CS2410
Computer Architecture

Bruce Childers
childers@cs.pitt.edu

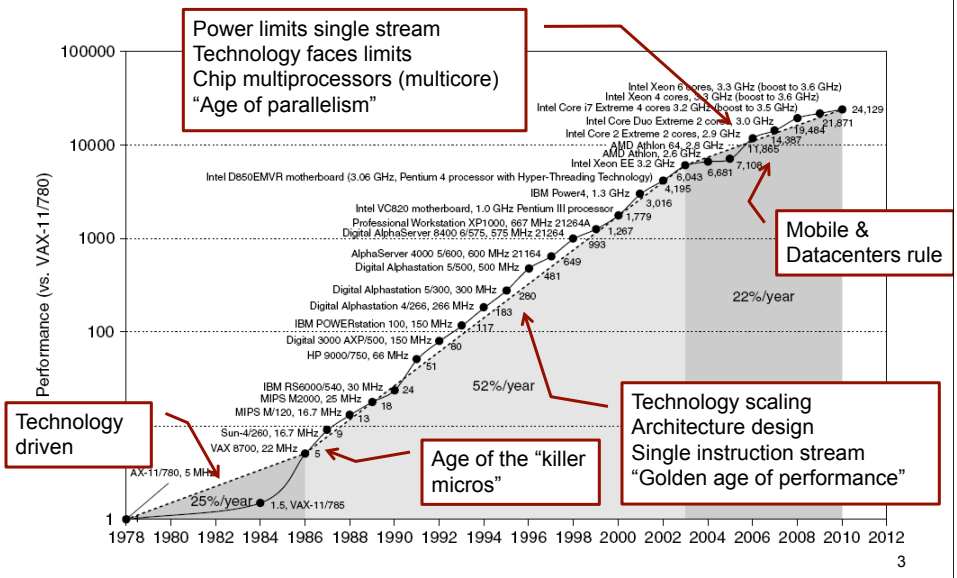
Dept. of Computer Science
University of Pittsburgh

http://www.cs.pitt.edu/~childers/CS2410

Administrative Issues

- **Book:** Computer Architecture – A Quantitative Approach, 5th ed., H&P
  - Other content added to "freshen up" course; extra reading material
  - E.g., Experimental methods, Near data computation, Accelerators
- **Requirements & Grading:**
  - **Mid-term exam:** 25%, approximately, Oct 10 – 18
  - **Final exam:** 30%, Dec 14
  - **Projects:** 30%, expected 2x: ISA, OOO
  - **Participation:** 15% (questions, discussion, contribution)
- **Schedule:**
  - **Out of town:** tentatively, Nov 28, 30, Dec 5
  - Schedule 1-2x make up classes
  - Possibile? Wednesday, Nov 23 (student holiday)
  - Saturday? Evening?

Growth of Processor performance



Huge Impact!

① **Computing capability available to end user**
- Compare your mobile to your 10-yr old desktop

② **Entirely new classes of computers**
- Mainframe=>Mini=>Desktop=>Portable=>Mobile=>Augmented

③ **Computers are all microprocessor based today**
- Age of killer micros: Custom design to commodity processors

④ **Increased performance (obviously?)**
- Better languages=>Increased productivity & robustness
- New applications
- New discoveries!

Classes of Computers (§1.2)

- 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
- Embedded Computers
  - Emphasis: price, energy, application-specific performance

Parallelism

- **Today: Performance is primarily about doing multiple "things" at the same time**
- Applications ***exhibit*** parallelism
  - ① Data-level parallelism (DLP): Many data items operated on at the same time
  - ② Task-level parallelism (TLP): Separate units of work that can operate independently and at the same time

Parallelism

- **Today: Performance is primarily about doing multiple "things" at the same time**

- System architectures ***exploit*** parallelism
  - ① Instruction-level parallelism (ILP): Data-level parallelism exploited in the same instruction stream
  - ② Vector/GPU architectures: Data-level parallelism exploited by single instruction applied on multiple data items
  - ③ Thread-level parallelism (TLP): Data or task-level parallelism exploited by separate instruction streams that may interact
  - ④ Request-level parallelism (RLP): Largely independent tasks for separate request streams (users)

Flynn's Taxonomy

classic von Neumann

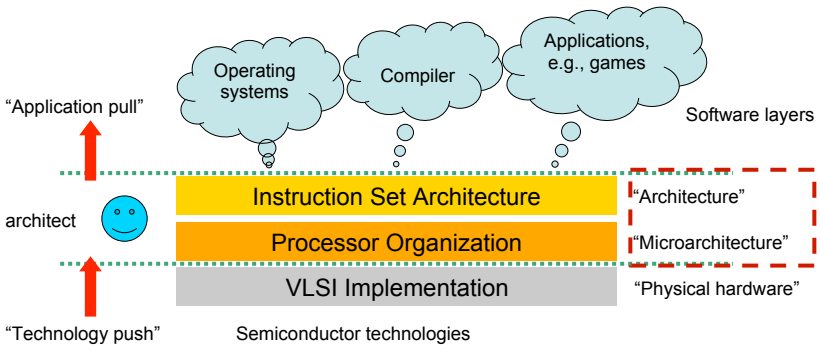
SISD **Single instruction stream** **Single data stream**	**(SIMD)** **Single instruction stream** **Multiple data stream**
MISD **Multiple instruction stream** **Single data stream**	**(MIMD)** **Multiple instruction stream** **Multiple data stream**

Does not make much sense

What is Computer Architecture? (§1.3)

- **1950s to 1960s:**
  Computer Architecture Course = Computer Arithmetic
- **1970s and 1980s:**
  Computer Architecture Course = Instruction Set Design, especially ISA appropriate for compilers
- **1990s:**
  Computer Architecture Course = Design of CPU, memory system, I/O system, Multiprocessors
- **2000's:**
  Embedded computing, server farms, power management issues, network processors.
- **2012:**
  All of the above + multicores + GPUs + cloud computing

What is Computer Architecture?



