EECS 252
Graduate Computer Architecture
Lecture 2

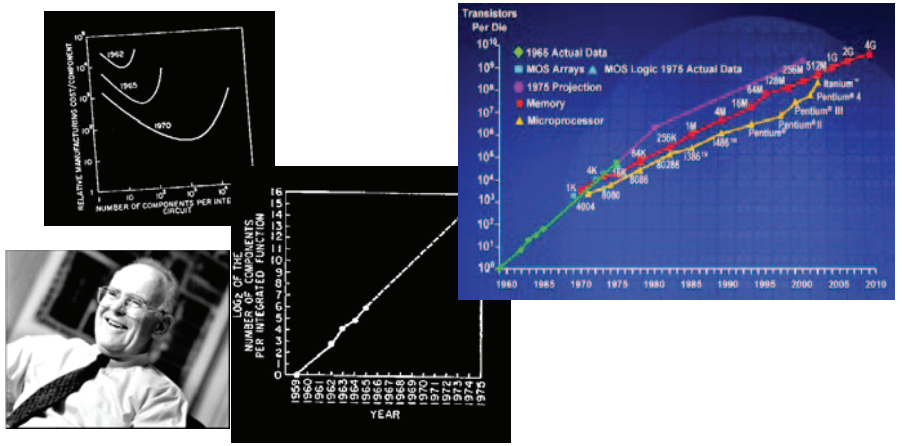
$\aleph_0$

Review of Instruction Sets and Pipelines
January 23th, 2012

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs252

Review: Moore's Law

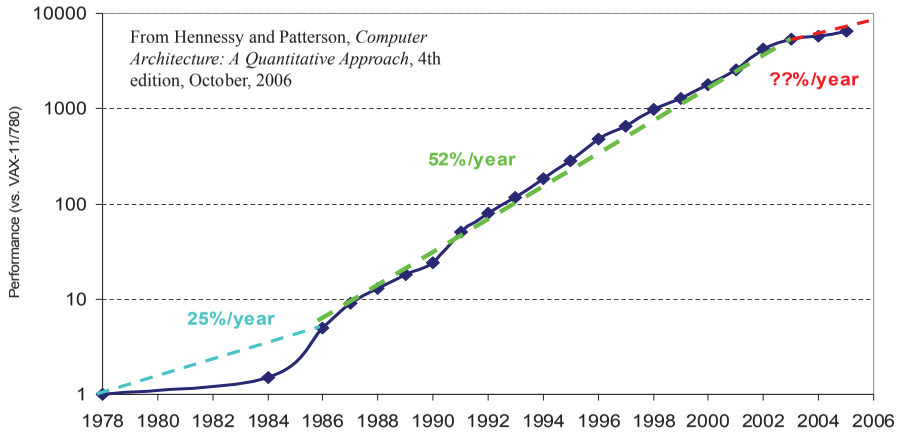


- "Cramming More Components onto Integrated Circuits"
  - Gordon Moore, Electronics, 1965
- # on transistors on cost-effective integrated circuit double every 18 months

"Bell's Law" – new class per decade

- Enabled by technological opportunities
- Smaller, more numerous and more intimately connected
- Brings in a new kind of application
- Used in many ways not previously imagined

Review: Crossroads: Uniprocessor Performance



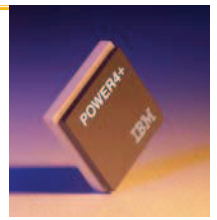
- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

