a href="#">Assignment 2 out</a>
Jan 31	Performance Optimization I: Work Distribution
Feb 5	Performance Optimization II: Locality and Communication

# Commenting and contributing to lectures

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

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

CMU 15-418, Spring 2013

[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)

- You are required to submit one well-thought-out 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!)

# 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)**
  - “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 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 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**
  - “@funkysenior17’s point is that the overhead of communication...”

# Written exercises

- **Every two-weeks we will have a take-home exercise**
  - **You must complete the exercise on your own**
  - **Distributed Wednesday night, due 10:00am on Friday**
  - **We will grade your work to give you feedback, but only a participation grade will go into the gradebook**

# Assignments

## ■ Four programming assignments

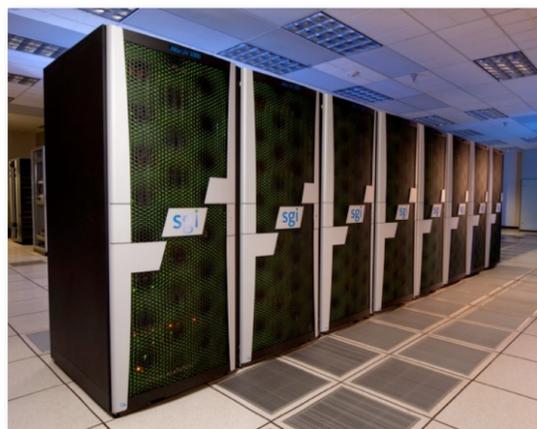
- **First assignment is done individually, the rest may be done in pairs**
- **Each uses a different parallel programming environment**



**Assignment 1: ISPC programming on multi-core CPUs (and Xeon Phi)**



**Assignment 2: CUDA programming on NVIDIA GPUs**



**Assignment 3:  
new assignment, to be announced  
(high core count shared memory)**



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

# Final project

- **6-week self-selected final project**
- **May be completed in pairs**
- **Start thinking about your project ideas TODAY!**
- **Announcing: the SIXTH annual 418/618 parallelism competition!**
  - Held during the final exam slot
  - Non-CMU judges... (previous years: from Intel, Apple, NVIDIA, Google)
  - Expect non-trivial prizes... (e.g., high-end GPUs, drones, iPads, solid state disks) and most importantly fame, glory, and respect from your peers.



# Check out last year's projects!

<http://15418.courses.cs.cmu.edu/spring2016/projects>

# Grades

**39% Programming assignments (4)**

**28% Exams (2)**

**28% Final project**

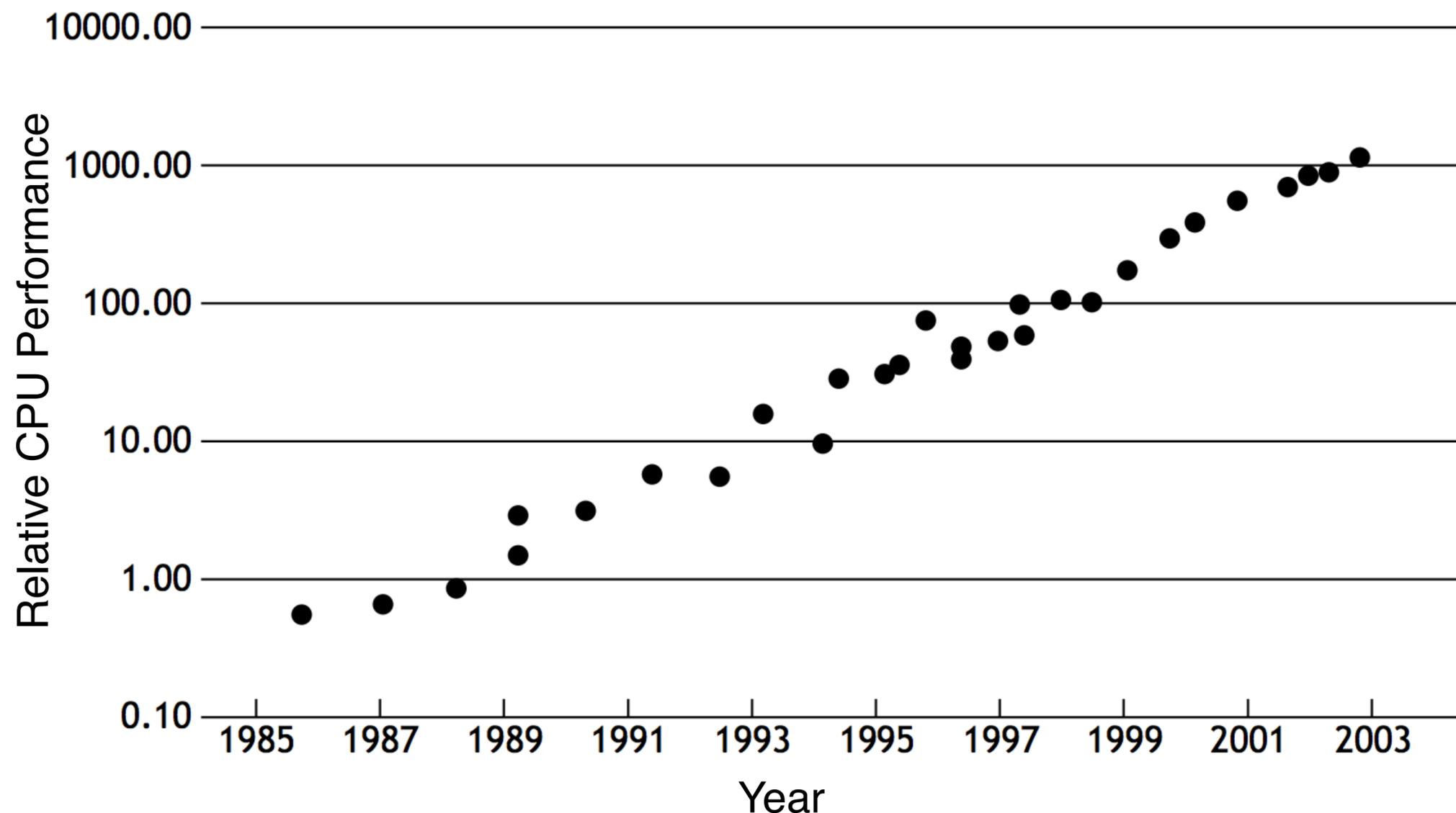
**5% Participation (exercises and lecture comments)**

