CESE 4085 Modern Computer Architecture

Lectures 1 & 2: Fundamentals

Administrative Information

- Course Instructors
  - Stephan Wong
  - Carlo Galuzzi
- Open door policy (appointment)
- Brightspace: announcements, slides, grades, …
- For urgent matters: MS Teams

Rules

- TU Delft regulations fully apply
- Special situations/problems – notify the instructors in advance
- Check the CESE 4085 Brightspace page regularly
- Always ask when you do not understand
- Feedback is welcome

Transition year (2024-2025)

- **New setup of the course in 2024-2025**
- Pointers to improve course are much appreciated
- Reading assignments no longer part of course
- Added a new topic: “compilers”, especially link with computer architecture
- Introduced 2 new labs (each 3 afternoons) – both must be succesfully passed to pass the course
- Course grade will be based on an exam (end of quarter)

2 new labs (2024-2025)

Lab 1:

- Afternoon 0: (at home) CVA6 intro, installation of tools
- Afternoon 1: introduction to SystemVerilog, simple FIFO
- Afternoon 2: branch prediction
- Afternoon 3: caching

Lab 2:

- Afternoon 0:
- Afternoon 1:
- Afternoon 2:
- Afternoon 3:

Course Objectives

- Provide theoretical and practical insight on computer architecture
  - Fundamental concepts and their application in the architecture of recent micro-processors and multi-processors.
  - Trade-offs (performance, cost, energy consumption) and their complexity
- Understand how to approach processor, multiprocessor, memory, interconnect, ... design
  - How difficult the problem is
  - Design challenges and metrics
- Provide … on compilers
  - sdfasdf
- Understand how ...
  - asdfadf

MCA Lecture Schedule (2024-2025)

Week	Date	Content	Topic
Week 1	Mon. 10-02	CA Fundamentals (1)	Course intro, short history, definitions, RISC-V, technology trends, power, dependability
	Wed. 12-02	CA Fundamentals (2)	Performance, benchmarks, Amdahl's law, CPU, design principles
Week 2	Mon. 17-02	Memories (1)	Hierarchy, caches (block placement, performance (1))
	Wed. 19-02	Memories (2)	Caches (performance (2)), virtual memory
Week 3	Mon. 24-02	ILP (1)	Dependences vs. hazards, exposing ILP, branch prediction, dynamic scheduling, Tomasulo
	Wed. 26-02	ILP (2)	Multiple issue, VLIW, BTB, RAP, pitfalls & fallacies
Week 4	Mon. 03-03	DLP (1)	(can still be changed) Vector processors, SIMD *Other options can be: DLP-GPUs, Warehouse computing, domain-specific computing*
	Wed. 05-03	DLP (2)	(can still be changed) Multithreading, multiprocessing (cache coherence, snooping, directory) *Other options can be: DLP-GPUs, Warehouse computing, domain-specific computing*
Week 5	Mon. 10-03		
	Wed. 12-03		
Week 6	Mon. 17-03		
	Wed. 19-03		
Week 7	Mon. 24-03		
	Wed. 26-03		
Week 8	Mon. 31-03		
	Wed. 02-04		
Week 9	-		
	-		
Week 10	-		
	-		

MCA Labs Schedule (2024-2025)

Week	Date	Content	Topic
Week 1	-		
	-		
Week 2	Fr. 21-02	No Lab	
	-		
Week 3	Fr. 28-02	Lab 1 - Afternoon 1	Introduction to SystemVerilog and simple FIFO example
	-		
Week 4	Fr. 07-03	Lab 1 - Afternoon 2	Branch Prediction
	-		
Week 5	Fr. 14-03	Lab 1 – Afternoon 3	Caching
	-		
Week 6	Fr. 21-03	Lab 2 – Afternoon 1	
	-		
Week 7	Fr. 28-03	Lab 2 – Afternoon 2	
	-		
Week 8	Fr. 04-04	Lab 2 – Afternoon 3	
	-		
Week 9	-		
	-		
Week 10	-		
	-		

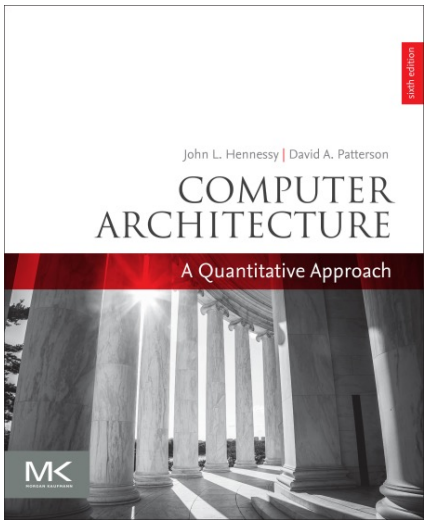
Grading Scheme (2024-2025)

- Succesfully pass both labs

- Obtain a pass (>= 6.0) for the exam

Study Material

- J.L. Hennessy and D.A. Patterson, Computer Architecture: A Quantitative Approach, 6th edition, Morgan Kaufman, 2019, ISBN 978-0-12-811905-1 (eBook ISBN: 9780128119068)



Chapter 1

Fundamentals of Quantitative Design and Analysis

Lecture 1 and 2 Overview (Lecture Goals)

Lecture 1:

- A short history (highlights)
- Trends in Computer Architecture & Computer Classes
- Defining Computer Architecture
- RISC-V ISA – the basics
- More trends (bandwidth vs latency, transistors)
- Power and Energy, Cost, Dependability

Lecture 2:

- Performance = time
- Benchmarks
- Performance → Arithmetic mean vs. geometric mean
- Amdahl's law
- CPI
- Fallacies & pitfalls, design principles

Combining topics of 6th and earlier editions of CA book

Alan Turing

The German Enigma machine
Decode by Turing-Welchman Bombe

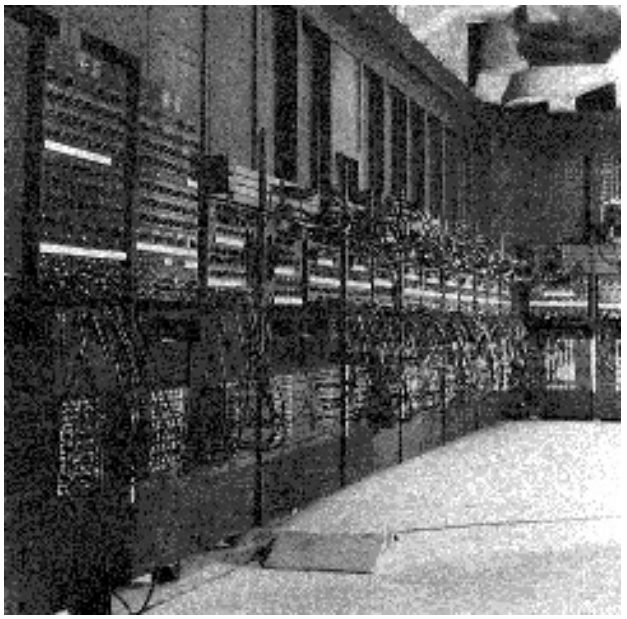
1912 (23 June): Birth, Paddington, London
1926-31: Sherborne School
1930: Death of friend Christopher Morcom
1931-34: Undergraduate at King's College, Cambridge University
1932-35: Studies quantum mechanics, probability, logic
1935: Elected fellow of King's College, Cambridge
1936: The Turing machine: *On Computable Numbers...* submitted
1936-38: At Princeton University. Ph.D. Papers in logic, algebra, number theory
1938-39: Return to Cambridge. Introduced to German Enigma cipher problem
1939-40 Devises the Bombe, machine for Enigma decryption
1939-42: Breaking of U-boat Enigma cipher, saving battle of the Atlantic
1943-45: Chief Anglo-American consultant. Introduced to electronics
1945: National Physical Laboratory, London
1946: Computer design, leading the world, formally accepted
1947-48: Papers on programming, neural nets, and prospects for artificial intelligence
1948: Manchester University
1949: Work on programming and world's first serious use of a computer
1950: Philosophical paper on machine intelligence: the Turing Test
1951: Elected FRS. Paper on non-linear morphogenesis theory
1952: Arrested and tried as a homosexual, loss of security clearance
1953-54: Unfinished work in biology and physics
1954 (7 June): Death by cyanide poisoning, Wilmslow, Cheshire.

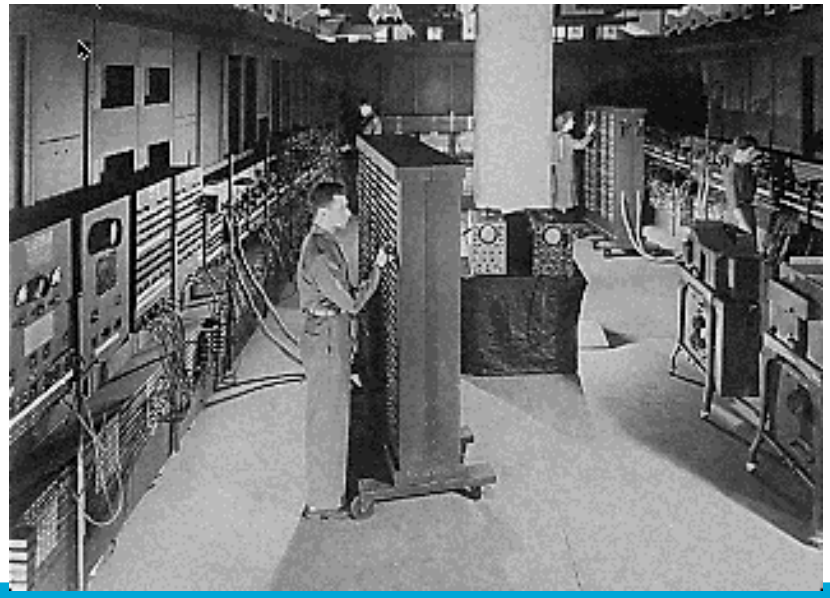
A bit of history: Eniac

Ecker & Mauchly
Moore School of the U. Of Pennsylvania

Used to compute ballistic trajectories.
Each new problem required rewiring
and setting 3000 switches.

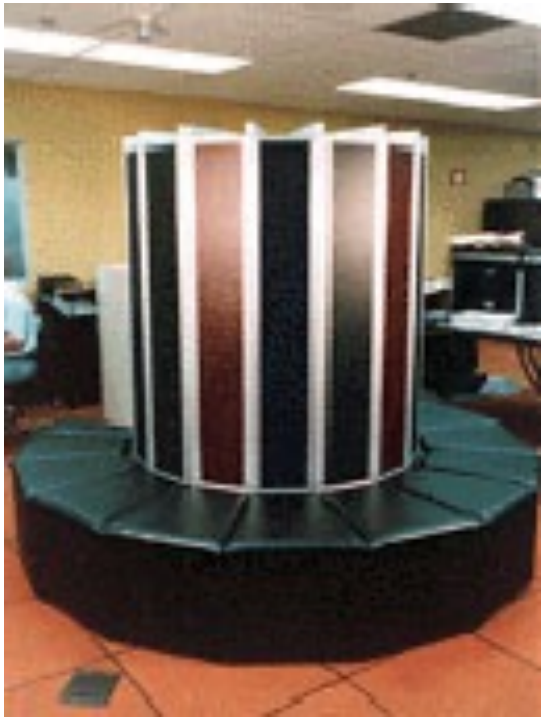
Later Von Neumann joined the project.





1943-1945
30 tons
Consumed 200 kilowatts
19000 vacuum tubes

Cray



The first Cray-1® system was installed at Los Alamos National Laboratory in 1976 for $8.8 million.
It boasted a world-record speed of 160 million floating-point operations per second (160 megaflops) and an 8 megabyte (1 million word) main memory.
The Cray-1's architecture reflected its designer's penchant for bridging technical hurdles with revolutionary ideas. In order to increase the speed of this system, the Cray-1 had a unique "C" shape which enabled integrated circuits to be closer together. No wire in the system was more than four feet long.
To handle the intense heat generated by the computer, Cray developed an innovative refrigeration system using Freon.

