# CSE305 Computer Architecture

## Performance

**Daehoon Kim**
**Department of EECS, DGIST**

# Defining (Speed) Performance

- **To maximize performance, need to minimize execution time**

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

If X is n times faster than Y, then

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

# 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

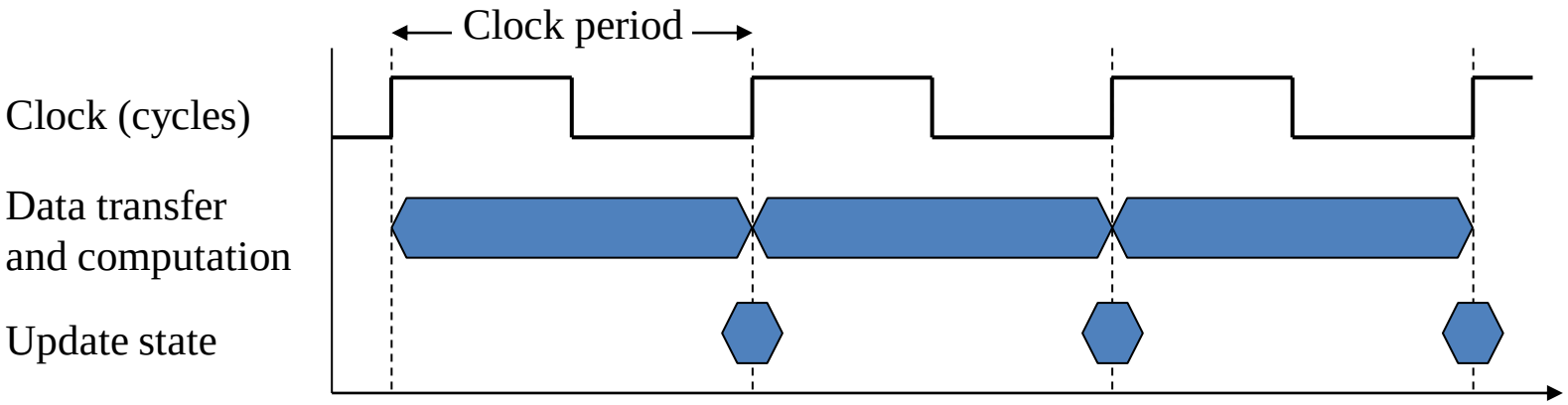
$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = n$$

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

So A is 1.5 times faster than B (i.e., speedup is 1.5)

# CPU Clocking

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

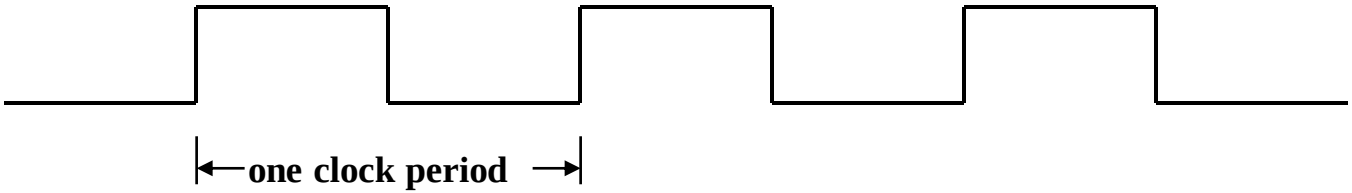



  - Clock frequency (rate): cycles per second
    - e.g., 4.0GHz = 4000MHz = $4.0 \times 10^{9}$Hz
  - Clock period (clock cycle time): duration of a clock cycle
    - e.g., 250ps = 0.25ns = $250 \times 10^{-12}$s

# Machine Clock Rate

- **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 => 100 MHz clock rate

5 nsec clock cycle => 200 MHz 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 => 2 GHz clock rate

250 psec clock cycle => 4 GHz clock rate

200 psec clock cycle => 5 GHz clock rate

# Performance Factors

- **CPU execution time (CPU time) : time the CPU spends working on a task**

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

or

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

- **How to improve performance?**
  - **Reducing either *the length of the clock cycle (or increasing clock rate) or the number of clock cycles***
  - **Hardware designer must often trade off clock rate against cycle count**

# 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 1.2 times as many clock cycles as computer A to run the program.**

