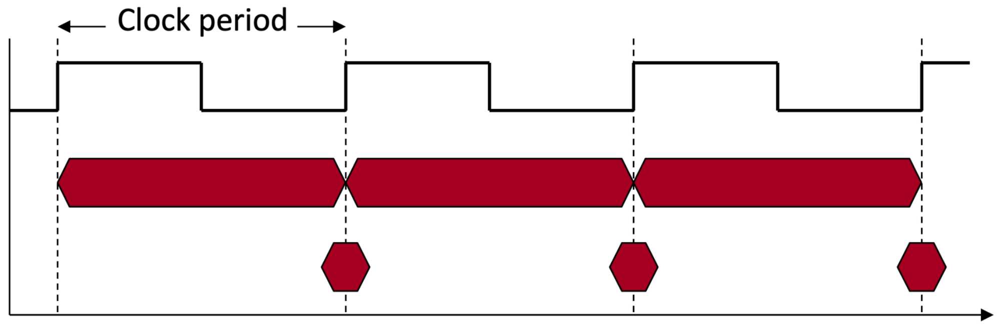
CMSC 411

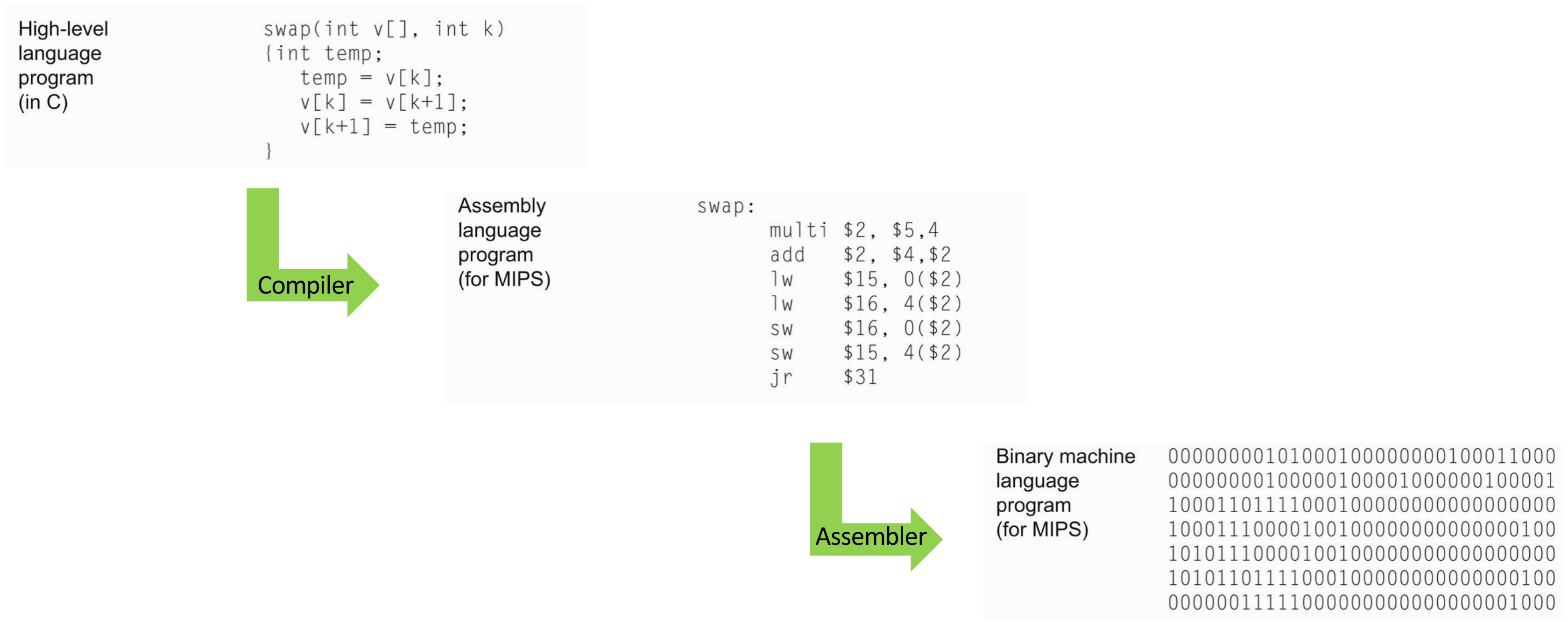
Performance (Cont…), Benchmarks, and Performance Pitfalls

