CS 207 D

Computer Architecture

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 evasive

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

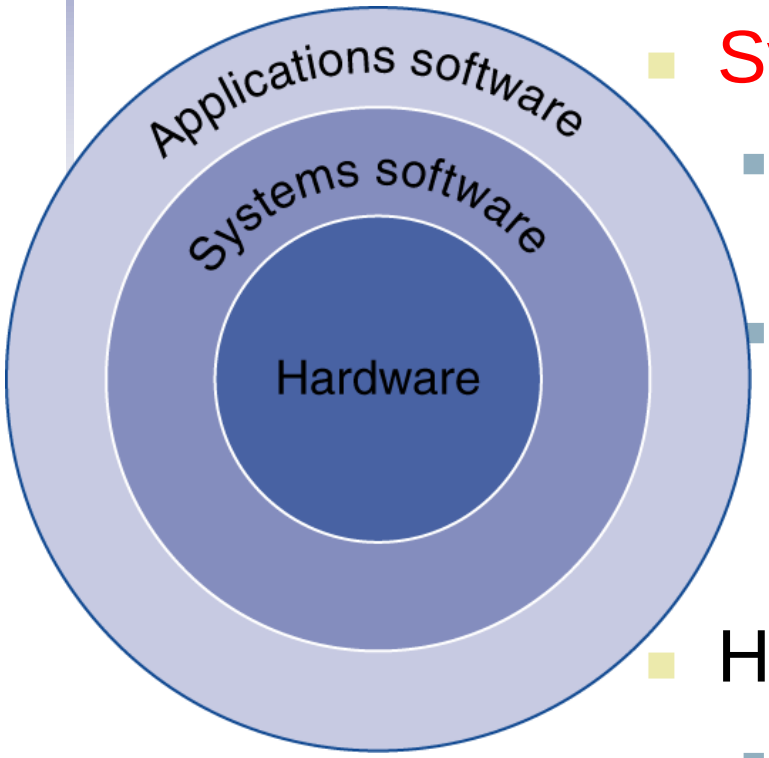
- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance
- What is parallel processing

Understanding Performance

- Algorithm
  - Determines number of operations executed
- Programming language, compiler and 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

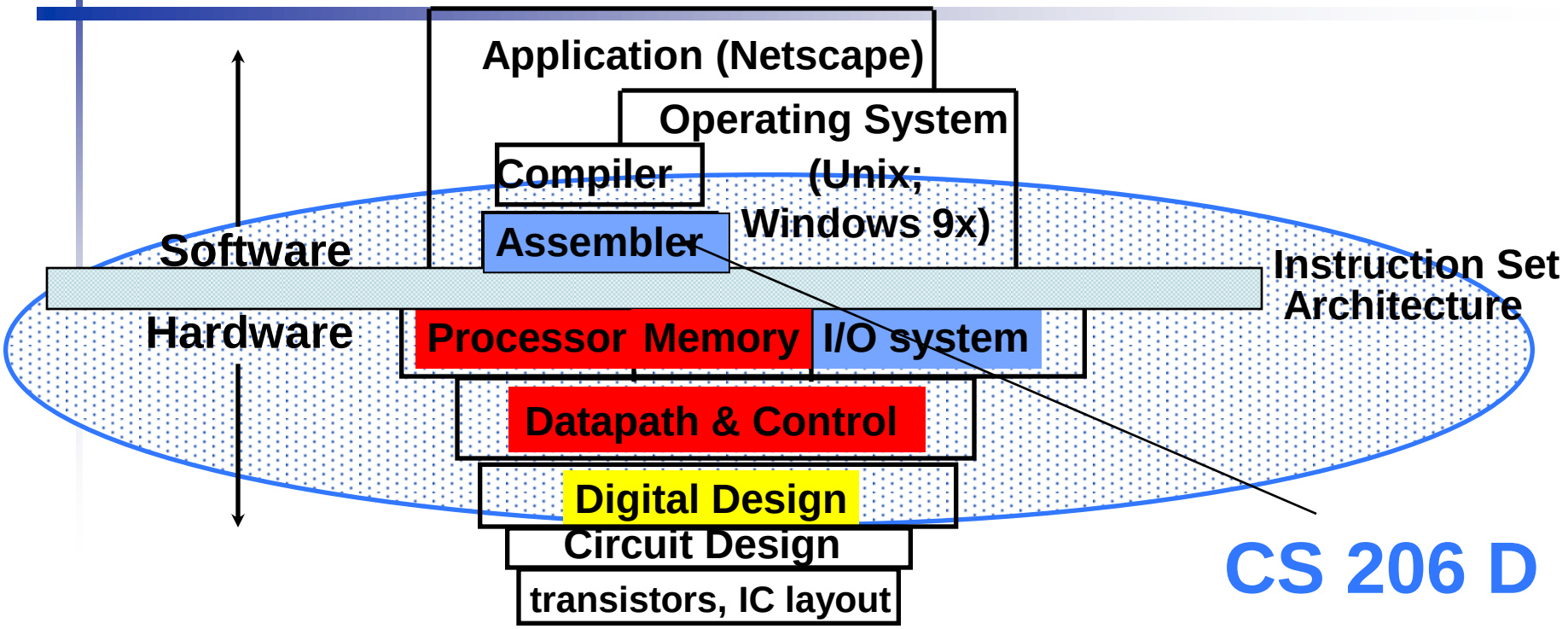
How the algorithm, programming language, processor and OS affect the computer performance?

