CSC 411

Computer Organization (Spring 2022)
Lecture 4: Performance

Prof. Marco Alvarez, University of Rhode Island

Quick notes

- Getting access to a Linux machine
  - download your favorite OS and install
    - standalone or side-by-side with Windows
  - built-in Unix subsystem for Windows 10
    - https://docs.microsoft.com/en-us/windows/wsl/install-win10
  - use a virtual machine

SI prefixes

Factor	Name	Symbol	Factor	Name	Symbol
10^{24}	yotta	Y	10^{-1}	deci	d
10^{21}	zetta	Z	10^{-2}	centi	c
10^{18}	exa	E	10^{-3}	milli	m
10^{15}	peta	P	10^{-6}	micro	μ
10^{12}	tera	T	10^{-9}	nano	n
10^{9}	giga	G	10^{-12}	pico	p
10^{6}	mega	M	10^{-15}	femto	f
10^{3}	kilo	k	10^{-18}	atto	a
10^{2}	hecto	h	10^{-21}	zepto	z
10^{1}	deka	da	10^{-24}	yocto	y

"The 20 SI prefixes used to form decimal multiples and submultiples of SI units."

https://physics.nist.gov/cuu/Units/prefixes.html

Prefixes for binary multiples

Factor	Name	Symbol	Origin	Derivation
2^{10}	kibi	Ki	kilobinary: $(2^{10})^1$	kilo: $(10^3)^1$
2^{20}	mebi	Mi	megabinary: $(2^{10})^2$	mega: $(10^3)^2$
2^{30}	gibi	Gi	gigabinary: $(2^{10})^3$	giga: $(10^3)^3$
2^{40}	tebi	Ti	terabinary: $(2^{10})^4$	tera: $(10^3)^4$
2^{50}	pebi	Pi	petabinary: $(2^{10})^5$	peta: $(10^3)^5$
2^{60}	exbi	Ei	exabinary: $(2^{10})^6$	exa: $(10^3)^6$

"In December 1998 the International Electrotechnical Commission (IEC), the leading international organization for worldwide standardization in electrotechnology, approved as an IEC International Standard names and symbols for prefixes for binary multiples for use in the fields of data processing and data transmission."

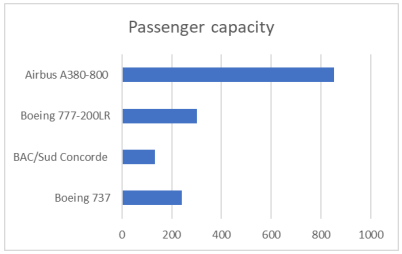
https://physics.nist.gov/cuu/Units/binary.html

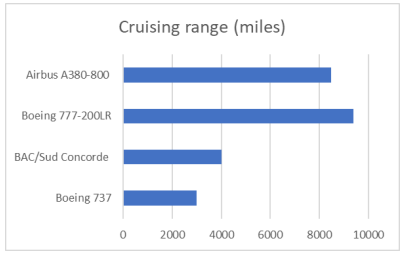
Examples

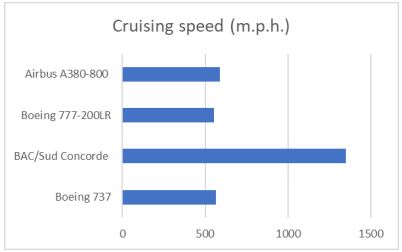
Examples and comparisons with SI prefixes	
one **kibibit**	1 Kibit = 2^{10} bit = **1024 bit**
one **kilobit**	1 kbit = 10^{3} bit = **1000 bit**
one **byte**	1 B = 2^{3} bit = **8 bit**
one **mebibyte**	1 MiB = 2^{20} B = **1 048 576 B**
one **megabyte**	1 MB = 10^{6} B = **1 000 000 B**
one **gibibyte**	1 GiB = 2^{30} B = **1 073 741 824 B**
one **gigabyte**	1 GB = 10^{9} B = **1 000 000 000 B**

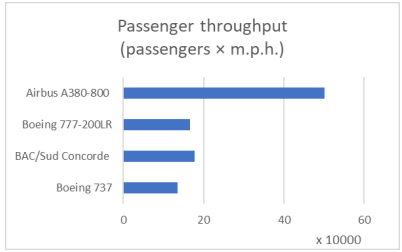
Performance

Think about performance …









Understanding performance

- Algorithm
  - determines number of operations executed
- Programming language, compiler, architecture
  - determines number of machine instructions executed per operation
- Processor and memory system
  - determines how fast instructions are executed
- I/O system (including OS)
  - determines how fast I/O operations are executed

