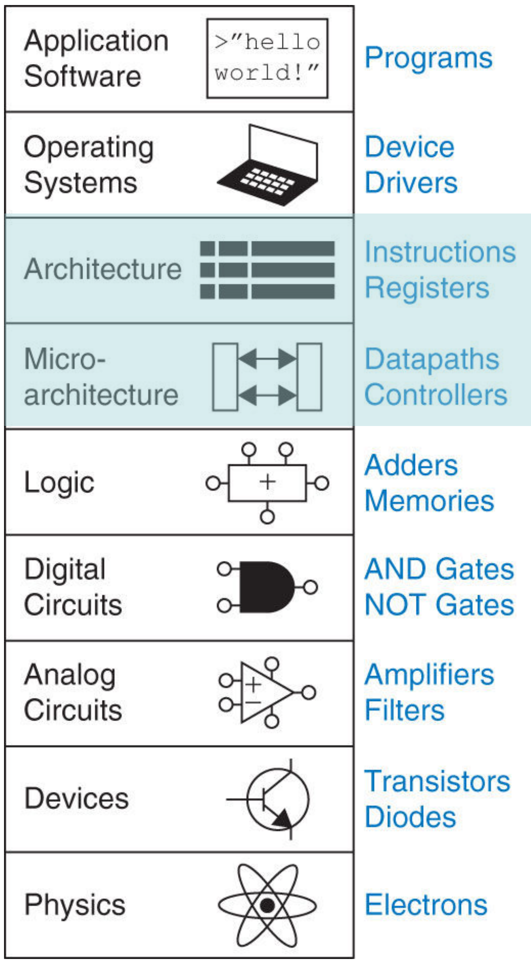
ELT3047 Computer Architecture

Lesson 2: Performance measurement

Hoang Gia Hung
Faculty of Electronics and Telecommunications
University of Engineering and Technology, VNU Hanoi