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

# Why parallelism?

# Some historical context: why not parallel processing?

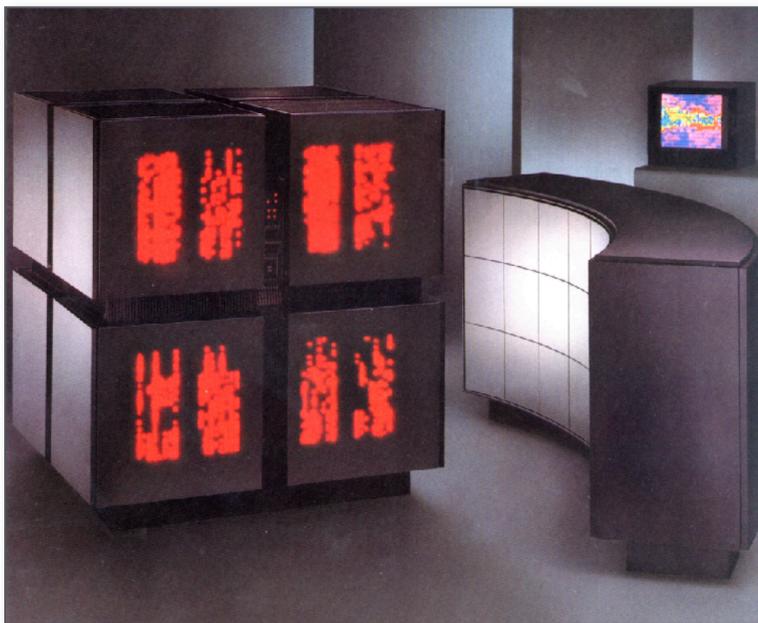
- **Single-threaded CPU performance doubling ~ every 18 months**
- **Implication: working to parallelize your code was often not worth the time**
  - **Software developer does nothing, code gets faster next year. Woot!**



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

The answer until 10-15 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**

# Review: what is a program?

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

```
401200: xor    %eax,%eax
401202: xor    %r9d,%r9d
401205: jmp    401222 <_Z12verifyResultPiS_ii+0x42>
401207: nopw  0x0(%rax,%rax,1)
40120e:
401210: mov    0x4(%rdi,%rax,1),%r10d
401215: add    $0x4,%rax
401219: mov    (%r11,%rax,1),%r8d
40121d: cmp    %r8d,%r10d
401220: jne    401248 <_Z12verifyResultPiS_ii+0x68>
401222: add    $0x1,%r9d
401226: cmp    %edx,%r9d
401229: jne    401210 <_Z12verifyResultPiS_ii+0x30>
40122b: add    $0x1,%esi
40122e: add    %rbx,%rdi
401231: add    %rbx,%r11
401234: cmp    %ecx,%esi
401236: jne    4011f1 <_Z12verifyResultPiS_ii+0x11>
401238: mov    $0x1,%eax
40123d: pop    %rbx
40123e: retq
40123f: xor    %r9d,%r9d
401242: nopw  0x0(%rax,%rax,1)
401248: mov    %r10d,%ecx
40124b: mov    %r9d,%edx
40124e: mov    $0x401be8,%edi
401253: xor    %eax,%eax
401255: callq 400a40 <printf@plt>
40125a: xor    %eax,%eax
40125c: pop    %rbx
40125d: retq
40125e: mov    $0x1,%eax
401263: 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...

