Computer Architecture

Chapter 1:
Performance

Joonho Kong

School of Electronics Engineering
Kyungpook National University

*** Thanks Prof. Sung Woo Chung in Korea University allowing me to use the ppt materials**

Main points for this chapter

- How can we quantify performance?
- What are the metrics for performance?
- To improve performance, how can we do?
- What factors do affect performance?
- What another metric for performance and their pitfalls?
- What is benchmark?
- What are Amdahl’s law and its implications for computer architecture design?
- What is power wall?

How to define performance

- There are many ways to define something as “the best”
- Airplane example

Airplane	Passenger Capacity	Cruising time (miles)	Cruising speed (m.p.h)	Throughput (passengers * m.p.h)
Boeing 777	375	5,256	610	228,750
Boeing 747	① 416	7,156	610	286,700
Airbus 380	525	8,200	560	① 294,000
BAC/Sud Concorde	132	4,000	① 1,350	178,200
Douglas DC-8-50	146	① 8,720	544	79,424

- What is “the best”?

Applying to computers…

- How do you decide which computer is the best?
  - Processor speed
  - System bus speed
  - Memory size
  - Disk storage
  - Graphics card / subsystem
  - Power consumption
  - Price

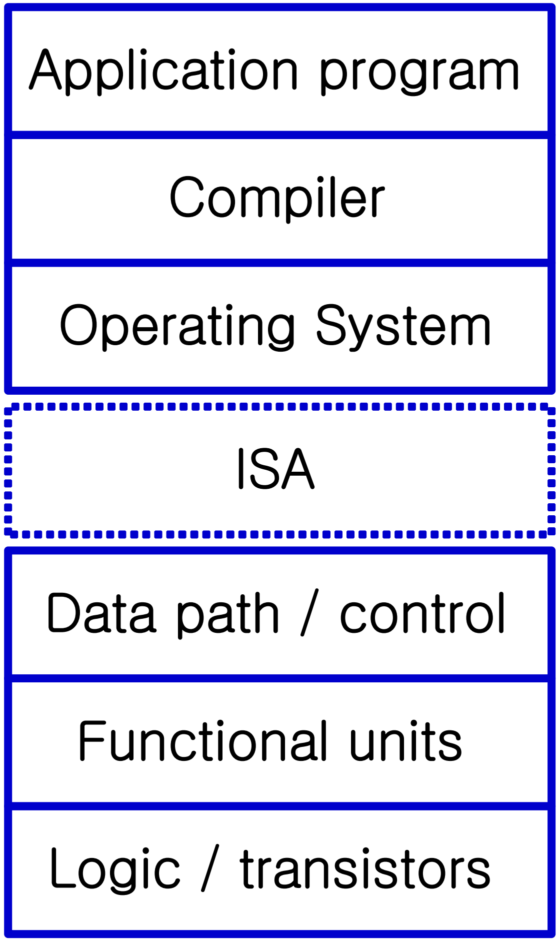
- How do you decide which one to buy?
  - Tradeoff between cost and performance

Quantifying the performance

- Important when
  - Purchasing a computer
  - Evaluating new technologies
  - Writing software
  - Implementing an instruction set
  - Designing a new architecture

- Therefore, it is important to understand how to define performance and what the limitations are of those metrics

Example of metrics in computers



Application program

Compiler

Operating System

ISA

Data path / control

Functional units

Logic / transistors

Answers a month

Operations per second

(Millions) of instructions per second: MIPS

(Millions) of (F.P.) operations per second: MFLOPS/s

Cycles per instruction: CPI

Clock frequency: MHz

- Most of metrics are related to time
  - Time is the most classical metric for computers

Performance metrics for computer systems

- Execution time
  - How fast?
  - time taken for doing a certain work
  - e.g., This computer takes 10 seconds to run a certain program

- Throughput
  - How much?
  - amount of work done per time
  - e.g., This SSD can read or write 10 Giga-byte data per second

Measuring time

- Looking at measuring CPU performance, we are primarily concerned with execution time

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

- To compare, we say “X is n times faster than Y”

$$n = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X}$$

- Increase performance, decrease execution time
  - Improve performance, improve execution time

Example of performance comparison

- X and Y do their homework
  - X takes 5 hours
  - Y takes 10 hours
- Compare the performance

