CS2100 COMPUTER ORGANISATION

http://www.comp.nus.edu.sg/~cs2100/

Lecture #24

Performance

NUS National University of Singapore | School of Computing

Lecture #24: Performance

1. Performance Definition
2. Factors Affecting Performance
3. Amdahl’s Law

1. Performance: Two Viewpoints

- "Computer X is faster than computer Y" is an ambiguous statement
    - "Fast" can be interpreted in different ways
- Fast = Response Time
    - The duration of a program execution is shorter
- Fast = Throughput
    - More work can be done in the same duration
- We focus on the first viewpoint in this section

1. Execution Time: Comparison

- Performance is in “units of things per second”
  - Bigger is better
- Response time is in “number of seconds”
  - Smaller is better

$$performance_x = \frac{1}{time_x}$$

- Speedup *n*, between *x* and *y* is

$$Speedup = \frac{time_x}{time_y} = \frac{performance_y}{performance_x}$$

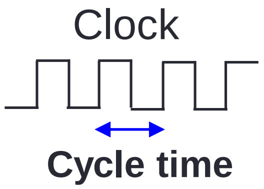
1. Execution Time: Refining Definition

- There are different measures of execution time in computer performance:
- **Elapsed time** (aka wall-clock time)
  - Counts everything (including disk and memory accesses, I/O, etc.)
  - Not so good for comparison purposes
- **CPU time**
  - Doesn’t 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 the lines of code in the program

1. Execution Time: Clock Cycles

- Instead of reporting execution time in seconds, we often use **clock cycles** (basic time unit in machine).

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



- **Cycle time** (or cycle period or clock period)
  - Time between two consecutive rising edges, measured in seconds.

- **Clock rate** (or clock frequency)

  = 1 / cycle-time

  = number-of-cycles/second (unit is Hz; 1 Hz = 1 cycle/second)

1. Execution Time: Version 1.0

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

- Therefore, to improve performance (everything else being equal), you can do the following:
  - ↓ Reduce the number of cycles for a program, or
  - ↓ Reduce the clock cycle time, or equivalently,
  - ↑ Increase the clock rate

Exercise 1: Clock Cycle & Clock Rate

- Program P runs in 10 seconds on machine A, which has a 400 MHz clock.
- Suppose we are trying to build a new machine B that will run this program in 6 seconds. Unfortunately, the increase in clock rate has an averse effect on 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 target at to hit our goal?
- Answer:

Let C be the number of clock cycles required for that program.

For A: Time = 10 sec. = C × 1/400MHz

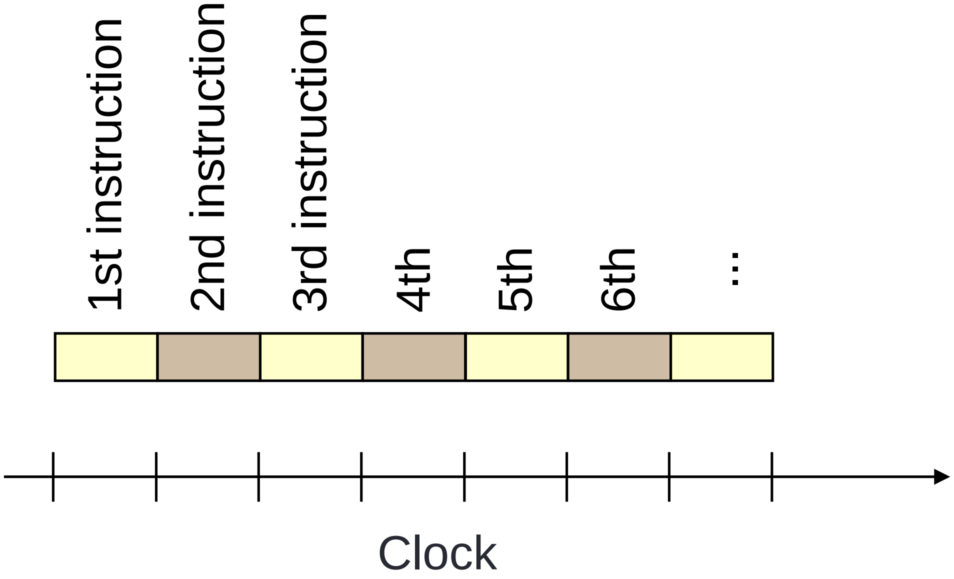
For B: Time = 6 sec. = (1.2 × C) × 1/***clock_rateB***

