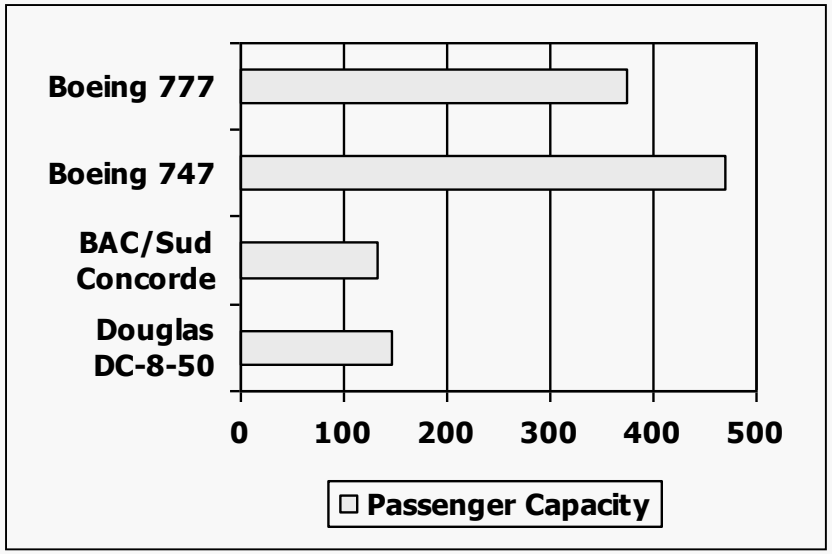
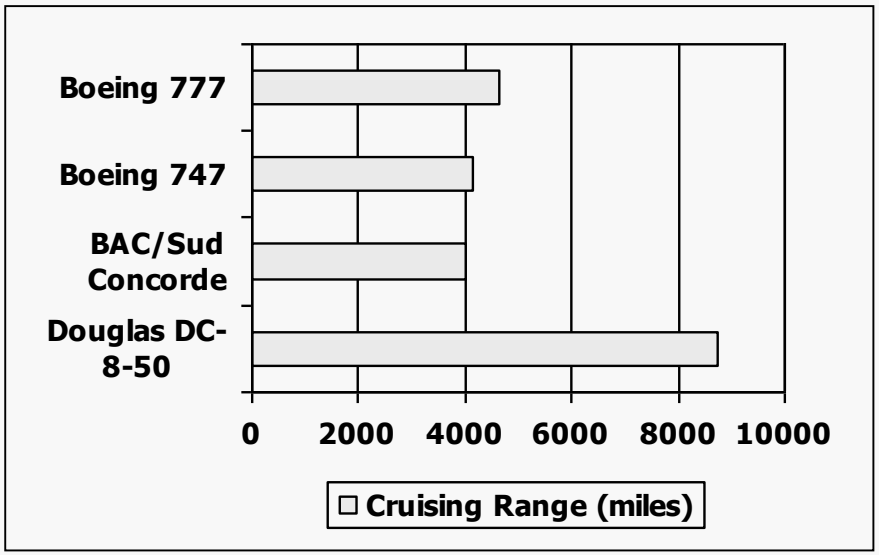
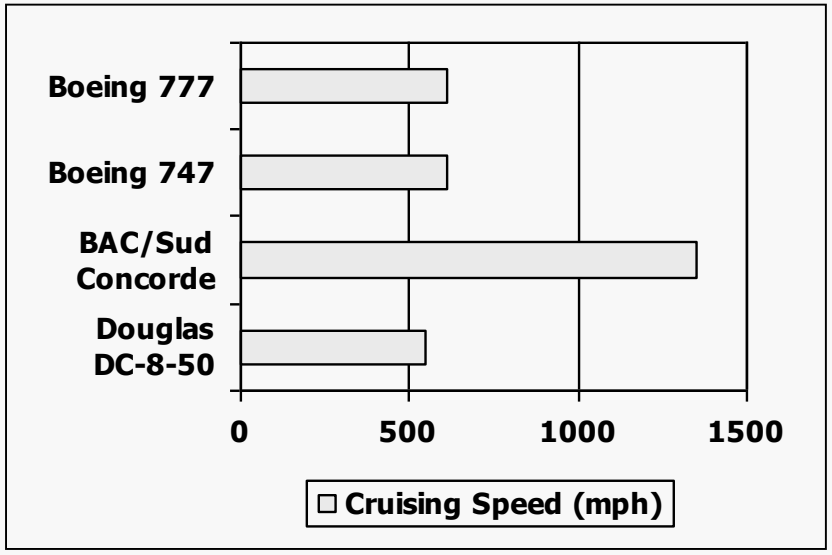
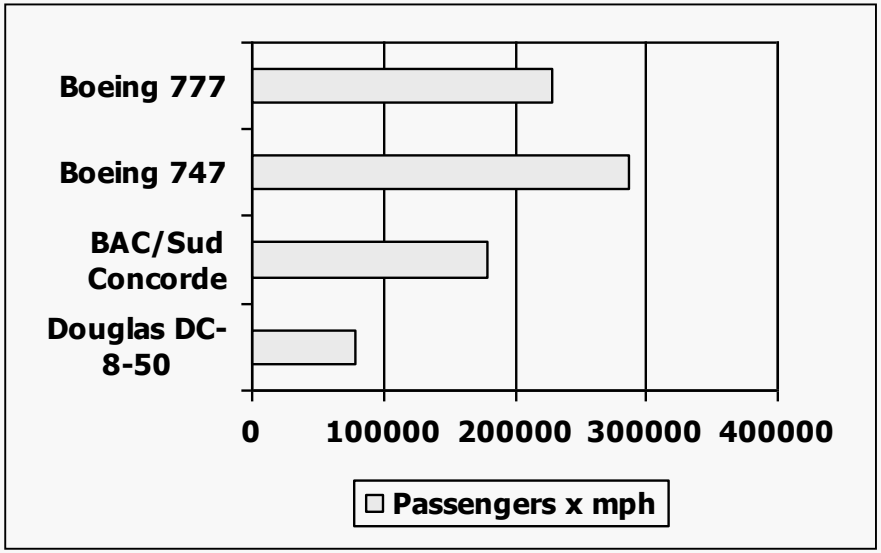
# Defining Performance

