COMP 140
Advanced Computer Architecture

Chris Gregg

Summer 2014

About Your Instructor

Tufts Lecturer, Computer Science

My Background

- PhD from the University of Virginia, Computer Engineering. Thesis: Heterogeneous Scheduling between CPUs and GPUs
- Intern at AMD, Summer 2010
- High school physics / CS teacher, 2002-2008
- Master's degree in Education, Harvard, 2002
- U.S. Naval Officer (Cryptology / Information Warfare), 1994 – 2001 (Active Duty), 2004-Present (Navy Reserves)
- Bachelor's of Electrical Engineering, Johns Hopkins University, 1994

What will we learn in this course?

- Fundamentals of Computer Architecture, and the Current State of Computer Hardware
- The Processor: Instruction Set Principles
- The Processor: Building a Data Path
- Instruction Pipelining: Data and Control Hazards
- Exploiting Memory Hierarchy: Caches, Virtual Memory
- Data-Level Parallelism (Vector, SIMD, and GPU architectures)
- Labs: Building a MIPS simulator, GPU programming

Logistics

Class Web Page:

http://www.cs.tufts.edu/comp/140

Piazza Page:

http://piazza.com/tufts/summer2014/comp140

Textbooks (available online through Tufts):

Computer Organization and Design, 4th ed., Hennessy and Patterson, 2009. ("COAD")
http://app.knovel.com/web/toc.v/cid:kpCAAQAE11

Computer Architecture, A Quantitative Approach, 5th ed. Hennessy and Patterson, 2012. ("CAQA")
http://app.knovel.com/web/toc.v/cid:kpCODTHSI3

Lectures -vs- Labs

We will break each day into two parts: lecture / discussion, and lab work.

Please bring a laptop to class if you have one, and please install the following programs (both require Java):

Logisim:

http://ozark.hendrix.edu/~burch/logisim/

MARS:

http://courses.missouristate.edu/KenVollmar/MARS/MARS_4_4_Aug2013/Mars4_4.jar

Lectures -vs- Labs

In lecture, we will primarily learn from COAD, supplemented from CAQA.

In lab, we will build a MIPS simulator from the ground up, and we will also spend about a week learning about GPU programming (CUDA).

Required Work

Homework sets (40%):

- Four assignments

Final Exam (20%)

- Take home exam, due 27 June.

Labs and Class Participation (40%)

- In-class labs, building the simulator and working on small GPU programs.

Class Survey

What do you look for when buying a computer?

Classes of Computers

Personal Mobile Device (PMD)

- e.g. start phones, tablet computers
- Emphasis on energy efficiency and real-time

Desktop Computing

- Emphasis on price-performance

Servers

- Emphasis on availability, scalability, throughput

Clusters / Warehouse Scale Computers

- Used for "Software as a Service (SaaS)"
- Emphasis on availability and price-performance
- Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks

Embedded Computers

- Emphasis: price

Class Survey

How can you make your programs run faster?

Class Survey

How can you make your programs run faster?

Possible answers:

- Better programming (different language, lower-level, better compiler, refactoring, etc.)
- Faster processor
- Multicore processors, leveraging parallelism
- More memory
- Faster disk (SSD…)
- Faster network

Current Trends in Architecture

Cannot continue to leverage Instruction-Level parallelism (ILP)

- Single processor performance improvement ended in 2003

New models for performance:

- Data-level parallelism (DLP)
- Thread-level parallelism (TLP)
- Request-level parallelism (RLP)