Below Your Program



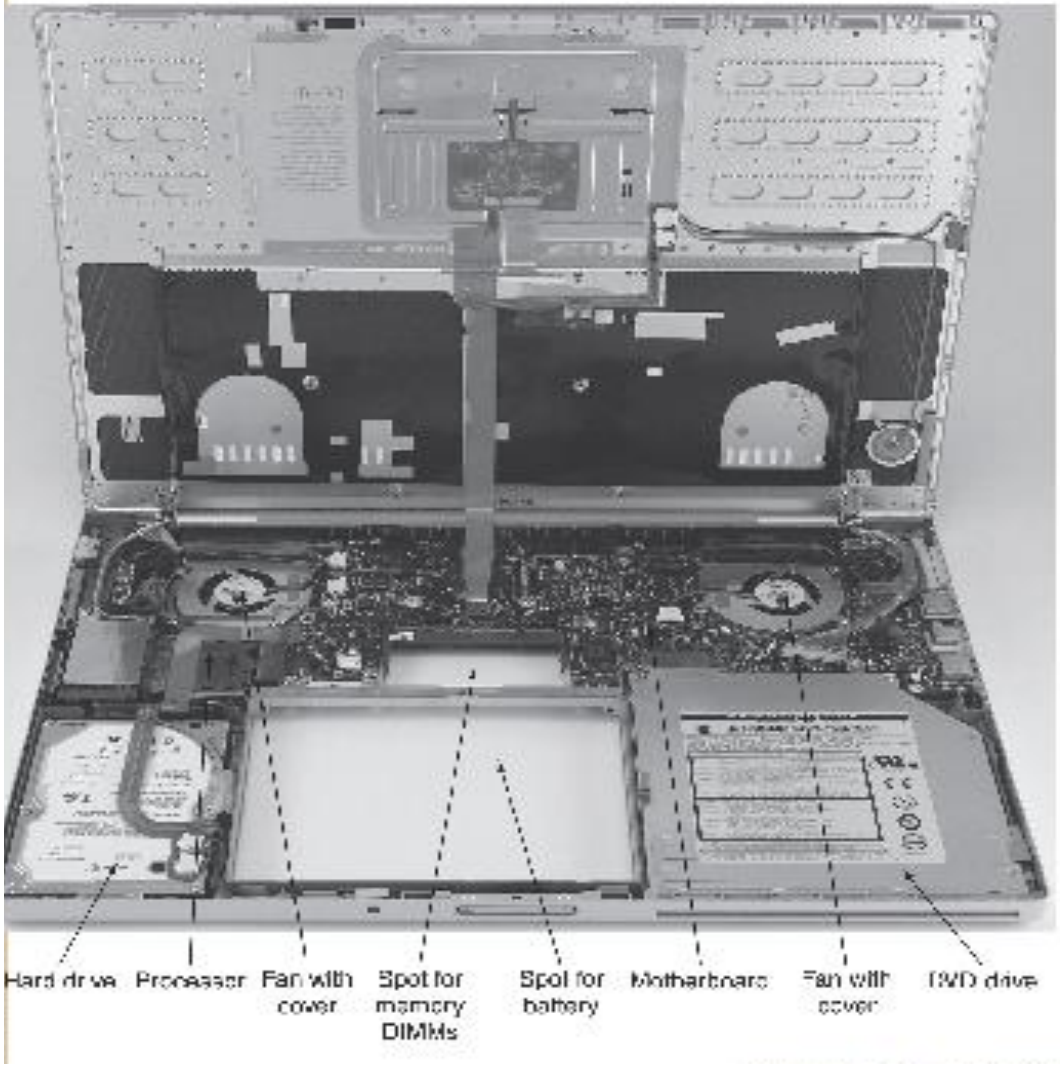
- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O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

What is “Computer Architecture”?