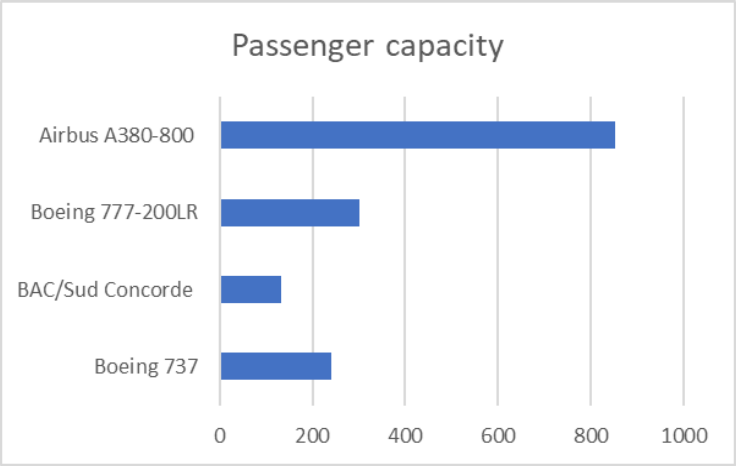
Last lesson review

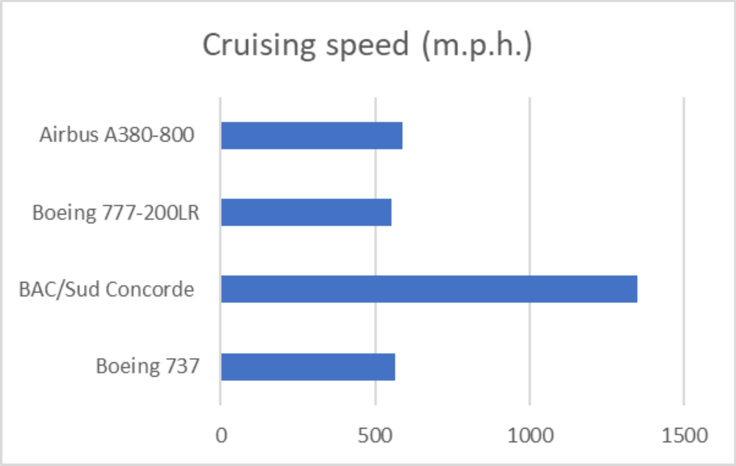


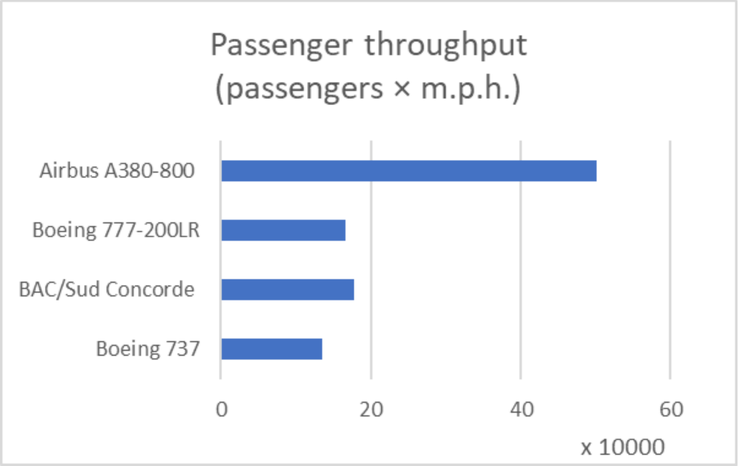
- Fundamental concepts in computer architechure
  - Computer definition
  - Computer evolution
  - Technology and cost trends
  - Classes of modern computers
- Computer architecture
  - Abstract layers
  - ISA and computer organization
  - Stored program concept
- Binary representations of numbers
- **Today's lecture**: performance measurement and reporting

Why do we need measuring computer performance?

- Which is the best computer?
  - Perceived differently from different perspective.
  - Aviation analogy: cruising speed? passenger throughput?
- Understanding performance
  - Key to underlying organizational motivation
  - Knowing which hardware/software factors affect the performance







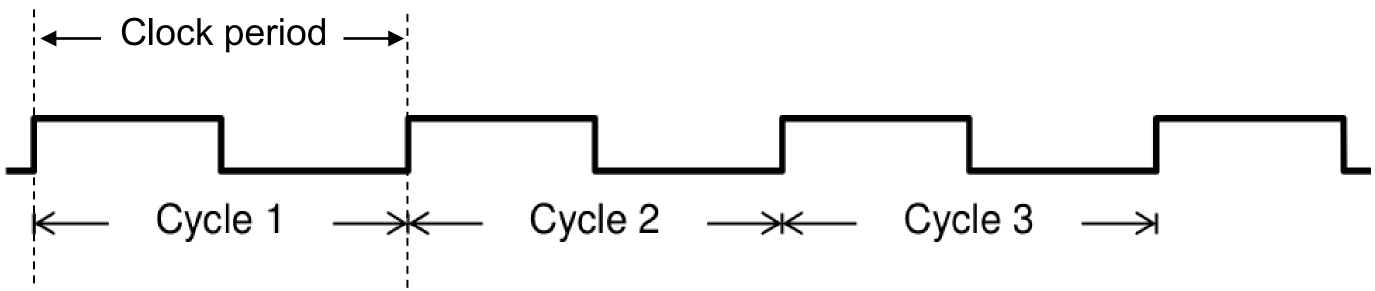
Defining computer performance

- Response time
  - Time between start and completion of a task (on a machine X), as observed by end user: $Performance_X = \frac{1}{Execution\ time_X}$
- Throughput
  - Total work done per unit time e.g., tasks/transactions/… per hour
- Response time vs. Throughput
  - Decreasing execution time improves throughput: less time to run a task ⇒ more tasks can be executed
  - Increasing throughput can also improve response time: even execution time of individual sequential tasks is not changed, more tasks can be executed in parallel ⇒ less waiting time in scheduling queue.
- In this course, we will primarily focus on **response time**
  - Performance are relative: engineers want future generation of computers (X) they are designing to be n times faster than the current generation (Y)

$$\frac{Performance_X}{Performance_Y} = \frac{Execution\ time_Y}{Execution\ time_X} = n$$