Computer Technology

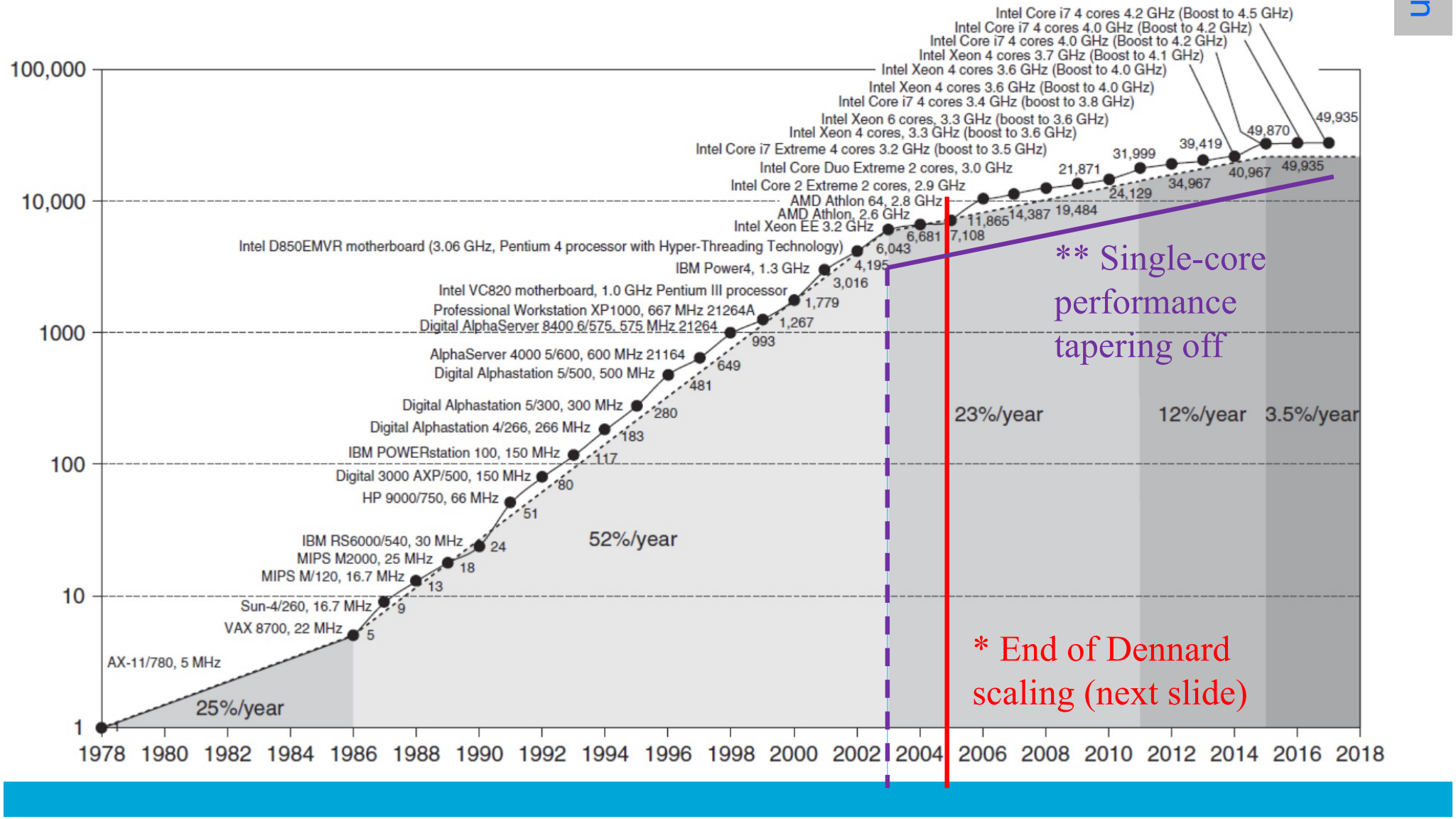
Performance improvements:

- Improvements in semiconductor technology
  - Feature size, clock speed
- Improvements in computer architectures
  - Enabled by HLL compilers, UNIX
  - Lead to RISC architectures

Together have enabled:

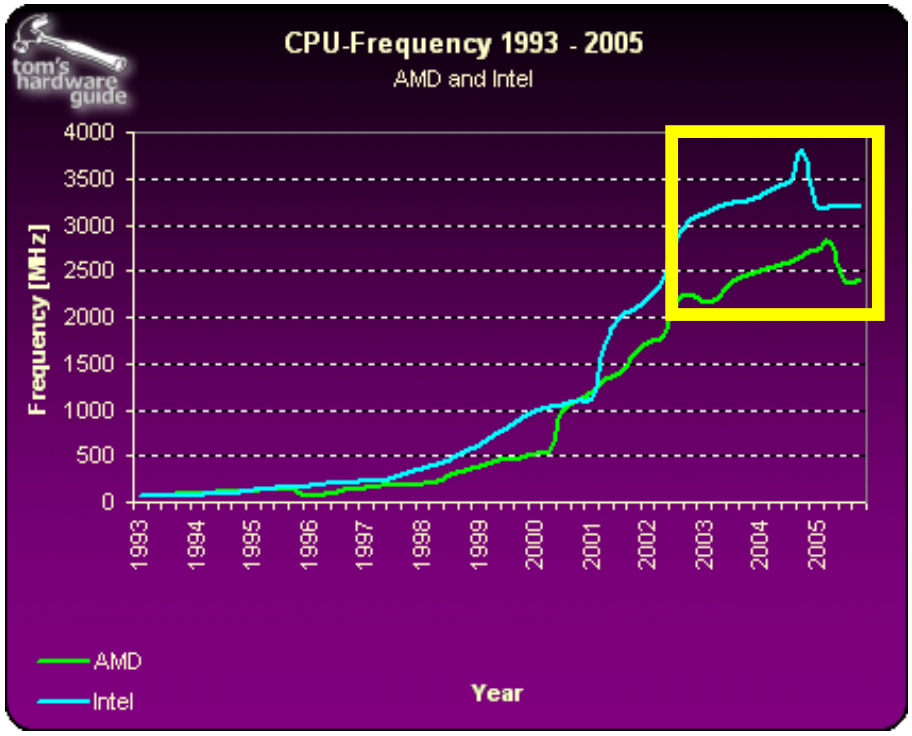
- Lightweight computers
- Productivity-based managed/interpreted programming languages

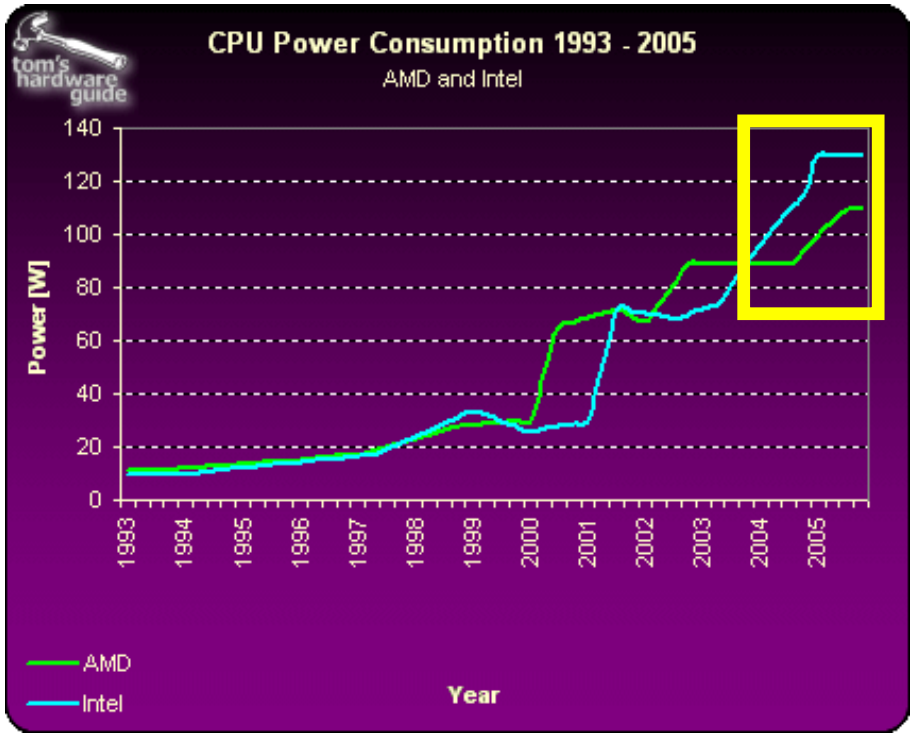
Single Processor Performance



~2005: Frequency & Power leveling off

From EE1D2 – Digital Systems course





From: www.tomshardware.com

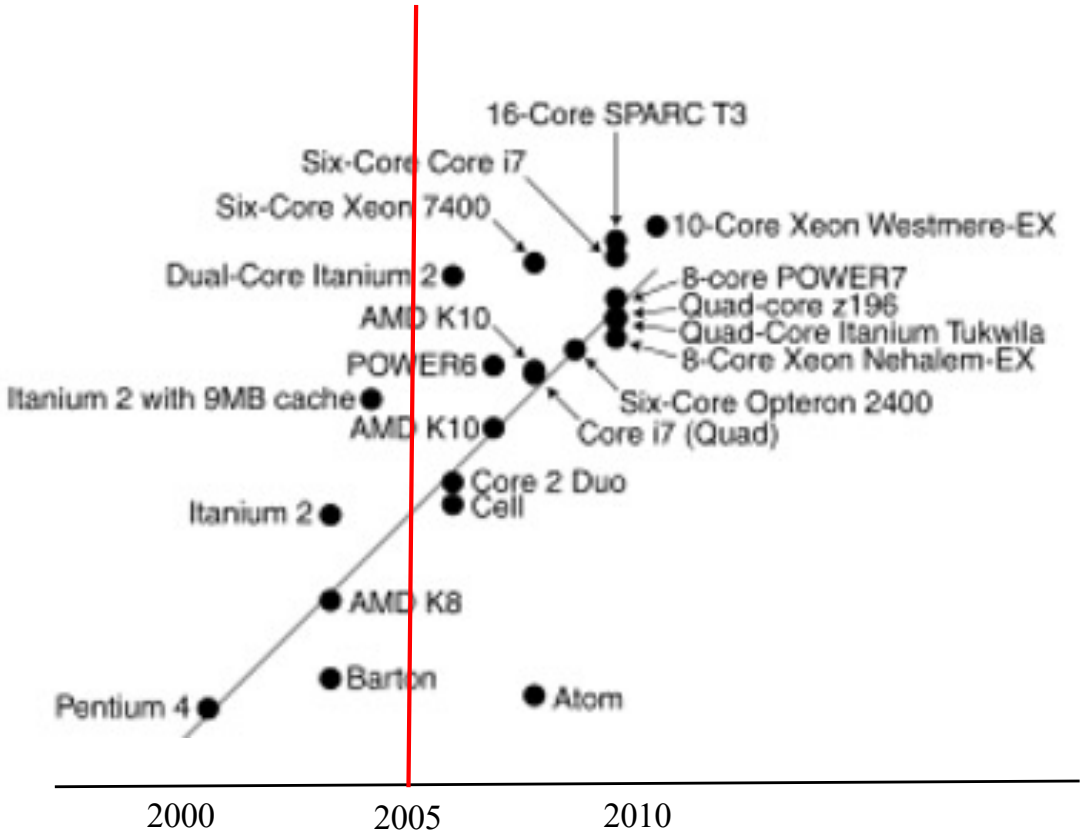
- Dennard Scaling (power density remains constant) ended 2005-2007
- However, Moore's Law (#transistors doubles every ~2 yrs) continued
- What was the effect??

Homogeneous Computing

From EE1D2 – Digital Systems course

Terminology (after 2005):

- Dual-core
- Quad-core
- Six-core
- 8-core
- 10-core
- 16-core
- "Just" more of the same core



Current Trends in Architecture

Cannot continue to leverage Instruction-Level parallelism (ILP)

- Single processor performance improvement ended in 2003

New models for performance:

