Chapter 1

Computer Abstractions and Technology

The Computer Revolution

§1.1 Introduction

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

Classes of Computers

- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff

- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized

Classes of Computers

- Supercomputers
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market

- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

The PostPC Era
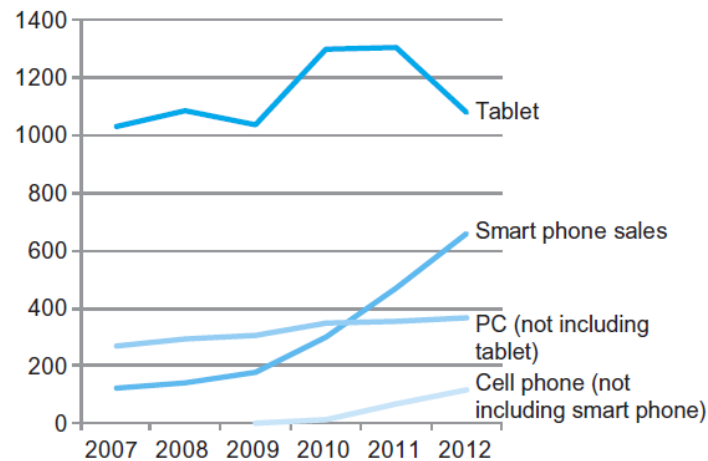
The PostPC Era

- Personal Mobile Device (PMD)
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses
- Cloud computing
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google

What You Will Learn

- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance
- What is parallel processing

Understanding Performance

- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

Eight Great Ideas

- 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

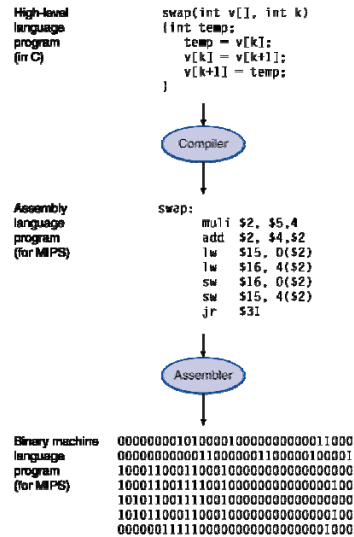- *Dependability via* redundancy

Below Your Program
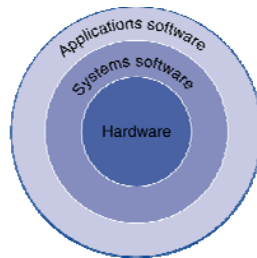
- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers
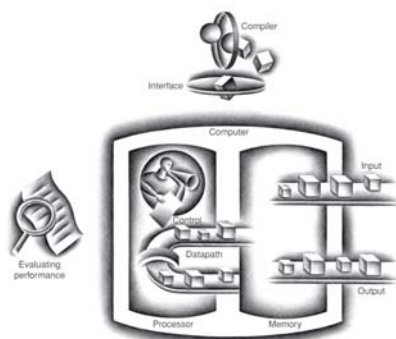
Applications software

Systems software

Hardware

Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Compiler

Assembly language program (for MIPS)

```
swap:
        muli $2, $5,4
        add  $2, $4,$2
        lw   $15, 0($2)
        lw   $16, 4($2)
        sw   $16, 0($2)
        sw   $15, 4($2)
        jr   $31
```

Assembler

Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000000110000110000100001001
10001100011000100000000000000000
10001100111100010000000000000100
10101100111100010000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Components of a Computer

The BIG Picture

- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
   - For communicating with other computers

§1.4 Under the Covers

Touchscreen

- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously

Through the Looking Glass

- LCD screen: picture elements (pixels)
  - Mirrors content of frame buffer memory

