Computer “Performance”

Readings: 1.6-1.8

BIPS (Billion Instructions Per Second) vs. GHz (Giga Cycles Per Second)

Throughput (jobs/seconds) vs. Latency (time to complete a job)

Measuring “best” in a computer

Performance Example: Homebuilders

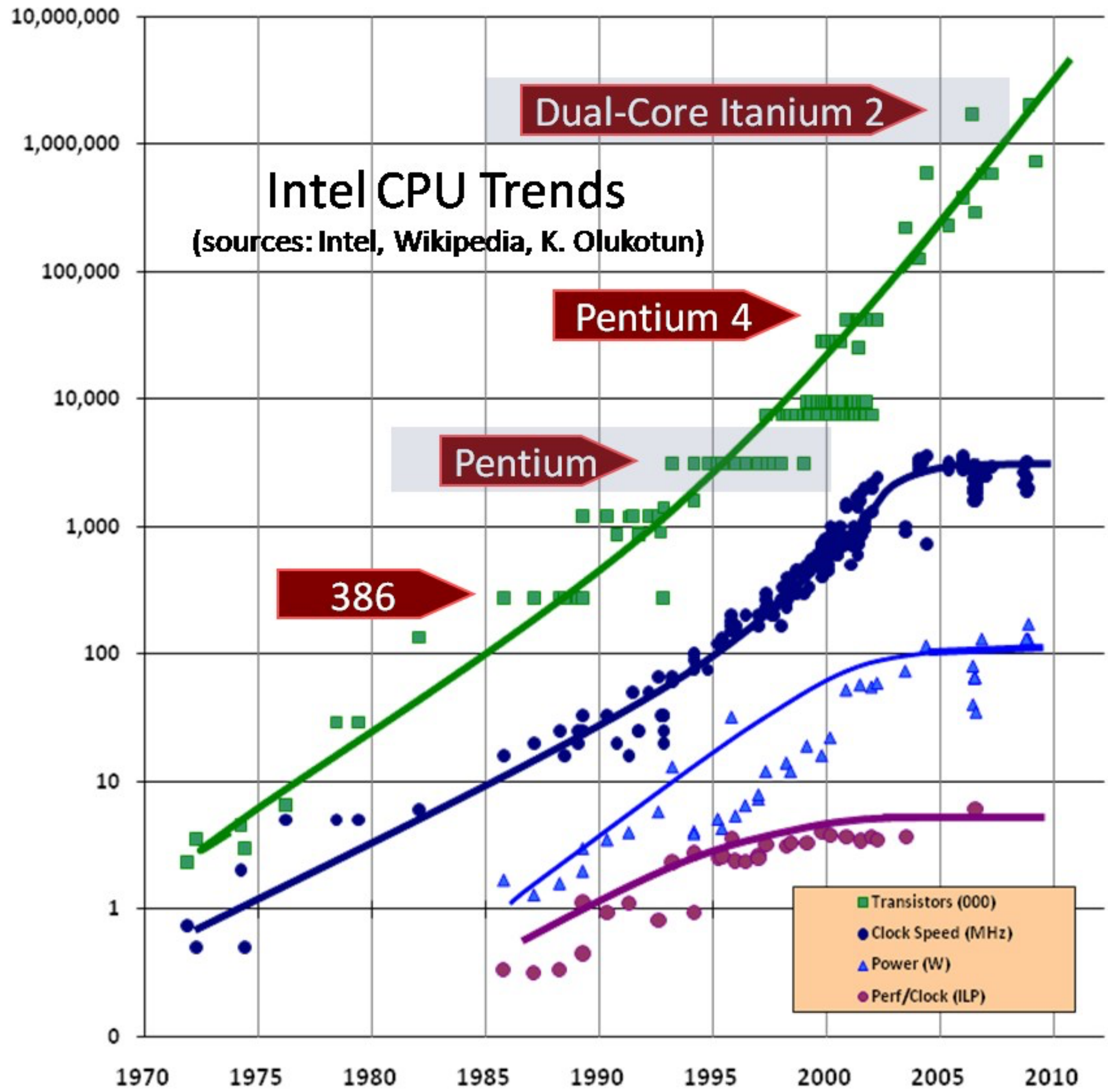
Builder	Time per House	Houses Per Month	House Options	Dollars Per House
Self-build	24 months	1/24	Infinite	$200,000
Contractor	3 months	1	100	$400,000
Prefab	6 months	1,000	1	$250,000

Which is the “best” home builder?

- Homeowner on a budget?
- Rebuilding Haiti?
- Moving to wilds of Alaska?

Which is the “speediest” builder?

- Latency: how fast is one house built?
- Throughput: how long will it take to build a large number of houses?


Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
Dual-Core Itanium 2
Pentium 4
Pentium
386
Transistors (000)
Clock Speed (MHz)
Power (W)
Perf/Clock (ILP)
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
0
1970
1975
1980
1985
1990
1995
2000
2005
2010

Computer Performance

Primary goal: execution time (time from program start to program completion)

$$Performance = \frac{1}{ExecutionTime}$$

To compare machines, we say "X is n times faster than Y"

$$n = \frac{Performance_x}{Performance_y} = \frac{ExecutionTime_y}{ExecutionTime_x}$$

Example: Machine *Orange* and *Grape* run a program

Orange takes 5 seconds, Grape takes 10 seconds

Orange is _____ times faster than Grape

Execution Time

Elapsed Time
- counts everything *(disk and memory accesses, I/O , etc.)*
- a useful number, but often not good for comparison purposes

CPU time
- doesn't count I/O or time spent running other programs
- can be broken up into system time, and user time

Example: Unix “time” command

```
linux15.ee.washington.edu> time javac CircuitViewer.java
3.370u 0.570s 0:12.44 31.6%
```

Our focus: user CPU time
- time spent executing the lines of code that are "in" our program
- But *elapsed time* is hugely important and what matters in the “real world”

CPU Time

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} * \text{Clock period}$$

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} * \frac{1}{\text{Clock rate}}$$

Application example:

A program takes 10 seconds on computer *Orange*, with a 400MHz clock. Our design team is developing a machine *Grape* with a much higher clock rate, but it will require 1.2 times as many clock cycles. If we want to be able to run the program in 6 second, how fast must the clock rate be?

CPI

How do the # of instructions in a program relate to the execution time?

$$\text{CPU clock cycles for a program} = \text{Instructions for a program} * \text{Average Clock Cycles per Instruction } \textbf{(CPI)}$$

$$\text{CPU execution time for a program} = \text{Instructions for a program} * \text{CPI} * \frac{1}{\text{Clock rate}}$$

CPI Example

Suppose we have two implementations of the same instruction set (ISA).

For some program

Machine A has a clock cycle time of 10 ns. and a CPI of 2.0
Machine B has a clock cycle time of 20 ns. and a CPI of 1.2

What machine is faster for this program, and by how much?

Computing CPI

Different types of instructions can take very different amounts of cycles

Memory accesses, integer math, floating point, control flow

$$CPI = \sum_{types} \left(Cycles_{type} * Frequency_{type} \right)$$

Instruction Type	Type Cycles	Type Frequency	Cycles * Freq
ALU	1	50%	
Load	5	20%	
Store	3	10%	
Branch	2	20%	
		CPI:	

CPI & Processor Tradeoffs

Instruction Type	Type Cycles	Type Frequency
ALU	1	50%
Load	5	20%
Store	3	10%
Branch	2	20%

How much faster would the machine be if:

1. A data cache reduced the average load time to 2 cycles?
2. Branch prediction shaved a cycle off the branch time?
3. Two ALU instructions could be executed at once?

Warning 1: Amdahl's Law

The impact of a performance improvement is limited by what is NOT improved:

$$\text{Execution time after improvement} = \text{Execution time of unaffected} + \text{Execution time affected} * \frac{1}{\text{Amount of improvement}}$$

Example: Assume a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to speed up multiply to make the program run 4 times faster?

5 times faster?

Warning 2: BIPs, GHz ≠ Performance

Higher MHz (clock rate) doesn't always mean better CPU

- Orange computer: 1000 MHz, CPI: 2.5, 1 billion instruction program
- Grape computer: 500MHz, CPI: 1.1, 1 billion instruction program

Higher MIPs (million instructions per second) doesn't always mean better CPU

- 1 GHz machine, with two different compilers
- Compiler A on program X: 10 Billion ALU, 1 Billion Load
- Compiler B on program X: 5 Billion ALU, 1Billion Load

Execution Time: A ____ B ____

MIPS: A ____ B ____

Instruction Type	Type Cycles
ALU	1
Load	5
Store	3
Branch	2

Processor Performance Summary

Machine performance:

$$\text{CPU execution time for a program} = \text{Instructions for a program} * \text{CPI} * \frac{1}{\text{Clock rate}}$$

Better performance:

_____ number of instructions to implement computations

_____ CPI

_____ Clock rate

Improving performance must balance each constraint

Example: RISC vs. CISC

CPI = Cycles per instruction

varies by type of instruction and dynamic processor state

useful for rough performance estimation. e.g. “Loads take X” ADDs Y

IPC = Instructions per cycle

In some ways IPC = 1 / CPI but not really

we use IPC in architecture to measure instruction parallelism

Most important thing about measuring performance:

Speedup = 1 /((1 - fraction) + fraction/P)