Ergun Simsek

Fall 2022 - Lecture 3

Last Week





High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Assembly language program (for MIPS)

```
swap:
      multi $2, $5,4
      add   $2, $4,$2
      lw    $15, 0($2)
      lw    $16, 4($2)
      sw    $16, 0($2)
      sw    $15, 4($2)
      jr    $31
```

Binary machine language program (for MIPS)

```
00000000101000100000000100011000
00000000100000100001000000100001
10001101111000100000000000000000
10001110000100100000000000000100
10101110000100100000000000000000
10101101111000100000000000000100
00000011111000000000000000001000
```

Instruction Count and CPI

Last week, we said that we always want to improve the performance

Instruction Count for a program

- Determined by
  - Program
  - Instruction set architecture (ISA)
  - Compiler

Average cycles per instruction ("CPI")

- Determined by CPU hardware
- If different instructions have different CPI
  - Average CPI affected by instruction mix

Program Clock Cycles

Instead of reporting execution time in seconds, we often use clock cycle counts

- Why? A newer generation of the same processor…
  - Often has the same cycle counts for the same program
  - But often has different clock speed (ex, 1 GHz changes to 1.5 GHz)

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

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

Instruction Count and CPI

Some operations (e.g., division) require longer time to execute

Longer time = A higher number of cycles

CPI: Average number of cycles per instruction

$$\text{Clock Cycles} = \text{Instruction Count} \times \underline{\text{Cycles per Instruction}}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

CPI Examples

PDP-11, VAX, Intel 8086:	CPI > 1
Load/Store RISC machines	CPI = 1
• MIPS, SPARC, PowerPC, miniMIPS	
Modern CPUs	CPI < 1
• Pentium4	0.33
• Xeon	0.25
• M2	?

How to Improve Performance?

Many ways to write the same equations:

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

So, to improve performance (everything else being equal) you can either

- Decrease the # of required cycles for a program;
- Decrease the clock cycle time or, said another way,
- Increase the clock rate;
- Decrease the CPI (average clocks per instruction).

Example

- Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock.
- We are trying to help a computer designer build a new machine B, to run this program in 6 seconds.
- The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program.
- What clock rate should we tell the designer to target?

Computer A runs the code using 10*2e9 cycles.

Computer B will require 1.2 times more, so it will use 1.2* 10*2e9 cycles.

1.2* 10*2e9 = 6 * frequency_of_the_new_CPU ➔ it has t be 4 GHz

Now that we understand cycles

A given program will require

- some number of instructions (machine instructions)
- some number of cycles
- some number of seconds

We have a vocabulary that relates these quantities:

- cycle time (seconds per cycle)
- clock rate (cycles per second)
- CPI (average clocks per instruction)
    - ➢ a floating point intensive application might have a higher CPI

Performance Traps

Performance is determined by the execution time of a program that you care about.

Do any of the other variables equal performance?

- # of cycles to execute program?
- # of instructions in program?
- # of cycles per second?
- average # of cycles per instruction?
- average # of instructions per second?

Common pitfall:

- Thinking that only one of the variables is indicative of performance when it really is not!

CPI Example

Suppose we have two implementations of the same instruction set architecture (ISA) on two computers and we are running the same program:

- Computer A has a clock cycle time of 250 ps and a CPI of 2.0
- Computer B has a clock cycle time of 500 ps and a CPI of 1.2

1. Which quantity (e.g., clock rate, CPI) is the same for two implementations?

   Instruction Count

2. What machine is faster for this program, and by how much?

$$\begin{aligned} Time_A &= InstructionCount * CPI_A * CycleTime_A \\ &= IC * 2.0 * 250\,ps = IC * 500\,ps \end{aligned}$$

$$\begin{aligned} Time_B &= InstructionCount * CPI_B * CycleTime_B \\ &= IC * 1.2 * 500\,ps = IC * 600\,ps \end{aligned}$$

$$\text{Relative Performance} = \frac{Time_B}{Time_A} = \frac{600}{500} = 1.2$$

A is faster by 1.2 X

CPI in More Detail

If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

Weighted average CPI:

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\boxed{\textbf{Relative frequency}}} \right)$$

Example: Compiler's Impact

- Two different compilers are being tested for a 500 MHz machine with three different classes of instructions: Class A, Class B, and Class C, which require 1, 2, and 3 cycles (respectively).
- Both compilers are used to produce code for a large piece of software.
- The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 2 million Class C instructions.
- The second compiler's code uses 7 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.
- Which program uses fewer instructions?
  - $\text{Instructions}_1 = (5+1+2) \times 10^6 = 8 \times 10^6$
  - $\text{Instructions}_2 = (7+1+1) \times 10^6 = 9 \times 10^6$

Which sequence uses fewer clock cycles?