These require explicit restructuring of the application

Understanding Performance

Algorithm

- Determines number of operations executed

Programming language, compiler, architecture

- Determine number of machine instructions executed per operation

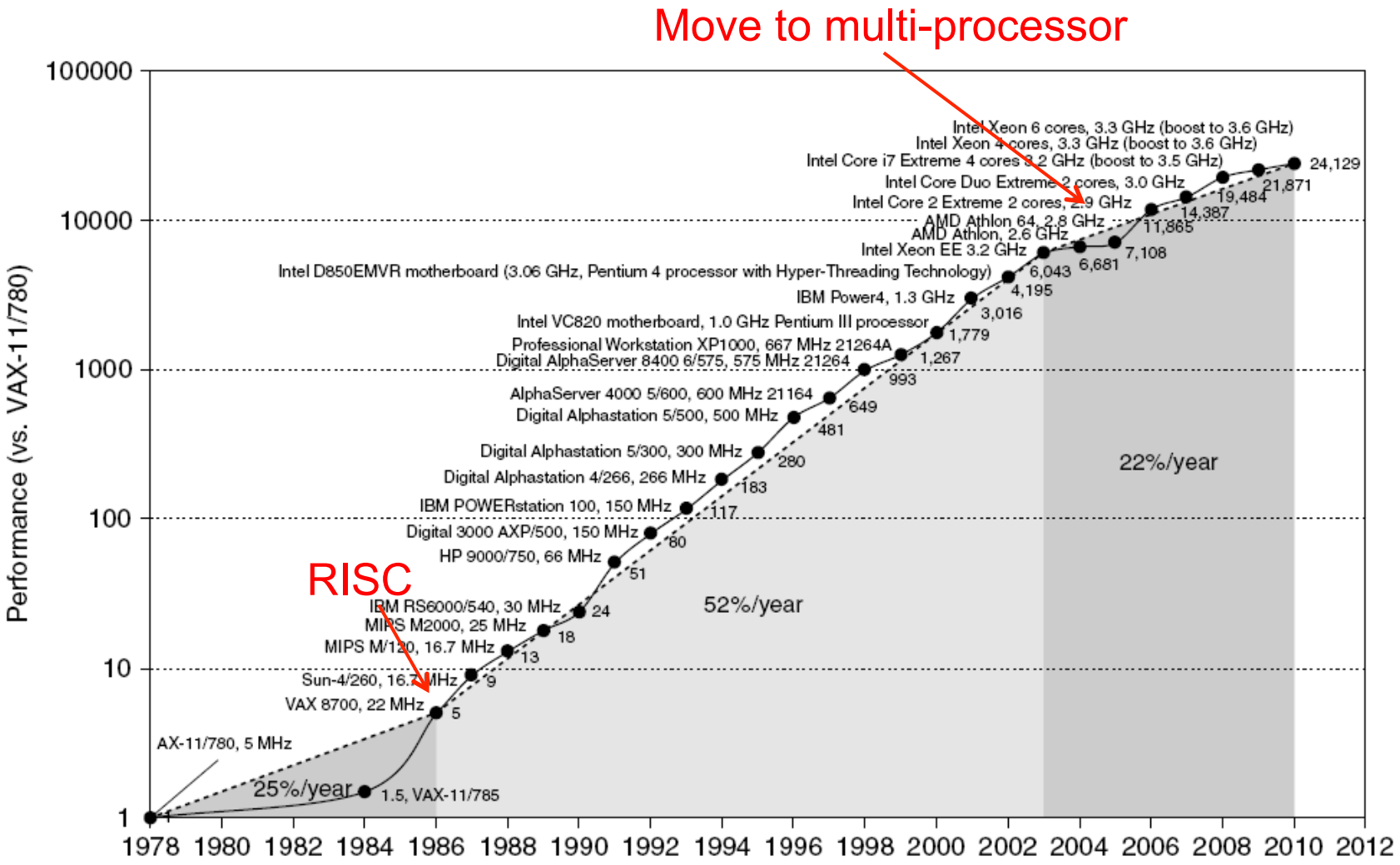
Processor and memory system

- Determine how fast instructions are executed

I/O system (including OS)

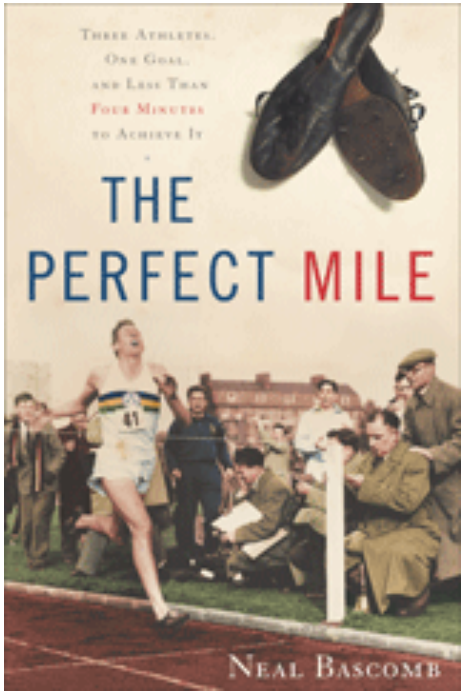
- Determines how fast I/O operations are executed

