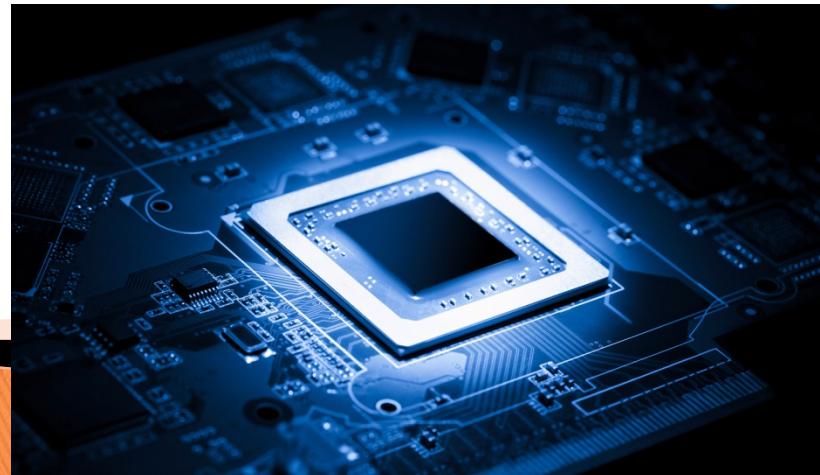


CSE 331

Computer Organization

Lecture 1

Computer Abstractions and Technology



Introduction

- ▶ This course is all about how computers work
- ▶ But what do we mean by a computer?
 - Different types: desktop, servers, embedded devices
 - Different uses: automobiles, graphics, finance, genomics...
 - Different manufacturers: Intel, IBM, AMD, Sun...
 - Different underlying technologies and different costs!

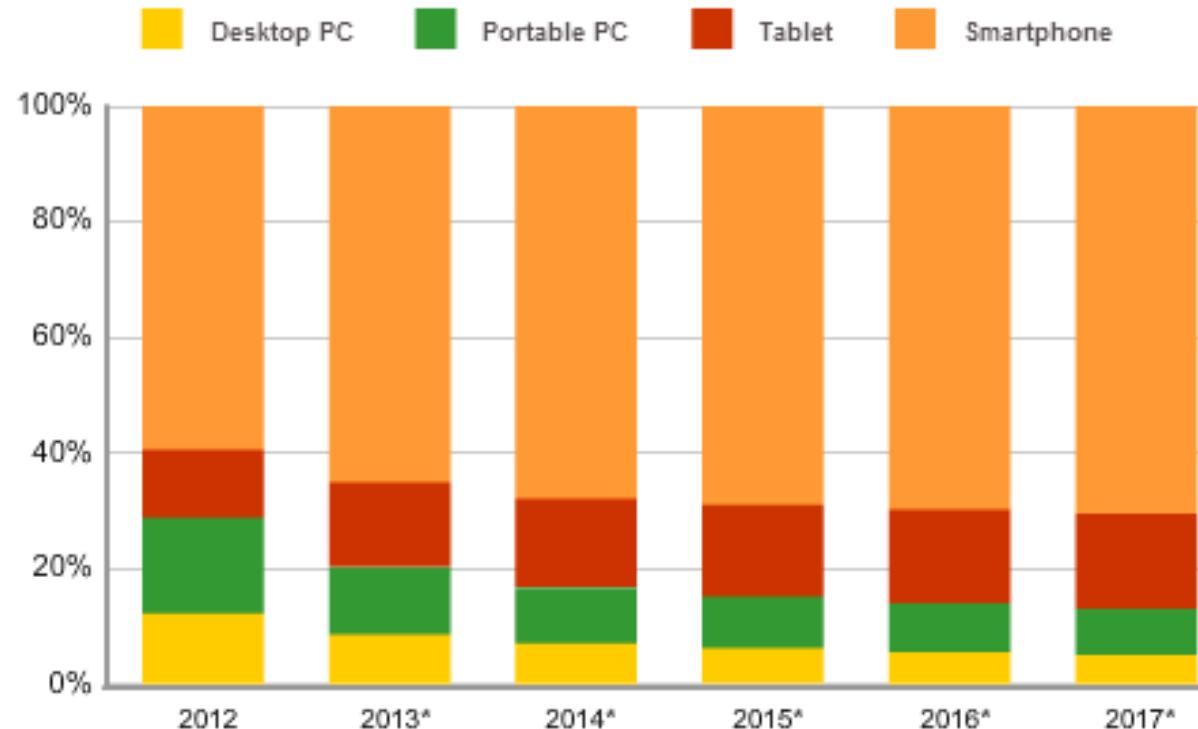
Classes of Computers

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

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

Worldwide Smart Connected Device Forecast* Market Share by Product Category, 2012-2017



POST-PC Era: Personal Mobile Device (PMD)

- Battery operated
- Connects to the Internet
- Hundreds of dollars
- Smart phones, tablets, electronic glasses

The Computer Revolution

Computer



2013
Samsung



1973
Motorola



Electrical

1896
İsveç

Computer



**2016
Raikkonen**



1989

Mechanical



1946

IoT



The Computer Revolution

- ▶ 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

Fun Facts



The original transistor built by Bell Labs in 1947 was large enough that it was pieced together by hand. By contrast, more than 100 million 22nm tri-gate transistors could fit onto the head of a pin.¹



More than 6 million 22nm tri-gate transistors could fit in the period at the end of this sentence.²



A 22nm tri-gate transistor's gates are so small you could fit more than 4,000 of them across the width of a human hair.³



If a typical house shrunk as transistors have, you would not be able to see a house without a microscope. To see a 22nm feature with the naked eye, you would have to enlarge a chip to be larger than a house.⁴



Compared to Intel's first microprocessor, the 4004, introduced in 1971, a 22nm CPU runs over 4,000 times as fast and each transistor uses about 5,000 times less energy. The price per transistor has dropped by a factor of about 50,000.



A 22nm transistor can switch on and off well over 100 billion times in one second. It would take you around 2,000 years to flick a light switch on and off that many times.⁵

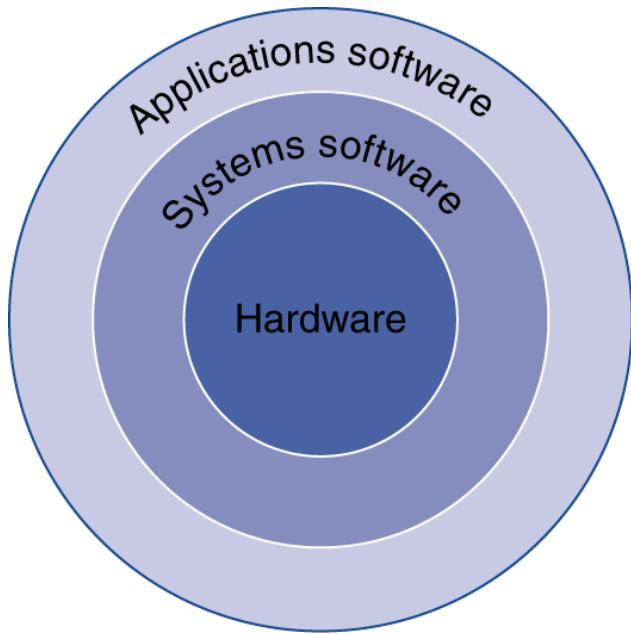


It's one thing to design a tri-gate transistor but quite another to get it into high volume manufacturing. Intel's factories produce over 5 billion transistors every second. That's 150,000,000,000,000 transistors per year, the equivalent of over 20 million transistors for every man, woman and child on earth.



The 3rd Generation Intel® Core™ processor — quad core, contains 1.48 billion transistors. If transistors were people, Intel's chip has more transistors than the population of China at approximately 1.3 billion people.

Below Your Program



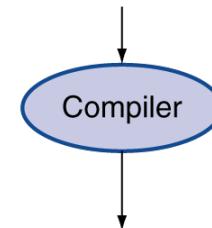
- ▶ **Application software**
 - Written in high-level language
- ▶ **System software**
 - Compiler: translates HLL code to machine code
 - Operating System: service code
 - Handling input/output
 - Managing memory and storage
 - Scheduling tasks & sharing resources
- ▶ **Hardware**
 - Processor, memory,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

Abstraction

- ▶ 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;
}
```

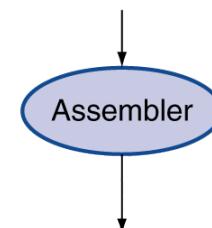


Assembly language program (for MIPS)

```

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

```



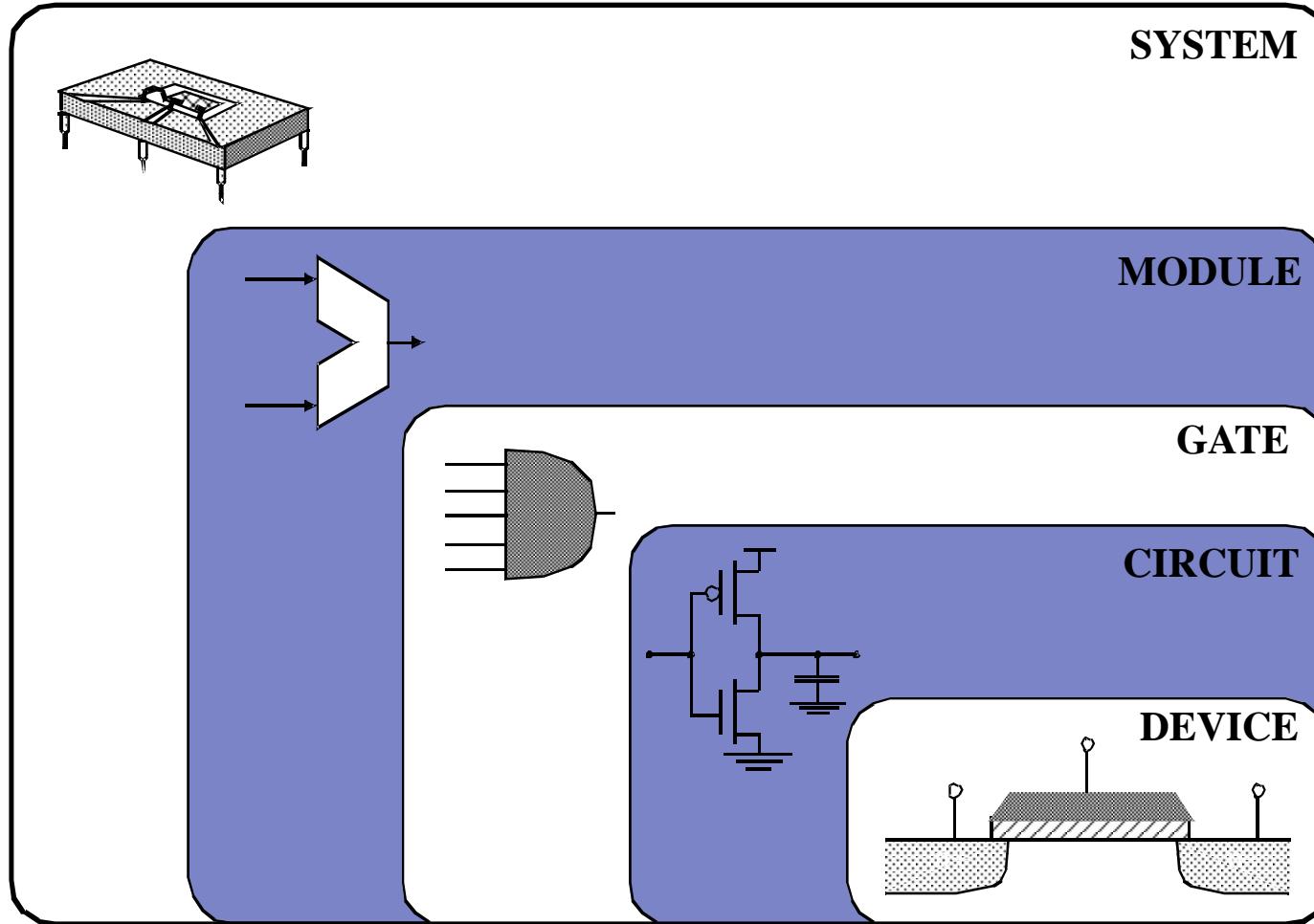
Binary machine
language
program
(for MIPS)