Opening the Box

Capacitive multitouch LCD screen

3.8 V, 25 Watt-hour battery

Computer board
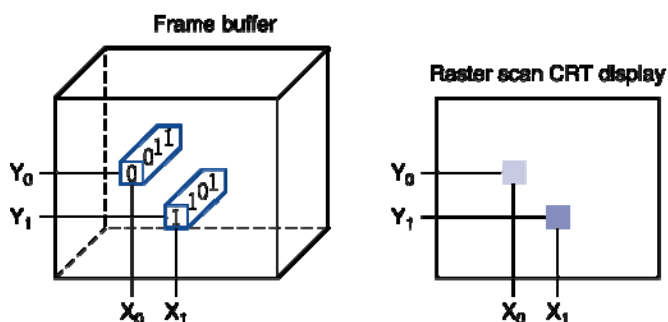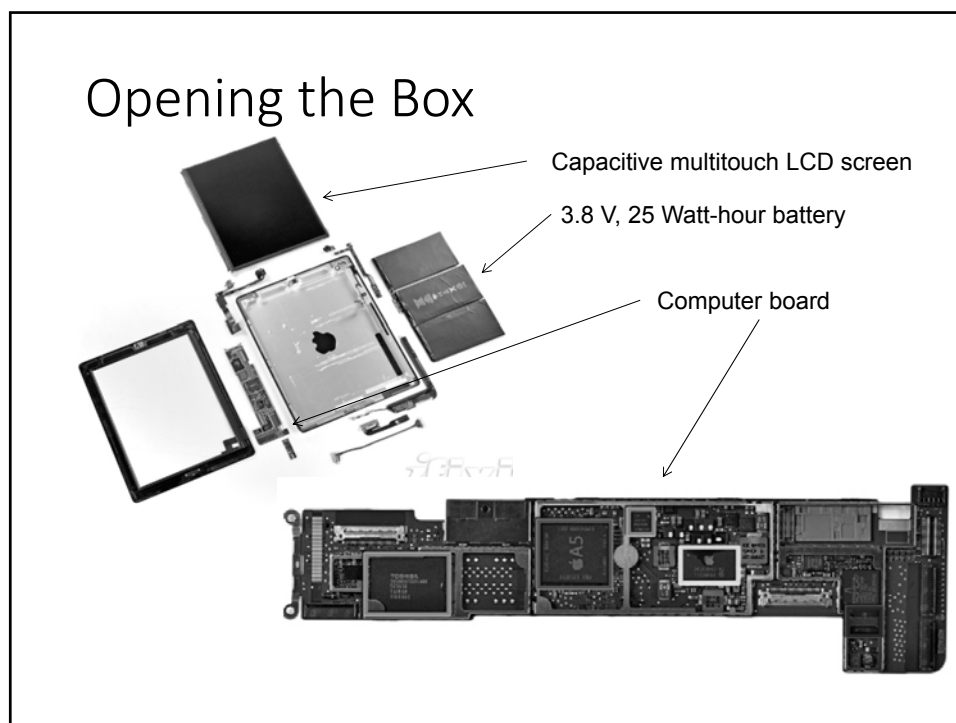
Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, …
- Cache memory
  - Small fast SRAM memory for immediate access to data

Inside the Processor

- Apple A5

Abstractions

The BIG Picture

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - The hardware/software interface
- Application binary interface
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface

A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
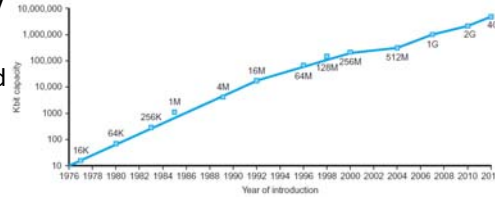 - Flash memory
  - Optical disk (CDROM, DVD)

Networks

- Communication, resource sharing, nonlocal access
- Local area network (LAN): Ethernet
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth

Technology Trends

- Electronics technology continues to evolve
 - Increased capacity and performance
  - Reduced cost

DRAM capacity

| Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Semiconductor Technology

- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators
  - Switch

Manufacturing ICs
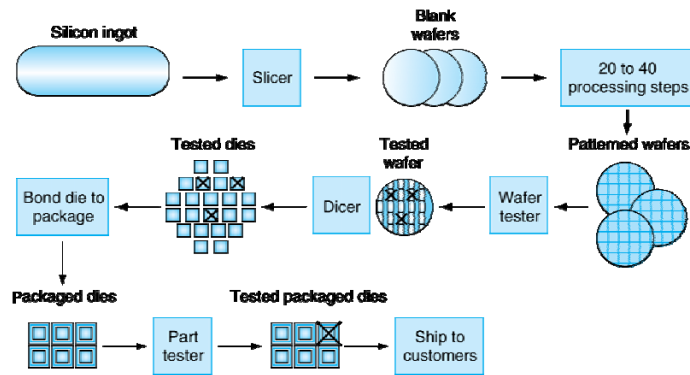
- Yield: proportion of working dies per wafer