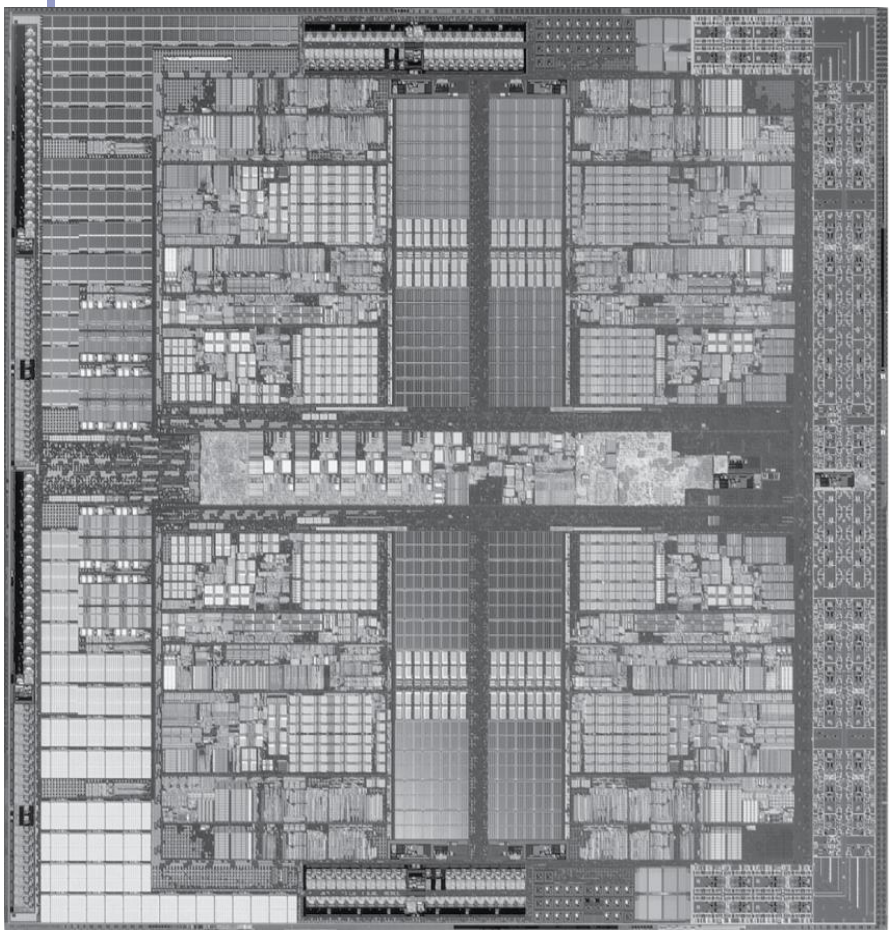
- Key Idea: *levels of abstraction*
  - hide unnecessary implementation details
  - helps us cope with enormous complexity of real systems

