Performance

Dr. Ergun Simsek (simsek@umbc.edu)

Lecture 2

PERFORMANCE

- *What is it?*
- *Why do we care?*
- *When we say one computer has better performance than another, what do we really mean?*

Does your code work faster on a MacBook Pro or an HP Spectre?


MacBook Pro or HP Spectre?

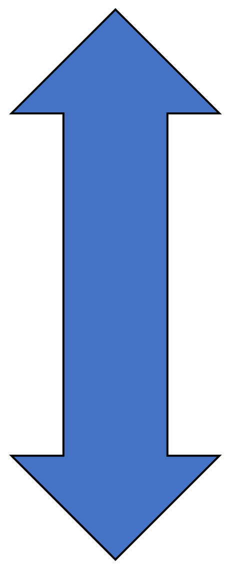
Levels of Transformation

"The purpose of computing is insight"

Richard Hamming

Problem
Algorithm
Program/Language
Runtime System (VM, OS, MM)
ISA (Architecture)
Microarchitecture
Logic
Electronics
Electrons

Sample Problem: Finding the optimum path between two addresses during rush hour



Abstraction *

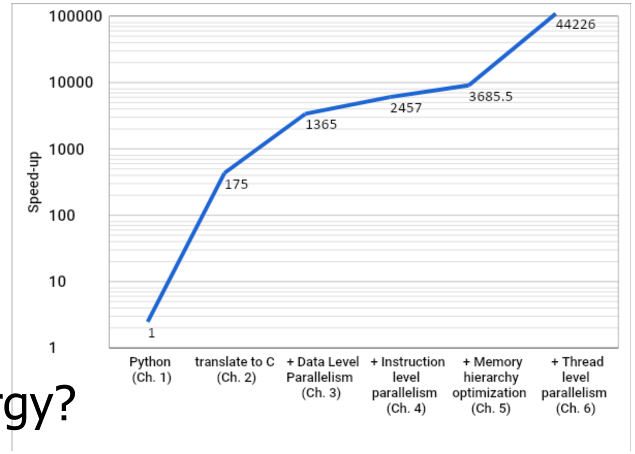
We solve this problem with electrons!

* A higher level only needs to know about the interface to the lower level, not how the lower level is implemented

If abstraction is great, then why would you want to know what goes on underneath or above?

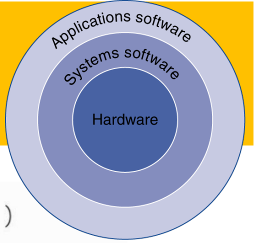
Crossing the Abstraction Layers

- As long as everything goes well, not knowing what happens in the underlying level (or above) is not a problem.



- What if
  - The program you wrote is running slow?
  - The program you wrote does not run correctly?
  - The program you wrote consumes too much energy?
- What if
  - The hardware you designed is too hard to program?
  - The hardware you designed is too slow because it does not provide the right primitives to the software?
- What if
  - You want to design a much more efficient and higher performance system?

Below Your Program



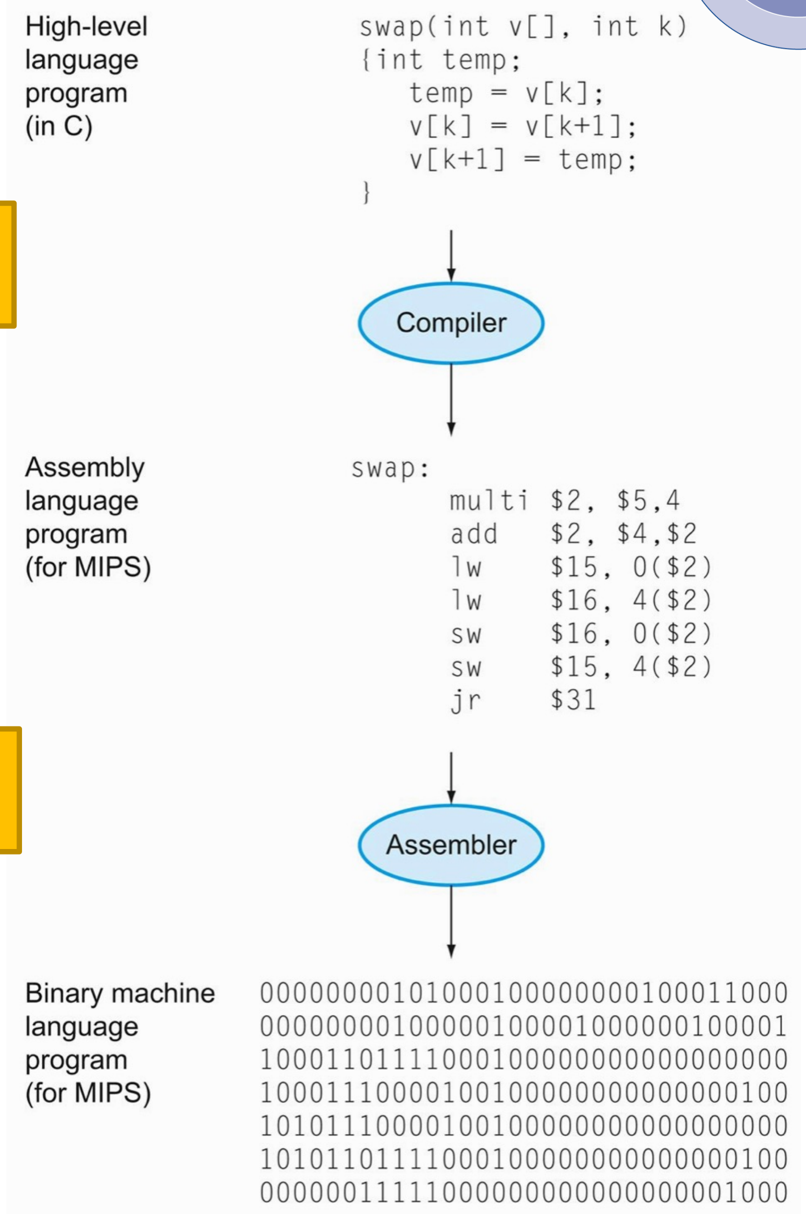
- A programmer writes a high-level language (HLL) program