```
000000001010000100000000000011000  
00000000000110000001100000100001  
1000110001100010000000000000000000  
1000110011110010000000000000000000100  
10101100111100100000000000000000000000  
101011000110001000000000000000000000000100  
000000011111000000000000000000000000000000001000
```

Instruction Set Architecture

- ▶ A very important abstraction
 - interface between hardware and low-level software
 - standardizes instructions, machine language bit patterns, etc.
 - advantage: *different implementations of the same architecture*
 - disadvantage: *sometimes prevents using new innovations*
- ▶ Modern instruction set architectures:
 - IA-32, PowerPC, MIPS, SPARC, ARM, and others

Design Abstraction Levels

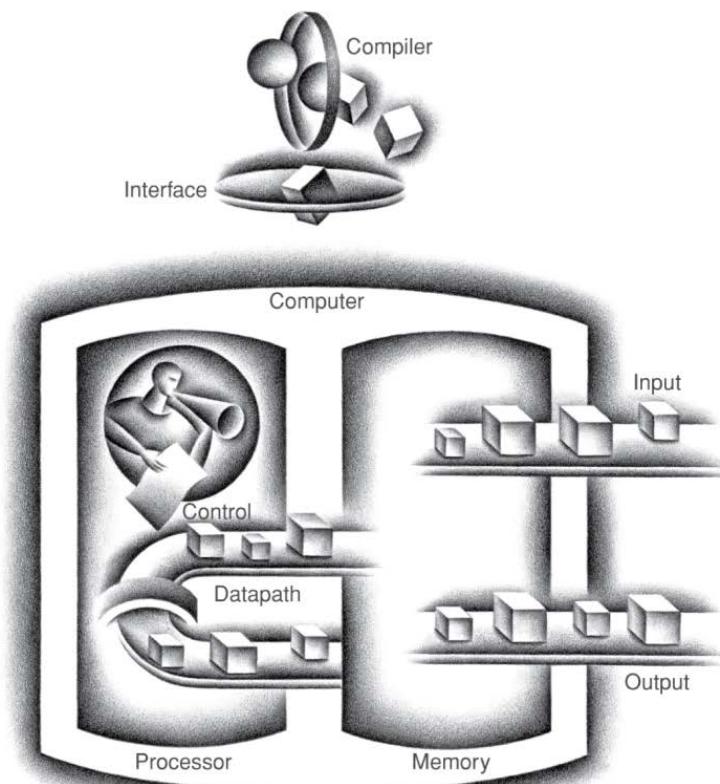


Components of a Computer

The BIG Picture

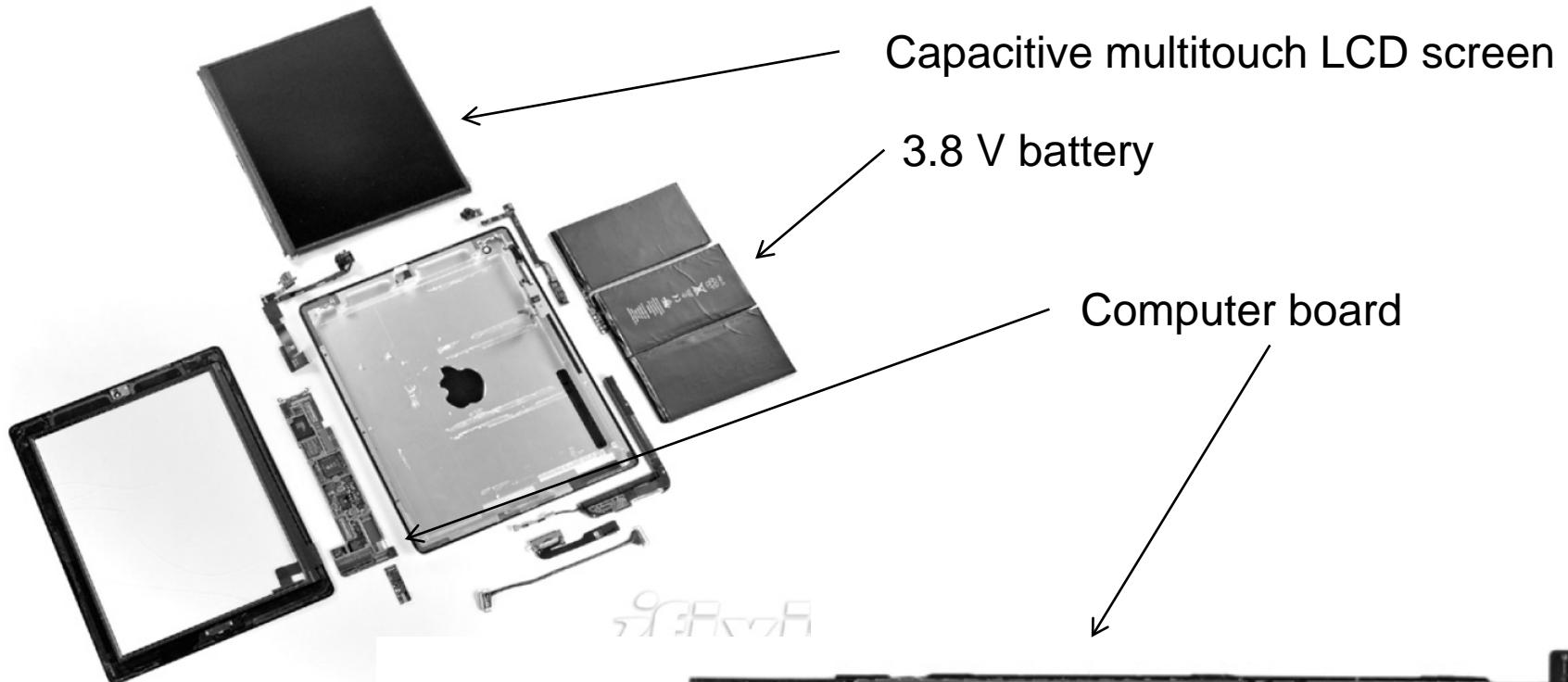


Evaluating performance

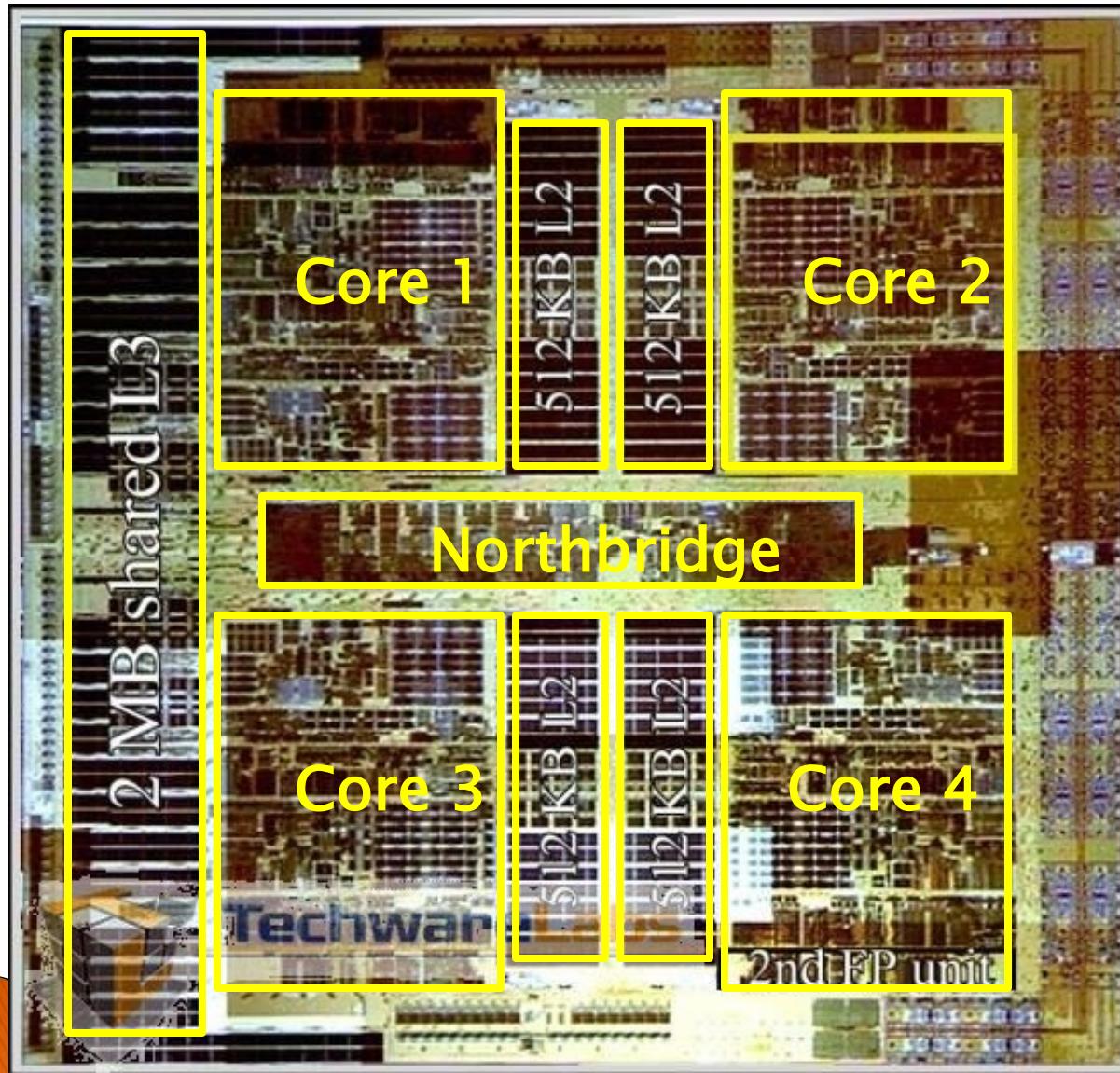


- ▶ Same components for all kinds of computer
 - Desktop, server, embedded
- ▶ Input/output includes
 - User-interface devices
 - Display, keyboard, mouse
 - Storage devices
 - Hard disk, CD/DVD, flash
 - Network adapters
 - For communicating with other computers

Opening the Box



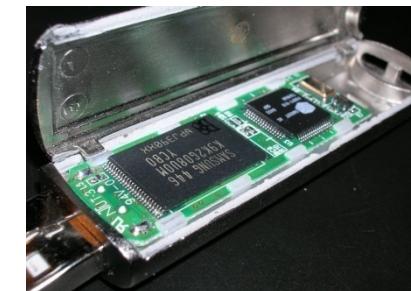
Inside the Processor: AMD's Barcelona



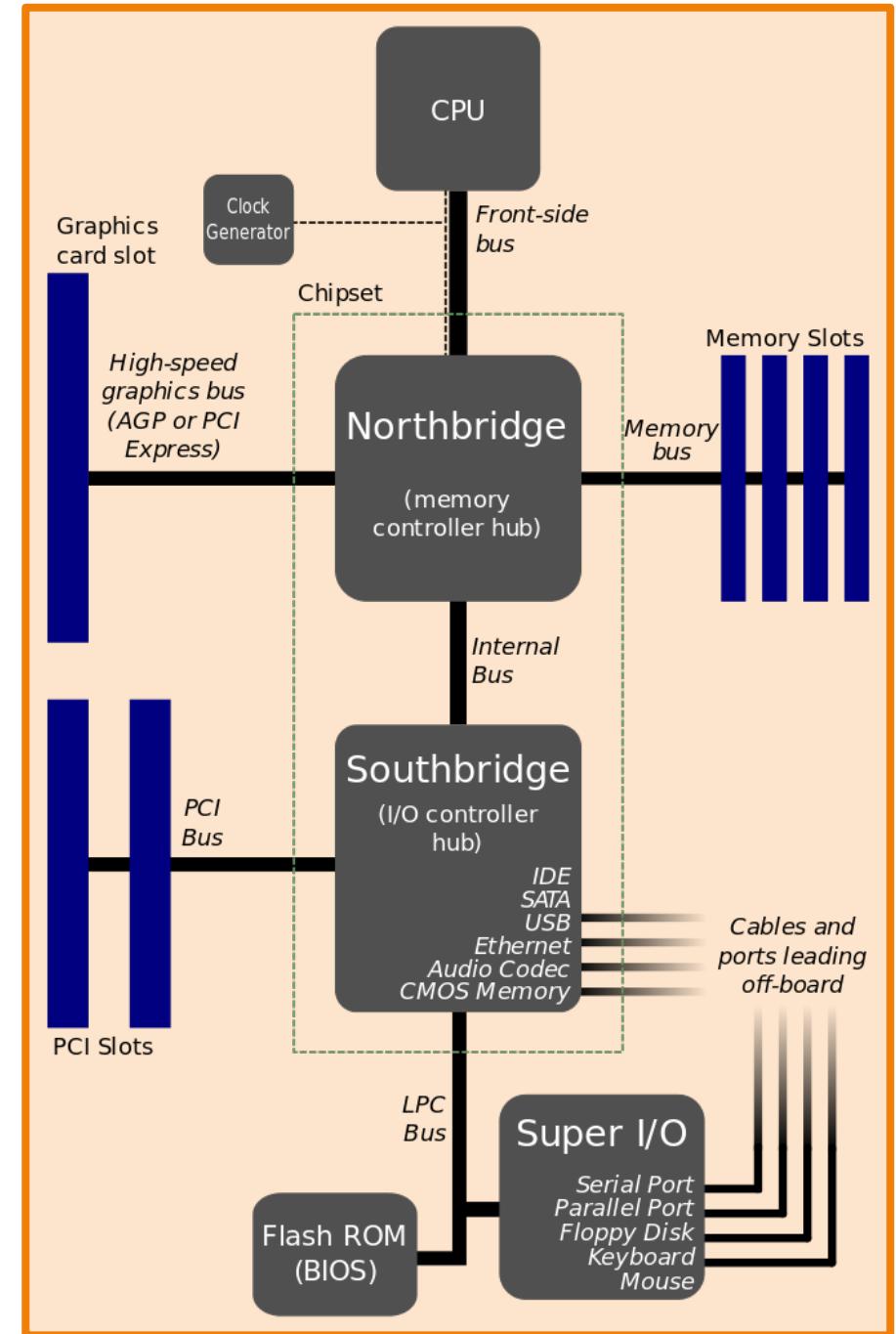
- ❑ Four out-of-order cores on one chip
- ❑ 1.9 GHz clock rate
- ❑ 65nm technology
- ❑ Three levels of caches (L1, L2, L3) on chip
- ❑ Integrated Northbridge

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)



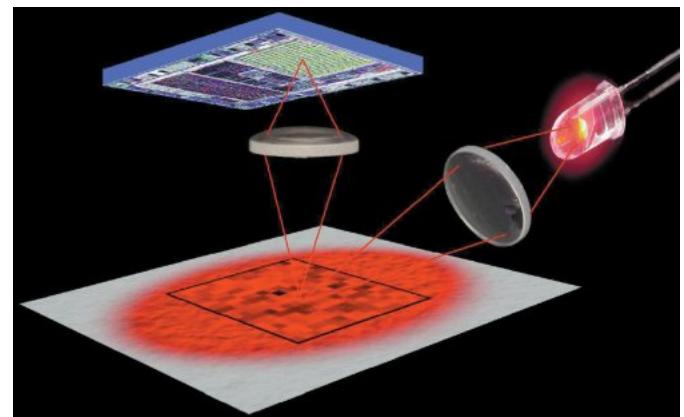
NorthBridge & Southbridge



<http://www.hardwaresecrets.com/everything-you-need-to-know-about-the-pci-express/2/>

Anatomy of a Mouse

- ▶ Optical mouse
 - LED illuminates desktop
 - Small low-res camera
 - Basic image processor
 - Looks for x, y movement
 - Buttons & wheel
- ▶ Supersedes roller-ball mechanical mouse

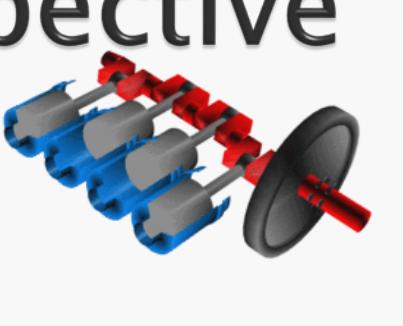


<https://www.youtube.com/watch?v=SXQfT7c-9rU>

Historical Perspective



Elephant Clock



https://www.youtube.com/watch?v=BCc-A_Oy6SY

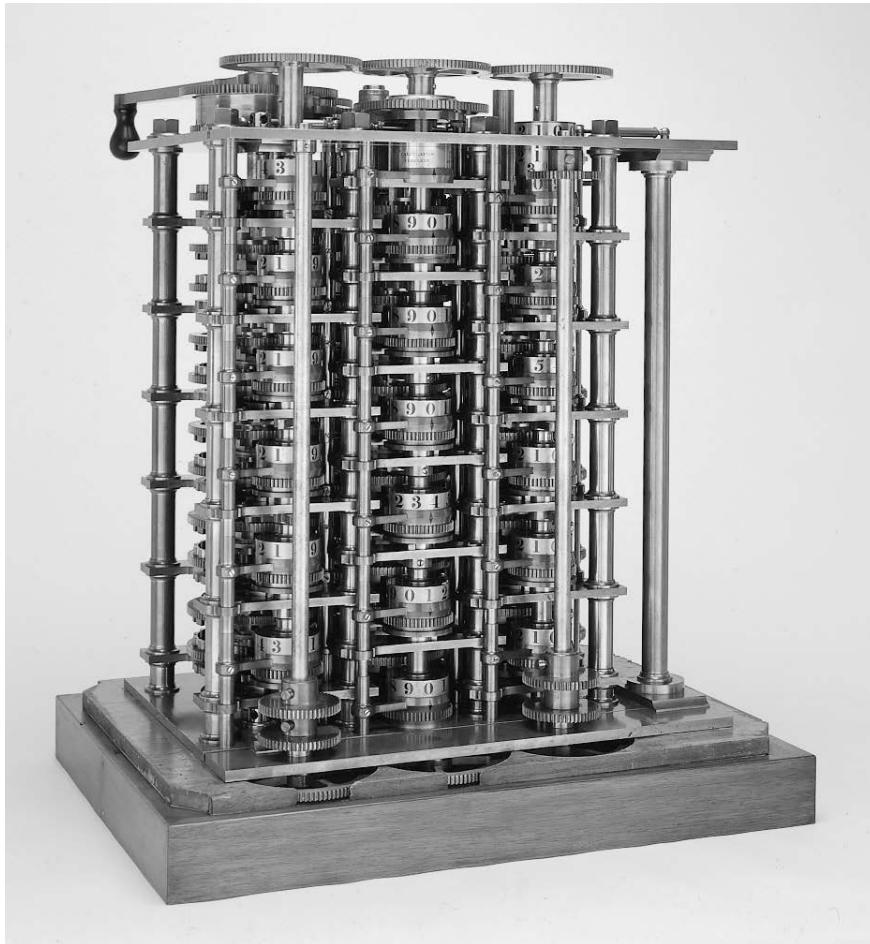
<https://www.youtube.com/watch?v=HLvYTJVOWi4>

Al-Jazari

The "castle clock", an astronomical clock invented by Al-Jazari in 1206, is considered to be the earliest programmable analog computer.

- The zodiac, the solar and lunar orbits
- Crescent moon-shaped pointer travelling across a gateway causing automatic doors to open every hour.
- Five robotic musicians who play music when struck by levers operated by a camshaft attached to a water wheel.
- The length of day and night could be re-programmed every day in order to account for the changing lengths of day and night throughout the year.

Historical Perspective



The First Computer

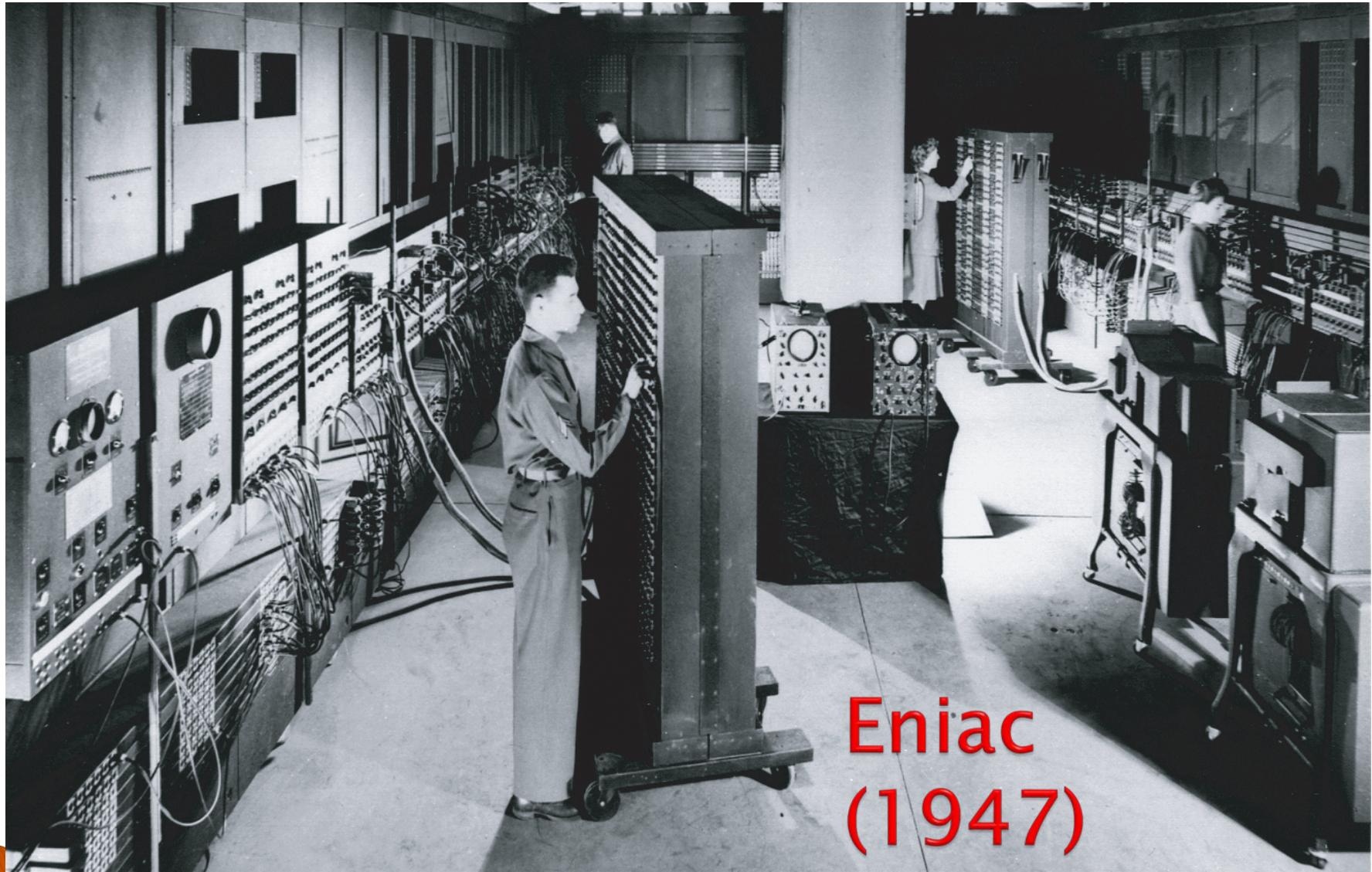
The Babbage
Difference Engine
(1832)

25,000 parts
cost: £17,470

Historical Perspective

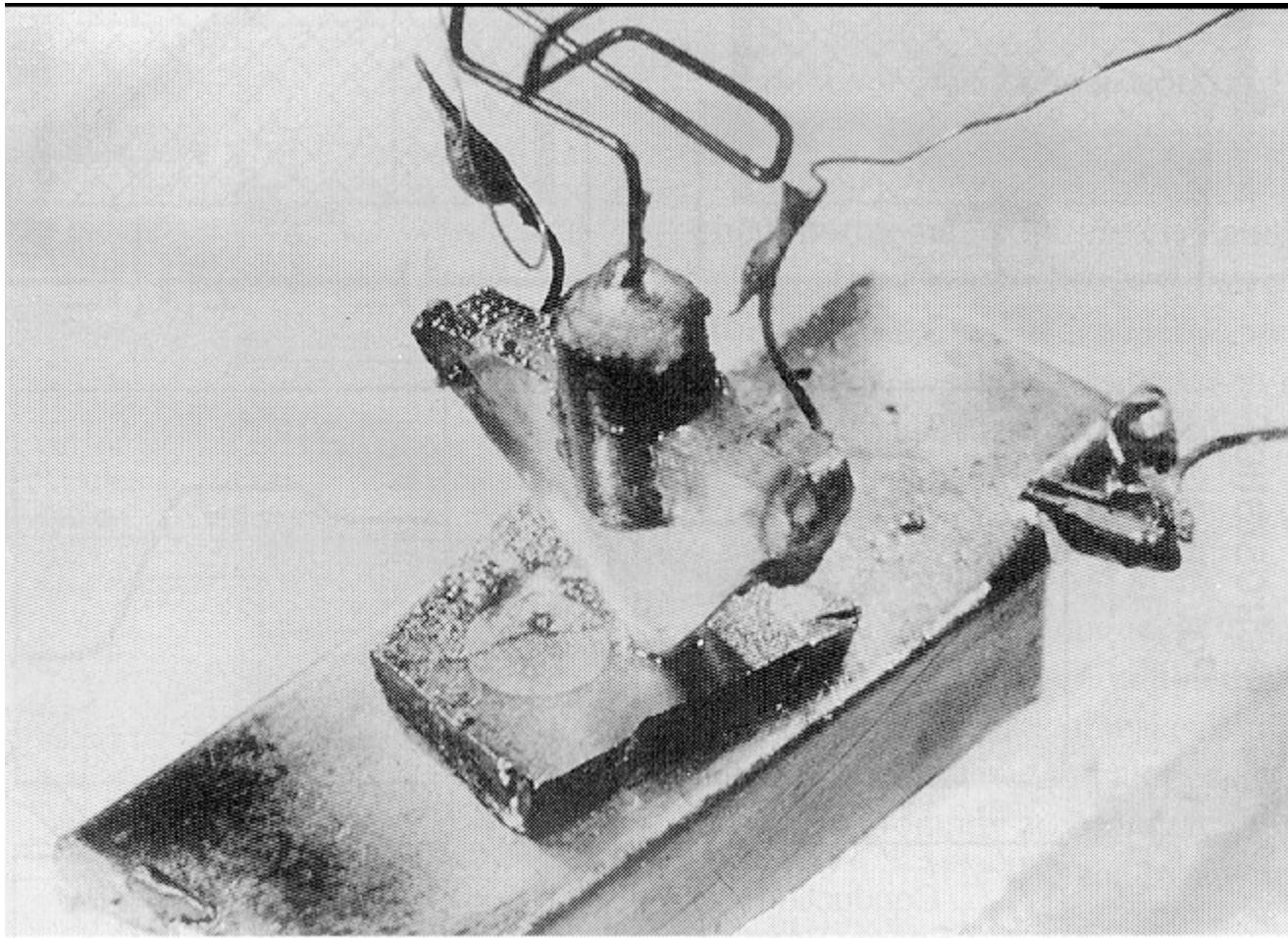
- ▶ ENIAC built in World War II was the first general purpose computer
