55:132/22C:160
High Performance Computer Architecture

Spring 2011

Instructor Information

- Instructor: Jon Kuhl (That's me)
  - Office: 4322 SC
  - Office Hours: 9:00-10:30 a.m. TTh (Other times by appointment)
  - E-mail: kuhl@engineering.uiowa.edu
  - Phone: (319) 335-5958
- TA: t.b.d.
  - Office: t.b.d
  - Office hours: t.b.d.

Class Info.

- Website:
  www.engineering.uiowa.edu/~hpca
- Texts:

  Required: Hennessy and Patterson, ***Computer Architecture—A Quantitative Approach***, Morgan Kaufmann, Fourth Edition, 2007

  Supplemental: Thomas and Moorby, ***The Verilog Hardware Description Language, Fifth Edition***, Springer Verlag, 2008.

  Additional Reference: Shen and Lipasti, ***Modern Processor Design--Fundamentals of Superscalar Processors***, McGraw Hill, 2005.

Course Objectives

- Understand quantitative measures for assessing and comparing processor performance
- Understand modern processor design techniques, including:
  - Pipelining
  - high performance memory architecture
  - instruction-level parallelism
  - multi-threading
  - Multi-core architecture
- Master the use of modern design tools (HDLs) to design and analyze processors
- Do case studies of contemporary processors
- Discuss future trends in processor design

Expected Background

- A previous course in computer architecture/organization covering:
  - Instruction set architecture (ISA)
  - Addressing modes
  - Assembly language
  - Basic computer organization
  - Memory system organization
    - Cache
    - virtual
  - Etc.
- 22C:060 or 55:035 or equivalent

Course Organization

- Homework assignments--several
- Several projects (design/analysis exercises using the Verilog HDL and ModelSim simulation environment)
- Two exams:

  Midterm—Th. March 10, in class

  Final—Mon. May 9, noon-2:00 p.m.

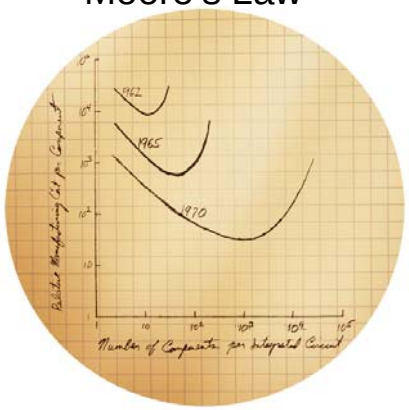
Course Organization--continued

- Grading:
  - Exams:
    - Better of midterm/final exam score: 35%
    - Poorer of midterm/final exam scores: 25%
  - Homework: 10%
  - Projects 30%

Historical Perspectives

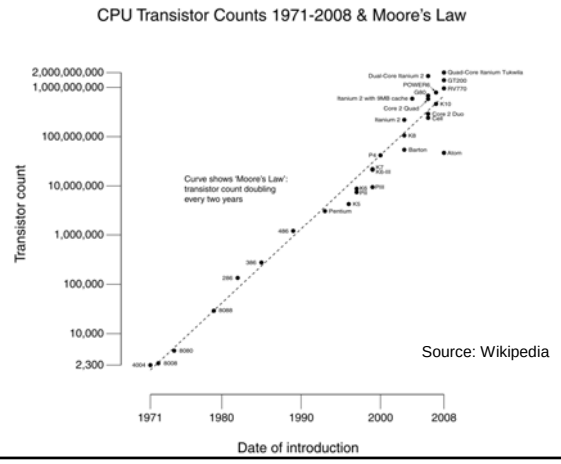
- The Decade of the 1970's: *"Birth of Microprocessors"*
  - Programmable Controller
  - Single-Chip Microprocessors
  - Personal Computers (PC)
- The Decade of the 1980's: *"Quantitative Architecture"*
  - Instruction Pipelining
  - Fast Cache Memories
  - Compiler Considerations
  - Workstations
- The Decade of the 1990's: *"Instruction-Level Parallelism"*
  - Superscalar,Speculative Microarchitectures
  - Aggressive Compiler Optimizations
  - Low-Cost Desktop Supercomputing