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

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

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

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

# Clock Cycles per Instruction

- **Performance equation w/ the number of instructions?**
  - **The compiler generates the instructions to execute**
  - **Execution time must depend on the number of instructions in a program**

$$\text{\# CPU clock cycles for a program} = \text{\# Instructions for a program} \times \text{Average clock cycles per instruction}$$

- ***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**

# The Performance Equation

- **Our basic performance equation is then**

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{clock period}$$

or

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{clock rate}}$$

- **Calculate avg. CPI?**
  - **Can measure the CPU execution time by running the program**
  - **The clock rate is usually given**
  - **Can measure overall instruction count by using profilers or simulators without knowing all of the implementation details**
  - **Modern processors offer HW performance counter measuring CPI or IPC**
- **CPI varies by instruction type and ISA implementation**

# Using the Performance Equation

- **Computers A and B implement the same ISA. Computer A has a clock cycle time of 250 ps and an average CPI of 2.0 for a program and computer B has a clock cycle time of 500 ps and an average 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

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

# Effective CPI

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

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

  - **Where $IC_i$ is the count (percentage) of the number of instructions of class i executed**
  - **$CPI_i$ 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 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 |

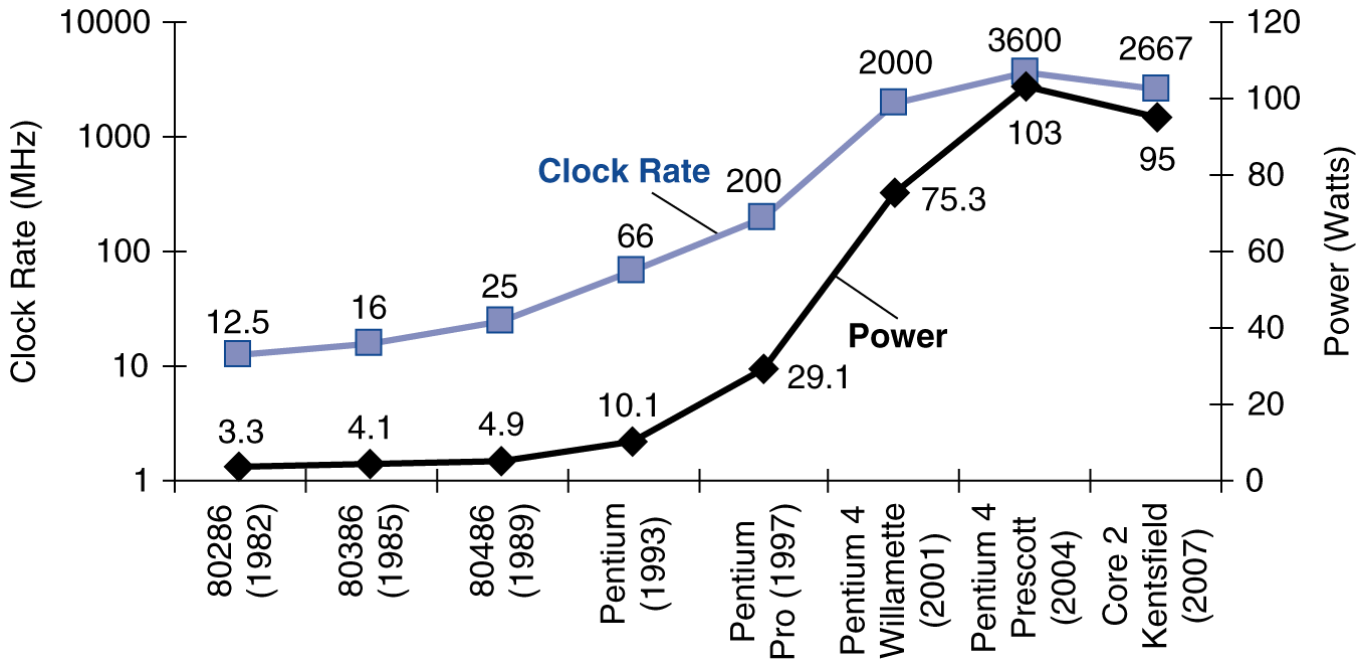
- **How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?**