- $\text{Cycles}_1 = (5(1)+1(2)+2(3)) \times 10^6 = 13 \times 10^6$
- $\text{Cycles}_2 = (7(1)+1(2)+1(3)) \times 10^6 = 12 \times 10^6$

$\text{CPI}_1 = ?\ \ 13/8 = 1.625$

$\text{CPI}_2 = ?\ \ 12/9 = 1.33$

CPI Example

Alternative compiled code versions using instructions in classes A, B, C. Which version has a lower CPI?

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

Version 1: IC = 5

Clock Cycles
= 2×1 + 1×2 + 2×3
= 10

Avg. CPI = 10/5 = 2.0

Version 2: IC = 6

Clock Cycles
= 4×1 + 1×2 + 1×3
= 9

Avg. CPI = 9/6 = 1.5

Performance Summary

The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Performance depends on

- Algorithm: affects IC, possibly CPI
- Programming language: affects IC, CPI
- Compiler: affects IC, CPI
- Instruction set architecture: affects IC, CPI, Cycle Time

Benchmarks

Performance best determined by running a real application

- Use programs typical of expected workload
- Or, typical of expected class of applications
  - ➢ e.g., compilers/editors, scientific applications, graphics, etc.

Small benchmarks

- nice for architects and designers
- easy to standardize
- can be abused

SPEC (System Performance Evaluation Cooperative)

- companies have agreed on a set of real program and inputs
- can still be abused
- valuable indicator of performance (and compiler technology)

https://www.spec.org/

SPEC 2017

throughput

time

Kilo Lines of Code

Integers

SPECrate®2017 Integer	SPECspeed®2017 Integer	Language [1]	KLOC [2]	Application Area
500.perlbench_r	600.perlbench_s	C	362	Perl interpreter
502.gcc_r	602.gcc_s	C	1,304	GNU C compiler
505.mcf_r	605.mcf_s	C	3	Route planning
520.omnetpp_r	620.omnetpp_s	C++	134	Discrete Event simulation - computer network
523.xalancbmk_r	623.xalancbmk_s	C++	520	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	C	96	Video compression
531.deepsjeng_r	631.deepsjeng_s	C++	10	Artificial Intelligence: alpha-beta tree search (Chess)
541.leela_r	641.leela_s	C++	21	Artificial Intelligence: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	Fortran	1	Artificial Intelligence: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	C	33	General data compression

