# CSE3421
# Computer Architecture

# Performance metrics

Xiaodong Zhang

## What is Performance?

- Three ways to measure a computer system
  - Response time: the time between the start and completion of a task (often measured by a user)
    - In economics: completion time for a product (e.g. the assembly line of Ford in 1913)
  - Throughput: number of tasks completed in a time unit
    - In economics: productivity (related to efficiency)
  - Resource consumption: resource used to complete task
    - For example, a kernel context switch uses 1348 CPU cycles on a 2.86GHz P4 processor
    - In economics: cost or expense of production (related to investment)
- Why do we need these metrics?

## Performance Metrics

- Provide a basis for comparison and a metric for evaluation
- Purchasing perspective
  - given a collection of machines, which has the
    - best performance ?
    - least cost ?
    - best cost/performance?
- Design perspective
  - faced with design options, which has the
    - best performance improvement ?
    - best cost/performance?
    - A planning question: *Should Intel design chips with 32 "wimpy" (500 MHz) cores or 4 "beefy" (3GHz) cores?*

- Our goal is to understand what factors in the architecture contribute to overall system performance and the relative importance (and cost) of these factors

## Performance questions

- How do the following changes affect response time, throughput, and power consumption?
  - Increasing the processor clock rate by 2X for an application that calculates derivatives
    - CPU Throughput and Power up, Response time down
  - Adding an additional processor for the derivative calculator
    - CPU Response time, Throughput and Power up
  - Implementing a faster algorithm to compute derivatives
    - Throughput up, CPU Power and Response time down
  - Adding an additional processor for a web crawler
    - Response Time and Throughput and CPU Power up
  - **Efficient algorithms design is most powerful**
- Why additional resources degrade performance?
  - Cost of process coordination, context switch, low parallelism…

## Defining (Speed) Performance

- Maximizing performance => to minimize execution time

$$\text{performance}_X = 1 / \text{execution\_time}_X$$

**If X is n times faster than Y, then**

$$\frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution\_time}_Y}{\text{execution\_time}_X} = n$$

- Decreasing response time almost always improves throughput

## A Relative Performance Example

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

## A Relative Performance Example

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

We know that A is n times faster than B if

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = n$$

The performance ratio is $$\frac{15}{10} = 1.5$$

So A is 1.5 times faster than B

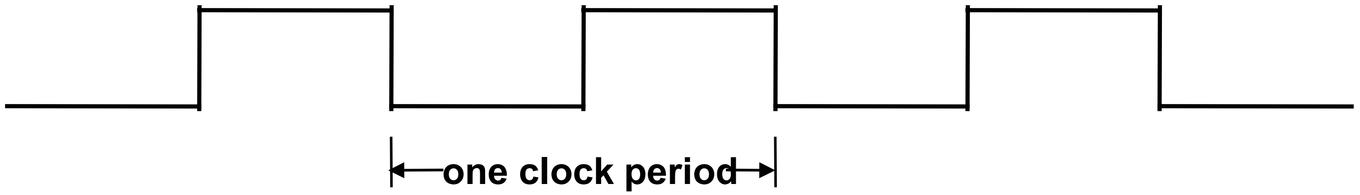
## Time units and data size units

- Time units in the computing world
  - 1 millisecond (ms) = $10^{-3}$ second
  - 1 microsecond (us) = $10^{-6}$ second
  - 1 nanosecond (ns) = $10^{-9}$ second
  - 1 picosecond (ps) = $10^{-12}$ second
  - 1 femtosecond (fs) = $10^{-15}$ second
  - 1 attosecond (as) = $10^{-18}$ second
- Data size units in the computing world
  - 1 kilobyte (KB) = $10^{3}$ bytes
  - 1 megabytes (MB) = $10^{6}$ bytes
  - 1 gigabytes (GB) = $10^{9}$ bytes
  - 1 terabytes (TB) = $10^{12}$ bytes
  - 1 petabytes (PB) = $10^{15}$ bytes
  - 1 exabytes (EB) = $10^{18}$ bytes
  - 1 zettabyte (ZB) = $10^{21}$ bytes
  - 1 yottabyte (YB) =$10^{24}$ bytes