  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 Trends




- **In CMOS IC technology**

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

# Reducing Power

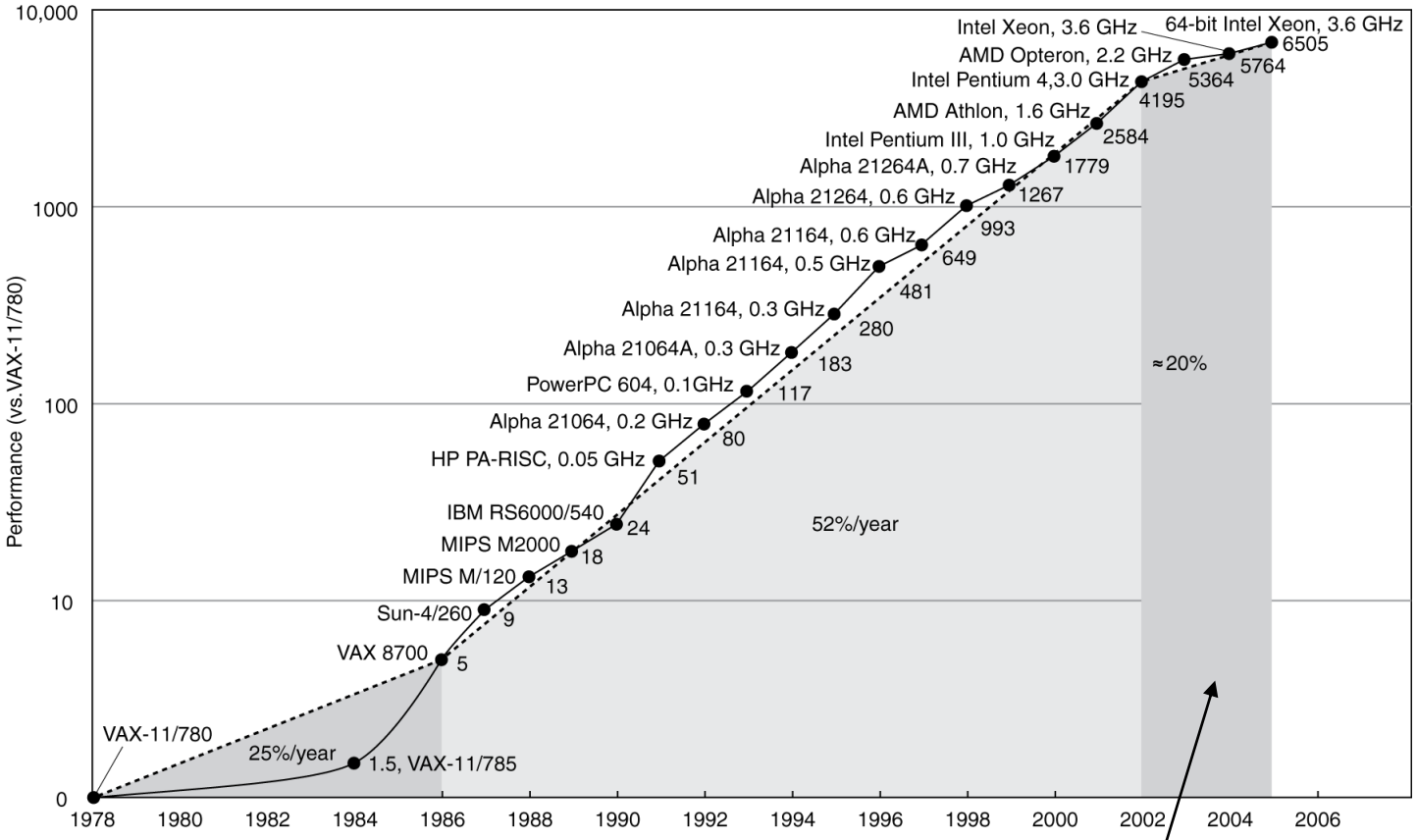
- **Suppose a new CPU has**
  - **15% capacitive load, 15% voltage, and 15% frequency reduction**

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

- **How to overcome the power wall**
  - **Reduce frequency -> reduce performance also**
  - **Reduce voltage further -> leading to more leakage power**

- **How else can we improve performance?**

# Uniprocessor Performance




Constrained by power, instruction-level parallelism, memory latency

# Multi-core Processors

- **Multi-core processors**
  - **More than one core 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**

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

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

- Can't be done!

- ***Make the common case fast!***