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X} = \frac{1}{5 \text{ hours}} = 0.2$$

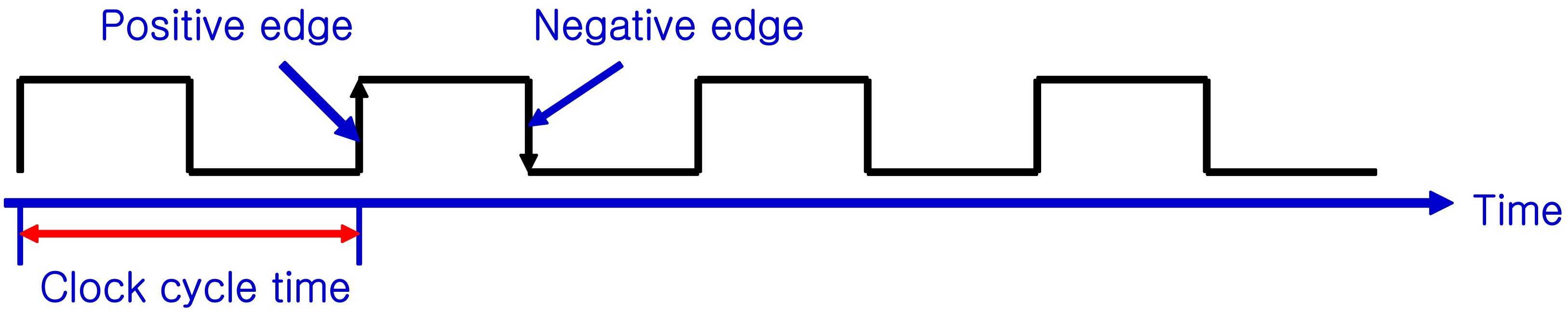
$$\text{Performance}_Y = \frac{1}{\text{Execution time}_Y} = \frac{1}{10 \text{ hours}} = 0.1$$

- So, X is two times faster than Y

$$n = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{0.2}{0.1} = 2$$

Clock cycle time vs. Clock rate

- Clock cycle time
  - Time required for a clock pulse to make transitions: 0 → 1 → 0

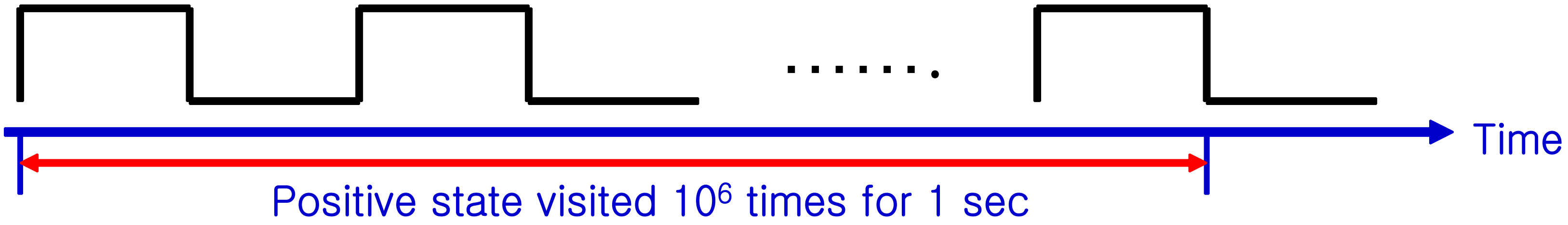


  - In other words, the time duration between positive (negative) edges

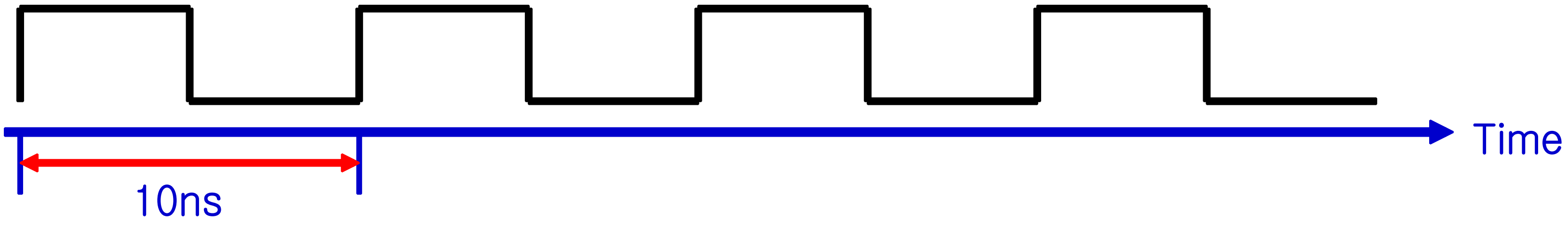
- Clock rate
  - Inverse of clock cycle time
  - # of times to visit positive (negative) state per second
  - Unit: Hz or MHz or GHz