 - Used for computing artillery firing tables
 - 30 tonnes
 - 25 m long by 3 m high
 - Each of the twenty 10 digit registers was half meter long
 - Used 18,000 vacuum tubes
 - Performed 1900 additions per second

Historical Perspective



Eniac
(1947)

The Transistor Revolution



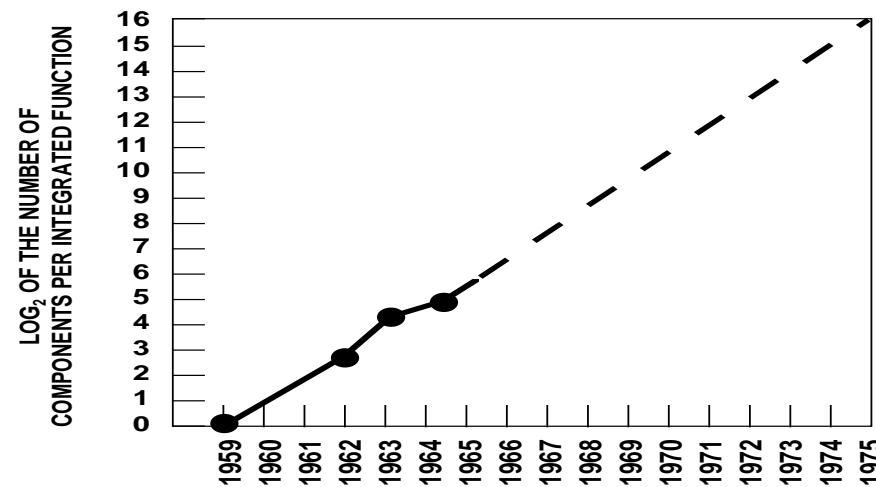
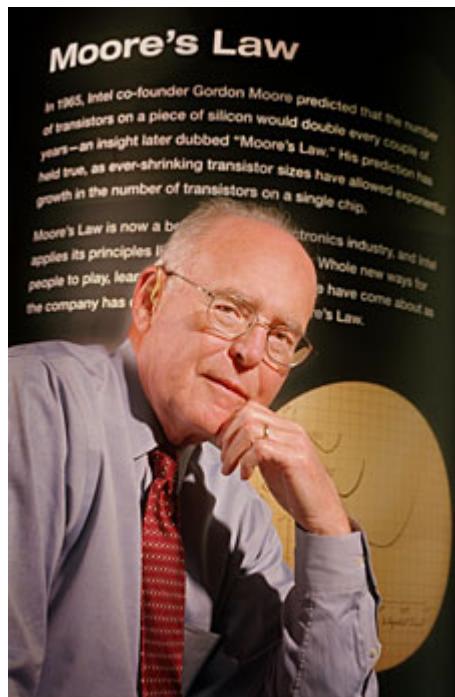
First transistor
Bell Labs, 1948

\$1 million First Commercial Computer: UNIVAC I (1951)



Moore's Law

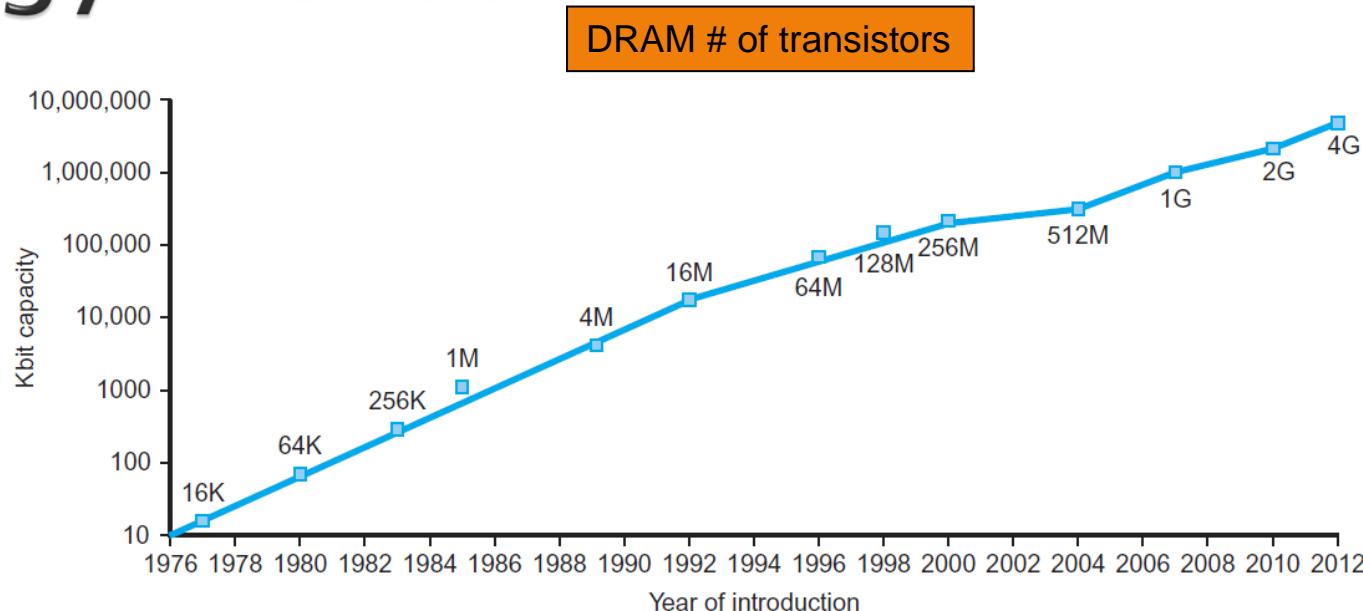
- In 1965, Gordon Moore noted that the number of transistors on a chip doubled every 18 to 24 months.



Electronics, April 19, 1965.

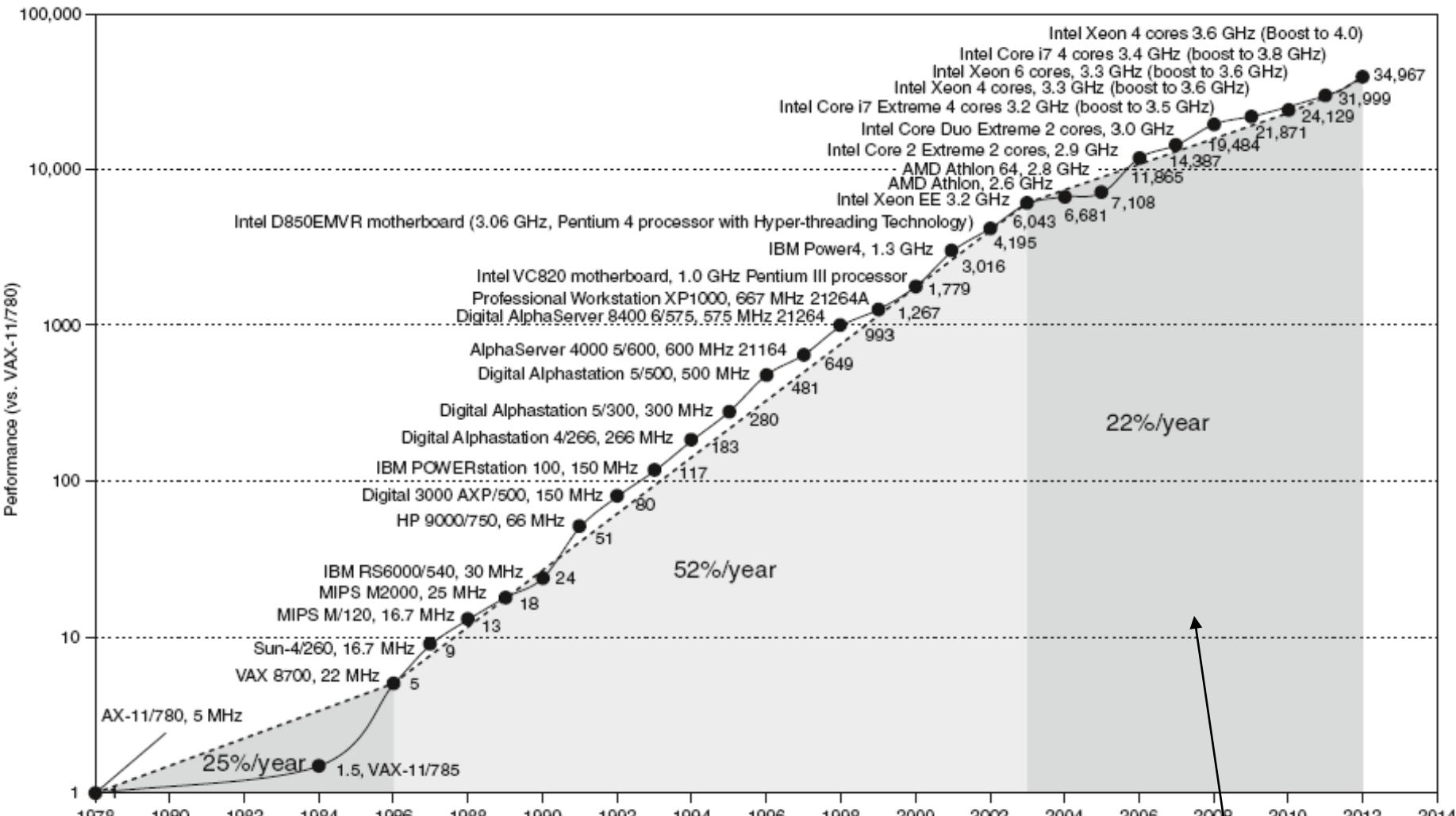
Technology Trends

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



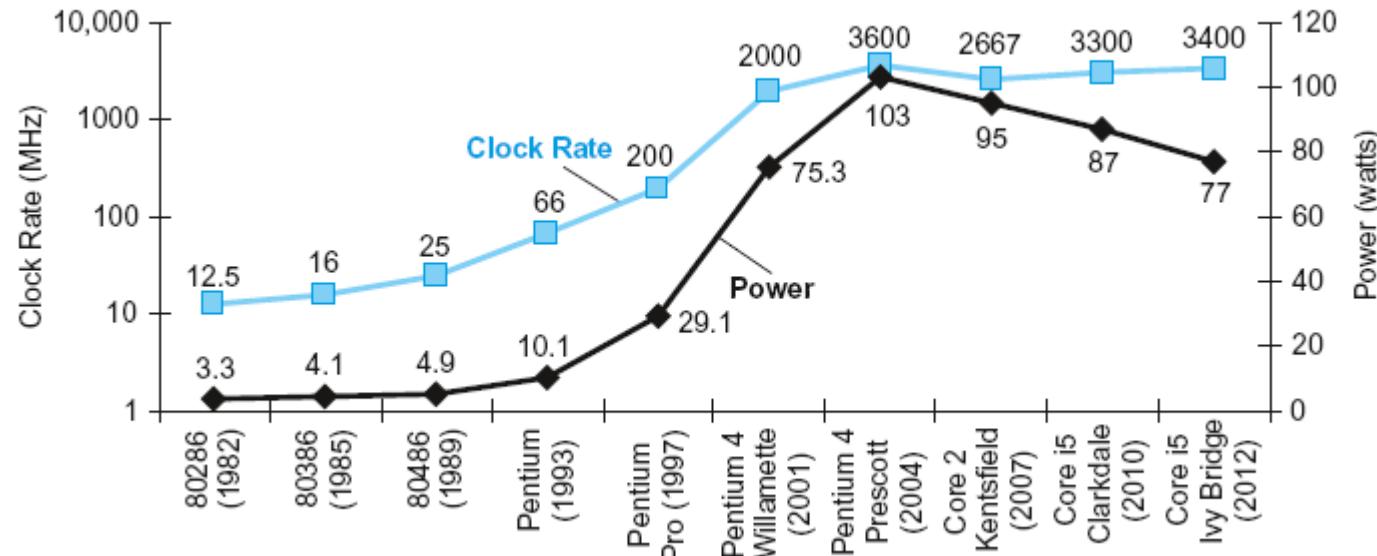
| Year | Technology | Relative performance/cost |
|------|----------------------------|---------------------------|
| 1951 | Vacuum tube | 1 |
| 1965 | Transistor | 35 |
| 1975 | Integrated circuit (IC) | 900 |
| 1995 | Very large scale IC (VLSI) | 2,400,000 |
| 2013 | Ultra large scale IC | 250,000,000,000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

Uniprocessor Performance



Constrained by power, instruction-level parallelism,
memory latency

Power Trends



- ▶ In CMOS IC technology

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

Reducing Power

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

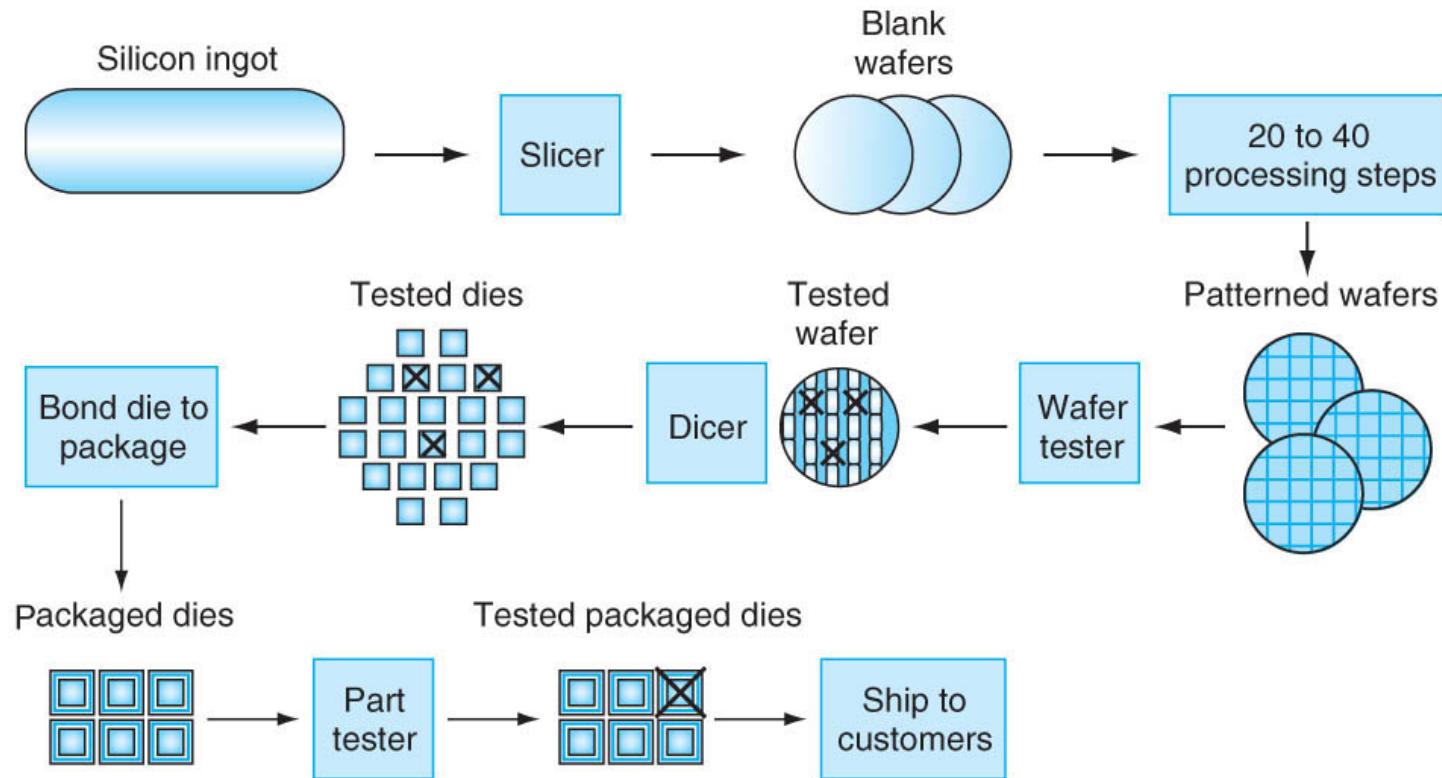
$$\frac{P_{\text{new}}}{P_{\text{old}}} = \frac{C_{\text{old}} \times 0.85 \times (V_{\text{old}} \times 0.85)^2 \times F_{\text{old}} \times 0.85}{C_{\text{old}} \times V_{\text{old}}^2 \times F_{\text{old}}} = 0.85^4 = 0.52$$

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

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

Manufacturing

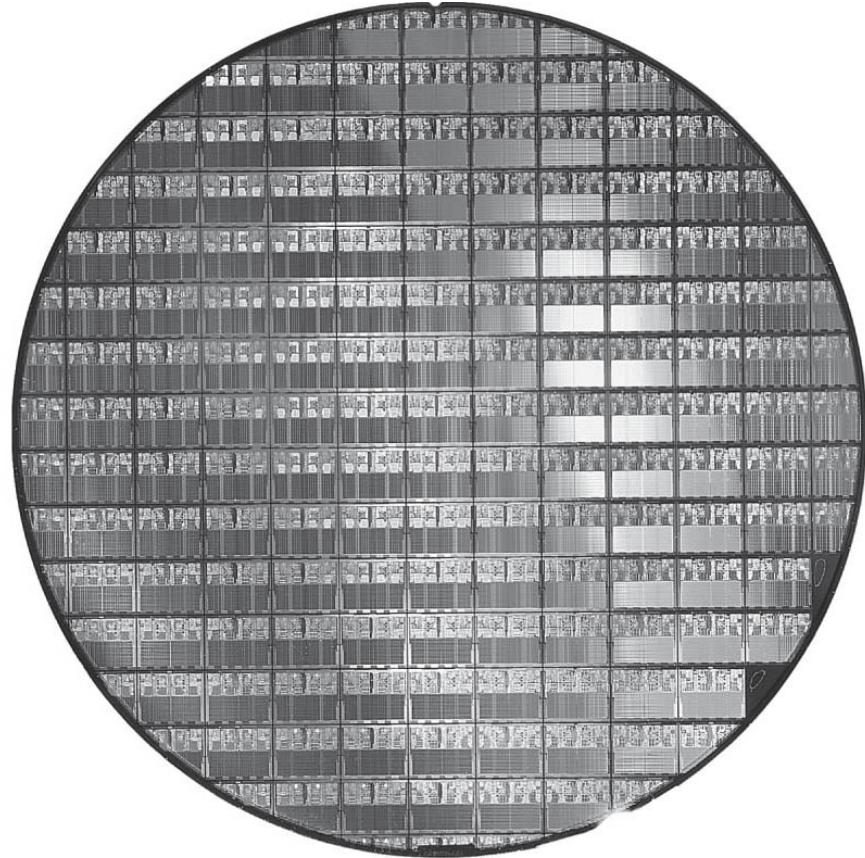


<https://www.youtube.com/watch?v=qm67wbB5Gml>

Manufacturing

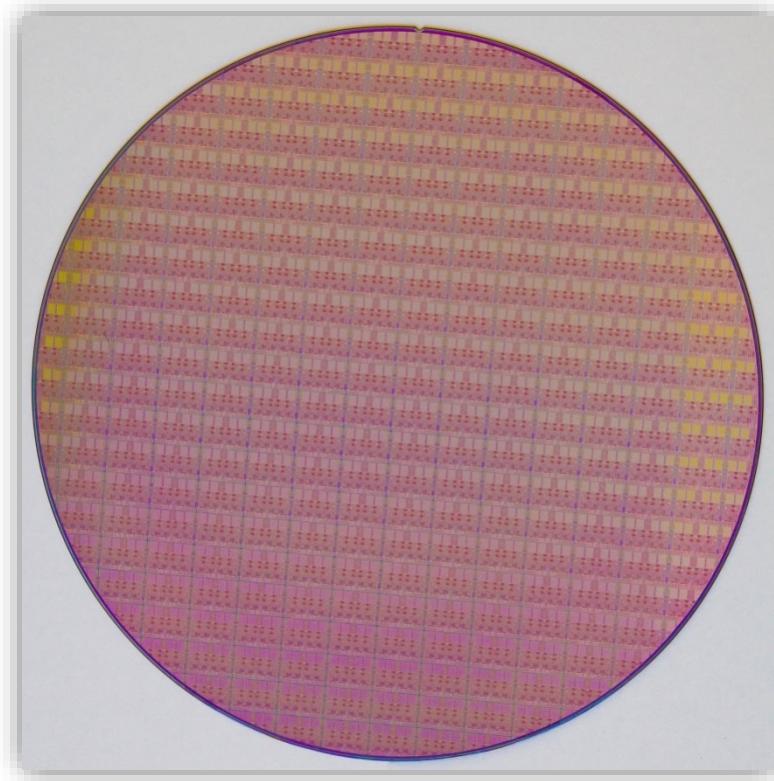


Single Crystal Silicon Ingot



- ▶ X2: 300mm wafer, 117 chips, 90nm technology

Intel Core i7 Wafer



- ▶ 300mm wafer, 280 chips, 32nm technology
- ▶ Each chip is 20.7 x 10.5 mm

Integrated Circuit Cost

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

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

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

- ▶ 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

$$Yield = \frac{\text{No. of good chips per wafer}}{\text{Total number of chips per wafer}} \times 100\%$$

| Chip | Metal layers | Line width | Wafer cost | Def./ cm ² | Area mm ² | Dies/wafer | Yield | Die cost |
|--------------|--------------|------------|------------|-----------------------|----------------------|------------|-------|----------|
| 386DX | 2 | 0.90 | \$900 | 1.0 | 43 | 360 | 71% | \$4 |
| 486 DX2 | 3 | 0.80 | \$1200 | 1.0 | 81 | 181 | 54% | \$12 |
| Power PC 601 | 4 | 0.80 | \$1700 | 1.3 | 121 | 115 | 28% | \$53 |
| HP PA 7100 | 3 | 0.80 | \$1300 | 1.0 | 196 | 66 | 27% | \$73 |