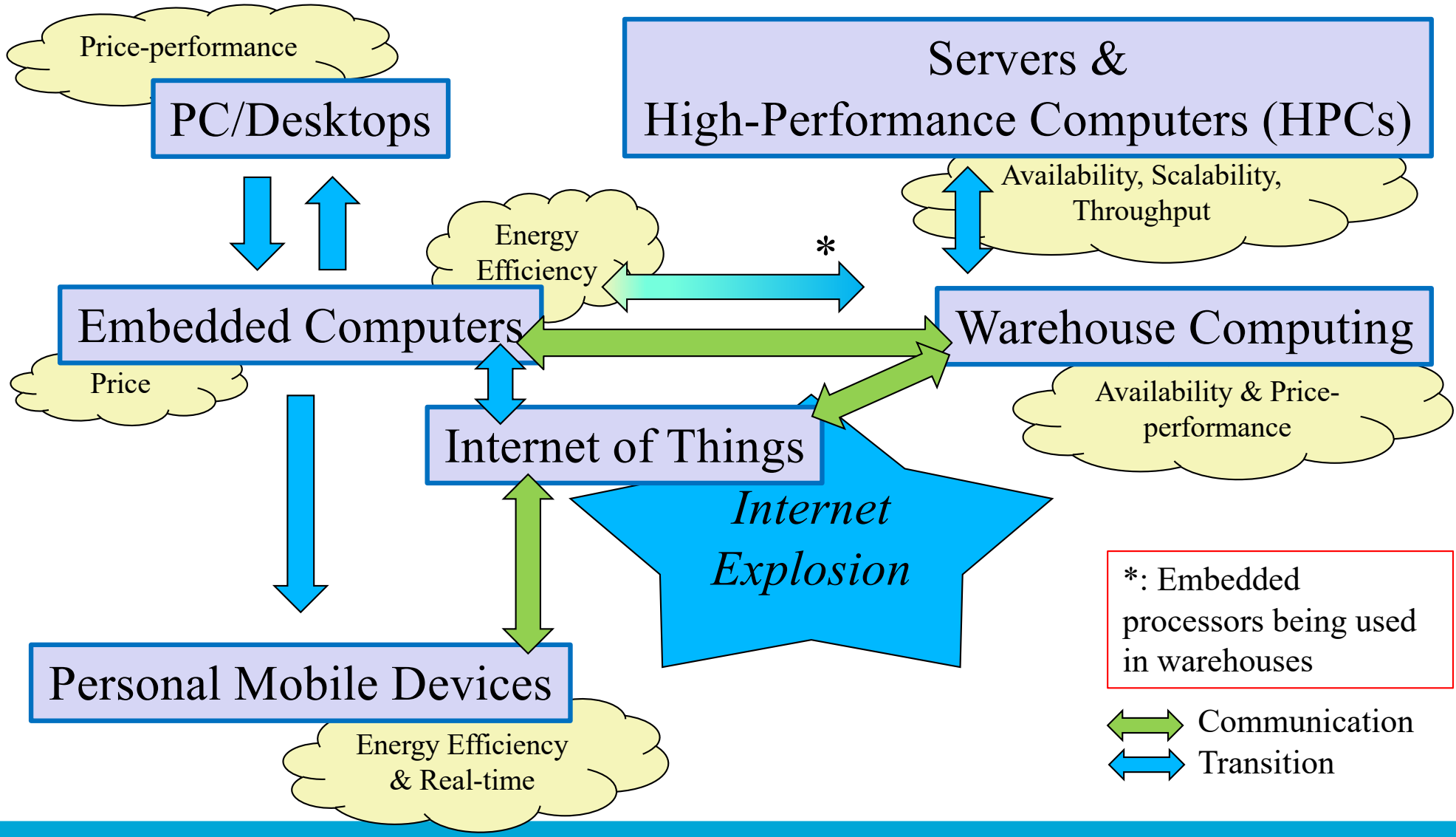
- Data-level parallelism (DLP)
- Thread-level parallelism (TLP)
- Request-level parallelism (RLP)

These require explicit restructuring of the application

Classes of Computers

- Personal Mobile Device (PMD)
  - e.g. start phones, tablet computers
  - Emphasis on energy efficiency and real-time
- Desktop Computing
  - Emphasis on price-performance
- Servers
  - Emphasis on availability, scalability, throughput
- Clusters / Warehouse Scale Computers
  - Used for "Software as a Service (SaaS)"
  - Emphasis on availability and price-performance
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks
- Internet of Things/Embedded Computers
  - Emphasis: price

A Visual (My Personal Interpretation)



Parallelism

Classes of parallelism in applications:

- Data-Level Parallelism (DLP)
- Task-Level Parallelism (TLP)

Classes of architectural parallelism:

- Instruction-Level Parallelism (ILP)
- Vector architectures/Graphic Processor Units (GPUs)
- Thread-Level Parallelism
- Request-Level Parallelism

Flynn's Taxonomy (1966)

Single instruction stream, single data stream (SISD)

Single instruction stream, multiple data streams (SIMD)

- Vector architectures
- Multimedia extensions
- Graphics processor units

Multiple instruction streams, single data stream (MISD)

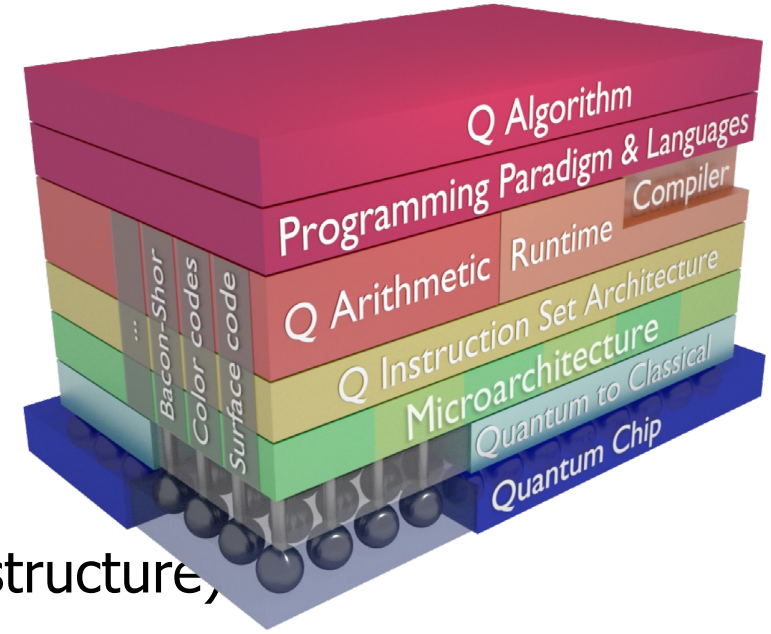
- No commercial implementation

Multiple instruction streams, multiple data streams (MIMD)

- Tightly-coupled MIMD
- Loosely-coupled MIMD

Architecture/Organization/Implementation

What is Computer Architecture?



Traditional view —> 3 levels:

- Architecture
  - Describes the functionalities
  - "it is a book"
- Organization (~ micro-architecture)
  - Describes the major components (structure)
- Implementation
  - Technology

- —> **Advantages**:
  - Independent development, (program) compatibility over decades
  - Exploit advantages in all 3 levels (e.g., coding, pipelining, CMOS)

Defining Computer Architecture

"Old" view of computer architecture:

- Instruction Set Architecture (ISA) design
- i.e. decisions regarding:
  - registers, memory addressing, addressing modes, instruction operands, available operations, control flow instructions, instruction encoding

"Real" computer architecture:

- Specific requirements of the target machine
- Design to maximize performance within constraints: cost, power, and availability
- Includes ISA, microarchitecture, hardware

Iron Law

From CSE4130 – Computer Engineering course

Iron Law

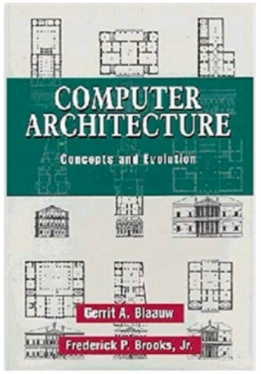
$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) (CPI) (cycle time)

Architecture --> Implementation --> Realization

Compiler Designer **Processor Designer** **Chip Designer**



Remember Blaauw Brooks?

19

Computer Engineering course

There is a more in-depth introduction to the RISC-V ISA in the Computer Engineering course (CESE4130) with much more details.

Instruction Set Architecture

(1) Class of ISA

- General-purpose registers
- Register-memory vs load-store

RISC-V registers

- 32 g.p., 32 f.p.

Register	Name	Use	Saver
x0	zero	constant 0	n/a
x1	ra	return addr	caller
x2	sp	stack ptr	callee
x3	gp	gbl ptr	
x4	tp	thread ptr	
x5-x7	t0-t2	temporaries	caller
x8	s0/fp	saved/ frame ptr	callee
x9	s1	saved	callee
x10-x17	a0-a7	arguments	caller
x18-x27	s2-s11	saved	callee
x28-x31	t3-t6	temporaries	caller
f0-f7	ft0-ft7	FP temps	caller
f8-f9	fs0-fs1	FP saved	callee
f10-f17	fa0-fa7	FP arguments	callee
f18-f27	fs2-fs21	FP saved	callee
f28-f31	ft8-ft11	FP temps	caller

Instruction Set Architecture

(2) Memory addressing

RISC-V: byte addressed, aligned accesses faster

(3) Addressing modes

RISC-V: Register, immediate, displacement (base+offset)

Other examples: autoincrement, indexed, PC-relative

(4) Types and size of operands

RISC-V: 8-bit, 32-bit, 64-bit

Instruction Set Architecture

(5) Operations

RISC-V: data transfer, arithmetic, logical, control, floating point

See Fig. 1.5 in text

(6) Control flow instructions

Use content of registers (RISC-V) vs. status bits (x86, ARMv7, ARMv8)

Return address in register (RISC-V, ARMv7, ARMv8) vs. on stack (x86)

(7) Encoding

Fixed (RISC-V, ARMv7/v8 except compact instruction set) vs. variable length (x86)

Trends in Technology

Integrated circuit technology (Moore's Law)

- Transistor density: 35%/year
- Die size: 10-20%/year
- Integration overall: 40-55%/year

DRAM capacity: 25-40%/year (slowing)

- 8 Gb (2014), 16 Gb (2019), possibly no 32 Gb

Flash capacity: 50-60%/year

- 8-10X cheaper/bit than DRAM

Magnetic disk capacity: recently slowed to 5%/year

- Density increases may no longer be possible, maybe increase from 7 to 9 platters
- 8-10X cheaper/bit then Flash
- 200-300X cheaper/bit than DRAM

Bandwidth and Latency

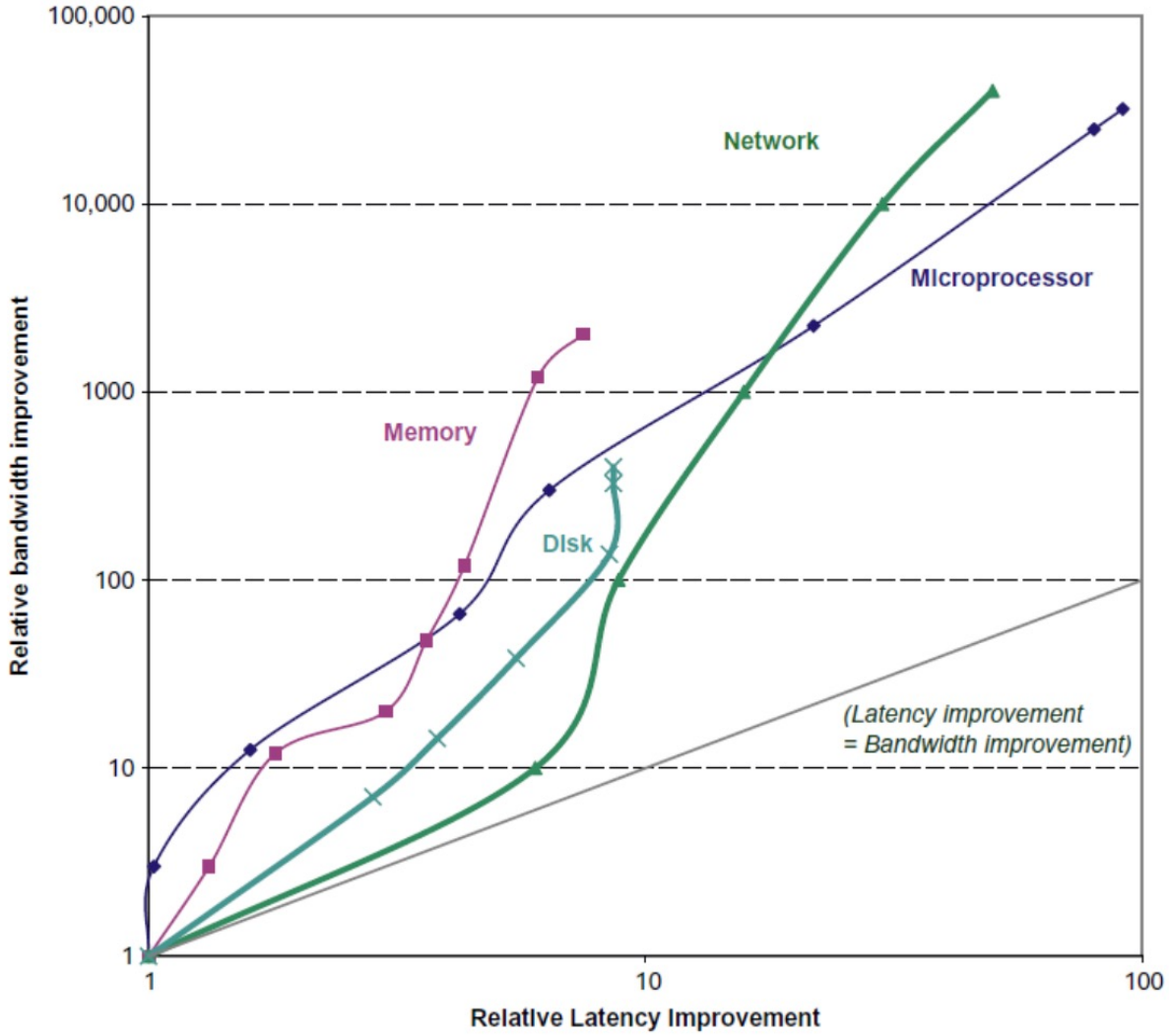
Bandwidth or throughput

- Total work done in a given time
- 32,000-40,000X improvement for processors
- 300-1200X improvement for memory and disks

Latency or response time

- Time between start and completion of an event
- 50-90X improvement for processors
- 6-8X improvement for memory and disks