Response time and throughput

- Response time
  - how long it takes to do a task
- Throughput
  - total work done per unit time
  - e.g., tasks/transactions/… per hour
- We'll focus on **response time** for now…

Quizz

- In the following scenarios, what are the changes: (A) decreasing response time, (B) increasing throughput, or (C) both?
  - replacing the processor in a computer with a faster version
  - adding additional processors to a system that uses multiple processors for separate tasks

Performance and speedup

- Performance

$$\text{performance}_X = \frac{1}{\text{execution time}_X}$$

- Relative performance (speedup):
  - "X is **n** times faster than Y"

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

- Example: time taken to run a program
  - 10s on A, 15s on B
  - n = 15s / 10s = 1.5
  - A is **1.5** times faster than B

Measuring execution time

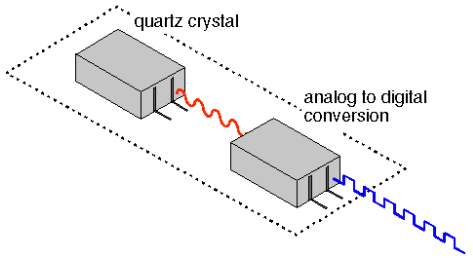
- **Elapsed time** (wall clock time, response time)
  - total response time, including all aspects: processing, I/O, OS overhead, idle time
  - determines system performance
- CPU execution time (CPU time)
  - time spent procession a job, discounts I/O time, other jobs' shares
  - comprises user **CPU time** and **system CPU time** (difficult to separate accurately)
- Different applications are affected by different performance aspects
  - one must have a clear definition of what performance metric to use to improve system/program performance

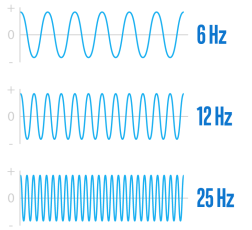
```
$ time ls
$ time host uri.edu
$ time (seq 1000 | wc -l)
$ time (seq 10000000 | wc -l)
```

The time command may show different outputs if run on a Linux or Mac systems, in general it shows "elapsed time", "CPU time", and "system time"

CPU clocking

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





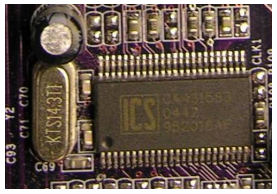
"A clock cycle is a single electronic pulse of a CPU. During each cycle, a CPU can perform a basic operation. Most CPU processes require multiple clock cycles."

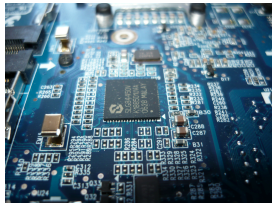
https://www.engineersgarage.com/how_to/how-clock-in-computer-works/
https://www.intel.com/content/www/us/en/gaming/resources/cpu-clock-speed.html

Clock generator

A **clock generator** is an electronic oscillator that produces a clock signal for use in synchronizing a circuit's operation. The signal can range from a simple symmetrical square wave to more complex arrangements. The basic parts that all clock generators share are a resonant circuit and an amplifier.

The resonant circuit is usually a quartz piezo-electric oscillator, although simpler tank circuits and even RC circuits may be used.

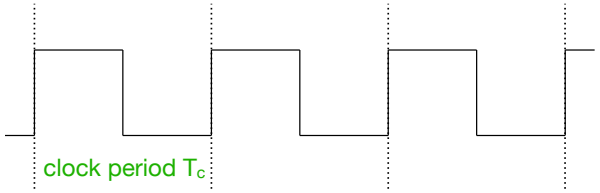




https://en.wikipedia.org/wiki/Clock_generator

CPU clocking

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



$$f_c = \frac{1}{T_c}$$

CPU execution time

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

- Performance improved by:
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B aiming for 6s CPU time
  - can do faster clock, but causes 1.2 times the number of clock cycles required by the program
- What clock rate should the designer target?

$$\text{clock cycles}_A = \text{CPU time}_A \times \text{clock rate}_A = 10s \times 2GHz = 20 \times 10^9$$
$$\text{clock rate}_B = \frac{\text{clock cycles}_B}{\text{CPU time}_B} = \frac{1.2 \times \text{clock cycles}_A}{6s}$$
$$\text{clock rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4GHz$$