Models to overcome Instruction Level Parallelism (ILP) limitations:

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

Trends in Technology(§1.4)

- Integrated circuit technology
  - Transistor density: +35%/year (feature size decreases)
  - Die size: +10-20%/year
  - Integration overall: +40-55%/year
- DRAM capacity: +25-40%/year (growth is slowing)
  (memory usage doubles every year)
- Flash capacity: +50-60%/year
  - 15-20X cheaper/bit than DRAM
- Magnetic disk technology: +40%/year
  - 15-25X cheaper/bit then Flash and 300-500X cheaper/bit than DRAM
- Clock rate stopped increasing but supply voltage is decreasing

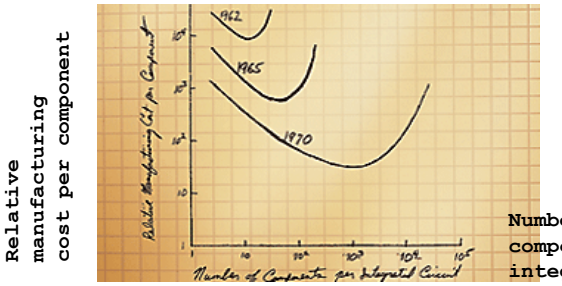
Bandwidth and Latency

Latency lags bandwidth in the last 30 years

- Bandwidth or throughput
  - Total work done in a given time
  - 10,000-25,000X improvement for processors
  - 300-1200X improvement for memory and disks
- Latency or response time
  - Time between start and completion of an event
  - 30-80X improvement for processors
  - 6-8X improvement for memory and disks

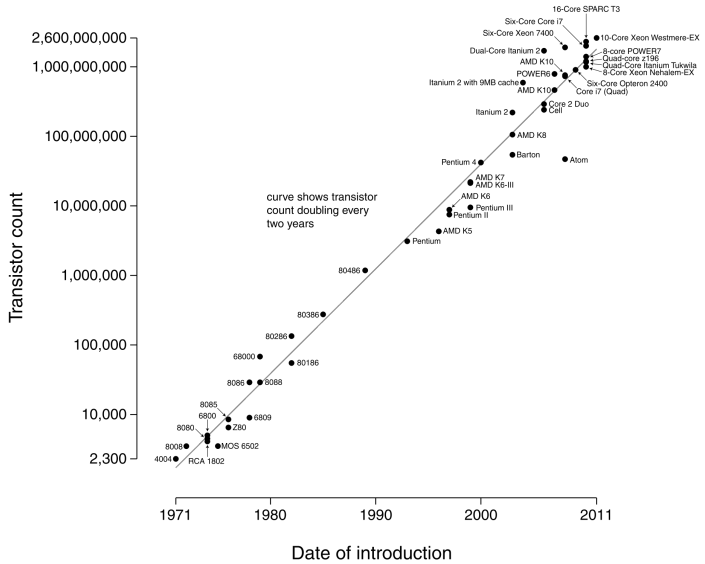
Transistors and wires

- Feature size
  - Minimum size of transistor or wire in x or y dimension
  - 10 microns in 1971 to 22 nm in 2012, 10 nm in 2017
  - Transistor performance scales linearly
    - Wire delay does not improve with feature size!
  - Integration density scales quadratically



Moore's note (1965)

Microprocessor Transistor Counts 1971-2011 & Moore's Law



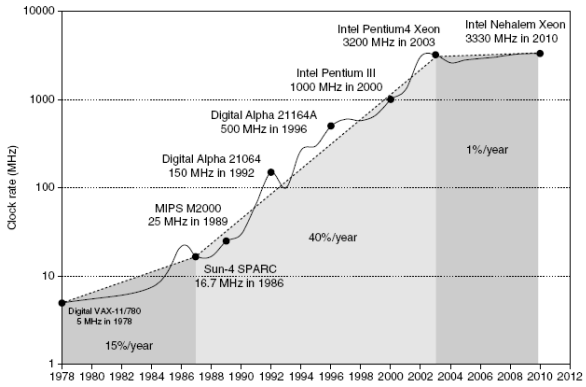
Switches

- Building block for digital logic
  - NAND, NOR, NOT, …
- Technology advances have provided designers with switches that are
  - Faster;
  - Lower power;
  - More reliable (e.g., vacuum tube vs. transistor); and
  - Smaller.
- Nano-scale technologies will not continue promising the same good properties

Power and Energy (§1.5)

