inst.eecs.berkeley.edu/~cs61c

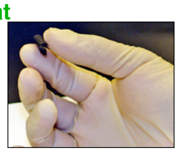
CS61C : Machine Structures

Lecture #29 Performance & Parallel Intro

2007-8-14

Scott Beamer, Instructor

"Paper" Battery Developed by Researchers at Rensselaer



www.bbc.co.uk

"Last time…"

- Magnetic Disks continue rapid advance: 60%/yr capacity, 40%/yr bandwidth, slow on seek, rotation improvements, MB/$ improving 100%/yr?
  - Designs to fit high volume form factor
  - PMR a fundamental new technology
    - breaks through barrier
- RAID
  - Higher performance with more disk arms per $
  - Adds option for small # of extra disks
  - Can nest RAID levels
  - Today RAID is > tens-billion dollar industry, 80% nonPC disks sold in RAIDs, started at Cal

Peer Instruction

1. RAID 1 (mirror) and 5 (rotated parity) help with performance and availability
2. RAID 1 has higher cost than RAID 5
3. Small writes on RAID 5 are slower than on RAID 1

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	TTF
7:	TTT

Why Performance? Faster is better!

- Purchasing Perspective: given a collection of machines (or upgrade options), which has the
  - best performance ?
  - least cost ?
  - best performance / cost ?
- Computer Designer Perspective: faced with design options, which has the
  - best performance improvement ?
  - least cost ?
  - best performance / cost ?
- All require basis for comparison and metric for evaluation!
- Solid metrics lead to solid progress!

Two Notions of "Performance"

Plane	DC to Paris	Top Speed	Passen-gers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concorde	3 hours	1350 mph	132	178,200

- Which has higher performance?
- Interested in time to deliver 100 passengers?
- Interested in delivering as many passengers per day as possible?
- In a computer, time for one task called
  Response Time or Execution Time
- In a computer, tasks per unit time called
  Throughput or Bandwidth

Definitions

- Performance is in units of things per sec
  - bigger is better
- If we are primarily concerned with response time
  - $\text{performance}(x) = \dfrac{1}{\text{execution\_time}(x)}$

" F(ast) is *n* times faster than S(low) " means…

$$n = \frac{\text{performance}(F)}{\text{performance}(S)} = \frac{\text{execution\_time}(S)}{\text{execution\_time}(F)}$$

Example of Response Time v. Throughput

- Time of Concorde vs. Boeing 747?
  - Concord is 6.5 hours / 3 hours = 2.2 times faster
- Throughput of Boeing vs. Concorde?
  - Boeing 747: 286,700 pmph / 178,200 pmph = 1.6 times faster
- Boeing is 1.6 times ("60%") faster in terms of throughput
- Concord is 2.2 times ("120%") faster in terms of flying time (response time)

We will focus primarily on response time.

Words, Words, Words…

- Will (try to) stick to "n times faster"; its less confusing than "m % faster"
- As faster means both decreased execution time and increased performance, to reduce confusion we will (and you should) use "improve execution time" or

  "improve performance"

What is Time?

- Straightforward definition of time:
  - Total time to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, ...
  - "real time", "response time" or "elapsed time"
- Alternative: just time processor (CPU) is working only on your program (since multiple processes running at same time)
  - "CPU execution time" or "CPU time"
  - Often divided into system CPU time (in OS) and user CPU time (in user program)

How to Measure Time?

- Real Time ⇒ Actual time elapsed
- CPU Time: Computers constructed using a clock that runs at a constant rate and determines when events take place in the hardware
  - These discrete time intervals called clock cycles (or informally clocks or cycles)
  - Length of clock period: clock cycle time (e.g., 2 nanoseconds or 2 ns) and clock rate (e.g., 500 megahertz, or 500 MHz), which is the inverse of the clock period; use these!