Instruction count and CPI

- Instruction Count for a program
  - previous formulas do not consider number of instructions
  - instructions are determined by program, ISA and compiler
- Average cycles per instruction (CPI)
  - average of clock cycles each instruction takes to execute, over all the instructions executed by the program
  - determined by the CPU hardware

$$\text{CPU clock cycles} = \text{instruction count} \times \text{CPI}$$
$$\text{CPU time} = \text{instruction count} \times \text{CPI} \times \text{clock cycle time}$$
$$= \frac{\text{instruction count} \times \text{CPI}}{\text{clock rate}}$$

Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA, which computer is **faster**, and **by how much**?

$$\text{CPU time}_A = \text{ic} \times \text{CPI}_A \times \text{cycle time}_A$$
$$= ic \times 2.0 \times 250ps = ic \times 500ps$$
$$\text{CPU time}_B = \text{ic} \times \text{CPI}_B \times \text{cycle time}_B$$
$$= ic \times 1.2 \times 500ps = ic \times 600ps$$
$$\frac{\text{CPU time}_B}{\text{CPU time}_A} = \frac{ic \times 600ps}{ic \times 500ps} = 1.2$$

CPI in more detail

- If different instruction classes take different numbers of cycles

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{instruction count}_i)$$

- Calculating the (average) CPI using a weighted average

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{instruction count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\text{instruction count}_i}{\text{instruction count}} \right)$$

CPI example

- Alternative compiled code sequences using instructions in classes A, B, C. Which code sequence executes the most instructions? Which sequence is 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
  - clock cycles = 2×1 + 1×2 + 2×3 = 10
  - average CPI = 10/5 = 2.0
- Sequence 2
  - clock cycles = 4×1 + 1×2 + 1×3 = 9
  - average CPI = 9/6 = 1.5

Quizz

- Consider 3 different processors executing the same instructions
  - P1 has a clock cycle time of 0.33ns and a CPI of 1.5
  - P2 has a clock cycle time of 0.40ns and a CPI of 1
  - P3 has a clock cycle time of 0.25ns and a CPI of 2.2
- Questions
  - which has the highest clock rate? what is it?
  - which is the fastest computer? which is the slowest?

Quizz

- Case 1:
  - a program takes 4 billion instructions
  - each instruction completes in an average of 1.5 cycles
  - clock speed is 1 GHz
- Case 2:
  - a program takes 2 billion instructions
  - each instruction completes in an average of 6 cycles
  - clock speed is 1.5 GHz
- Which one is better?

Performance summary

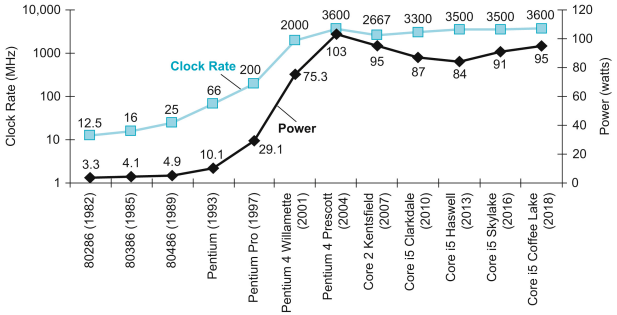
- Performance depends on:
  - **algorithm**: affects IC, possibly CPI
  - **programming language**: affects IC, CPI
  - **compiler**: affects IC, CPI
  - **ISA**: affects IC, CPI, T_c

CPU time = instruction count × CPI × clock cycle time

The Power Wall

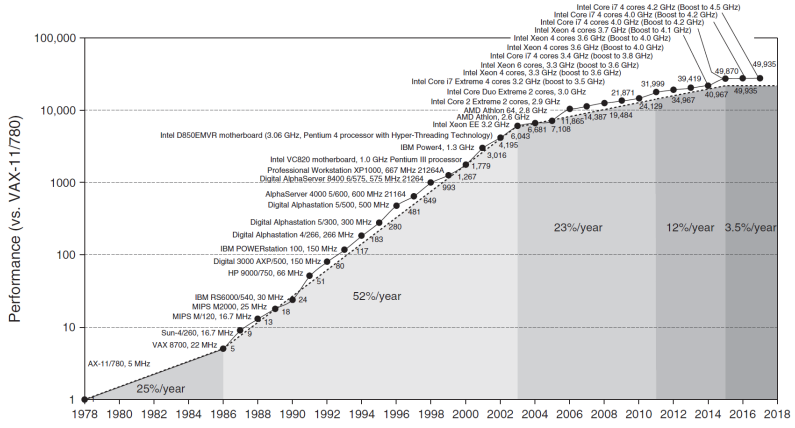
Power trends

- Clock rate and power are correlated
  - Power $\propto 1/2 \times$ capacitive load $\times$ voltage$^2 \times$ frequency
- We have run into a practical limit for cooling