Which airplane has the best performance?





©2005-2020 WD McQuain

# Computer Performance: TIME, TIME, TIME

*Response Time* (*latency*): how long it takes to complete a task

*Throughput:* total # of tasks completed per unit time

For now, we will focus on response time… and define

$$\text{Performance} = \frac{1}{\text{Execution Time}}$$


©2005-2020 WD McQuain

# 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

Our focus: user CPU time

- time spent executing the lines of code that are "in" our program


©2005-2020 WD McQuain

# Relative Performance

$$\text{Performance} = \frac{1}{\text{Execution Time}}$$

$$\text{Relative Performance} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

$$= \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} \quad \text{(aka speedup)}$$

Example: time taken to run a program

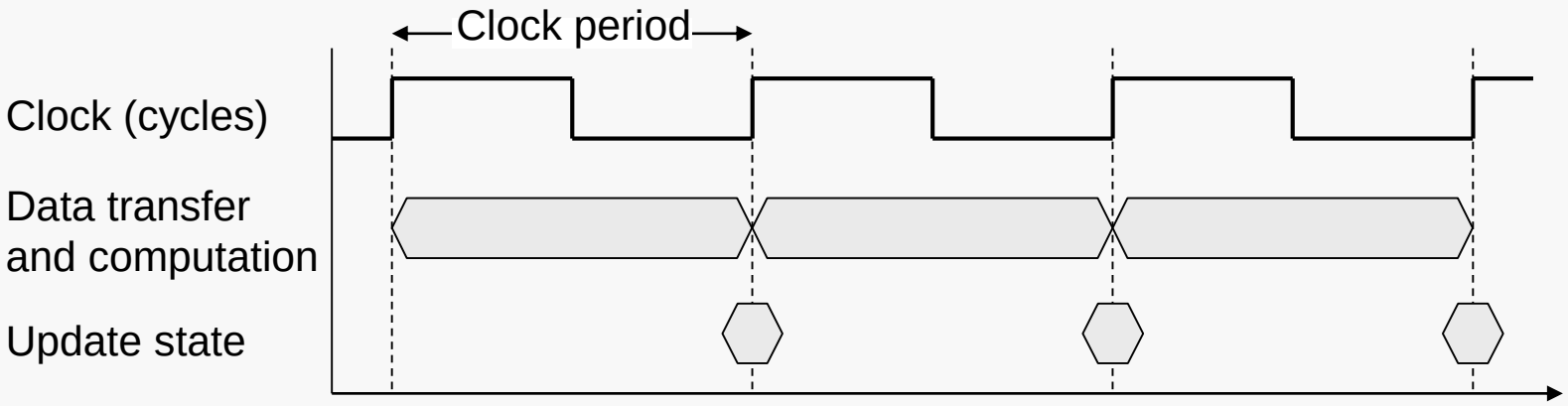
- 10s on A, 15s on B
- $\text{Execution Time}_B$ / $\text{Execution Time}_A$
  = 15s / 10s = 1.5
- So A is 1.5 times as fast as B (and B is 2/3 as fast as A)

wrt this program


©2005-2020 WD McQuain

# CPU Clocking

Operation of digital hardware elements is governed by a constant-rate clock.




