Computer Architecture

Chapter 1

Performance

Performance

- **How to measure, how to report, and how to summarize performance**
- **Useful to make intelligent choices**
- **See through the marketing hype**
- **Key to understanding underlying organizational motivation**

$$\text{performance} = f(\text{factor}_1, \text{factor}_2, \ldots, \text{factor}_n)$$

Why is some hardware better than others for different programs?

What factors of system performance are hardware related?
(e.g., Do we need a new machine, or a new operating system?)

How does the machine's instruction set affect performance?

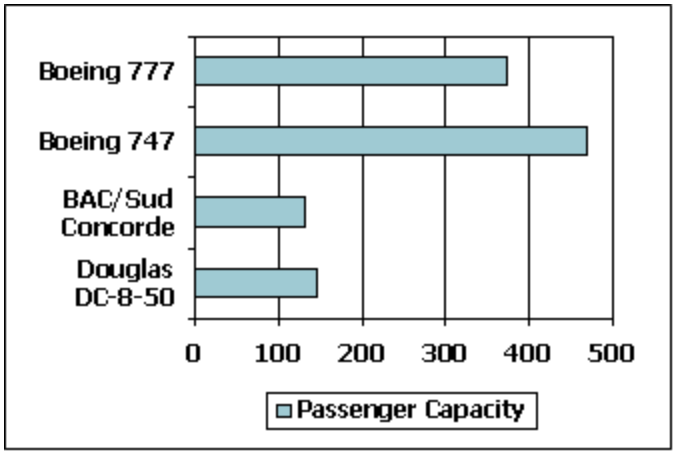

747-400F

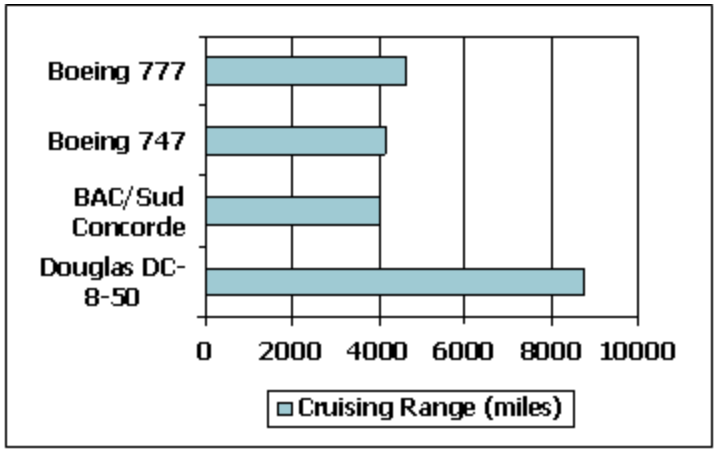

BOEING
777

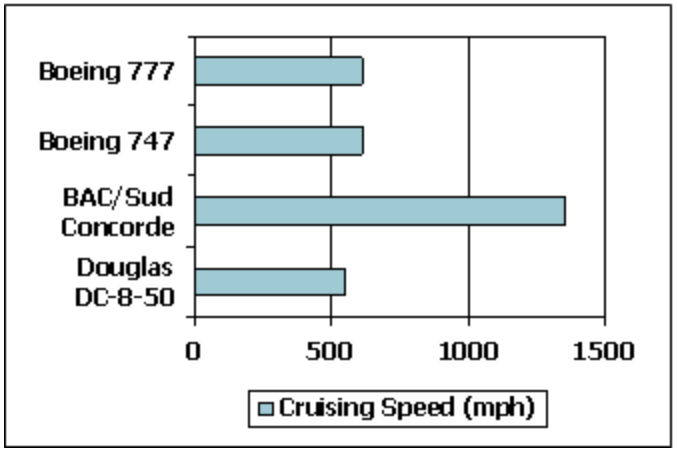

AIR FRANCE

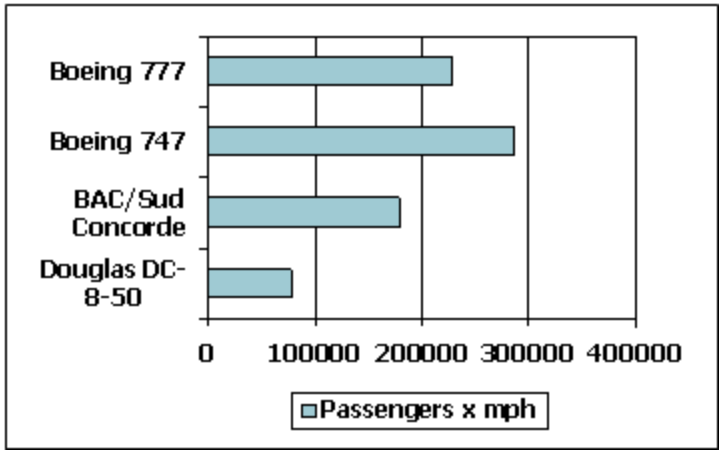

ARAX AIRLINES
Photo Copyright © Felix Goetting
PLANESPOTTERS.NET

Which of these airplanes has the best performance?









- **Highest Speed? Longest Range? Largest Capacity?**
- **Single Person? 450 Passengers?**

Define Performance !

Computer Performance: TIME, TIME, TIME

- **Response Time (latency) : time to do the task**
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?
- **Throughput: tasks per unit time**
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done?

How are response time and throughput affected by