Bandwidth and Latency



Log-log plot of bandwidth and latency milestones

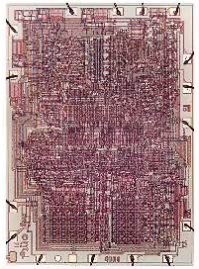
Rule of thumb:

"Bandwidth grows by at least the square of the improvement in latency"

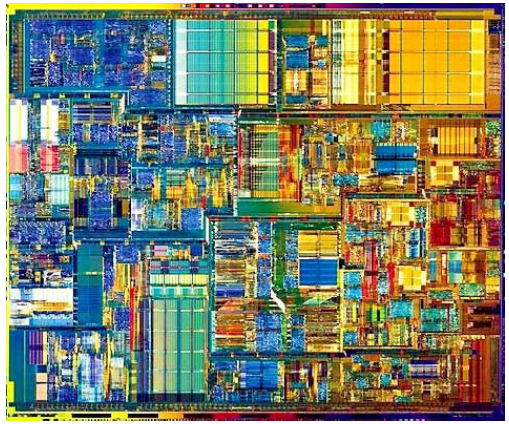
Transistors and Wires

- Feature size
  - Minimum size of transistor or wire in x or y dimension
  - 10 microns in 1971 to .011 microns in 2017
  - Transistor performance scales linearly
    - Wire delay does not improve with feature size!
  - Integration density scales quadratically

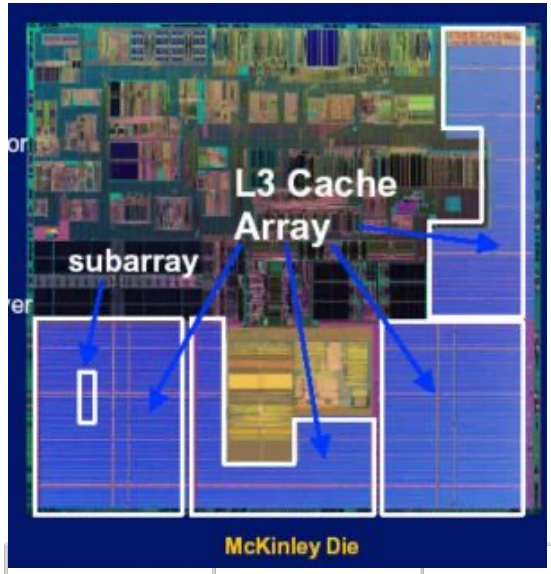
Processor Transistor Count



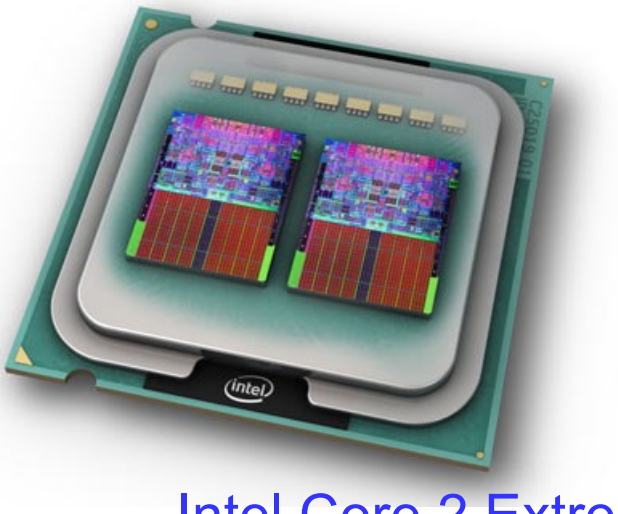
Intel 4004, 2300 tr .(1971)



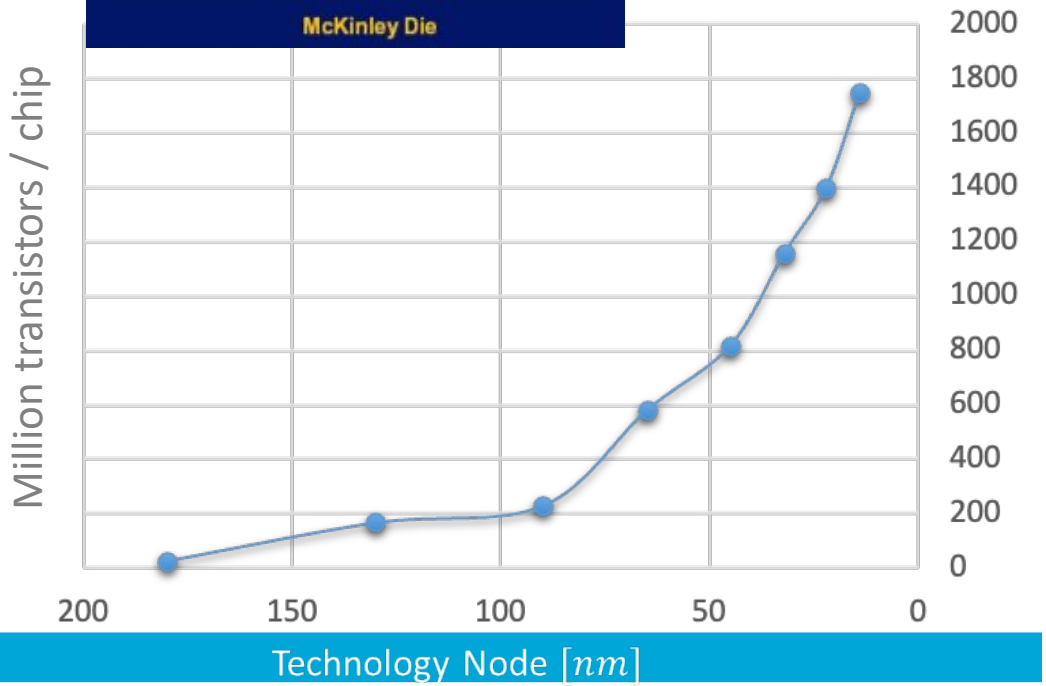
Intel P4 – 55M tr. (2001)



Intel McKinley – 221M tr. (2001)



Intel Core 2 Extreme Quad-core 2x291M tr. (2006)



Technology Directions: SIA Roadmap (from 1999)

Year	1999	2002	2005	2008	2011	2014	
Feature size (nm)	180	130	100	70	50	35	~V (lower)
Logic trans/cm^2	6.2M	18M	39M	84M	180M	390M	~V (more)
Cost/trans (mc)	1.735	.580	.255	.110	.049	.022	Not checked
#pads/chip	1867	2553	3492	4776	6532	8935	Nc
Clock (MHz)	1250	2100	3500	6000	10000	16900	X (lower)
Chip size (mm^2)	340	430	520	620	750	900	X (lower)
Wiring levels	6-7	7	7-8	8-9	9	10	Nc
Power supply (V)	1.8	1.5	1.2	0.9	0.6	0.5	Nc
High-perf pow (W)	90	130	160	170	175	183	Nc

- International Technology Roadmap for Semiconductors (ITRS) provides yearly updates (www.itrs.net)

More info (from Wikipedia)

Dual-core Itanium 2	1,700,000,000[22]	2006	Intel	90 nm	596 mm²
Six-core Core i7 Ivy Bridge E	1,860,000,000	2013	Intel	22 nm	256 mm²
Duo-core + GPU Core i7 Broadwell-U	1,900,000,000[23]	2015	Intel	14 nm	133 mm²
Six-core Xeon 7400	1,900,000,000	2008	Intel	45 nm	503 mm²
Quad-core Itanium Tukwila	2,000,000,000[24]	2010	Intel	65 nm	699 mm²
Apple A8 (dual-core ARM64 "mobile SoC")	2,000,000,000	2014	Apple	20 nm	89 mm²
8-core POWER7+ 80 MB L3 cache	2,100,000,000	2012	IBM	32 nm	567 mm²
Six-core Core i7/8-core Xeon E5 (Sandy Bridge-E/EP)	2,270,000,000[25]	2011	Intel	32 nm	434 mm²
8-core Xeon Nehalem-EX	2,300,000,000[26]	2010	Intel	45 nm	684 mm²
8-core Core i7 Haswell-E	2,600,000,000[27]	2014	Intel	22 nm	355 mm²
10-core Xeon Westmere-EX	2,600,000,000	2011	Intel	32 nm	512 mm²
Six-core zEC12	2,750,000,000	2012	IBM	32 nm	597 mm²
Apple A8X (tri-core ARM64 "mobile SoC")	3,000,000,000	2014	Apple	20 nm	
8-core Itanium Poulson	3,100,000,000	2012	Intel	32 nm	544 mm²
IBM z13	3,990,000,000	2015	IBM	22 nm	678 mm²
12-core POWER8	4,200,000,000	2013	IBM	22 nm	650 mm²
15-core Xeon Ivy Bridge-EX	4,310,000,000[28]	2014	Intel	22 nm	541 mm²
62-core Xeon Phi	5,000,000,000	2012	Intel	22 nm	
Xbox One main SoC	5,000,000,000	2013	Microsoft/AMD	28 nm	363 mm²
18-core Xeon Haswell-E5	5,560,000,000[29]	2014	Intel	22 nm	661 mm²
IBM z13 Storage Controller	7,100,000,000	2015	IBM	22 nm	678 mm²

Processor (SoC) – 2022 Apple M1 Ultra (20-core, 64-bit) (~114 billion transistors, 5nm)
GPU – 2022 NVIDIA GH100 Hopper (~80 billion transistors, 4nm, 814 mm²)
FPGA – 2021 Xilinx Versal VP1802 (~92 billion transistors, 7nm)

Power and Energy

Problem: Get power in, get power out

Thermal Design Power (TDP)

- Characterizes sustained power consumption
- Used as target for power supply and cooling system
- Lower than peak power (1.5X higher), higher than average power consumption

Clock rate can be reduced dynamically to limit power consumption (*throttling → switching off*)

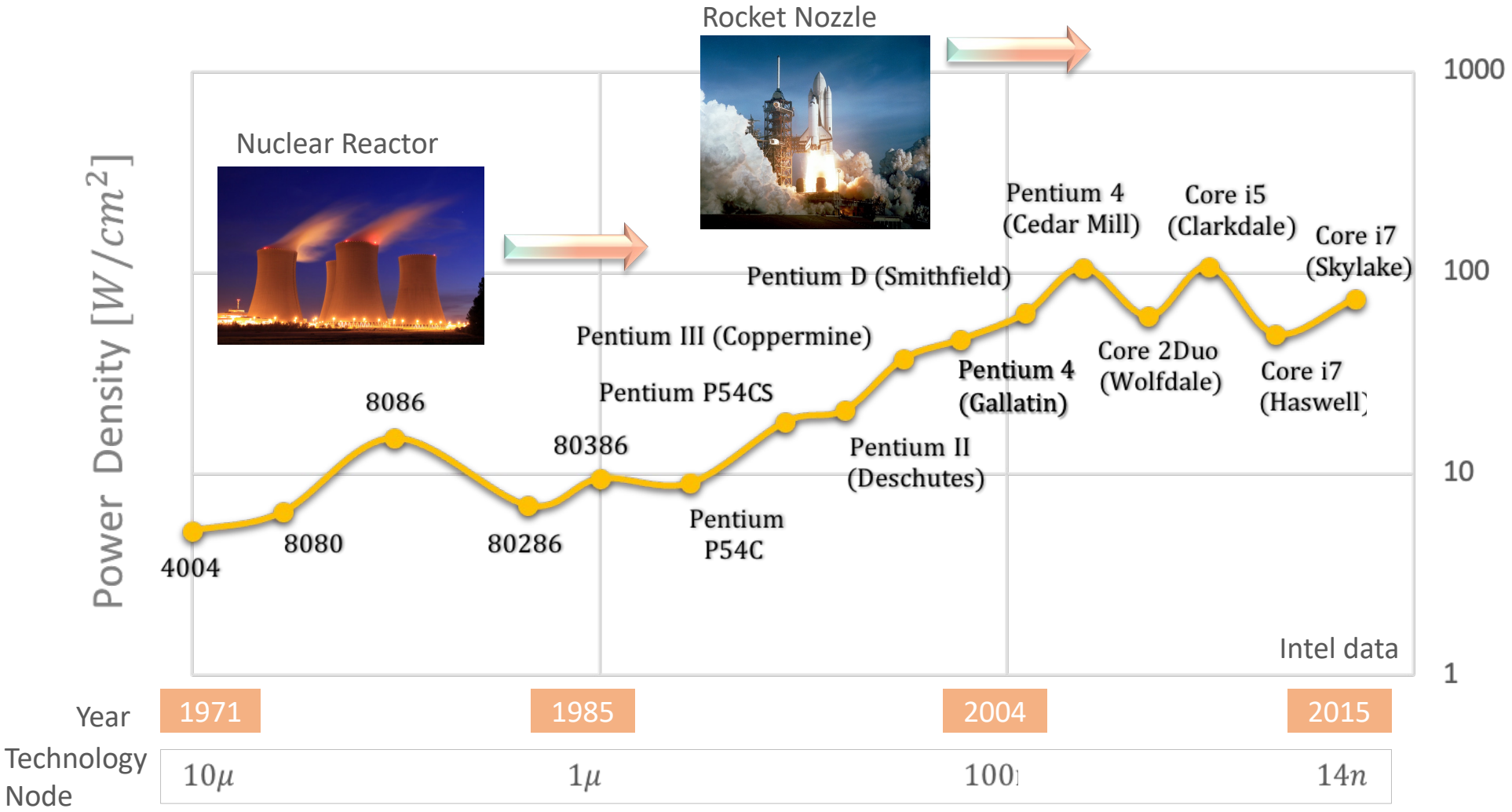
Energy per task is often a better measurement

Trends in Power in ICs

- Power Issues
  - How to bring it in and distribute around the chip? (many pins just for power supply and ground, interconnection layers for distribution)
  - How to remove the heat (dissipated power)
- Why worry about power?
  - Battery life in portable and mobile platforms
  - Power consumption in desktops, server farms
    - Cooling costs, packaging costs, reliability, timing
    - Power density: 30 W/cm2 in Alpha 21364 (3x of typical hot plate)
  - Environment?
    - IT consumes 10% of energy in the US

Power becomes a first class architectural design constraint

CMOS Scaling – Power Density



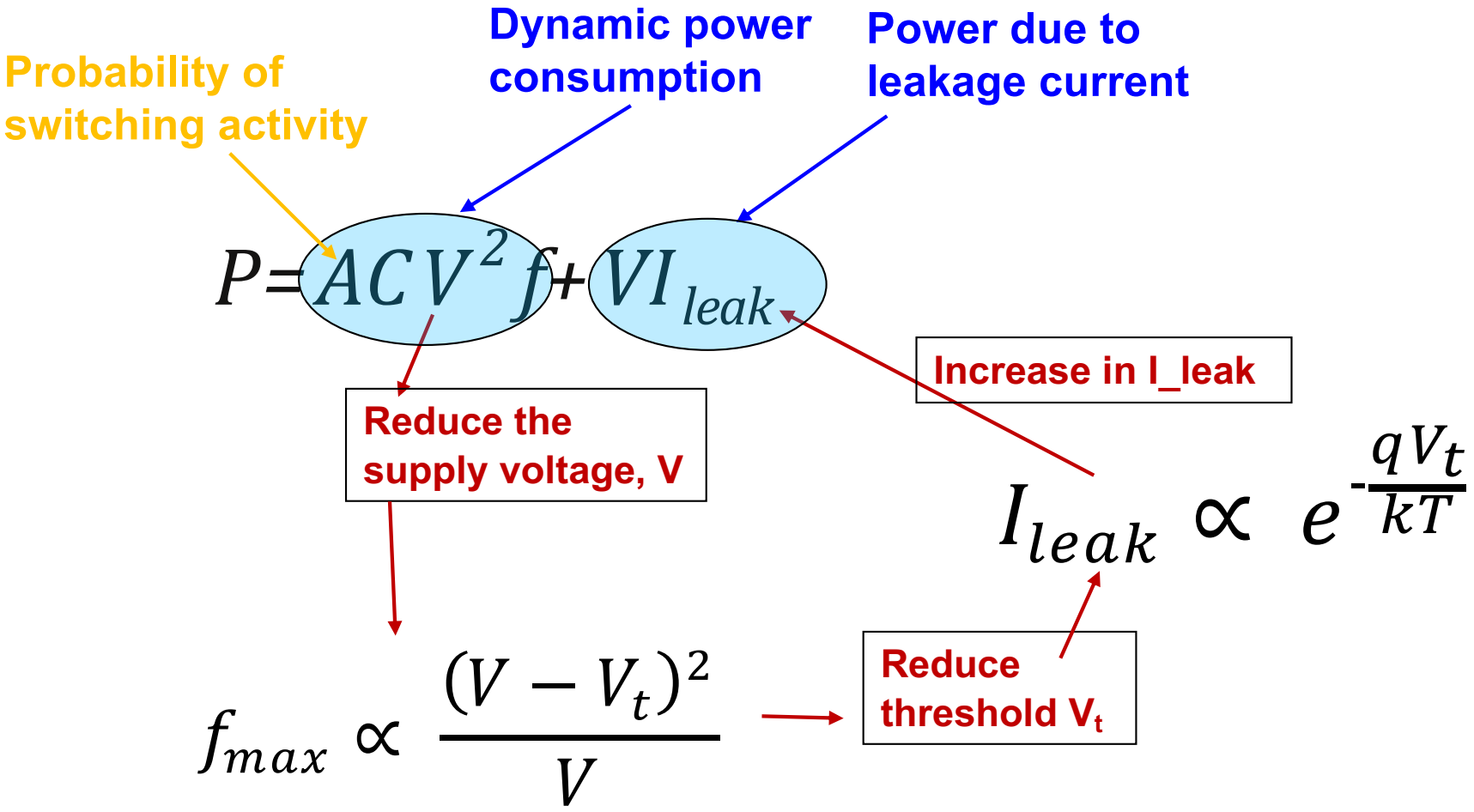
CMOS Power Equations

Dynamic power consumption

Power due to leakage current

$$P = ACV^2 f + VI_{leak}$$

CMOS Power Equations



Dynamic 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

Power

Intel 80386 consumed ~ 2 W

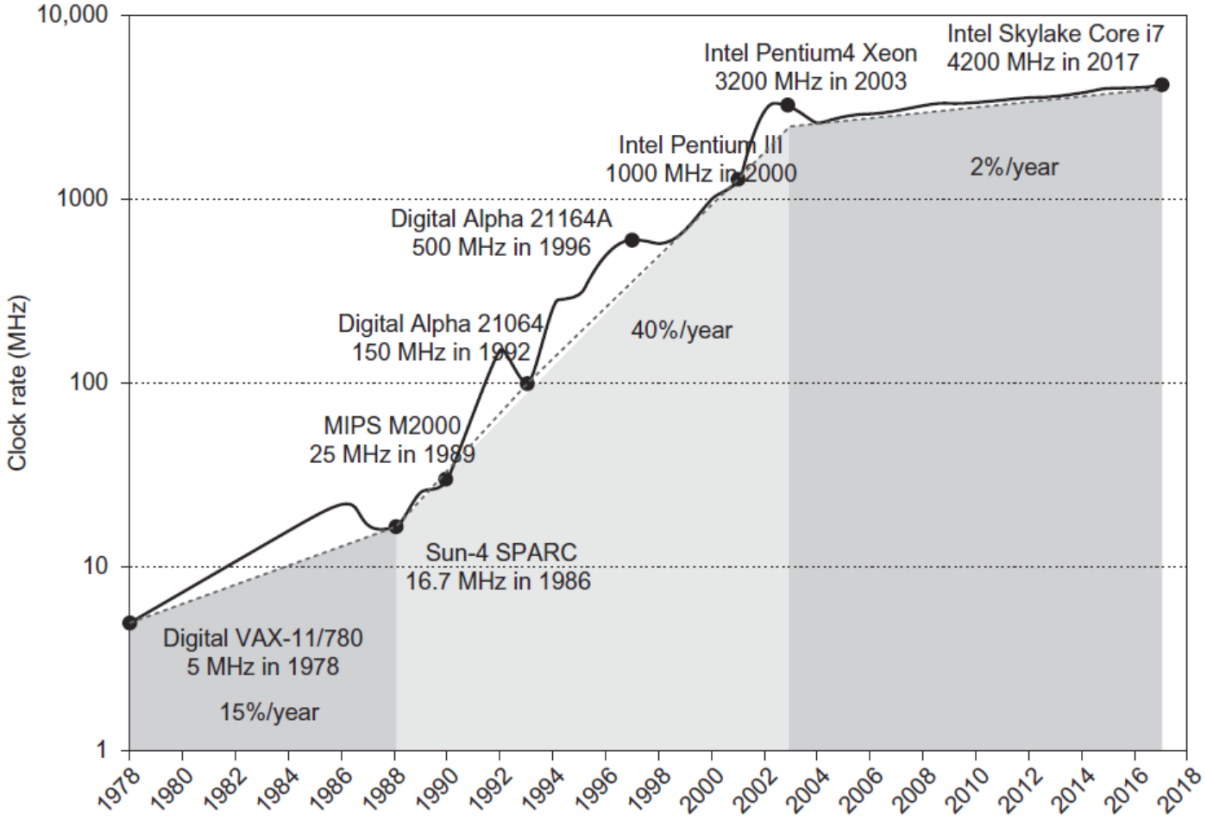
3.3 GHz Intel Core i7 consumes 130 W

Heat must be dissipated from 1.5 x 1.5 cm chip

This is the limit of what can be cooled by air

Sidenote:

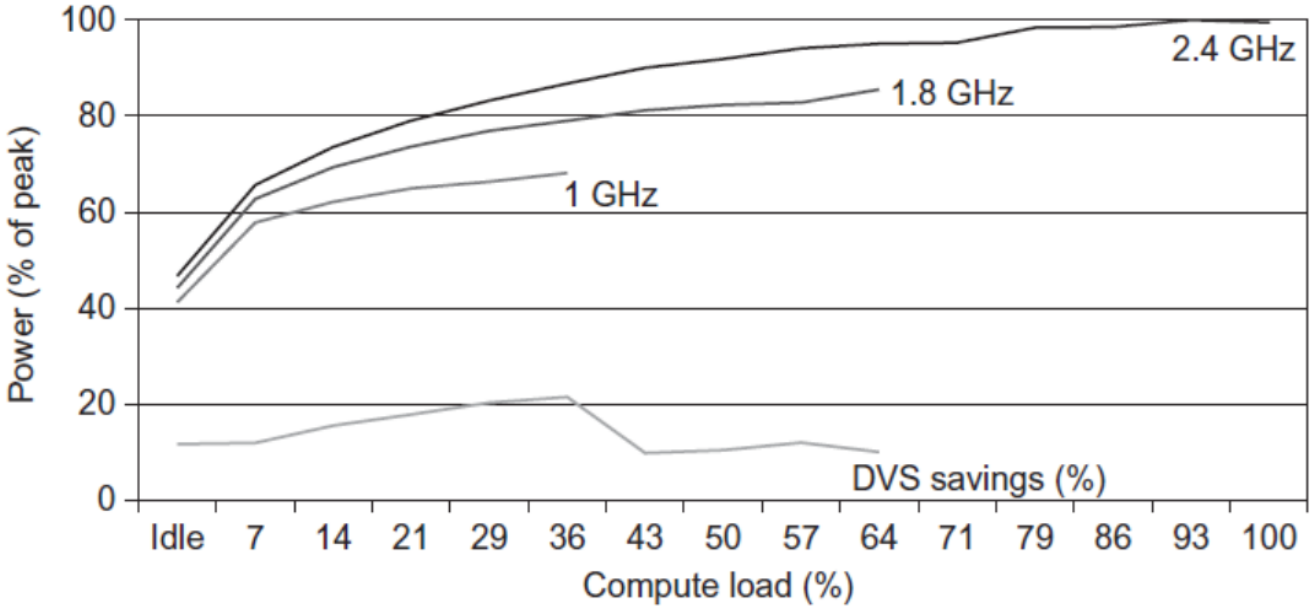
* Why are data servers built close to rivers?



