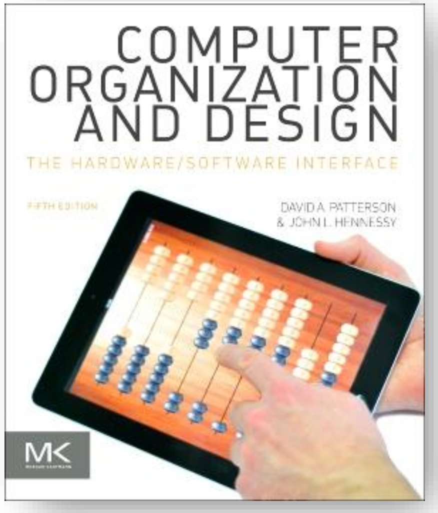
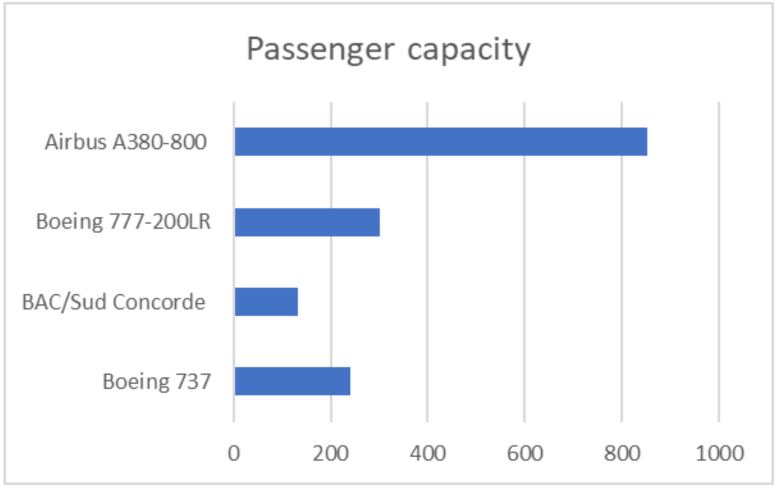

COMPUTER ORGANIZATION AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
FIFTH EDITION
DAVID A. PATTERSON & JOHN L. HENNESSY
MK

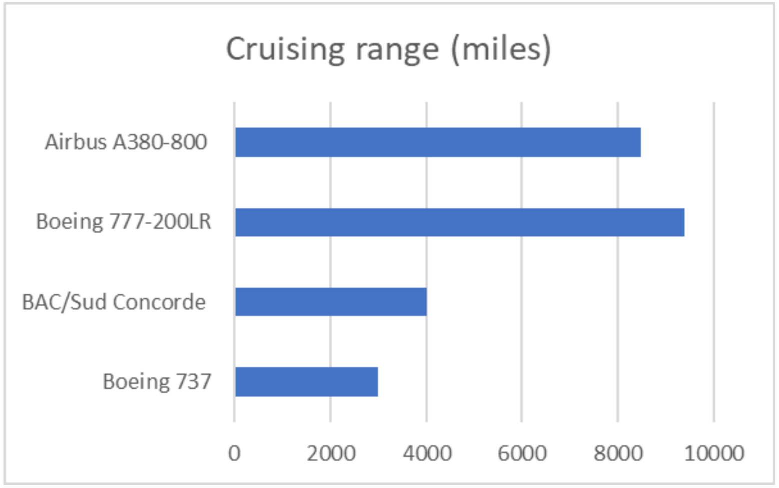
ESE 345 Computer Architecture

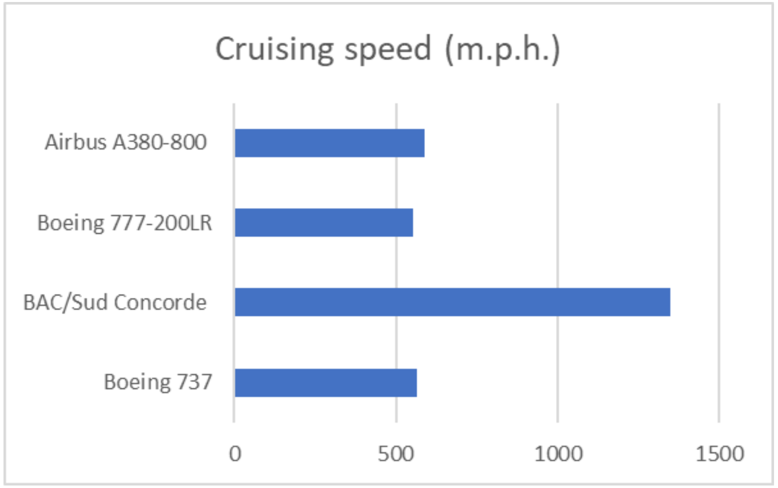
Performance and Energy Consumption

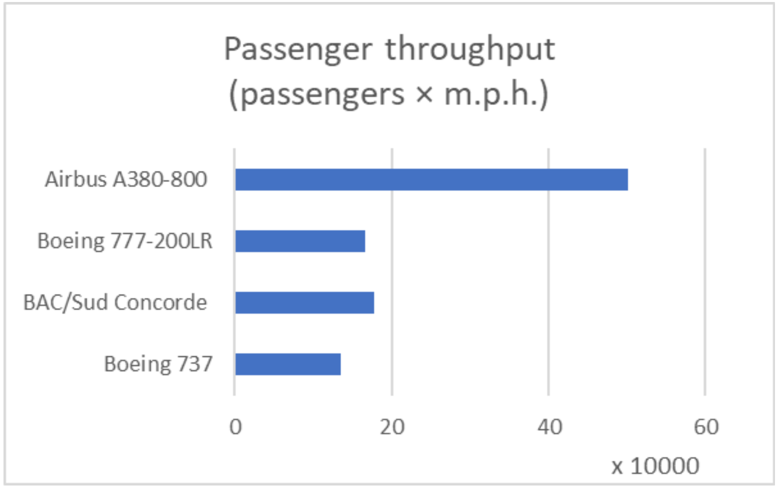
Defining Performance

- Which airplane has the best performance?









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
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

Relative Performance

- Define Performance = 1/Execution Time
- "X is n time faster than Y"

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

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A
    = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

Analyze the Right Measurement!

Guides CPU design

CPU Time:

Time the CPU spends running program under measurement.

Measuring CPU time (Unix):

```
% time <program name>
25.77u  0.72s  0:29.17 90.8%
```

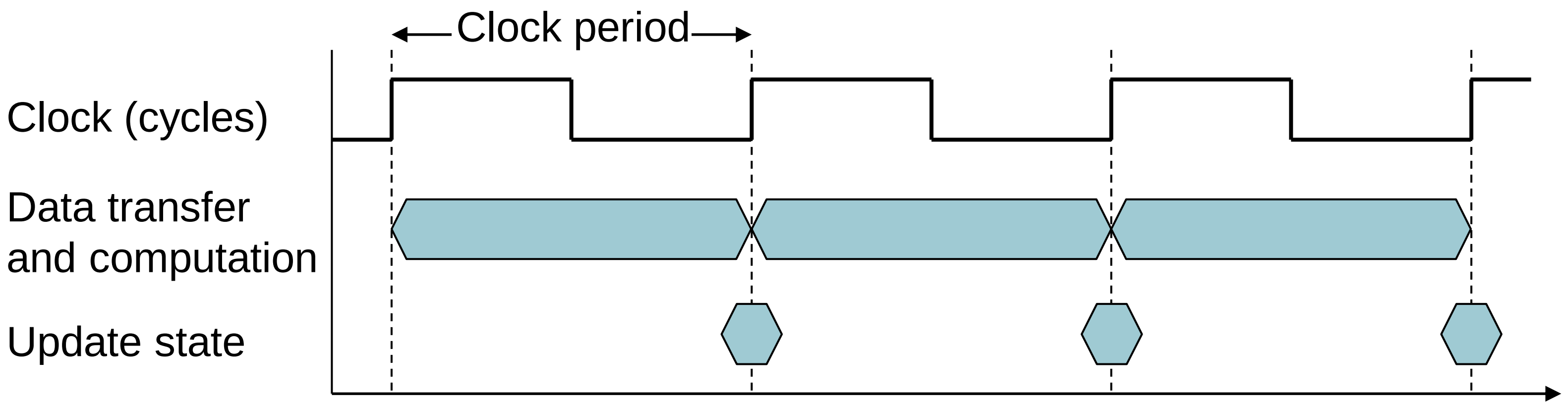
Response Time:

Guides system design

Total time: CPU Time + time spent waiting (for disk, I/O, ...).