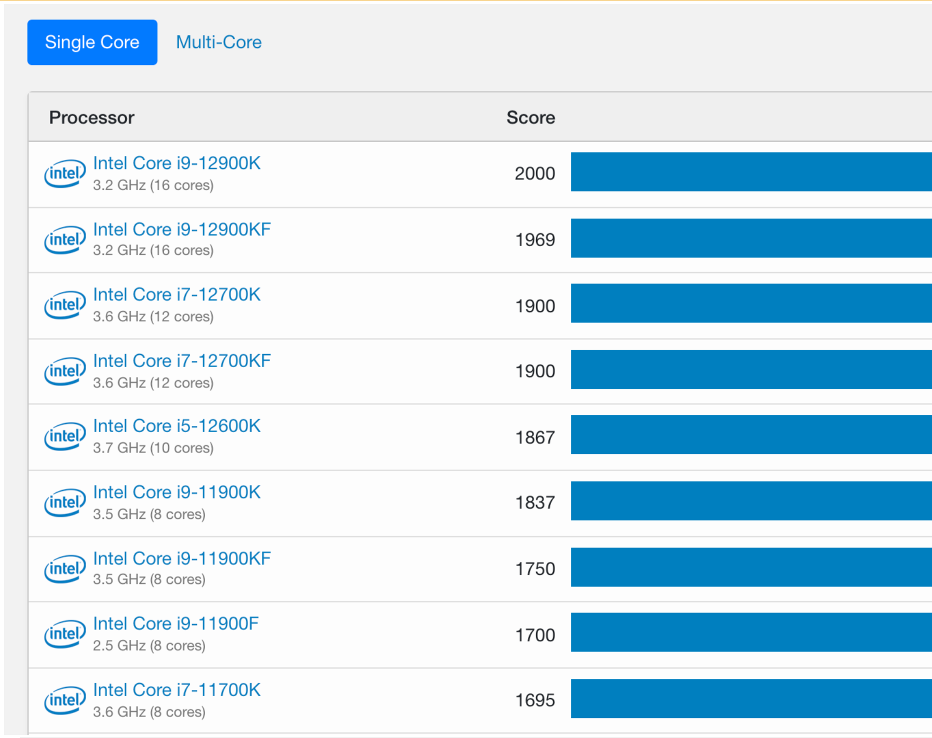
Floating Point

SPECrate®2017 Floating Point	SPECspeed®2017 Floating Point	Language [1]	KLOC [2]	Application Area
503.bwaves_r	603.bwaves_s	Fortran	1	Explosion modeling
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	257	Physics: relativity
508.namd_r		C++	8	Molecular dynamics
510.parest_r		C++	427	Biomedical imaging: optical tomography with finite elements
511.povray_r		C++, C	170	Ray tracing
519.lbm_r	619.lbm_s	C	1	Fluid dynamics
521.wrf_r	621.wrf_s	Fortran, C	991	Weather forecasting
526.blender_r		C++, C	1,577	3D rendering and animation
527.cam4_r	627.cam4_s	Fortran, C	407	Atmosphere modeling
	628.pop2_s	Fortran, C	338	Wide-scale ocean modeling (climate level)
538.imagick_r	638.imagick_s	C	259	Image manipulation
544.nab_r	644.nab_s	C	24	Molecular dynamics
549.fotonik3d_r	649.fotonik3d_s	Fortran	14	Computational Electromagnetics
554.roms_r	654.roms_s	Fortran	210	Regional ocean modeling

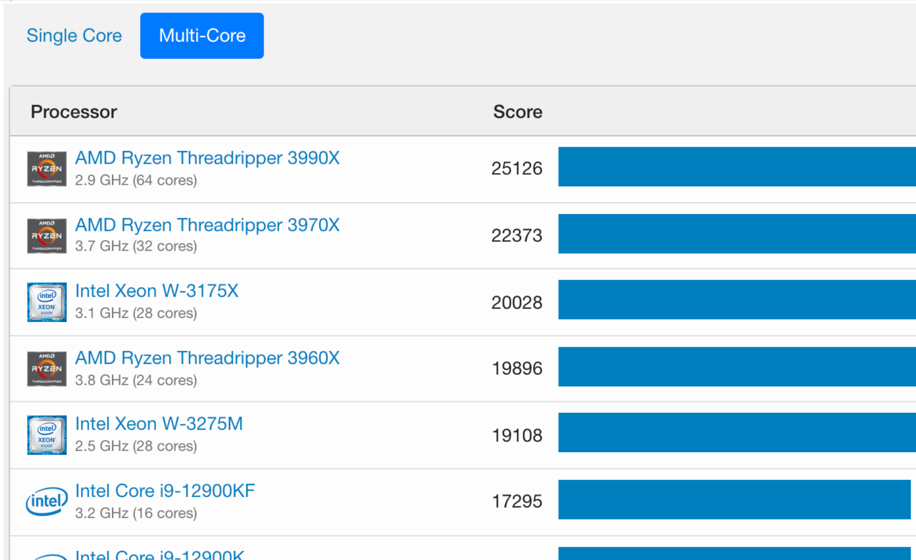
Other Popular Benchmarks

Several others popular

- industry uses SPEC
- but ordinary consumers use others
  - ➢ more representative of the work they do!
    - – e.g., gaming, Photoshop/Aperture, copying huge files, multimedia coding/decoding, etc.
  - ➢ Geekbench is quite popular!