•Replacing the processor with a faster version?
•Adding more processors?

We'll focus on response time for now…

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 (vs. System CPU time)**
  - time spent executing the lines of code that are "in" our program

```
% time myprog
   90.7u          12.9s          2:39           65%

   User CPU       System CPU     Elapsed Time   (CPU/Elapsed)*100
```

Book's Definition of Performance

- **For some program running on machine X,**
  - $\text{Performance}_X = 1 / \text{Execution time}_X$
- **"X is n times faster than Y"**
  - $\text{Performance}_X / \text{Performance}_Y = n$
- **Problem:**
  - machine A runs a program in 10 seconds
  - machine B runs the same program in 15 seconds

$\text{Performance}_A / \text{Performance}_B = n$

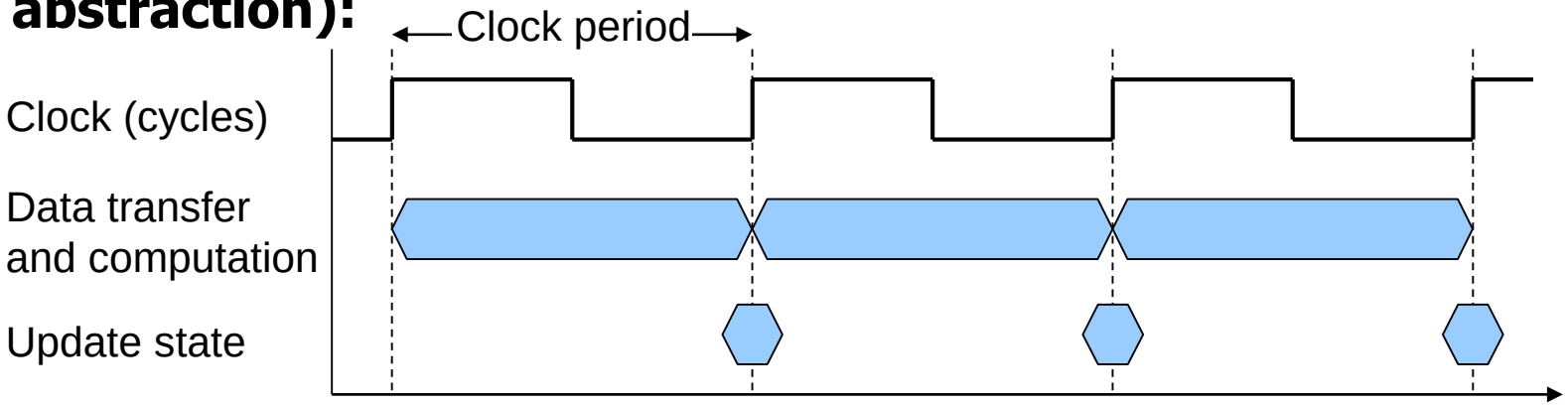
$\text{Execution time}_B / \text{Execution time}_A = n = 1.5$

Clock Cycles

- **Instead of reporting execution time in seconds, we often use cycles**

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycles}}$$

- **Clock ticks indicate when to start activities (one abstraction):**



- **Clock cycle time = time between ticks = seconds per cycle**
- **Clock rate (frequency) = cycles per second (1 Hz. = 1cycle/sec)**

A 200 Mhz clock has a $\frac{1}{200 \times 10^6} \times 10^9 = 5$ nanoseconds cycle time.