CPU Clocking

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



  - Clock frequency (rate) = 1/Clock period
  - 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 = 1/250ps

CPU 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

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

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

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

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

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

Instruction Count and CPI

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

$$\begin{aligned}\text{CPU Time} &= \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time} \\ &= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}\end{aligned}$$

- Instruction Count (IC) for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction (CPI)
  - Depends on program, CPU hardware and compiler
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

CPI Example

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

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

$$= \text{I} \times 2.0 \times 250\text{ps} = \text{I} \times 500\text{ps}$$

A is faster…

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

$$= \text{I} \times 1.2 \times 500\text{ps} = \text{I} \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{\text{I} \times 600\text{ps}}{\text{I} \times 500\text{ps}} = 1.2$$

…by this much

How Calculate the 3 Components?

- Clock Cycle Time: in specification of computer (Clock Rate in advertisements)
- Instruction Count:
  - Count instructions in loop of small program
  - Use simulator to count instructions
  - Hardware counter in spec. register (most CPUs)

° CPI:

- Calculate: $\dfrac{\text{Execution Time / Clock cycle time}}{\text{Instruction Count}}$
- Hardware counter in special register (most CPUs)

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

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

Calculating Average CPI

- First find *CPI_i* for each individual instruction (**`add`, `sub`, `and`**, etc.)
- Next use (when it's given) or calculate relative frequency *f_i* of each individual instruction

$$f_i = ICi/IC$$

- Finally multiply these two for each instruction and add them up to get final CPI

$$CPI = \sum_{i=1}^{n} f_i * CPIi$$

Example with Bonus Points to Earn!

Op	$Freq_i$	CPI_i	Prod	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	15%	2	.3	(20%)
Branch	15%	2	.3	(20%)
			1.5	

Instruction Mix

(Where time spent)

- What if you can make branch instructions twice as fast (CPI_{br}= 1 cycle) but clock rate (CR) will decrease by 12%? Will it be a speedup or slowdown and how much?
- The 1st person who gives the right answer gets bonus points.

New CPI = 1.35

Time before change = IC*1.5 / CR = IC*1.5/CR

Time after change = IC*1.35 / (0.88*CR) = IC*1.534/CR

Time before/Time after = IC*1.5*CR / IC*1.534*CR = 0.978 => 2.2% slowdown

Speedup < 1 because the time after is greater than the time before the change

Another CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

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
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

CPU Performance Law

The BIG Picture

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

$$\begin{aligned}\text{CPU Time} &= \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time} \\ &= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}\end{aligned}$$

- Performance depends on
  - **Algorithm**: affects IC, possibly CPI
  - **Programming language**: affects IC, CPI
  - **Compiler**: affects IC, CPI
  - **Instruction set architecture**: affects IC, CPI, T_c
  - **Processor microarchitecture (organization)**: affects CPI, T_c
  - **Technology**: affects T_c

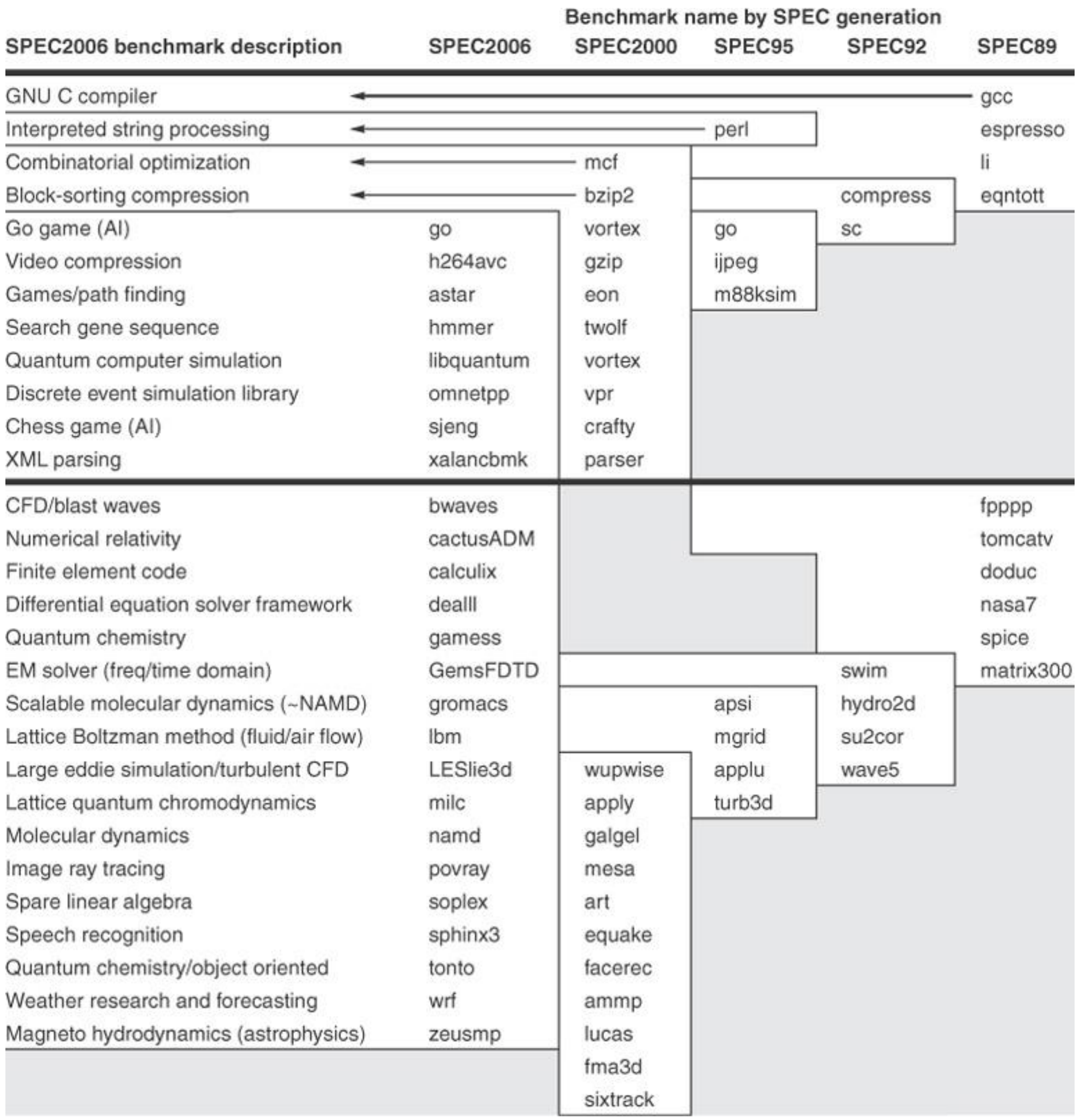
What Programs Measure for Comparison?

- Ideally run typical programs with typical input before purchase,
  or before even build machine
  - Called a "workload"; For example:
  - Engineer uses compiler, spreadsheet
  - Author uses word processor, drawing program, compression software
- In some situations it's hard to do
  - Don't have access to machine to "benchmark" before purchase
  - Don't know workload in future

SPEC CPU Benchmarks

- Standard Performance Evaluation Corporation (SPEC) www.spec.org
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - SPEC95: 8 integer (gcc, compress, li, ijpeg, perl, ...) & 10 floating-point (FP) programs (hydro2d, mgrid, applu, turbo3d, ...)
  - SPEC2000: 11 integer (gcc, bzip2, …), 18 FP (mgrid, swim, ma3d, …)
  - Separate average for integer and FP
  - Benchmarks distributed in source code
  - Compiler, machine designers target benchmarks, so try to change every 3 years

SPEC CPU Benchmark Generations



SPEC2006 benchmark description	Benchmark name by SPEC generation				
	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
GNU C compiler	←	←	←	←	gcc
Interpreted string processing	←	←	perl		espresso
Combinatorial optimization	←	mcf			li
Block-sorting compression	←	bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hmmer	twolf			
Quantum computer simulation	libquantum	vortex			
Discrete event simulation library	omnetpp	vpr			
Chess game (AI)	sjeng	crafty			
XML parsing	xalancbmk	parser			
CFD/blast waves	bwaves				fpppp
Numerical relativity	cactusADM				tomcatv
Finite element code	calculix				doduc
Differential equation solver framework	dealII				nasa7
Quantum chemistry	gamess				spice
EM solver (freq/time domain)	GemsFDTD			swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	hydro2d	
Lattice Boltzman method (fluid/air flow)	lbm		mgrid	su2cor	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	wave5	
Lattice quantum chromodynamics	milc	apply	turb3d		
Molecular dynamics	namd	galgel			
Image ray tracing	povray	mesa			
Spare linear algebra	soplex	art			
Speech recognition	sphinx3	equake			
Quantum chemistry/object oriented	tonto	facerec			
Weather research and forecasting	wrf	ammp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

How Summarize Suite Performance (1/4)

- Arithmetic average of execution time of all programs?
  - But they vary by 4X in speed, so some would be more important than others in arithmetic average
- Could add a weights per program, but how pick weight?
  - Different companies want different weights for their products
- **SPECRatio**: Normalize execution times to reference computer, yielding a ratio proportional to performance =

$$\frac{\text{time on reference computer}}{\text{time on computer being rated}}$$

How Summarize Suite Performance (2/4)

- If program SPECRatio on Computer A is 1.25 times bigger than Computer B, then

$$1.25 = \frac{SPECRatio_A}{SPECRatio_B} = \frac{\dfrac{ExecutionTime_{reference}}{ExecutionTime_A}}{\dfrac{ExecutionTime_{reference}}{ExecutionTime_B}}$$

$$= \frac{ExecutionTime_B}{ExecutionTime_A} = \frac{Performance_A}{Performance_B}$$

- Note that when comparing 2 computers as a ratio, execution times on the reference computer drop out, so **choice of reference computer is irrelevant**

How Summarize Suite Performance (3/4)

- Since ratios, proper mean is geometric mean (SPECRatio unitless, so arithmetic mean meaningless)

$$GeometricMean = \sqrt[n]{\prod_{i=1}^{n} SPECRatio_i}$$

1. Geometric mean of the ratios is the same as the ratio of the geometric means
2. Ratio of geometric means
   = Geometric mean of performance ratios
   $\Rightarrow$ choice of reference computer is irrelevant!

- These two points make geometric mean of ratios attractive to summarize performance

How Summarize Suite Performance (4/4)

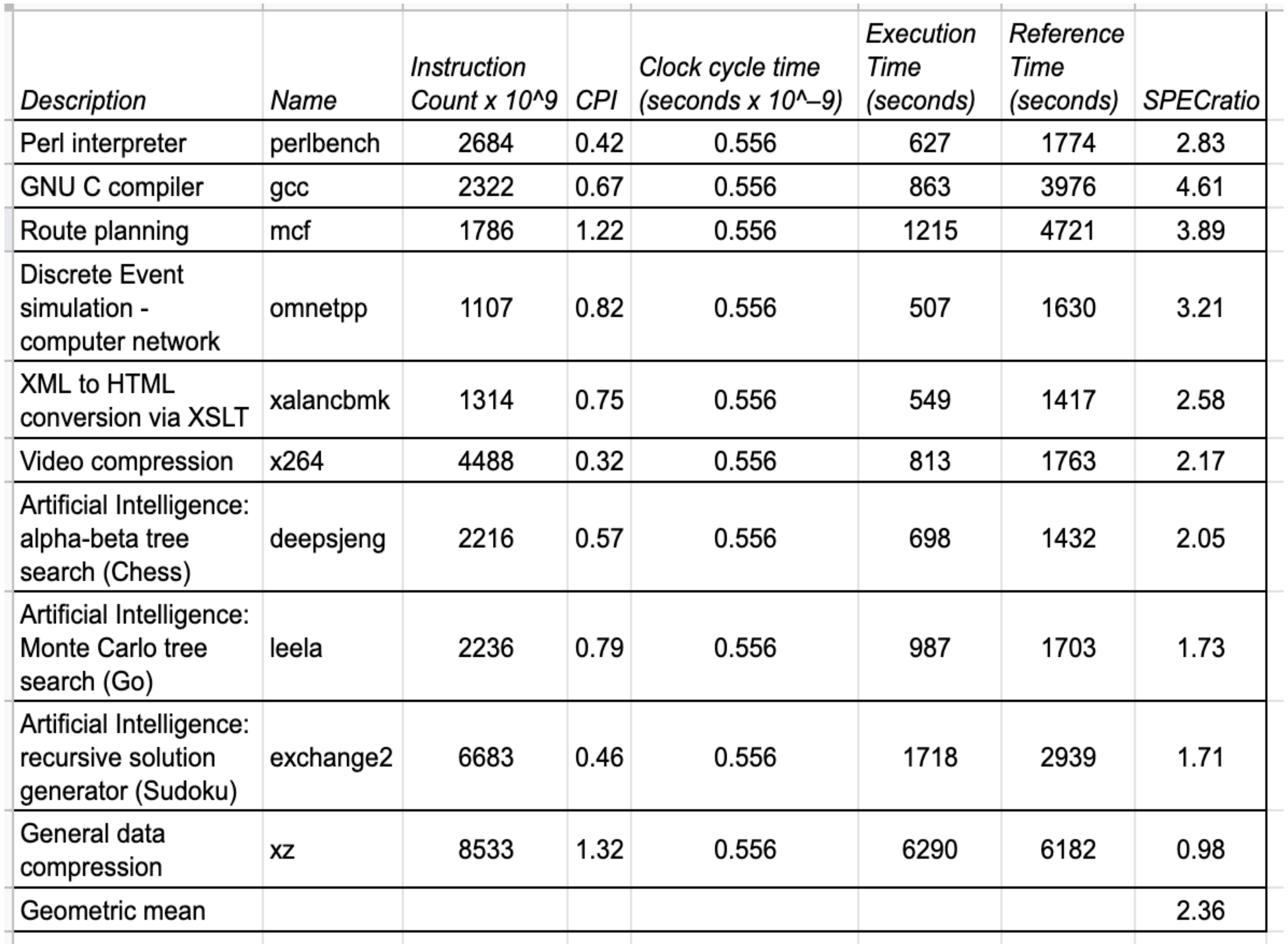
- Does a single mean well summarize performance of programs in benchmark suite?
- Can decide if mean a good predictor by characterizing variability of distribution using standard deviation
- Like geometric mean, geometric standard deviation is multiplicative rather than arithmetic
- Can simply take the logarithm of SPECRatios, compute the standard mean and standard deviation, and then take the exponent to convert back:

$$GeometricMean = \exp\left(\frac{1}{n} \times \sum_{i=1}^{n} \ln(SPECRatio_i)\right)$$

$$GeometricStDev = \exp(StDev(\ln(SPECRatio_i)))$$

- The geometric standard deviation, denoted by σ_g, is calculated as follows: $\log \sigma_g = [1/n \sum^n_{i=1} (\log x_i - \log G)^2]^{1/2}$.
- where $G = \sqrt[n]{x_1 \cdot x_2 \cdot \ldots \cdot x_n}$ is the geometric mean of SPECRatios (x_1 . x_n).