## Basic Computing Unit: Clock Rate and Cycle Time

- Clock rate (clock cycles per second in MHz or GHz) is inverse of clock cycle time (clock period)

$$CC = 1 / CR$$



10 nsec clock cycle = $10*10^{-9} = 10^{-8}$ sec => 1/CC = 100 MHz ($10^6$) clock rate

2 nsec clock cycle => 500 MHz clock rate

1 nsec ($10^{-9}$) clock cycle => 1 GHz ($10^9$) clock rate

500 psec clock cycle = $500*10^{-12}$ sec => 1/CC = 2 GHz clock rate

250 psec clock cycle => 4 GHz clock rate

Increasing the number of transistors on chip, reducing the cycle time, Why?

the higher transistor density, the lower interconnection delay

## The Core Model of CPU Execution Time

- CPU execution time (or CPU time): time the CPU spends working on a task
  - Does not include time waiting for I/O or running other programs

$$\text{CPU execution time for a program} = \#\text{ CPU clock cycles for a program} \times \text{clock cycle time}$$

or

$$\text{CPU execution time for a program} = \frac{\#\text{ CPU clock cycles for a program}}{\text{clock rate}}$$

- Can improve performance by reducing either the length of the clock cycle or the number of clock cycles required for a program

## Improving Performance Example

- A program runs on computer A with a 2 GHz clock in 10 seconds. What clock rate must computer B run at to run this program in 6 seconds? Unfortunately, to accomplish this, computer B will require an additional 1.2 times as many clock cycles.

$$\text{CPU time}_A = \frac{\text{\# CPU clock cycles}_A}{\text{clock rate}_A}$$

$$\text{\# CPU clock cycles}_A = 10 \text{ sec} \times 2 \times 10^9 \text{ cycles/sec}$$
$$= 20 \times 10^9 \text{ cycles}$$

$$\text{CPU time}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{\text{clock rate}_B}$$

$$\text{clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = 4 \text{ GHz}$$

Why Computer B needs more cycles?

* Due to different architecture design

## Improving Performance Example

- Computer C has a clock rate of 3 GHz and will require only 1.05 as many clock cycles as computer A. How long will the program take?

$$\text{CPU time}_A = \frac{\#\text{ CPU clock cycles}_A}{\text{clock rate}_A}$$

$$\#\text{ CPU clock cycles}_A = 10 \text{ sec} \times 2 \times 10^9 \text{ cycles/sec} = 20 \times 10^9 \text{ cycles}$$

$$\text{CPU time}_C = \frac{1.05 \times 20 \times 10^9 \text{ cycles}}{\text{clock rate}_C}$$

$$= \frac{21 \times 10^9 \text{ cycles}}{3 \times 10^9 \text{ cycles/sec}} = 7 \text{ seconds}$$

## Extending the Model: Clock Cycles per Instruction

- Not all instructions take the same amount of time to execute
  - One way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction

# CPU clock cycles for a program = # Instructions for a program x Average clock cycles per instruction

- Clock cycles per instruction (CPI) – the average number of clock cycles each instruction takes to execute
  - A way to compare two different implementations of the same ISA

| | CPI for this instruction class | | |
|---|---|---|---|
| | A | B | C |
| CPI (#cycles) | 1 | 2 | 3 |

## Effective (Average) CPI

- Computing the overall effective CPI is done by looking at the different types of instructions and their individual cycle counts and averaging

$$\text{Overall effective CPI} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{IC}_i)$$

  - Where $\text{IC}_i$ is the percentage of instructions of class i executed
  - $\text{CPI}_i$ is the (average) number of clock cycles per instruction for that instruction class
  - n is the number of instruction classes

- The overall effective CPI varies by instruction mix – a measure of the dynamic frequency of instructions across one or many programs

What similar formula do you use all the time?

$$\text{GPA} = \sum_{i=1}^{n} (\text{Grade}_i \times \%\text{CH}_i)$$

## Comparing CPI

- Performance analysis rule: Conclusive experiments vary according to only 1 factor: CPI is machine dependent.
  - Can we compare the CPI of loading Firefox on Intel P4 Northwood to loading Firefox on a Sun Niagara? What does it tell us?