Moore's Law



Moore's Law (1965)

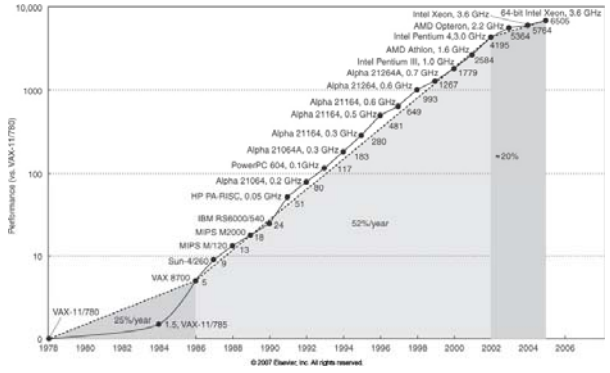
- The number of devices that can be integrated on a single piece of silicon will double roughly every 18-24 months
- Moore's law has held true for 40 years and will continue to hold for at least another decade, probably longer.



The Computer Architect's Challenge

- Make Moore's Law apply also to computer chip ***performance*** as well as density
- That is, make sure that the additional chip density (complexity) is utilized efficiently.
  - Note that Moore's law has roughly held for both chip density and clock frequency– chips have been getting faster as well as denser.
  - So fully exploiting the increase in density and clock speed should lead to performance increases well exceeding the growth rate of Moore's Law.

Processor Performance 1978-2006



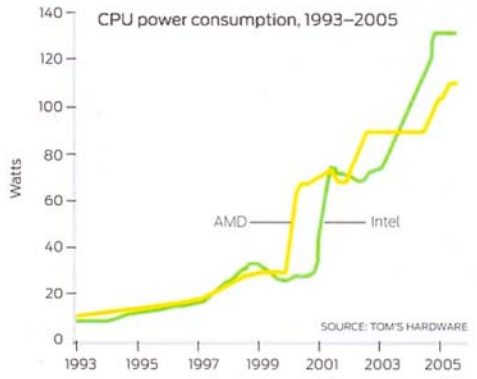
So What's Going On Here?

- In recent years, the increase in processor performance has begun to level off
  - No longer tracking Moore's law
- Have Computer Architects Failed?

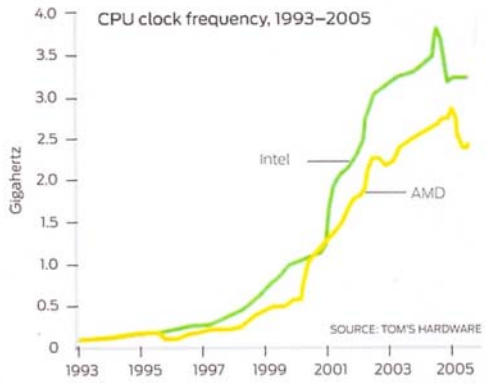
What is Constraining Performance?

- Diminishing Returns on Attempts to Exploit Instruction-level Parallelism
- Power and Heat Dissipation Issues
- Stagnating Clock Rates
- Lagging Memory Latencies

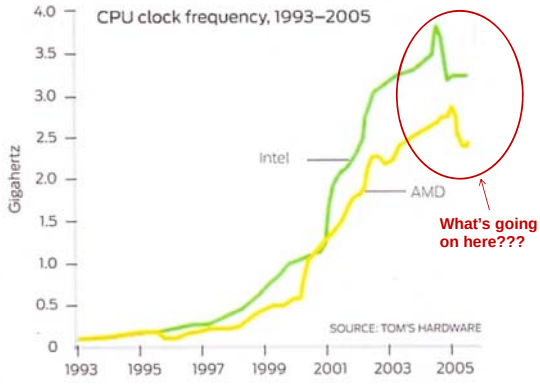
Moore's Law--Processor Power Consumption