Example of clock cycle time

- A machine is running at 100MHz
  - Clock rate = 100MHz = 100 * 10^6 cycles / sec

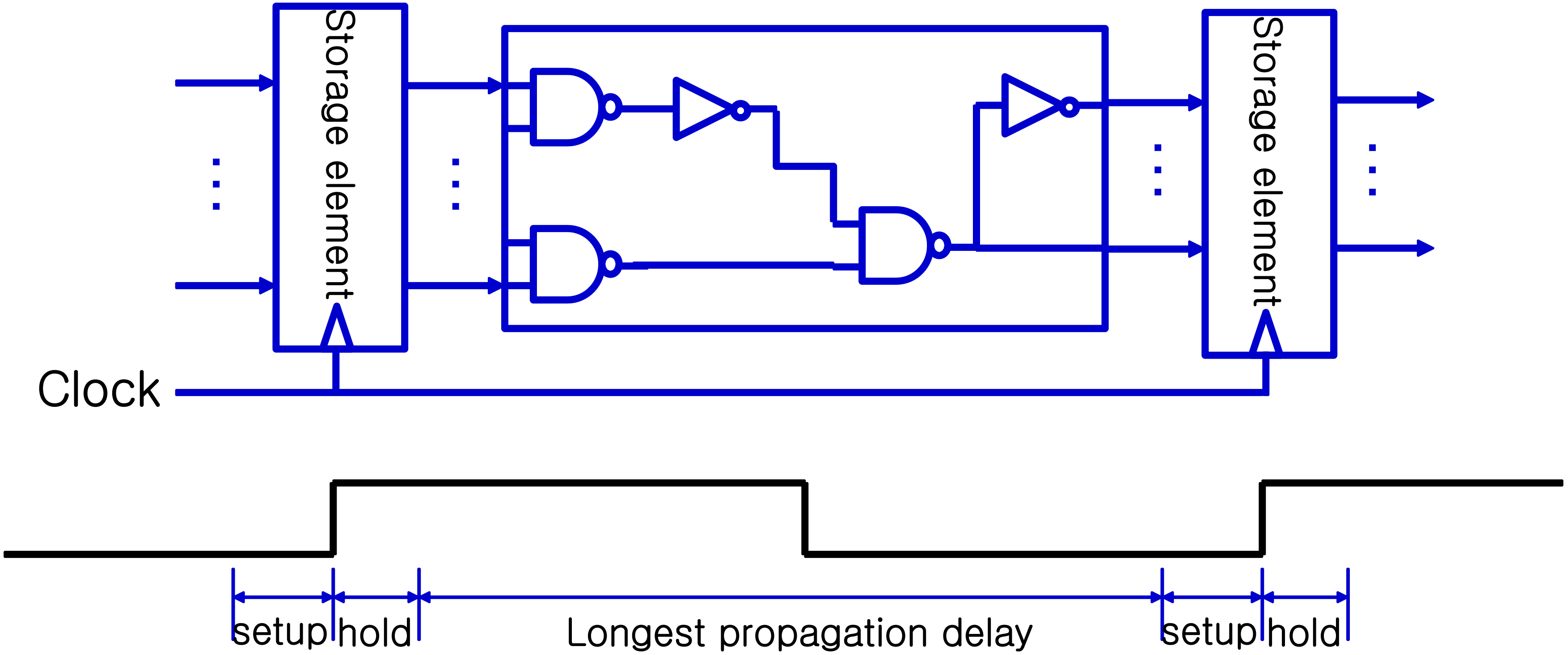


  - Clock cycle time = 1/(100*10^6) cycles / sec = 10ns



How is clock cycle time determined?