Therefore, ***clock_rateB*** = 10 × 400 × 1.2/6 MHz **= 800 MHz**

1. Clock Cycles: Proportional?

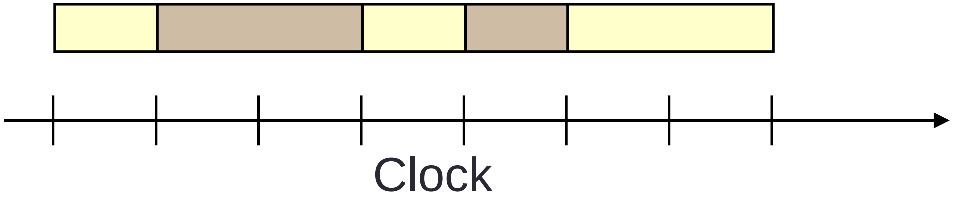
- Each instruction takes *n* clock cycles
- An execution consists of *I* number of instructions
- Does it mean we need *I* × *n* cycles to finish it?

→ Number of cycles is proportional to number of instructions?



1. Clock Cycles: Inst Type Dependent

- Different instructions take different amount of time to finish:



- For example:
  - ***Multiply*** instruction may take longer than an ***Add*** instruction.
  - ***Floating-point*** operations take longer than ***integer*** operations.
  - Accessing ***memory*** takes more time than accessing ***registers***

1. Execution Time: Introducing CPI

- A given program will require

Some number of instructions (machine instructions)

↓ × **average Cycle per Instruction (CPI)**

Some number of cycles

↓ × **cycle time**

Some number of seconds

- We use the *average* CPI as different instructions take different number of cycles to finish

1. Execution Time: Version 2.0

- Average Cycle Per Instruction (CPI)