Moore's Law—Clock Frequency



Moore's Law—Clock Frequency



Relationship between clock rate and power

- Intel Estimate[1]:
  - Increasing clock rate by 25% will yield approx. 15% performance increase
  - But power consumption will be ***doubled***
- Power Consumption/Heat Dissapation issues are ushering a new era in CPU design
  - Focus on performance per watt
  - Causing fundamental rethinking of archtecture

[1] Phillip E. Ross, " Why Clock Frequency Stalled", **IEEE Spectrum,** April, 2008

Performance and Cost

- How should we measure performance?
- Not so simple
  - Scientific simulation – FP performance
  - Program development – Integer performance
  - Commercial workload – Memory, I/O

Performance of Computers

- Want to buy the fastest computer for what you want to do?
  - Workload is all-important
  - Correct measurement and analysis
- Want to design the fastest computer for what the customer wants to pay?
  - Cost is always an important criterion
- Speed is not always the only performance criteria:
  - Power
  - Area

Defining Performance

- What is important to whom?
- Computer system user
  - Minimize elapsed time for program = time_end – time_start
  - Called response time
- Computer center manager
  - Maximize completion rate = #jobs/second
  - Called throughput

Improve Performance

- Improve (a) response time or (b) throughput?
  - Faster CPU
    - Helps both (a) and (b)
  - Add more CPUs
    - Helps (b) and perhaps (a) due to less queuing

Simple Performance Comparison

- Machine A is n times faster than machine B iff perf(A)/perf(B) = time(B)/time(A) = n
- Machine A is x% faster than machine B iff perf(A)/perf(B) = time(B)/time(A) = 1 + x/100
- E.g. time(A) = 10s, time(B) = 15s
  - 15/10 = 1.5 => A is 1.5 times faster than B
  - 15/10 = 1.5 => A is 50% faster than B

Possible Performance Metrics

- MIPS and MFLOPS
- MIPS (Millions of Instructions per Second
  = instruction count/(execution time x 10^6)
  = clock rate/(CPI x 10^6)
- MFLOPS (millions of floating pt. ops per second)
  – Generally refers peak (max. possible) rate
- Both have serious shortcomings

Problems with MIPS

- E.g. without FP hardware, an FP op may take 50 single-cycle instructions
- With FP hardware, only one 2-cycle instruction

- Thus, adding FP hardware:
  - CPI increases (why?) 50/50 => 2/1
  - Instructions/program decreases (why?) 50 => 1
  - Total execution time decreases 50 => 2
- BUT, MIPS gets worse! 50 MIPS => 2 MIPS

Problems with MIPS

- Ignores program
- Usually used to quote peak performance
  – Ideal conditions => guarantee not to exceed!
- When is MIPS ok?
  – Same compiler, same ISA
  – E.g. same binary running on Pentium-III, IV
  – Why? Instr/program is constant and can be ignored

Other Metrics

- MFLOPS = FP ops in program/(execution time x 10^6)
- Assuming FP ops independent of compiler and ISA
  – Often safe for numeric codes: matrix size determines # of FP ops/program
  – However, not always safe:
    - Missing instructions (e.g. FP divide, sqrt/sin/cos)
    - Optimizing compilers
- Relative MIPS and normalized MFLOPS
  – Normalized to some common baseline machine
    - E.g. VAX MIPS in the 1980s

Which Programs?