- Closely related to logic design
- Assumptions
  - All storage elements have same clock
  - Edge-triggered clocking
  - Design always work if the clock is slow enough



Meaning of timing parameters

- Longest propagation delay
  - A.k.a critical path delay
  - Critical path: a path that takes the most timing delays among many combinational paths
  - Typically identified by timing analysis or static timing analysis

Execution time

- We will use CPU execution time frequently as the metric of how long a program should run
  - Execution time = Clock cycles for program * Clock cycle time
- Since clock cycle time is the inverse of clock rate
  - **Execution time =** $\dfrac{\textbf{Clock cycles for program}}{\textbf{Clock rate}}$

Measuring clock cycles

- CPU clock cycles / program is not so intuitive
  - Program is a set (ordered) of instructions
- CPI (Cycles Per Instruction) is used so frequently
  - The # of cycles per instruction varies, so CPI is an average value
  - IPC?

Using CPI

- Therefore, we can rewrite
  - Execution time = Instructions * CPI * Clock cycle time
  - Improved performance (reduced execution time) is possible with increased clock rate (reduced clock cycle time), lower CPI, or reduced instructions
  - Designers have to balance the length of each cycle and the number of cycles required

CPI Example

- Machine A: 1ns clock and CPI of 2.0
- Machine B: 2ns clock and CPI of 0.5
- Which is faster?

Example solution

- Solve CPU time for each machine
  - Execution time_A = I * 2.0 * 1ns = 2.0 * I ns
  - Execution time_B = I * 0.5 * 2ns = 1.0 * I ns

- Compare performance

$$\frac{\textbf{Performance}_A}{\textbf{Performance}_B} = \frac{\textbf{Execution time}_B}{\textbf{Execution time}_A} = \frac{\textbf{1.0 * I ns}}{\textbf{2.0 * I ns}} = \textbf{0.5}$$

- So, machine A is 0.5 times faster than machine B

  = Machine B is 2 times faster than machine A
  - You must consider both
    - CPI
    - Clock rate

CPI variability

- Different types of instructions often take different numbers of cycles on the same processor
- CPI is often reported for classes of instructions
  - Clock cycles = $\sum_{i=1}^{n} (\mathbf{CPI_i \times C_i})$
  - CPI_i : the CPI for the class of instructions
  - C_i: the count of that type of instructions

CPI from instruction mix

- $$\text{CPI} = \frac{\sum_{i=1}^{n} (\,\text{CPI}_i \times C_i\,)}{\text{Instruction Count}} = \sum_{i=1}^{n} (\,\text{CPI}_i \times \boxed{\frac{C_i}{\text{Instruction Count}}}\,)$$