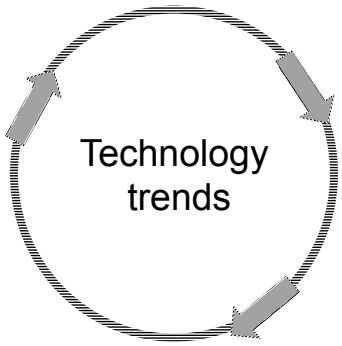
- Need to get power in and out (thermal implications)
- Dynamic energy (to switch transistors)
  - Energy = Power * Time
  - Energy is proportional to Voltage^2
  - Power is proportional to (Voltage^2 x Clock frequency)
- Dynamic Frequency Scaling (reduces power not energy) → voltage scaling
- Static power is proportional to the voltage
  - Static power is “idle” power lost during no computation
- Memory power modes and turning off cores

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
- There is a limit on what can be cooled by air

Computer Engineering Methodology



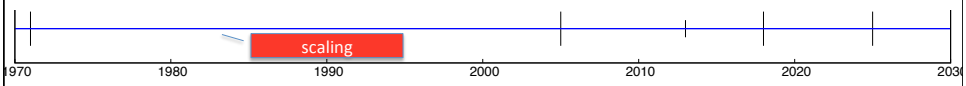
Evaluate Existing Systems for Bottlenecks

Simulate New Designs and Organizations

Implement Next Generation System

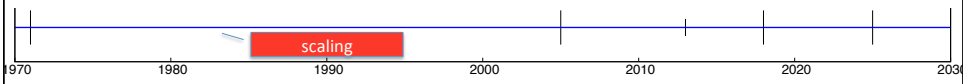
The Trends

The Trends

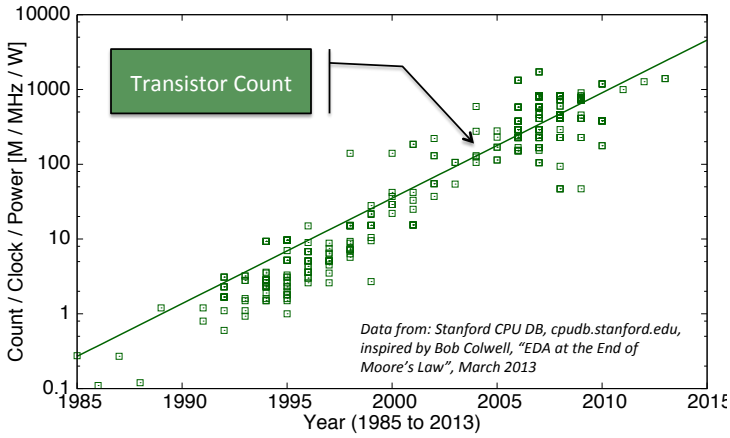


- Moore's Law: Doubling transistors, every two years

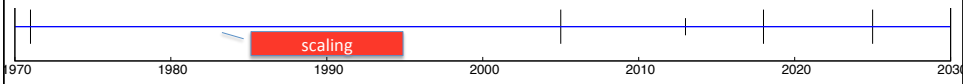
The Trends



- Moore's Law: Doubling transistors, every two years

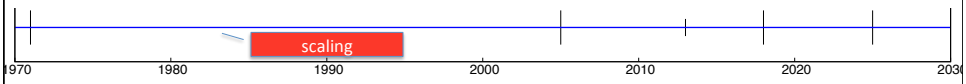


The Trends



- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors

The Trends



- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors

Parameter	Scaling Factor
Dimension (Tox, W, L)	1/k
Doping concentration (Na)	k
Voltage	1/k
Current	1/k
Capacitance	1/k
Delay time	1/k
Transistor power	$1/k^2$
Power density	1

Relies on *voltage scaling*

$P=N\times C\times f\times \mathbf{V^2}$

$=(k^2)\times(1/k)\times(k)\times\mathbf{(1/k^2)}$

$= 1$ *power density constant*

Limited by subthreshold voltage (Vt)

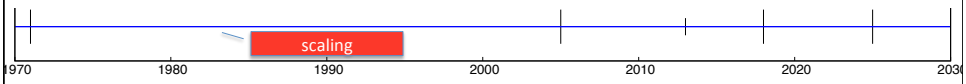
- Vt stopped scaling
- V stopped scaling
- Tox stopped scaling
- Na stopped scaling
- Everything is leakier!

Power density is doubling

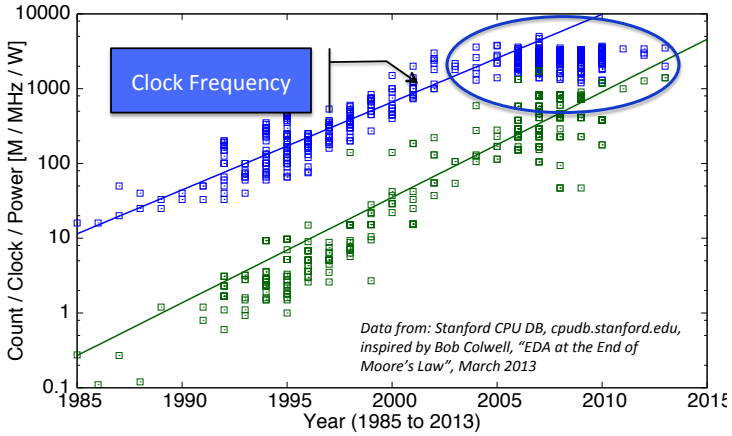
$P=(k^2)\times(1/k)\times(k)\times\mathbf{(1)}$

$= k^2$, k conventionally 1.4, so 2×

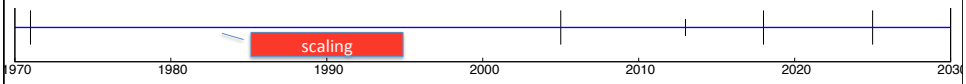
The Trends



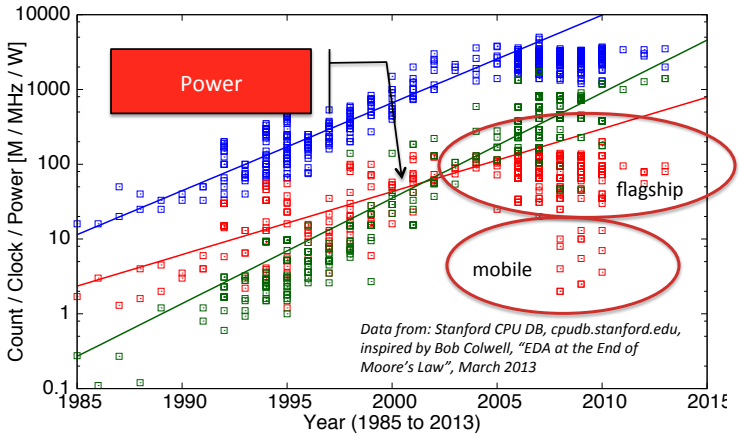
- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors



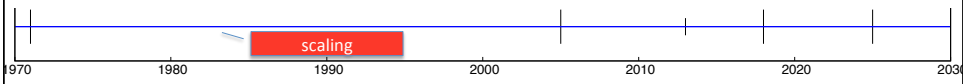
The Trends



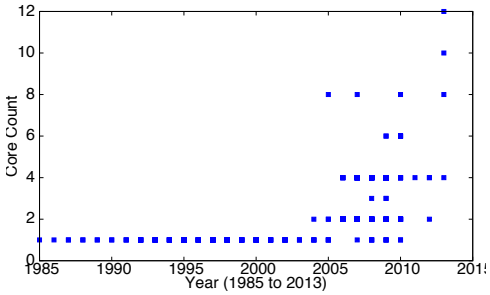
- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors

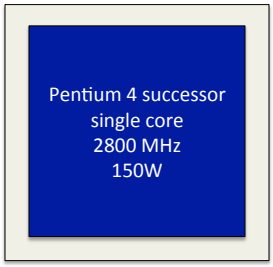


The Trends

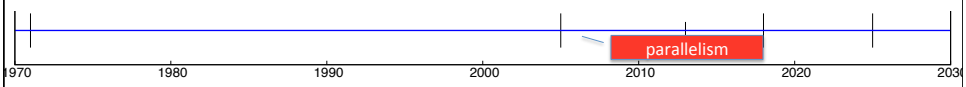


- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors

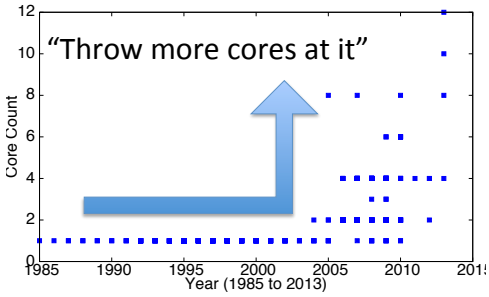


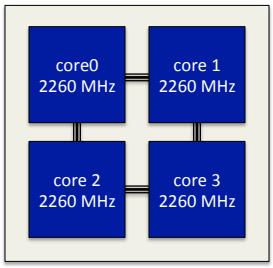


The Trends



- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors





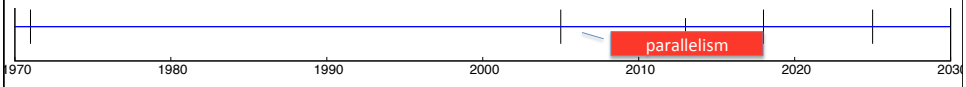
Chip Multiprocessors

Same cores, same ISA

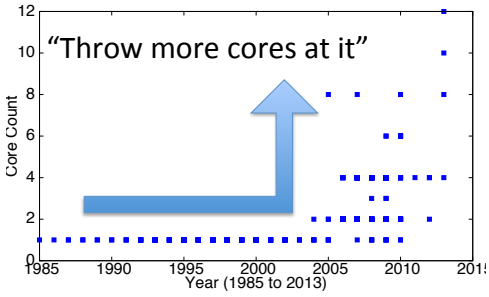
Less focus on ILP w/multiple tasks

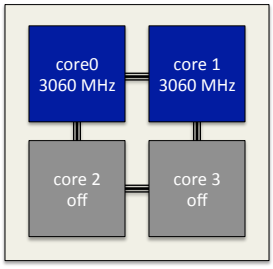
Caching, network on chip

The Trends



- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors



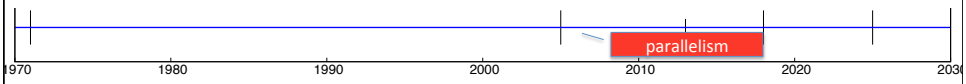


Chip Multiprocessors

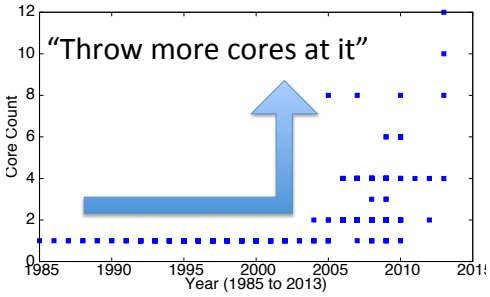
Power limits active cores

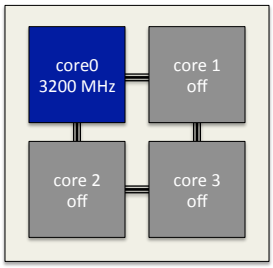
Turbo Boost (DVFS)