Intel Core i7 Wafer
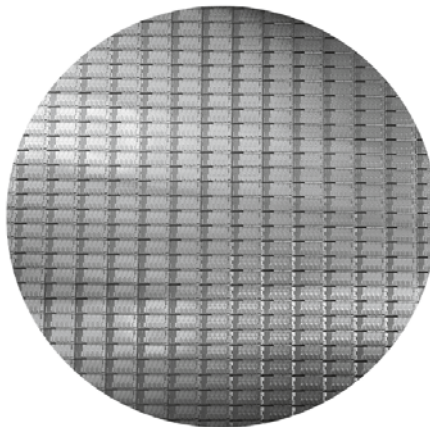
- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm

Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area/Die area}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area/2}))^2}$$

- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

Performance

- Measure, Report, and Summarize
- Make intelligent choices
- See through the marketing hype
- Key to understanding underlying organizational motivation
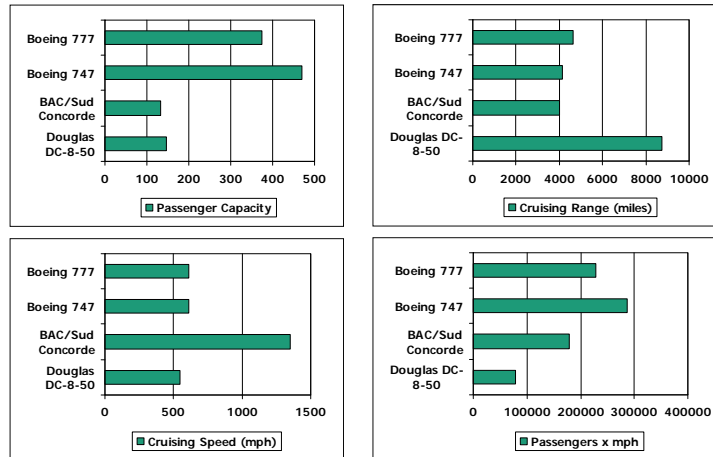
Why is some hardware better than others for different programs?

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?

Defining Performance

• Which airplane has the best performance?

§1.6 Performance

Computer Performance: TIME, TIME, TIME

• Response Time (latency)
  — 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
  — 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?

• *If we upgrade a machine with a new processor what do we increase?*

• *If we add a new machine to the lab what do we increase?*

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"

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time$_B$ / Execution Time$_A$ = 15s / 10s = 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
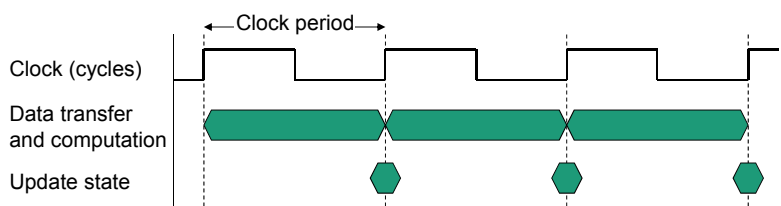
- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

CPU Time

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

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

Clock Cycles

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

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

- Clock "ticks" indicate when to start activities (one abstraction):
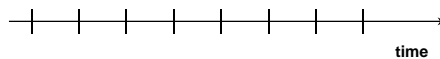
$\vdash\!\!+\!\!+\!\!+\!\!+\!\!+\!\!+\!\!+\!\!+\!\!\rightarrow$ **time**

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

A 4 GHz clock has a $\frac{1}{4 \times 10^9} \times 10^{12} = 250 \text{ picoseconds (ps)}$ cycle time

How to Improve Performance

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

So, to improve performance (everything else being equal) you can either (increase or decrease?)

_____ the # of required cycles for a program, or

_____ the clock cycle time or, said another way,

_____ the clock rate.

How many cycles are required for a program?

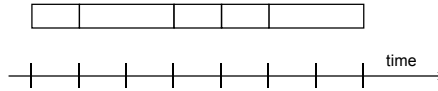- Could assume that number of cycles equals number of instructions
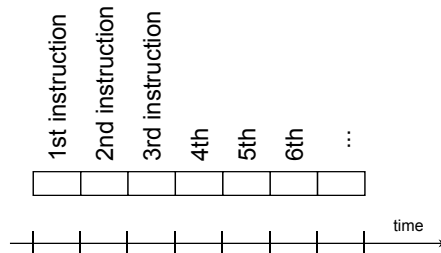
This assumption is incorrect,

   different instructions take different amounts of time on different machines.