    No, we cannot compare CPIs between architectures

  - Can we compare the CPI of loading Firefox to the CPI of loading Internet Explorer on the same machine?

    Yes.

## Comparing CPI in the same architecture

- Computers A and B share the same architecture design. Computer A has a clock cycle time of 250 ps (picosecond, $10^{-12}$ ) and an effective CPI of 2.0 for a program and computer B has a clock cycle time of 500 ps and an effective CPI of 1.2 for the same program. Which computer is faster and by how much?

Each computer executes the same number of instructions, *I*, so

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

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

Clearly, A is faster … by the ratio of execution times

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

## The Performance Equation

- Our basic performance equation is then

$$\text{CPU time} = \text{Instruction\_count} \times \text{CPI} \times \text{clock\_cycle}$$

or

$$\text{CPU time} = \frac{\text{Instruction\_count} \times \text{CPI}}{\text{clock\_rate}}$$

- These equations separate the three key factors that affect performance
  - Can measure the CPU execution time by running the program
  - The clock rate is usually given
  - Can measure overall instruction count by using profilers/ simulators without knowing all of the implementation details
  - CPI varies by instruction type and hardware design for which we must know the implementation details

## Compiler Benefits in Today's Processor

- Comparing performance for bubble (exchange) sort
  - To sort 100,000 words with the array initialized to random values on a Pentium 4 with a 3.06 GHz clock rate, with 2 GB DRAM memory, using Linux version 2.4.20

| gcc opt | Relative performance | Clock cycles (M) | Instr count (M) | CPI |
|---|---|---|---|---|
| None | 1.00 | 158,615 | 114,938 | **1.38** |
| O1 (medium) | 2.37 | 66,990 | **37,470** | 1.79 |
| O2 (full) | 2.38 | 66,521 | 39,993 | 1.66 |
| O3 (proc mig) | 2.41 | **65,747** | 44,993 | 1.46 |

- The unoptimized code has the best CPI, the O1 version has the lowest instruction count, but the O3 version offers the best mix, O2 is ranked second.
- The execution time represents the final performance

## What does a compiler do for optimization?

- Eliminate redundant instructions
- Reordering the sequence of instructions
- Selecting effective instructions (not necessarily low CPIs)
- Grouping data accesses to reuse the cache
- …

## Determinates of CPU Performance

CPU time = Instruction_count x CPI x cycle_time

| | Instruction _count | CPI | clock_cycle |
|---|---|---|---|
| Algorithm | X | X | |
| Programming language | X | X | |
| Compiler | X | X | |
| Instruction Set | X | X | |
| Organization (pipeline, etc.) | | X | X |
| Chip technology | | | X |

## A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ |
|---|---|---|---|
| ALU | 50% | 1 | . |
| Load | 20% | 5 | |
| Store | 10% | 3 | |
| Branch | 20% | 2 | |
| | | | $\Sigma =$ |

How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

How does this compare with using branch prediction to shave a cycle off the branch time?

What if two ALU instructions could be executed at once?

## A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ | | | |
|---|---|---|---|---|---|---|
| ALU | 50% | 1 | .5 | .5 | .5 | **.25** |
| Load | 20% | 5 | 1.0 | **.4** | 1.0 | 1.0 |
| Store | 10% | 3 | .3 | .3 | .3 | .3 |
| Branch | 20% | 2 | .4 | .4 | **.2** | .4 |
| | | | $\sum =$ 2.2 | **1.6** | **2.0** | **1.95** |

How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

CPU time new = **1.6** x IC x CC so 2.2/1.6 means 37.5% faster

How does this compare with using branch prediction to shave a cycle off the branch time?