Single Processor Performance

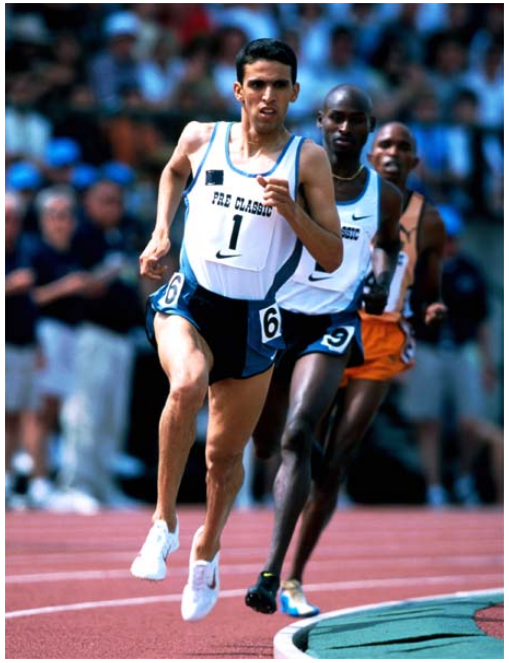


By Comparison: The Mile Run

Roger Bannister
1954 3:59.4

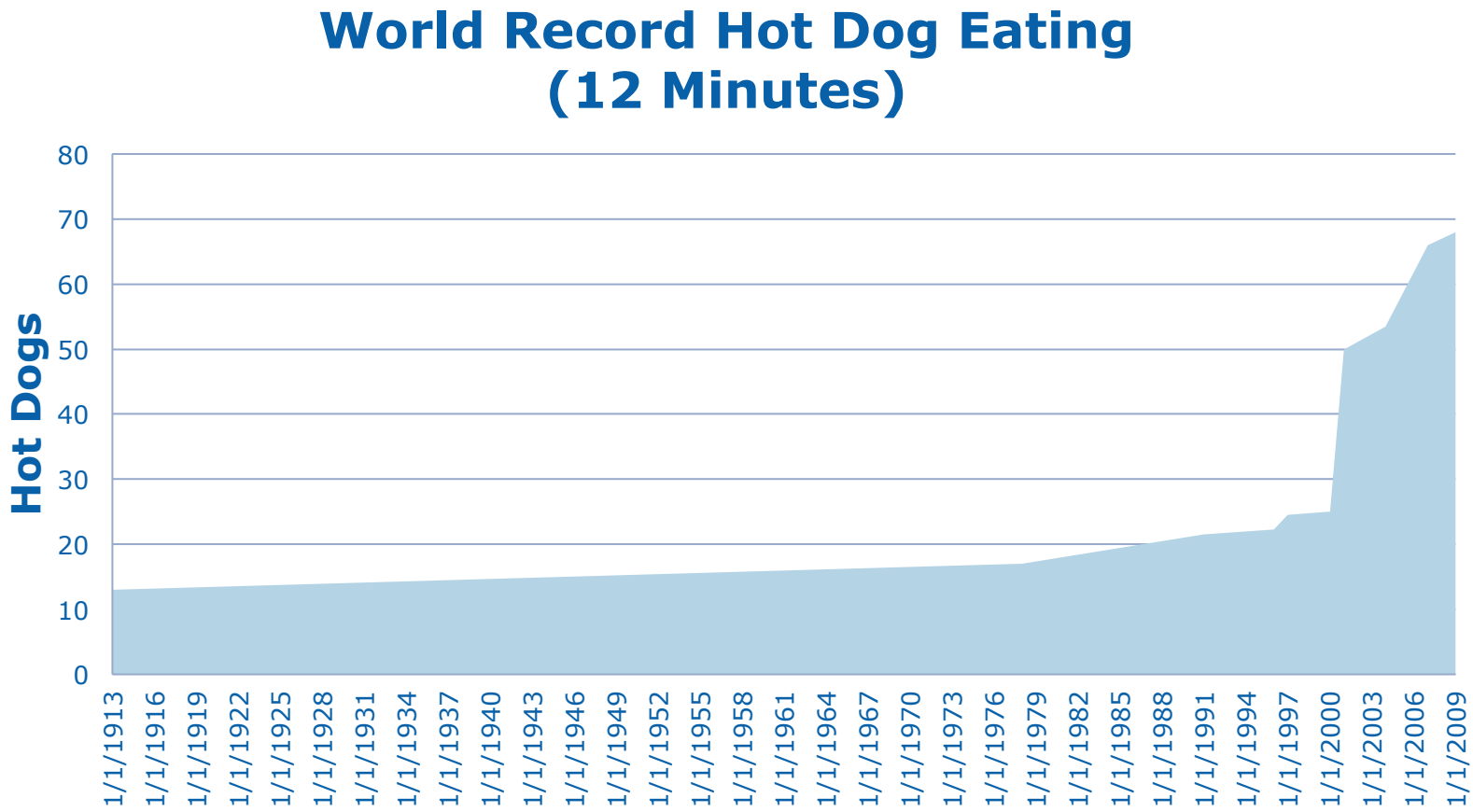


Hicham El Guerrouj
1999 3:43.13

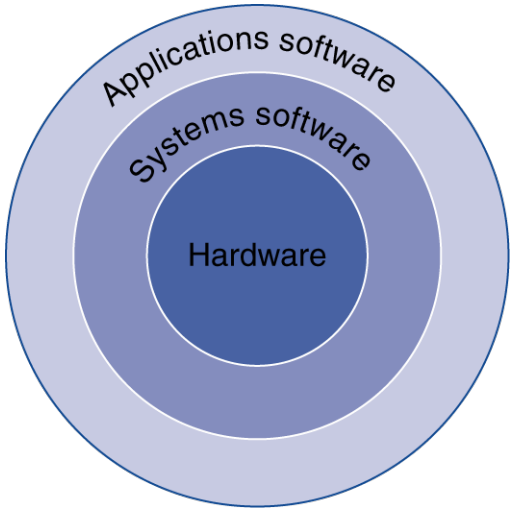


7% Improvement in 45 years

A Successful Endeavor



Below Your Program



Application software

- Written in high-level language

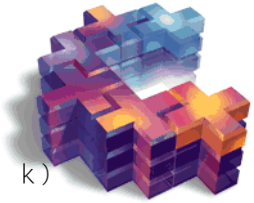
System software

- 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

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)