| DEC Alpha | 3 | 0.70 | \$1500 | 1.2 | 234 | 53 | 19% | \$149 |
| Super Sparc | 3 | 0.70 | \$1700 | 1.6 | 256 | 48 | 13% | \$272 |
| Pentium | 3 | 0.80 | \$1500 | 1.5 | 296 | 40 | 9% | \$417 |

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

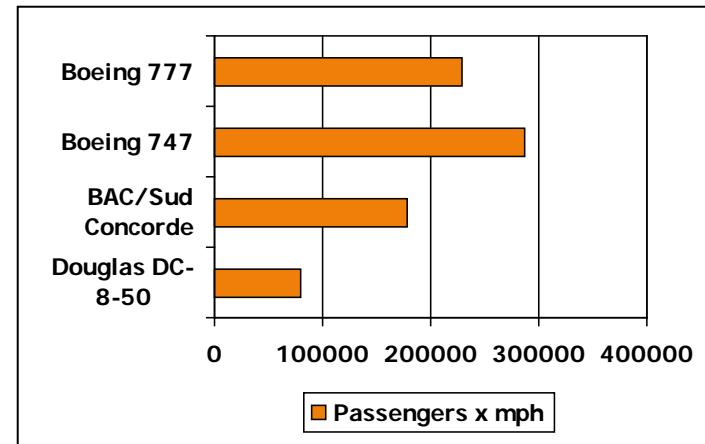
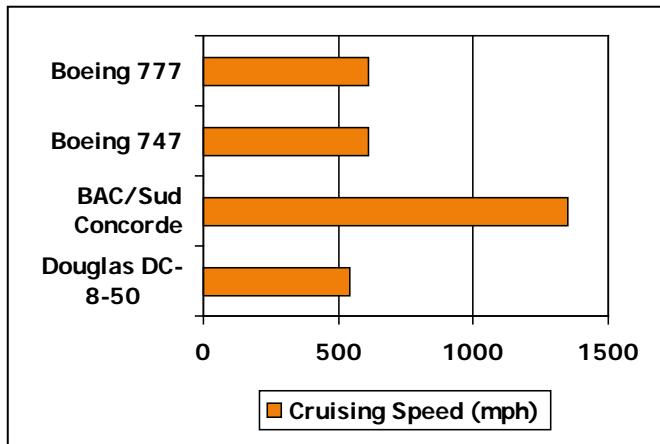
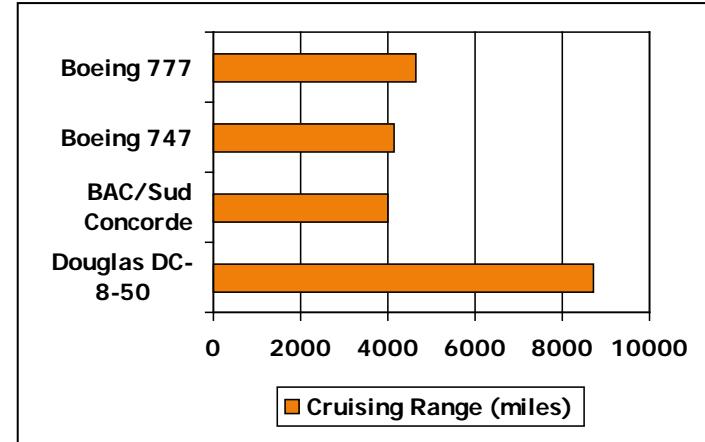
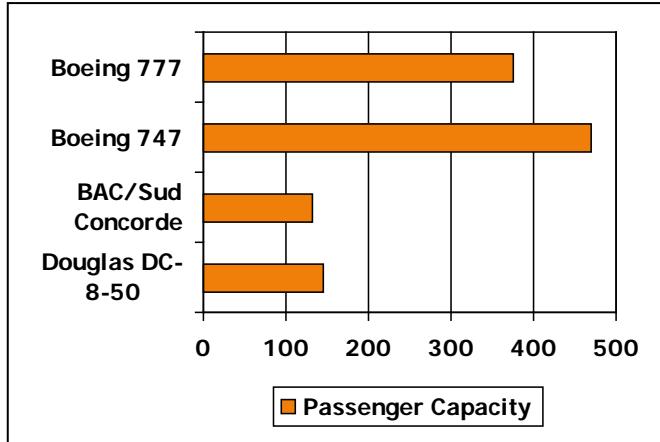


COMMON CASE FAST



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
 - $\text{Execution Time}_B / \text{Execution Time}_A$
 $= 15s / 10s = 1.5$
 - 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

Basic Measurement Metrics

▶ Comparing Machines

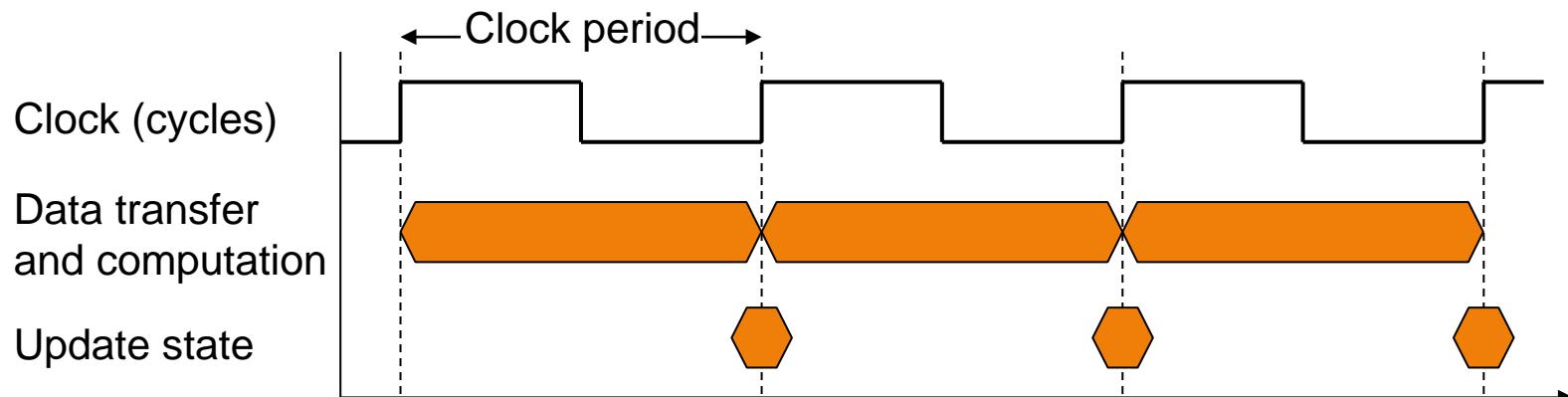
- Metrics
 - Execution time
 - Throughput
 - CPU time
 - MIPS – millions of instructions per second
 - MFLOPS – millions of floating point operations per second

▶ Comparing Machines Using Sets of Programs

- Arithmetic mean, weighted arithmetic mean
- Benchmarks

CPU Clocking

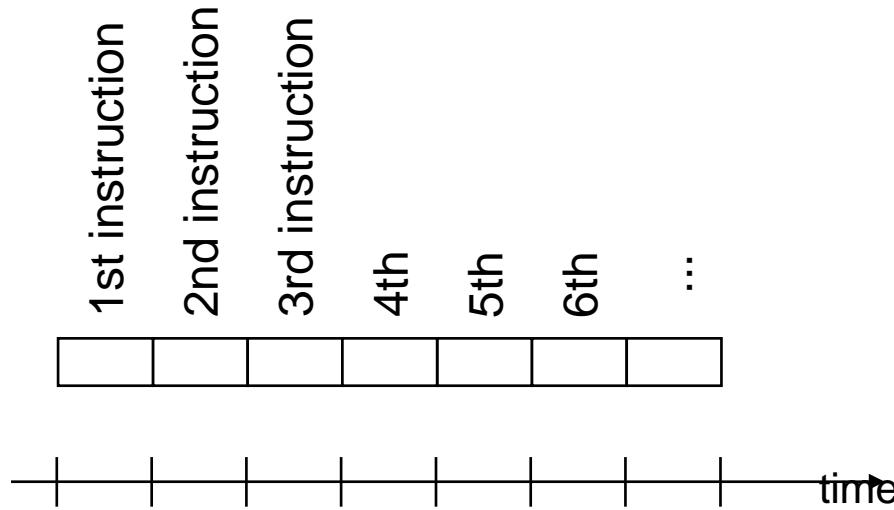
- ▶ Operation of digital hardware governed by a constant-rate clock



- 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., $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$

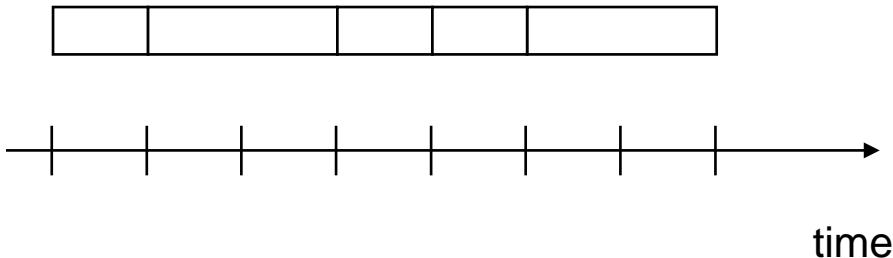
How Many Cycles are Required for a Program?

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



- This assumption is incorrect, different instructions take different amounts of time on different machines.

Different Numbers of Cycles for Different Instructions



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

Now That We Understand Cycles

- ▶ A given program will require
 - some number of instructions (machine instructions)
 - some number of clock 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 (cycles per instruction)

Computing CPU Time

- The time to execute a given program can be computed as

$$\text{CPU time} = \text{CPU clock cycles} \times \text{clock cycle time}$$

- Since clock cycle time and clock rate are reciprocals

$$\text{CPU time} = \text{CPU clock cycles} / \text{clock rate}$$

- The number of CPU clock cycles can be determined by

$$\text{CPU clock cycles} = (\text{instructions/program}) \times (\text{clock cycles/instruction})$$

$$= \text{Instruction count} \times \text{CPI}$$

which gives

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{clock cycle time}$$

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} / \text{clock rate}$$

- The units for CPU time are

$$\text{CPU time} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{clock cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{clock cycle}}$$

CPU Time Example

► Example 1:

- CPU clock rate is 1 MHz
- Program takes 45 million cycles to execute
- What's the CPU time?

$$45,000,000 * (1 / 1,000,000) = 45 \text{ seconds}$$

■ Example 2:

- CPU clock rate is 500 MHz
- Program takes 45 million cycles to execute
- What's the CPU time

$$45,000,000 * (1 / 500,000,000) = 0.09 \text{ seconds}$$