> Can we write this function in a more efficient way? Should we use another language?

- A compiler converts the (HLL) program to an assembly language program

> Would another compiler or ISA lead to less # of inst.s? Is using 64-bits good idea?

- An assembler converts the assembly language program into a binary machine language program



This's Why We Care About Performance

- To make intelligent design choices
- Key to understanding underlying computer organization
  - Why is some hardware faster than others for different programs?
  - What factors of system performance are hardware related?

    e.g., Do we need a new machine or more RAM or a GPU

    … or a new operating system?
  - How does a machine's instruction set affect its performance?

Does your code work faster on a MacBook Pro or an HP Spectre?

- Algorithm
- Programming language, compiler, architecture
- Processor and memory system
- I/O system (including OS)

What is Performance?

- Loosely:
  - How fast can a computer complete a task
- Examples of "tasks":
  - Short tasks:
    - Crunch a bunch of numbers (say calculate mean)
    - Display a PDF document
    - Respond to a game console button press
  - Longer ones:
    - Video editing
    - Numerical optimization
    - Large-scale inversion

Our main parameter to measure performance will be TIME

Let's make some definitions

Execution Time vs Throughput

- Response time
or ~Execution time
or ~Latency

The total time required to complete a task

- Throughput
or Bandwidth

The number of tasks completed per unit time

Cars drive 60 km/h over a 1 km long bridge. A car thus requires 1 minute to cross the bridge. Cars stay separated by about 100 m, so 1 car enters and another exits the bridge every 6 seconds.

Task = Crossing the bridge
The execution time is 1 minute
The throughput is 10 cars/minute

Which airplane is the "best"?

Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (m.p.h.)	Passenger throughput (passengers × m.p.h.)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

- How much larger is the 747's capacity than the Concorde?
  - 3.6 X *("X" means "factor of")*
- How much faster is the Concorde than the 747?
  - 2.2 X
- It is roughly 4000 miles from Baltimore to Barcelona. If our goal is evacuating as many people as possible from Baltimore to Barcelona, which airplane would you use?

Boeing 747's throughput

$$470 \times \frac{610}{4000} = 71.7 \text{ passengers/hr}$$

Concorde's throughput

$$132 \times \frac{1350}{4000} = 44.6 \text{ passengers/hr}$$

Execution Time in Computing!

- Elapsed Time/Wall Clock Time
  - counts everything (disk and memory accesses, I/O, etc.)
  - includes the impact of other programs
  - a useful number, but often not good for comparison purposes

- CPU time
  - does not include I/O or time spent running other programs
  - can be broken up into system time and **user time**

- Our focus: user CPU time
  - time spent executing actual instructions of "our" program

Let's look at these definitions more closely

Time as a Measure of Computer Performance

The total time to complete a task (include everything) is called

Program execution time	=	Wall clock time	=	Response time	=	Elapsed time

The actual time the CPU spends computing for a specific task (include holds/waits at the CPU but not I/O) is called

CPU execution time	=	CPU time

Time as a Measure of Computer Performance

User CPU time	The CPU time spent in a program itself (does not include holds/waits or IO)
System CPU time	The CPU time spent in the operating system performing tasks on behalf of the program

On a RISC device with a single processor

Program execution time = System CPU time

Example

- ❑ A task runs alone on a CPU.
- ❑ The task starts by running for 5 ms.
- ❑ The task then waits for 4 ms while the operating system runs some instructions to access disk.
- ❑ The CPU is then idle for 2 ms while waiting for data from disk.
- ❑ Finally, the task runs another 10 ms and completes.

- Elapsed time = System performance **= 5 + 4 + 2 + 10 = 21 ms**
- CPU time **= 5 + 4 + 10 = 19 ms**
- User CPU time = CPU performance **= 5 + 10 = 15 ms**

Design Tradeoffs

Performance is rarely the sole factor.