Measuring Time using Clock Cycles (1/2)

- CPU execution time for a program

  = Clock Cycles for a program x Clock Period

- or

$$= \frac{\text{Clock Cycles for a program}}{\text{Clock Rate}}$$

Clock 

Measuring Time using Clock Cycles (2/2)

- One way to define clock cycles:

Clock Cycles for program

= Instructions for a program (called "Instruction Count")

x Average Clock cycles Per Instruction (abbreviated "CPI")

- CPI one way to compare two machines with same instruction set, since Instruction Count would be the same

Performance Calculation (1/2)

- CPU execution time for program
  = Clock Cycles for program
  x Clock Cycle Time
- Substituting for clock cycles:

  CPU execution time for program
  = (Instruction Count x CPI)
  x Clock Cycle Time

  = Instruction Count x CPI x Clock Cycle Time

Performance Calculation (2/2)

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

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

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

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}}$$

- Product of all 3 terms: if missing a term, can't predict time, the real measure of performance

How Calculate the 3 Components?

- Clock Cycle Time: in specification of computer (Clock Rate in advertisements)
- Instruction Count:
  - Count instructions in loop of small program
  - Use simulator to count instructions
  - Hardware counter in spec. register
    - (Pentium II,III,4)
- CPI:
  - Calculate: $\frac{\text{Execution Time / Clock cycle time}}{\text{Instruction Count}}$
  - Hardware counter in special register (PII,III,4)

Calculating CPI Another Way

- First calculate CPI for each individual instruction (`add`, `sub`, `and`, etc.)
- Next calculate frequency of each individual instruction
- Finally multiply these two for each instruction and add them up to get final CPI (the weighted sum)

Example (RISC processor)

Op	$Freq_i$	CPI_i	Prod	(% Time)
ALU	50%	1	.5	(23%)
Load	20%	5	1.0	(45%)
Store	10%	3	.3	(14%)
Branch	20%	2	.4	(18%)
	Instruction Mix		2.2	(Where time spent)

- What if Branch instructions twice as fast?

What Programs Measure for Comparison?

- Ideally run typical programs with typical input before purchase, or before even build machine
  - Called a "workload"; For example:
  - Engineer uses compiler, spreadsheet
  - Author uses word processor, drawing program, compression software
- In some situations its hard to do
  - Don't have access to machine to "benchmark" before purchase
  - Don't know workload in future

Benchmarks

- Obviously, apparent speed of processor depends on code used to test it
- Need industry standards so that different processors can be fairly compared
- Companies exist that create these **benchmarks**: "typical" code used to evaluate systems
- Need to be changed every ~5 years since designers could (and do!) target for these standard benchmarks

Example Standardized Benchmarks (1/2)

- Standard Performance Evaluation Corporation (SPEC) SPEC CPU2006
  - CINT2006 12 integer (perl, bzip, gcc, go, ...)
  - CFP2006 17 floating-point (povray, bwaves, ...)
  - All relative to base machine (which gets 100) Sun Ultra Enterprise 2 w/296 MHz UltraSPARC II
  - They measure
    - System speed (SPECint2006)
    - System throughput (SPECint_rate2006)
  - `www.spec.org/osg/cpu2006/`

Example Standardized Benchmarks (2/2)

- SPEC
  - Benchmarks distributed in source code
  - Members of consortium select workload
    - 30+ companies, 40+ universities, research labs
  - Compiler, machine designers target benchmarks, so try to change every 5 years
  - SPEC CPU2006:

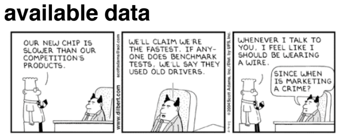
CINT2006		
perlbench	C	Perl Programming language
bzip2	C	Compression
gcc	C	C Programming Language Compiler
mcf	C	Combinatorial Optimization
gobmk	C	Artificial Intelligence : Go
hmmer	C	Search Gene Sequence
sjeng	C	Artificial Intelligence : Chess
libquantum	C	Simulates quantum computer
h264ref	C	H.264 Video compression
omnetpp	C++	Discrete Event Simulation
astar	C++	Path-finding Algorithms
xalancbmk	C++	XML Processing