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 = $9/6 = 1.5$

Computing CPI

- ▶ The CPI is the average number of cycles per instruction.
- ▶ If for each instruction type, we know its frequency and number of cycles need to execute it, we can compute the overall CPI as follows:

$$\text{CPI} = \sum \text{CPI} \times F$$

- ▶ For example

| Op | F | CPI | CPI x F | % Time |
|--------|------|-----|---------|--------|
| ALU | 50% | 1 | .5 | 23% |
| Load | 20% | 5 | 1.0 | 45% |
| Store | 10% | 3 | .3 | 14% |
| Branch | 20% | 2 | .4 | 18% |
| Total | 100% | | 2.2 | 100% |

Performance

- ▶ Performance is determined by execution time
- ▶ Do you think any of the variables is sufficient enough to determine computer performance?
 - # of cycles to execute program?
 - # of instructions in program?
 - # of cycles per second?
 - average # of cycles per instruction?
 - average # of instructions per second
- It is not true to think that one of the variables is indicative of performance.

Number of Instruction 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?

$$\text{\# of cycles for first code} = (2 * 1) + (1 * 2) + (2 * 3) = 10 \text{ cycles}$$

$$\text{\# of cycles for second code} = (4 * 1) + (1 * 2) + (1 * 3) = 9 \text{ cycles}$$

$$10 / 9 = 1.11 \text{ times}$$

$$\text{CPI for first code} = 10 / 5 = 2$$

$$\text{CPI for second code} = 9 / 6 = 1.5$$

Performance Example

- ▶ Computers M1 and M2 are two implementations of the same instruction set with the same instruction count.

M1 has a clock rate of 50 MHz and M2 has a clock rate of 100 MHz.

M1 has a CPI of 2.8 and M2 has a CPI of 3.2 for a given program.

How many times faster is M2 than M1 for this program?

$$\frac{\text{ExTime}_{M1}}{\text{ExTime}_{M2}} = \frac{\text{IC}_{M1} \times \text{CPI}_{M1} / \text{Clock Rate}_{M1}}{\text{IC}_{M2} \times \text{CPI}_{M2} / \text{Clock Rate}_{M2}} = \frac{2.8/50}{3.2/100} = 1.75$$

- What would the clock rate of M1 have to be for them to have the same execution time?

$$2.8 / \text{Clock Rate}_{M1} = 3.2 / 100 \implies \text{Clock Rate}_{M1} = 87.5 \text{ MHz}$$

Problems with Arithmetic Mean

- ▶ Applications do not have the same probability of being run
- ▶ For example, two machines timed on two benchmarks:

| | Machine A | Machine B |
|-----------|------------------|------------------|
| Program 1 | 2 seconds (%20) | 6 seconds (20%) |
| Program 2 | 12 seconds (%80) | 10 seconds (%80) |

$$\text{Average execution time}_A = (2 + 12) / 2 = 7 \text{ seconds}$$

$$\text{Average execution time}_B = (6 + 10) / 2 = 8 \text{ seconds}$$

$$\text{Weighted average execution time}_A = 2*0.2 + 12*0.8 = 10 \text{ seconds}$$

$$\text{Weighted average execution time}_B = 6*0.2 + 10*0.8 = 9.2 \text{ seconds}$$

Poor Performance Metrics

- ▶ Marketing metrics for computer performance included MIPS and MFLOPS
- ▶ MIPS : millions of instructions per second
 - MIPS = instruction count / (execution time × 10⁶)
 - For example, a program that executes 3 million instructions in 2 seconds has a MIPS rating of 1.5