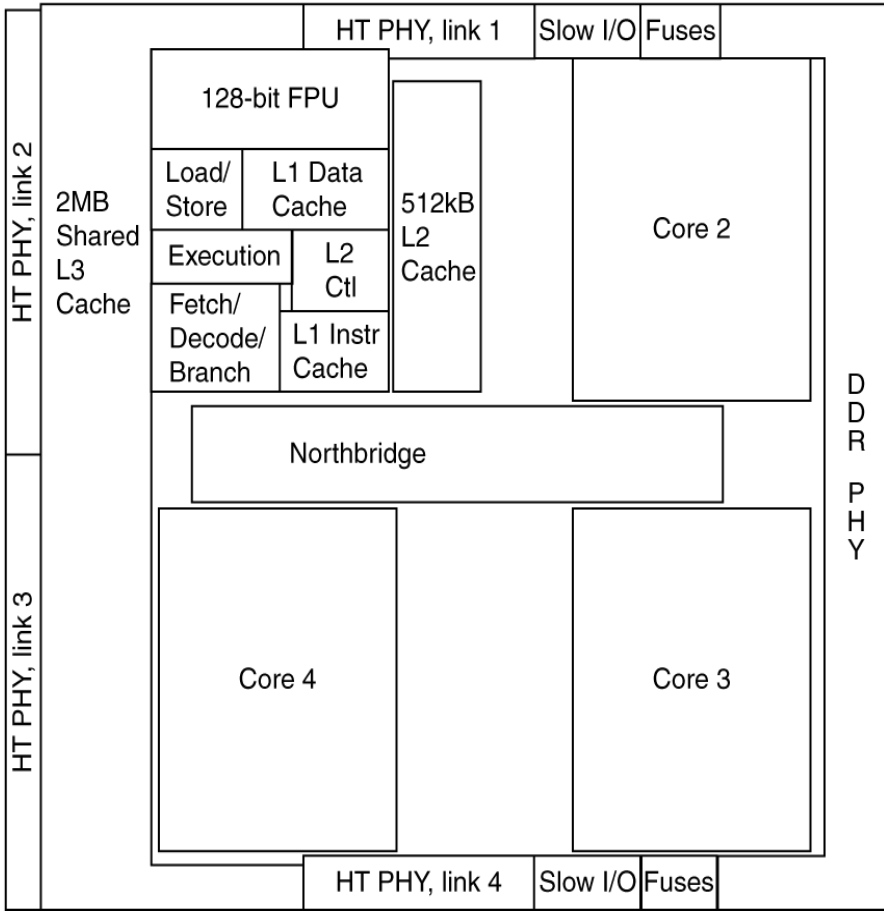
OPENING THE BOX



Inside the Processor

- AMD Barcelona: 4 processor cores





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
```

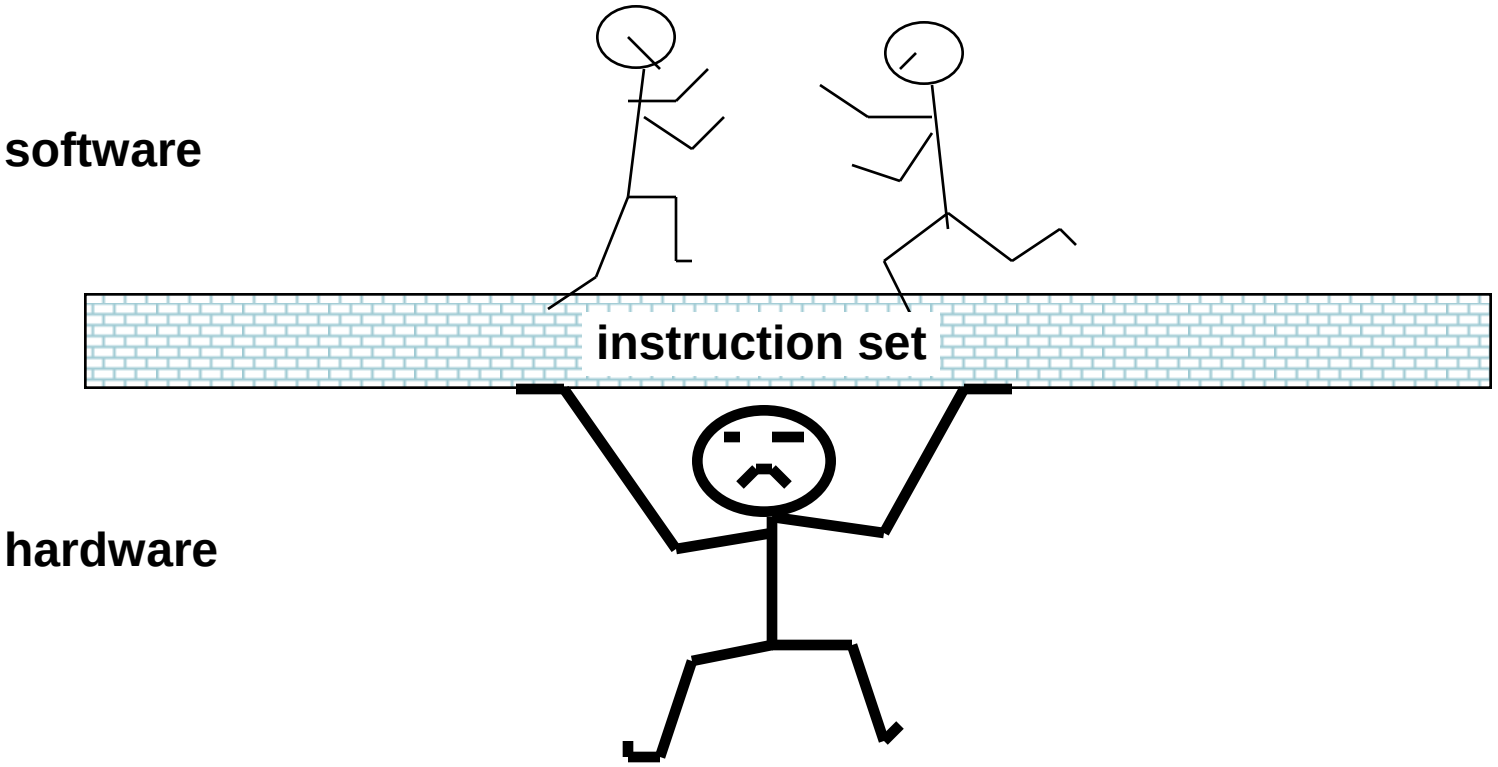
Assembler

Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000000011000000110000010001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Define the High level, assembly and machine language?

The Instruction Set: A Critical Interface



Abstractions - ISA

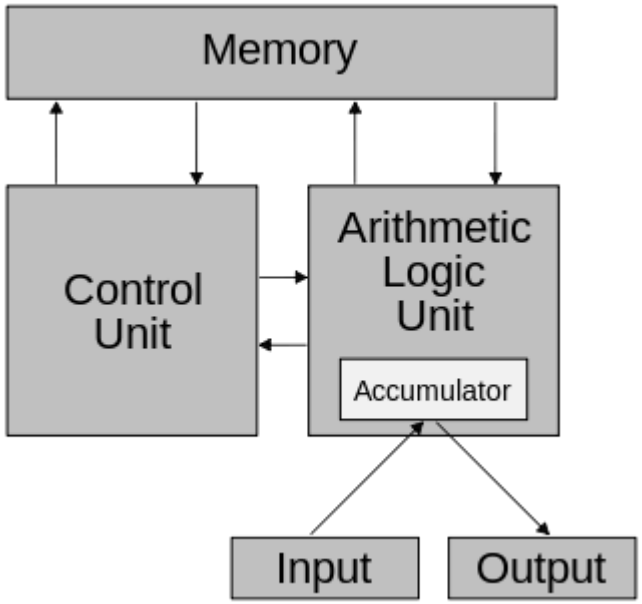
- Abstraction helps us deal with complexity
  - Hide lower-level detail
- **Instruction set architecture (ISA)**
  - **The hardware/software interface**
  - **Application binary interface**
- The ISA plus system software interface
- Implementation (Hardware obeys instruction)
- The details underlying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

Define ISA

An abstract interface between the hardware and lower(st)- level software that encompasses all the information necessary to write a machine language program; that will run correctly including instructions, memory, I/O, registers, etc.

The Von Neumann Computer

- Stored-Program Concept – Storing programs as numbers – by John von Neumann – Eckert and Mauchly worked in engineering the concept.
- Idea: A program is written as a sequence of instructions, represented by binary numbers. The instructions are stored in the memory just as data. They are read one by one, decoded and then executed by the CPU.



Historical Perspective

- **1944: The First Electronic Computer ENIAC (Electronic Numerical Integrator And Computer) at IAS Institute for Advanced Studies,** (18,000 vacuum tubes)
- **Decade of 70's (Microprocessors)**

  Programmable Controllers, Single Chip Microprocessors

  Personal Computers
- **Decade of 80's (RISC Architecture)**

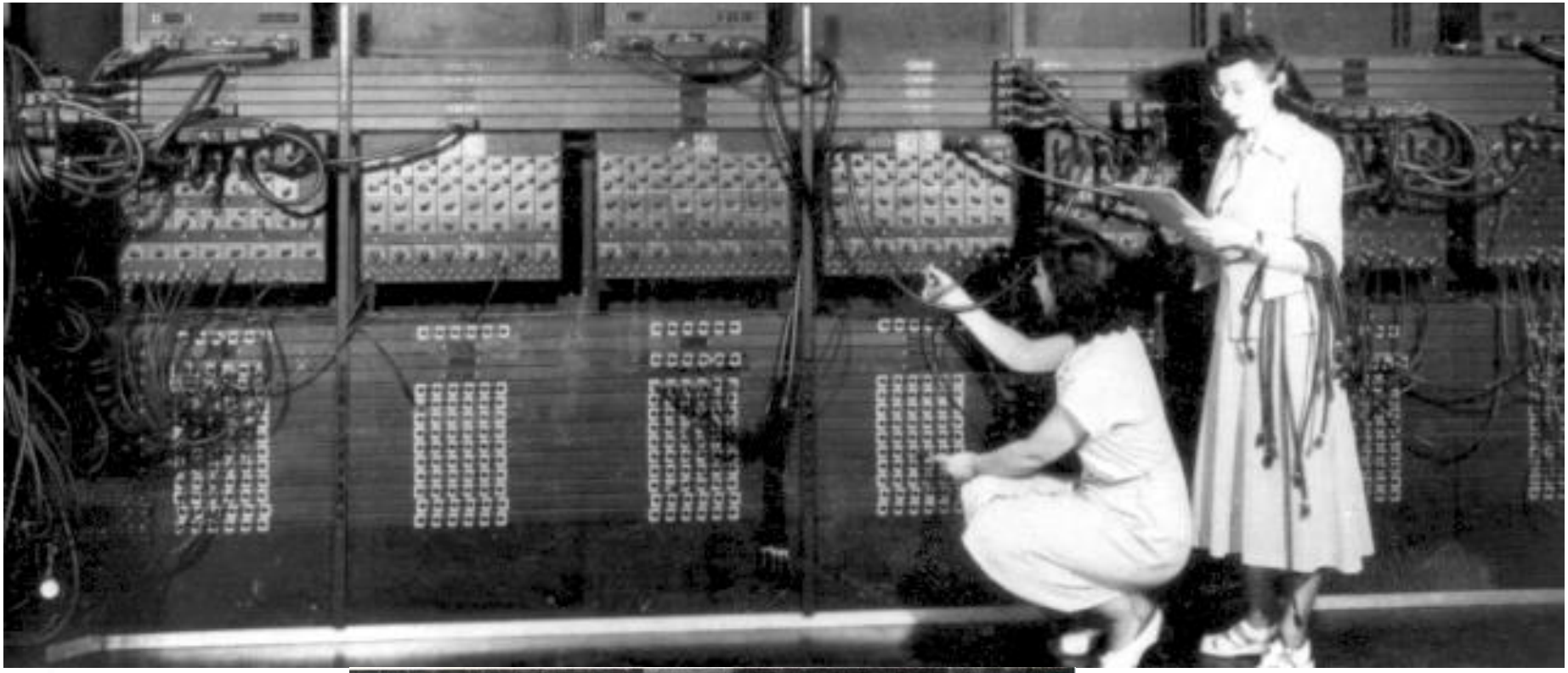
  Instruction Pipelining, Fast Cache Memories

  Compiler Optimizations
- **Decade of 90's (Instruction Level Parallelism)**