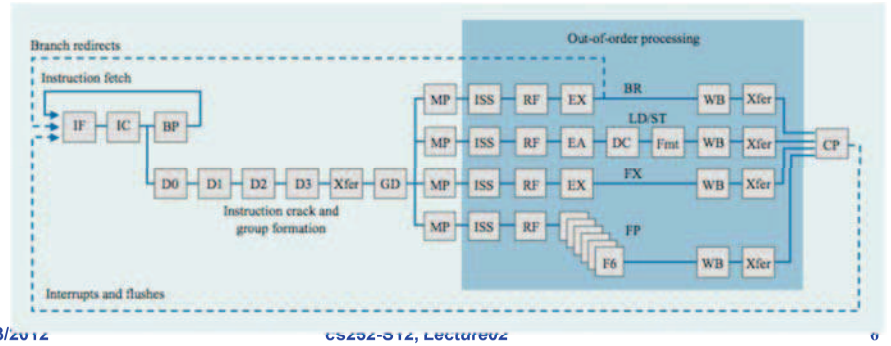
Modern ILP

- **Dynamically scheduled, out-of-order execution**
  - Current microprocessor 6-8 of instructions per cycle
  - Pipelines are 10s of cycles deep
    ⇒ many simultaneous instructions in execution at once
  - Unfortunately, hazards cause discarding of much work
- **What happens:**
  - Grab a bunch of instructions, determine all their dependences, eliminate dep's wherever possible, throw them all into the execution unit, let each one move forward as its dependences are resolved
  - Appears as if executed sequentially
  - On a trap or interrupt, capture the state of the machine between instructions perfectly
- **Huge complexity**
  - Complexity of many components scales as n^2 (issue width)
  - Power consumption big problem

IBM Power 4

- Combines: Superscalar and OOO
- Properties:
  - 8 execution units in out-of-order engine, each may issue an instruction each cycle.
  - In-order Instruction Fetch, Decode (compute dependencies)
  - Reordering for in-order commit





Have we reached the end of ILP?

- **Multiple processor easily fit on a chip**
- **Every major microprocessor vendor has gone to multithreaded cores**
  - Thread: loci of control, execution context
  - Fetch instructions from multiple threads at once, throw them all into the execution unit
  - Intel: hyperthreading, Sun:
  - Concept has existed in high performance computing for 20 years (or is it 40? CDC6600)
- **Vector processing**
  - Each instruction processes many distinct data
  - Ex: MMX
- **Raise the level of architecture – many processors per chip**

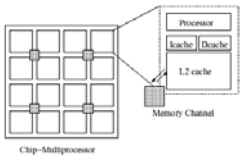
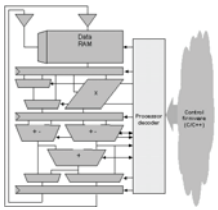


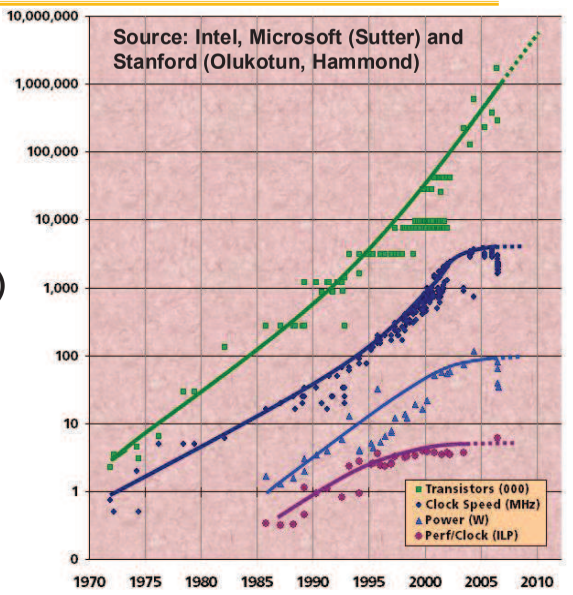
Figure 1. Chip-multiprocessor model.



Tensilica Configurable Proc

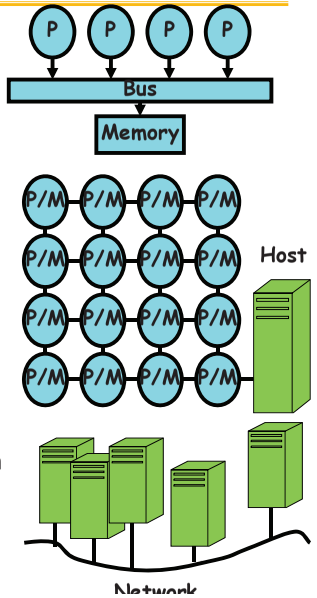
Limiting Forces: Clock Speed and ILP

- **Chip density is continuing increase ~2x every 2 years**
  - Clock speed is not
  - # processors/chip (cores) may double instead
- **There is little or no more Instruction Level Parallelism (ILP) to be found**
  - Can no longer allow programmer to think in terms of a serial programming model
- **Conclusion:**
  Parallelism must be exposed to software!