Reducing Power

Techniques for reducing power:

- Do nothing well, *i.e., turn off components when not in use*
- Dynamic Voltage-Frequency Scaling



- Low power state for DRAM, disks
- Overclocking, turning off cores

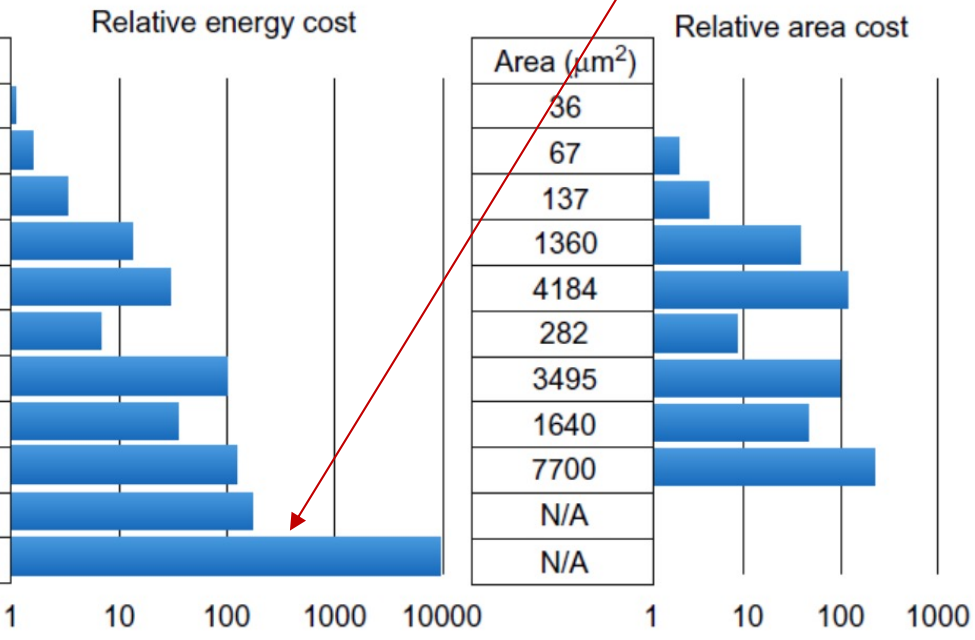
Static Power

Static power consumption

- 25-50% of total power
- $Current_{static}$ x Voltage
- Scales with number of transistors
- To reduce: power gating

Argument for ***in-memory computing***

Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FB Add	0.4	1360
32b FB Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FB Mult	1.1	1640
32b FB Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A



Trends in Cost

Cost driven down by learning curve

- Yield

DRAM: price closely tracks cost

Microprocessors: price depends on volume

- 10% less for each doubling of volume

Integrated Circuit Cost

Integrated circuit

Yield is the keyword!

$$Cost\ of\ integrated\ circuit = \frac{Cost\ of\ die + Cost\ of\ testing\ die + Cost\ of\ packaging\ and\ final\ test}{Final\ test\ yield}$$

$$Cost\ of\ die = \frac{Cost\ of\ wafer}{Dies\ per\ wafer\ x\ Die\ yield}$$

Note: wafers are round dies are rectangular

Bose-Einstein formula:

$$Dies\ per\ wafer = \frac{\pi \times (Wafer\ diameter/2)^2}{Die\ area} - \frac{\pi \times Wafer\ diameter}{\sqrt{2 \times Die\ area}}$$

Defects per unit area = 0.016-0.057 defects per square cm (2010)

N = process-complexity factor = 11.5-15.5 (40 nm, 2010)

$$Die\ yield = Wafer\ yield \times 1/(1 + Defects\ per\ unit\ area \times Die\ area)^N$$

Dependability

Module reliability

- Mean time to failure (MTTF)
- Mean time to repair (MTTR)
- Mean time between failures (MTBF) = MTTF + MTTR
- Availability = MTTF / MTBF

Dependability

- ICs used to be reliable
- Has changed when we move to 65 nm and smaller feature sizes
- A measure of reliability:

  Mean Time To Failure: MTTF = 1/FIT

  Failures In Time: FIT = failures/1 billion hours of operation

  MTTF of 10^6 hours equals $10^9/10^6$ = 1000 FIT
- Mean time to repair: MTTR (a measure for service interruption)
- Mean time between failures: MTBF = MTTF+MTTR
- Module availability = MTTF/(MTTF+MTTR) (a measure of the service accomplishment)

Dependability (no aging effects)

Exercise

- 10 disks, 1.000.000 hours MTTF
- 1 SCSI controller, 500.000 hour MTTF
- 1 power supply, 200.000 hour MTTF
- 1 fan 200.000 hour MTTF
- 1 SCSI cable, 1.000.000 hour MTTF

What is FIT?

Failure rate=10 x 1/1.000.000 + 1/500.000+1/200.000 + 1/200.000 + 1/1.000.000

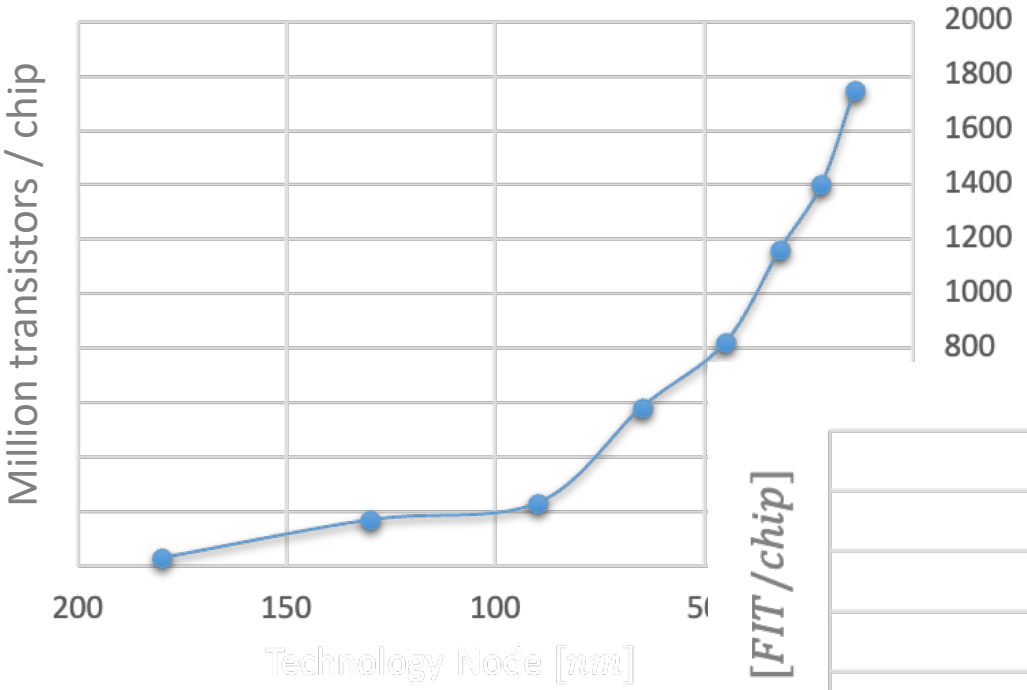
=(10+2+5+5+1)/1.000.000

=23/1.000.000

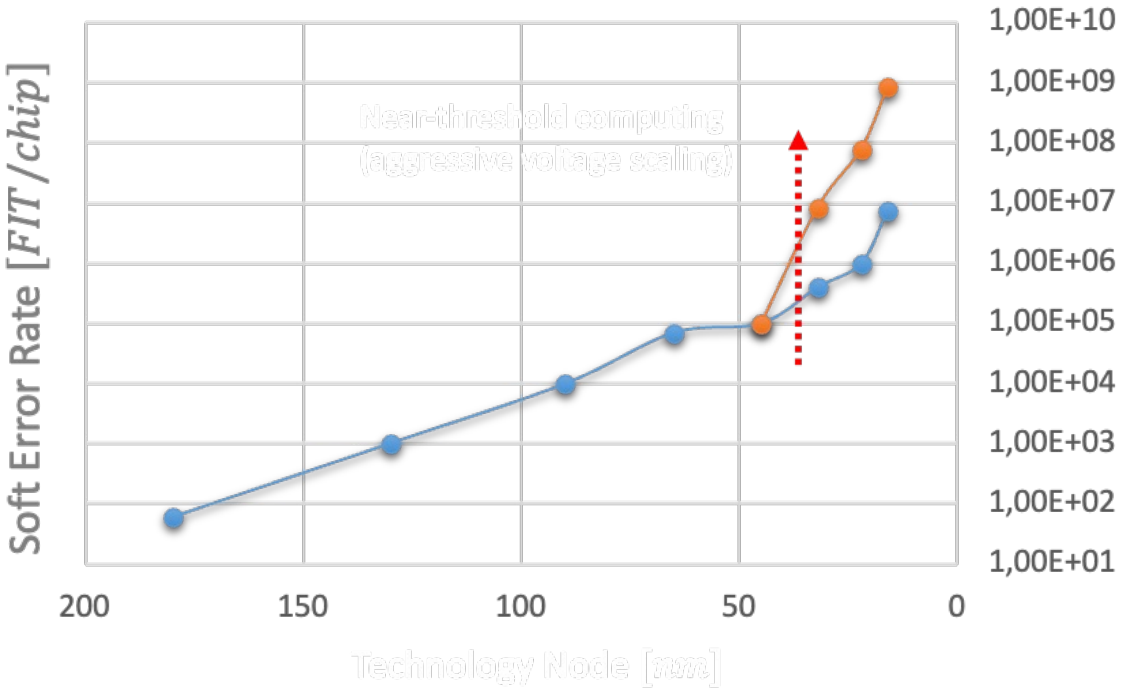
=23.000/1.000.000.000

➔23.000 FIT

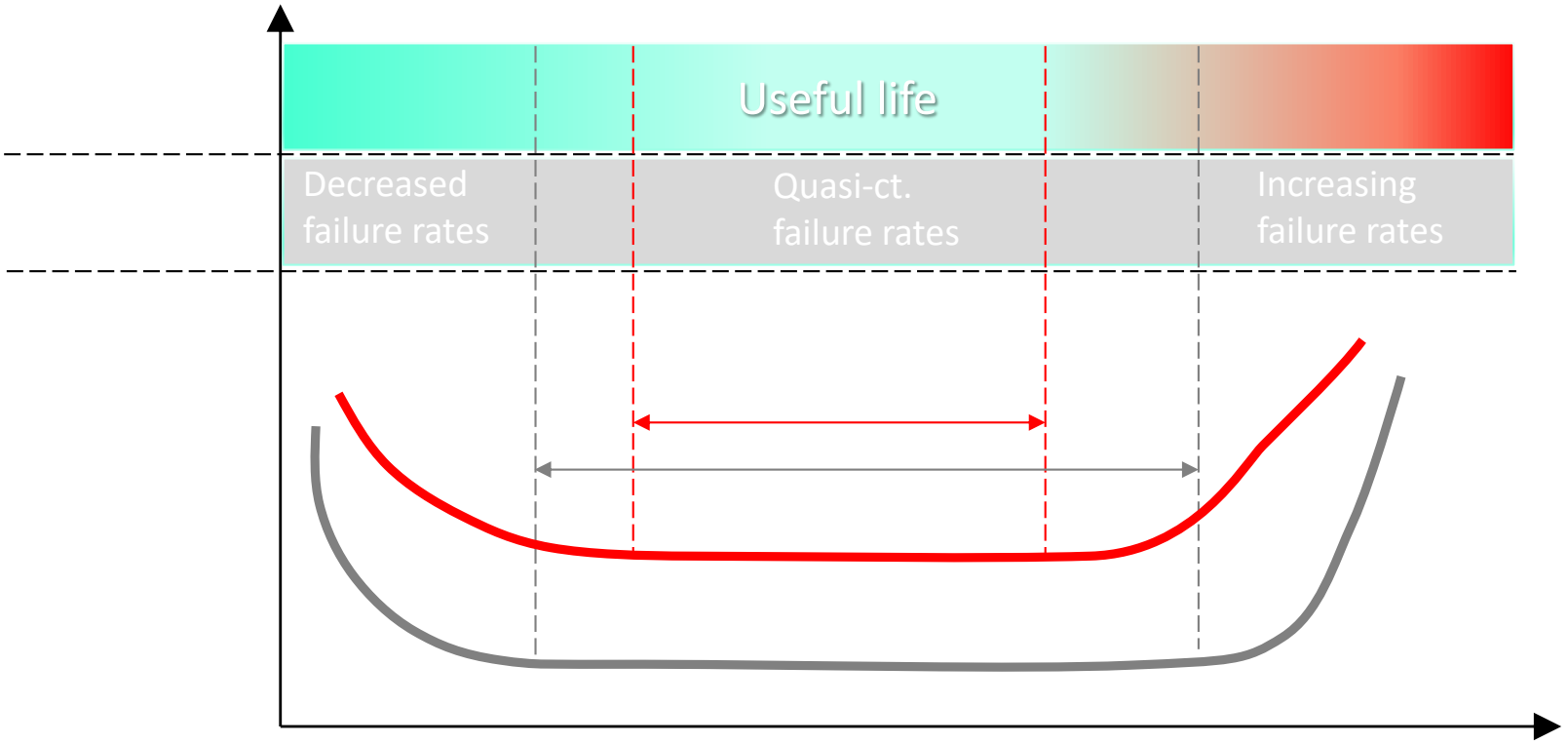
CMOS Scaling – Soft Error Rate

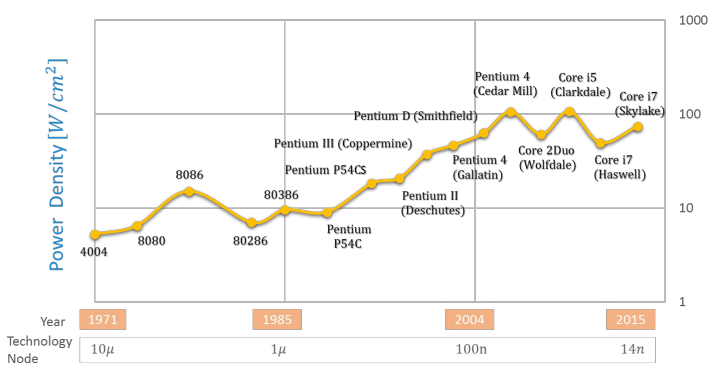


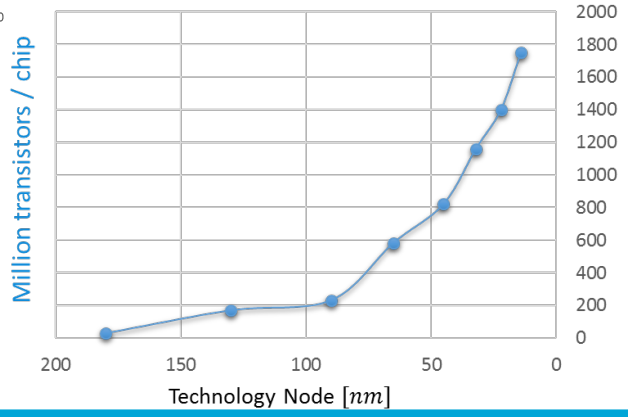
Situation worsens when applying near-threshold techniques