  Superscalar Processors, Instruction Level Parallelism (ILP),

  Aggressive Code Scheduling, Out of Order Execution
- **Decade of 2000's (Multi-core processors)**

  Thread Level Parallelism (TLP), Low Cost Supercomputing



Performance Growth In Perspective

What is important for computer performance ?

(One computer is better than another ?)

- Doubling every 18 months since 1982
- Doubling every 24 months since 1970

Technology => Dramatic Change

- Processor
  - 2X in performance every 1.5 years; 1000X performance in last decade (Moore's Law)
- Main Memory
  - DRAM capacity: 2x / 2 years; 1000X size in last decade
  - Cost/bit: improves about 25% per year
- Disk
  - capacity: > 2X in size every 1.5 years
  - Cost/bit: improves about 60% per year

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

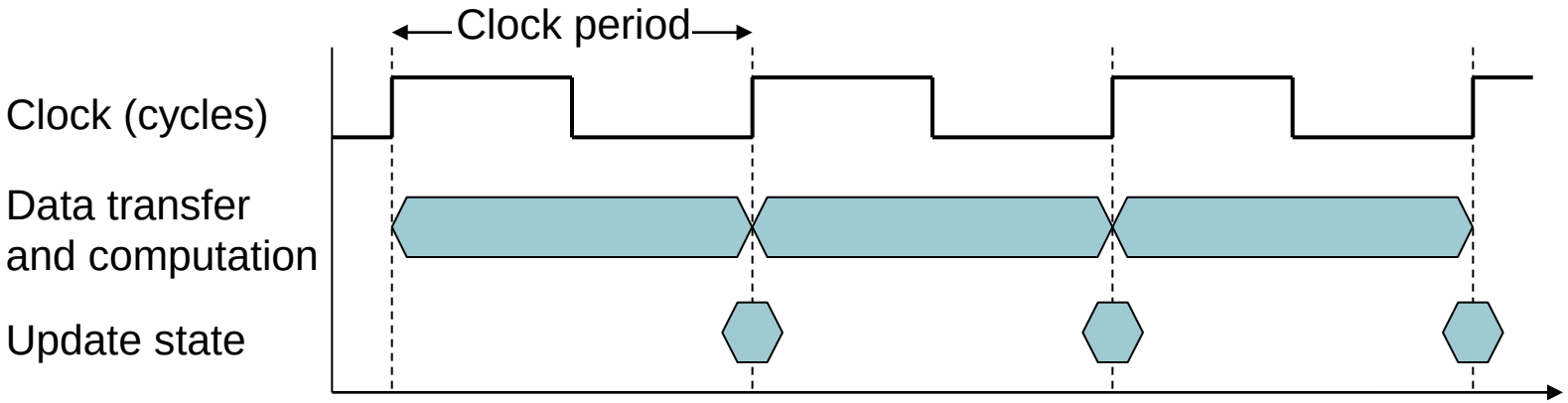
Measuring Execution Time

- **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

Compare between the elapsed and CPU times?

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**
  1. **Reducing number of clock cycles (good algorithm or hardware design)**
  2. **Increasing clock rate (good technology)**
  3. **Hardware designer must often trade off clock rate against cycle count**

The performance improves by increasing the number of clocks by good algorithm and hardware design (T?F)

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?

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

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

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

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

Instruction Count and CPI

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

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

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

$$= Ic \times CPI \times t$$

- **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?

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

$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