The Trends



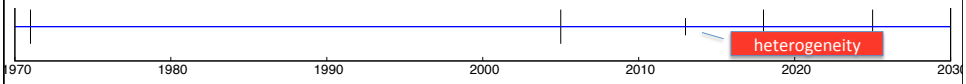
- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors





Chip Multiprocessors
Same cores, same ISA
Less focus on ILP w/multiple tasks
Caching, network on chip

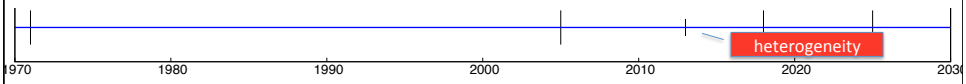
The Trends



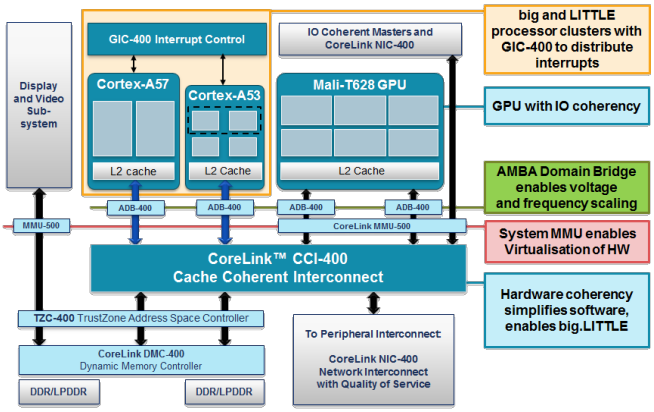
- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors

CMP + Heterogeneous
Partitioned design
Tailored to app task
Sep. ISA/program model
e.g., GPGPU
Asymmetry
Big and little cores
Tailored within task

The Trends



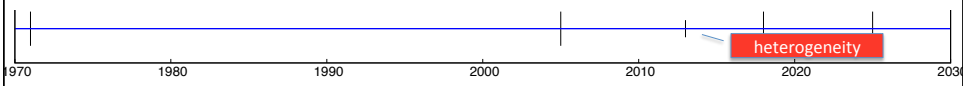
- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors



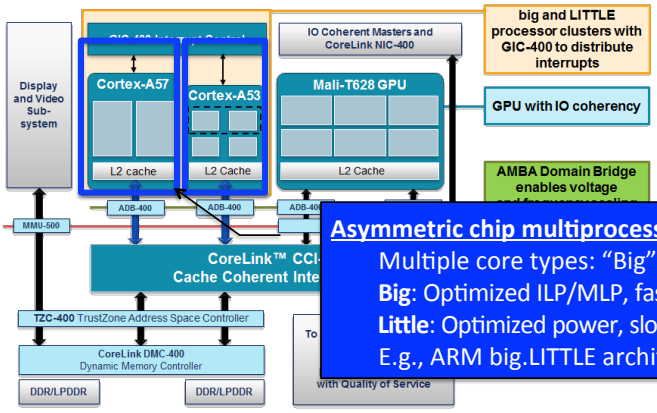
CMP + Heterogeneous
Partitioned design
Tailored to app task
Sep. ISA/program model
e.g., GPGPU

Asymmetry
Big and little cores
Tailored within task

The Trends

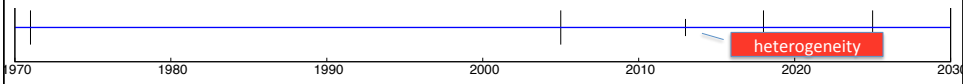


- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors

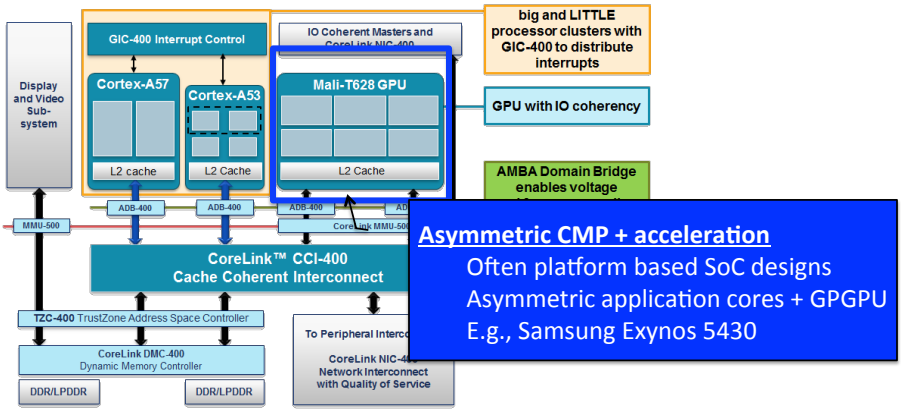


Asymmetric chip multiprocessors
Multiple core types: "Big" and "Little"
Big: Optimized ILP/MLP, fast clock speed
Little: Optimized power, slow clock speed
E.g., ARM big.LITTLE architecture