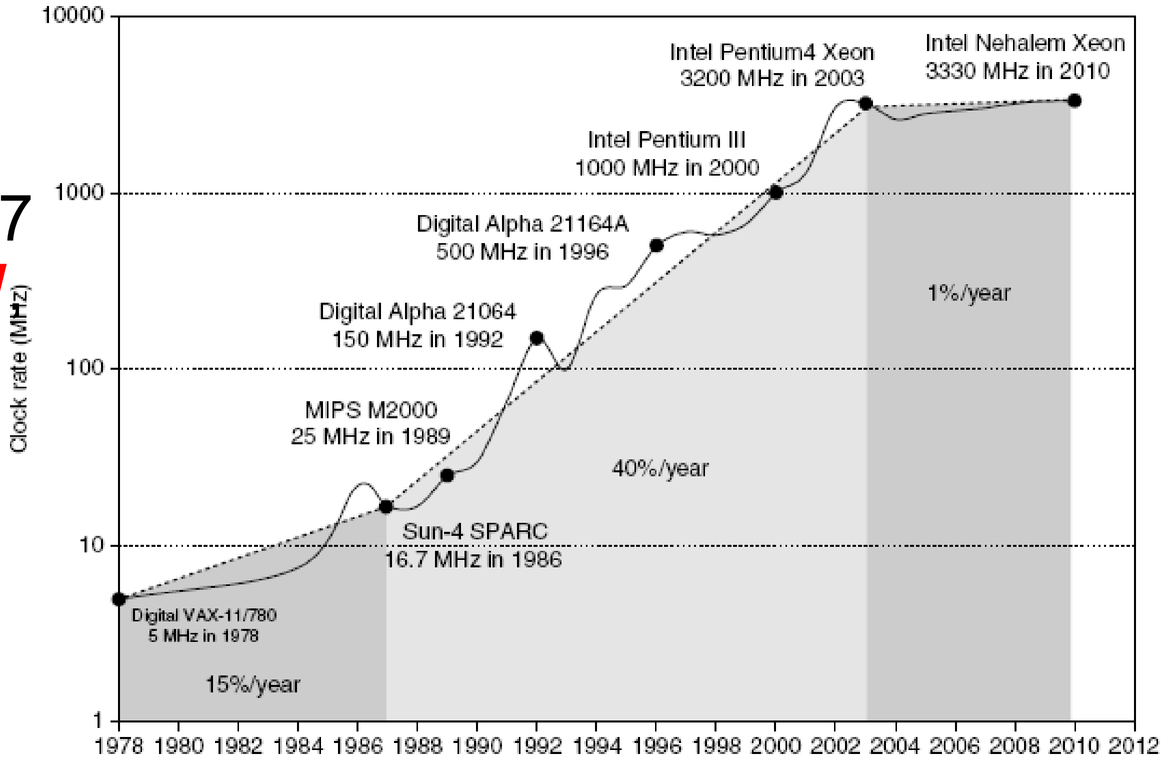
CPU time new = **2.0** x IC x CC so 2.2/2.0 means 10% faster

What if two ALU instructions could be executed at once?

CPU time new = **1.95** x IC x CC so 2.2/1.95 means 12.8% faster

## Power

- Intel 80386 (1985)
  12-40 MHz,
  consumed ~ **2 W**
- 3.3 GHz Intel Core i7
  consumed **130 W**
  2009
- Heat must be
  dissipated from
  chips
- This is the limit of
  what can be
  cooled by air



## Reducing Power and/or Energy

Techniques for reducing power:

- Dynamic Voltage-Frequency Scaling (adjustment)
- Low power state for DRAM, disks (sleeping mode)
- Over-clocking – increasing clock rate of a computing unit, but turning off other computing units (cores) for low power and high efficiency

## Performance beyond formulas

Linear (proportional) relationships on the same CPU running the same number of instructions:

- Overall effective CPI and execution time (the lower CPI, the lower execution time)
- # instructions / second and the execution performance (the higher #I/s, the higher perf)
- Execution time is determined by weights (percentages) of different classes of instructions, such as data-intensive (high percentages of Load/Store instructions), computing-intensive (high percentages of ALU instructions)