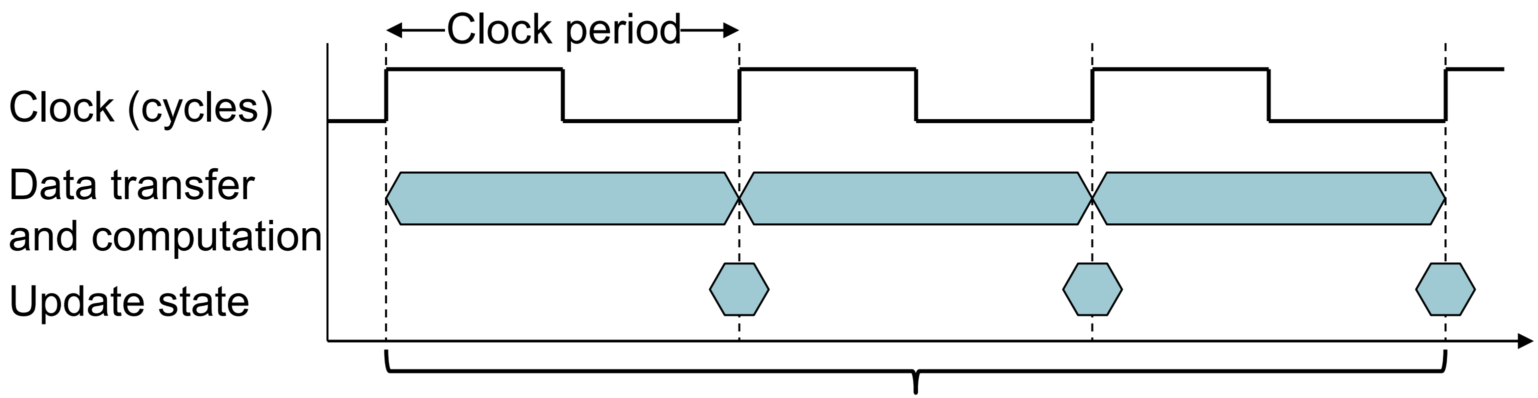
Measuring Execution Time

- Elapsed time = Total response time, including all aspects
  - Processing, I/O, OS overhead, idle time
  - Useful, but often not very good for designers (clock wall measurement?)
- Our focus: ***CPU time*** = time spent processing a given job
  - Doesn't count I/O time, other jobs' shares
  - Comprises ***user*** ***CPU time*** and ***system*** ***CPU time***
  - Can be measured in seconds or number of CPU clock cycles



- **Example:** A processor has clock frequency (clock rate) of 4GHz. What's its clock period (i.e. duration of a clock cycle)?
  - Answer: $1/(4 \times 10^9) = 250 \times 10^{-12}$ s (i.e. 250 ps).

CPU Performance and Its Factors



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

- Performance improved by
  - Reducing number of clock cycles required by a program
  - Increasing clock rate (i.e. reduce clock cycle time)
  - Hardware designer must often **trade off** clock rate against cycle count because many techniques that decrease the number of clock cycles may also increase the clock cycle time, as we will see in later chapters.

CPU Time Example

- Given Computer A: 2GHz clock, 10s CPU time. Task: design Computer B with following specs:
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?
- **Solution:**

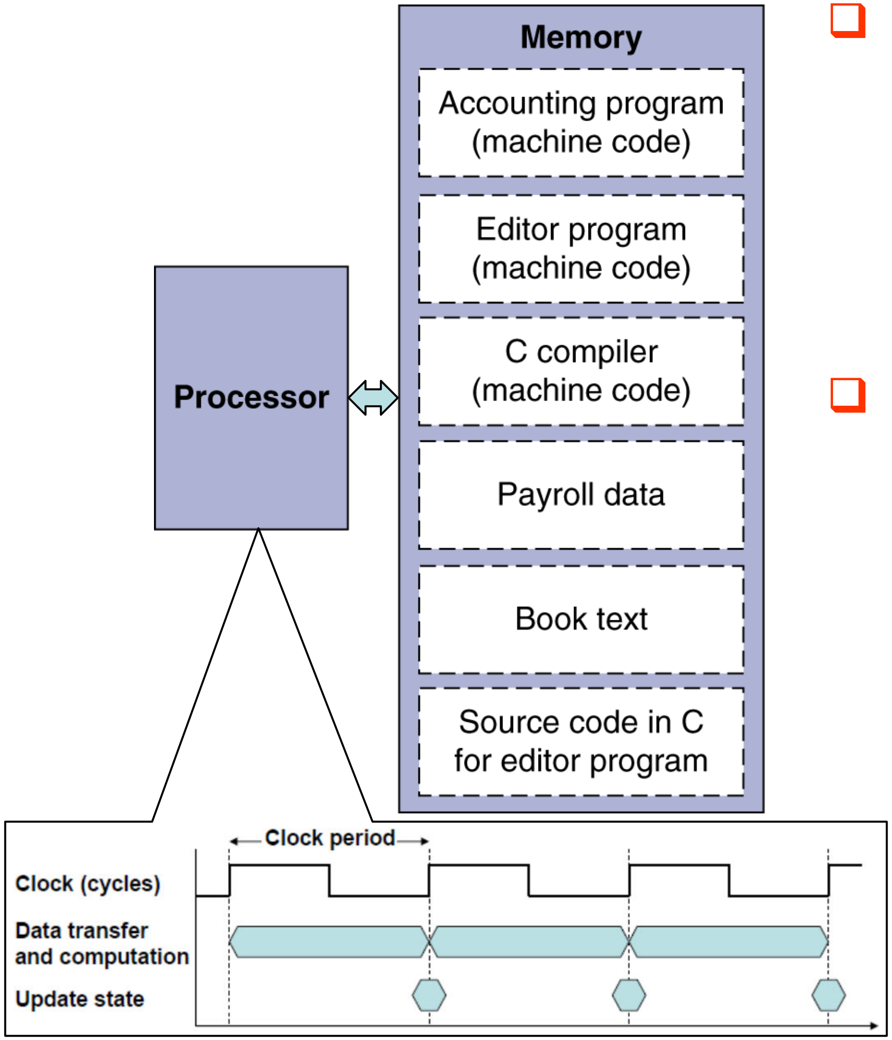
$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6s}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10s \times 2GHz = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4GHz$$