 - Advantage : Easy to understand and measure
 - Disadvantages : May not reflect actual performance, since simple instructions do better.
- ▶ MFLOPS : millions of floating point operations per second

MIPS Example

- ▶ Two different compilers are being tested for a 500 MHz. 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 billions Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.

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

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

MIPS Example (Con't)

| Code from | Instruction counts (in billions) for each instruction class | | |
|------------|--|----------|----------|
| | A (1 cy) | B (2 cy) | C (3 cy) |
| Compiler 1 | 5 | 1 | 1 |
| Compiler 2 | 10 | 1 | 1 |

$$\text{CPU Clock cycles}_1 = (5 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 10 \times 10^9$$

$$\text{CPU Clock cycles}_2 = (10 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 15 \times 10^9$$

$$\text{CPU time}_1 = 10 \times 10^9 / 500 \times 10^6 = 20 \text{ seconds}$$

$$\text{CPU time}_2 = 15 \times 10^9 / 500 \times 10^6 = 30 \text{ seconds}$$

$$\text{MIPS}_1 = (5 + 1 + 1) \times 10^9 / 20 \times 10^6 = 350$$

$$\text{MIPS}_2 = (10 + 1 + 1) \times 10^9 / 30 \times 10^6 = 400$$

Performance Summary

- ▶ The two main measure of performance are
 - **execution time** : time to do the task
 - **throughput** : number of tasks completed per unit time
- ▶ Performance and execution time are reciprocals. Increasing performance, decreases execution time.
- ▶ The time to execute a given program can be computed as:
$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{clock cycle time}$$
$$\text{CPU time} = \text{Instruction count} \times \text{CPI} / \text{clock rate}$$
- ▶ These factors are affected by compiler technology, the instruction set architecture, the machine organization, and the underlying technology.
- ▶ When trying to improve performance, look at what occurs frequently
=> make the common case fast.

Example

- ▶ Assume that a program runs in 100 seconds on a machine, with multiply operations responsible for 80 seconds. How much do I have to improve the speed of multiplication if I want my program to run 2 times faster.

Execution time after improvement =

$$\frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

$$50 \text{ seconds} = \frac{80 \text{ seconds}}{n} + (100 - 80 \text{ seconds})$$

$$n = \frac{80 \text{ seconds}}{30 \text{ seconds}} = 2.67$$

Amdahl's Law

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

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

- 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

- ▶ Speedup due to an enhancement is defined as:

$$\text{Speedup} = \frac{\text{ExTime old}}{\text{ExTime new}} = \frac{\text{Performance new}}{\text{Performance old}}$$

- ▶ Suppose that an enhancement accelerates a fraction
- ▶ **Fraction_{enhanced}** of the task by a factor **Speedup_{enhanced}**