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

$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

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

…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

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

Relative Instruction frequency

INSTRUCTION EXECUTION RATE

A processor is driven by a clock with a constant frequency f or, equivalently, a constant cycle time **t**, where $t = 1/f$

Ic : Instruction count for a program as the number of **machine instructions** executed for that program until it runs to completion or for *some defined time interval*

CPI i : the number of cycles required for instruction type **i**.

Ii : be the number of executed instructions of type **I** for a given program.

Then we can calculate an overall **CPI** as follows:

$$CPI = \frac{\sum_{i=1}^{n} (CPIi * Ii)}{Ic}$$

The processor time(***execution time)*** **T** needed to execute a given program

$$T = Ic * CPI * t$$

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

MIPS as a Performance Metric

A common measure of performance for a processor *is the rate at which instructions are executed, expressed as millions of instructions per second (**MIPS**)*, referred to as the **MIPS rate**.

- **Millions of instructions per second (MIPS)**
- **Millions of FLOating point instructions Per Second (MFLOPS)**

We can express the MIPS rate in terms of the clock rate and **CPI** as follows

$$\text{MIPS rate} = Ic / (T * 10^6) = f / (CPI * 10^6)$$

Where $T = Ic * CPI * t$

MFLOPS as a Performance Metric

Another common performance measure deals **Floating-point** performance is expressed as millions of floating-point operations per second (**MFLOPS**), defined as follows ***only with floating-point instructions.***

$$\text{MFLOPS} = \frac{\text{Number of executed floating-point operations in a program}}{\text{execution time} * 10^6}$$

$$\text{MFLOPS} = I_f / (T * 10^6)$$

~~MIPS~~ : *Meaningless Indicator of Performance*