- Execution time of what program?
- Best case – you always run the same set of programs
  - Port them and time the whole workload
- In reality, use benchmarks
  - Programs chosen to measure performance
  - Predict performance of actual workload
  - Saves effort and money
  - Representative? Honest? Benchmarketing…

Types of Benchmarks

- Real programs
  - representative of real workload
  - only accurate way to characterize performance
  - requires considerable work
- Kernels or microbenchmarks
  - "representative" program fragments
  - good for focusing on individual features not big picture
- Instruction mixes
  - instruction frequency of occurrence; calculate CPI

Benchmarks: SPEC2000

- System Performance Evaluation Cooperative
  - Formed in 80s to combat benchmarketing
  - SPEC89, SPEC92, SPEC95,Spec2000, now SPEC2006
- 12 integer and 14 floating-point programs
  - Sun Ultra-5 300MHz reference machine has score of 100
  - Report geometric mean of ratios to reference machine

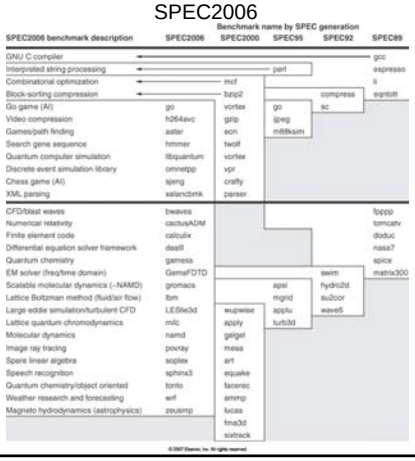
Benchmarks: SPEC CINT2000

Benchmark	Description
164.gzip	Compression
175.vpr	FPGA place and route
176.gcc	C compiler
181.mcf	Combinatorial optimization
186.crafty	Chess
197.parser	Word processing, grammatical analysis
252.eon	Visualization (ray tracing)
253.perlbmk	PERL script execution
254.gap	Group theory interpreter
255.vortex	Object-oriented database
256.bzip2	Compression
300.twolf	Place and route simulator

Benchmarks: SPEC CFP2000

Benchmark	Description
168.wupwise	Physics/Quantum Chromodynamics
171.swim	Shallow water modeling
172.mgrid	Multi-grid solver: 3D potential field
173.applu	Parabolic/elliptic PDE
177.mesa	3-D graphics library
178.galgel	Computational Fluid Dynamics
179.art	Image Recognition/Neural Networks
183.equake	Seismic Wave Propagation Simulation
187.facerec	Image processing: face recognition
188.ammp	Computational chemistry
189.lucas	Number theory/primality testing
191.fma3d	Finite-element Crash Simulation
200.sixtrack	High energy nuclear physics accelerator design
301.apsi	Meteorology: Pollutant distribution

SPEC2006



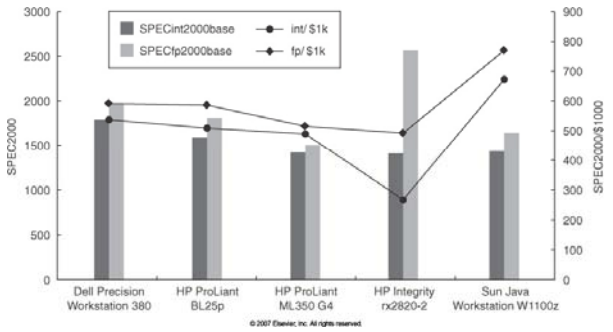
Benchmark Pitfalls

- Benchmark not representative
  - If your workload is I/O bound, SPECint is useless
- Benchmark is too old
  - Benchmarks age poorly; benchmarketing pressure causes vendors to optimize compiler/hardware/software to benchmarks
  - Need to be periodically refreshed