Processor	Score
Intel Core i9-12900K 3.2 GHz (16 cores)	2000
Intel Core i9-12900KF 3.2 GHz (16 cores)	1969
Intel Core i7-12700K 3.6 GHz (12 cores)	1900
Intel Core i7-12700KF 3.6 GHz (12 cores)	1900
Intel Core i5-12600K 3.7 GHz (10 cores)	1867
Intel Core i9-11900K 3.5 GHz (8 cores)	1837
Intel Core i9-11900KF 3.5 GHz (8 cores)	1750
Intel Core i9-11900F 2.5 GHz (8 cores)	1700
Intel Core i7-11700K 3.6 GHz (8 cores)	1695



Processor	Score
AMD Ryzen Threadripper 3990X 2.9 GHz (64 cores)	25126
AMD Ryzen Threadripper 3970X 3.7 GHz (32 cores)	22373
Intel Xeon W-3175X 3.1 GHz (28 cores)	20028
AMD Ryzen Threadripper 3960X 3.8 GHz (24 cores)	19896
Intel Xeon W-3275M 2.5 GHz (28 cores)	19108
Intel Core i9-12900KF 3.2 GHz (16 cores)	17295
Intel Core i9-12900K	

Fallacies and pitfalls

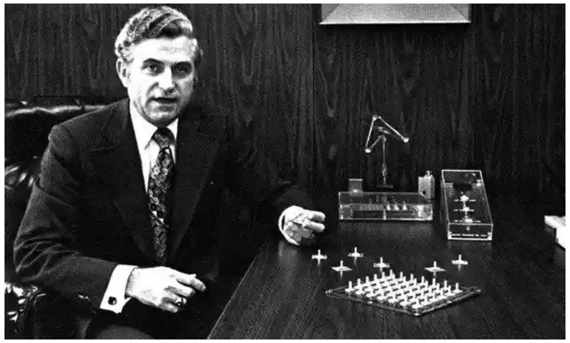
1. The improvement of one aspect of a computer to increase overall performance by an amount proportional to the size of the improvement.
2. Computers at low utilization use little power
3. Designing for performance and designing for energy efficiency are unrelated goals
4. Using a subset of the performance equation is a better performance metric

W R O N G

Amdahl's Law

Possibly the most important law regarding computer performance:

$$t_{improved} = \frac{t_{affected}}{r_{speedup}} + t_{unaffected}$$



Gene Amdahl (1973)

- Principle: <u>Make the common case fast!</u>
- Eventually, performance gains will be limited by what cannot be improved
  - e.g., you can raise the highway speed limit, but the city speed limit stays the same

Amdahl’s Law: Example

$$t_{improved} = \frac{t_{affected}}{r_{speedup}} + t_{unaffected}$$

Suppose a program runs in 100 seconds on a machine, where multiplies are executed 80% of the time.

How much do we need to improve the speed of multiplication if we want the program to run 4 times faster?

$25 = 80/r + 20$ $\quad r = 16x$

How about making it 5 times faster?

$20 = 80/r + 20$ $\quad r = ?$

Example

Suppose we enhance a machine making all floating-point instructions run FIVE times faster.

If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if only half of the 10 seconds is spent executing floating-point instructions?

$5/5 + 5 = 6$ Relative Perf $= 10/6 = 1.67$ x

We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show at least a speedup of 3.

What percentage of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

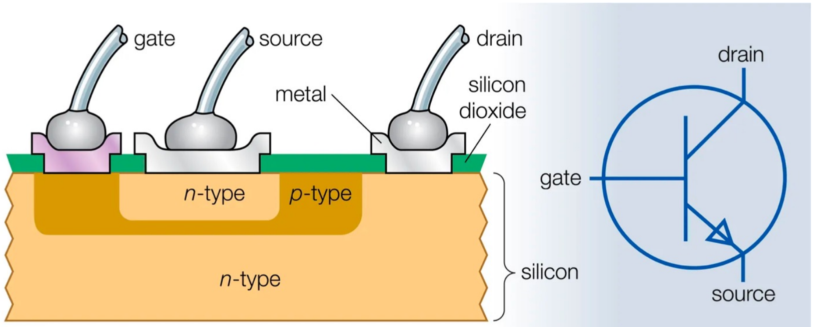
$10/3 = f/5 + (10 - f) = 10 - 4f/5$ $f = 83.33$

Power Consumption

Power = Energy consumed per unit time

Two contributors to power consumption

- Dynamic power
  - power consumed when doing actual work
  - called dynamic because components and wires are switching between '0' and '1'
- Static or 'leakage' power
  - power consumed even when everything is idle or 'static'
  - due to some small amount of 'leakage' current that still flows