it enough to use MIPS to differentiate between different computers?

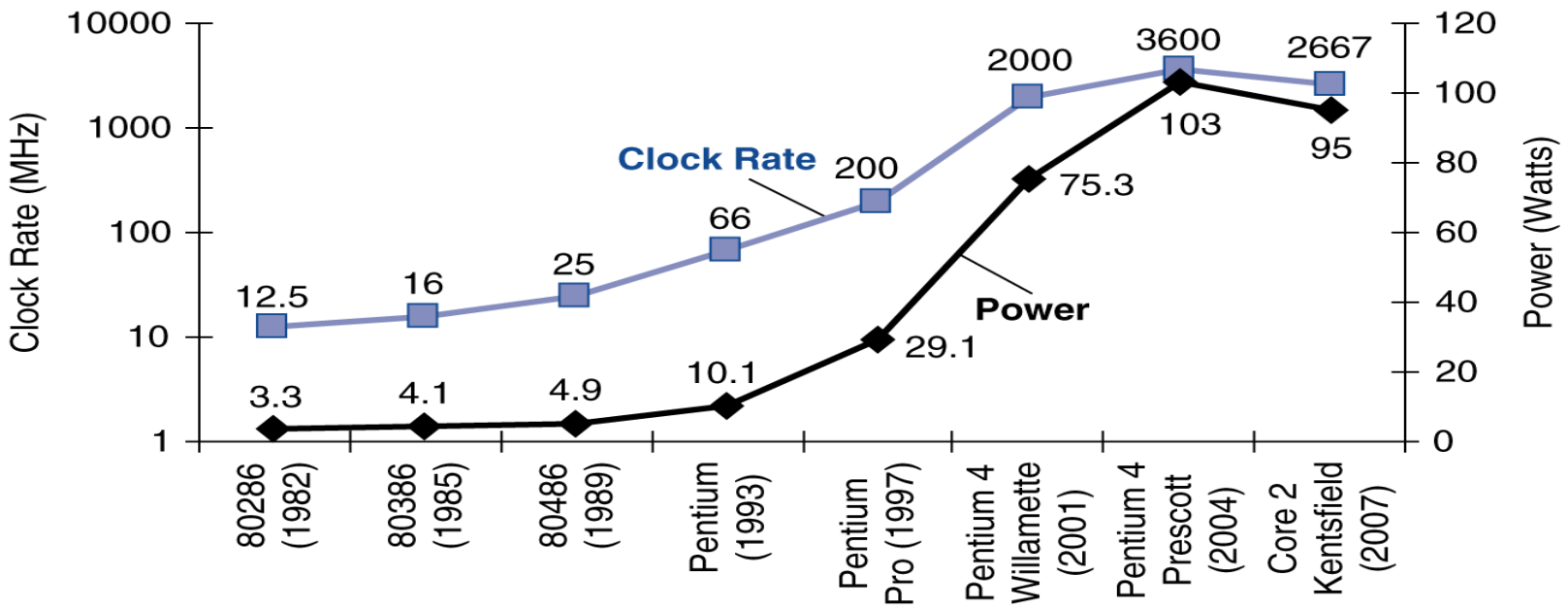
Performance Summary

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 (**Complementary metal–oxide–semiconductor**) IC technology

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

×30 | 5V → 1V | ×1000

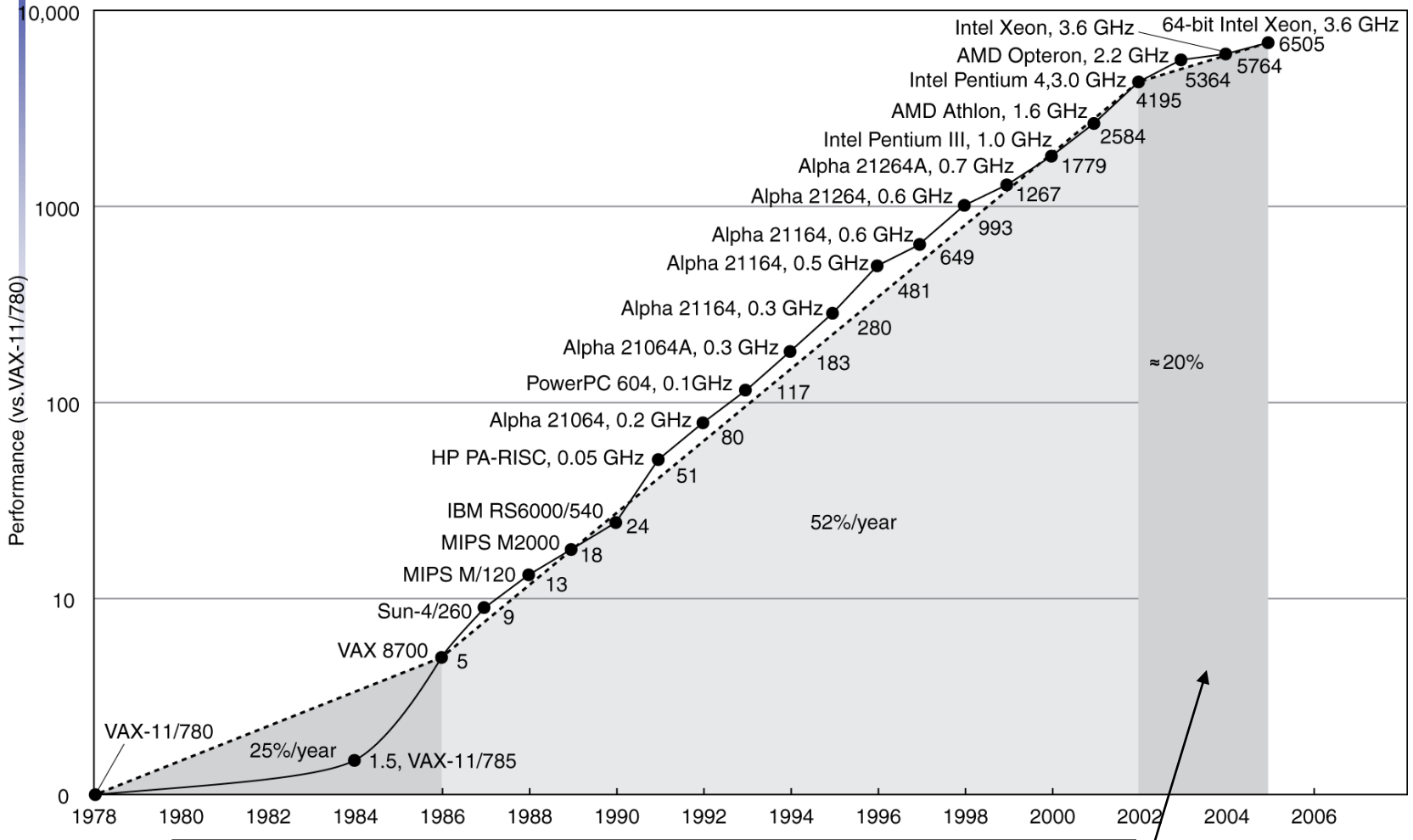
Reducing Power

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

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

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

Uniprocessor Performance



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

Benchmarks

- Programs designed to test performance, Written in high level language, Portable
- Represents style of task
  - Systems, numerical, commercial
- Easily measured
- Widely distributed
- E.g. **System Performance Evaluation Corporation (SPEC)**
  - CPU2006 for computation bound https://www.spec.org/
    - 17 floating point programs in C, C++, Fortran
    - 12 integer programs in C, C++
    - 3 million lines of code
  - **Speed** and **rate** metrics
    - Single task and throughput (rate of a machine carrying out a number of tasks)

System Performance Evaluation Corporation SPEC Speed Metric

- **Single task**
- Base runtime defined for each benchmark using **reference machine**
- Results are reported as ratio of reference time to system run time
  - **$Tref_i$ execution time for benchmark i on reference machine**
  - **$Tsut_i$ execution time of benchmark i on test system**

$$r_i = \frac{Tref_i}{Tsut_i}$$

- **Overall performance** calculated by averaging ratios for all ***12*** integer benchmarks
  - **Use geometric mean**
    - Appropriate for normalized numbers such as ratios

$$r_G = \left(\prod_{i=1}^{n} r_i \right)^{1/n}$$

, $\prod x_i = x_1 \cdot x_2 \cdot \ldots \cdot x_n$

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)

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

SPEC Rate Metric

- Measures **throughput** or rate of a machine carrying out a number of tasks