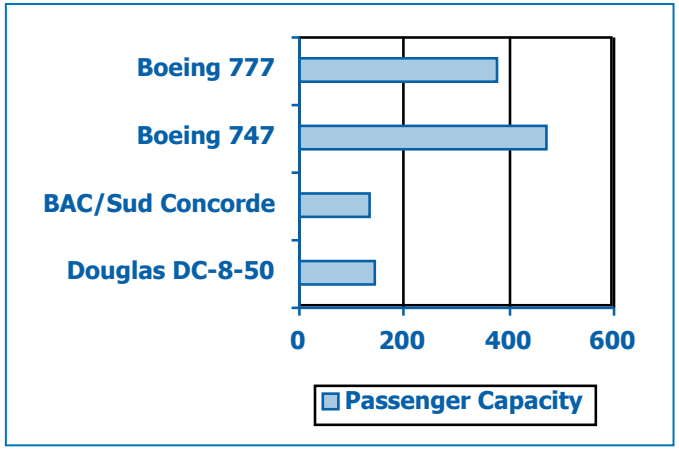
```
00000000101000010000000000011000
00000000000011000000110000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

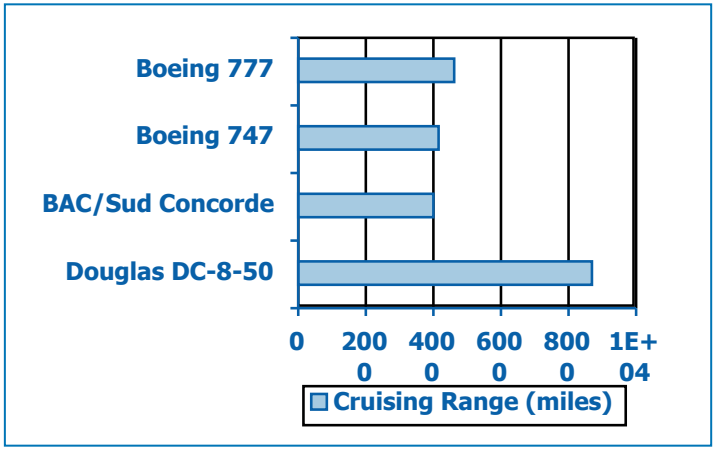
Class Survey

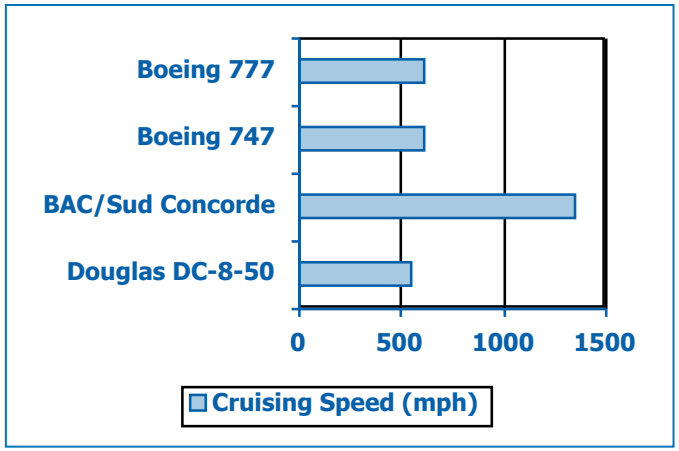
What does it mean to have better performance?

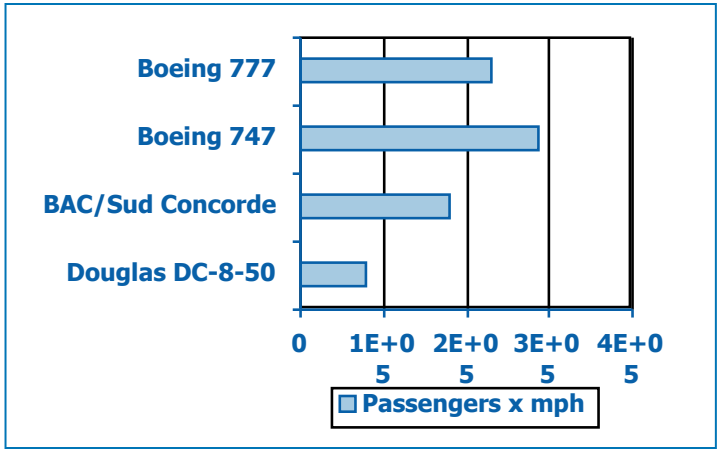
Defining Performance

Which airplane has the best performance?









Response Time and Throughput

Response time (execution time)

- How long it takes to do a task
  - Individual users want to reduce this

Throughput (bandwidth)

- Total work done per unit time
  - e.g., tasks/transactions/… per hour
  - Servers try to reduce this

How are response time and throughput affected

- By replacing the processor with a faster version?
- By adding more processors?

We'll focus on response time for now…

Relative Performance

- **Define Performance = 1/Execution Time**
- **"X is n times 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**
  - 20s on A, 30s on B
  - Execution Time$_B$ / Execution Time$_A$ = 30s / 20s = 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

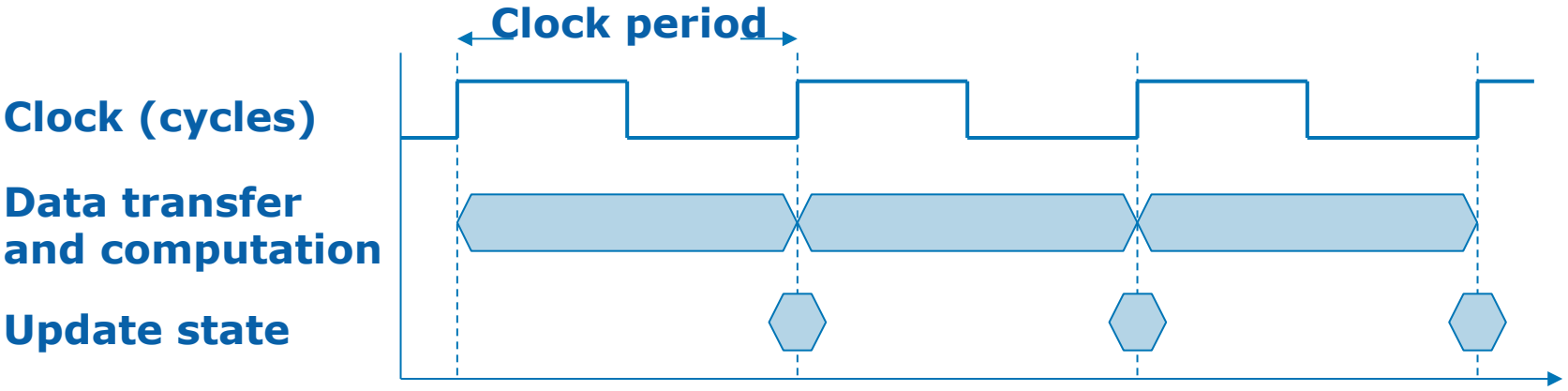
CPU time

- 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

Programs such as Linux "time" can give you some of this information.

CPU Clocking

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



- **Clock period (*T*): duration of a clock cycle**
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- **Clock frequency (*f*, rate): cycles per second**
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz
- ***f* = 1/*T***

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

CPU Time Example

Computer A: 2GHz clock, 20s CPU time

Designing Computer B

- Aim for 12s CPU time
- Can do faster clock, but causes 1.2 × clock cycles

How fast must Computer B's clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{12\text{s}}$$

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

$$= 20\text{s} \times 2\text{GHz} = 40 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 40 \times 10^9}{12\text{s}} = \frac{48 \times 10^9}{12\text{s}} = 4\text{GHz}$$

Instruction Count and CPI

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

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

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

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 = 300ps, CPI = 2.0

Computer B: Cycle Time = 600ps, CPI = 1.4

Same ISA