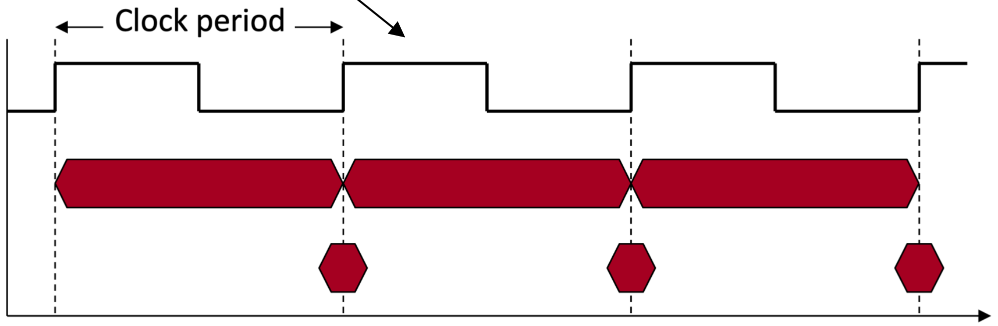
Dynamic Power Consumption

Energy consumed due to switching activity:

- All wires and transistor gates have capacitance
- Energy required to charge a capacitance, *C*, to V_{DD} is CV_{DD}^2
- Circuit running at frequency *f*: transistors switch (from 1 to 0 or vice versa) at that frequency
- Capacitor is charged 0.5*f* times per second

$$P_{dynamic} = ½CV_{DD}^2f$$

C is the total capacitance of circuit ("capacitive load")
V_{DD} is the supply voltage
f is the switching frequency



Static Power Consumption

Power consumed when no gates are switching

- Caused by the *quiescent supply current*, I_{DD} (also called the *leakage current*)

$$P_{static} \text{ or } P_{leakage} = I_{DD}V_{DD}$$

V_{DD} is the supply voltage
I_{DD} is the leakage current

Power Consumption Example

Estimate the power consumption of a wireless handheld computer

- V_{DD} = 1.2 V
- C = 20 nF
- f = 1 GHz
- I_{DD} = 20 mA

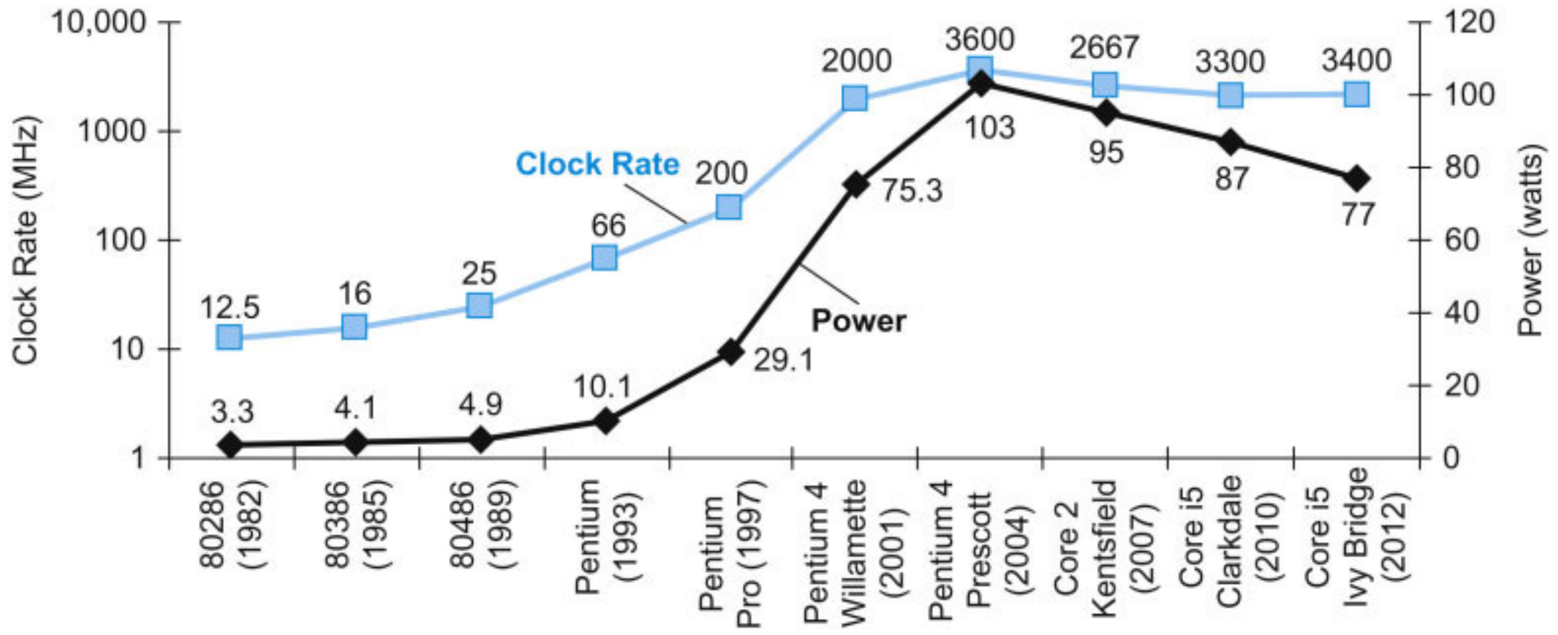
$P = \frac{1}{2} C V_{DD}^2 f + I_{DD} V_{DD}$