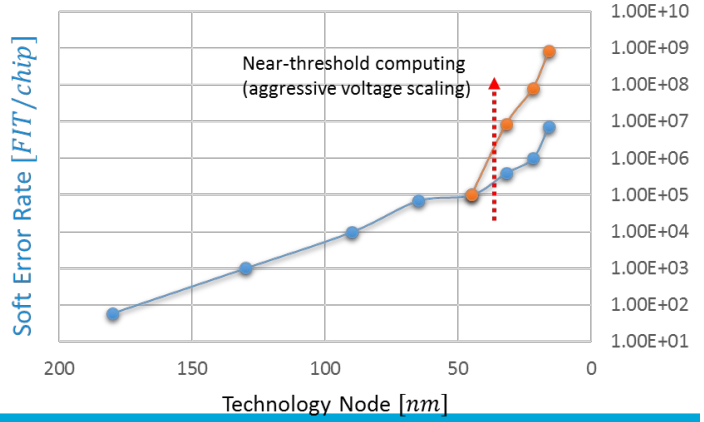


Reliability & Life Time









Performance

- "X is n times faster than Y" means

$$\frac{\text{ExecutionTime(Y)}}{\text{ExecutionTime(X)}} = n$$

- Performance can be viewed as the reciprocal of execution time. Hence

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

- *Increasing* performance means *decreasing* execution time

Measuring Performance

Typical performance metrics:

- Response time
- Throughput

Speedup of X relative to Y

- Execution time$_Y$ / Execution time$_X$

Execution time

- Wall clock time: includes all system overheads
- CPU time: only computation time

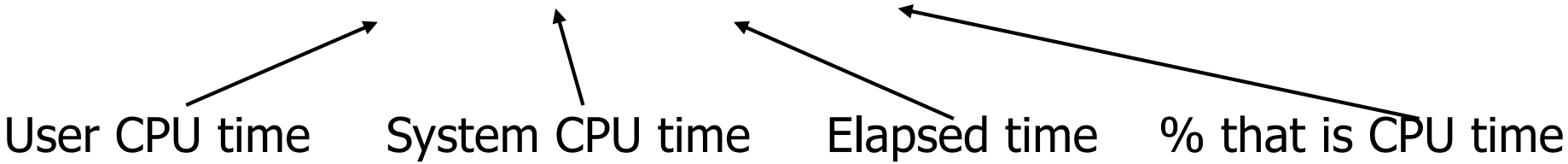
Benchmarks

- Kernels (e.g. matrix multiply)
- Toy programs (e.g. sorting)
- Synthetic benchmarks (e.g. Dhrystone)
- Benchmark suites (e.g. SPEC06fp, TPC-C)

Performance = time, time, time . . .

- Wall-clock time = response time = elapsed time
  - The latency to complete a task
  - Includes everything: disk accesses, I/O, OS overhead, etc.
- CPU time = the time CPU is computing, not the time waiting for I/O or running other programs
- User CPU time = CPU time spent in program (user mode)
- System CPU time = time spent in the OS (kernel mode)
- Unix `time` command output

```
90.7u 12.9s 2:39 65%
```



Benchmarks

- Benchmark: a program used to evaluate performance

- If you buy a new car, most people go for a test drive
  - Which road?
  - Which weather (conditions)?
  - Which load (# of persons)?
  - Which gear?
  - **—> how to approach (every day) reality?**

- This is less common for PCs. Most people do not measure and compare the execution times of their workloads

Benchmarks

- Five types of programs used to evaluate performance (in order of accuracy):
  - Real applications
    - Often encounter portability problems
    - Often some important activity has to be eliminated
  - Modified (or scripted) applications
    - To enhance portability
    - To focus on a particular aspect of system performance (e.g., to create a CPU-oriented benchmarks, I/O may be removed or restructured)
    - Scripts are used to reproduce interactive behavior

Benchmarks

- Kernels
  - Small, key pieces from real programs
  - Livermore loops and Linpack
  - Kernels are best used to isolate performance of individual features
- Toy benchmarks
  - Quicksort, Sieve of Eratosthenes, …
  - Small, easy to understand, but not useful to evaluate performance
- Synthetic benchmarks
  - "Typical" instruction mixes
  - Whetstone and Dhrystone

Benchmark Suites

- Benchmark suite = a collection a benchmarks that measure the performance with a variety of applications
- Most successful attempt to create a standard: SPEC (Standard Performance Evaluation Cooperative)

- Some Windows benchmark suites:
  - Business Winstone: Netscape and several office products
  - CC Winstone: Photoshop, Premiere, and various audio-editing programs
  - Winbench: CPU performance, video performance,disk performance

- Emergence of multi-core processors also led to multi-core benchmarks

- Note: benchmarks follow trends – scientific, video, embedded, 3D, IO, web, signal processing, and (recently) AI, …

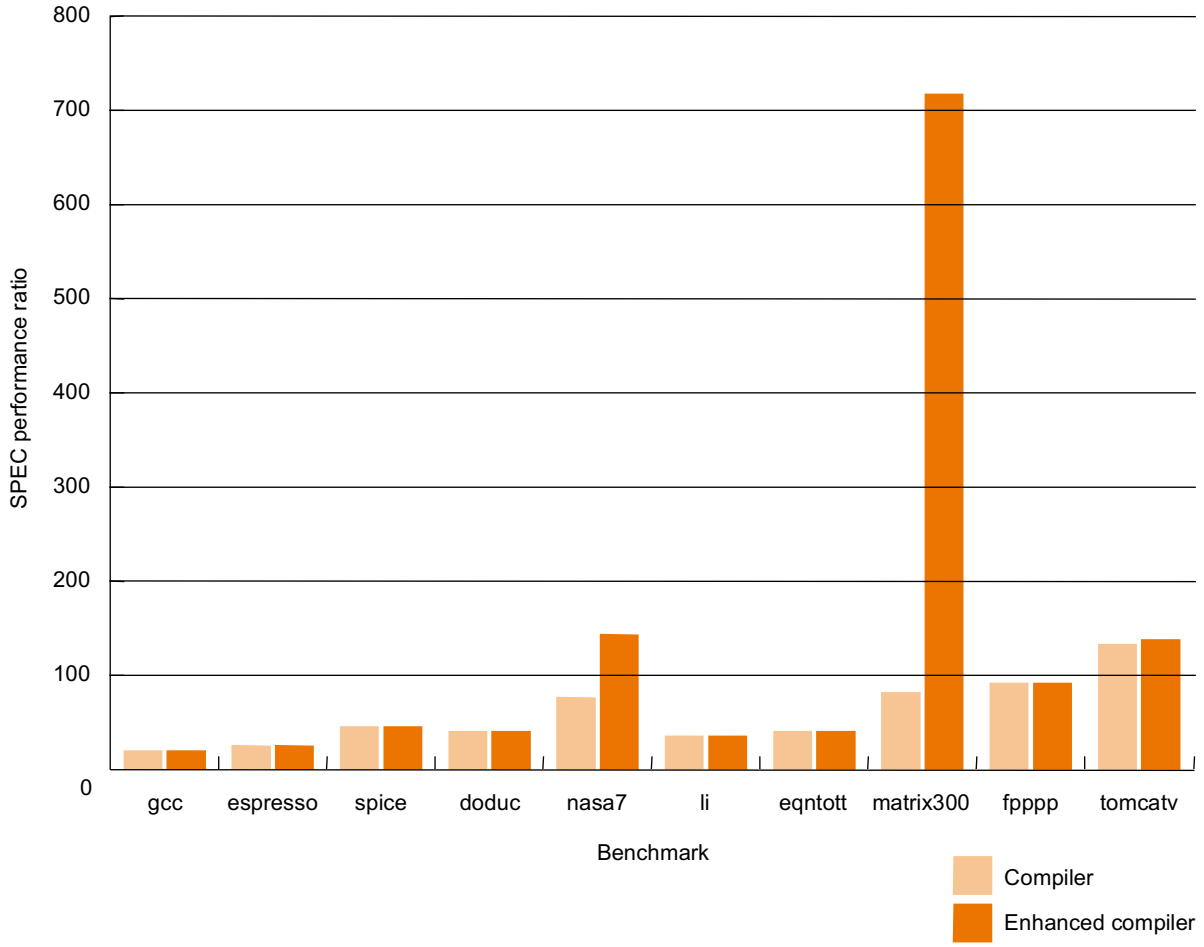
SPEC CPU Benchmarks History

- SPEC89
  - 10 C and Fortran programs yielding a single number ("SPECmark")
- SPEC92
  - 6 integer programs (SPECint92) and 14 fp programs (SPECfp92)
  - Compiler flags unlimited
- SPEC95
  - New set of programs, 8 int and 10 fp
  - Single flag setting for all programs
- SPEC CPU2000
  - 11 int programs (CINT2000) and 14 fp programs (CFP2000)
- SPEC CPU 2006
  - 12 int programs (CINT2006) and 17 fp programs (CFP2006)
  - C++ programs too

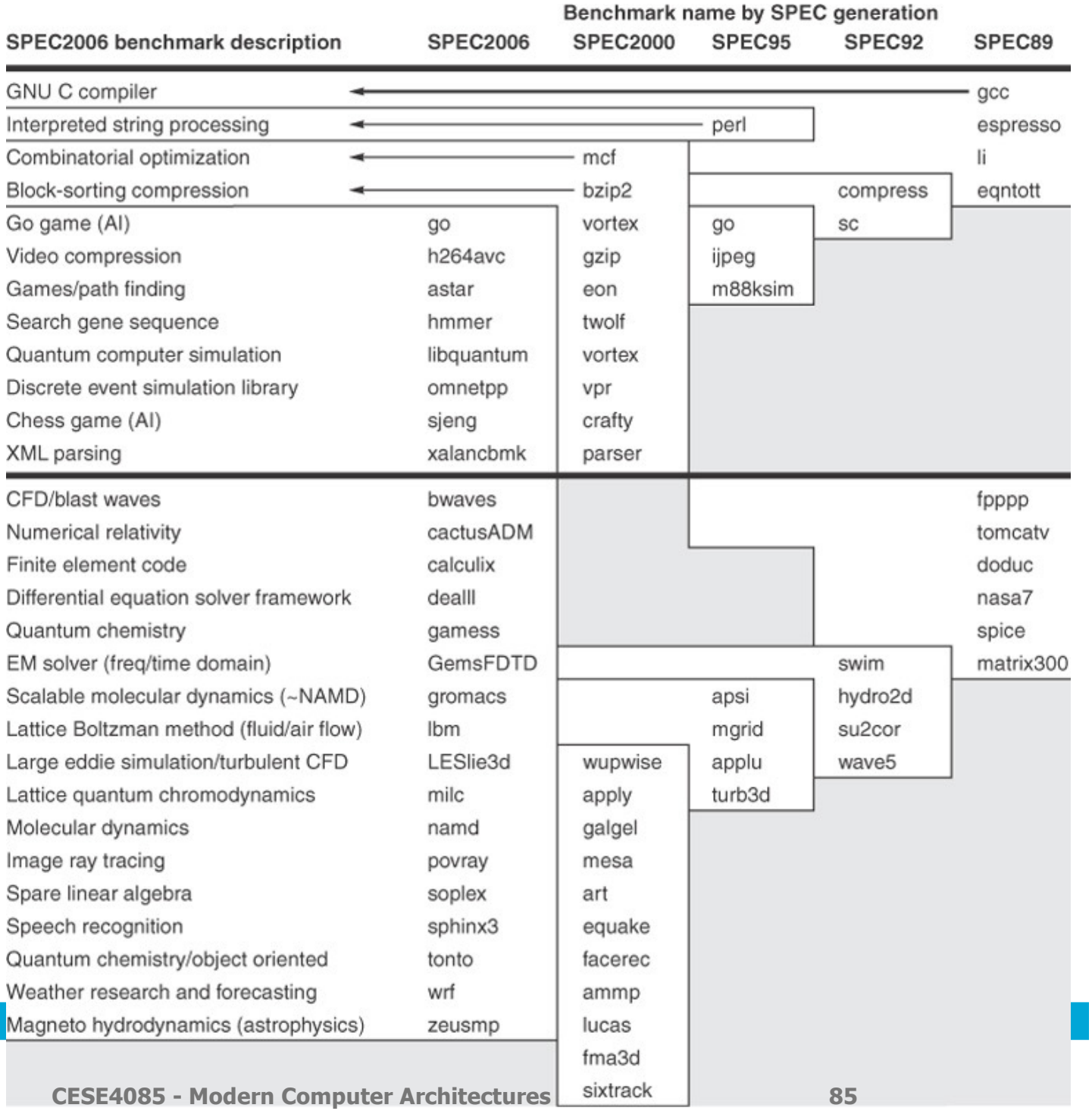
www.spec.org

SPEC '89

- Compiler "enhancements" ☺! and performance



SPEC CPU Benchmarks



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			

Some Other SPEC Benchmarks

- SPEC CPU 2006 is aimed at CPU performance
  - Real programs, modified for portability and to minimize role of I/O
- Graphics benchmarks:
  - SPECviewperf: 3D rendering performance under OpenGL
  - SPECapc: applications that make extensive use of graphics
- Server applications have significant I/O activity
  - SPECSFS: file server benchmark
  - SPECWeb: web server benchmark