```
401200: xor    %eax,%eax
401202: xor    %r9d,%r9d
401205: jmp    401222 <_Z12verifyResultPiS_ii+0x42>
401207: nopw  0x0(%rax,%rax,1)
40120e:
401210: mov    0x4(%rdi,%rax,1),%r10d
401215: add    $0x4,%rax
401219: mov    (%r11,%rax,1),%r8d
40121d: cmp    %r8d,%r10d
401220: jne    401248 <_Z12verifyResultPiS_ii+0x68>
401222: add    $0x1,%r9d
401226: cmp    %edx,%r9d
401229: jne    401210 <_Z12verifyResultPiS_ii+0x30>
40122b: add    $0x1,%esi
40122e: add    %rbx,%rdi
401231: add    %rbx,%r11
401234: cmp    %ecx,%esi
401236: jne    4011f1 <_Z12verifyResultPiS_ii+0x11>
401238: mov    $0x1,%eax
40123d: pop    %rbx
40123e: retq
40123f: xor    %r9d,%r9d
401242: nopw  0x0(%rax,%rax,1)
401248: mov    %r10d,%ecx
40124b: mov    %r9d,%edx
40124e: mov    $0x401be8,%edi
401253: xor    %eax,%eax
401255: callq 400a40 <printf@plt>
40125a: xor    %eax,%eax
40125c: pop    %rbx
40125d: retq
40125e: mov    $0x1,%eax
401263: retq
```

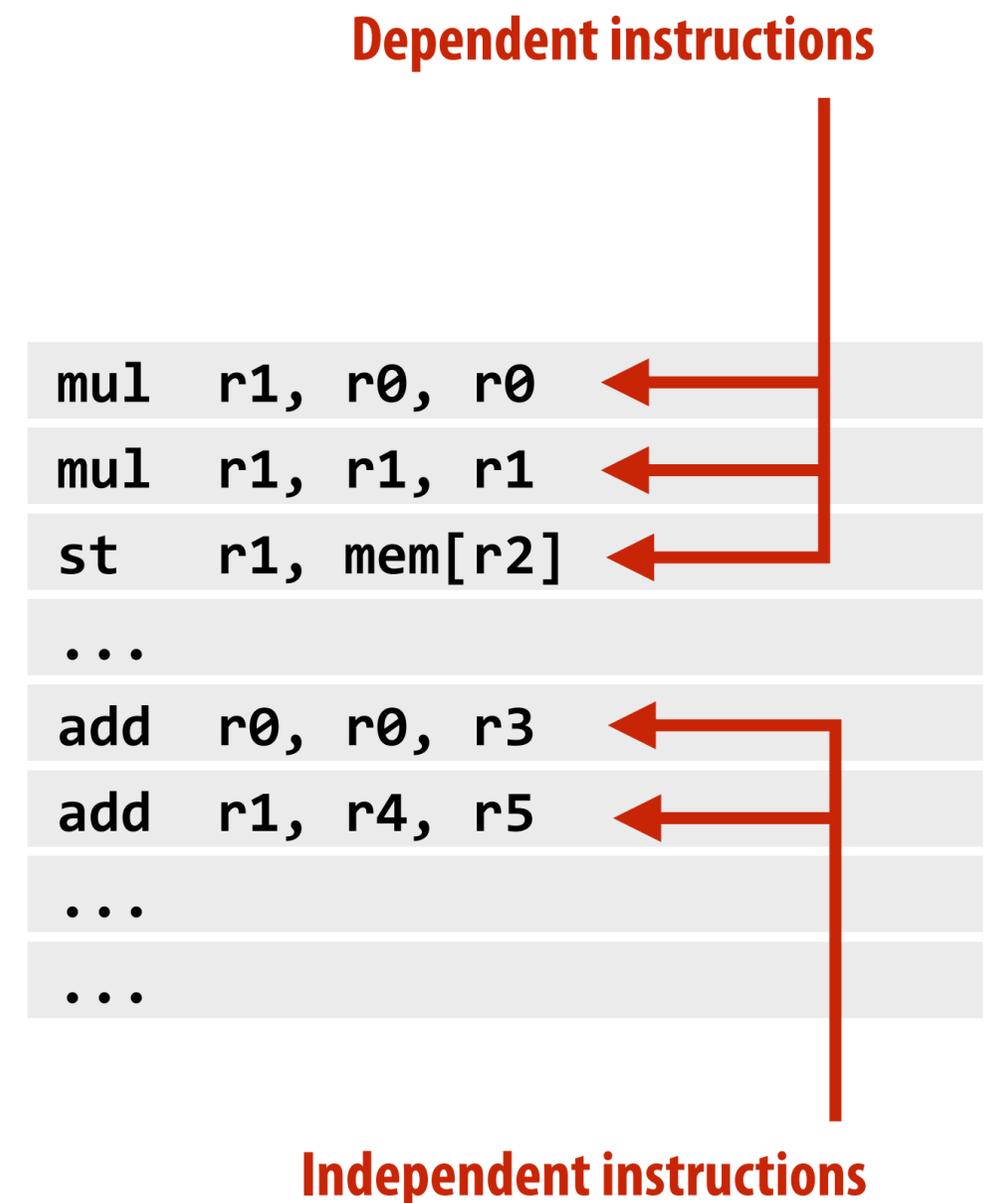


# Instruction level parallelism (ILP)

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

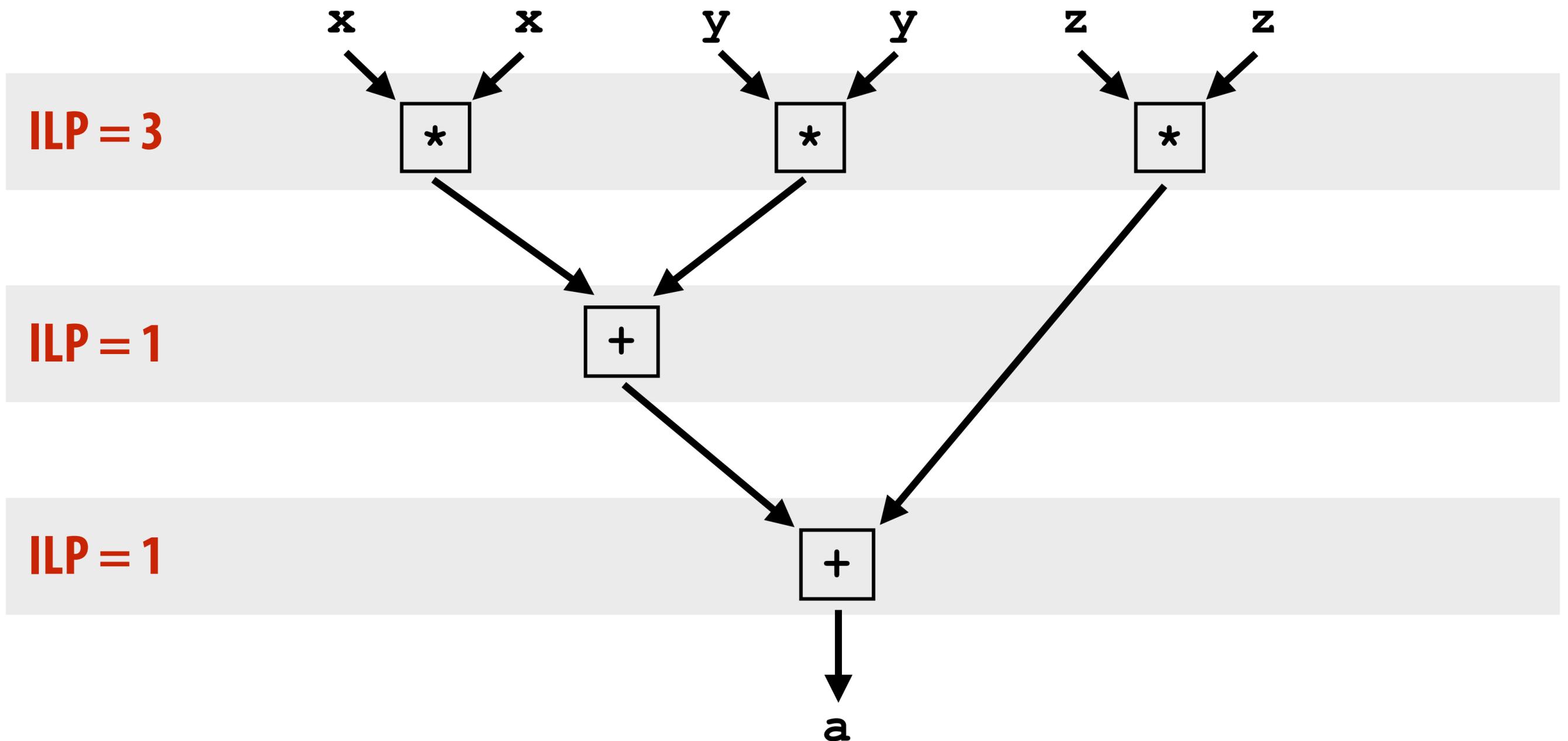
- **Instruction level parallelism (ILP)**

- Idea: Instructions must appear to be executed in program order. BUT independent instructions can be executed simultaneously by a processor without impacting program correctness
