

---

# **C335**

## **Computer Structures**

### **Computer Abstractions and Technology (Part #3)**

Dr. Liqiang Zhang

Department of Computer and Information Sciences

# Computer Abstractions and Technology

- ❑ What is computer architecture?
- ❑ What forces drive computer architecture?
- ❑ Performance

# Performance Metrics

---



## ❑ Purchasing perspective

- given a collection of machines, which has the
  - best performance ?
  - least cost ?
  - best cost/performance?

## ❑ Design perspective

- faced with design options, which has the
  - best performance improvement ?
  - least cost ?
  - best cost/performance?

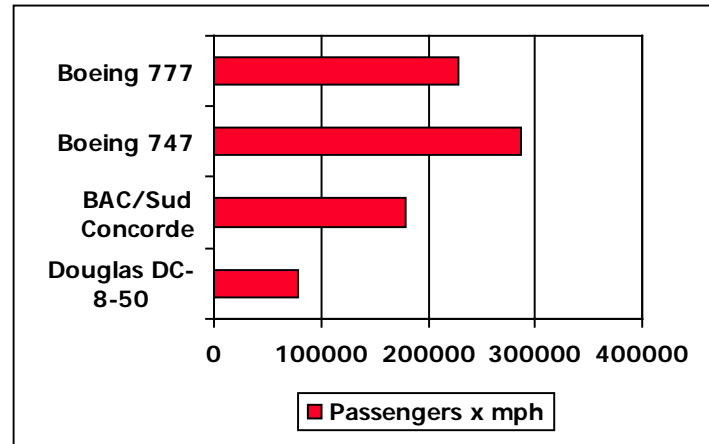
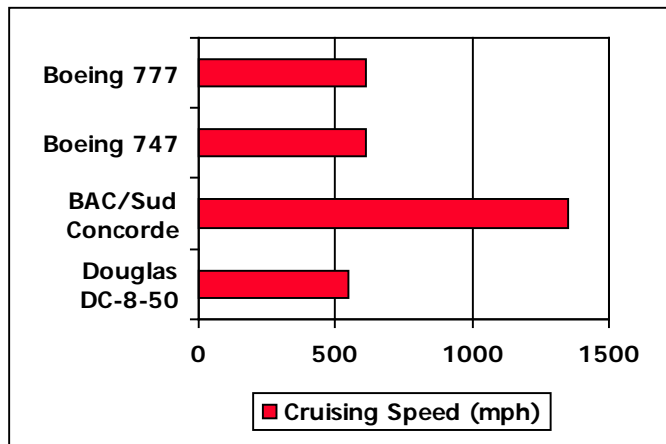
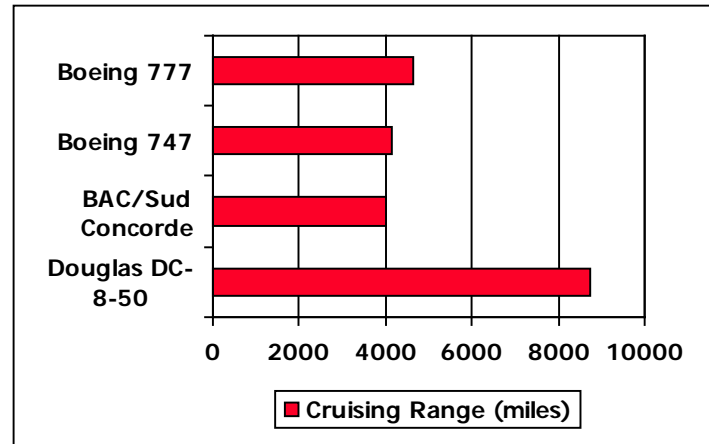
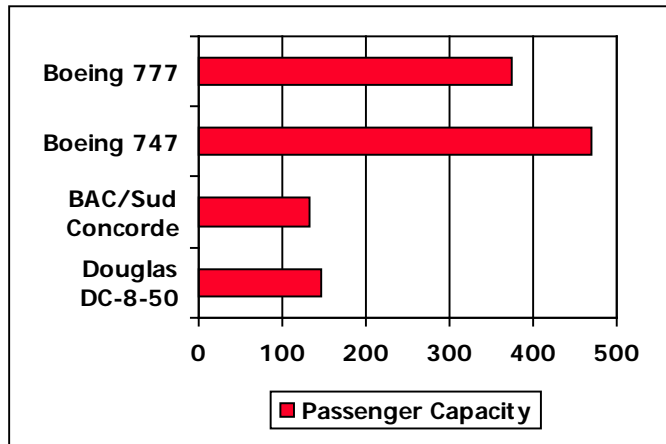
## ❑ Both require