CPI = (CPU time × Clock rate) / Instruction count

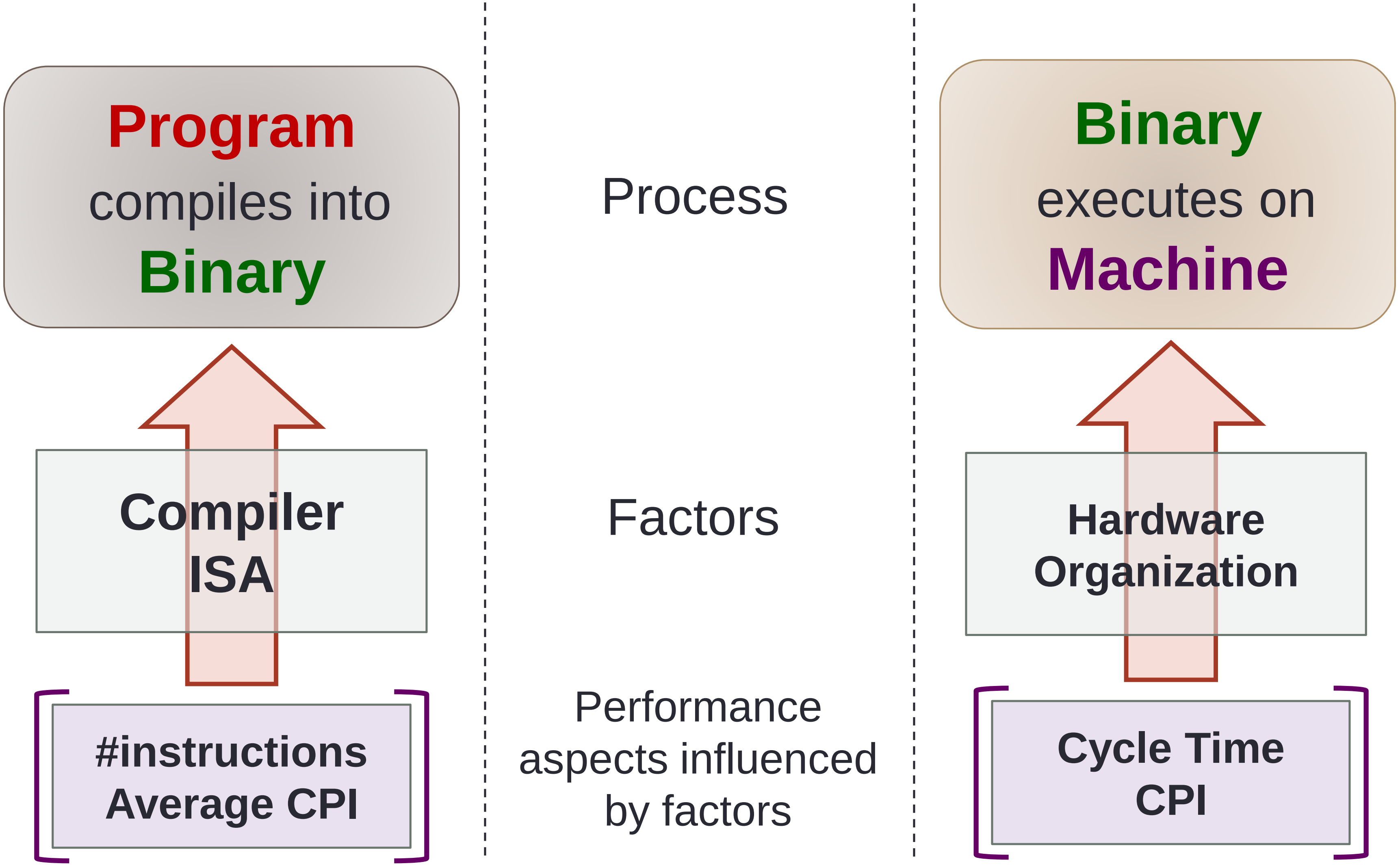
= Clock cycles / Instruction count

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \text{ x } \frac{\text{Cycles}}{\text{Instruction}} \text{ x } \frac{\text{Seconds}}{\text{Cycle}}$$

$$CPI = \sum_{k=1}^{n} (CPI_k \times F_k) \quad \text{where} \quad F_k = \frac{I_k}{\text{Instruction count}}$$

$$I_k = \text{Instruction frequency}$$

2. Performance: Influencing Factors



Program compiles into **Binary**

Process

Binary executes on **Machine**

Compiler
ISA

Factors

Hardware
Organization

#instructions
Average CPI

Performance
aspects influenced
by factors

Cycle Time
CPI

2. Program → Binary: Factors

- **Compiler:**
  - Different compilers may generate different binary codes
    - e.g. gnu vs intel c/c++ compiler
  - Different optimization may generate different binary codes
    - e.g. different optimization level in ***gnu c compiler***
- **Instruction Set Architecture:**
  - The same high level statement can be translated differently depending on the ISA
    - e.g. same C program under ***Intel*** machine vs ***Sunfire*** server

Exercise 2: Impact of Compiler

- Given a program P, a compiler can generate 2 different binaries on a target machine. On that machine, there are 3 classes of instructions: Class A, Class B, and Class C, and they require 1, 2, and 3 cycles respectively.
- First binary has 5 instructions: **2 of A, 1 of B, and 2 of C**.
  Second binary has 6 instructions: **4 of A, 1 of B, and 1 of C**.

1. Which code is faster? By how much?
2. What is the (average) CPI for each code?

- Answer:

  Let T be the cycle time.

  Time(code1) = (2×1 + 1×2 + 2×3) × T = 10T

  Time(code2) = (4×1 + 1×2 + 1×3) × T = 9T

  Time(code1)/Time(code2) = 10/9 = 1.1 → Code2 is 1.1 times faster

  CPI(code1) = (2×1 + 1×2 + 2×3) / (2 + 1 + 2) = **2**

  CPI(code2) = (4×1 + 1×2 + 1×3) / (4+ 1 + 1) = **1.5**

2. Execution of Binary Code: Factors

- **Machine**
  - More accurately the hardware implementation
  - Determine **cycle time** and **cycle per instruction**
- **Cycle Time**:
  - Different clock frequencies (e.g. 2GHz vs 3.6GHz)
- **Cycles Per Instruction:**
  - Design of internal mechanism (e.g. specific accelerator to improve floating point performance)

Exercise 3: Impact of Machine

- Suppose we have 2 implementations of the same ISA, and a program is run on these 2 machines.
- Machine A has a clock cycle time of 10 ns and a CPI of 2.0. Machine B has a clock cycle time of 20 ns and a CPI of 1.2.

1. Which machine is faster for this program? By how much?

- Answer:

Let N be the number of instructions.

Machine A: Time = N × 2.0 × 10 ns

Machine B: Time = N × 1.2 × 20 ns

Performance(A)/Performance(B) = Time(B)/Time(A)

= (1.2 × 20) / (2.0 × 10) = **1.2**

Exercise 4: All Factors (1/4)

- You are given 2 machine designs M1 and M2 for performance benchmarking. Both M1 and M2 have the same ISA, but different hardware implementations and compilers. Assuming that the clock cycle times for M1 and M2 are the same, performance study gives the following measurements for the 2 designs.

Instruction class	For M1		For M2	
	CPI	No. of instructions executed	CPI	No. of instructions executed
A	1	3,000,000,000,000	2	2,700,000,000,000
B	2	2,000,000,000,000	3	1,800,000,000,000
C	3	2,000,000,000,000	3	1,800,000,000,000
D	4	1,000,000,000,000	2	900,000,000,000

Exercise 4: All Factors (2/4)

a) What is the CPI for each machine?