- Superscalar execution: processor dynamically finds independent instructions in an instruction sequence and executes them in parallel



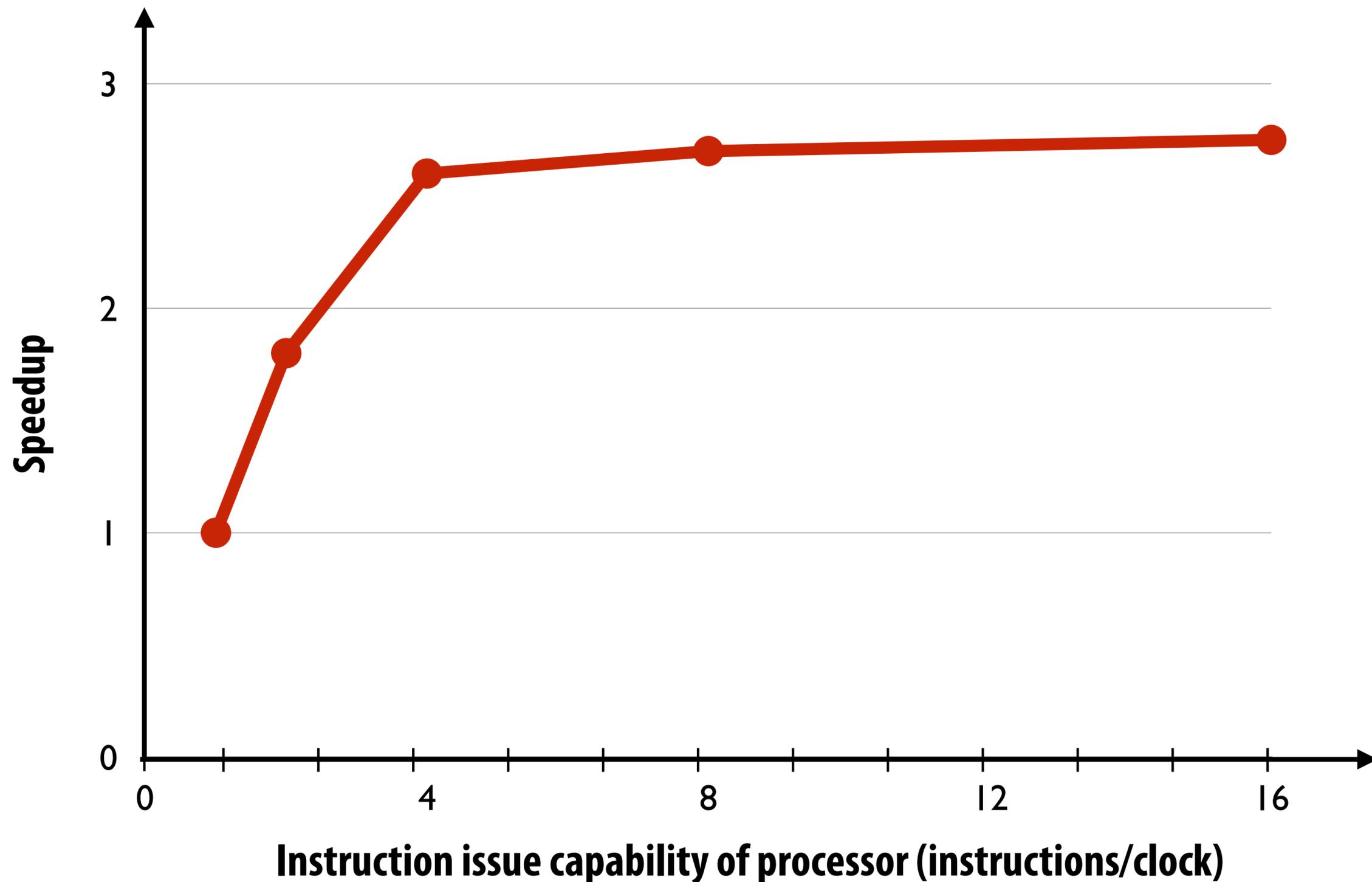
# ILP example

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

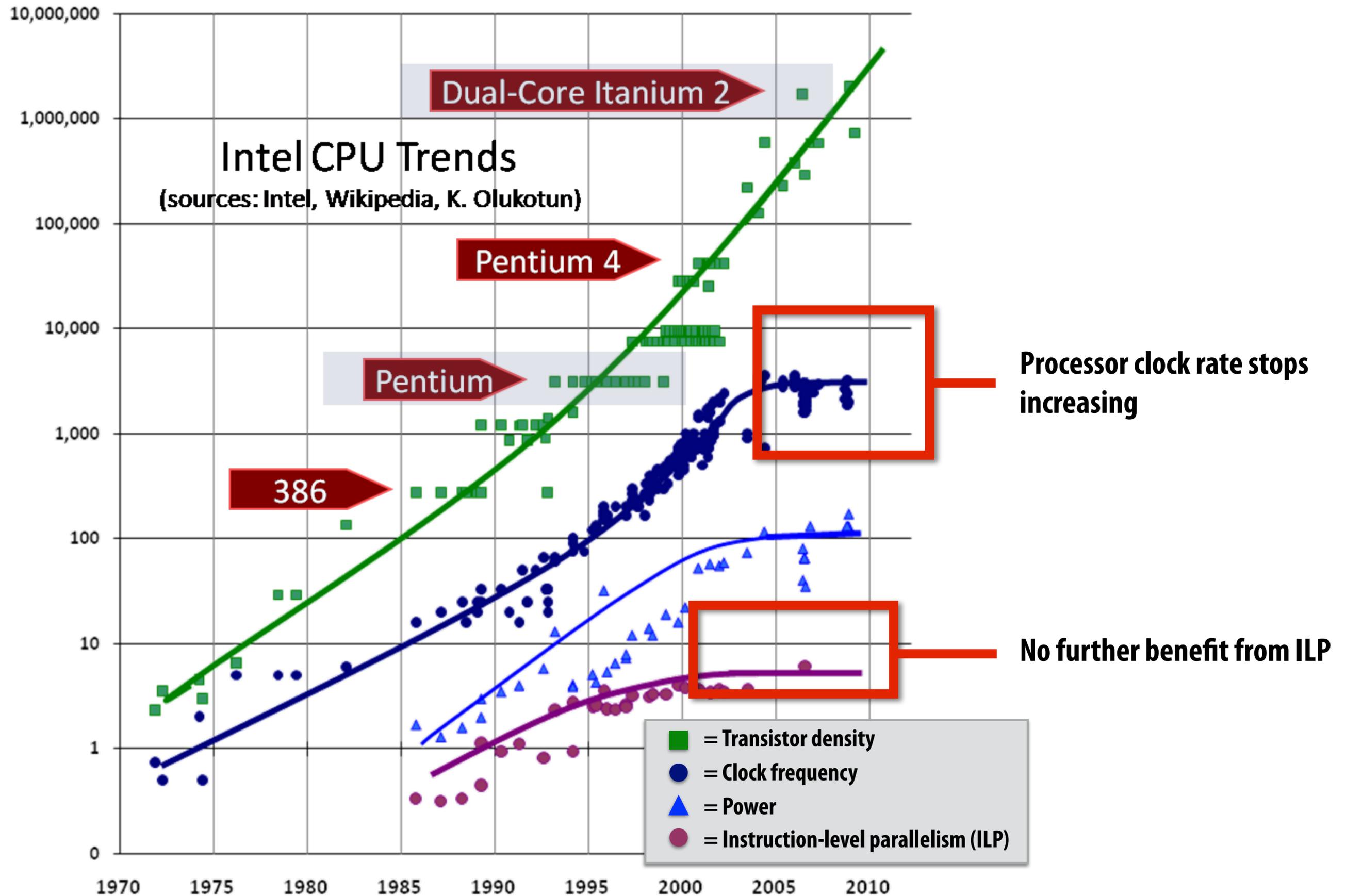


# Diminishing returns of superscalar execution

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)



# ILP tapped out + end of frequency scaling



# The “power wall”

**Power consumed by a transistor:**

**Dynamic power  $\propto$  capacitive load  $\times$  voltage<sup>2</sup>  $\times$  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**

	<u>TDP</u>
<b>Intel Core i7 (in this laptop):</b>	<b>45W</b>
<b>Intel Core i7 2700K (fast desktop CPU):</b>	<b>95W</b>