frequency of appearance of the type i instructions

- CPI Example

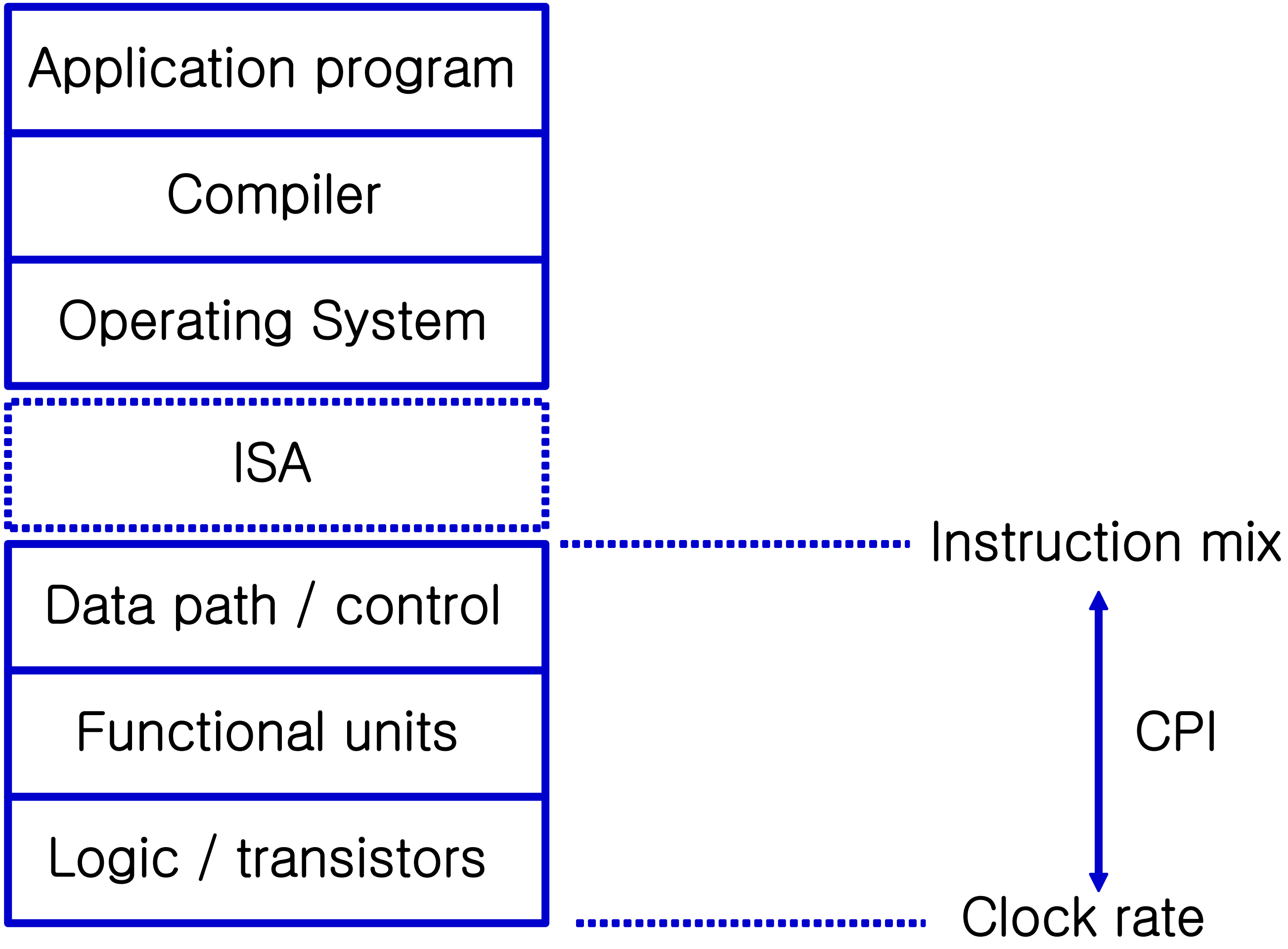
Instruction Class	Appearance Frequency	CPI_i
Instruction type 1	43%	1
Instruction type 2	21%	2
Instruction type 3	12%	2
Instruction type 4	24%	2

CPI = 0.43 x 1 + 0.21 x 2 + 0.12 x 2 + 0.24 x 2 = 1.57

Clock cycles = 1.57 * Instruction Count

Tradeoffs

- Instruction count, CPI, and clock rate present tradeoffs



Aspects of CPU performance

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

	Instruction count	CPI	Clock rate
Program	V		
Compiler	V		
ISA	V	V	
Organization		V	V
Technology			V

□ Ambiguous!! You only have to understand the meaning

Another popular performance metrics

- MIPS (million instructions per second)
    - MIPS = $\dfrac{\text{Instruction count}}{\text{Execution time} \times 10^6}$
    - Problems
        - It does not take into account the instruction set

- MFLOPS (million floating-point operations per second)
    - Operations rather than instruction
    - E.g. floating-point addition, multiplication, …
    - Often used for quantifying supercomputing performance or domain-specific architecture performance

A wrong use case of MIPS

- Consider a 500MHz machine

Class	CPI
Class A	1
Class B	2
Class C	3

- Consider the two compilers

Code from	Instruction counts (millions)		
	A	B	C
Compiler1	5	1	1
Compiler2	10	1	1

- Which compiler produce faster code? Has a higher MIPS?

A wrong use case of MIPS: solution (I)

- Compute clock cycles
  - Clock cycles = $\sum_{i=1}^{n} (\mathbf{CPI_i \times C_i})$
  - Clock cycles$_{comp1}$ = (1*5M) + (2*1M) + (3*1M) = 10M
  - Clock cycles$_{comp2}$ = (1*10M) + (2*1M) + (3*1M) = 15M
- Execution time
  - Execution time = (Instruction count * CPI) / Clock rate
= Clock cycles / Clock rate
  - Execution time$_{comp1}$ = 10M/500M = 0.02sec
  - Execution time$_{comp2}$ = 15M/500M = 0.03sec
- Code from compiler 1 is 1.5 times faster!