- basis for comparison
- metrics for evaluation

## ❑ Our goal is to understand what factors in the architecture contribute to overall system performance and the relative importance (and cost) of these factors

# Defining Performance

❑ Which airplane has the best performance?



# Response Time and Throughput

---

## ❑ Response time

- How long it takes to do a task

## ❑ Throughput

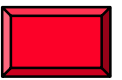
- Total work done per unit time
  - e.g., tasks/transactions/... per hour

## ❑ 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...

# Relative Performance



- ❑ Define Performance = 1/Execution Time
- ❑ “X is  $n$  time faster than Y”

$$\begin{aligned} & \text{Performance}_X / \text{Performance}_Y \\ &= \text{Execution time}_Y / \text{Execution time}_X = n \end{aligned}$$

- Example: time taken to run a program
  - 10s on A, 15s on B
  - $\text{Execution Time}_B / \text{Execution Time}_A$   
 $= 15\text{s} / 10\text{s} = 1.5$
  - So A is 1.5 times faster than B

# Measuring Execution Time

---

## ❑ Elapsed time

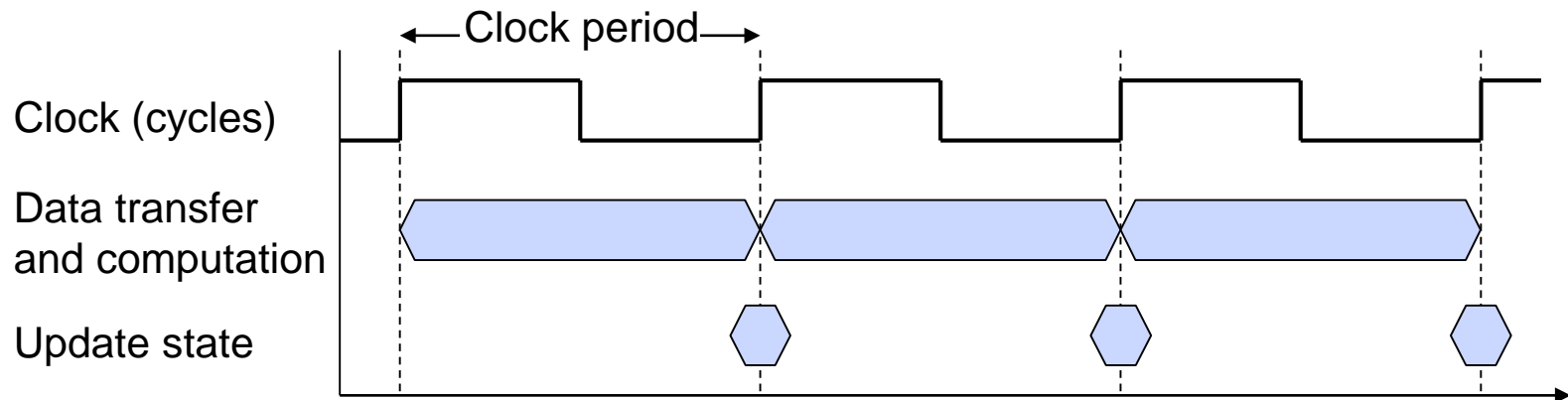
- Total response time, including all aspects
  - Processing, I/O, OS overhead, idle time
- Determines system performance

## ❑ CPU time

- Time spent processing a given job
  - Discounts I/O time, other jobs' shares
- Comprises user CPU time and system CPU time
- Different programs are affected differently by CPU and system performance

# CPU Clocking

- ❑ Operation of digital hardware governed by a constant-rate clock



- Clock period: duration of a clock cycle
  - e.g.,  $250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$
- Clock frequency (rate): cycles per second
  - e.g.,  $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$



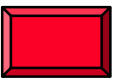
# CPU Time

---

$$\begin{aligned}\text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}\end{aligned}$$

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

# CPU Time Example



- ❑ Computer A: 2GHz clock, 10s CPU time
- ❑ Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- ❑ How fast must Computer B clock be?