Instruction Count and CPI



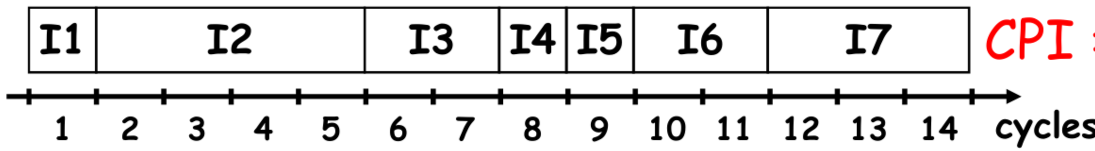
- ❑ Program = series of instructions stored in memory, sequentially fetched & executed at a constant clock rate.
  - #clock cycles depends on #instructions in a program (**instruction count**) & #cycles required by each instruction
- ❑ Instructions take **<u>different</u>** number of cycles to execute.
  - E.g. multiplication > addition, floating point operations > integer operations.
  - #clock cycles = **IC** x **CPI** (*<u>average</u>* #cycles per instruction).

$$CPU\ time = IC \times CPI \times Clock\ period = \frac{IC \times CPI}{Clock\ rate}$$

  - CPI provides one way of comparing different implementations of the same ISA (since IC would be the same).

CPI in More Detail

- The average number of cycles per instruction



CPI = 14/7 = 2

  - It's unrealistic to count #clock cycles for every instruction in a program.

- In practice, CPI depends on a wide variety of design details
  - HW factors: the memory system and the processor structure;
  - SW factors: the mix of instruction types executed in an application

- Each instruction **classes** (e.g. ALU, MEM, …) has different CPI
  - If a program has n different classes of instructions with corresponding CPI_i and instruction count IC_i, then $Clock\ cycles = \sum_{i=0}^{n} CPI_i \times IC_i$.
  - The (weighted average) CPI of the program is

$$CPI = \frac{Clock\ cycles}{IC} = \sum_{i=1}^{n}\left(CPI_i \times \frac{IC_i}{IC}\right)$$

Relative frequency

CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C. Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5
- Clock Cycles
  $= 2\times1 + 1\times2 + 2\times3$
  $= 10$
- Avg. CPI = 10/5 = 2.0

- Sequence 2: IC = 6
- Clock Cycles
  $= 4\times1 + 1\times2 + 1\times3$
  $= 9$
- Avg. CPI = 9/6 = 1.5

➢ Sequence 2 is faster, even though it executes one extra instruction.

Performance summary

- To execute, a given program will require
  - Some number of machine instructions = instruction count
  - An average number of clock cycles to run each instruction = CPI
- Therefore: (The Classic CPU Performance Equation)

$$\begin{aligned} CPU\ time &= IC \times CPI \times Clock\ period \\ &= \frac{IC \times CPI}{Clock\ rate} \end{aligned}$$

  - Cycle time: reciprocal of the CPU frequency, provided by manufacturer
  - Instruction count: measured by software tools (profiler) or hardware counters
  - CPI: obtain by simulation or hardware counters
- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, Clock cycle time

Performance Example 1

- Suppose we have two implementations of the same ISA for a given program. Which one is faster and by how much?