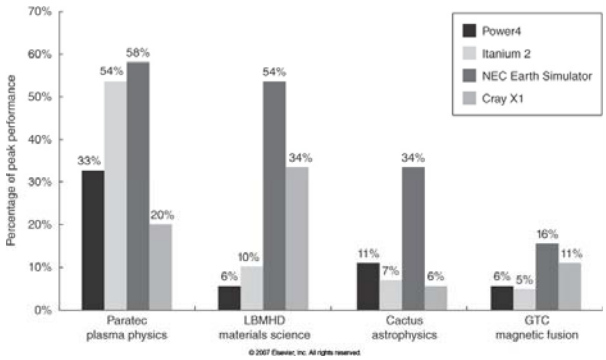
Benchmark Pitfalls

- Choosing benchmark from the wrong application space
  - e.g., in a realtime environment, choosing gcc
- Choosing benchmarks from no application space
  - e.g., synthetic workloads, esp. unvalidated ones
- Using toy benchmarks (dhrystone, whetstone)
  - e.g., used to prove the value of RISC in early 80's
- Mismatch of benchmark properties with scale of features studied
  - e.g., using SPECINT for large cache studies

Performance versus cost-performance



The Problem with Peak Performance Metrics(MFLOPS)



Processor Performance Equation

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

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

(code size) (CPI) (cycle time)

Scalar to Superscalar

- Scalar processor—Fetches and issues at most one instruction per machine cycle
- Superscalar processor-- Fetches and issues multiple instructions per machine cycle
- Can also define superscalar in terms of how many instructions can complete execution in a given machine cycle.
- Note that only a superscalar architecture can achieve a CPI of less than 1

Processor Performance Equation

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

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

(code size) (CPI) (cycle time)

- In the 1980's (decade of pipelining):
  - CPI: 5.0 => 1.15
- In the 1990's (decade of superscalar):
  - CPI: 1.15 => 0.5 (best case)

Processor Performance Equation

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

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

(code size) (CPI) (cycle time)

Architecture --> Implementation --> Realization

Compiler Designer Processor Designer Chip Designer

PPE—Considering Power

$$\frac{\text{Time}}{\text{Program}} \times \text{Watts} =$$

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

(code size) (CPI) (cycle time)

Architecture --> Implementation --> Realization

Compiler Designer Processor Designer Chip Designer

Processor Performance Equation

- Instructions/Program
  - Instructions executed, not static code size
  - Determined by algorithm, compiler, ISA
- Cycles/Instruction
  - Determined by ISA and CPU organization
  - Overlap among instructions reduces this term
- Time/cycle
  - Determined by technology, organization, clever circuit design

Overall Goal

- Minimize time, which is the product, NOT isolated terms
- Common error to miss terms while devising optimizations
  - E.g. ISA change to decrease instruction count
  - BUT leads to CPU organization which makes clock slower
- Bottom line: terms are inter-related
- This is the crux of the RISC vs. CISC argument

PPE Example

- Machine A: clock 1ns, CPI 2.0, for program P
- Machine B: clock 2ns, CPI 1.2, for program P
- Which is faster and how much?

  Time/Program = instr/program x cycles/instr x sec/cycle

  Time(A) = N x 2.0 x 1 = 2N

  Time(B) = N x 1.2 x 2 = 2.4N

  Compare: Time(B)/Time(A) = 2.4N/2N = 1.2
- So, Machine A is 20% faster than Machine B for this program

PPE Example

Keep clock(A) @ 1ns and clock(B) @2ns

For equal performance, if CPI(B)=1.2, what is CPI(A)?

Time(B)/Time(A) = 1 = (Nx2x1.2)/(Nx1xCPI(A))

CPI(A) = 2.4

PPE Example

- Keep CPI(A)=2.0 and CPI(B)=1.2
- For equal performance, if clock(B)=2ns, what is clock(A)?

Time(B)/Time(A) = 1 = (N x 2.0 x clock(A))/(N x 1.2 x 2)

clock(A) = 1.2ns

Another Example

OP	Freq	Cycles
ALU	43%	1
Load	21%	1
Store	12%	2
Branch	24%	2

- Assume stores can execute in 1 cycle by slowing clock 15%
- Should this be implemented?