CFP2006		
bwaves	Fortran	Fluid Dynamics
gamess	Fortran	Quantum Chemistry
milc	C	Physics / Quantum Chromodynamics
zeusmp	Fortran	Physics / CFD
gromacs	C,Fortran	Biochemistry / Molecular Dynamics
cactusADM	C,Fortran	Physics / General Relativity
leslie3d	Fortran	Fluid Dynamics
namd	C++	Biology / Molecular Dynamics
dealll	C++	Finite Element Analysis
soplex	C++	Linear Programming, Optimization
povray	C++	Image Ray-tracing
calculix	C,Fortran	Structural Mechanics
GemsFDTD	Fortran	Computational Electromegnetics
tonto	Fortran	Quantum Chemistry
lbm	C	Fluid Dynamics
wrf	C,Fortran	Weather
sphinx3	C	Speech recognition

Performance Evaluation: An Aside Demo

If we're talking about performance, let's discuss the ways shady salespeople have fooled consumers (so you don't get taken!)

5. Never let the user touch it
4. Only run the demo through a script
3. Run it on a stock machine in which "no expense was spared"
2. Preprocess all available data
1. Play a movie



Peer Instruction

A. Rarely does a company selling a product give unbiased performance data.

B. The Sieve of Eratosthenes and Quicksort were early effective benchmarks.

C. A program runs in 100 sec. on a machine, `mult` accounts for 80 sec. of that. If we want to make the program run 6 times faster, we need to up the speed of `mults` by AT LEAST 6.

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	TTF
7:	TTT

"And in conclusion…"

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

- Latency v. Throughput
- Performance doesn't depend on any single factor: need Instruction Count, Clocks Per Instruction (CPI) and Clock Rate to get valid estimations
- User Time: time user waits for program to execute: depends heavily on how OS switches between tasks
- CPU Time: time spent executing a single program: depends solely on design of processor (datapath, pipelining effectiveness, caches, etc.)
- Benchmarks
  - Attempt to predict perf, Updated every few years
  - Measure everything from simulation of desktop graphics programs to battery life
- Megahertz Myth
  - MHz ≠ performance, it's just one factor

Administrivia

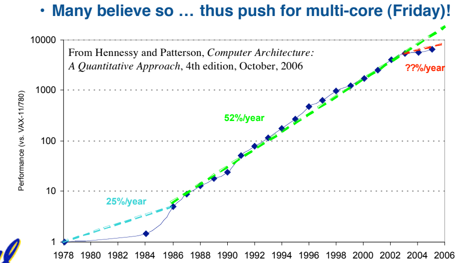
- HW8 due tonight at 11:59pm (no slip)
- Put in regrade requests now for any assignment past HW2
- Final: Thursday 7-10pm @ 10 Evans
  - NO backpacks, cells, calculators, pagers, PDAs
  - 2 writing implements (we'll provide write-in exam booklets) – pencils ok!
  - Two pages of notes (both sides) 8.5"x11" paper
  - One green sheet
- Scott is holding extra OH today 4-6 in 329 Soda
- Course Survey last lecture, 2pts for doing it

Big Problems Show Need for Parallel

- Simulation: the Third Pillar of Science
  - Traditionally perform experiments or build systems
  - Limitations to standard approach:
    - Too difficult – build large wind tunnels
    - Too expensive – build disposable jet
    - Too slow – wait for climate or galactic evolution
    - Too dangerous – weapons, drug design
  - Computational Science:
    - Simulate the phenomenon on computers
    - Based on physical laws and efficient numerical methods
- Search engines needs to build an index for the entire Internet
- Pixar needs to render movies
- Desire to go green and use less power
- Intel, Microsoft, Apple, Dell, etc. would like to sell you a new computer next year

What Can We Do?

- Wait for our machines to get faster?
  - Moore's law tells us things are getting better; why not stall for the moment?
- Moore on last legs!
  - Many believe so … thus push for multi-core (Friday)!