Clock period: duration of a clock cycle

- e.g., 250ps = 0.25ns = $250\times10^{-12}$s

Clock frequency (rate): cycles per second

- e.g., 4.0GHz = 4000MHz = $4.0\times10^{9}$Hz

$$\text{Clock Cycle Time} = \frac{1}{\text{Clock Rate}}$$


©2005-2020 WD McQuain

# CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

Smaller is better!

# cycles X time/cycle

Performance improved by

- Reducing number of clock cycles
- Increasing clock rate
- Hardware designer must often trade off clock rate against cycle count


©2005-2020 WD McQuain

# CPU Time Example

Computer A: 2GHz clock, 10s CPU time (to execute a particular program)

Designing Computer B

- Aim for 6s CPU time (to execute the same program)
- Can increase clock rate, but that requires a 20% increase in clock cycles

How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6s}$$

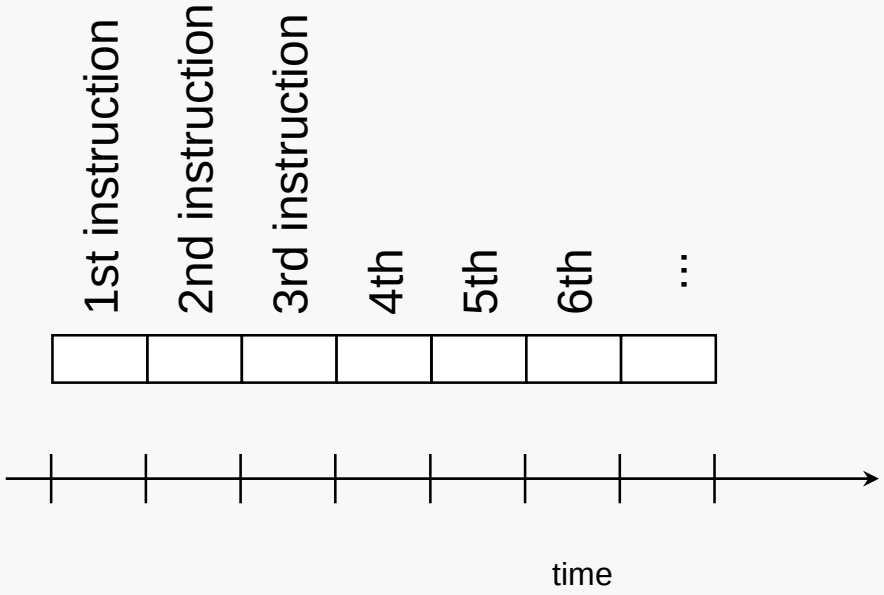
$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$
$$= 10s \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4\text{GHz}$$


©2005-2020 WD McQuain

# How many cycles are required for a program?

Could assume that number of cycles equals number of instructions:




This assumption is incorrect,

- different instructions may take different numbers of cycles on same machine
- same instruction may take different number of cycles on different machines

Why? Hint: remember that these are machine instructions, not lines of C code


©2005-2020 WD McQuain

# Now that we understand cycles…

A given program will require

- some number of instructions (machine instructions)
- some number of cycles
- some number of seconds

We have a vocabulary that relates these quantities:

- cycle time (seconds per cycle)
- clock rate (cycles per second)
- CPI (cycles per instruction)

  *a floating point intensive application might have a higher CPI*

- MIPS (millions of instructions per second)

  *this would be higher for a program using simple instructions*


©2005-2020 WD McQuain

# Performance

Performance is determined by execution time

Do any of the other variables equal performance?

- # of cycles to execute program?
- # of instructions in program?
- # of cycles per second?
- average # of cycles per instruction?
- average # of instructions per second?