A wrong use case of MIPS: solution (II)

- Computer MIPS
  - MIPS = $\dfrac{\text{Instruction count}}{\text{Execution time} \times 10^6}$

  - $\text{MIPS}_{\text{comp1}}$ = (5M+1M+1M) / (0.02M) = 350
  - $\text{MIPS}_{\text{comp2}}$ = (10M+1M+1M) / (0.03M) = 400

- Code from compiler 2 is faster??
  - Fails to give a right answer!

Benchmarks

- Users often want a performance metric
- A benchmark is distillation of the attributes of a workload
  - Real applications usually work best, but using them is not always feasible
- Desirable attributes
  - Relevant: meaningful within the target domain
  - Understandable
  - Good metric(s)
  - Scalable
  - Coverage: does not oversimplify important factors in the target domain
  - Acceptance: vendors and users embrace it

Benchmarks (cont)

- de facto industry standard benchmarks for CPU
  - SPEC

- SPEC
  - Standard Performance Evaluation Cooperative
  - Founded in 1988 by EE times, SUN, HP, MIPS, Apollo, DEC
  - Several different SPEC benchmarks
  - Most include a suite of several different applications (such as integer and floating point components often reported separately)
  - For more information, visit http://www.spec.org
  - The latest version: SPEC CPU 2017

- Synthetic benchmarks do not come from real executables, but are designed to approximate a machine’s performance
  - Synthetic benchmarks such as Whetstone and Dhrystone are not really measuring anything people run and can be optimized

Amdahl's law

Execution speedup is proportional to the size of the improvement and the amount affected

- Execution time after improvement

$$= \left[\frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}\right]$$

- Or

$$\mathbf{ExTime_{new} = ExTime_{old} \times \left[(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}\right]}$$

Example – Amdahl's law

- Floating point instructions improved to run 2X; but only 10% of actual instructions are FP

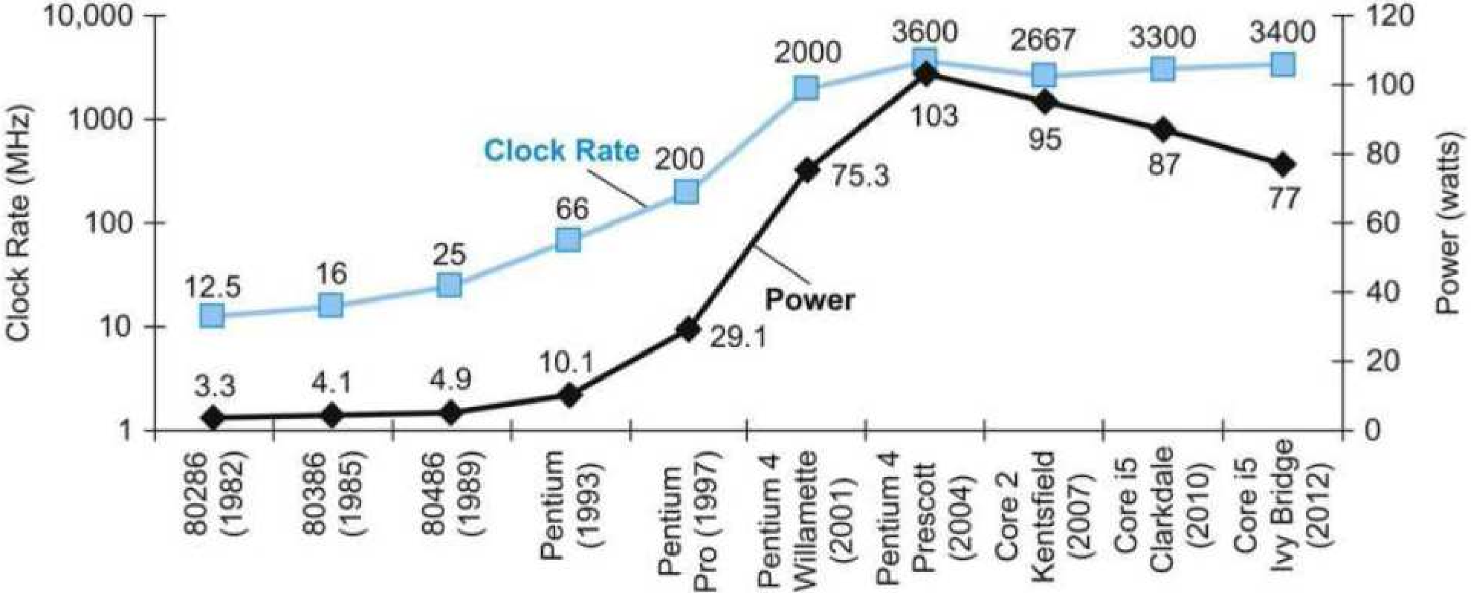
$$\mathbf{ExTime_{new} = ExTime_{old} \times (0.9 + 0.1/2) = 0.95 \times ExTime_{old}}$$