Machine	Clock cycle	CPI
A	250 (ps)	2.0
B	500 (ps)	1.2

- **Solution:**

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$
$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$
$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

A is faster…

…by this much

Performance Example 2

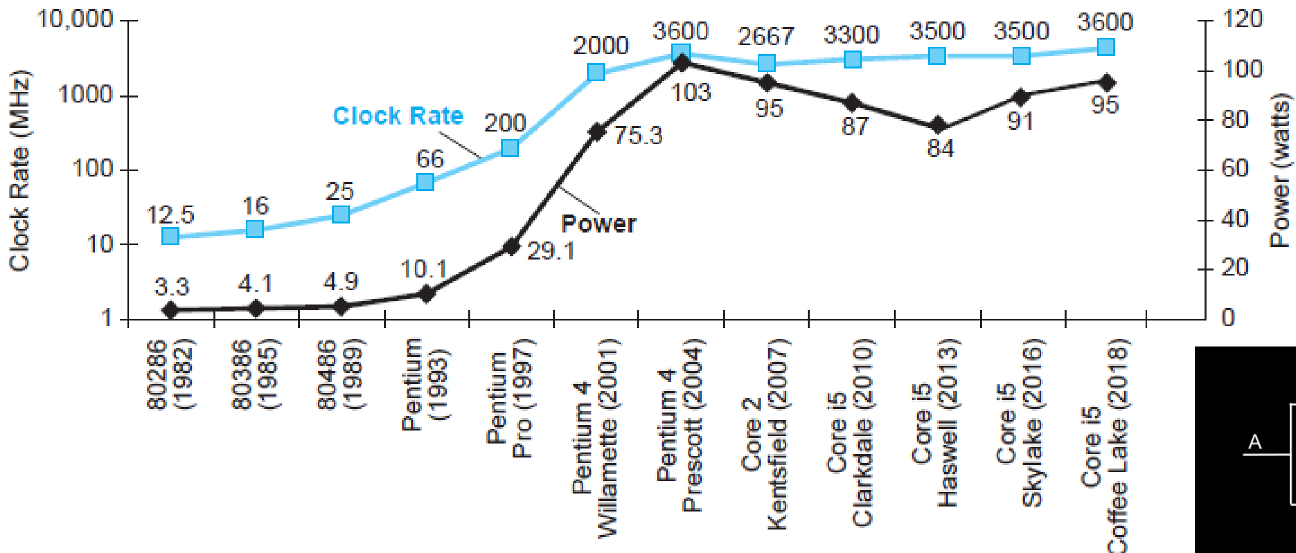
- Given: instruction mix of a program on a computer

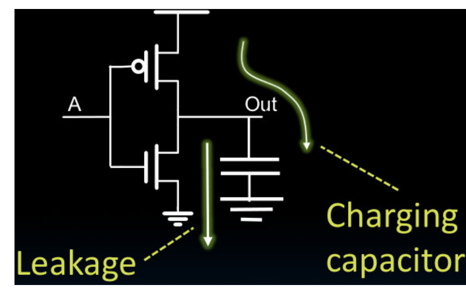
Class_i	Freq_i	CPI_i
ALU	50%	1
Load	20%	5
Store	10%	3
Branch	20%	2

  - What is average CPI? What is the % of time used by each instruction class?
  - How faster would the machine be if load time is 2 cycles? What if two ALU instructions could be executed at once?

- **Solution:**
  - Average CPI = 0.5x1+0.2x5+0.1x3+0.2x2 = 2.2. Time percentages: 23%, 45%, 14%, 18%.
  - If load time is 2 cycles: $\frac{old\ CPU\ time}{new\ CPU\ time} = \frac{old\ CPI}{new\ CPI} = \frac{2.2}{1.6} = 1.38$
  - If two ALU instructions could be executed at once: $\frac{old\ CPI}{new\ CPI} = \frac{2.2}{1.95} = 1.13$