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

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

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

# Instruction Count and CPI

$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$

$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$   
 $= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$

- ❑ Instruction Count for a program
  - Determined by program, ISA and compiler
- ❑ Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

# CPI Example

- ❑ Computer A: Cycle Time = 250ps, CPI = 2.0
- ❑ Computer B: Cycle Time = 500ps, CPI = 1.2
- ❑ Same ISA
- ❑ Which is faster, and by how much?

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps}\end{aligned}$$

A is faster...

$$\begin{aligned}\text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps}\end{aligned}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2$$

...by this much

# 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

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

Relative frequency

# 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: IC = 5

- Clock Cycles  
 $= 2 \times 1 + 1 \times 2 + 2 \times 3$   
 $= 10$
- Avg. CPI =  $10/5 = 2.0$

- Sequence 2: IC = 6

- Clock Cycles  
 $= 4 \times 1 + 1 \times 2 + 1 \times 3$   
 $= 9$
- Avg. CPI =  $9/6 = 1.5$

# Performance Summary

## The BIG Picture

$$\begin{aligned}\text{CPU Time} &= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} \\ &= \text{IC} \times \text{CPI} \times T_c\end{aligned}$$

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



# Example

Op	Freq	CPI <sub>i</sub>	Freq x CPI <sub>i</sub>
ALU	50%	1	.5
Load	20%	5	1.0
Store	10%	3	.3
Branch	20%	2	.4
			$\Sigma = 2.2$

.5   .5   .25  
.4   1.0   1.0  
.3   .3   .3  
.4   .2   .4  
1.6   2.0   1.95

- ❑ How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

CPU time new =  $1.6 \times IC \times T_c$  so  $2.2/1.6$  means 37.5% faster

- ❑ How does this compare with using branch prediction to shave a cycle off the branch time?

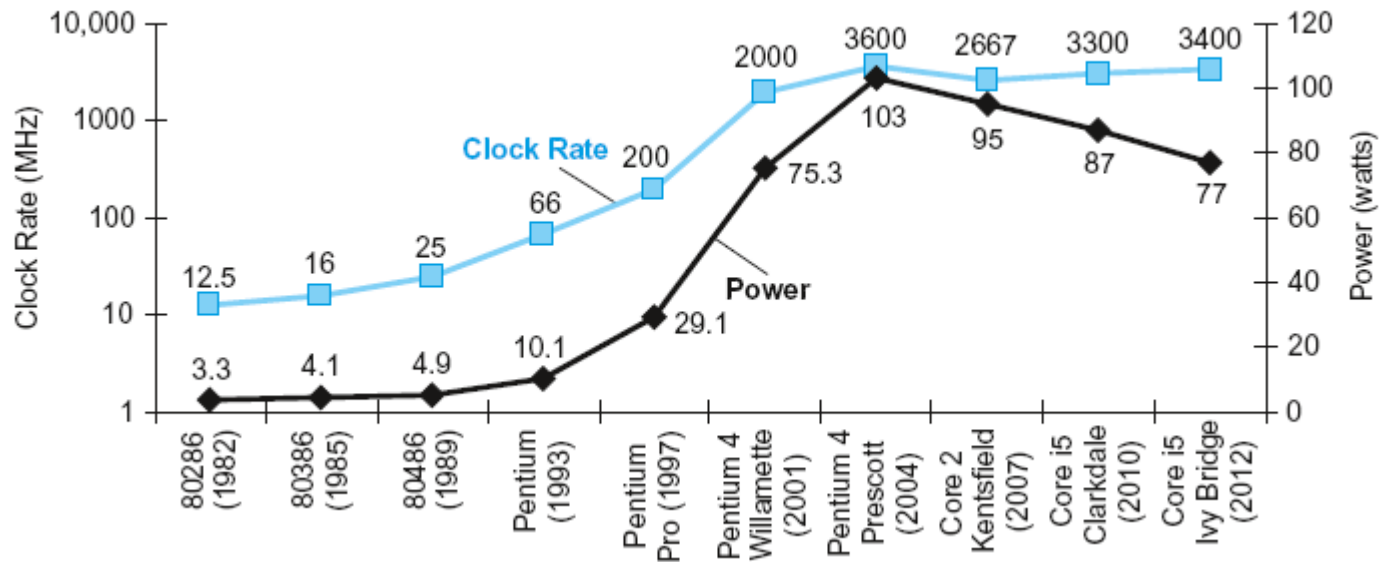
CPU time new =  $2.0 \times IC \times T_c$  so  $2.2/2.0$  means 10% faster