How to Improve Performance

$$\text{CPU exec. Time} = \frac{\text{CPU clock cycles}}{\text{program}} \times \text{Clock cycle time}$$

$$= \frac{\text{CPU clock cycles}}{\text{program}} \times \frac{1}{\text{Clock rate}}$$

- **So, to improve performance (everything else being equal) you can either**
  - _____ the # of required cycles for a program, or
  - _____ the clock cycle time or, said another way,
  - _____ the clock rate.

Example

Our favorite program runs in 10 seconds on computer A, which has a 400 Mhz. clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds.

The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program.

What clock rate should we tell the designer to target?

Solution

- **CPU time$_A$ = 10 sec = CPU clock cycles$_A$ / Clock rate$_A$**

- **CPU time$_B$ = 6 sec = 1.2 x CPU clock cycles$_A$ / Clock rate$_B$**

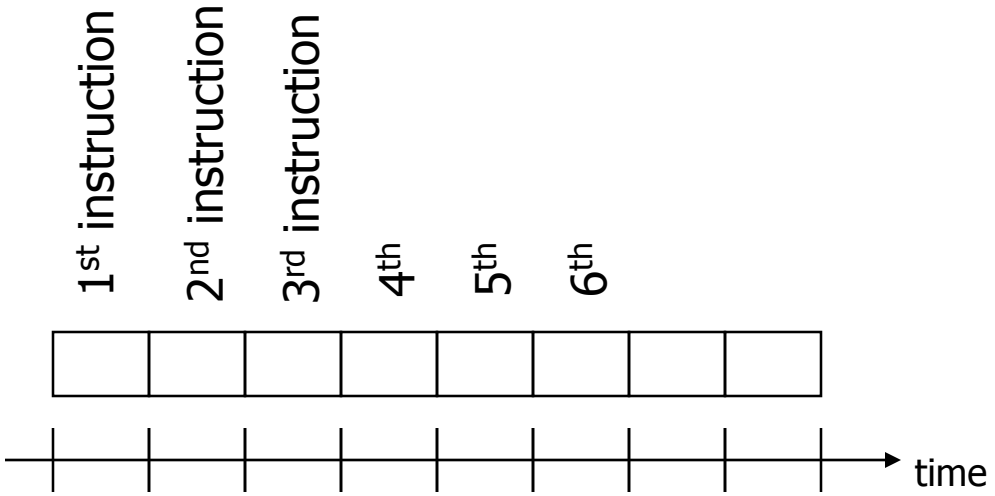
Solution

- **CPU time$_A$ = 10 sec = CPU clock cycles$_A$ / Clock rate$_A$**
  CPU clock cycles$_A$ = 10 sec x 400 x 10^6 cycles/sec
  = 4000 x 10^6 cycles

- **CPU time$_B$ = 6 sec = 1.2 x CPU clock cycles$_A$ / Clock rate$_B$**
  Clock rate$_B$ = 1.2 x 4000 x 10^6 cycles / 6 sec
  = 800 x 10^6 cycles/sec = 800 MHz

How many cycles are required for a program?

- **Could assume that # of cycles = # of instructions**

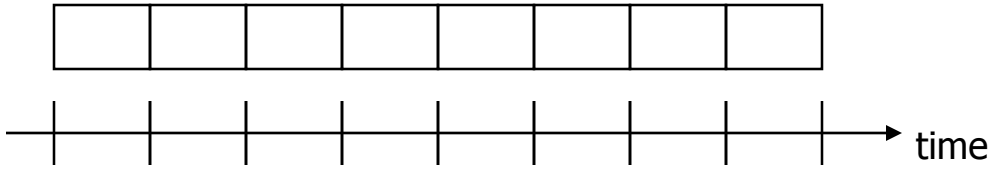


This assumption is incorrect,

different instructions take different amounts of time on different machines.

Why?