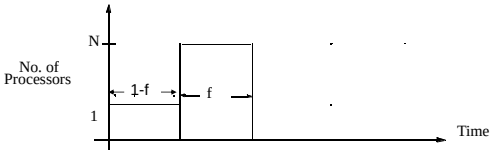
Example-- Let's do the math:

OP	Freq	Cycles
ALU	43%	1
Load	21%	1
Store	12%	2
Branch	24%	2

- Old CPI = 0.43 + 0.21 + 0.12 x 2 + 0.24 x 2 = 1.36
- New CPI = 0.43 + 0.21 + 0.12 + 0.24 x 2 = 1.24
- Speedup = old time/new time
  = {P x old CPI x T}/{P x new CPI x 1.15 T}
  = (1.36)/(1.24 x 1.15) = 0.95
- Answer: Don't make the change

Amdahl's Law

(Originally formulated for *vector processing)*

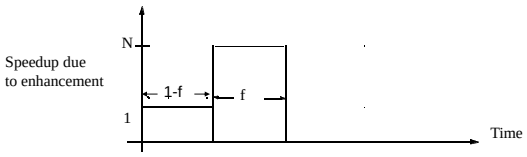


- f = fraction of program that is vectorizable
- (1-f) = fraction that is *serial*
- N = speedup for vectorizable portion
- Overall speedup:

$$Speedup = \frac{1}{(1-f) + \frac{f}{N}}$$

Generalization of Amdahl's Law

(To apply to any processor performance enhancement)



- f = fraction of program that can take advantage of the enhancement
- (1-f) = fraction that cannot take advantage
- N = speedup for enhanced portion
- Overall speedup:

$$Speedup = \frac{1}{(1-f) + \frac{f}{N}}$$

Amdahl's Law--Continued

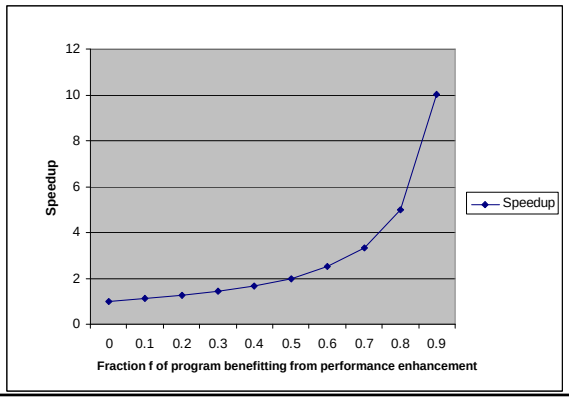
- Performance bottleneck
- Even if N is infinite
  - Performance limited by non-enhanceable portion (1-f)

$$\lim_{N\to\infty} \frac{1}{1-f+\frac{f}{N}} = \frac{1}{1-f}$$

Ramifications of Amdahl's Law

- Consider: f = 0.9, (1-f) = 0.1
  For N → , Speedup → 10
- Consider: f = 0.5, (1-f) = 0.5
  For N→ infinity, Speedup → 2
- Consider: f – 0.1, (1-f) = 0.9
  For N → infinity, Speedup → 1.1

Maximum Achievable Speedup



Amdahl's Law Example

- An enhancement to a processor architecture is proposed that would decrease the CPI for floating point multiply instructions from 20 cycles to 1 cycle (a speedup of 20). The CPI of all other instructions will be unchanged. What will be the overall processor speedup resulting from this modification?

Amdahl's Law Example (continued)

Suppose that, in the original design, floating point multiplies accounted for 6% of the total execution time of a "typical program"

Then by Amdahl's law the speedup due to the enhanced floating point multiply will be

$$S = \frac{1}{(1 - 0.06) + 0.06/20} = 1.06$$

Amdahl's Law Example (continued)

Now suppose that, for a different program, floating point multiplies account for 60% of the total execution time in the original design

Then by Amdahl's law the speedup due to the enhanced floating point multiply (for this particular program) will be

$$S = \frac{1}{(1 - 0.6) + 0.6/20} = 2.33$$