$= \frac{1}{2}(20\text{ nF})(1.2\text{ V})^2(1\text{ GHz}) + (20\text{ mA})(1.2\text{ V})$

$= \mathbf{14.4\ W} + \mathbf{24\ mW}$

$= \mathbf{14.424\ W}$

Power Trends



In CMOS IC technology

$$\text{Dynamic Power} = \frac{1}{2}\underbrace{\text{Capacitive load}}_{\boxed{\times 30}} \times \underbrace{\text{Voltage}^2}_{\boxed{5V \rightarrow 1V}} \times \underbrace{\text{Frequency}}_{\boxed{\times 1000}}$$

Reducing Power

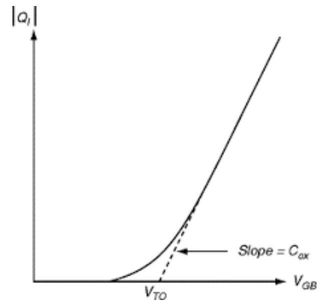
How power reduction is achieved if we replace the old CPU wit a new one which has

- 85% of capacitive load of old CPU
- 15% voltage and 15% frequency reduction

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

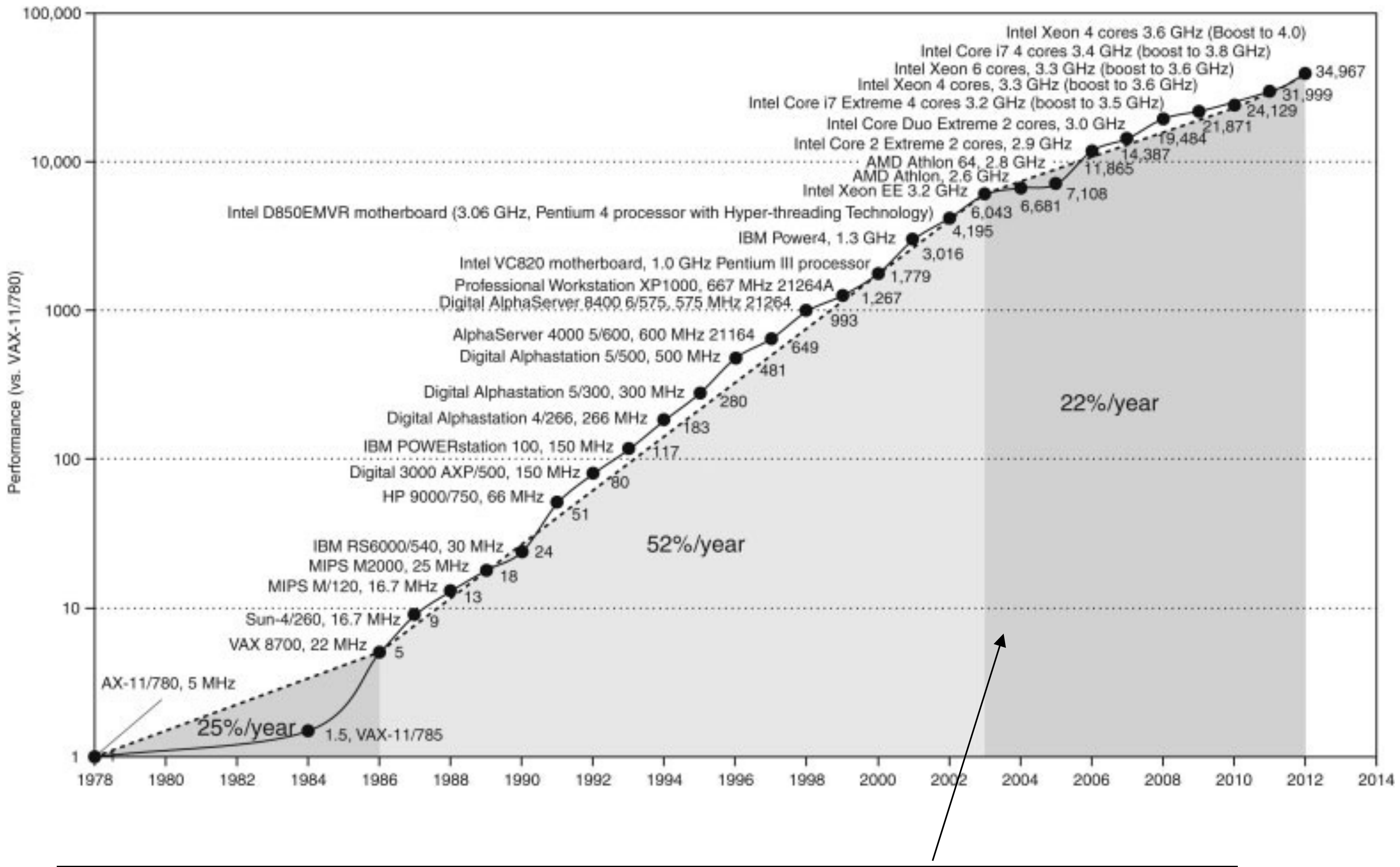
The power wall

- We cannot reduce voltage further



Threshold Voltage near Leakage Limit

Moore's Law: Uniprocessor Perf.



Constrained by power, instruction-level parallelism, memory latency

Fallacies and pitfalls

1. The improvement of one aspect of a computer to increase overall performance by an amount proportional to the size of the improvement.
2. Computers at low utilization use little power
3. Designing for performance and designing for energy efficiency are unrelated goals
4. Using a subset of the performance equation is a better performance metric

W
R
O
N
G

Fallacy: Low Power at Idle

AMD X4 power benchmark:

- At 100% load: 295W (max power)
- At 50% load: 246W (83% max power)
- At 10% load: 180W (still consumes 61% of max power)

Google data center

- Mostly operates at 10% - 50% load
- At 100% load less than 1% of the time

Industry challenge: Design processors to make power proportional to load

Fallacies and pitfalls

1. The improvement of one aspect of a computer to increase overall performance by an amount proportional to the size of the improvement.
2. Computers at low utilization use little power
3. Designing for performance and designing for energy efficiency are unrelated goals