Different numbers of cycles for different instructions



- **Multiplication takes more time than addition**
- **Floating point operations take longer than integer ones**
- **Accessing memory takes more time than accessing registers**

Important point:
changing the cycle time often changes the number
of cycles required for various instructions (more later)

Relating Processor Metrics

- **(CPU) Execution time per program**
  - **= CPU clock cycles/program X Clock cycle time**
  - **= CPU clock cycles/program X 1/Clock rate (frequency)**

- **CPU clock cycles/program**
  - **= Instructions/program X Clock cycles Per Instruction**
- **Clock cycles Per Instruction (CPI)**

Execution time = Seconds/Program
= Instructions/Program x Cycles/Inst. X Seconds/Cycle

Performance is determined by execution time.

Clock Cycles Per Instruction (CPI)

- **CPI = average # of clock cycles per instruction**
  - CPI = Clock Cycles / Instruction Count (IC)

- **Affected by both:**
  - cost for each instruction type (implementation, ISA)
  - the frequency of different instructions (Instruction Mix, program)

- **How to compute CPI**

For n instruction types,

$$CPI = \sum_{i=1}^{n} CPI_i \times F_i \qquad \text{where } F_i = \text{Instruction Count}_i / \text{Instruction Count}$$

Example CPI Calculation

Inst. Type	Freq F_i	Cycles CPI_i	CPI Contribution	% time
ALU	50%	1	0.5	33%
Load	20%	2	0.4	27%
Store	10%	2	0.2	13%
Branch	20%	2	0.4	27%
	Typical Mix		1.5	

CPI Example

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

- For some program,
  Machine A has a clock cycle time of 1 ns. and a CPI of 2.0
  Machine B has a clock cycle time of 2 ns. and a CPI of 1.2

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

Solution

- **Same ISA, so the same number N of instructions.**
- **CPU clock cycles$_A$ = N x 2.0**
- **CPU clock cycles$_B$ = N x 1.2**
- **CPU time$_A$ = N x 2.0 x 1 = 2.0N ns**
- **CPU time$_B$ = N x 1.2 x 2 = 2.4N ns**

$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A}$$

$$= \frac{2.4N}{2N} = 1.2$$

Alternative Performance Metrics: MIPS

- **Something other than time**
- **MIPS: Millions of Instructions Per Second**

  MIPS = Instruction Count / (Time x 10^6)

  = Clock Rate / (CPI x 10^6)
- **Flaws in using MIPS**
  - Machines with different instruction sets?
    - 1 instruction for 1 sec. VS. 10 instructions for 1 sec
  - Programs with different instruction mixes?

- **Can be uncorrelated with performance!**

MIPS example

- **Two different compilers are being tested for a 4 GHz. machine with three different classes of instructions:**

Instruction class	CPI for this class
A	1
B	2
C	3

Both compilers are used to produce code for a large piece of software.

Code sequence	Instruction counts for inst class		
	A	B	C
Compiler 1	5B	1B	1B
Compiler 2	10B	1B	1B

Which sequence will be faster according to MIPS?
Which sequence will be faster according to execution time?

Solution

- CPU clock cycles$_1$ = (5×1 + 1×2 + 1×3) × 10^9 =10×10^9
- CPU clock cycles$_2$ = (10×1 + 1×2 + 1×3) × 10^9 = 15×10^9

- Exec. time$_1$ =

- Exec. Time$_2$ =

- MIPS1 =

- MIPS2 =

Solution

- CPU clock cycles$_1$ = $(5\times1 + 1\times2 + 1\times3) \times 10^9 = 10\times10^9$
- CPU clock cycles$_2$ = $(10\times1 + 1\times2 + 1\times3) \times 10^9 = 15\times10^9$

- Exec. time$_1$ = $\dfrac{10\times10^9}{4\times10^9}$ = 2.5 sec
- Exec. Time$_2$ = $\dfrac{15\times10^9}{4\times10^9}$ = 3.75 sec
- MIPS1 = $\dfrac{(5+1+1) \times 10^9}{2.5\times10^6}$ = 2800
- MIPS2 = $\dfrac{(10+1+1) \times 10^9}{3.75\times10^6}$ = 3200

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:**
  - clock cycle time (seconds per cycle)
  - clock rate (cycles per second)
  - CPI (clock cycles per instruction)
  - MIPS (millions of instructions per second)

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 the variables is indicative of performance when it really isn't.

of Instructions Example

- **A compiler designer is trying to decide between two code sequences for a particular machine.**

Instruction class	CPI for this class
A	1
B	2
C	3

Code sequence	Instruction counts for inst class		
	A	B	C
1	2	1	2
2	4	1	1

Which sequence will be faster? How much?
What is the CPI for each sequence?