Uniprocessor performance (response time)

- Constrained by power, instruction-level parallelism, and memory latency



Multiprocessors

- Multicore microprocessors
  - more than one processor 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

The multicore effect

processors	exec. time/ processor	time w/overhead	speedup	actual speedup/ideal speedup
1	100			
2	50	54	100/54 = 1.85	1.85/2 = .93
4	25	29	100/29 = 3.44	3.44/4 = 0.86
8	12.5	16.5	100/16.5 = 6.06	6.06/8 = 0.75
16	6.25	10.25	100/10.25 = 9.76	9.76/16 = 0.61

Benchmarking Processors

https://www.spec.org/

SPEC CPU 2017 Basics

Q6. What does SPEC CPU 2017 measure?

SPEC CPU 2017 focuses on compute intensive performance, which means these benchmarks emphasize the performance of:

- **Processor** - The CPU chip(s).
- **Memory** - The memory hierarchy, including caches and main memory.
- **Compilers** - C, C++, and Fortran compilers, including optimizers.

SPEC CPU 2017 intentionally depends on all three of the above - not just the processor.

SPEC CPU 2017 is not intended to stress other computer components such as networking, graphics, Java libraries, or the I/O system. Note that there are other SPEC benchmarks that focus on those areas.

Q8. What does SPEC provide?

SPEC CPU 2017 is distributed as an ISO image that contains:

- Source code for the benchmarks
- Data sets
- A tool set for compiling, running, validating and reporting on the benchmarks
- Pre-compiled tools for a variety of operating systems
- Source code for the SPEC CPU 2017 tools, for systems not covered by the pre-compiled tools
- Documentation
- Run and reporting rules

The documentation is also available at www.spec.org/cpu2017/Docs/index.html, including the Unix and Windows installation guides.

Q12. What is a CPU 2017 "suite"?

A *suite* is a set of benchmarks that are run as a group to produce one of the overall metrics.

SPEC CPU 2017 includes four suites that focus on different types of compute intensive performance:

Short Tag	Suite	Contents	Metrics	How many copies? What do Higher Scores Mean?
`intspeed`	SPECspeed®2017 Integer	10 integer benchmarks	`SPECspeed2017_int_base` `SPECspeed2017_int_peak`	SPECspeed suites always run one copy of each benchmark. Higher scores indicate that less time is needed.
`fpspeed`	SPECspeed®2017 Floating Point	10 floating point benchmarks	`SPECspeed2017_fp_base` `SPECspeed2017_fp_peak`	
`intrate`	SPECrate®2017 Integer	10 integer benchmarks	`SPECrate2017_int_base` `SPECrate2017_int_peak`	SPECrate suites run multiple concurrent copies of each benchmark. The tester selects how many. Higher scores indicate more *throughput* (work per unit of time)
`fprate`	SPECrate®2017 Floating Point	13 floating point benchmarks	`SPECrate2017_fp_base` `SPECrate2017_fp_peak`	

Q15. What are "SPECspeed" and "SPECrate" metrics?

There are many ways to measure computer performance. Among the most common are:

- Time - For example, seconds to complete a workload.
- Throughput - Work completed per unit of time, for example, jobs per hour.

SPECspeed is a time-based metric; SPECrate is a throughput metric.

- The *base* metrics (such as SPECspeed2017_int_base) require that all modules of a given language in a suite must be compiled using the same flags, in the same order. All reported results must include the base metric.
- The optional *peak* metrics (such as SPECspeed2017_int_peak) allow greater flexibility. Different compiler options may be used for each benchmark, and feedback-directed optimization is allowed.

SPECspeed 2017 Integer benchmarks

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

GeoMean

Running on a 1.8 GHz Intel Xeon E5-2650L

Geometric mean

- Summaries as **geometric mean** of performance ratios

$$\sqrt[n]{\prod_{i=1}^{n} \text{execution time ratio}_i}$$

- Always summarize relative performance with the **geometric mean**
  - not affected by the baseline performance

Summarizing performance

- Summaries as **geometric mean** of performance ratios (speedup)
  - not affected by the baseline performance

$$\sqrt[n]{\prod_{i=1}^{n} \text{execution time ratio}_i}$$

- Summaries as **arithmetic mean** of execution times

$$\frac{1}{n}\sum_{i=1}^{n} \text{execution time}_i$$

SPECpower

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

SPECpower_ssj2008 running on a dual socket 2.2 GHz Intel Xeon Platinum 8276L with 192 GiB of DRAM and one 80 GB SSD disk.