4. Using a subset of the performance equation is a better performance metric

W R O N G

Multiprocessors

"Multicore" microprocessors

- More than one processor core per chip

Requires explicitly parallel programming

- Hardware executes multiple instructions at once
  - Ideally, hidden from the programmer
- Hard to do
  - Programming for performance
  - Load balancing
  - Optimizing communication and synchronization
  - But, newer OSs and libraries have been designed for this

Fallacies and pitfalls

1. The improvement of one aspect of a computer to increase overall performance by an amount proportional to the size of the improvement.
2. Computers at low utilization use little power
3. Designing for performance and designing for energy efficiency are unrelated goals
4. Using a subset of the performance equation is a better performance metric

W R O N G

A Useless Computer Performance Measure: MIPS

Millions of Instructions per Second

Frequency in MHz

$$MIPS = \frac{\text{clocks/sec}}{\text{AVE(clocks/instruction)}}$$

Unfortunate coincidence:
This "MIPS" has nothing to do with the name of the MIPS processor we will be studying!

CPI (Average Clocks Per Instruction)

Pitfall: Cannot compare MIPS of two different processors if they run different sets of instructions!

➔ **Meaningless Indicator of Processor Speed!**

Remember

Performance is specific to a particular program

- Total execution time is a consistent summary of performance

For a given architecture, the performance comes from:

- increases in clock rate (without adverse CPI affects)
- improvements in processor organization that lower CPI
- compiler enhancements that lower CPI and/or instruction count

Improvements in one aspect of a machine's performance do not bring linear improvement in the total performance

Power is a limiting factor

- Use parallelism to improve performance