Solution

- # of instructions$_1$ = 5
- # of instructions$_2$ = 6
- CPU clock cycles$_1$ = 2 + 2 + 6 = 10 cycles
- CPU clock cycles$_2$ = 4 + 2 + 3 = 9 cycles

$CPI_1 = 10/5 = 2$ $CPI_2 = 9/6 = 1.5$

Understanding Program Performance

Hardware or software component	Affects what?	How?
Algorithm	Instruction count, possibly CPI	The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more divides, it will tend to have a higher CPI.
Programming language	Instruction count, CPI	The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher CPI instructions.
Compiler	Instruction count, CPI	The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in complex ways.
Instruction set architecture	Instruction count, clock rate, CPI	The instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor.

Benchmarks

- **Performance best determined by running a real application**
  - Use programs typical of expected workload
  - Or, typical of expected class of applications

    e.g., compilers/editors, scientific applications, graphics, etc.
  - Difficult to find how to speed up benchmarks (i.e., analysis too difficult)
- **Small benchmarks**
  - nice for architects and designers (esp. in the early stages)
  - easy to standardize
  - easy to implement
  - can be abused
- **Report: reproducibility**

Performance Summary

	Computer A	Computer B
Program 1	**1**	**10**
Program 2	**1000**	**100**
Total time	**1001**	**110**

- **How to characterize with a single number?**
- **Arithmetic mean =** $\frac{1}{n}\sum_{i=1}^{n}\text{Time}_i$
- **Weighted AM =** $\sum_{i=1}^{n}\text{w}_i \times \text{Time}_i$

Normalized Performance

- **Normalized execution time is handy for scaling performance**

	P1 secs	P2 secs	Relative to A			Relative to B		
			P1	P2	AM	P1	P2	AM
A	10	50	1	1	1	0.1	50	25
B	100	1	10	0.02	5	1	1	1

  - From total execution time, A is faster than B by the factor of 101/60 = 1.7 (equal # of runs of P1 and P2)
  - Arith. Mean of relative time say either
    - A is 5 times faster than B
    - B is 25 time faster than A

- **Do not summarize *relative execution time with Arith. Mean*.**

Geometric Mean $\sqrt[n]{\prod \text{Exec. Time Ratio}_i}$

- **Geometric mean used to summarize relative measure**

	P1 secs	P2 secs	Relative to A			Relative to B		
			P1	P2	GM	P1	P2	GM
A	10	50	1	1	1	0.1	50	2.2
B	100	1	10	0.02	0.45	1	1	1

  - Relative to A : B is 2.2X faster
  - Relative to B : B is 2.2X faster

- **(BUT) two unfortunate properties**
  - Drawback 1: does not track execution time
  - Drawback 2: rewards all improvements equally
    - On machine B, contribution of 50 sec in P1
      = contribution of 0.5 sec in P2.

SPEC (System Performance Evaluation Cooperative)

- **5 companies have agreed on a set of real programs and inputs (Apollo, HP, DEC, MIPS, Sun) in 1988.**
- **SPEC89: normalized to a VAX-11/780**
- **SPEC2000: normalized to a Sun Ultra5_10 300MHz**
- **SPEC2006**
  - 12 integer benchmarks (CINT2006)
  - 17 floating-point benchmarks (CFP2006)

SPECratio

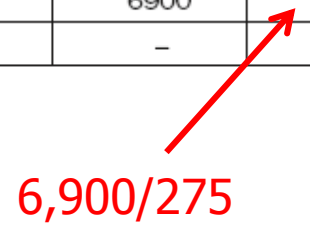
- **For each SPEC benchmark program p (1≤p≤N), measure T_p.**
- **Compute geometric mean:**

$$\sqrt[N]{\prod_{i=1}^{N} \frac{T_{reference(p)}}{T_p}}$$

- **BUT:**
  - What workload does this match? NONE
  - What good is the geometric mean? NONE