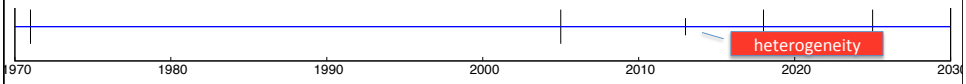
The Trends



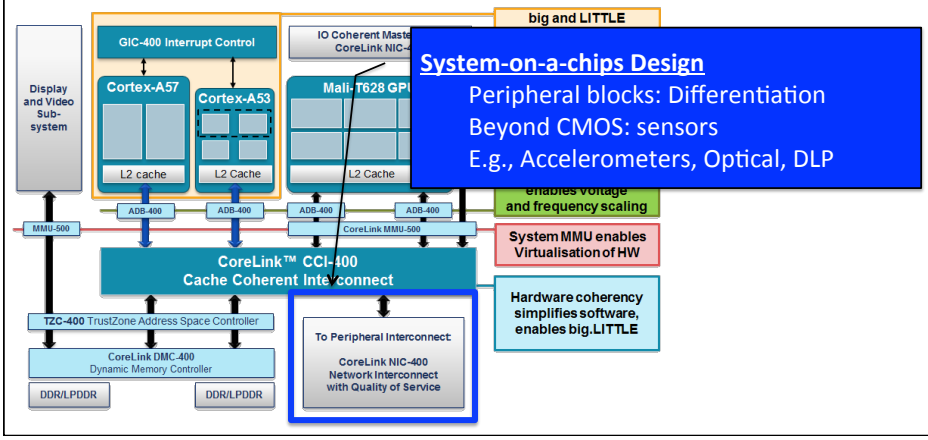
- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors



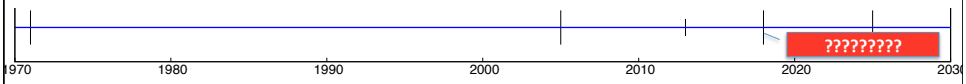
The Trends



- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors



The Trends



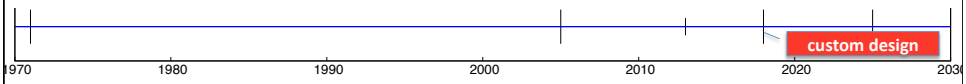
- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors

But now, we're slowly approaching the
End of Moore's Law!

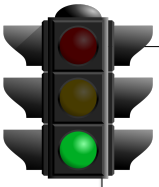
Less and less benefit from scaling
More difficult to shrink (process, materials, etc.)
Economics (many $B fab plants)

E.g., Bob Colwell (architect of P6), predicts end around 7-5nm circa 2020-2022 (HotChips, 2013)

The Trends



- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors

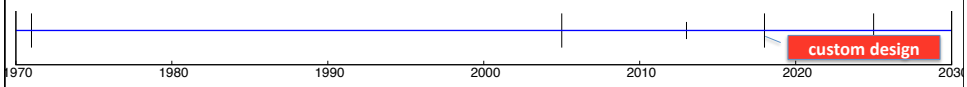


Parallelism + Heterogeneity + *X*
X = customization, close specialization

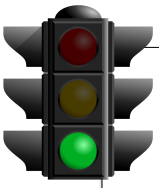
Evidence: SoC in Embedded Systems
E.g., ARM+TMSC+AcmeInc = Platform Customization
Leading edge is often high cost, high volume

Power + Performance: ***X*=App-Task Customization**
Economics: $3-5B fab amortized over longer periods

The Trends



- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors



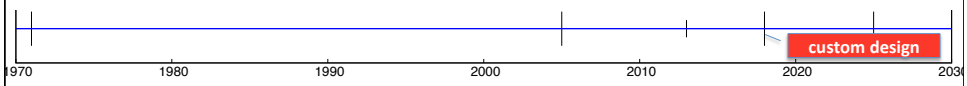
Power/Energy per Operation, Quality of Service

More economical: Reaches to more domains
- Differentiation: user experience, applications
- At both the Edge and the Cloud

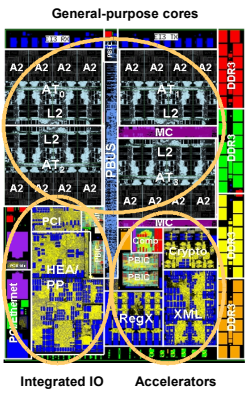
Partially custom: Reduce expense, design time
- Automated, quick-turn composition
- A few truly custom blocks

Short time span: More design customization
- 12 to 18 month from design to deployed
- Aggressive EDA/design turns

The Trends



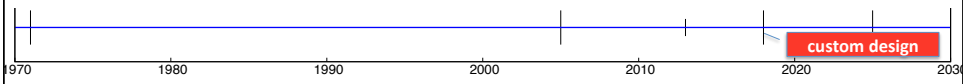
- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors



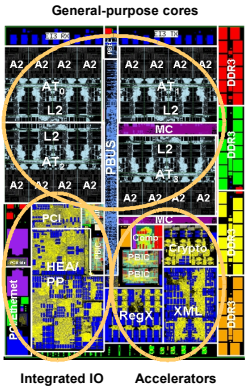
IBM PowerEN

8 app. cores
RegEx accelerator
XML accelerator
Crypto accelerator
Compression

The Trends

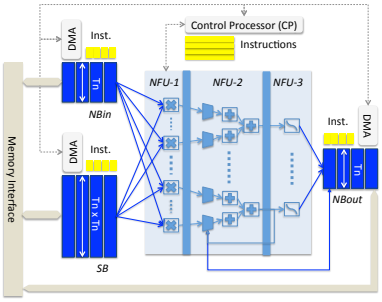


- Moore's Law: Doubling transistors, every two years
- Dennard Scaling: Faster, lower power & smaller transistors



IBM PowerEN

8 app. cores
RegEx accelerator
XML accelerator
Crypto accelerator
Compression

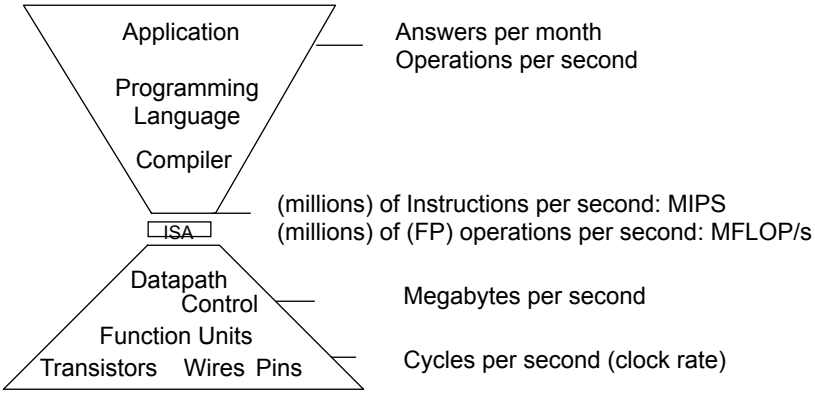


DianNao: A Small-Footprint High-throughput Accelerator for Ubiquitous Machine Learning

And Many More...

Parallelism
Heterogeneity
Customization
Vertical stacking (3D)
Additional: Memory + ***compute*** (Near data computation)
Additional: Alternative memory technologies
Etc.

Performance (§1.8)



Measuring Performance

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

Performance measurement Tools

- Benchmarks (Kernels, toy programs, synthetic), Traces, Mixes
- Hardware: Cost, delay, area, power estimation
- Simulation (many levels, ISA, RT, Gate, Circuit, …)
- Analytical modeling and Queuing Theory
- Rules of Thumb
- Fundamental “Laws”/Principles (ex. Amdahl Law).

SPEC: Standard Performance Evaluation Corporation
http://www.spec.org/

- **First Round 1989**
  - 10 programs yielding a single number ("SPECmarks")
- **Second Round 1992**
  - SPECInt92 (6 integer programs) and SPECfp92 (14 floating pt. programs).
  - Unlimited compiler flags settings.
- **Third Round 1995**
  - new set of programs: SPECint95 and SPECfp95
  - "benchmarks useful for 3 years"
  - Single flag setting for all programs: SPECint_base95, SPECfp_base95
- **Newest version is SPEC CPU 2006**
  - Many other benchmarks
  - Virtualization, parallel performance (HPC), Java, Cloud …

SPEC CPU 2006

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			

- 12 integer programs
  - 9 use C
  - 3 use C++
- 17 floating-point programs
  - 3 use C
  - 4 use C++
  - 6 use Fortran
  - 4 use a mixture of C and Fortran
- Package available at / afs/cs.pitt.edu/ projects/spec-cpu2006

Other benchmarks

- MediaBench – Embedded applications (pretty small)
  - http://cares.icsl.ucla.edu/MediaBench
- PARSEC – multithreaded general purpose applications
  - http://parsec.cs.princeton.edu/
- Transaction processing- TPC-C, SPECjbb
- Embedded Microprocessor Benchmark Consortium (EEMBC)
  - www.eembc.org
  - Auto, IoT, web (mobile), telecom, text & image, entertainment, … etc.
- Stanford parallel benchmarks (SPLASH), NAS, proxy apps..

Performance Evaluation

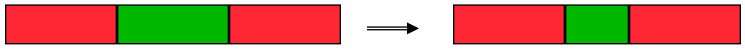
- For better or worse, benchmarks shape a field.
- Good products created when we have:
  - Good benchmarks
  - Good ways to summarize performance
- Given that sales is a function, in part, of performance relative to competition, companies invest in improving performance summary
- Reproducibility is important (should provide details of experiments)
- If benchmarks/summary are inadequate, then choose between improving product for real programs vs. improving product to get more sales ===> Sales almost always wins!
- Execution time and power are the measure of computer performance!