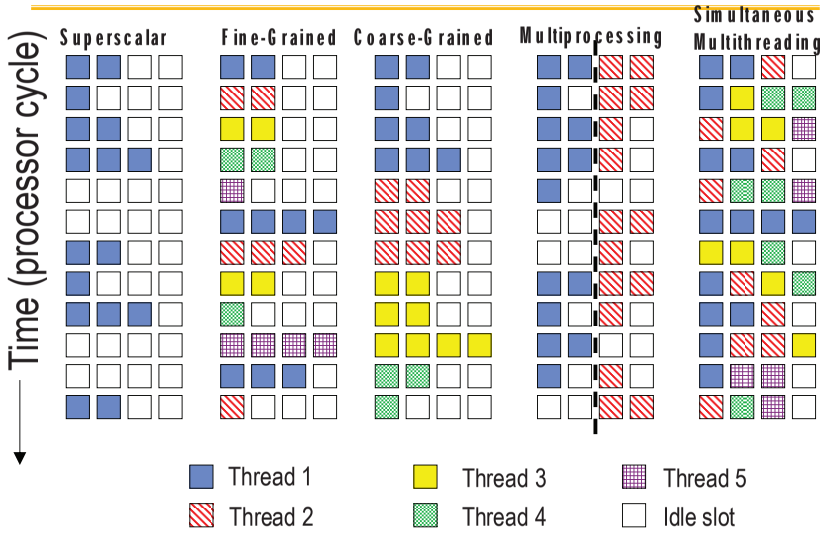


Examples of MIMD Machines

- **Symmetric Multiprocessor**
  - Multiple processors in box with shared memory communication
  - Current MultiCore chips like this
  - Every processor runs copy of OS
- **Non-uniform shared-memory with separate I/O through host**
  - Multiple processors
    - » Each with local memory
    - » general scalable network
  - Extremely light "OS" on node provides simple services
    - » Scheduling/synchronization
  - Network-accessible host for I/O
- **Cluster**
  - Many independent machine connected with general network
  - Communication through messages



Categories of Thread Execution



Today: Quick review of everything you should have learned

$\aleph_0$

(A countably-infinite set of computer architecture concepts)

Metrics used to Compare Designs

- **Cost**
  - Die cost and system cost
- **Execution Time**
  - average and worst-case
  - Latency vs. Throughput
- **Energy and Power**
  - Also peak power and peak switching current
- **Reliability**
  - Resiliency to electrical noise, part failure
  - Robustness to bad software, operator error
- **Maintainability**
  - System administration costs
- **Compatibility**
  - Software costs dominate

What is Performance?

- Latency (or response time or execution time)
  - time to complete one task
- Bandwidth (or throughput)
  - tasks completed per unit time

Definition: Performance

- Performance is in units of things per sec
  - bigger is better
- If we are primarily concerned with response time

$$\text{performance}(x) = \frac{1}{\text{execution_time}(x)}$$

" X is n times faster than Y" means

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

Performance: What to measure

- Usually rely on benchmarks vs. real workloads
- To increase predictability, collections of benchmark applications-- *benchmark suites* -- are popular
- SPECCPU: popular desktop benchmark suite
  - CPU only, split between integer and floating point programs
  - SPECint2000 has 12 integer, SPECfp2000 has 14 integer pgms
  - SPECCPU2006 to be announced Spring 2006
  - SPECSFS (NFS file server) and SPECWeb (WebServer) added as server benchmarks
- Transaction Processing Council measures server performance and cost-performance for databases
  - TPC-C Complex query for Online Transaction Processing
  - TPC-H models ad hoc decision support
  - TPC-W a transactional web benchmark
  - TPC-App application server and web services benchmark

Summarizing Performance

System	Rate (Task 1)	Rate (Task 2)
A	10	20
B	20	10

Which system is faster?

… depends who's selling

System	Rate (Task 1)	Rate (Task 2)	Average
A	10	20	15
B	20	10	15

Average throughput

System	Rate (Task 1)	Rate (Task 2)	Average
A	0.50	2.00	1.25
B	1.00	1.00	1.00

Throughput relative to B

System	Rate (Task 1)	Rate (Task 2)	Average
A	1.00	1.00	1.00
B	2.00	0.50	1.25

Throughput relative to A

Summarizing Performance over Set of Benchmark Programs

- Arithmetic mean of execution times t_i (in seconds):

$$\left(\frac{1}{n}\right)\sum_i t_i$$

- Harmonic mean of execution rates r_i (MIPS/MFLOPS):

$$\frac{n}{\sum_i \left(\frac{1}{r_i}\right)}$$

- Both equivalent to workload where each program is run the same number of times
- Can add weighting factors to model other workload distributions