Performance - clock rate relation

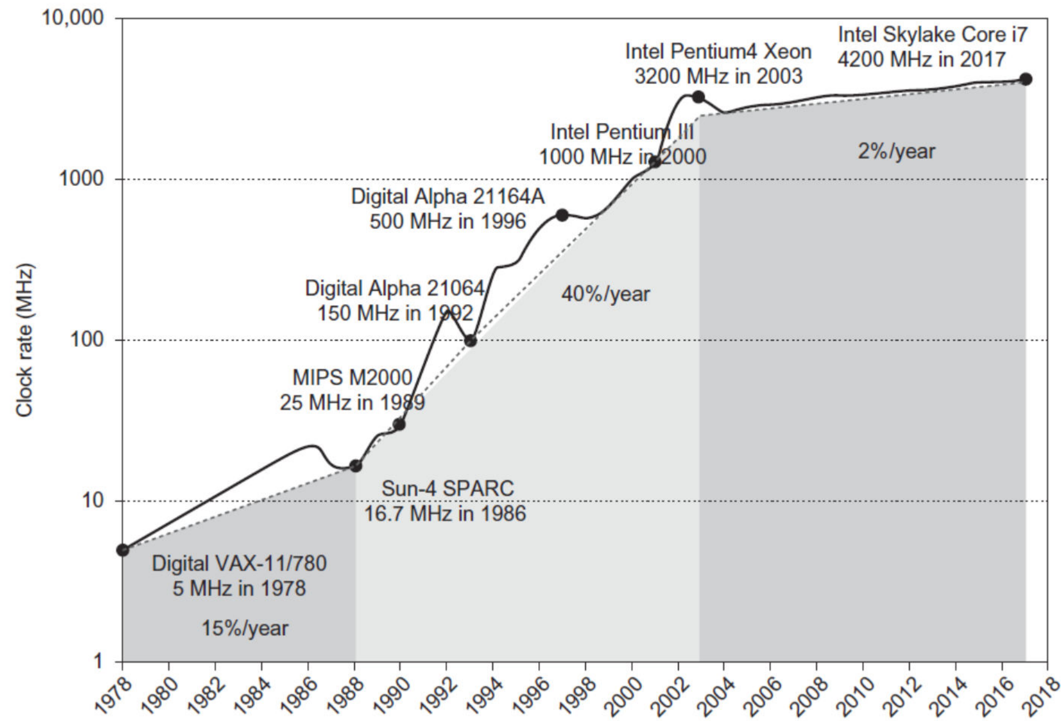




- In CMOS IC technology:
  - Power = P_{static} (~40%, due to leakage) + $P_{dynamic}$ (~60%, due to capacitance charging when signals change between 0 and 1).
  - $P_{dynamic} = \frac{1}{2} C V_{DD}^2 f \rightarrow$ raising clock rate increases power consumption.
- Problem: Get power in, get power out
  - Intel 80386 consumed ~ 2 W, 3.3 GHz Intel Core i7 consumes 130 W
  - Heat must be dissipated from 1.5 x 1.5 cm chip $\rightarrow$ limit of what can be cooled by air

The power wall

- Techniques to ↓ power
  - Reduce voltage, but there's a limit (↑ leakage power → transistors don't fully turn off).
  - Frequency scaling
  - Power gating
- Still, in recent years
  - Size of transistors (= capacitance) not shrinking as much.



- Power becomes a growing concern – the "power wall"
- How else can we improve performance? switch to multiprocessors
  - More than one processor per chip
  - Hard to do: programming, load balancing, optimizing communication & synchronization.

MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Faster machine ⇒ larger MIPS
  - Relations to previous performance measures

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

  - Similar concept: MFLOPS = millions of floating point operations per second
- Advantage: easy to understand/marketing.
- Drawbacks:
  - Cannot compare computers with different instruction sets because the instruction count will differ
  - Varies between programs on the same computer
  - Higher MIPS rating does not always mean better performance

MIPS example

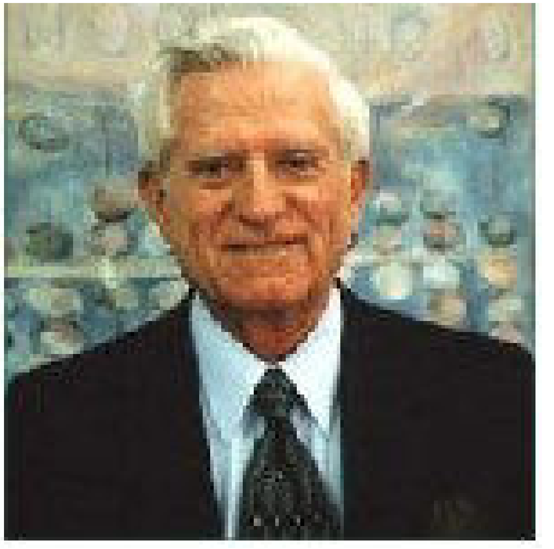
- Two different compilers are being tested on the same program for a 4 GHz machine with three different classes of instructions: Class A, Class B, and Class C, which require 1, 2, and 3 cycles, respectively.
- The instruction count produced by the first compiler is 5 billion Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.
- The second compiler produces 10 billion Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.
- **Questions**:
  - Which compiler produces a higher MIPS?
  - Which compiler produces a better execution time?