Amdahl's Law (§ 1.9)

Speedup due to enhancement E:

$$Speedup_{overall} = \frac{ExTime_{withoutE}}{ExTime_{withE}} = \frac{Performance_{withE}}{Performance_{withoutE}}$$

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



$$\frac{ExTime_{withE}}{ExTime_{withoutE}} = \left(1 - fraction_{enhanced}\right) + \frac{fraction_{enhanced}}{S}$$

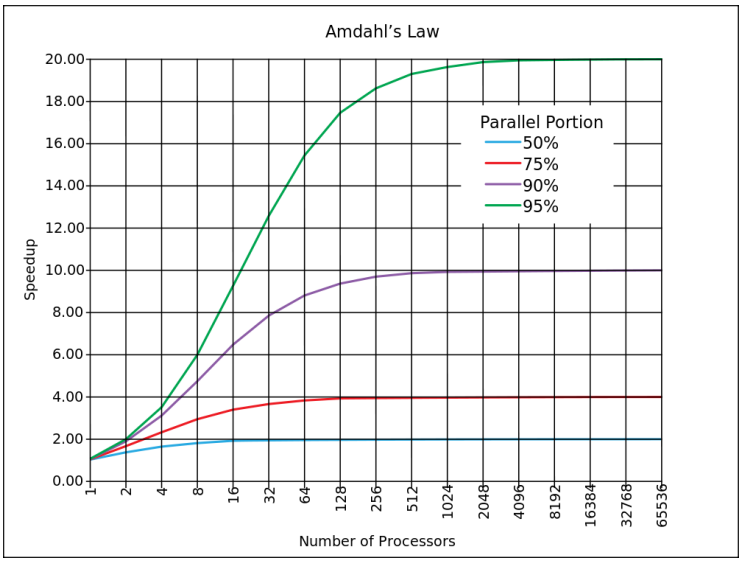
Example: Floating point instructions can be improved to run 2X; but only 10% of actual instructions are FP. What is the overall speedup?

A Second Example

- There's also a diminishing return in making an enhancement go faster, given how often it's used. Now, suppose we keep F=0.2 but change S from 10,000 to 2:

S	Sp	
10000	1.25	
1000	1.25	
100	1.25	***-- no further gain beyond 1.25 seedup!***
10	1.22	
8	1.22	
6	1.2	***-- not much gain beyond this point***
5	1.19	
3	1.15	
2	1.11	

Diminishing Returns



How to Summarize Performance

- Arithmetic mean (weighted arithmetic mean)
  - ex: tracks execution time: $\sum_{i=1}^{n} \frac{T_i}{n}$ or $\sum_{i=1}^{n} W_i * T_i$
- Harmonic mean (weighted harmonic mean) of rates
  - ex: track MFLOPS: $\frac{n}{\sum_{i=1}^{n} \frac{1}{Rate_i}}$
- Normalized execution time is handy for scaling performance (e.g., X times faster than Pentium 4)
- Geometric mean ==> $\sqrt[n]{\prod_{i=1}^{n} execution\_ratio_i}$

  where the execution ratio is relative to a reference machine

CPU time: Quantifying Individual Program Performance

- **Measure of program execution time**
  - Typically without system time
  - Purely the time to execute some “code”, given the processor, cache and memory
- **Classic CPU time**
  - **CPU time = IC * CPI * CC**
  - IC is instruction count
  - CPI is average cycles per instruction
  - CC is the clock cycle time
- **Speedup is the ratio of two CPU times**
  - Speedup = CPU time A / CPU time B

Computing CPU time

Average Cycles per Instruction (*CPI*) = $\sum_{j=1}^{n} CPI_j * F_j$

Where CPI_j is the number of cycles needed to execute instructions of type *j*, and F_j is the percentage (fraction) of instructions that are of type *j*.

Example: Base Machine (Reg / Reg)

Op	Freq	Cycles	$CPI_j * F_j$	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			1.5	

Typical Mix

Instructions Per Cycle (IPC) = 1 /*CPI*

$$CPU \quad time = Cycle \quad time \quad * \sum_{j=1}^{n} CPI_j \quad * \quad I_j$$

Where I_j is the number of instructions of type *j*, and

Cycle time is the inverse of the *clock rate*.

$$CPU\ time = \text{total \# of instructions} \times CPI \times Cycle\ time$$

Example: For some programs,

Machine A has a clock cycle time of 10 ns. and a CPI of 2.0
Machine B has a clock cycle time of 20 ns. and a CPI of 1.2

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

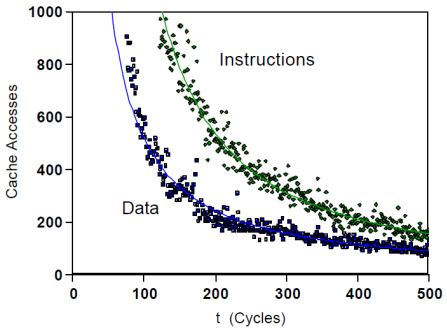
Aspects of CPU Performance

$$CPU\_time = \frac{Seconds}{program} = \frac{Instructions}{program} \times \frac{Cycles}{Instructions} \times \frac{Seconds}{Cycle}$$

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

Principle of locality

- Locality found in memory access instructions
  - Temporal locality: if an item is referenced, it will tend to be referenced again soon
  - Spatial locality: if an item is referenced, items whose addresses are close by tend to be referenced soon
- 90/10 locality rule
  - A program executes about 90% of its instructions in 10% of its code
- The working set concept
- We will look at how this principle is exploited in various microarchitecture techniques



Hartstein et al. JILP 2008