Normalized Execution Time and Geometric Mean

- Measure speedup up relative to reference machine:

$$\text{ratio} = \frac{t_{Ref}}{t_A}$$

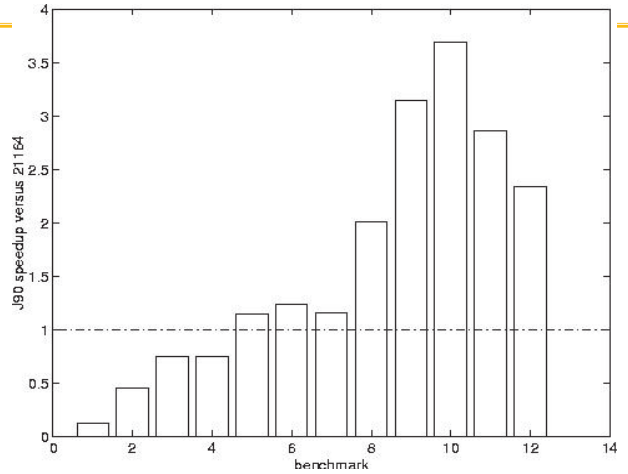
- Average time ratios using geometric mean:

$$\sqrt[n]{\prod_i ratio_i}$$

- Insensitive to machine chosen as reference
- Insensitive to run time of individual benchmarks
- Used by SPEC89, SPEC92, SPEC95, …, SPEC2006

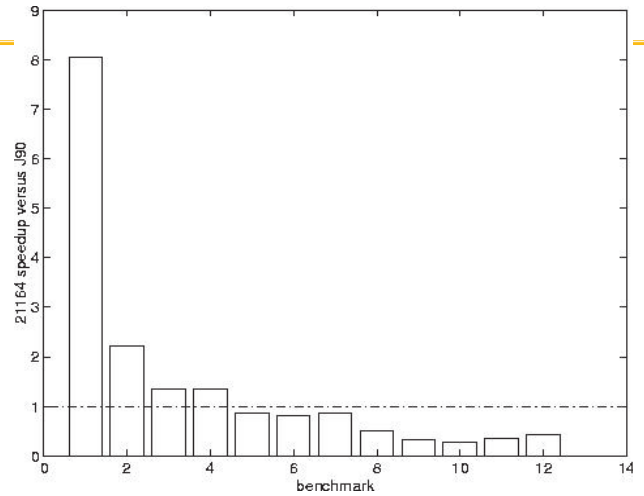
….. But beware that choice of reference machine can suggest what is "normal" performance profile:

Vector/Superscalar Speedup



- 100 MHz Cray J90 vector machine versus 300MHz Alpha 21164
- [LANL Computational Physics Codes, Wasserman, ICS'96]
- Vector machine peaks on a few codes????

Superscalar/Vector Speedup



- 100 MHz Cray J90 vector machine versus 300MHz Alpha 21164
- [LANL Computational Physics Codes, Wasserman, ICS'96]
- Scalar machine peaks on one code???

How to Mislead with Performance Reports

- Select pieces of workload that work well on your design, ignore others
- Use unrealistic data set sizes for application (too big or too small)
- Report throughput numbers for a latency benchmark
- Report latency numbers for a throughput benchmark
- Report performance on a kernel and claim it represents an entire application
- Use 16-bit fixed-point arithmetic (because it's fastest on your system) even though application requires 64-bit floating-point arithmetic
- Use a less efficient algorithm on the competing machine
- Report speedup for an inefficient algorithm (bubblesort)
- Compare hand-optimized assembly code with unoptimized C code
- Compare your design using next year's technology against competitor's year old design (1% performance improvement per week)
- Ignore the relative cost of the systems being compared
- Report averages and not individual results
- Report speedup over unspecified base system, not absolute times
- Report efficiency not absolute times
- Report MFLOPS not absolute times (use inefficient algorithm)

[David Bailey "Twelve ways to fool the masses when giving performance results for parallel supercomputers"]