- **Multiple copies** of benchmarks **run simultaneously**
  - Typically, same as number of processors
- Ratio is calculated as follows:
  - $Tref_i$ reference execution time for benchmark i
  - N number of copies run simultaneously
  - Tsuti ***elapsed*** time from start of execution of program on all N processors until completion of all copies of program
  - Again, a geometric mean is calculated

$$r_i = \frac{N \times Tref_i}{Tsut_i}$$

SPEC Power Benchmark

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

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

- (ssj_ops) The number of Server-Side Java OPerationS
- **performance per watt** is a measure of the energy efficiency of a particular computer architecture or computer hardware. It measures the rate of computation that can be delivered by a computer for every watt of power consumed.
- **FLOPS** (Floating Point Operations Per Second) **per watt** is a common measure

Amdahl's Law

- ***Potential speed up of program using multiple processors***
- Improving an aspect of a computer and expecting a proportional improvement in overall performance

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

Example: 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?

- How much improvement in multiply performance to get 5× overall?

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

- Can't be done!

Amdahl's Law Formula

- *It deals with the potential speedup of a program using multiple processors compared to a single processor.*
- For program running on single processor
  - Fraction ***f*** of code infinitely **parallelizable** with no scheduling overhead
  - Fraction (**1-*f***) of code inherently **serial**
  - T is total execution time for program on single processor
  - N is number of processors that fully exploit parallel portions of

$$Speedup = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{T(1-f)+Tf}{T(1-f)+\frac{Tf}{N}} = \frac{1}{(1-f)+\frac{f}{N}}$$

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