Let Y = 1,000,000,000,000

CPI(M1) = (3Y×1 + 2Y×2 + 2Y×3 + Y×4) / (3Y + 2Y + 2Y + Y)
= 17Y / 8Y = **2.125**

CPI(M2) = [(3Y×2 + 2Y×3 + 2Y×3 + Y×2) × 0.9] / [(3Y + 2Y + 2Y + Y) × 0.9]
= 20Y / 8Y = **2.5**

b) Which machine is faster? By how much?

Let C be cycle time.

Time(M1) = 2.125 × (8Y × C)
Time(M2) = 2.5 × (8Y × 0.9 × C) = 2.25 × (8Y × C)

M1 is faster than M2 by 2.25 / 2.125 = **1.0588**

Exercise 4: All Factors (3/4)

c) To further improve the performance of the machines, a new compiler technique is introduced. The compiler can simply eliminate all class D instructions from the benchmark program without any side effects. (That is, there is no change to the number of class A, B and C instructions executed in the 2 machines.) With this new technique, which machine is faster? By how much?

Let Y = 1,000,000,000,000; Let C be cycle time.

CPI(M1) = $(3Y\times1 + 2Y\times2 + 2Y\times3) / (3Y + 2Y + 2Y) = 13Y / 7Y = 1.857$

CPI(M2) = $[(3Y\times2 + 2Y\times3 + 2Y\times3) \times 0.9] / [(3Y + 2Y + 2Y) \times 0.9]$
= $18Y / 7Y = 2.571$

Time(M1) = $1.857 \times (7Y \times C)$

Time(M2) = $2.571 \times (0.9 \times 7Y \times C) = 2.314 \times (7Y \times C)$

M1 is faster than M2 by 2.314 / 1.857 = **1.246**

Exercise 4: All Factors (4/4)

d) Alternatively, to further improve the performance of the machines, a new hardware technique is introduced. The hardware can simply execute all class D instructions in zero times without any side effects. (There is still execution for class D instructions.) With this new technique, which machine is faster? By how much?

Let Y = 1,000,000,000,000; Let C be cycle time.

CPI(M1) = (3Y×1 + 2Y×2 + 2Y×3 + Y×0) / (3Y + 2Y + 2Y + Y)
= 13Y / 8Y = 1.625

CPI(M2) = [(3Y×2 + 2Y×3 + 2Y×3 + Y×0) × 0.9] / [(3Y + 2Y + 2Y + Y) × 0.9]
= 18Y / 8Y = 2.25

Time(M1) = 1.625 × (8Y × C)

Time(M2) = 2.25 × (0.9 × 8Y × C) = 2.025 × (8Y × C)

M1 is faster than M2 by 2.025 / 1.625 = **1.246**

Summary: Key Concepts

- Performance is specific to a particular program on a specific machine
  - Total execution time is a consistent summary of performance
- For a given architecture, performance increase comes from:
  - Increase in clock rate (without adverse CPI effects)
  - Improvement in processor organization that lowers CPI
  - Compiler enhancement that lowers CPI and/or instruction count

Pitfall:
Expecting improvement in one aspect of a machine’s performance to affect the total performance.

3. Amdahl's Law (1/3)

- **Pitfall:** Expecting the improvement of one aspect of a machine to increase performance by an amount proportional to the size of the improvement.
- Example:
  - Suppose a program runs in 100 seconds on a machine, with multiply operations 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?

100 (total time) = 80 (for multiply) + UA (unaffected)

➔ UA = 20

100/4 (new total time) = 80/Speedup (for multiply) + UA

➔Speedup = 80/5 = **16**

3. Amdahl's Law (2/3)

- Example (continued):
  - How about making it 5 times faster?

100 (total time) = 80 (for multiply) + UA (unaffected)

100/5 (new total time) = 80/Speedup (for multiply) + UA

➔Speedup = 80/0 = ??? (impossible!)

- There is no way we can enhance multiply to achieve a fivefold increase in performance, if multiply accounts for only 80% of the workload.

3. Amdahl's Law (3/3)

- **Amdahl's law:**
  - Performance is limited to the non-speedup portion of the program

- Execution time after improvement

  = Execution time of unaffected part

  + (Execution time of affected part / Speedup)

- **Corollary of Amdahl's law:**
  - Optimize the common case first!

Exercise 5: Amdahl's Law

- 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 **12 seconds**, what will the speedup be if **half of the 12 seconds** is spent executing floating-point instructions?

Time = 6 (UA) + 6 (fl-pt) / 5 = 7.2 sec.

Speedup = 12 / 7.2 = **1.67**

Exercise 6: Amdahl's Law

- We are looking for a benchmark to show off the **new floating-point unit** described in the previous example, and we 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?

Speedup = 3 = 100 / (Time_fl / 5 + 100 – Time_fl)

Time_Fl = **83.33 sec.**

End of File