<b>NVIDIA GTX 780 GPU</b>	<b>250W</b>
<b>Mobile phone processor</b>	<b>1/2 - 2W</b>
<b>World’s fastest supercomputer</b>	<b>megawatts</b>
<b>Standard microwave oven</b>	<b>700W</b>

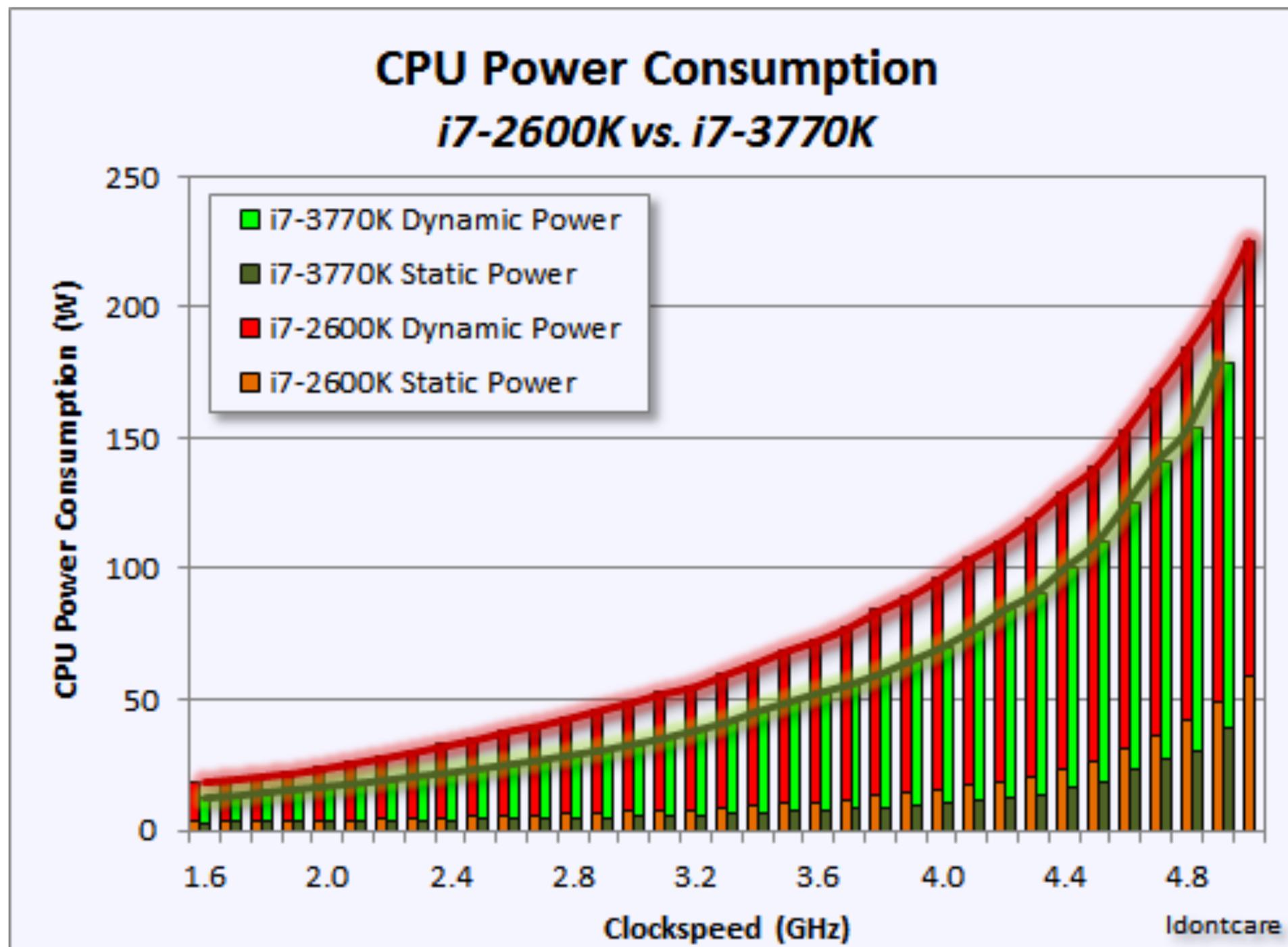


# Power draw as a function of frequency

Dynamic power  $\propto$  capacitive load  $\times$  voltage<sup>2</sup>  $\times$  frequency

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

Maximum allowed frequency determined by processor's core voltage



# Single-core performance scaling

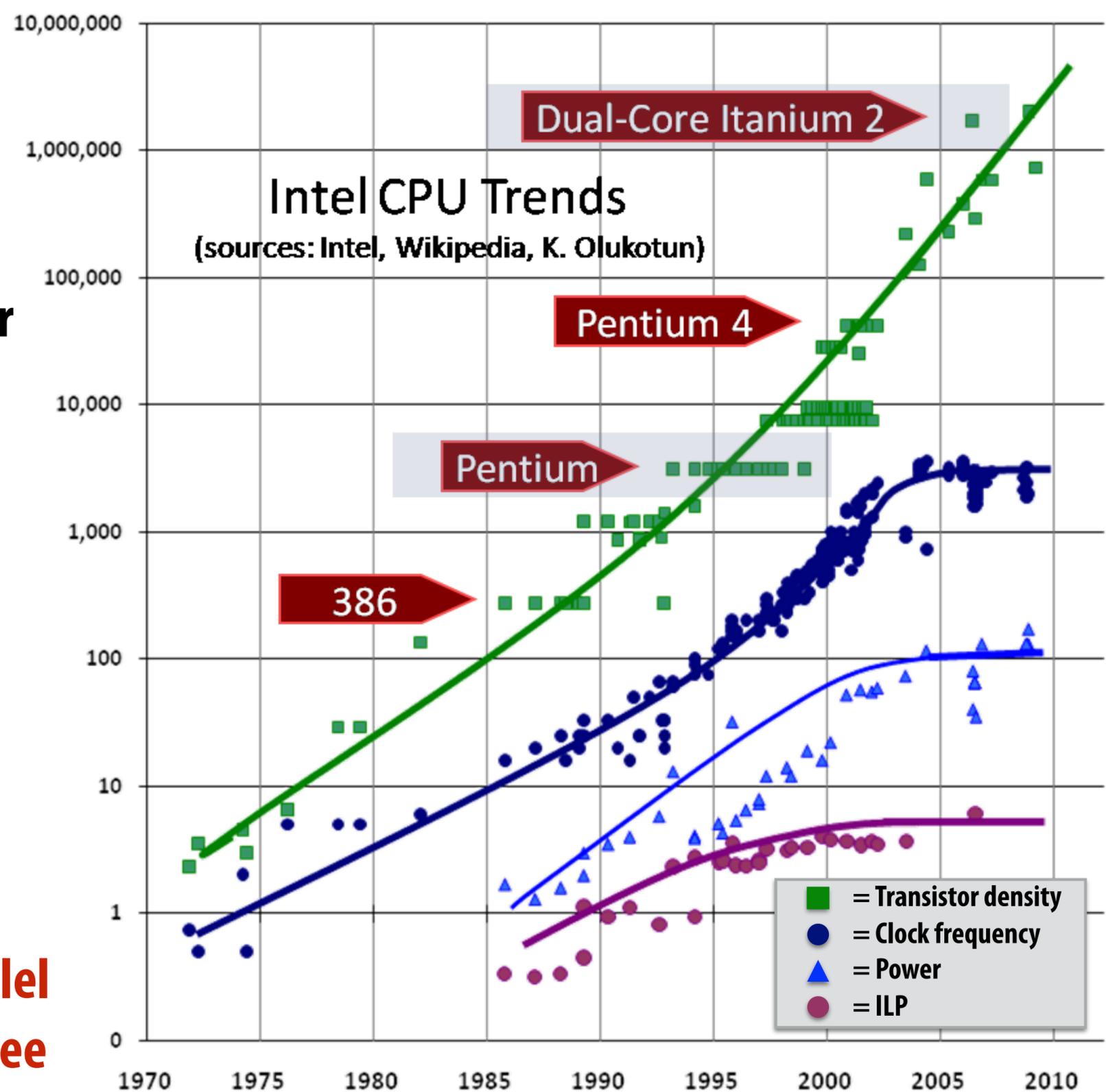
The rate of single-instruction stream performance scaling has decreased (almost to zero)