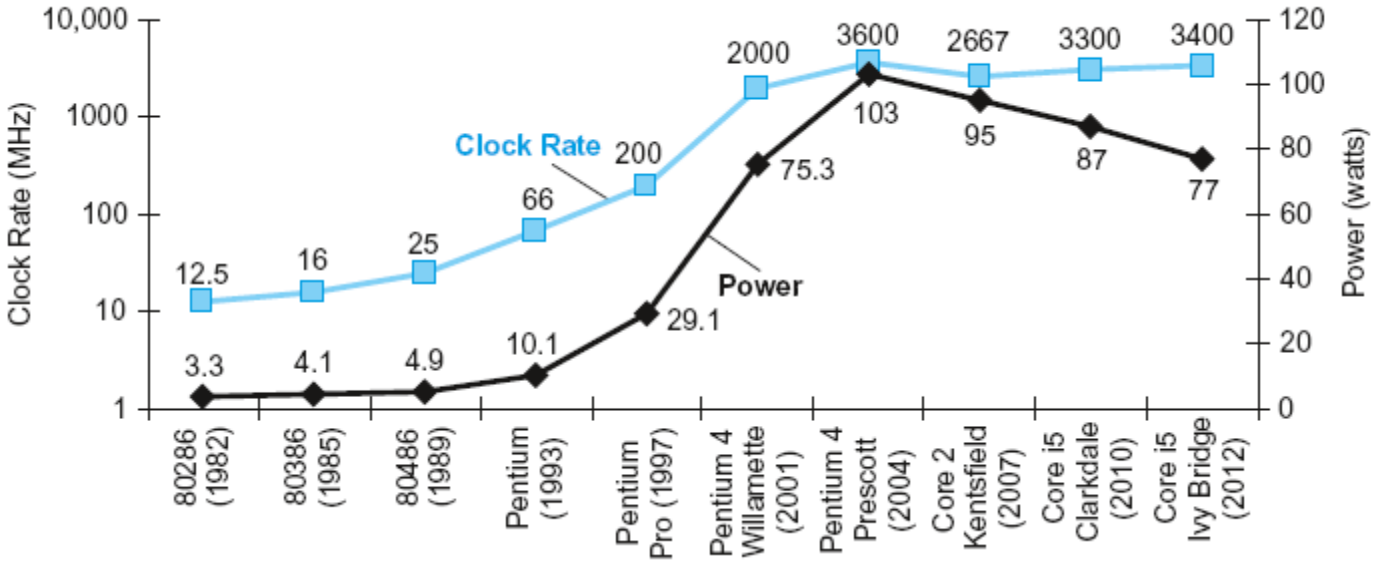
CINT2006 for Intel Core i7 920

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



Power is also an important metric

- **Power increasing rapidly**
- **Power = Capacitive load X Voltage^2 X Frequency switched**



Clock grew by a factor of 1000 while power grew by only a factor of 30
Why?

Voltage was reduced about 15% per generation (5V → 1V)