$$\mathbf{Speedup_{overall} = \frac{ExTime_{old}}{0.95 \times ExTime_{old}} = 1.053}$$

$$\boxed{\mathbf{Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}}}}$$

Power wall

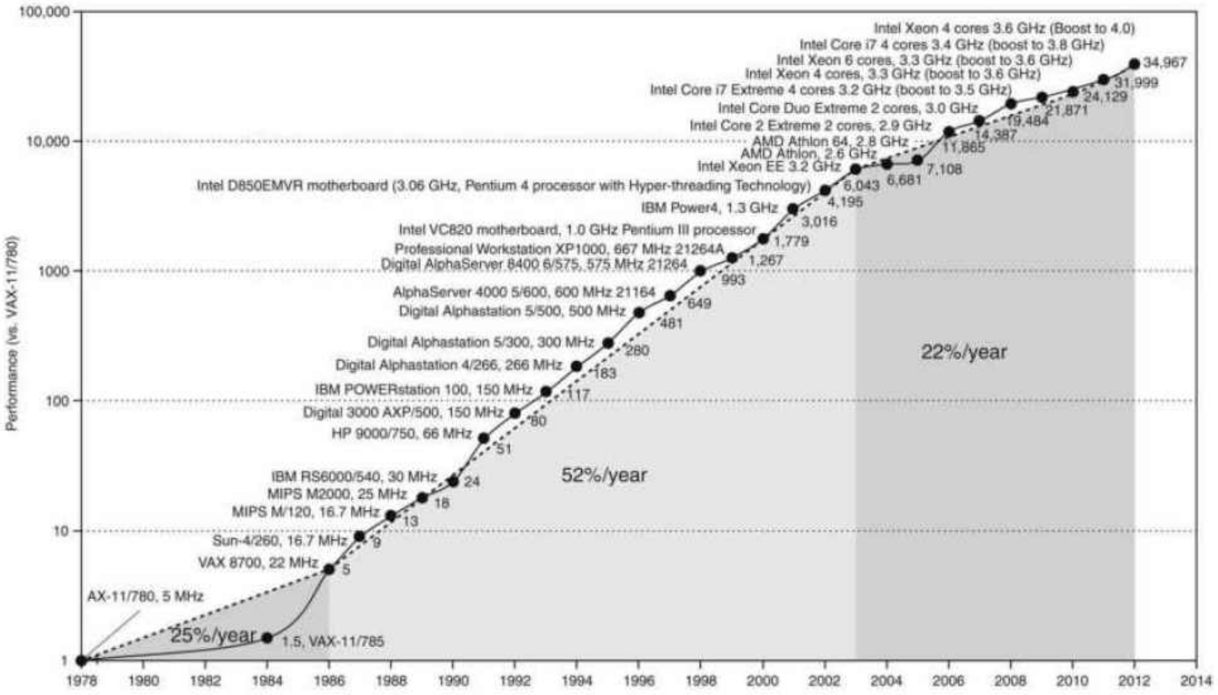
- Power consumption is getting more and more important!



  - Cost
  - Thermal – Deeply related to performance
  - The main reason we are using multi-core CPUs
    - Add more cores, reduce clock frequency, and extract parallelism from software
    - Still hard to exploit full performance available from CPU, why?

Trends in performance improvements

- Improvement rate per year is saturating



- Limits of power
- Limits in Instruction-Level Parallelism
- Memory wall
- Limits in improving performance of general purpose CPU
  - A rise of domain-specific architecture: e.g., GPU, NPU
  - Nor for a wide range of workloads but for specific workloads