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

Different numbers of cycles for different instructions

time

- 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)*

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?

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

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10s \times 2\text{GHz} = 20 \times 10^9$$

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

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

Example

- Our favorite program runs in 10 seconds on computer A, which has a 4 GHz. 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?"

- Don't Panic, can easily work this out from basic principles

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?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$

$$= I \times 2.0 \times 250ps = I \times 500ps \quad \leftarrow \boxed{\text{A is faster...}}$$

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$

$$= I \times 1.2 \times 500ps = I \times 600ps$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600ps}{I \times 500ps} = 1.2 \quad \leftarrow \boxed{\text{...by this much}}$$

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

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

$$\underbrace{\qquad\qquad}_{\boxed{\text{Relative frequency}}}$$

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

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

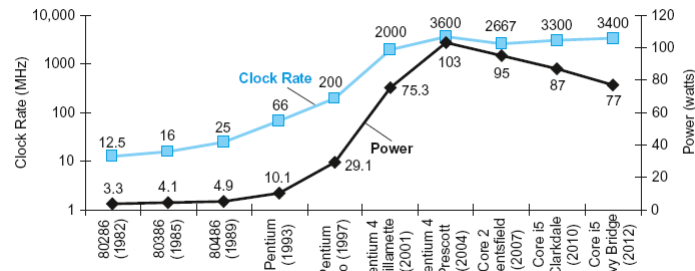## Performance Summary

The BIG Picture

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

- 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

Power Trends

• In CMOS IC technology

$$Power = Capacitive\ load \times Voltage^2 \times Frequency$$

×30   5V → 1V   ×1000

Reducing Power

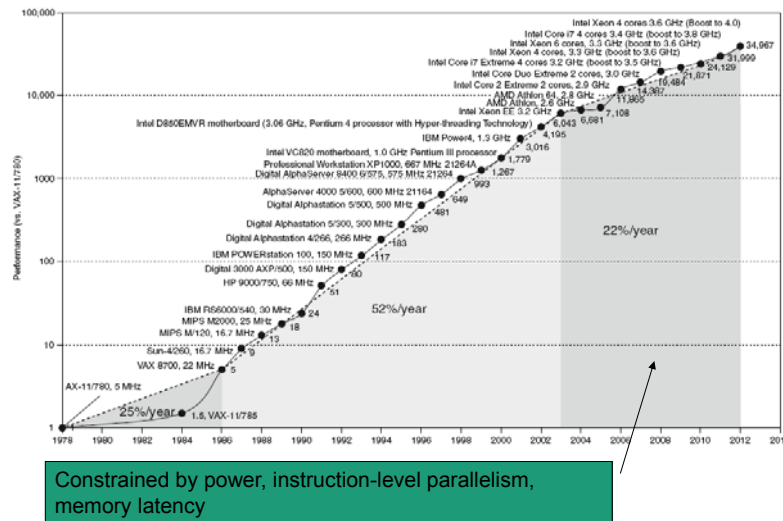• Suppose a new CPU has
  • 85% of capacitive load of old CPU
  • 15% voltage and 15% frequency reduction

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

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

Uniprocessor Performance

§1.8 The Sea Change: The Switch to Multiprocessors

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 10^9	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

SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

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

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
Σssj_ops/Σpower =		2,490

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

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

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

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

  - Can't be done!

- Corollary: make the common case fast

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?"

  How about making it 5 times faster?

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

- 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. One benchmark we are considering runs for 100 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?

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

$$MIPS = \frac{Instruction\ count}{Execution\ time \times 10^6}$$

$$= \frac{Instruction\ count}{\frac{Instruction\ count \times CPI}{Clock\ rate} \times 10^6} = \frac{Clock\ rate}{CPI \times 10^6}$$

- CPI varies between programs on a given CPU

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 1 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?

Concluding Remarks

- Cost/performance is improving
    - Due to underlying technology development
- Hierarchical layers of abstraction
    - In both hardware and software
- Instruction set architecture
    - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
    - Use parallelism to improve performance

§1.9 Concluding Remarks