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!**



# 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 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

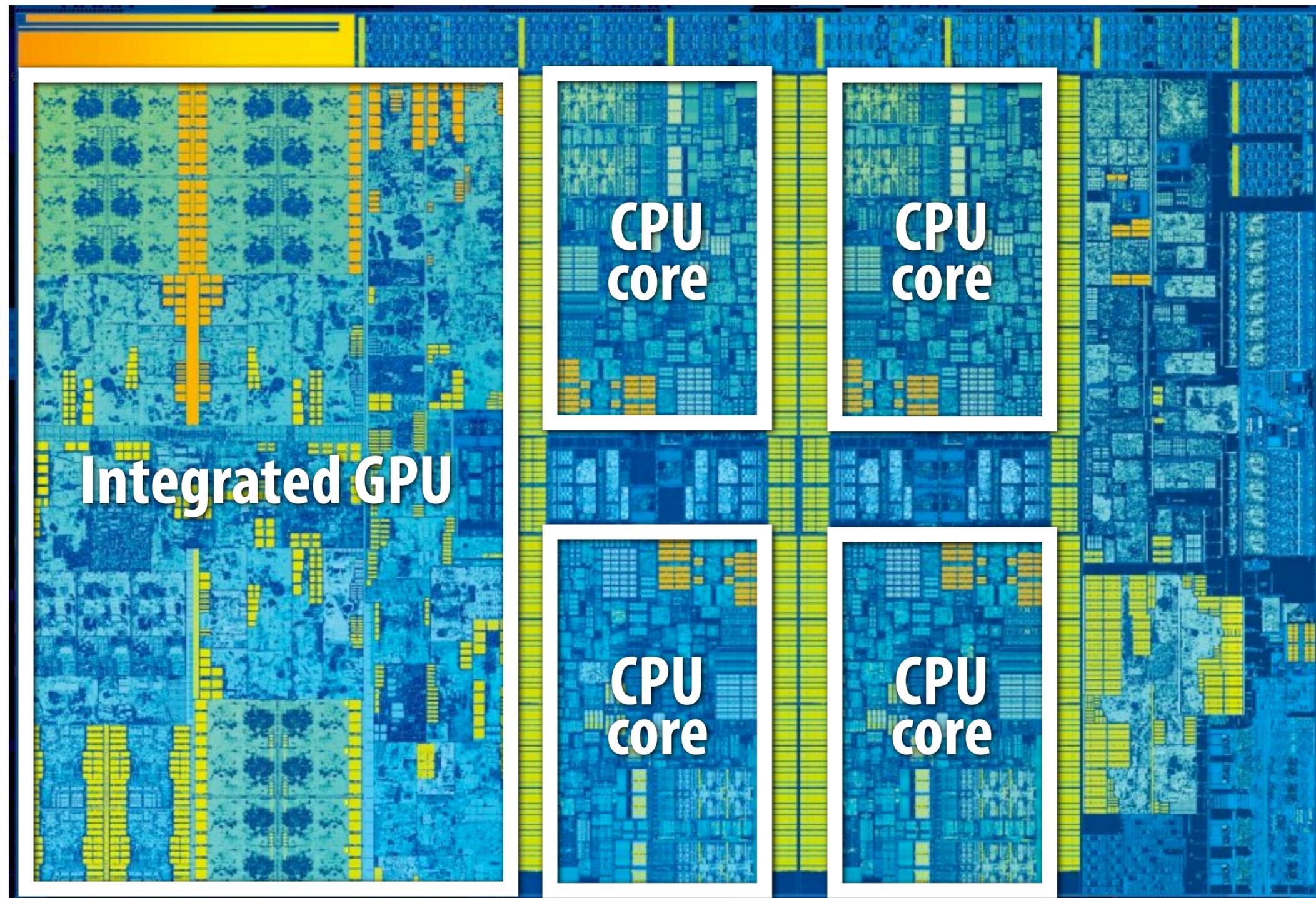
## ■ The answer today:

- Because it is the primary way to achieve significantly higher application performance for the foreseeable future \*

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

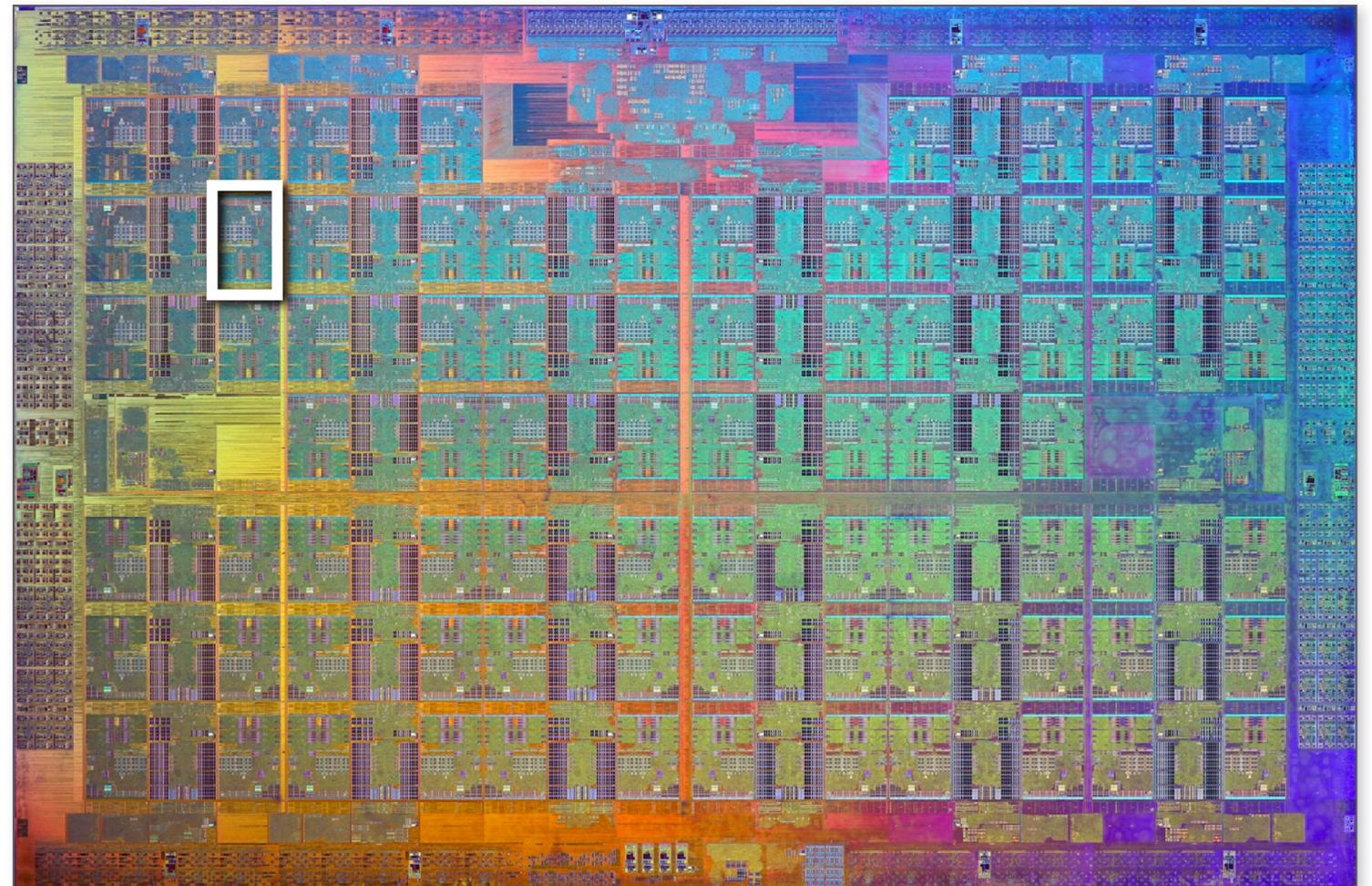
# Intel Skylake (2015) (aka "6th generation Core i7")

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



# Intel Xeon Phi 7290 “coprocessor” (2016)

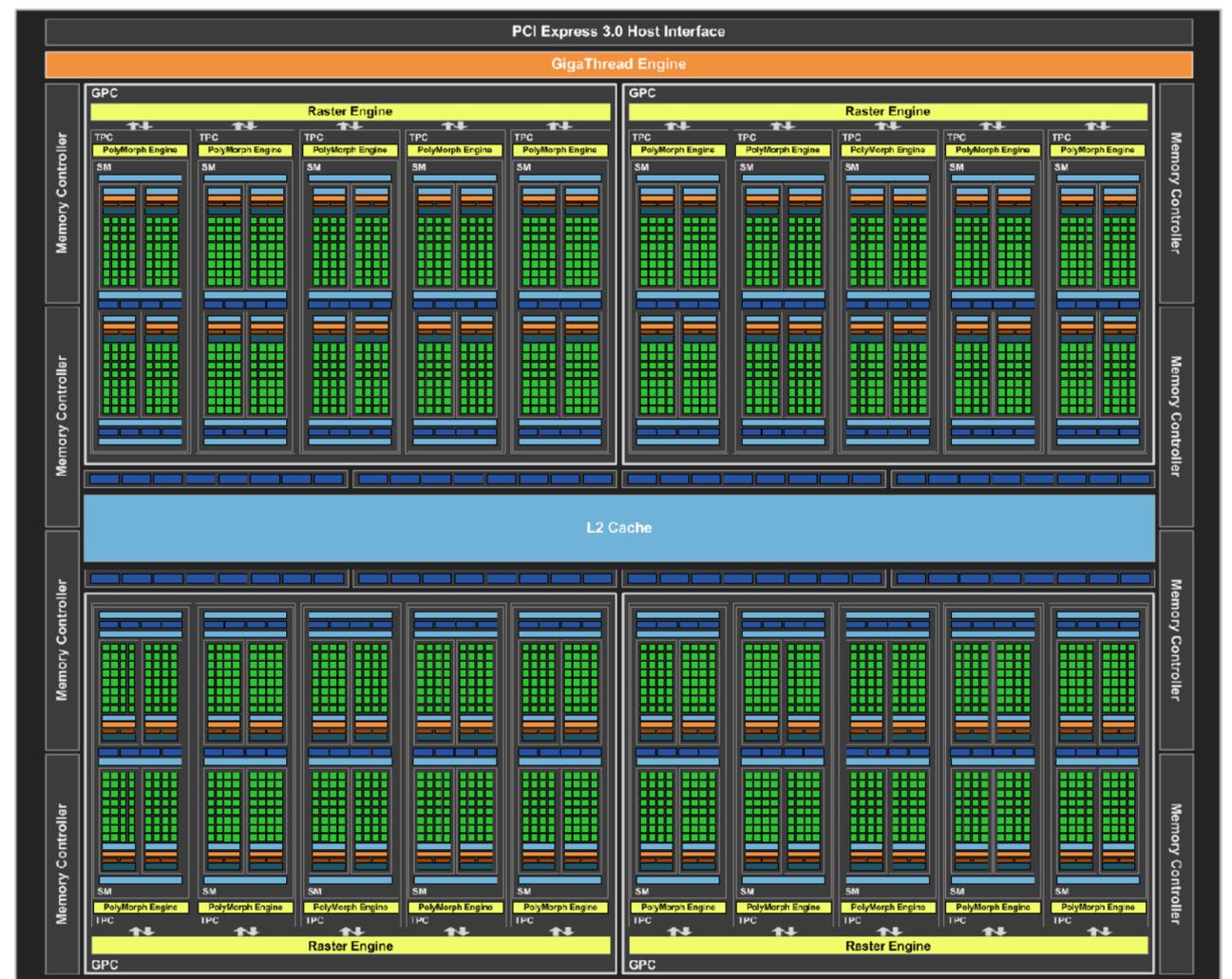
72 cores (1.5 Ghz)



# NVIDIA Maxwell GTX 1080 GPU (2016)

20 major processing blocks

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

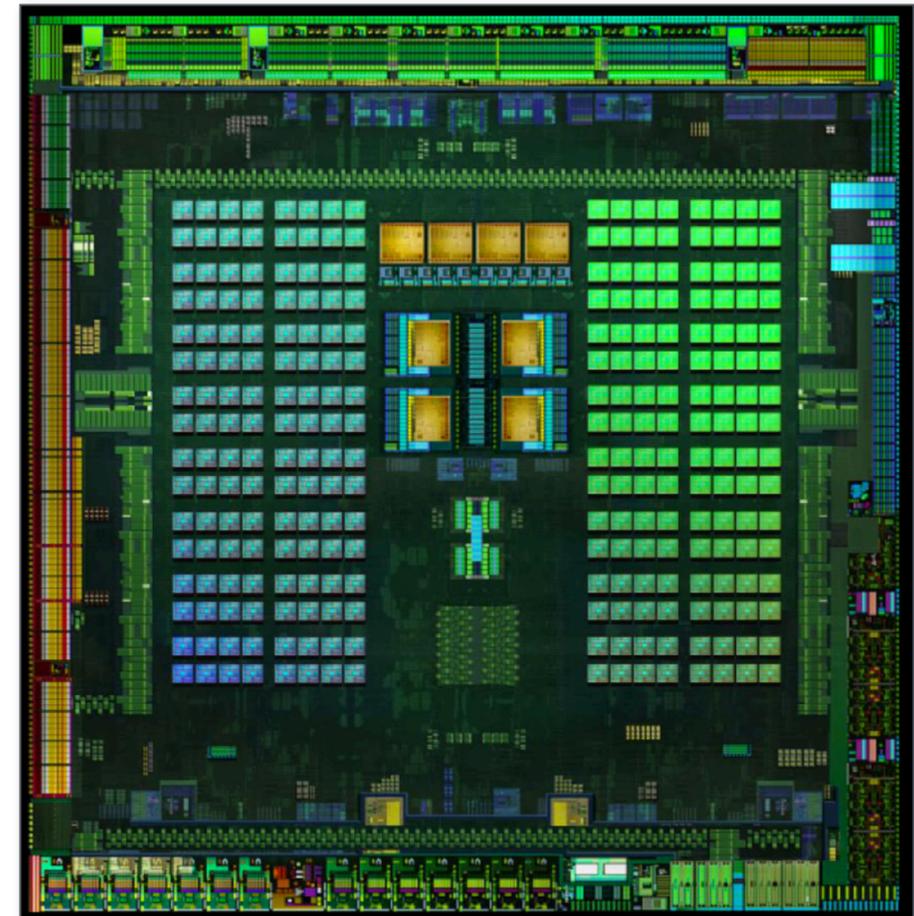


# Mobile parallel processing

**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 X1:  
4 ARM A57 CPU cores +  
4 ARM A53 CPU cores +  
NVIDIA GPU + image processor...**

# Mobile parallel processing

**Raspberry Pi 3**

**Quad-core ARM A53 CPU**



# Supercomputing

- **Today: clusters of multi-core CPUs + GPUs**
- **Oak Ridge National Laboratory: Titan (#3 supercomputer in world)**
  - **18,688 x 16 core AMD CPUs + 18,688 NVIDIA K20X GPUs**



# Summary

- **Today, single-thread-of-control 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 15-418/618!**