- What are the other factors?
  - Cost
  - Energy/power consumption

- Frequently used compound metrics
  - Performance/Cost (throughput/$)
  - Performance/Power (throughput/watt)
  - Work/Energy (total work done per joule)
    - for battery-powered devices

Performance

- For some program running on machine X,

  Performance_X = Program Executions / Time_X (executions/sec)

  = 1 / Execution Time_X

- Relative Performance

  "X is *n* times faster than Y"

  Performance_X / Performance_Y = *n*

- Example:
  - Machine A runs a program in 20 seconds
  - Machine B runs the same program in 25 seconds
    - By how much is A faster than B?

$\text{Performance}_A = 1/20$ $\text{Performance}_B = 1/25$

$(1/20)/(1/25) = 1.25$

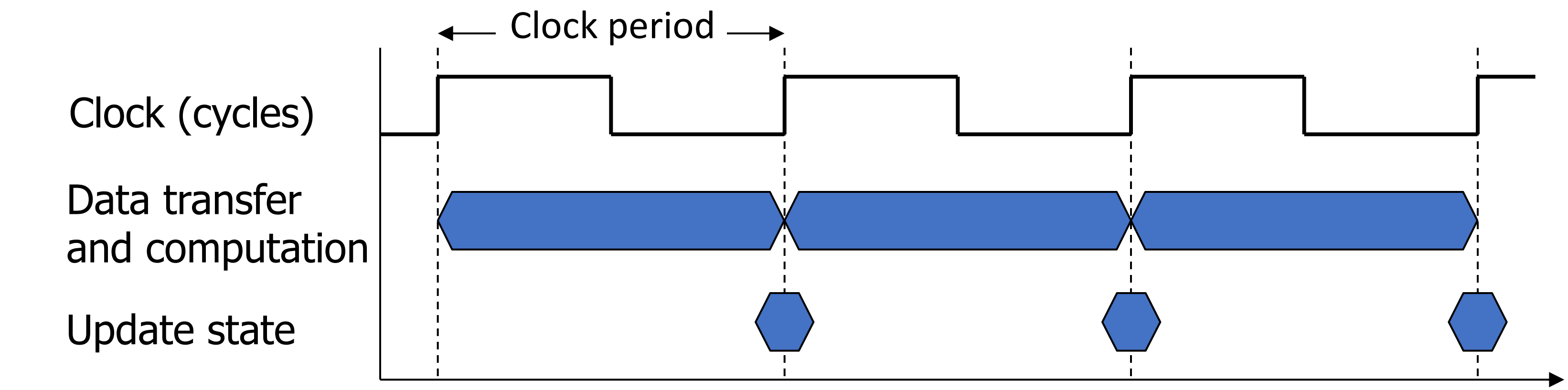
Machine A is 1.25 times faster than Machine B

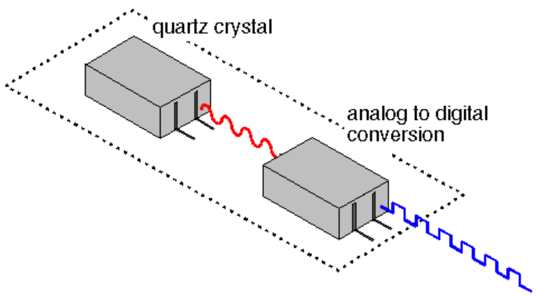
Performance: Pitfalls of using %

- Same Example:
  - By how much is B slower than A?
    - Correct answer is: B is $(Perf_B\text{-}Perf_A)/Per_A$ faster/slower
      - (0.04 - 0.05) / (0.05) = -20% faster = 20% slower
  - A is 25% faster than B; B is 20% slower
  - Some people find percentages confusing. We better use ratios.
  - Also: percentages are only good up to 100%
    - Don’t say A is 13000% faster than B

CPU Clocking

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

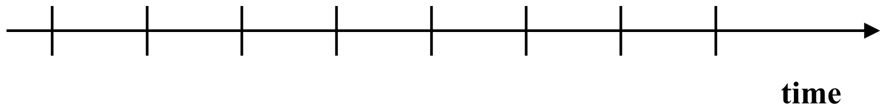




- Clock period: duration of a clock cycle
  - e.g., 270ps = 0.27ns = 270×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 1/(0.27ns) = 3.7GHz = 3700MHz = 3.7×10^{9}Hz

Program Clock Cycles

- Instead of reporting execution time in seconds, we often use clock cycle counts
  - Why? A newer generation of the same processor…
    - Often has the same cycle counts for the same program
    - But often has different clock speed (ex, 1 GHz changes to 1.5 GHz)
- Clock "ticks" indicate when machine state changes
  - an abstraction: allows time to be discrete instead of continuous



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

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

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

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
    - Instruction set architecture (ISA)
    - compiler
- Average cycles per instruction ("CPI")
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix