



西安交通大学
XIAN JIAOTONG UNIVERSITY

Modern Computer Architecture

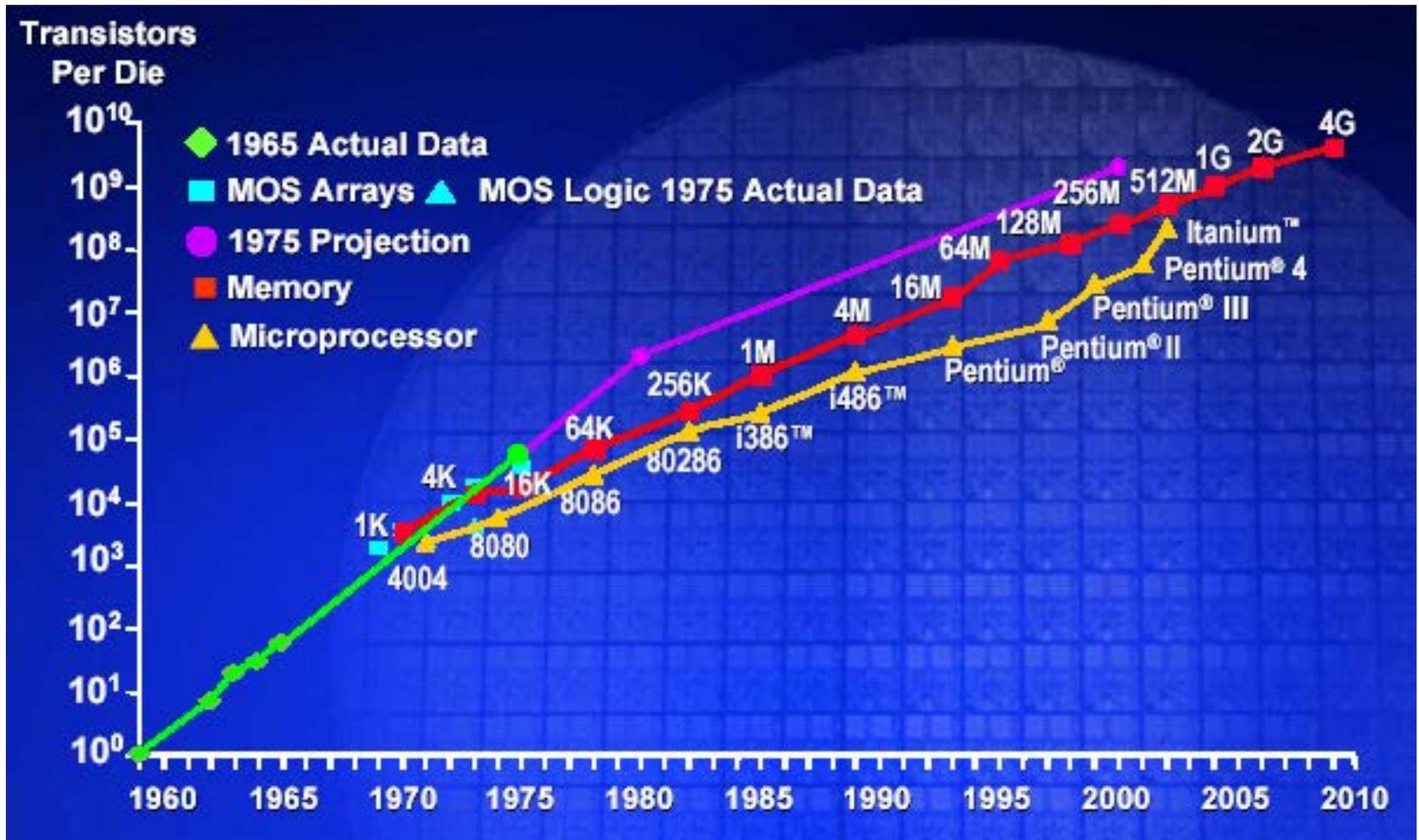
Lecture 1 Fundamentals of Quantitative Design and Analysis (II)

Hongbin Sun

国家集成电路人才培养基地

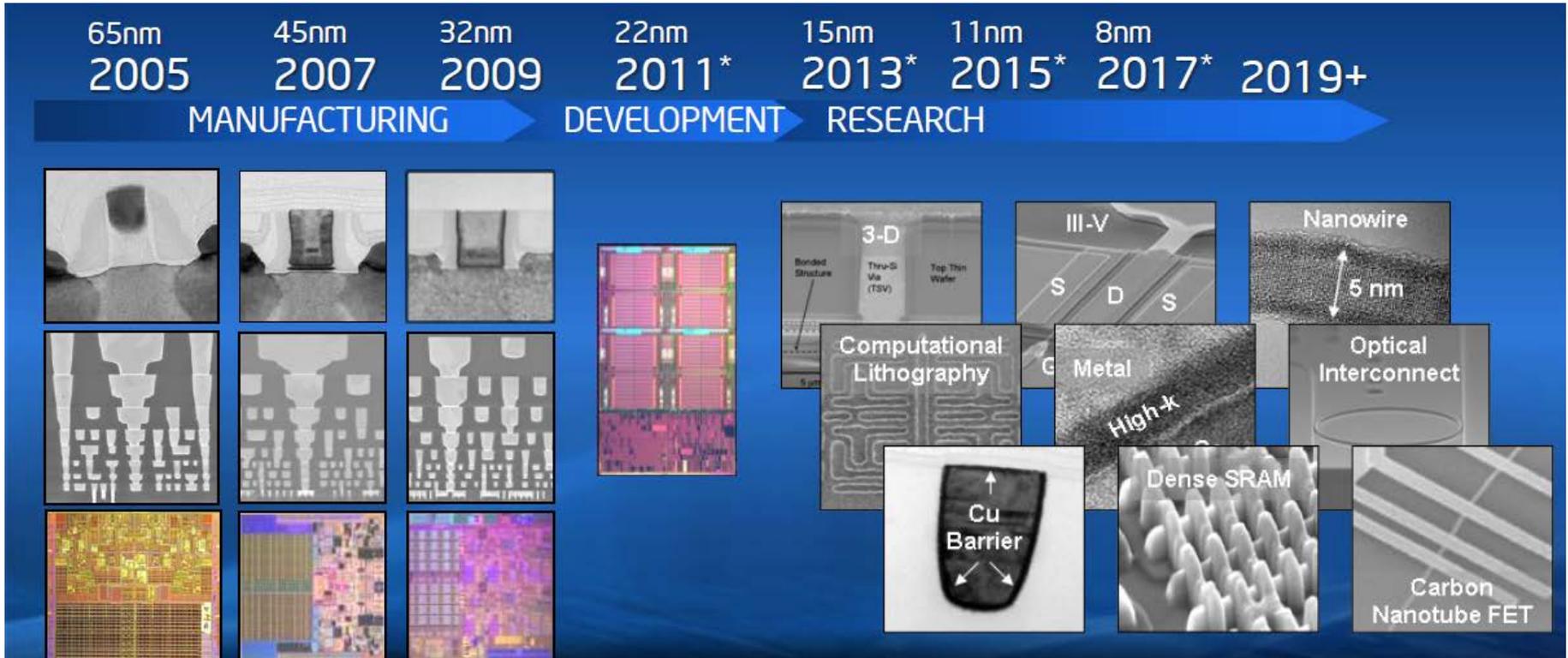
Xi'an Jiaotong University

1.4 Trends in Technology



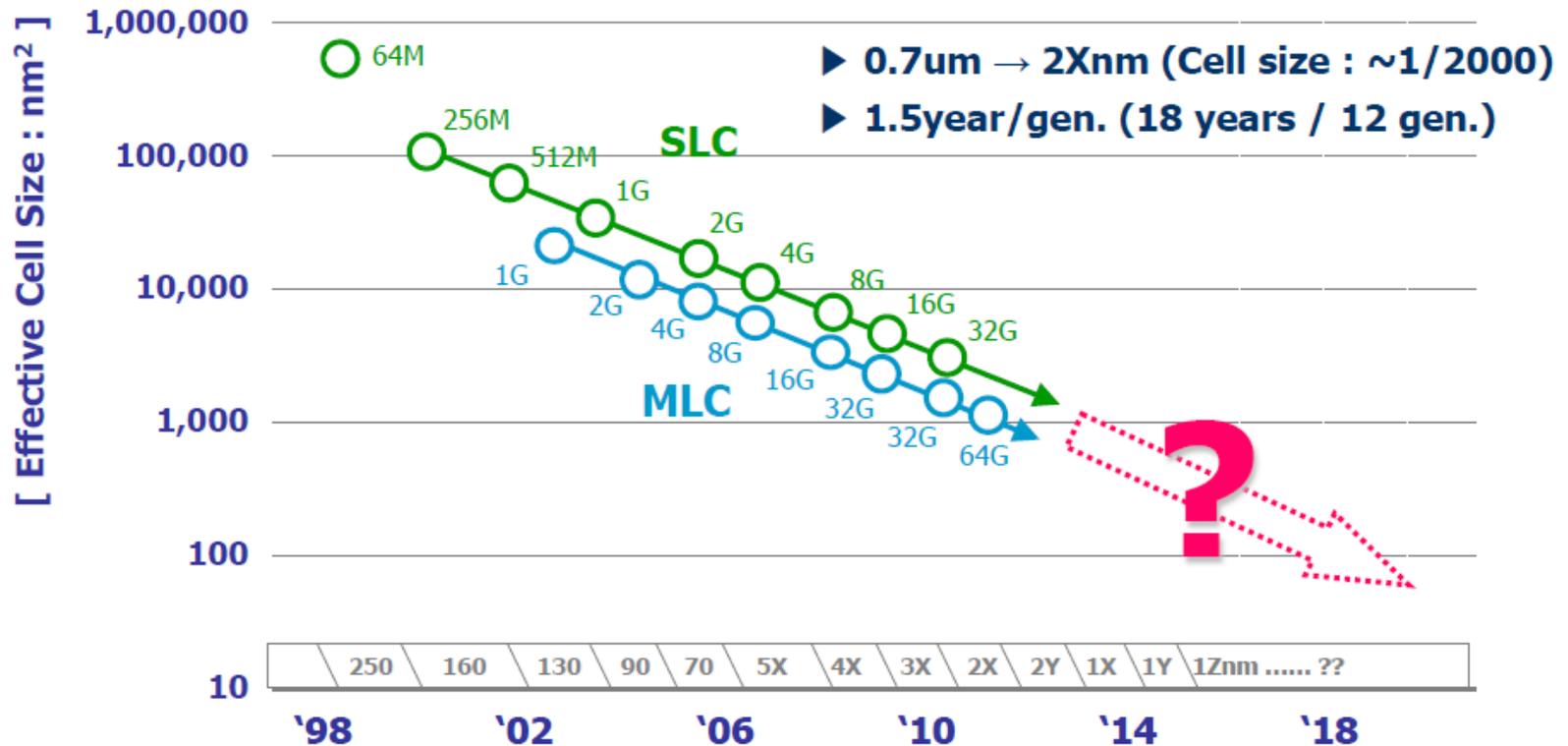
Logic: transistor density 35%/year, die size 10-20%/year, capacity 40-55%/year
DRAM: capacity 25-40%/year, and maybe stop in the middle of this decade.

Processor Technology

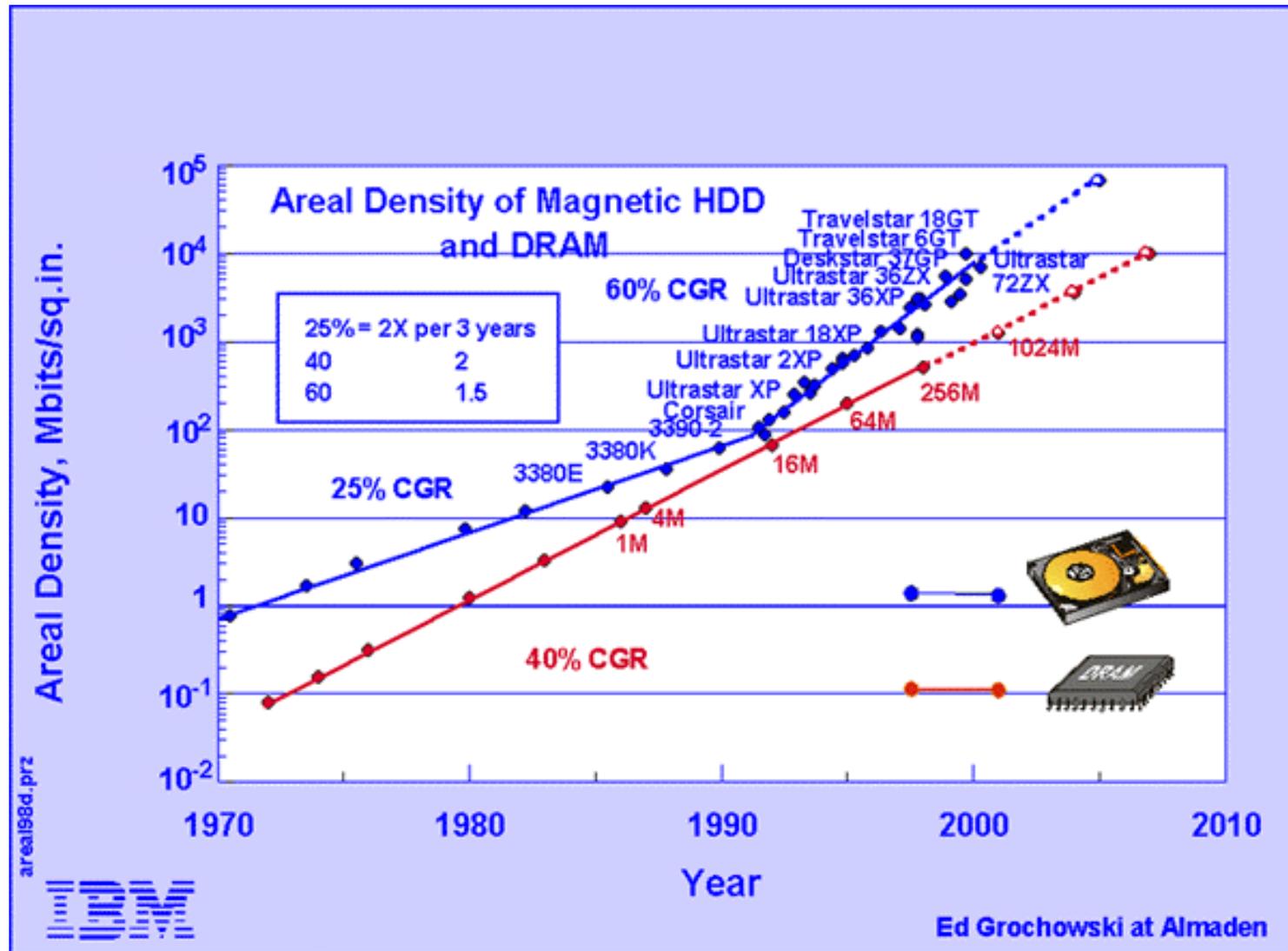


Semiconductor Flash

Conventional FG NAND cell has been scaled down over 18 years.

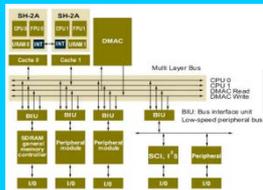


Magnetic Disk Technology

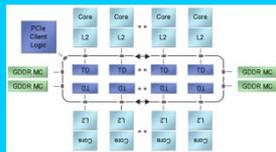


Network Technology

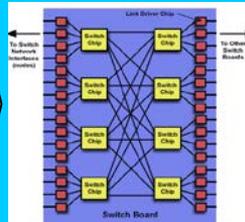
Bus
(core ≤ 8)



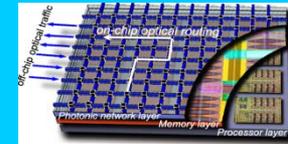
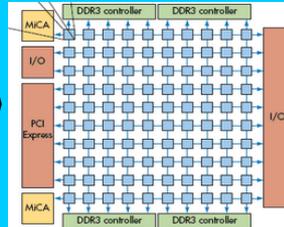
Ring
(cores < 10)



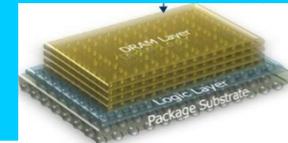
Crossbar
(cores < 16)



Crossbar
(cores ≤ 100)

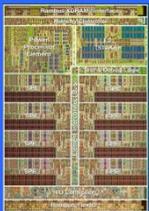


Optical Network
(cores > 100)



3D Topology
(cores > 100)

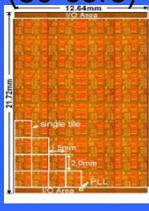
IBM Cell
(8-core)



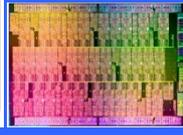
Sun SPAC T1
(8-core)



Intel TeraFlop80
(80-core)



Intel SCC48
(48-core)



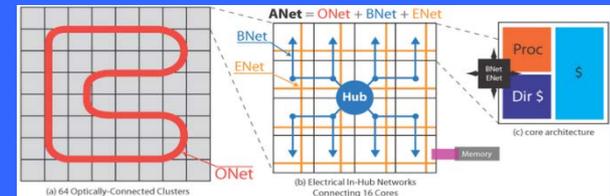
Tilera
(64/100cores)



Sun SPAC T5
(16-core)

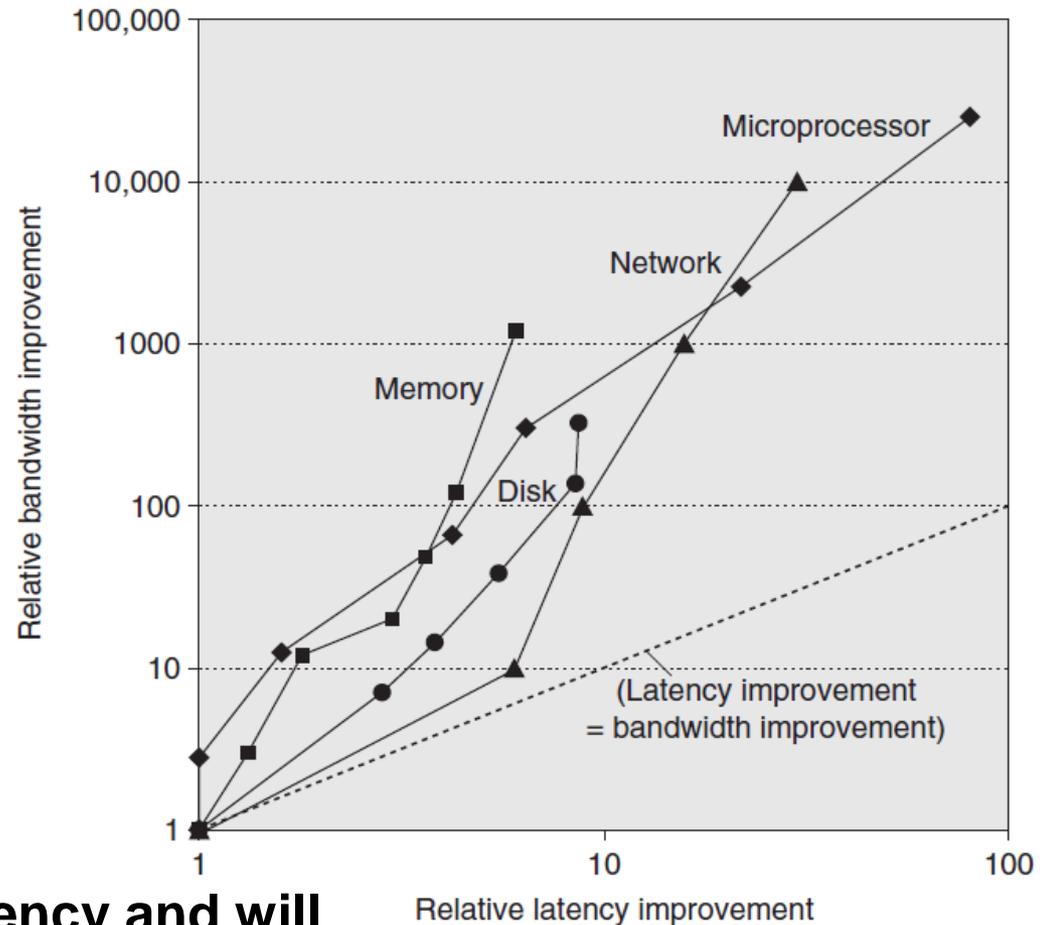


ATAC (1024 core)



Bandwidth over Latency

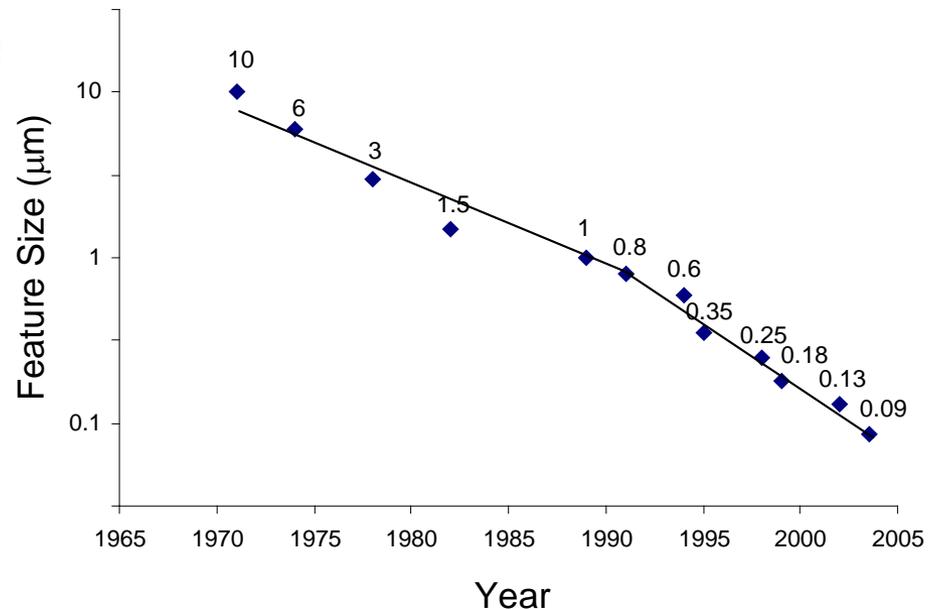
- **Bandwidth** or throughput is the total work done in a given time.
- **Latency** or response time is the time between the start and the completion of an event.



Bandwidth has outpaced latency and will likely continue to do so.

Technology Scaling

- The only constant in VLSI is constant change
- Feature size shrinks by 30% every 2-3 years
 - Transistors become cheaper
 - Transistors become faster
 - Wires do not improve (and may get worse)
- Scale factor S
 - Typically $S = \sqrt{2}$
 - Technology nodes



Device Scaling Assumption

- **What changes between technology nodes?**
- **Constant Field Scaling**
 - All dimensions ($x, y, z \Rightarrow W, L, t_{ox}$)
 - Voltage (V_{DD})
 - Doping levels
- **Lateral Scaling**
 - Only gate length L
 - Often done as a quick gate shrink ($S = 1.05$)

Device Scaling

Table 4.15 Influence of scaling on MOS device characteristics

Parameter	Sensitivity	Constant Field	Lateral
Scaling Parameters			
Length: L		$1/S$	$1/S$
Width: W		$1/S$	1
Gate oxide thickness: t_{ox}		$1/S$	1
Supply voltage: V_{DD}		$1/S$	1
Threshold voltage: V_{tn}, V_{tp}		$1/S$	1
Substrate doping: N_A		S	1
Device Characteristics			
β	$\frac{W}{L} \frac{1}{t_{ox}}$	S	S
Current: I_{ds}	$\beta(V_{DD} - V_t)^2$	$1/S$	S
Resistance: R	$\frac{V_{DD}}{I_{ds}}$	1	$1/S$
Gate capacitance: C	$\frac{WL}{t_{ox}}$	$1/S$	$1/S$
Gate delay: τ	RC	$1/S$	$1/S^2$
Clock frequency: f	$1/\tau$	S	S^2
Dynamic power dissipation (per gate): P	CV^2f	$1/S^2$	S
Chip area: A		$1/S^2$	1
Power density	P/A	1	S
Current density	I_{ds}/A	S	S

- Gate capacitance per micron is nearly independent of process
- But ON resistance * micron improves with process
- Gates get faster with scaling (good)
- Dynamic power goes down with scaling (good)
- Current density goes up with scaling (bad)
- Velocity saturation makes lateral scaling unsustainable

Results of Device Scaling

- **The fact that transistor count improves quadratically with a linear improvement in transistor performance is both the challenge and the opportunity.**
 - **4-bit, 8-bit, 16-bit, 32-bit, to 64-bit microprocessor.**
 - **Multiple processors per chip**
 - **Wider SIMD units**
 - **Speculative execution**
 - **Caches**

Wire Scaling Assumption

- **Wire thickness**
 - Hold constant vs. reduce in thickness
- **Wire length**
 - Local / scaled interconnect
 - **Global interconnect**
 - Die size scaled by $D_c \approx 1.1$

Wire Scaling

Table 4.16 Influence of scaling on interconnect characteristics

Parameter	Sensitivity	Reduced Thickness	Constant Thickness
Scaling Parameters			
Width: w			$1/S$
Spacing: s			$1/S$
Thickness: t		$1/S$	1
Interlayer oxide height: b			$1/S$
Local/Scaled Interconnect Characteristics			
Length: l			$1/S$
Unrepeated wire RC delay	$l^2 t_{wu}$	1	between $1/S, 1$
Repeated wire delay	lt_{wr}	$\sqrt{1/S}$	between $1/S, \sqrt{1/S}$
Global Interconnect Characteristics			
Length: l			D_c
Unrepeated wire RC delay	$l^2 t_{wu}$	$S^2 D_c^2$	between $SD_c^2, S^2 D_c^2$
Repeated wire delay	lt_{wr}	$D_c \sqrt{S}$	between $D_c, D_c \sqrt{S}$

Observations

- **Capacitance per micron is remaining constant**
 - About 0.2 fF/ μm
 - Roughly 1/10 of gate capacitance
- **Local wires are getting faster**
 - Not quite tracking transistor improvement
 - But not a major problem
- **Global wires are getting slower**
 - No longer possible to cross chip in one cycle
- **Wire delay has become a major design limitation for large integrated circuits and is often more critical than transistor switching delay.**

ITRS

- **Semiconductor Industry Association forecast**
 - **Intl. Technology Roadmap for Semiconductors**

Year	2009	2012	2015	2018	2021
Feature size (nm)	34	24	17	12	8.4
L_{gate} (nm)	20	14	10	7	5
V_{DD} (V)	1.0	0.9	0.8	0.7	0.65
Billions of transistors/die	1.5	3.1	6.2	12.4	24.7
Wiring levels	12	12	13	14	15
Maximum power (W)	198	198	198	198	198
DRAM capacity (Gb)	2	4	8	16	32
Flash capacity (Gb)	16	32	64	128	256

2007

1.5 Trends in Power and Energy

- **Three primary concerns about power and energy:**
 - **What is the maximum power a processor ever requires?**
 - **What is the sustained power consumption?**
 - Thermal design power (TPD)
 - **Energy and energy efficiency**
- **Dynamic power:**
$$\text{Power}_{dynamic} = 0.5 \times \text{CapacitiveLoad} \times \text{Voltage}^2 \times \text{FrequencySwitched}$$
- **For mobile devices, energy is the better metric**
$$\text{Energy}_{dynamic} = \text{CapacitiveLoad} \times \text{Voltage}^2$$
- For a fixed task, slowing clock rate (frequency switched) reduces power, but not energy

Dynamic Power and Energy

- Capacitive load a function of number of transistors connected to output and technology, which determines capacitance of wires and transistors
- Dropping voltage helps both, so went from 5V to 1V
- To save energy & dynamic power, most CPUs now turn off clock of inactive modules (e.g. Fl. Pt. Unit)
- **Example: Suppose 15% reduction in voltage results in a 15% reduction in frequency. What is impact on dynamic power?**

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

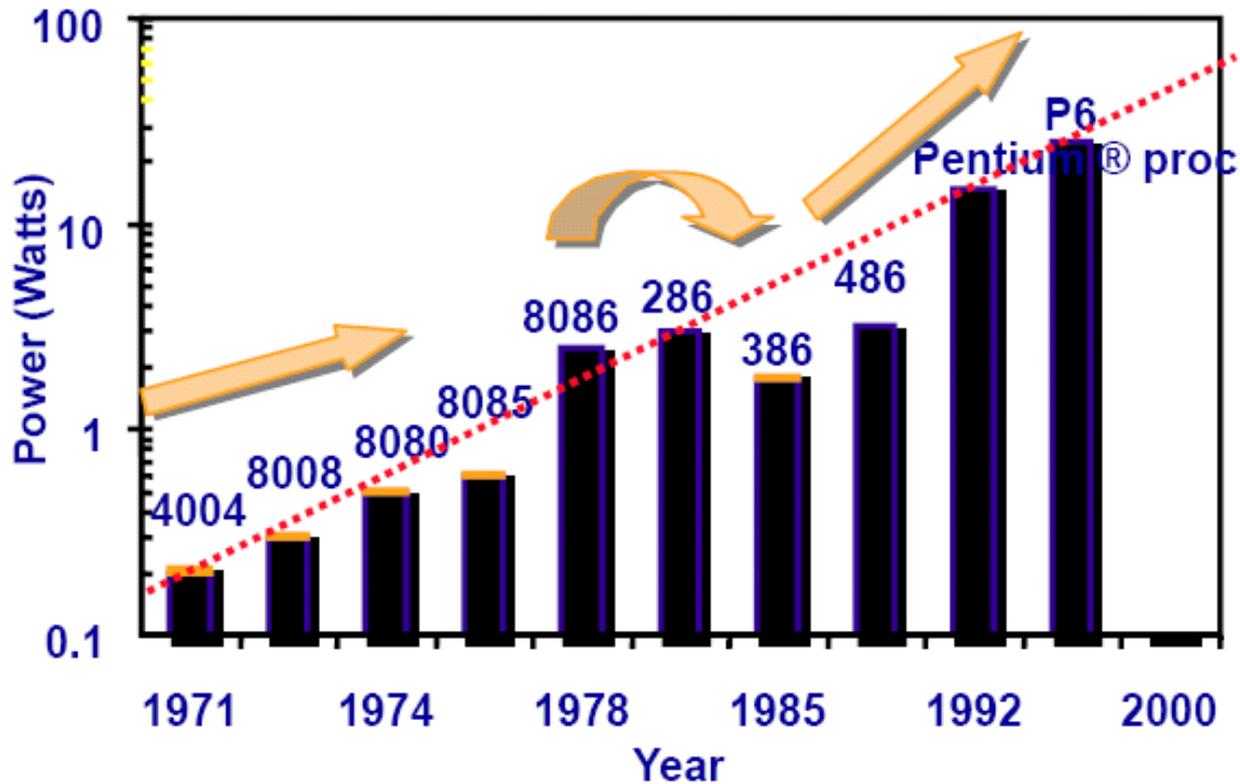
Static Power and Energy

- Because leakage current flows even when a transistor is off, now *static power* becomes important too.

$$Power_{static} = Current_{static} \times Voltage$$

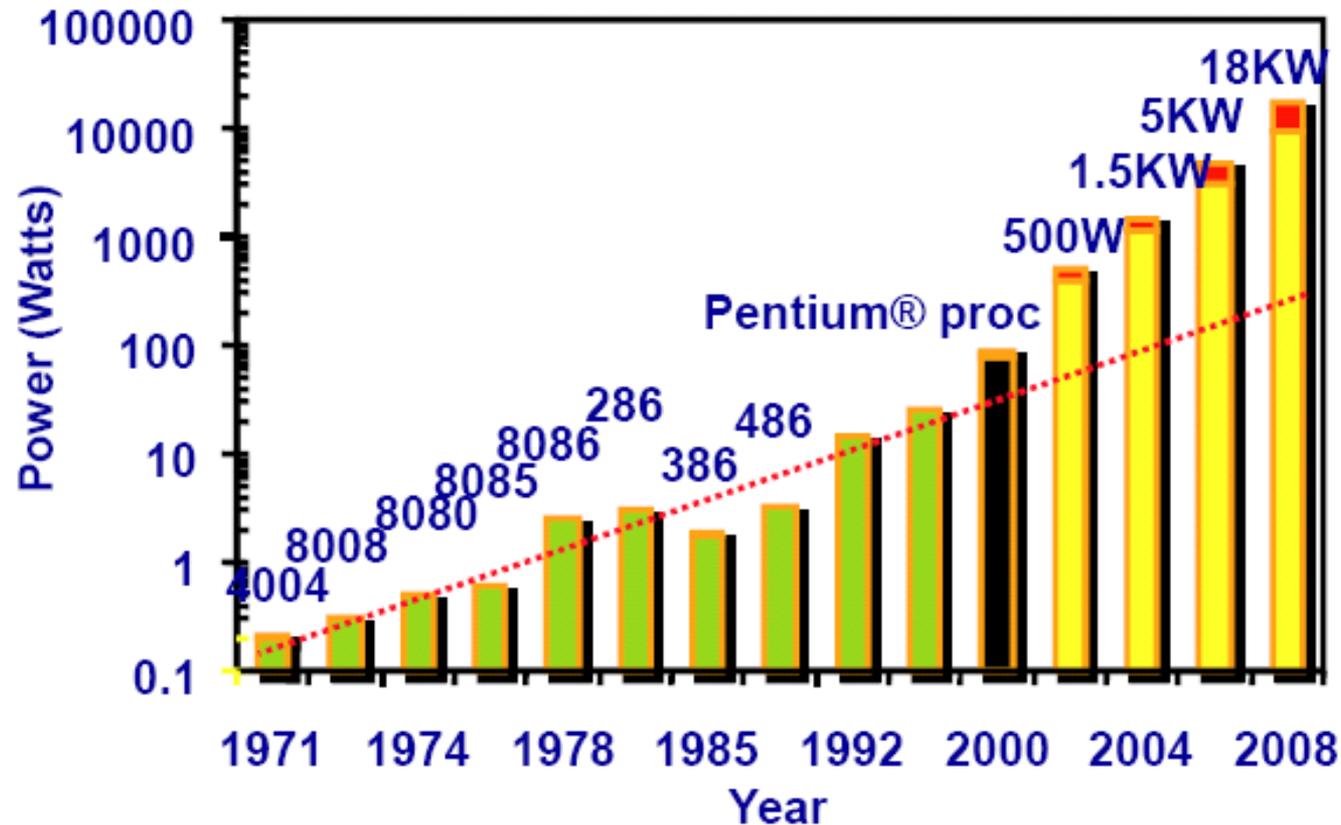
- Leakage current increases in processors with smaller transistor sizes
- Increasing the number of transistors increases power even if they are turned off
- In 2006, goal for leakage is 25% of total power consumption; high performance designs at 40%
- Very low power systems even gate voltage to inactive modules to control loss due to leakage

Power dissipation



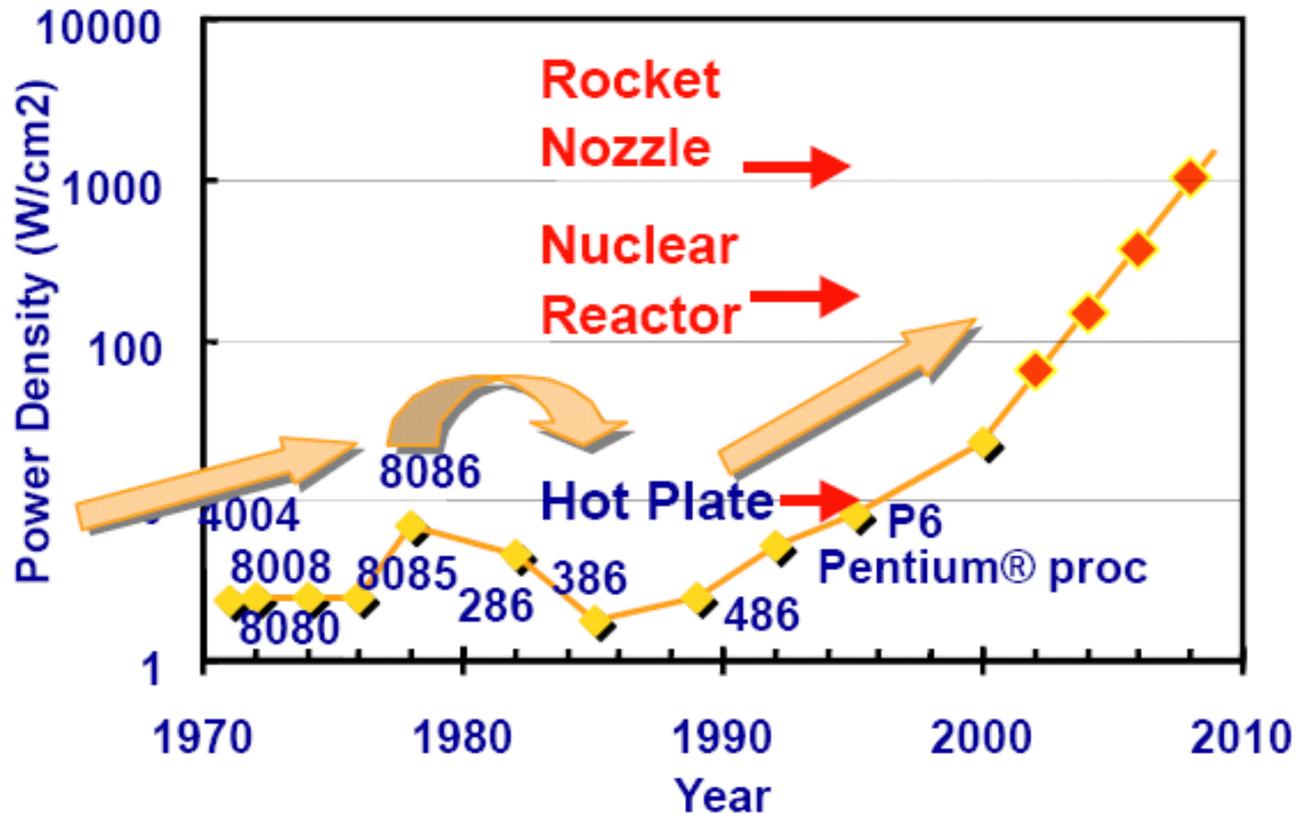
Lead Microprocessors power continues to increase

Power will be a major problem



Power delivery and dissipation will be prohibitive

Power density



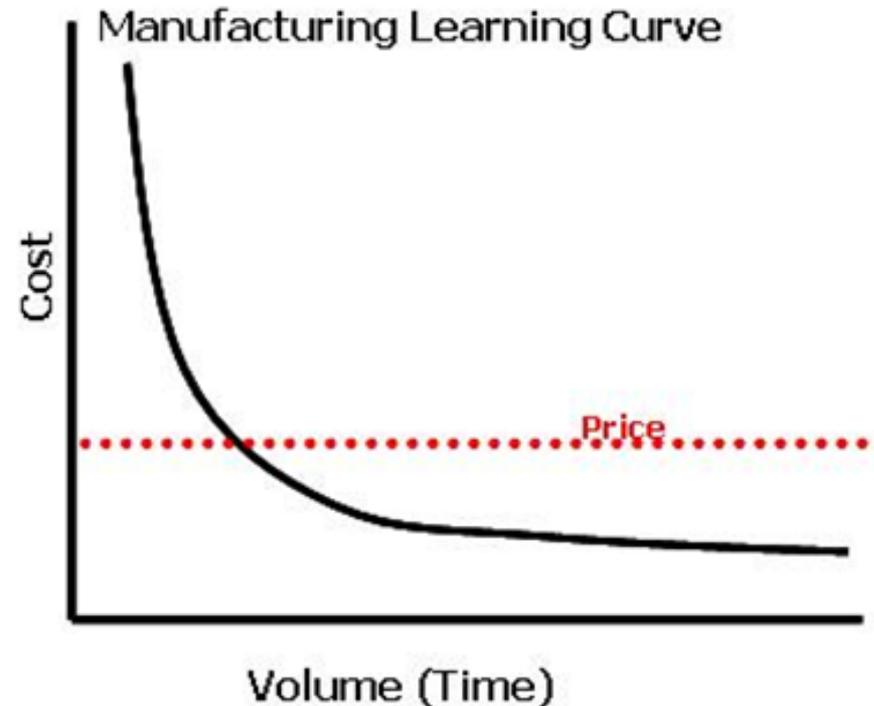
Power density too high to keep junctions at low temp

Low Power Tech. in Modern Processor

- **Do nothing well:** turn off the clock of inactive modules to save energy and dynamic power.
- **Dynamic Voltage-Frequency Scaling (DVFS)**
- **Design for typical case:** offer low power modes to save energy.
- **Overclocking:** Intel started offering **Turbo mode** in 2008.

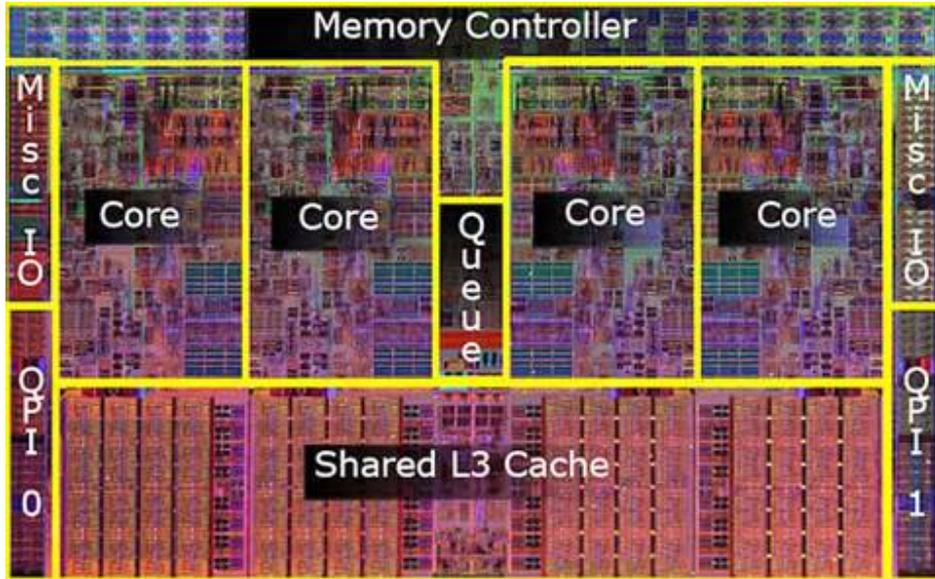
1.6 Trends in Cost

- **Time:** The price drops with time, learning curve increases
- **Volume:** The price drops with volume increase
- **Commodities:** Many manufacturers produce the same product, Competition brings prices down

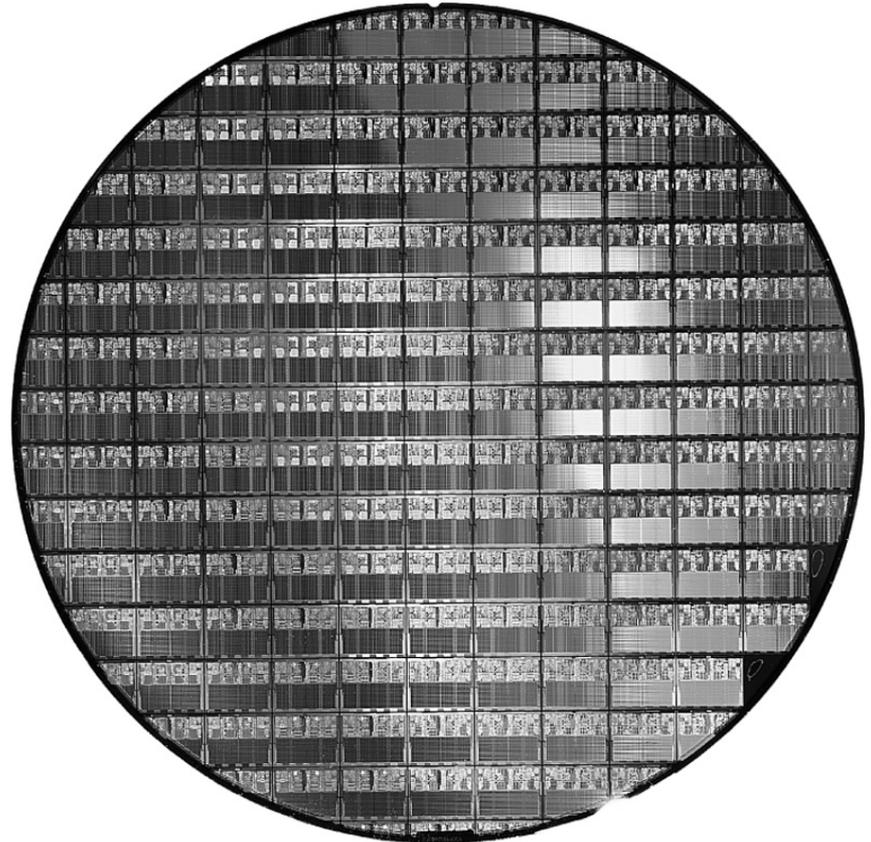


In the past 25 years, much of the personal computer industry has become a commodity business.

Die and Wafer



Photograph of an Intel Core i7 microprocessor die. The dimensions are 18.9 mm by 13.6 mm (257 mm²) in a 45 nm process. (Courtesy Intel.)



© 2007 Elsevier, Inc. All rights reserved.

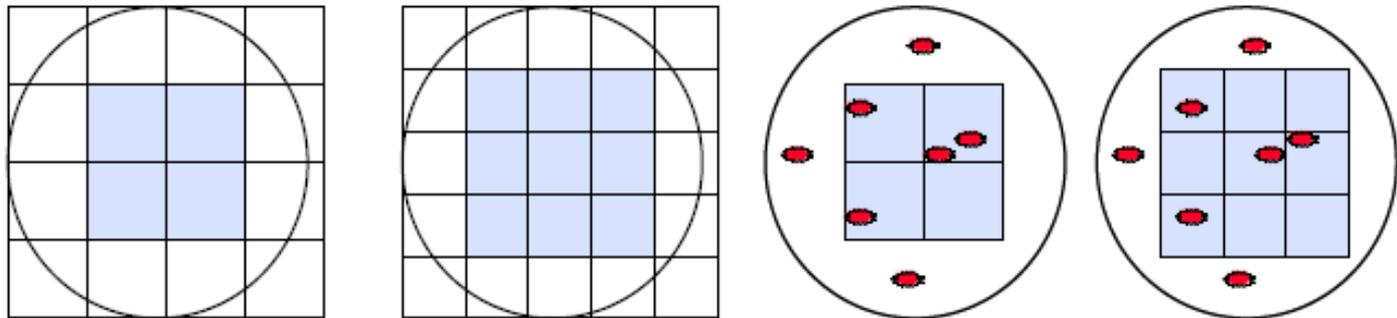
This 300 mm wafer contains 280 full sandy bridge dies, each 20.7 by 10.5 mm in a 32 nm processor.

Cost of Integrated Circuit

$$\text{IC cost} = \frac{\text{Die cost} + \text{Testing cost} + \text{Packaging cost}}{\text{Final test yield}}$$

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} * \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi * (\text{Wafer_diam} / 2)^2}{\text{Die Area}} - \frac{\pi * \text{Wafer_diam}}{\sqrt{2} * \text{Die Area}} - \text{Test dies}$$



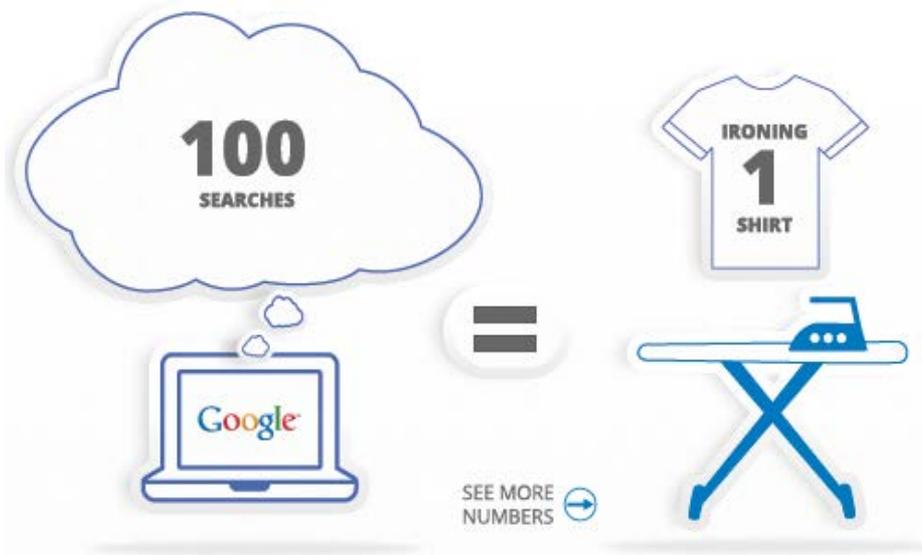
$$\text{Die Yield} = \text{Wafer yield} * \left\{ 1 + \frac{\text{Defects_per_unit_area} * \text{Die_Area}}{\alpha} \right\}^{-\alpha}$$

Die Cost goes roughly with die area⁴

Cost of Integrated Circuit

- **Example:** Find the number of dies per 300mm wafer for a die that is 1.5 cm on a side and for a die that is 1.0 on a side.
- **Example:** Find the die yield for dies that are 1.5cm on a side and 1.0cm on a side, assuming a defect density of 0.031 per cm^2 and N is 13.5.

Cost of Manufacturing vs. Cost of Operation



- Google's data center electricity use is about 0.01% of total worldwide electricity use and less than 1 percent of global data center electricity use in 2010
- **Green Power**

1.7 Dependability

- ▶ How decide when a system is operating properly?
- ▶ Infrastructure providers now offer Service Level Agreements (SLA) to guarantee that their networking or power service would be dependable
- ▶ Systems alternate between 2 states of service with respect to an SLA:
 1. **Service accomplishment**, where the service is delivered as specified in SLA
 2. **Service interruption**, where the delivered service is different from the SLA
- ▶ **Failure** = transition from state 1 to state 2
- ▶ **Restoration** = transition from state 2 to state 1

Dependability

- ▶ **Module reliability** = measure of continuous service accomplishment (or time to failure). 2 metrics
 1. **Mean Time To Failure (MTTF)** measures Reliability
 2. **Failures In Time (FIT)** = $1/\text{MTTF}$, the rate of failures
 - Traditionally reported as failures per billion hours of operation
- ▶ **Mean Time To Repair (MTTR)** measures Service Interruption
 - ▶ **Mean Time Between Failures (MTBF)** = $\text{MTTF} + \text{MTTR}$
- ▶ **Module availability** measures service as alternate between the 2 states of accomplishment and interruption (number between 0 and 1, e.g. 0.9)
- ▶ **Module availability** = $\text{MTTF} / (\text{MTTF} + \text{MTTR})$

1.8 Measuring, Reporting, and Summarizing Performance

Plane	DC to Paris	Speed	Passengers	Throughput (pmp)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concorde	3 hours	1350 mph	132	178,200

- **Time to run the task (ExTime)**
 - Execution time, response time, latency
- **Tasks per day, hour, week, sec, ns ... (Performance)**
 - Throughput, bandwidth

Performance Comparison

- **"X is n times faster than Y" means**

$$\frac{\text{ExTime}(Y)}{\text{ExTime}(X)} = \frac{\text{Performance}(X)}{\text{Performance}(Y)}$$

- **Speed of Concorde vs. Boeing 747**
- **Throughput of Boeing 747 vs. Concorde**

Benchmarks

- **Real applications and application suites**
 - E.g., SPEC CPU2000, SPEC2006, TPC-C, TPC-H
- **Kernels**
 - “Representative” parts of real applications
 - Easier and quicker to set up and run
 - Often not really representative of the entire app
- **Toy programs, synthetic benchmarks, etc.**
 - Not very useful for reporting
 - Sometimes used to test/stress specific functions/features

SPEC CPU (Integer)

Benchmark name by SPEC generation

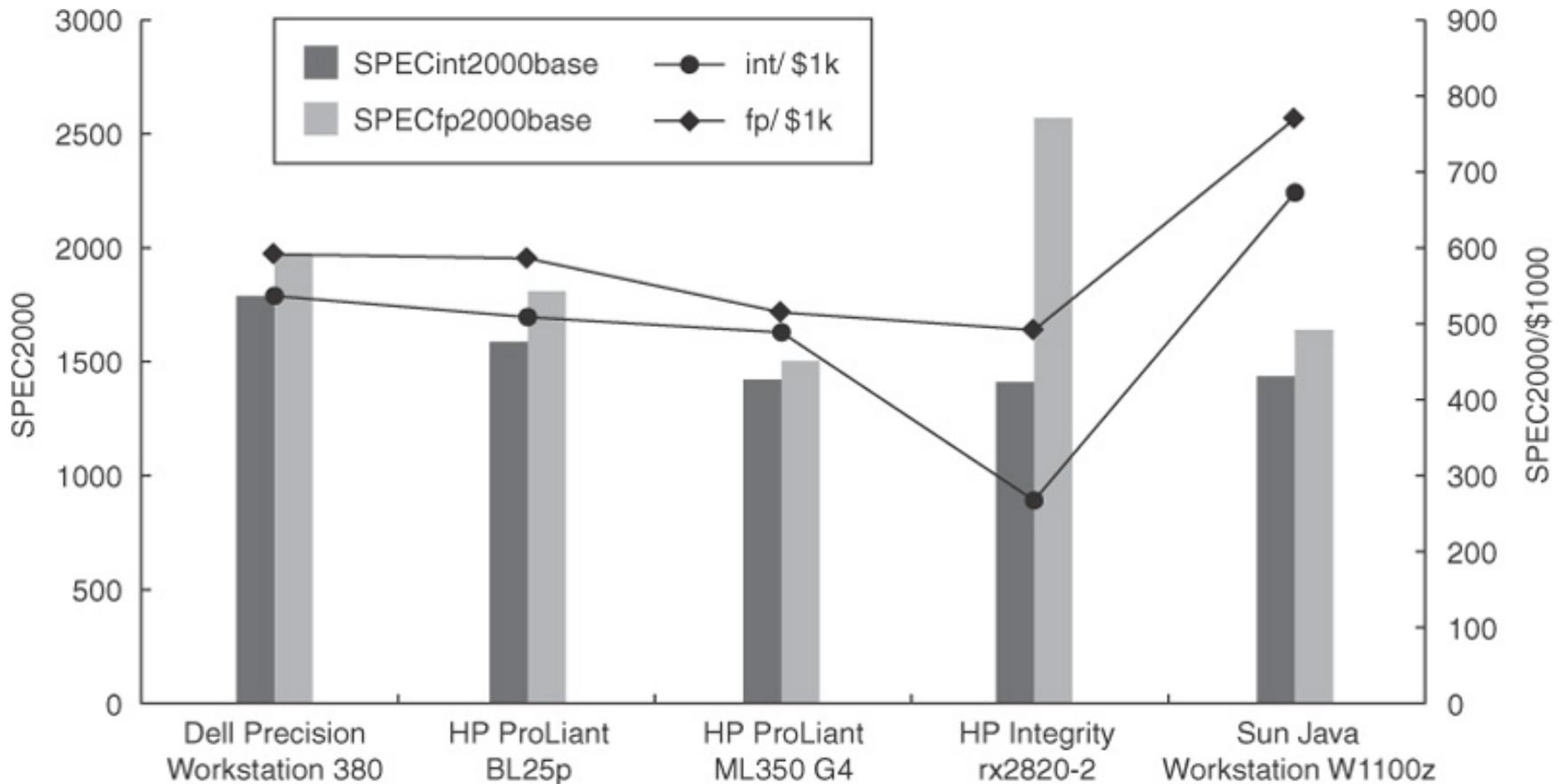
SPEC2006 benchmark description	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
GNU C compiler					gcc
Interpreted string processing			perl		espresso
Combinatorial optimization		mcf			li
Block-sorting compression		bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path 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			

“Representative” applications keeps growing with time!

SPEC CPU (Floating Point)

CFD/blast waves	bwaves			fpppp
Numerical relativity	cactusADM			tomcatv
Finite element code	calculix			doduc
Differential equation solver framework	deall			nasa7
Quantum chemistry	gamess			spice
EM solver (freq/time domain)	GemsFDTD		swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	
Lattice Boltzman method (fluid/air flow)	lbm		mgrid	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	hydro2d
Lattice quantum chromodynamics	milc	apply	turb3d	su2cor
Molecular dynamics	namd	galgel		wave5
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	ampp		
Magneto hydrodynamics (astrophysics)	zeusmp	lucas		
		fma3d		
		sixtrack		

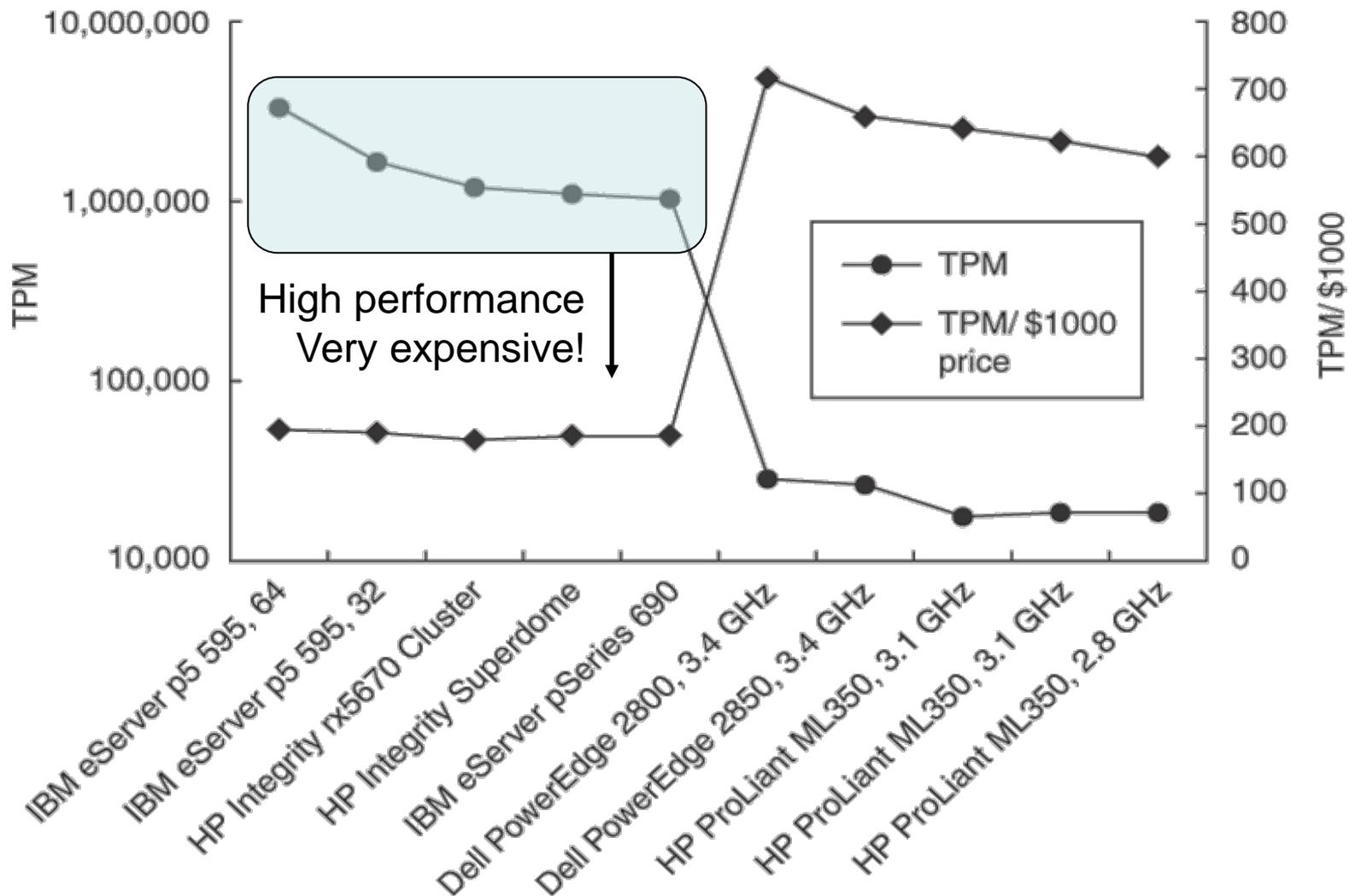
Price-Performance



TPC Benchmarks

- **Measure transaction-processing throughput**
- **Benchmarks for different scenarios**
 - **TPC-C: warehouses and sales transactions**
 - **TPC-H: ad-hoc decision support**
 - **TPC-W: web-based business transactions**
- **Difficult to set up and run on a simulator**
 - **Requires full OS support, a working DBMS**
 - **Long simulations to get stable results**

Throughput-Server Perf/Cost



1.9 Quantitative Principles of Computer Design

- **Take Advantage of Parallelism**
 - Data level parallelism
 - Request level parallelism
 - Instruction level parallelism
- **Principle of Locality, program property**
 - Temporal locality
 - Spatial locality
- **Focus on the Common Case**

Amdahl's Law (I)

- Amdahl's law defines the *speedup* that can be gained by using a particular feature.

$$\text{Speedup} = \frac{\text{Execution Time without Enhancement}}{\text{Execution Time with Enhancement}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}}$$

What if enhancement does not enhance everything?

$$\text{Speedup} = \frac{\text{Execution Time without using Enhancement at all}}{\text{Execution Time using Enhancement when Possible}}$$

$$\text{Execution Time}_{\text{new}} = \text{Execution Time}_{\text{old}} \times \left((1 - \text{Fraction}_{\text{Enhanced}}) + \frac{\text{Fraction}_{\text{Enhanced}}}{\text{Speedup}_{\text{Enhanced}}} \right)$$

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

Caution: fraction
of What?

Amdahl's Law (II)

- **Make the Common Case Fast**

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

$\text{Speedup}_{\text{Enhanced}} = 20$ $\text{Fraction}_{\text{Enhanced}} = 0.1$ **VS** $\text{Speedup}_{\text{Enhanced}} = 1.2$ $\text{Fraction}_{\text{Enhanced}} = 0.9$

$$\text{Speedup} = \frac{1}{\left((1 - 0.1) + \frac{0.1}{20} \right)} = 1.105$$

$$\text{Speedup} = \frac{1}{\left((1 - 0.9) + \frac{0.9}{1.2} \right)} = 1.176$$

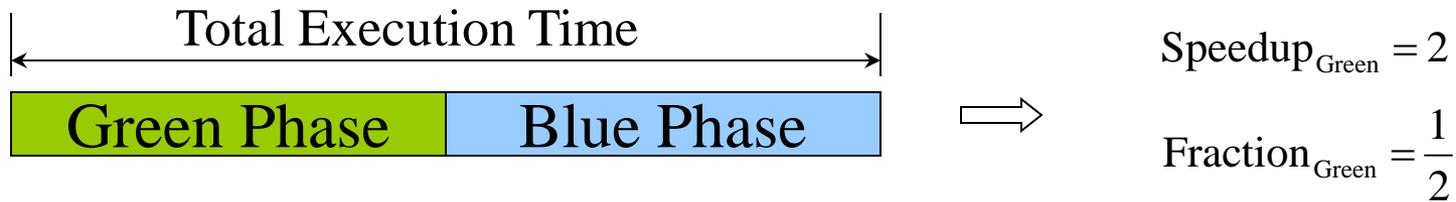
Important: Principle of locality

Approx. 90% of the time spent in 10% of the code

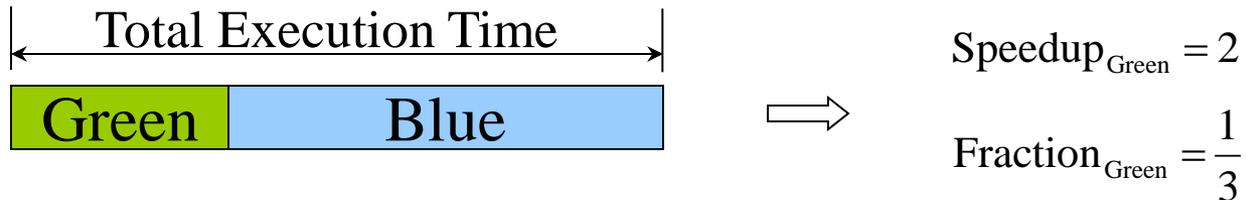
Amdahl's Law (III)

- **Diminishing Returns**

Generation 1



Generation 2 $\text{Speedup}_{\text{Overall}} = 1.33$ over Generation 1



Generation 3 $\text{Speedup}_{\text{Overall}} = 1.2$ over Generation 2



Car Analogy

- 
- From GT to Mall of Georgia (35mi)
 - you’ve got a “Turbo” for your car, but can only use on highway
 - Spaghetti Junction to Mall of GA (23mi)
 - avg. speed of 60mph
 - avg. speed of 120mph with Turbo
 - GT to Spaghetti junction (12 mi)
 - stuck in *bad* rush hour traffic
 - avg. speed of 5 mph

Turbo gives *100% speedup across 66% of the distance...*
... but only results in <10% reduction on **total trip time**
(which is a <11% speedup)

Now Consider Price-Performance

- **Without Turbo**

- Car costs \$8,000 to manufacture
- Selling price is \$12,000 → \$4K profit per car
- If we sell 10,000 cars, that's \$40M in profit

- **With Turbo**

- Car costs extra \$3,000
- Selling price is \$16,000 → \$5K profit per car
- **But only a few gear heads buy the car:**
 - We only sell 400 cars and make \$2M in profit

CPU Design is Similar

- **What does it cost me to add some performance enhancement?**
- **How much effective performance do I get out of it?**
 - 100% speedup for small fraction of time wasn't a big win for the car example
- **How much more do I have to charge for it?**
 - Extra development, testing, marketing costs
- **How much more can I charge for it?**
 - Does the market even care?
- **How does the price change affect volume?**

The Processor Performance Equation

$$\text{CPU time} = \text{CPU Clock Cycles} \times \text{Clock cycle time}$$

$$\text{CPU time} = \text{Instruction Count} \times \text{Cycles Per Instruction} \times \text{Clock cycle time}$$

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

CPI: clock cycles per instruction

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

IPC: instructions per clock, the inverse of CPI

Aspects of CPU Performance

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

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X		
Inst. Set.	X	X	
Organization		X	X
Technology			X

Car Analogy

- Need to drive from Klaus to CRC
 - “Clock Speed” = 3500 RPM
 - “CPI” = 5250 rotations/km *or* 0.19 m/rot
 - “Insts” = 800m

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



$$800 \text{ m} \times \frac{1 \text{ rotation}}{0.19 \text{ m}} \times \frac{1 \text{ minute}}{3500 \text{ rotations}}$$

= 1.2 minutes

CPU Version

- Program takes 33 billion instructions to run
- CPU processes insts at 2 cycles per inst
- Clock speed of 3GHz

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

Sometimes clock cycle time given instead (ex. cycle = 333 ps)

IPC sometimes used instead of CPI

= 22 seconds

The Processor Performance Equation (2)

CPU time = CPU Clock Cycles \times Clock cycle time

$$\text{CPU time} = \left(\sum_{i=1}^n \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

For each kind
of instruction

How many cycles it
takes to execute an
instruction of this kind

How many instructions
of this kind are there in
the program

CPU performance w/ different instructions

Instruction Type	Frequency	CPI
Integer	40%	1.0
Branch	20%	4.0
Load	20%	2.0
Store	10%	3.0

$$\text{CPU time} = \left(\sum_{i=1}^n \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

Total Insts = 50B, Clock speed = 2 GHz

The overall CPI

$$\text{CPI} = \frac{\sum_{i=1}^n \text{IC}_i \times \text{CPI}_i}{\text{Instruction count}} = \sum_{i=1}^n \frac{\text{IC}_i}{\text{Instruction count}} \times \text{CPI}_i$$

Comparing Performance

- **“X is n times faster than Y”**

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

- **“Throughput of X is n times that of Y”**

$$\frac{\text{Tasks per unit time}_X}{\text{Tasks per unit time}_Y} = n$$

If Only it Were That Simple

- “X is n times faster than Y on A”

$$\frac{\text{Execution time of app A on machine Y}}{\text{Execution time of app A on machine X}} = n$$

- **But what about different applications (or even parts of the same application)**
 - X is 10 times faster than Y on A, and 1.5 times on B, but Y is 2 times faster than X on C, and 3 times on D, and...

So does X have better performance than Y?

Which would you buy?

Summarizing Performance

- **Arithmetic mean**
 - Average execution time
 - Gives more weight to longer-running programs
- **Weighted arithmetic mean**
 - More important programs can be emphasized
 - But what do we use as weights?
 - Different weight will make different machines look better

Speedup

	Machine A	Machine B
Program 1	5 sec	4 sec
Program 2	3 sec	6 sec

What is the speedup of A compared to B on Program 1?

What is the speedup of A compared to B on Program 2?

What is the average speedup?

What is the speedup of A compared to B on Sum(Program1, Program2) ?

Normalizing & the Geometric Mean

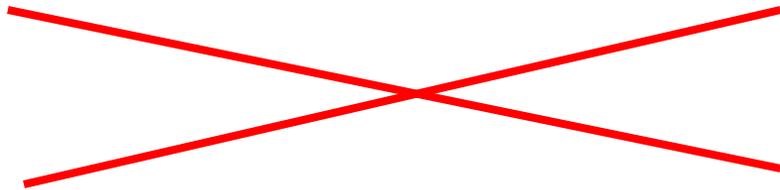
- **Speedup of arithmetic means \neq arithmetic mean of speedup**
- **Use geometric mean:** $\sqrt[n]{\prod_{i=1}^n \text{Normalized execution time on } i}$
- **Neat property of the geometric mean:**
Consistent whatever the reference machine
- **Do not use the arithmetic mean for normalized execution times**

CPI/IPC

- **Often when making comparisons in comp-arch studies:**
 - Program (or set of) is the same for two CPUs
 - The clock speed is the same for two CPUs
- **So we can just directly compare CPI's and often we use IPC's**

Average CPI vs. "Average" IPC

- Average CPI = $(CPI_1 + CPI_2 + \dots + CPI_n)/n$



Not Equal to A.M. of CPI!!!

- A.M. of IPC = $(IPC_1 + IPC_2 + \dots + IPC_n)/n$
- Must use *Harmonic Mean* to remain ∞ to runtime

Harmonic Mean

- **H.M.($x_1, x_2, x_3, \dots, x_n$) =**

$$\frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \frac{1}{x_3} + \dots + \frac{1}{x_n}}$$

- **What in the world is this?**
 - Average of inverse relationships

A.M.(CPI) vs. H.M.(IPC)

- “Average” IPC =
$$\frac{1}{\text{A.M.}(CPI)}$$
$$= \frac{1}{\frac{CPI_1}{n} + \frac{CPI_2}{n} + \frac{CPI_3}{n} + \dots + \frac{CPI_n}{n}}$$
$$= \frac{n}{CPI_1 + CPI_2 + CPI_3 + \dots + CPI_n}$$
$$= \frac{1}{\frac{1}{IPC_1} + \frac{1}{IPC_2} + \frac{1}{IPC_3} + \dots + \frac{1}{IPC_n}} = \text{H.M.}(IPC)$$