- ❑ What if two ALU instructions could be executed at once?

CPU time new =  $1.95 \times IC \times T_c$  so  $2.2/1.95$  means 12.8% faster



# Power Trends



❑ In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30

5V → 1V

×1000

# Reducing Power

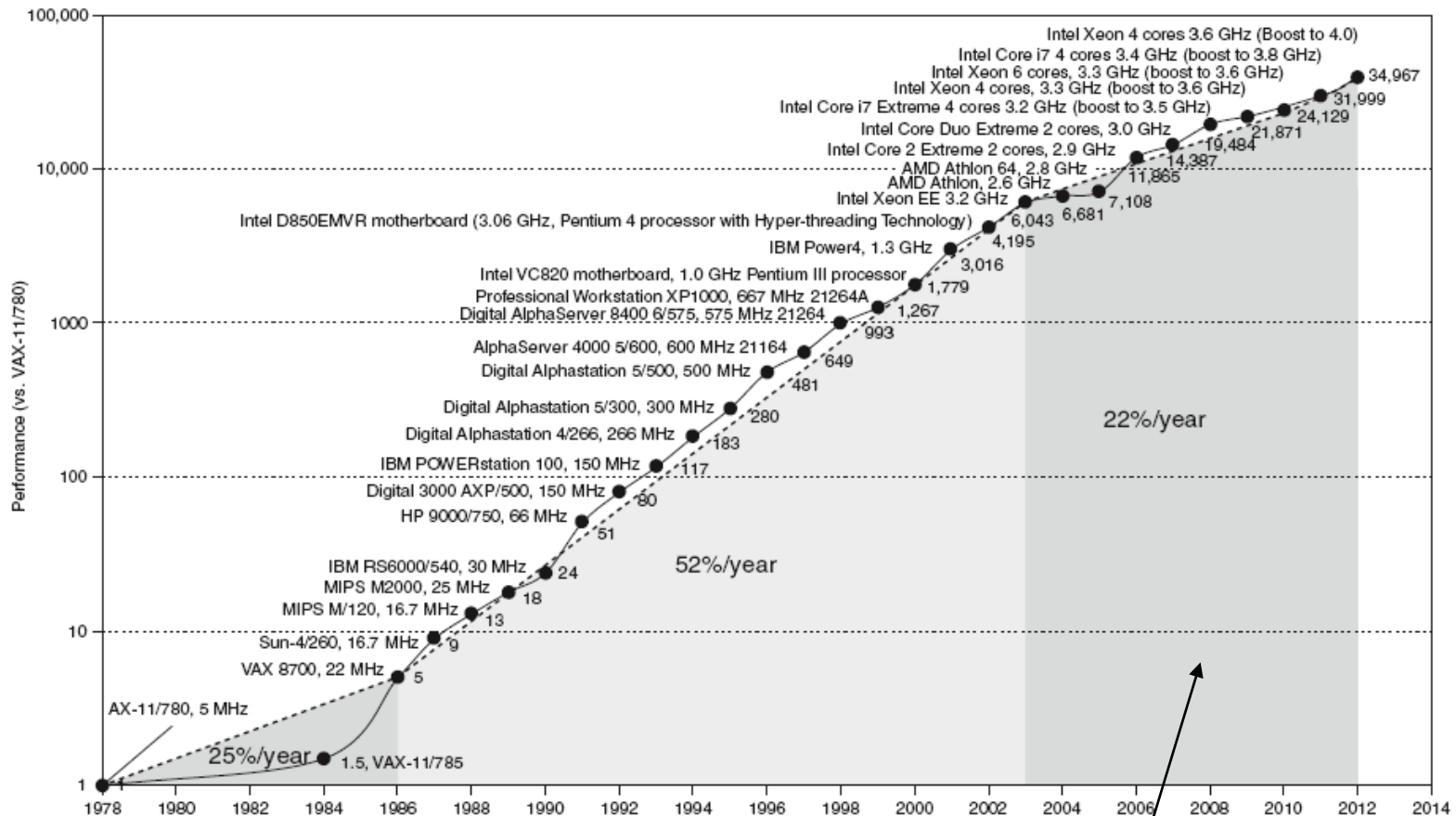
---

- ❑ Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{\text{new}}}{P_{\text{old}}} = \frac{C_{\text{old}} \times 0.85 \times (V_{\text{old}} \times 0.85)^2 \times F_{\text{old}} \times 0.85}{C_{\text{old}} \times V_{\text{old}}^2 \times F_{\text{old}}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

# 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

# SPEC CPU Benchmark

---

- ❑ Programs used to measure performance
  - Supposedly typical of actual workload
- ❑ Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, ...
- ❑ SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

$$\sqrt[n]{\prod_{i=1}^n \text{Execution time ratio}_i}$$

# CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10 <sup>9</sup>	CPI	Clock cycle time (seconds x 10 <sup>-9</sup> )	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

# SPEC Power Benchmark

---

- ❑ Power consumption of server at different workload levels
  - Performance: ssj\_ops/sec
  - Power: Watts (Joules/sec)

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

# SPECpower\_ssj2008 for Xeon X5650

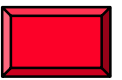
---

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
$\Sigma \text{ssj\_ops} / \Sigma \text{power} =$		2,490



# Fallacies and Pitfalls

---



- ❑ **Fallacies**: commonly held misconceptions
- ❑ **Pitfalls**: generalizations of principles that are true in a limited context.

- ❑ *A pitfall that traps many designers and reveals an important relationship in computer design:*

*“Expecting the improvement of one aspect of a computer to increase overall performance by an amount proportional to the size of the improvement.”*

# Pitfall: Amdahl's Law

---