Common pitfall: thinking one of these variables (alone) is indicative of performance, when none really is


©2005-2020 WD McQuain

# CPI Example

Computer A: Cycle Time = 250ps, CPI = 2.0
Computer B: Cycle Time = 500ps, CPI = 1.2
Same ISA
For a given program, which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= \text{I} \times 2.0 \times 250\text{ps} = \text{I} \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= \text{I} \times 1.2 \times 500\text{ps} = \text{I} \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{\text{I} \times 600\text{ps}}{\text{I} \times 500\text{ps}} = 1.2$$

…by this much


©2005-2020 WD McQuain

# CPI in More Detail

If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left( \text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$


©2005-2020 WD McQuain

# CPI Example

Alternative compiled code sequences using instructions in classes A, B, C

| Class | A | B | C |
|---|---|---|---|
| CPI for class | 1 | 2 | 3 |
| IC in sequence 1 | 2 | 1 | 2 |
| IC in sequence 2 | 4 | 1 | 1 |

Sequence 1: A B C A C

- IC = 5
- Clock Cycles
  = 2×1 + 1×2 + 2×3
  = 10
- Avg. CPI = 10/5 = 2.0

Sequence 2: A B A A C A

- IC = 6
- Clock Cycles
  = 4×1 + 1×2 + 1×3
  = 9
- Avg. CPI = 9/6 = 1.5


©2005-2020 WD McQuain

# # of Instructions Example

A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C, and they require one, two, and three cycles (respectively).

The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C
The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C.

Which sequence will be faster? How much?

What is the CPI for each sequence?


©2005-2020 WD McQuain

# MIPS example

Two different compilers are being tested for a 4 GHz. machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 2 million Class C instructions.

The second compiler's code uses 10 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

Which sequence will be faster according to MIPS?

Which sequence will be faster according to execution time?


©2005-2020 WD McQuain

# Amdahl's Law

Execution Time After Improvement =

Execution Time Unaffected + ( Execution Time Affected / Amount of Improvement )

Example:

Suppose a program runs in 100 seconds on a machine, with multiply instructions responsible for 80 seconds of this time.

How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?

How about making it 5 times faster?

*Principle: Make the common case fast*


©2005-2020 WD McQuain

# Example 1

Suppose we enhance a machine making all floating-point instructions run five times faster.

If the execution time of some benchmark program before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?


©2005-2020 WD McQuain

# Example 2

We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show a speedup of 3 (i.e., take 1/3 as long to run).

One benchmark we are considering runs for 90 seconds with the old floating-point hardware.

How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?


©2005-2020 WD McQuain

# Remember

Performance is specific to a particular program/s

- Total execution time is a consistent summary of performance

For a given architecture performance increases come from:

- increases in clock rate (if that yields no adverse CPI affects)
- improvements in processor organization that lower CPI
- compiler enhancements that lower CPI (different distribution of instructions) and/or instruction count
- algorithm/language choices that affect instruction count and/or instruction distribution

Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance


©2005-2020 WD McQuain

# Performance Summary

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Performance depends on

- Algorithm: affects IC, possibly CPI
- Programming language: affects IC, CPI
- Compiler: affects IC, CPI
- Instruction set architecture: affects IC, CPI, $T_c$


©2005-2020 WD McQuain

# Benchmarks

*benchmark*

A program or collection of programs selected for use in comparing computer performance

Kinds of benchmarks:

- Kernels (e.g. matrix multiply)
- Toy programs (e.g. sorting)
- Synthetic benchmarks (e.g. Dhrystone)
- Benchmark suites (e.g. SPEC06fp, TPC-C)

A benchmark can be used to isolate different aspects of hardware performance.