$$\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times \left[\frac{(1 - \text{Fraction}_{\text{enhanced}})}{\text{Speedup}_{\text{enhanced}}} + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right]$$

$$\text{Speedup} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Example of Amdahl's Law

- Floating point instructions are improved to run twice as fast, but only 10% of the time was spent on these instructions originally. How much faster is the new machine?

$$\text{Speedup} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$
$$\text{Speedup} = \frac{1}{(1 - 0.1) + 0.1/2} = 1.053$$

- The new machine is 1.053 times as fast, or 5.3% faster.
- How much faster would the new machine be if floating point instructions become 100 times faster?

$$\text{Speedup} = \frac{1}{(1 - 0.1) + 0.1/100} = 1.109$$

Estimating Performance Improvements

- ▶ Assume a processor currently requires 10 seconds to execute a program and processor performance improves by 50 percent per year.
- ▶ By what factor does processor performance improve in 5 years?

$$(1 + 0.5)^5 = 7.59$$

- ▶ How long will it take a processor to execute the program after 5 years?

$$\text{ExTime}_{\text{new}} = 10 / 7.59 = 1.32 \text{ seconds}$$

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)

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 |

SPEC2017 Int Benchmark

| SPECrate 2017 Integer | SPECspeed 2017 Integer | Language [1] | KLOC [2] | Application Area |
|-----------------------|------------------------|--------------|----------|--|
| 500.perlbench_r | 600.perlbench_s | C | 362 | Perl interpreter |
| 502.gcc_r | 602.gcc_s | C | 1,304 | GNU C compiler |
| 505.mcf_r | 605.mcf_s | C | 3 | Route planning |
| 520.omnetpp_r | 620.omnetpp_s | C++ | 134 | Discrete Event simulation - computer network |
| 523.xalancbmk_r | 623.xalancbmk_s | C++ | 520 | XML to HTML conversion via XSLT |
| 525.x264_r | 625.x264_s | C | 96 | Video compression |
| 531.deepsjeng_r | 631.deepsjeng_s | C++ | 10 | Artificial Intelligence: alpha-beta tree search (Chess) |
| 541.leela_r | 641.leela_s | C++ | 21 | Artificial Intelligence: Monte Carlo tree search (Go) |
| 548.exchange2_r | 648.exchange2_s | Fortran | 1 | Artificial Intelligence: recursive solution generator (Sudoku) |
| 557.xz_r | 657.xz_s | C | 33 | General data compression |

CPU2017 Metrics

Q15. What are "SPECspeed" and "SPECrate" metrics?

There are many ways to measure computer performance. Among the most common are:

- Time - For example, seconds to complete a workload.
- Throughput - Work completed per unit of time, for example, jobs per hour.

SPECspeed is a time-based metric; SPECrate is a throughput metric.

| Calculating SPECspeed Metrics | Calculating SPECrate Metrics |
|---|--|
| 1 copy of each benchmark in a suite is run. | The tester chooses how many concurrent copies to run |
| The tester may choose how many OpenMP threads to use. | OpenMP is disabled. |
| For each benchmark, a performance ratio is calculated as: <code>time on a reference machine / time on the SUT</code> | For each benchmark, a performance ratio is calculated as: <code>number of copies * (time on a reference machine / time on the SUT)</code> |