- Suppose a program runs in 100 seconds on a computer, with multiply operations responsible for 80 seconds of this time. How much do I have to improve the speed of multiplication if I want my program to run five times faster?

- **Amdahl's law:**

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

- For the problem:

$$20 = \frac{80}{n} + 20$$

■ Can't be done!

# What we can do with Amdahl's law?

---

- ❑ Can use Amdahl's law to estimate performance improvements when we know the time consumed for some function and its potential speedup.
  
- ❑ A corollary of Amdahl's law → a common theme in hardware design
  - *Make the common case fast*

# Fallacy: Low Power at Idle

---



- ❑ Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- ❑ Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- ❑ Consider designing processors to make power proportional to load

# Pitfall: MIPS as a Performance Metric



- ❑ MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\begin{aligned} \text{MIPS} &= \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} \\ &= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}} \times 10^6 = \frac{\text{Clock rate}}{\text{CPI} \times 10^6} \end{aligned}$$

- CPI varies between programs on a given CPU

# Eight Great Ideas in Computer Architecture

- ❑ Design for **Moore's Law**
- ❑ Use **abstraction** to simplify design
- ❑ Make the **common case fast**
- ❑ Performance via **parallelism**
- ❑ Performance via **pipelining**
- ❑ Performance via **prediction**
- ❑ **Hierarchy** of memories
- ❑ **Dependability** via redundancy



# Concluding Remarks

---



- ❑ Performance metrics
  - Response time and throughput
- ❑ CPU time
  - Clock cycles  $\times$  cycle time
  - $IC \times CPI \times T_c$
- ❑ Power is a limiting factor
  - Use parallelism to improve performance
- ❑ SPEC benchmark
  - For CPU and for Power
- ❑ Fallacies and Pitfalls
  - Amdahl's Law
  - Low power at idle
  - MIPS as performance metric
- ❑ Eight great ideas in computer architecture