CS 252 Administrivia

- Sign up! Web site is: http://www.cs.berkeley.edu/~kubitron/cs252
- Review: Chapter 1, Appendix A, B, C
- CS 152 home page, maybe "Computer Organization and Design (COD)2/e"
  - If did take a class, be sure COD Chapters 2, 5, 6, 7 are familiar
  - Copies in Bechtel Library on 2-hour reserve
- Resources for course on web site:
  - Check out the ISCA (International Symposium on Computer Architecture) 25th year retrospective on web site. Look for "Additional reading" below text-book description
  - Pointers to previous CS152 exams and resources
  - Lots of old CS252 material
  - Interesting links. Check out the: WWW Computer Architecture Home Page

CS 252 Administrivia

- First readings due today
  - Next two readings posted
- Read the assignment carefully, since the requirements vary about what you need to turn in
  - Submit results to website before class
    - » (will be a link up on handouts page)
  - You can have 5 total late days on assignments
    - » 10% per day afterwards
    - » Save late days!
  - If you access papers remotely, username and password:
    - » User: cs252
    - » Password: parallelism

RISC: The integrated systems view (Discussion of Papers)

- "The Case for the Reduced Instruction Set Computer"
  - Dave Patterson and David Ditzel
- "Comments on 'The Case for the Reduced Instruction Set Computer'"
  - Doug Clark and William Strecker
- ""Retrospective on High-Level Computer Architecture"
  - David Ditzel and David Patterson

In-class discussion of these papers

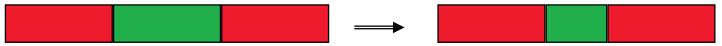
Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left[\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{\left(1 - Fraction_{enhanced}\right)}$$



Amdahl's Law example

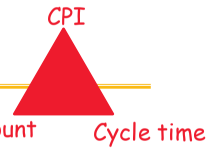
- New CPU 10X faster
- I/O bound server, so 60% time waiting for I/O

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

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

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

Computer 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	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

Cycles Per Instruction (Throughput)

"Average Cycles per Instruction"

CPI = (CPU Time * Clock Rate) / Instruction Count
= Cycles / Instruction Count

$$\text{CPU time} = \text{Cycle Time} \times \sum_{j=1}^{n} \text{CPI}_j \times \text{I}_j$$

$$\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times \text{F}_j \qquad \text{where } \text{F}_j = \frac{\text{I}_j}{\text{Instruction Count}}$$

"Instruction Frequency"

Example: Calculating CPI bottom up

Run benchmark and collect workload characterization (simulate, machine counters, or sampling)

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	(% 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 of instruction types in program

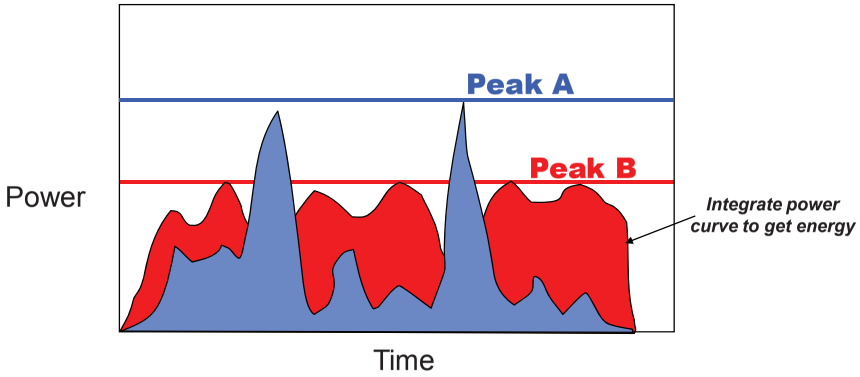
Design guideline: Make the common case fast

MIPS 1% rule: only consider adding an instruction of it is shown to add 1% performance improvement on reasonable benchmarks.

Power and Energy

- Energy to complete operation (Joules)
  - Corresponds approximately to battery life
  - (Battery energy capacity actually depends on rate of discharge)
- Peak power dissipation (Watts = Joules/second)
  - Affects packaging (power and ground pins, thermal design)
- di/dt, peak change in supply current (Amps/second)
  - Affects power supply noise (power and ground pins, decoupling capacitors)

Peak Power versus Lower Energy



- System **A** has higher peak power, but lower total energy
- System **B** has lower peak power, but higher total energy

Summary: Measurement

- Chip density is continuing increase ~2x every 2 years
  - Clock speed is not
  - # processors/chip (cores) may double instead
- Always have metrics and benchmarks in mind to justify comparisons between different systems
- Amdalls Law:

$$ExTime_{new} = ExTime_{old} \times \left[\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

- Performance measurement:

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