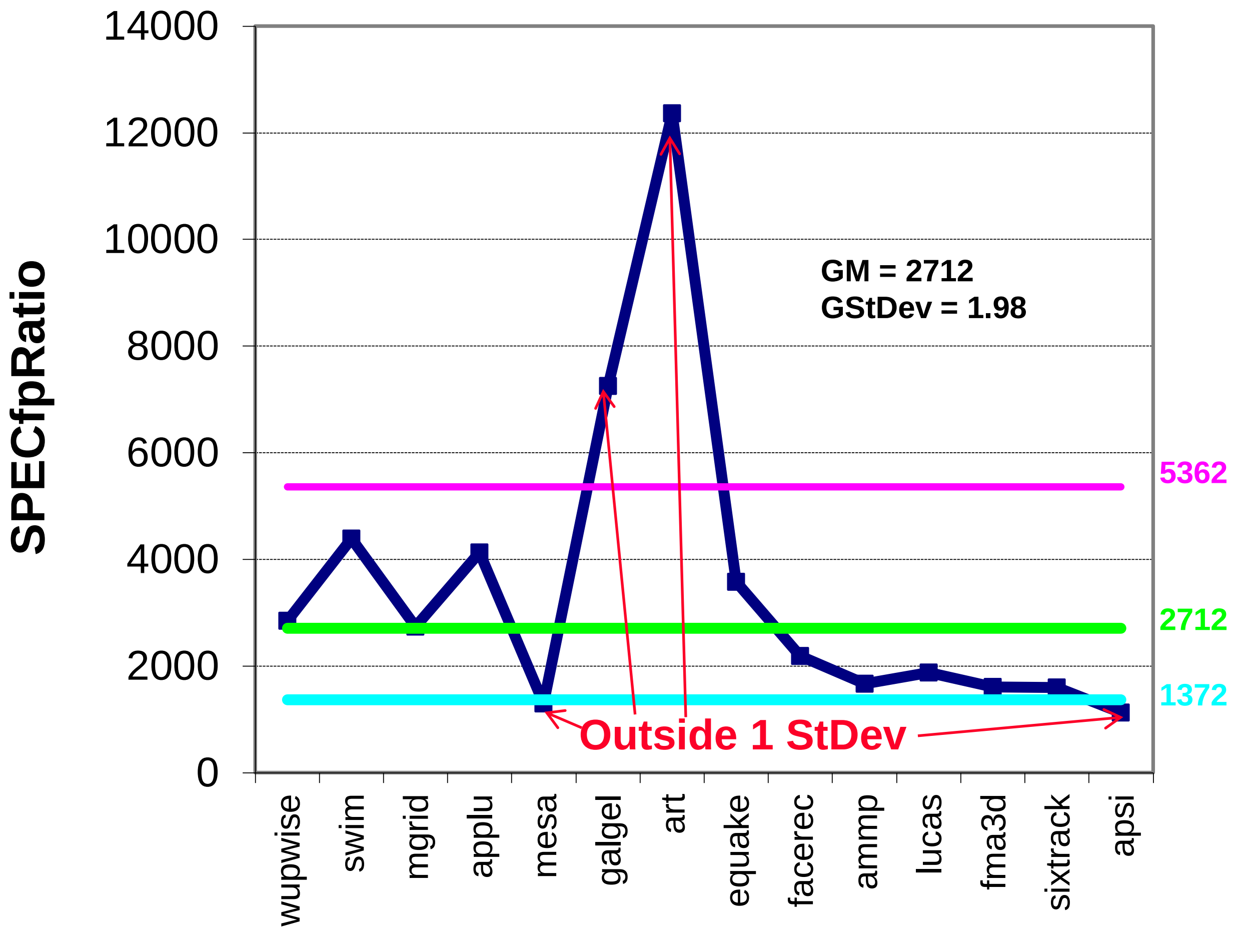
SPECspeed 2017 Integer Benchmarks on a 1.8 GHz Intel Xeon E5-2650L



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

Example Standard Deviation: (1/3)

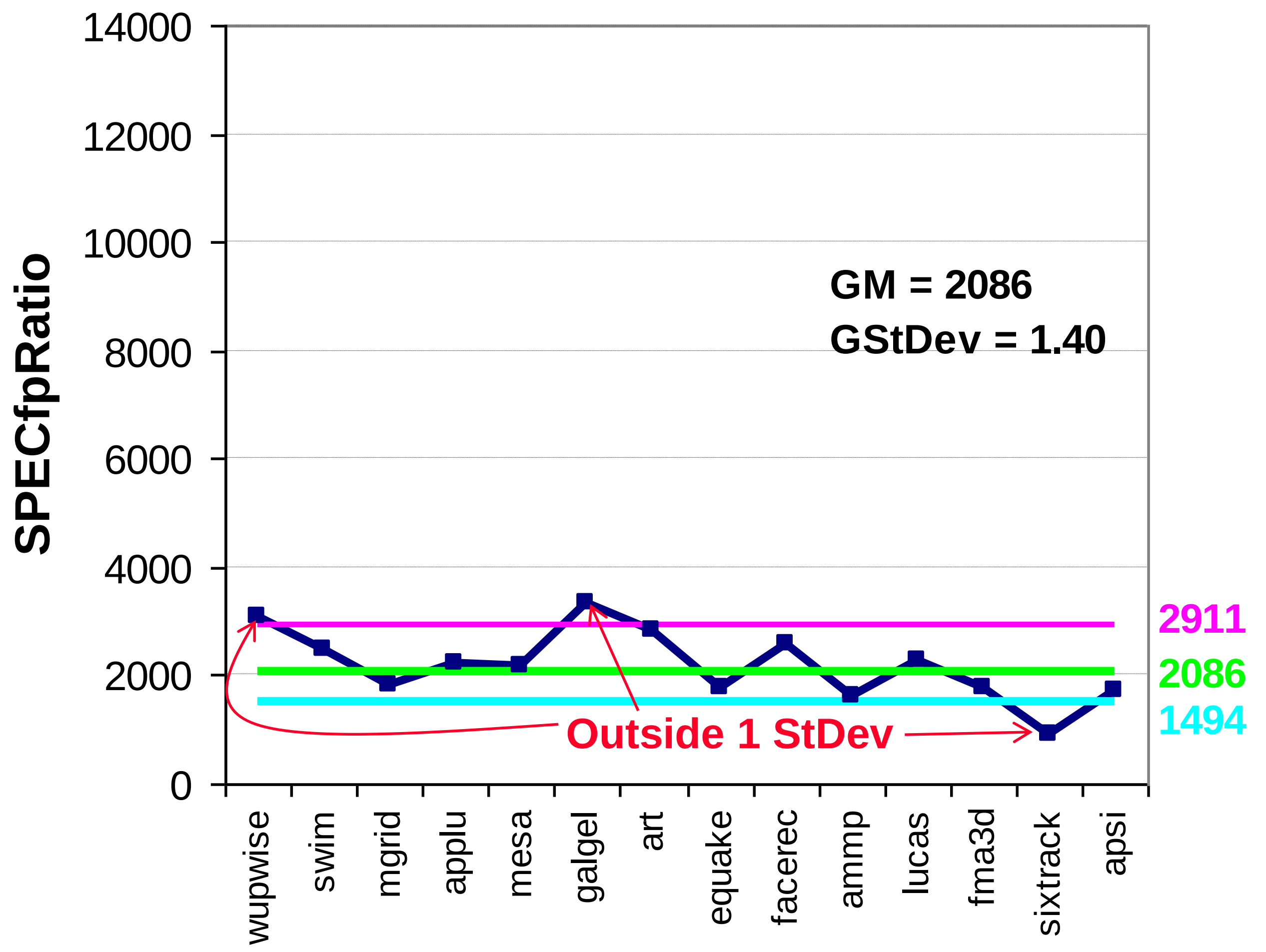
- GM and multiplicative StDev of SPECfp2000 for Itanium 2



Itanium 2 is 2712/100 times as fast as Sun Ultra 5 (GM), & range within 1 Std. Deviation is [13.72, 53.62]

Example Standard Deviation : (2/3)

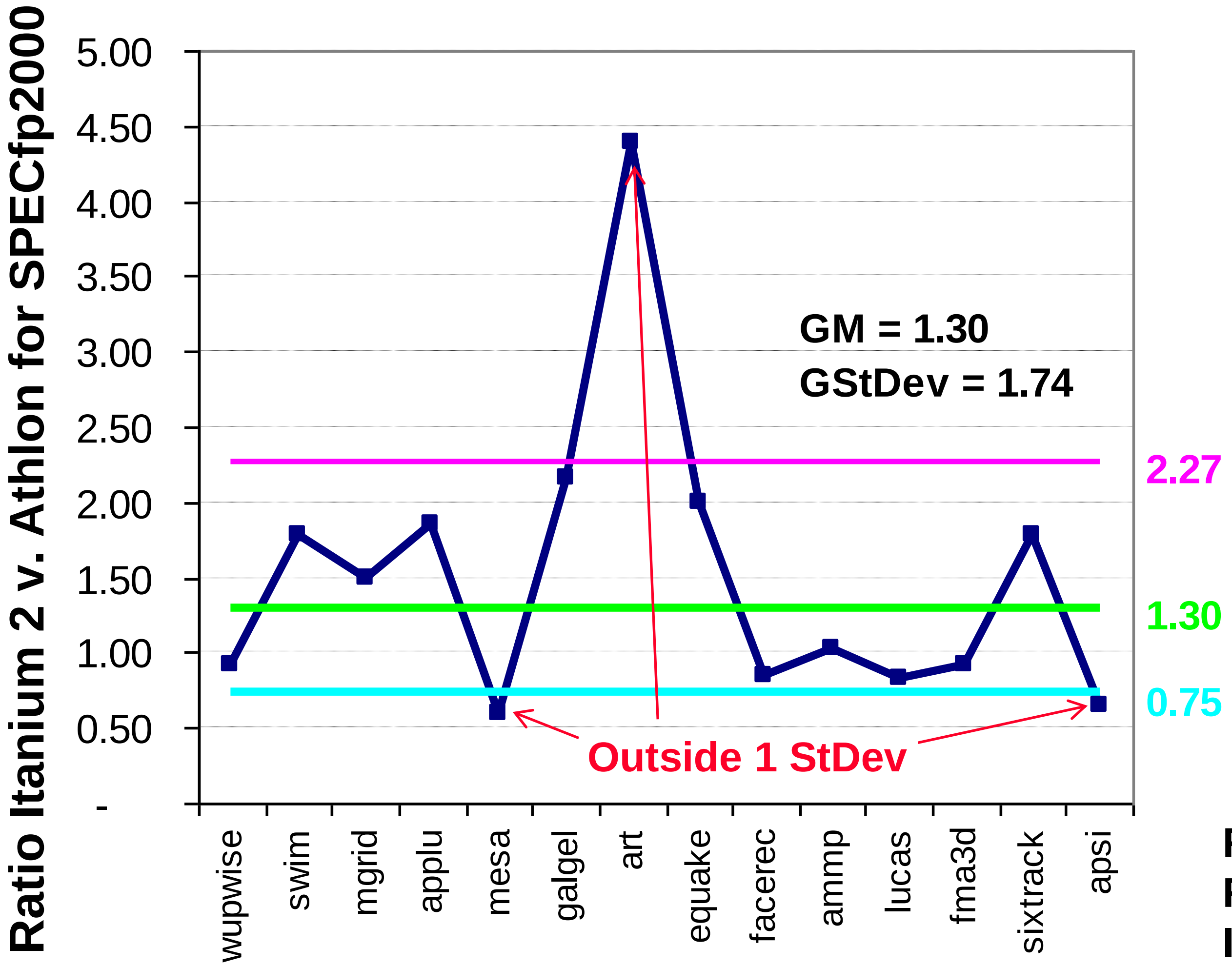
- **GM and multiplicative StDev of SPECfp2000 for AMD Athlon**



Athon is 2086/100 times as fast as Sun Ultra 5 (GM), & range within 1 Std. Deviation is [14.94, 29.11]

Example Standard Deviation (3/3)

- **GM and StDev Itanium 2 v Athlon**



Ratio execution times (At/It) = Ratio of SPECratios (It/At)
Itanium 2 1.30X Athlon (GM), 1 St.Dev. Range [0.75,2.27]

Comments on Itanium 2 and Athlon

- Standard deviation of 1.98 for Itanium 2 is much higher-- vs. 1.40--so results will differ more widely from the mean, and therefore are likely less predictable for Itanium 2
- Falling within one standard deviation:
  - 10 of 14 benchmarks (71%) for Itanium 2
  - 11 of 14 benchmarks (78%) for Athlon
- Thus, the results are quite compatible with a lognormal distribution (expect 68%)
- Itanium 2 vs. Athlon St.Dev is 1.74, which is high, so less confidence in claim that Itanium 1.30 times as fast as Athlon
  - Indeed, Athlon faster on 6 of 14 programs

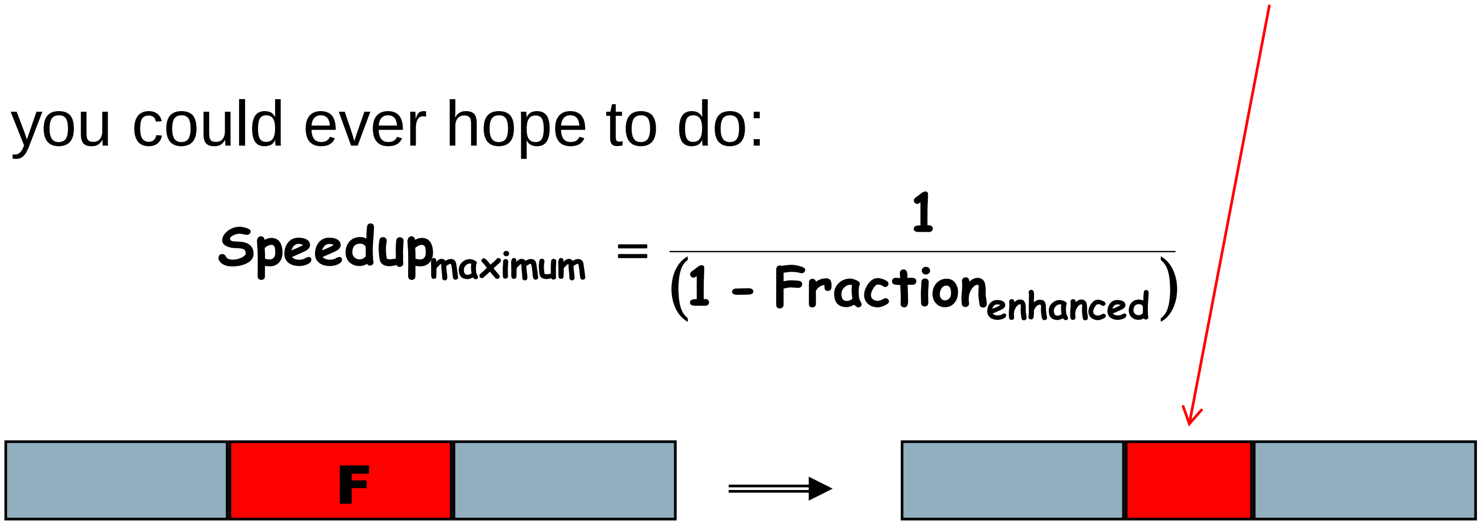
Amdahl's Law

$$\text{ExTime}_{\text{w/ Enh.}} = \text{ExTime}_{\text{w/o Enh.}} \times \left[\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right]$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{w/o Enh.}}}{\text{ExTime}_{\text{w/ Enh.}}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Best you could ever hope to do:

$$\text{Speedup}_{\text{maximum}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right)}$$



Amdahl's Law Example

- New CPU 10X faster
- I/O bound server, so 60% time waiting for I/O

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$= \frac{1}{\left(1 - 0.4\right) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

- Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster

Question

- Speedup $= \dfrac{1}{(1 - F) + \dfrac{F}{S}}$

Question: Suppose a program spends 80% of its time in a square root routine. How much must you speed up square root to make the program run 5 times faster?

(A) **10**

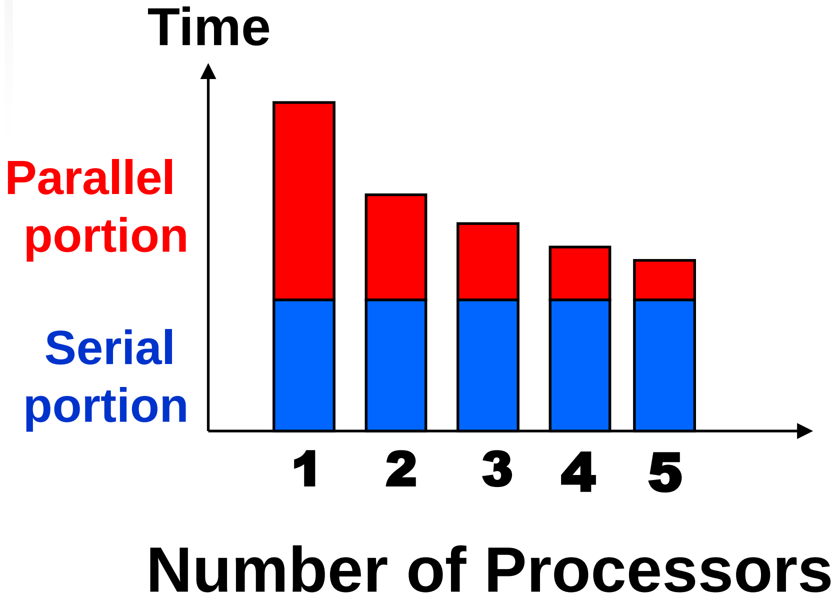
(B) **20**

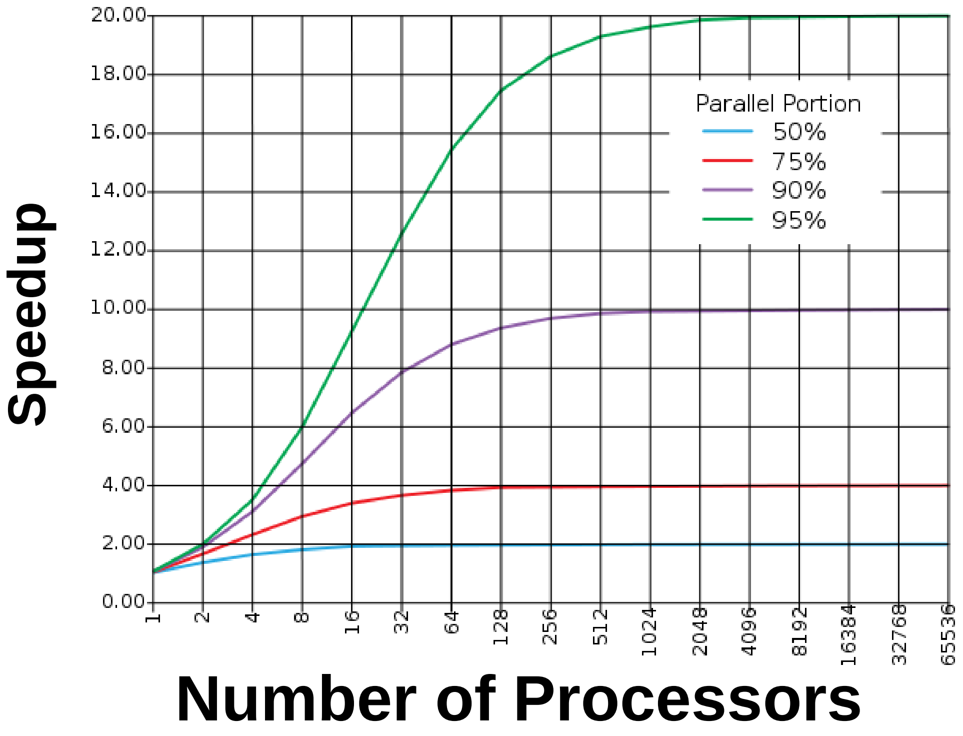
(C) **100**

(D) **None of the above**

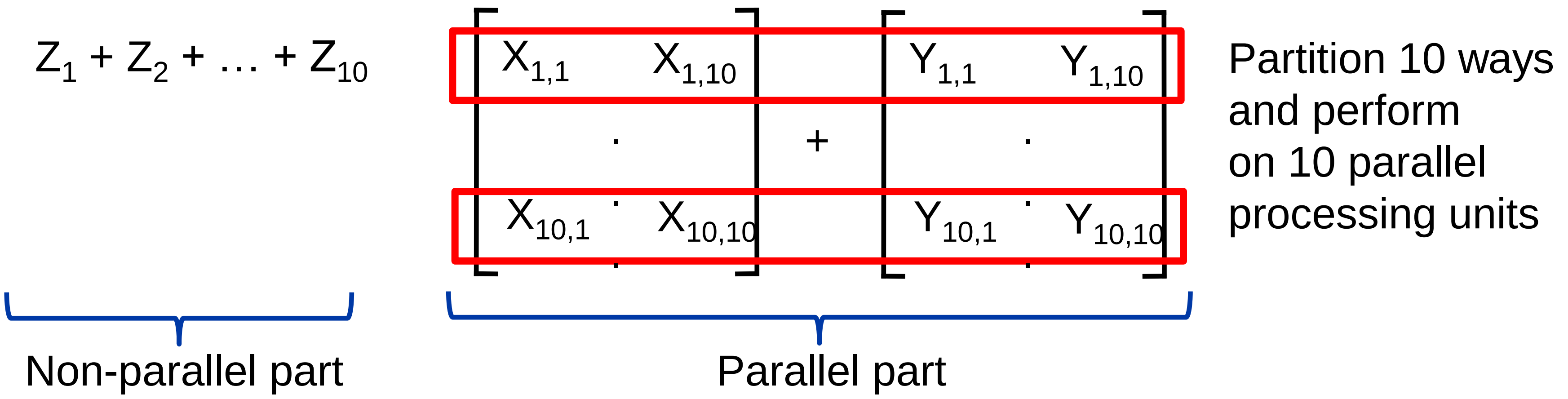
Consequence of Amdahl's Law

- The amount of speedup that can be achieved through parallelism is limited by the non-parallel portion of your program!





Parallel Speed-up Examples (1/2)



- 10 “scalar” operations (non-parallelizable)
- 100 parallelizable operations
  - Say, element-wise addition of two 10x10 matrices.
- 110 operations
  - 100/110 = .909 Parallelizable, 10/110 = 0.091 Scalar

Parallel Speed-up Examples (2/2)

Speedup w/ E = 1 / [(1-F) + F/S]

- Consider summing 10 scalar variables and two 10 by 10 matrices (matrix sum) on 10 processors

  Speedup = 1/(.091 + .909/10) = 1/0.1819 = 5.5

- What if there are 100 processors ?

  Speedup = 1/(.091 + .909/100) = 1/0.10009 = 10.0

- What if the matrices are 100 by 100 (or 10,010 adds in total) on 10 processors?

  Speedup = 1/(.001 + .999/10) = 1/0.1009 = 9.9

- What if there are 100 processors ?

  Speedup = 1/(.001 + .999/100) = 1/0.01099 = 91

Strong and Weak Scaling

- To get good speedup on a multiprocessor while keeping the problem size fixed is harder than getting good speedup by increasing the size of the problem
  - ***Strong scaling:*** When speedup is achieved on a parallel processor without increasing the size of the problem
  - ***Weak scaling:*** When speedup is achieved on a parallel processor by increasing the size of the problem proportionally to the increase in the number of processors (Gustafson's law)
- **Load balancing** is another important factor: every processor doing same amount of work
  - Just 1 unit with twice the load of others cuts speedup almost in half (bottleneck!)

Other Performance Metrics

- ***MIPS*** – Million Instructions Per Second

$$MIPS = \frac{Instruction\_count}{Time(s) \times 10^6}$$

- ***MFLOPS*** - Million Floating-point Operations Per Second

$$MFLOPS = \frac{Floating\_point\_ops/program}{Time(s) \times 10^6}$$

- ***PetaFLOPS*** - 10^{15} Floating-point Operations Per Second

$$PFLOPS = \frac{Floating\_point\_ops/program}{Time(s) \times 10^{15}}$$

Pitfall: MIPS as a Performance Metric

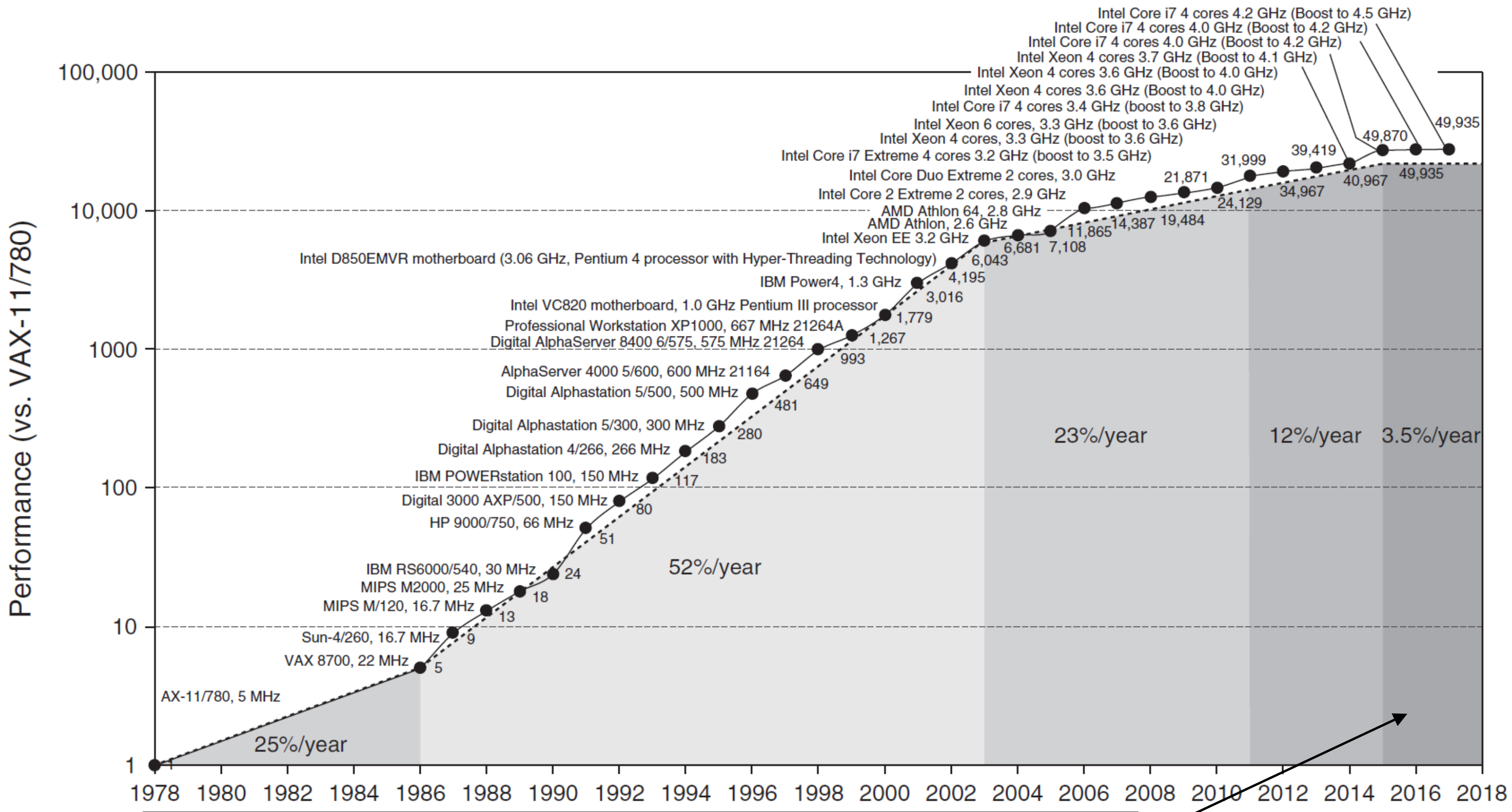
- MIPS: Millions of Instructions Per Second
  - Doesn’t account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

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

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

  - CPI varies between programs on a given CPU

Uniprocessor Performance



Constrained by power, instruction-level parallelism, 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

Top 5 Supercomputers (TOP500, Nov. 2019)

Rank	System	Cores	Rmax (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, **NVIDIA Volta GV100**, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory **United States**	2,414,592	148,600.0	10,096
2	**Sierra** - IBM Power System AC922LC, IBM POWER9 22C 3.1GHz, **NVIDIA Volta GV100**, Dual-rail Mellanox EDR Infiniband , IBM DOE/NNSA/LLNL **United States**	1,572,480	94,640.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi **China**	10,649,600	93,014.6	15,371
4	**Tianhe-2A (Milky Way -2A)-** TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou **China**	4,981,760	61,444.5	18,482
5	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR , Dell EMC Texas Advanced Computing Center/Univ. of Texas **United States**	448,448	23,516.4	?

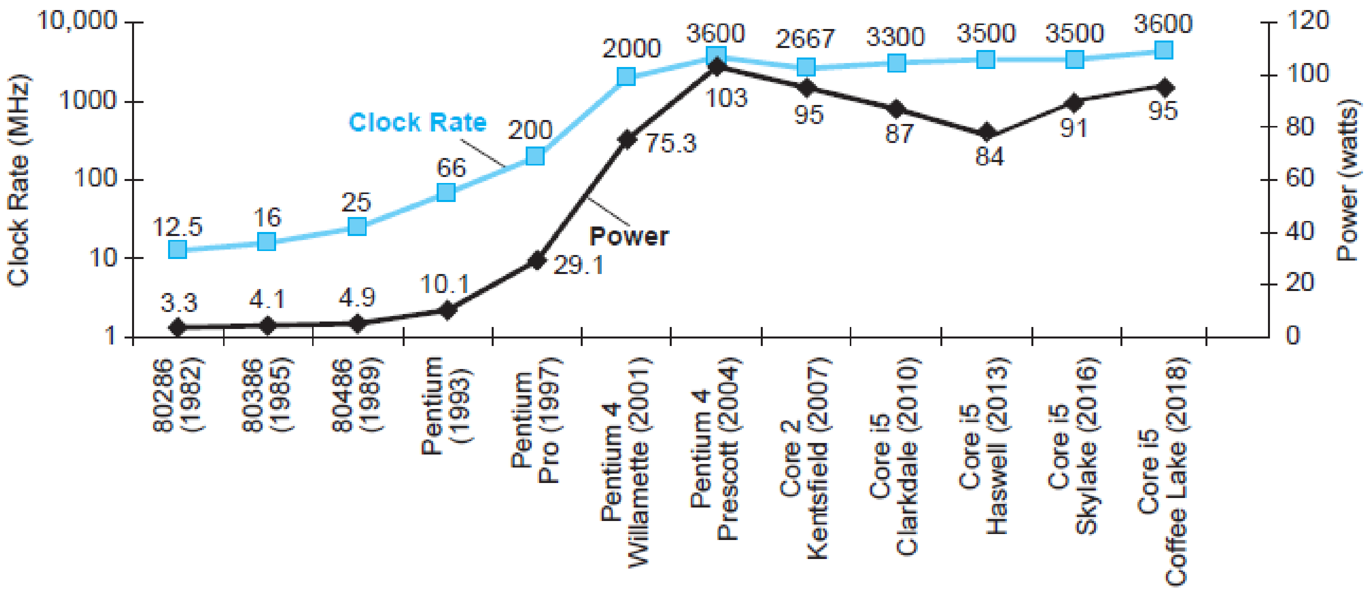
Top 5 Supercomputers (HPCG, June 2018)

The High-Performance Conjugate Gradient (HPCG) results, an alternative (sparse-matrix) computing benchmark

Rank	System	Cores	Performance (petaflops)
1	**Summit**	Oak Ridge National Laboratory, **U.S.A.**	**2.93**
2	**Sierra**	Lawrence Livermore National Laboratory, **U.S.A.**	**1.80**
3	**K computer**	Riken Advanced Institute for Computational Science, **Japan**	**0.60**
4	**Trinity**	Los Alamos National Laboratory, **U.S.A.**	**0.55**
5	**Piz Daint**	Swiss National Supercomputing Centre, **Switzerland**	**0.49**

Power Trends

- 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
- This is the limit of what can be cooled by air



- In CMOS IC technology

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

×30

5V → 1V

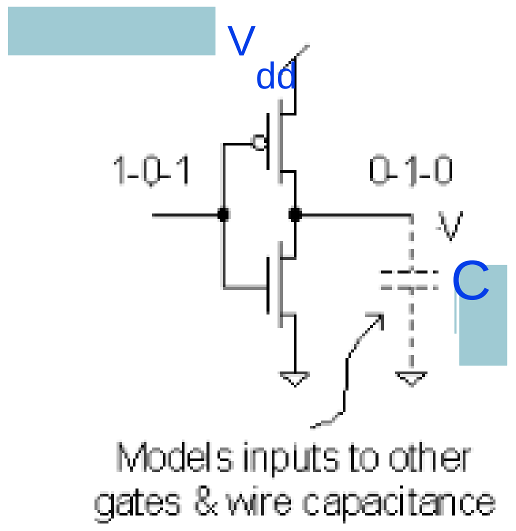
×1000

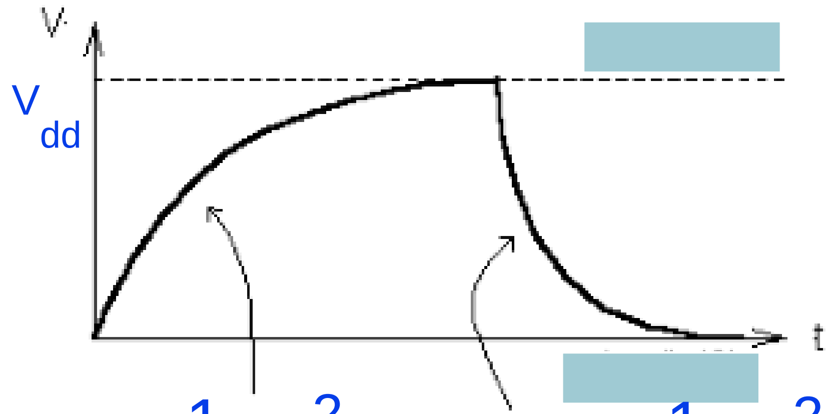
Energy and Power

- Dynamic energy
  - Transistor switch from 0 -> 1 or 1 -> 0
  - ½ x Capacitive load x Voltage^2
- Dynamic power
  - ½ x Capacitive load x Voltage^2 x Frequency switched
- Reducing clock rate reduces power, not energy
- Static power consumption
  - Current$_{static}$ x Voltage
  - Scales with number of transistors
  - To reduce: power gating

Switching Energy: Fundamental Physics

Every logic transition dissipates energy.





$E_{0\to1} = \frac{1}{2} C V_{dd}^2$ $\quad E_{1\to0} = \frac{1}{2} C V_{dd}^2$

Strong result: Independent of technology.

How can we limit switching energy?

(1) Reduce # of clock transitions. But we have work to do ...

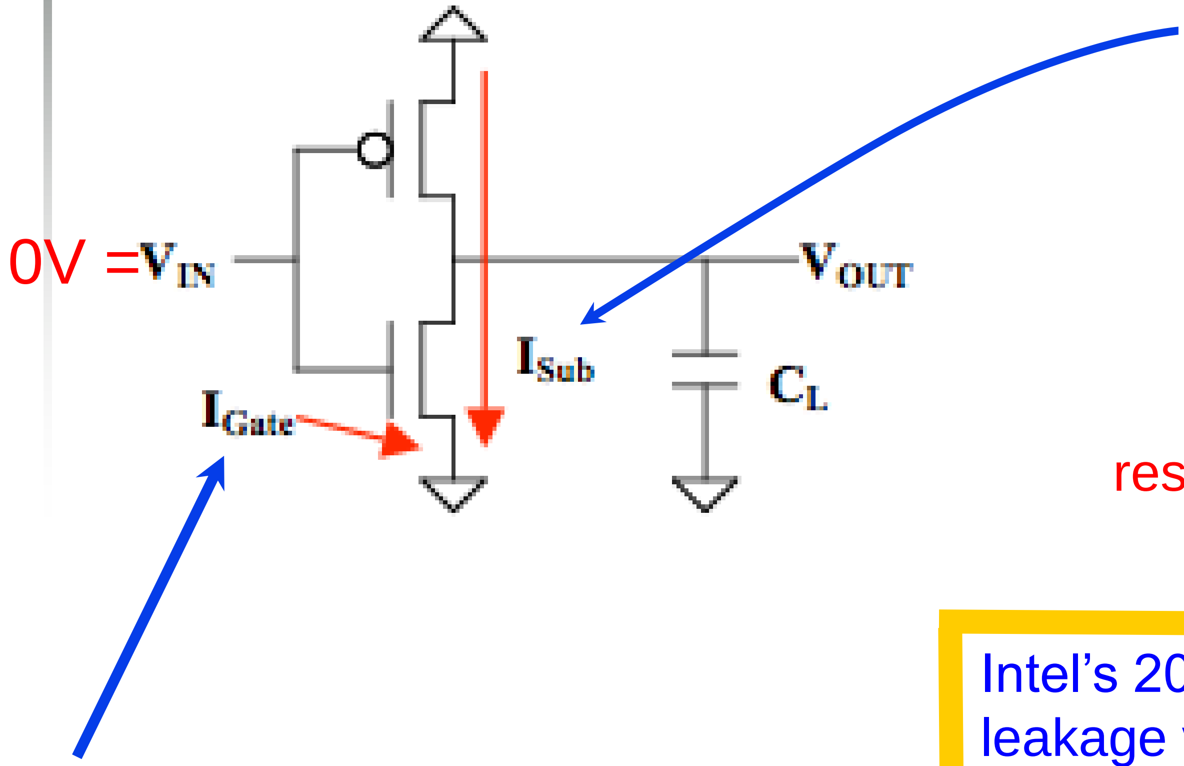
(2) Reduce Vdd. But lowering Vdd limits the clock speed ...

(3) Fewer circuits. But more transistors can do more work.

(4) Reduce C per node. One reason why we scale processes.

Second Factor: Leakage Currents

Even when a logic gate isn't switching, it burns power.

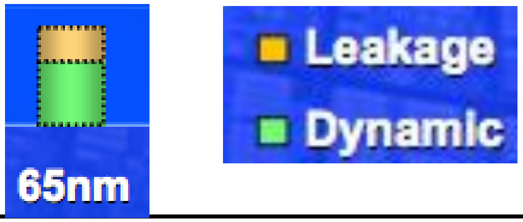


Isub: Even when this nFet is off, it passes an Ioff leakage current.

We can engineer any Ioff we like, but a lower Ioff also results in a lower Ion, and thus a lower maximum clock speed.

Igate: Ideal switches have zero DC current. But modern transistor gates are a few atoms thick, and are not ideal.

Intel's 2006 processor designs, leakage vs switching power



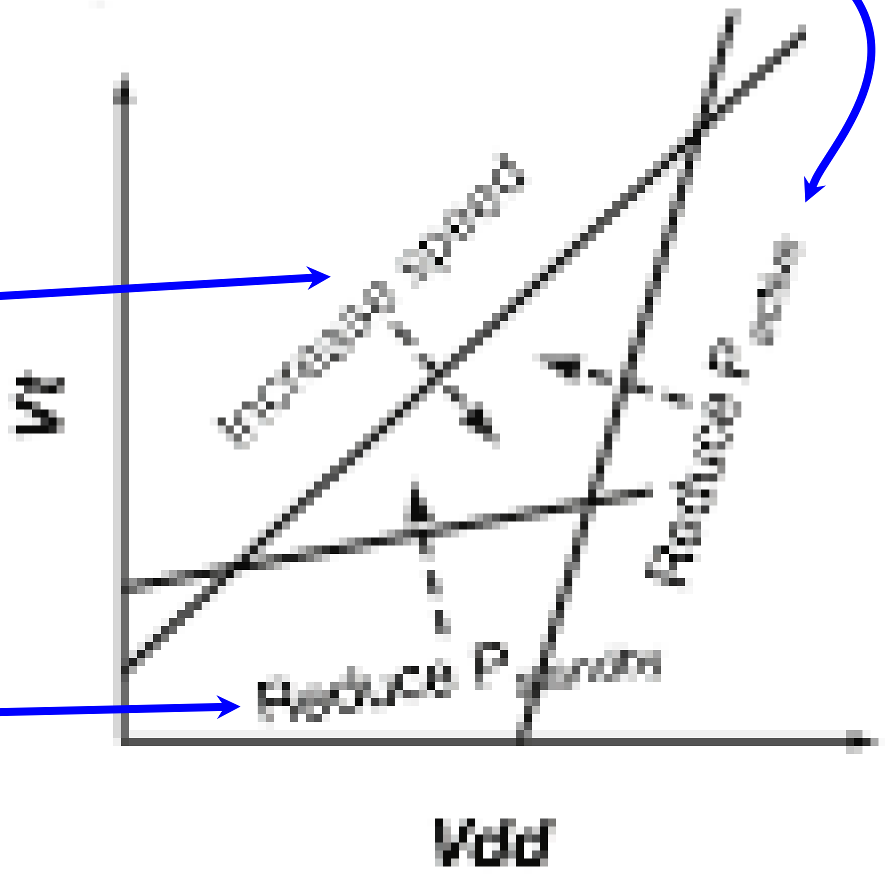
A lot of work was done to get a ratio this good ... 50/50 is common.

Device Engineers Trade Speed and Power

We can reduce CV^2 (P_{active}) by lowering V_{dd}.

We can increase speed by raising V_{dd} and lowering V_t.

We can reduce leakage ($P_{standby}$) by raising V_t.



From: Silicon Device Scaling to the Sub-10-nm Regime

Meikei Ieong,[1*] Bruce Doris,[2] Jakub Kedzierski,[1] Ken Rim,[1] Min Yang[1]

Example of Quantifying Power

- Suppose 15% reduction in voltage results in a 15% reduction in frequency. What is impact on dynamic power?

$$\begin{aligned}
Power_{dynamic} &= 1/2 \times CapacitiveLoad \times Voltage^2 \times FrequencySwitched \\
&= 1/2 \times .85 \times CapacitiveLoad \times (.85 \times Voltage)^2 \times FrequencySwitched \\
&= (.85)^3 \times OldPower_{dynamic} \\
&\approx 0.6 \times OldPower_{dynamic}
\end{aligned}$$

Acknowledgements

- These slides contain material developed and copyright by:
  - Morgan Kauffmann (Elsevier, Inc.)
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - Justin Hsia (UCB)