MIPS example solution

- First, we find the CPU cycles for both compilers.
  - CPU cycles (compiler 1) = $(5\times1 + 1\times2 + 1\times3)\times10^9 = 10\times10^9$
  - CPU cycles (compiler 2) = $(10\times1 + 1\times2 + 1\times3)\times10^9 = 15\times10^9$
- Next, we find the execution time for both compilers.
  - Execution time (compiler 1) = 10×10^9 cycles / 4×10^9 Hz = 2.5 sec
  - Execution time (compiler 2) = 15×10^9 cycles / 4×10^9 Hz = 3.75 sec
- Compiler1 generates faster program (less execution time).
- Now, we compute MIPS rate for both compilers.
  - MIPS (compiler 1) = $(5+1+1) \times 10^9 / (2.5 \times 10^6) = 2800$
  - MIPS (compiler 2) = $(10+1+1) \times 10^9 / (3.75 \times 10^6) = 3200$
- So, code from compiler 2 has a higher MIPS rating even though it is slower in execution time.

Amdahl's Law

- Performance improvements might not always be as good as they sound.
  - Improvement of **one aspect** does **NOT** necessarily lead to **proportional** improvement in **overall** performance.
  - How much?



- Speedup(E) due to an enhancement E is

$$Speedup(E) = \frac{Perf.with\ E}{Perf.before} = \frac{Ex.Time\ before}{Ex.Time\ with\ E}$$

  - If E improves a fraction f of execution time by a factor s, and the remaining time is unaffected: $Ex.Time\ with\ E = Ex.Time\ before \times \left(\frac{f}{s} + (1-f)\right)$
  - **Amdahl's Law**: Speedup(E) is measured by

$$\boxed{Speedup(E) = \frac{1}{\frac{f}{s} + (1-f)}}$$

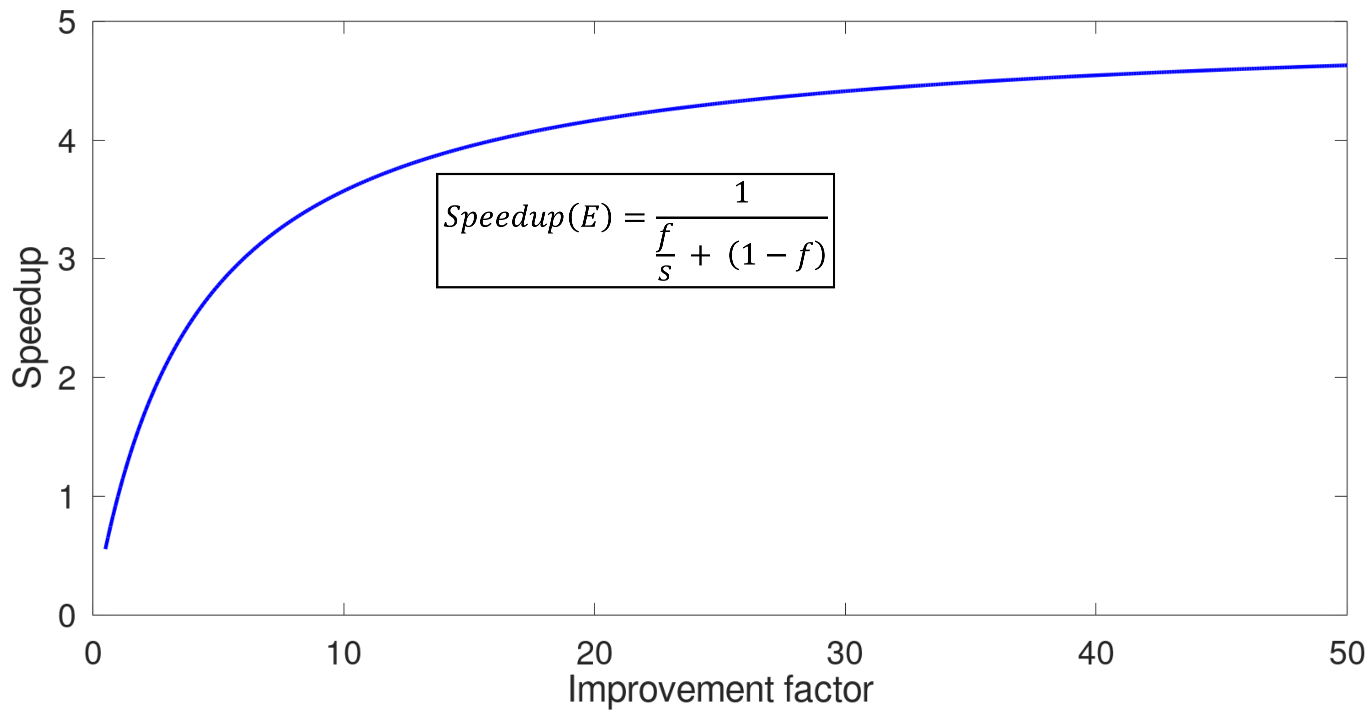
  - **Design principle:** Make common case fast!