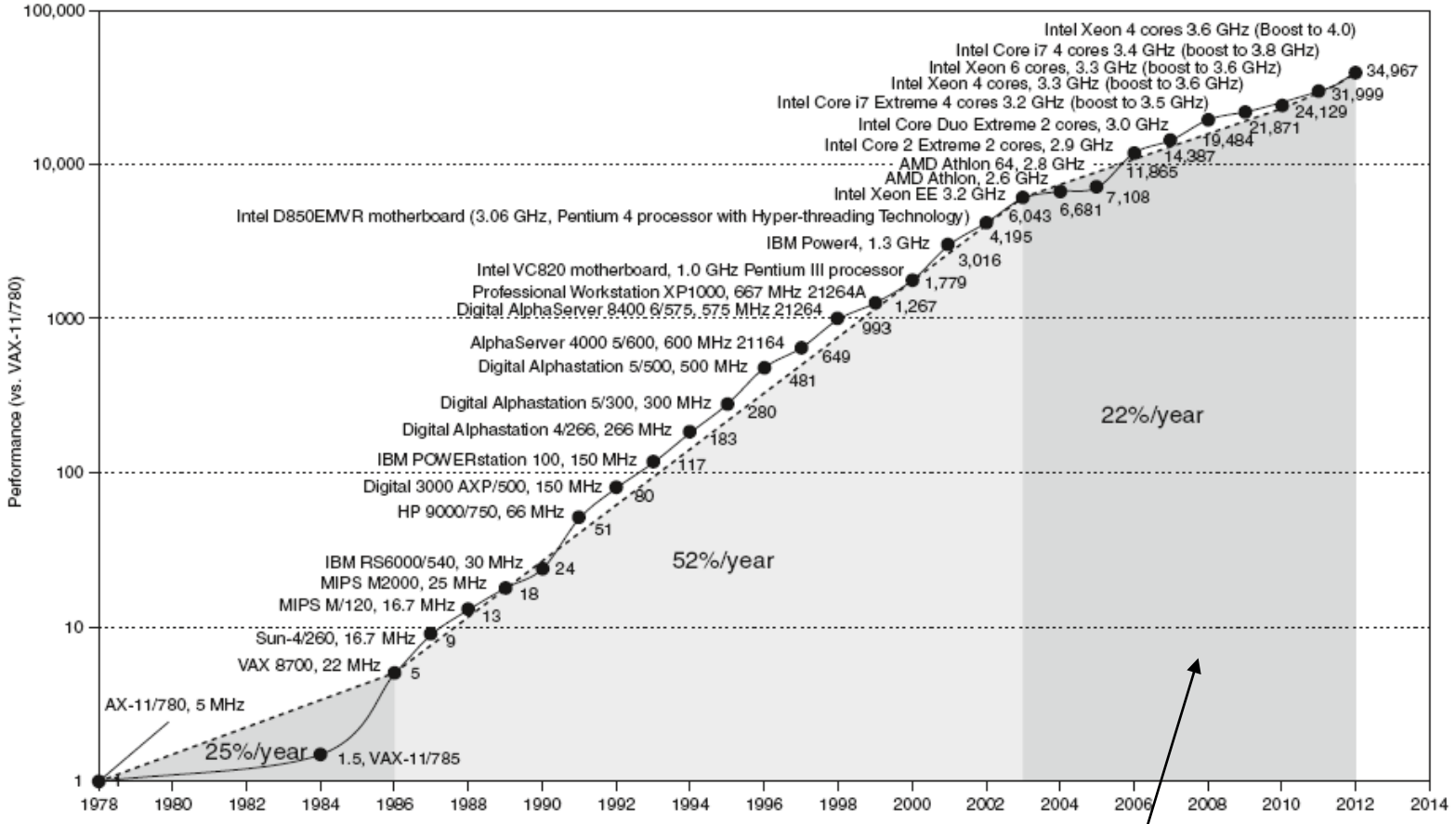
SPECpower_ssj2008 for Xeon X5650

- **Power consumption of server at different workload levels**
  - Performance: ssj_ops/sec (server side Java operations per second)
  - Power: Watts (Joules/sec)

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
Σssj_ops/Σpower =		2,490

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

Multiprocessors

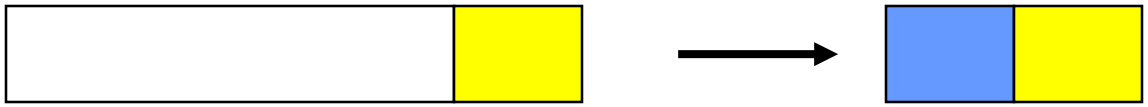
- **Multicore microprocessors**
  - More than one processor per chip
- **Requires explicitly parallel programming**
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

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 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?"

Make the common case fast

Example

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

- If the execution time of some benchmark **before the floating-point enhancement is 10 seconds**,

  $ExecTime_{old}$ **=**

- what will the speedup be if **half of the 10 seconds is spent executing floating-point instructions**?

  $ExecTime_{new}$ =

  SpeedUp =

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 (without adverse CPI affects)
  - improvements in processor organization that lower CPI
  - compiler enhancements that lower CPI and/or instruction count
- **Pitfall:**
  - expecting improvement in one aspect of a machine performance to affect the total performance