| Higher scores mean that less time is needed. | Higher scores mean that more work is done per unit of time. |

Example:

- The reference machine ran 600.perlbench_s in 1775 seconds.
- A particular SUT took about 1/5 the time, scoring about 5.
- More precisely: $1775/354.329738 = 5.009458$

Example:

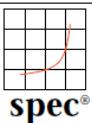
- The reference machine ran 1 copy of 500.perlbench_r in 1592 seconds.
- A particular SUT ran 8 copies in about 1/3 the time, scoring about 24.
- More precisely: $8 * (1592/541.52471) = 23.518776$

For both SPECspeed and SPECrate, in order to provide some assurance that results are repeatable, the entire process is repeated.

The tester may choose:

- a. To run the suite of benchmarks three times, in which case the tools select the medians.
- b. Or to run twice, in which case the tools select the lower ratios (i.e. slower). *New with CPU2017*

For both SPECspeed and SPECrate, the selected ratios are averaged using the Geometric Mean, which is reported as the overall metric.



SPEC CPU2017 Integer Speed Result

Copyright 2017-2018 Standard Performance Evaluation Corporation

| | |
|--|-------------------------------|
| Dell Inc. | SPECspeed2017_int_base = 8.97 |
| PowerEdge MX740c (Intel Xeon Platinum 8160 CPU, 2.10GHz) | SPECspeed2017_int_peak = 9.23 |

CPU2017 License: 55

Test Sponsor: Dell Inc.

Tested by: Dell Inc.

Test Date: Mar-2018

Hardware Availability: Sep-2017

Software Availability: Sep-2017

Hardware

CPU Name: Intel Xeon Platinum 8160
Max MHz.: 3700
Nominal: 2100
Enabled: 48 cores, 2 chips
Orderable: 1,2 chips
Cache L1: 32 KB I + 32 KB D on chip per core
L2: 1 MB I+D on chip per core
L3: 33 MB I+D on chip per chip
Other: None
Memory: 192 GB (12 x 16 GB 2Rx8 PC4-2666V-R)
Storage: 960 GB SAS SSD
Other: None

Software

OS: SUSE Linux Enterprise Server 12 SP3
4.4.114-94.11-default
Compiler: C/C++: Version 18.0.0.128 of Intel C/C++ Compiler for Linux;
Fortran: Version 18.0.0.128 of Intel Fortran Compiler for Linux
Parallel: Yes
Firmware: Version 0.3.15 released Mar-2018
File System: xfs
System State: Run level 3 (multi-user)
Base Pointers: 64-bit
Peak Pointers: 32/64-bit
Other: jemalloc memory allocator library V5.0.1

<https://www.spec.org/cpu2017/results/cint2017.html>

The reference machine is a historical Sun Microsystems server, the Sun Fire V490 with 2100 MHz UltraSPARC-IV+ chips. The UltraSPARC-IV+ was introduced in 2006, and is newer than the chip used in the CPU2000 and CPU2006 reference machines (the 300 MHz 1997 UltraSPARC II).

Summary of Performance Evaluation

- ▶ Response Time and Throughput
- ▶ CPU time = # instr x CPI x cycle time

- ▶ MIPS and MFLOPS poor performance metrics
- ▶ Amdahl's law provides an efficient method for determining speedup due to an enhancement.
- ▶ SPEC benchmarks to measure CPU performance

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