Which is faster, and by how much?

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

$$= I \times 2.0 \times 300\text{ps} = I \times 600\text{ps}$$

A is faster…

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

$$= I \times 1.4 \times 600\text{ps} = I \times 840\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 840\text{ps}}{I \times 600\text{ps}} = 1.4$$

…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 frequency

See book example, pg. 35

CPI Example

- **A compiler designer is considering two code sequences for a compiler.**
- **Alternative compiled code sequences using instructions in classes A, B, C. Average CPI?**

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	4	2	4
IC in sequence 2	8	2	2

- **Sequence 1: IC = 10**
  - Clock Cycles
    = 4×1 + 2×2 + 4×3
    = 20
  - Avg CPI= 20/10 = 2.0
- **Sequence 2: IC =12**
  - Clock Cycles
    = 8×1 + 2×2 + 2×3
    = 18
  - Avg CPI= 18/12 = 1.5

The Big Picture: Seconds per Program

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

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

Pitfall: Amdahl's Law

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

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

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

(See pg. 51 in COAD)

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

  - Can't be done!

- **Corollary: make the common case fast**

Benchmarking

1. Real applications
2. Modified applications
3. Kernels (small, critical parts of real applications)
4. Toy benchmarks
5. Synthetic benchmarks

SPEC CPU Benchmarks

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 Opteron X4 2356

Name	Description	IC×10^9	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.47	24	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.48	37	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
Geometric mean							11.7

High cache miss rates

Summary

Performance can be measured a number of ways

- **Know the ways, and the potential misconceptions**
- **Mostly boils down to the "standard performance equation"**

Class Survey

How does a computer *really* work?

Why design a processor simulator?

We will be designing a MIPS simulator in class.

If you really want to know how a computer works, you'd better understand it from the ground up.

We'll be discussing trade-offs that all computer architects need to make, and it is important to see those trade-offs at the hardware level.