Embedded Benchmarks

- Less developed than desktop and server benchmarks
- Most designers devise benchmarks that reflect their application
- One attempt: EDN Embedded Microprocessor Benchmark Consortium (EEMBC, pronounced *embassy*)
- Kernels and benchmarks that fall into 5 classes
  - Automotive/industrial
  - Consumer
  - Networking
  - Office automation
  - Telecommunications

Summarizing Performance

- If there is only one program, then it is clear which computer is faster
- When there is a collection of programs, things become tricky – how do you aggregate their performance?
- Example:

	Computer A	Computer B	Computer C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20
Total (secs)	1001	110	40

Summarizing Performance

- Total execution time is a consistent summary measure
  - B is 9.1 times faster than A for programs P1 and P2
  - C is 25 times faster than A for programs P1 and P2
- We might as well use *arithmetic mean*

$$\frac{1}{n}\sum_{i=1}^{n} Time_i$$

  where $Time_i$ is the execution time of program P_i
- What if the programs are not run equally as often?
- Two approaches:
  - Weighted execution time
  - Normalize execution times to a reference machine and take average

Weighted Execution Time

- Weighted execution time:
  - Each program is assigned weighting factor to indicate its relative frequency
  - Weighting factors add up to 1
- Weighted arithmetic mean: $\sum_{i=1}^{n} Weight_i \times Time_i$
- Suppose weight of P1 is 0.8 and weight of P2 0.2. Then

	Computer A	Computer B	Computer C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20
Arithmetic mean	500.5	55	20
Weighted mean	200.8	28	20
		200.8/28=7.16	200.8/20=10.04

Arithmetic mean vs. Geometric mean

Arithmetic Mean (AM)	Geometric Mean (GM)
1. It is calculated by dividing the sum of data values by the number of data values.	1. It is calculated by raising the product of data values by reciprocal of the number of data values.
2. For a given set of data values $x_1, x_2, x_3, \ldots, x_n$, the arithmetic mean = $(x_1 + x_2 + x_3 + \ldots + x_n) / n$.	2. For a given set of data values $x_1, x_2, x_3, \ldots, x_n$, the geometric mean = $(x_1 \cdot x_2 \cdot x_3 \cdot \ldots \cdot x_n)^{1/n}$.
3. AM ≥ GM always for any set of data values.	3. GM ≤ AM always for any set of data values.
4. It applies to both positive and negative values.	4. It applies only to positive values.
5. It is affected by outliers.	5. It is not much affected by outliers.
6. It is easy to use.	6. It is harder to use compared to AM.
7. It gives a good and accurate approximation when there is not much variation in the data.	7. It gives a good and accurate approximation when there is much variation in the data.
8. It is mostly used in the fields of maths and statistics.	9. It is used mostly in the field of finance.

Taken from:
https://www.cuemath.com/data/arithmetic-mean-vs-geometric-mean/

Normalized Execution Time

- Normalize execution times to a reference machine and take average
- Used by SPEC
- But ***don't use the arithmetic but rather geometric mean to average normalized execution time*** - it can lead to different results depending on the reference machine!

	Normalized to A			Normalized to B		
	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2
Arithmetic mean	1.0	5.05	10.01	5.05	1.0	1.1

- lower is better

Geometric Mean

Use geometric mean instead: (no physical significance!)

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

Where ExecutionTimeRatio$_i$ is the execution time of program P_i normalized to its execution time on the reference machine

	Normalized to A			Normalized to B		
	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2
Geometric mean	1.0	1.0	0.63	1.0	1.0	0.63

* Lower is better

Principles of Computer Design

Take Advantage of Parallelism

e.g. multiple processors, disks, memory banks, pipelining, multiple functional units

Principle of Locality

Reuse of data and instructions

Focus on the Common Case

Amdahl's Law

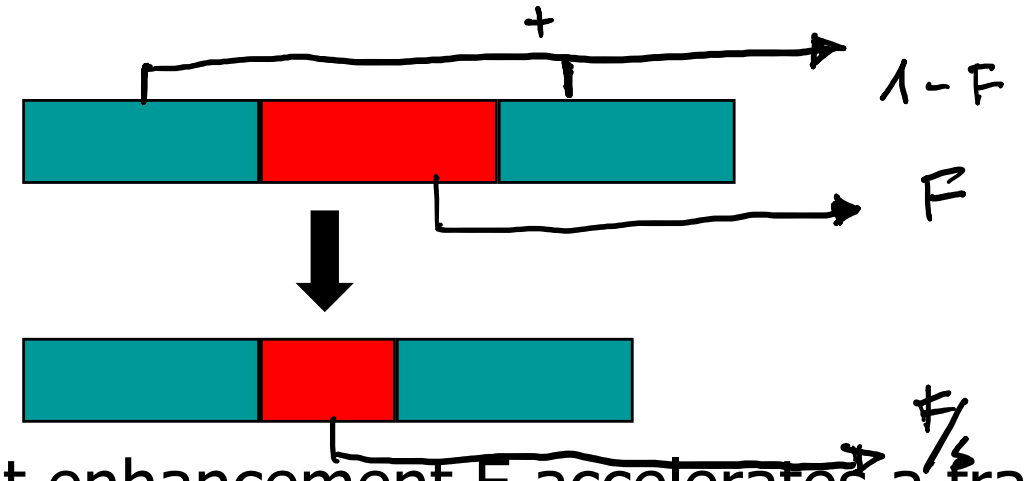
$$Execution\ time_{new} = Execution\ time_{old} \times \left((1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right)$$

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Amdahl's Law

Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected

Amdahl's Law

$$T_{new} = (1 - F)T_{old} + \frac{F}{S}T_{old} =$$

$$= T_{old}\left(1 - F + \frac{F}{S}\right)$$

$$S_{overal} = \frac{T_{old}}{T_{new}} = \frac{1}{1 - F + \frac{F}{S}}$$

F=fraction enchanced

S=speedup after enhancement

Amdahl's Law - Example

- Floating point instructions improved to run 2X; but only 10% of actual instructions are FP

$$\mathbf{T_{new} = T_{old} \times (0.9 + 0.1/2) = 0.95 \times T_{old}}$$

$$\mathbf{S_{overall} = \frac{1}{0.95} = 1.053}$$

- Side note: Be aware of papers reporting, e.g., > 1000 times kernel improvements ☺:

$$T_{new} = T_{old} \times (0.9 + 0.1/1000) = 0.9001 \times T_{old}$$

$$S_{overall} = 1/0.9001 = 1.111$$

Amdahl's Law

- Amdahl's law expresses an important principle of computer design:

Make the Common Case Fast

- It also expresses the *law of diminishing returns*
  - Additional enhancements lead to incremental improvements

- Examples:
  - Assume "0.5 of program" (F) is optimized -> max. speedup = 2
  - Assume "0.75 of program" (F) is optimized -> max. speedup = 4
  - Assume "0.9 of program" (F) is optimized -> max. speedup = 10
  - Assumed speedup (S) of "0.x of program" (F) is "infinite" -> S becomes infinite -> S(overall) = 1/1-F

Principles of Computer Design

The Processor Performance Equation

$$CPU\ time = CPU\ clock\ cycles\ for\ a\ program \times Clock\ cycle\ time$$

$$CPU\ time = \frac{CPU\ clock\ cycles\ for\ a\ program}{Clock\ rate}$$

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

$$CPU\ time = Instruction\ count \times Cycles\ per\ instruction \times Clock\ cycle\ time$$

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

Principles of Computer Design

Different instruction types having different CPIs

$$CPU\ clock\ cycles = \sum_{i=1}^{n} IC_i \times CPI_i$$

$$CPU\ time = \left(\sum_{i=1}^{n} IC_i \times CPI_i \right) \times Clock\ cycle\ time$$

Principles of Computer Design

Different instruction types having different CPIs

$$CPU\ clock\ cycles = \sum_{i=1}^{n} IC_i \times CPI_i$$

$$CPU\ time = \left(\sum_{i=1}^{n} IC_i \times CPI_i \right) \times Clock\ cycle\ time$$

CPU Performance Equation

- CPU time = Instruction_count x Cycle_time x CPI
  = Instruction_count x CPI / f

  where

  Instruction_count = number of instructions executed

  Cycle_time = reciprocal of frequency f (e.g. 1 GHz: 1 ns)

  CPI = cycles per instruction = Cycle_count/Instruction_count
- CPU performance depends on these three characteristics:
  - Cycle time (or clock frequency)
  - Cycles per instruction
  - Instruction count
- It is **equally dependent on these three**
  - 10% improvement in any of them leads to 10% overall improvement

CPU Performance

- Where can we improve what?

	Instruction count	CPI	Cycle time
Program	X	(X)	
Compiler	X	(X)	
Instruction set	X	X	(X)
Organization		X	X
Technology			X

Instruction Frequency

- Sometimes we know how many cycles each instruction takes and the frequency of each instruction
- Then average CPI can be calculated as

$$CPI = \sum_{i} CPI_i \times Freq_i$$

where

CPI_i is the average CPI for instruction i

$Freq_i = IC_i/IC$ is the frequency of instruction i

Example: Calculating CPI

Op	Freq	Cycles	CPI_i x $Freq_i$	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			CPI = 1.5	

Typical Mix

Cycles Per Instruction

- Some computer architecture researchers sometimes only report CPI (or IPC) because they assume instruction count and cycle time are unaffected
  - Needs to be substantiated
- In this case (only!) speedup can be calculated as follows:

$$\text{Speedup} = \frac{CPU_{old}}{CPU_{new}} = \frac{IC * clockcycle * CPI_{old}}{IC * clockcycle * CPI_{new}} = \frac{CPI_{old}}{CPI_{new}}$$

Fallacies and Pitfalls

All exponential laws must come to an end

- Dennard scaling (constant power density)
  - Stopped by threshold voltage
- Disk capacity
  - 30-100% per year to 5% per year
- Moore's Law
  - Most visible with DRAM capacity
  - ITRS disbanded
  - Only four foundries left producing state-of-the-art logic chips
  - 11 nm, 3 nm might be the limit

Fallacies and Pitfalls

Microprocessors are a silver bullet

- Performance is now a programmer's burden

Falling prey to Amdahl's Law

A single point of failure

Hardware enhancements that increase performance also improve energy efficiency, or are at worst energy neutral

Benchmarks remain valid indefinitely

- Compiler optimizations target benchmarks

Fallacies and Pitfalls

The rated mean time to failure of disks is 1,200,000 hours or almost 140 years, so disks practically never fail

- MTTF value from manufacturers assume regular replacement

Peak performance tracks observed performance

Fault detection can lower availability

- Not all operations are needed for correct execution

Three Design Principles

- Take advantage of parallelism
  - Pipelining, Instruction-Level Parallelism (ILP), Data-Level Parallelism (DLP), Thread-Level Parallelism (TLP)
- Principle of locality
  - Rule of thumb: programs spend 90% of their execution time in 10% of their code
  - We can predict with reasonable accuracy what instructions and data a program will use in the near future based on its recent past
- Make the common case fast
  - Amdahl's law

Summary #1

- **Trends**

	Capacity	Speed
Logic	2x in 3 years	2x in 3 years
DRAM	4x in 3 years	2x in 10 years
Disk	4x in 3 years	2x in 10 years

- **Time to run the task**
  - Execution time, response time, latency
- **Tasks per day, hour, week, sec, ns, …**
  - Throughput, bandwidth
- **"X is n times faster than Y" means**

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

Summary #2

- Amdahl's Law: $$S_{overal} = \frac{T_{old}}{T_{new}} = \frac{1}{1 - F + \frac{F}{S}}$$
- CPU performance equation:

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

- Execution time is the REAL measure of computer performance!
- Good products created when have:
  - Good benchmarks, good ways to summarize performance
- Next time: Dynamic Exploitation of ILP

END of NEW CH1