©2005-2020 WD McQuain

# SPEC Integer Benchmarks

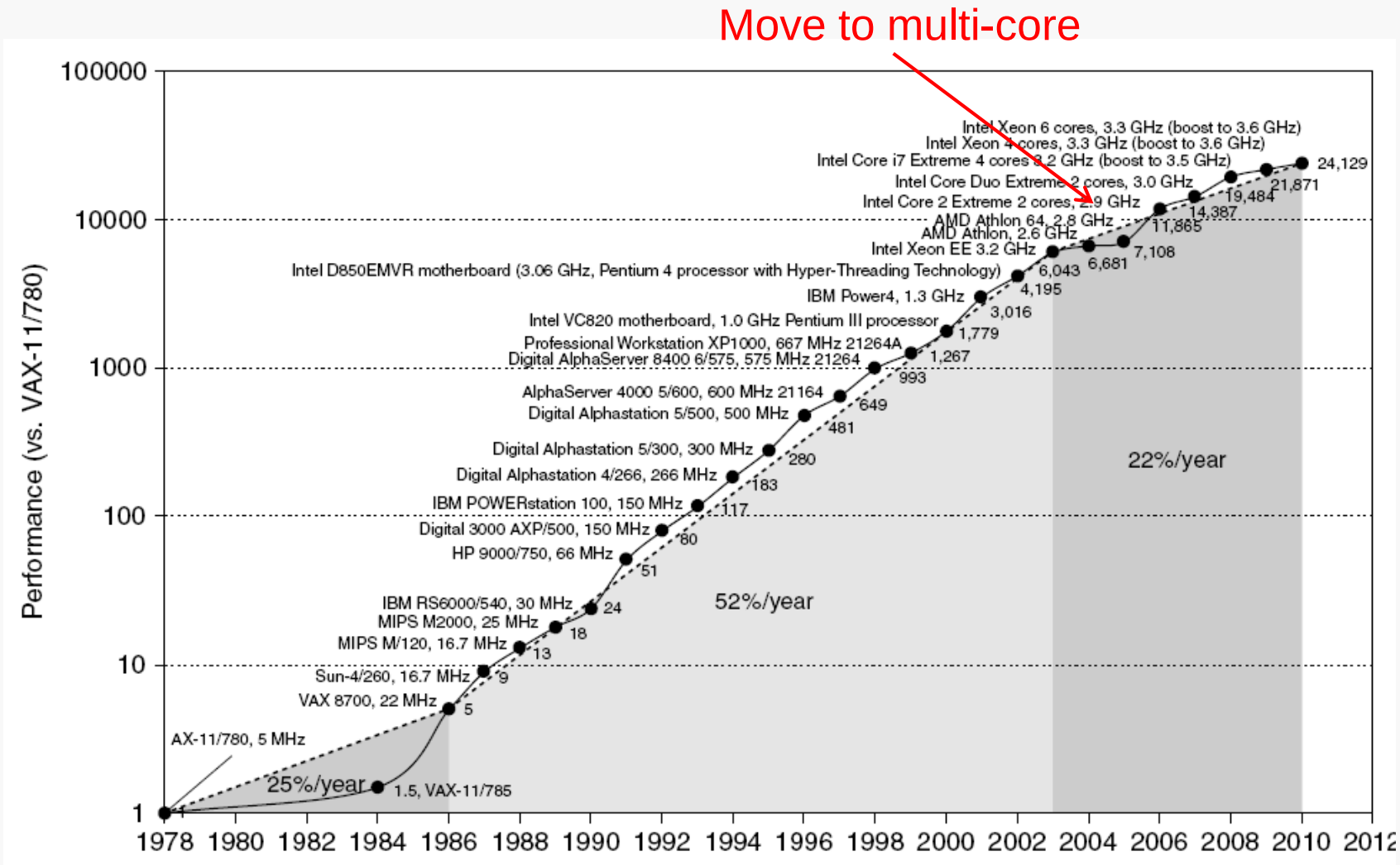
| Description | Name | Instruction Count x $10^9$ | CPI | Clock cycle time (seconds x $10^{-9}$) | Execution Time (seconds) | Reference Time (seconds) | SPECratio |
|---|---|---|---|---|---|---|---|
| Interpreted string processing | perl | 2252 | 0.60 | 0.376 | 508 | 9770 | 19.2 |
| Block-sorting compression | bzip2 | 2390 | 0.70 | 0.376 | 629 | 9650 | 15.4 |
| GNU C compiler | gcc | 794 | 1.20 | 0.376 | 358 | 8050 | 22.5 |
| Combinatorial optimization | mcf | 221 | 2.66 | 0.376 | 221 | 9120 | 41.2 |
| Go game (AI) | go | 1274 | 1.10 | 0.376 | 527 | 10490 | 19.9 |
| Search gene sequence | hmmer | 2616 | 0.60 | 0.376 | 590 | 9330 | 15.8 |
| Chess game (AI) | sjeng | 1948 | 0.80 | 0.376 | 586 | 12100 | 20.7 |
| Quantum computer simulation | libquantum | 659 | 0.44 | 0.376 | 109 | 20720 | 190.0 |
| Video compression | h264avc | 3793 | 0.50 | 0.376 | 713 | 22130 | 31.0 |
| Discrete event simulation library | omnetpp | 367 | 2.10 | 0.376 | 290 | 6250 | 21.5 |
| Games/path finding | astar | 1250 | 1.00 | 0.376 | 470 | 7020 | 14.9 |
| XML parsing | xalancbmk | 1045 | 0.70 | 0.376 | 275 | 6900 | 25.1 |
| Geometric mean | – | – | – | – | – | – | 25.7 |

FIGURE 1.18 SPECINTC2006 benchmarks running on a 2.66 GHz Intel Core i7 920. As the equation on page 35 explains, execution time is the product of the three factors in this table: instruction count in billions, clocks per instruction (CPI), and clock cycle time in nanoseconds.