Amdahl's Law example

- Suppose a program runs in 100 seconds on a machine, with multiply 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? 5 times faster?
- **Solution:**
  - Suppose we improve multiplication by a factor s
  - The 4 times faster overall execution time satisfies: 25 sec (4 times faster) = 80 sec / s + 20 sec
  - s = 80 / (25 – 20) = 80 / 5 = 16 → Improve the speed of multiplication by s = 16 times.
  - The 5 times faster overall execution time satisfies: 20 sec (5 times faster) = 80 sec / s + 20 sec
  - s = 80 / (20 – 20) = ∞ → Impossible to make 5 times faster!

Diminishing returns with improved performance

- Previous example shows that, according to Amdahl's law, we will approach a speedup of 5 asymptotically, regardless of how much the multiplication performance is improved.
  - Diminishing returns: if f is fixed, the total speedup rate diminishes with increased s.



$$Speedup(E) = \frac{1}{\frac{f}{s} + (1-f)}$$

Benchmarks

- As CPUs became more sophisticated → determine execution time becomes harder.
- **Benchmarking**: using real applications to measure performance
  - Supposedly typical of **actual workload**
  - Representatives of expected classes of applications (compilers, editors, scientific applications, graphics, ...) ← **make common case fast**.
  - Focus on reproducibility: must provide every detail so that another experimenter would need to duplicate the results
- SPEC (System Performance Evaluation Corporation)
  - Funded and supported by a number of computer vendors
  - Companies have agreed on a set of real program and inputs
  - Various benchmarks for CPU performance, graphics, high-performance computing, client- server models, file systems, Web servers, etc.
  - Valuable indicator of performance (and compiler technology)

SPEC CPU Benchmark

- Measure elapsed time to execute a selection of programs (with neglectable I/O), and normalized relative to reference machine
  - Summarize as **geometric mean** of performance ratios: $\sqrt[n]{\prod_{i=1}^{n} Perf.ratio_i}$

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^–9)	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Perl interpreter	perlbench	2684	0.42	0.556	627	1774	2.83
GNU C compiler	gcc	2322	0.67	0.556	863	3976	4.61
Route planning	mcf	1786	1.22	0.556	1215	4721	3.89
Discrete Event simulation - computer network	omnetpp	1107	0.82	0.556	507	1630	3.21
XML to HTML conversion via XSLT	xalancbmk	1314	0.75	0.556	549	1417	2.58
Video compression	x264	4488	0.32	0.556	813	1763	2.17
Artificial Intelligence: alpha-beta tree search (Chess)	deepsjeng	2216	0.57	0.556	698	1432	2.05
Artificial Intelligence: Monte Carlo tree search (Go)	leela	2236	0.79	0.556	987	1703	1.73
Artificial Intelligence: recursive solution generator (Sudoku)	exchange2	6683	0.46	0.556	1718	2939	1.71
General data compression	xz	8533	1.32	0.556	6290	6182	0.98
Geometric mean							2.36

SPECpower_ssj2008 for Xeon E5-2650L

Target Load %	Performance (ssj_ops)	Average Power (watts)
100%	4,864,136	347
90%	4,389,196	312
80%	3,905,724	278
70%	3,418,737	241
60%	2,925,811	212
50%	2,439,017	183
40%	1,951,394	160
30%	1,461,411	141
20%	974,045	128
10%	485,973	115
0%	0	48
Overall Sum	26,815,444	2,165
∑ssj_ops / ∑power =		12,385

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec, Power: Watts (Joules/sec)

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

Summary

- Various measures for computer performance
  - Execution time: the best performance measure for designers
  - MIPS/MFLOPS: easy to understand but contains many drawbacks
  - Benchmarks: use real applications – the best performance measure for users
- Factors affecting execution time
  - Instruction counts
  - CPI
  - Clock cycle time (rate)
- Power is a limiting factor (the power wall)
  - Use parallelism to improve performance
  - Improvement of one aspect does not necessarily lead to proportional improvement in overall performance (Amdahl's law)
- Next week: ISA design.