SPEC CPU2017 Results

These results have been submitted to SPEC; see the disclaimer before studying any results.

Available Results

Browse

The following are sets of available results since the announcement of the benchmark in June 2017.

- All CPU2017 Results

Results from all publication quarters, broken out by reported metric:

- Speed:
  [SPECspeed 2017 Integer, SPECspeed 2017 Floating Point]
- Throughput:
  [SPECrate 2017 Integer, SPECrate 2017 Floating Point]

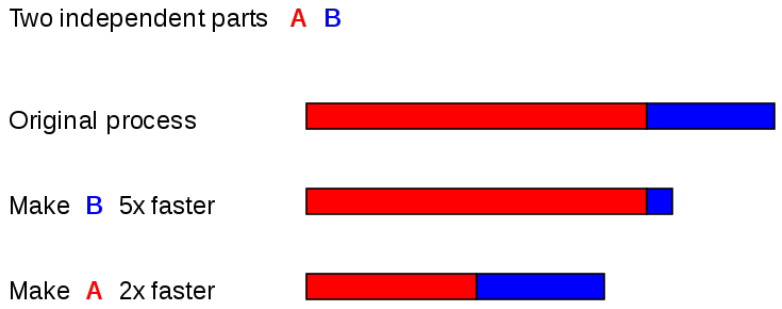
https://www.spec.org/cpu2017/results/

Pitfalls and Myths

Amdahl's law (pitfall)

- Amdahl's Law
  - "the **overall performance** improvement gained by **optimizing a single part** of a system is **limited by the fraction of time** that the improved part is actually used"
    - improving an aspect of a computer does not translate automatically into a proportional improvement in overall performance
- Architecture design is bottleneck-driven
  - **make the common case fast**
  - do not waste resources on a component that has little impact on overall performance/power

Which enhancement is better?



https://en.wikipedia.org/wiki/Amdahl%27s_law

Amdahl's law

- Assume:
  - S: overall system speedup, F: fraction enhanced, I: factor of improvement

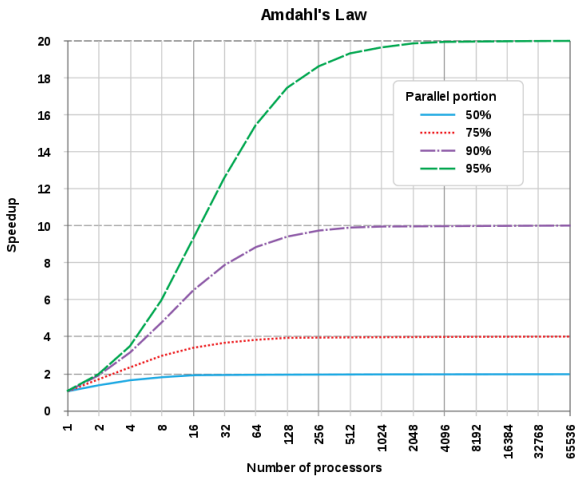
$$S = \frac{1}{\underbrace{(1-F)}_{\text{non-enhanced}} + \underbrace{\frac{F}{I}}_{\text{enhanced}}}$$

- Example
  - 40% of instructions in program A are multiplications, and a new processor runs multiplications faster by a factor of 3.4. What is the overall system speedup?

Amdahl's law (example)

- A program spends 40% of time in the CPU and 60% doing I/O
  - a new processor **10x faster** results in a reduction in total execution time (what is the speedup?)
- Amdahl's Law states that the **maximum speedup** will be 1.66)

Amdahl's law and parallel computing



https://en.wikipedia.org/wiki/Amdahl%27s_law

Low power at idle (fallacy)

- Look back at Xeon's SPECpower benchmark
  - at 100% load: 347W
  - at 50% load: 183W (52.7%)
  - at 10% load: 115W (33.1%)
- Google data center
  - mostly operates at 10% – 50% load
  - at 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

Datacenter tour

https://www.youtube.com/watch?v=80aK2_iwMOs

MIPS as a Performance Metric (pitfall)

- **MIPS**: Millions of Instructions Per Second
  - doesn't account for differences in ISAs between computers, nor differences in complexity between instructions

$$\text{MIPS} = \frac{\text{instruction count}}{\text{execution time} \times 10^6}$$

$$= \frac{\text{instruction count}}{\frac{\text{instruction count} \times \text{CPI}}{\text{clock rate}} \times 10^6}$$

$$= \frac{\text{clock rate}}{\text{CPI} \times 10^6}$$

CPI varies between programs on a given CPU