SPECratio is simply the reference time, which is supplied by SPEC, divided by the measured execution time. The single number quoted as SPECINTC2006 is the geometric mean of the SPECratios.


©2005-2020 WD McQuain

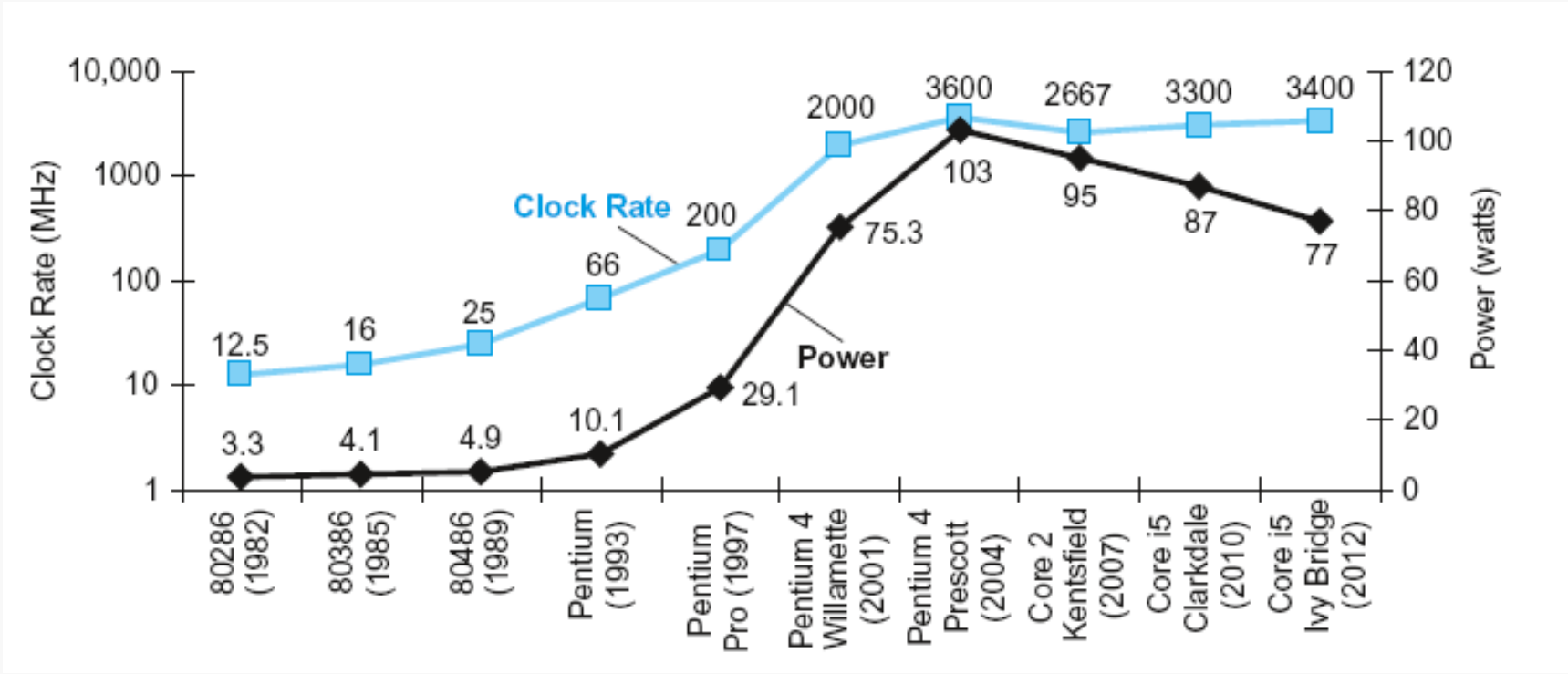
# Single-processor Evolution





©2005-2020 WD McQuain

# Why? The Power Wall




Intel VP Patrick Gelsinger (ISSCC 2001):

> If scaling continues at present pace, by 2005, high speed processors would have power density of nuclear reactor, by 2010, a rocket nozzle, and by 2015, surface of sun.


©2005-2020 WD McQuain