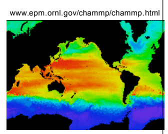
Let's Put Many CPUs Together!

- Distributed computing (SW parallelism)
  - Many separate computers (each with independent CPU, RAM, HD, NIC) that communicate through a network
  - Grids (home computers across Internet) and Clusters (all in one room)
  - Can be "commodity" clusters, 100K+ nodes
  - About being able to solve "big" problems, not "small" problems faster
- Multiprocessing (HW parallelism)
  - Multiple processors "all in one box" that often communicate through shared memory
  - Includes multicore (many new CPUs)

Performance Requirements

- Performance terminology
  - the FLOP: FLoating point OPeration
  - Computing power in FLOPS (FLOP per Second)
- Example: Global Climate Modeling
  - Divide the world into a grid (e.g. 10 km spacing)
  - Solve fluid dynamics equations for each point & minute
    - Requires about 100 Flops per grid point per minute
  - Weather Prediction (7 days in 24 hours):
    - 56 Gflops
  - Climate Prediction (50 years in 30 days):
    - 4.8 Tflops
- Perspective
  - Pentium 4 3GHz Desktop Processor
    - ~6-12 Gflops
    - Climate Prediction would take ~50-100 years

www.epm.ornl.gov/chammp/chammp.html



Reference:http://www.hpcwire.com/hpcwire/hpcwireWWW/04/0827/108259.html

The Future of Parallelism

"Parallelism is the biggest challenge since high level programming languages. It's the biggest thing in 50 years because industry is betting its future that parallel programming will be useful."

– David Patterson

Distributed Computing Themes

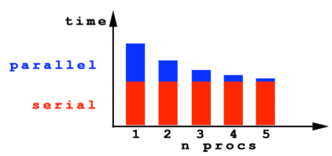
- Let’s network many disparate machines into one compute cluster
- These could all be the same (easier) or very different machines (harder)
- Common themes
  - “Dispatcher” gives jobs & collects results
  - “Workers” (get, process, return) until done
- Examples
  - SETI@Home, BOINC, Render farms
  - Google clusters running MapReduce

Distributed Computing Challenges

- Communication is fundamental difficulty
  - Distributing data, updating shared resource, communicating results
  - Machines have separate memories, so no usual inter-process communication – need network
  - Introduces inefficiencies: overhead, waiting, etc.
- Need to parallelize algorithms
  - Must look at problems from parallel standpoint
  - Tightly coupled problems require frequent communication (more of the slow part!)
  - We want to decouple the problem
    - Increase data locality
    - Balance the workload

Things to Worry About: Parallelizing Code

- Applications can almost never be completely parallelized; some serial code remains



- s is serial fraction of program, P is # of processors
- Amdahl’s law:

Speedup(P) = Time(1) / Time(P)

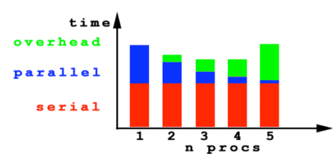
$\leq 1 / (s + ((1-s) / P))$, and as $P \to \infty$

$\leq 1/s$

- Even if the parallel portion of your application speeds up perfectly, your performance may be limited by the sequential portion

But… What About Overhead?

- Amdahl’s law ignores overhead
  - E.g. from communication, synchronization
- Amdahl’s is useful for bounding a program’s speedup, but cannot predict speedup



Peer Instruction of Assumptions

1. Writing & managing SETI@Home is relatively straightforward; just hand out & gather data
2. Most parallel programs that, when run on N (N big) identical supercomputer processors will yield close to N x performance increase
3. The majority of the world’s computing power lives in supercomputer centers

	ABC
1:	FFF
2:	FFT
3:	FTF
4:	FTT
5:	TFF
6:	TFT
7:	TTF
8:	TTT

Summary

- Parallelism is necessary
  - It looks like the future of computing…
  - It is unlikely that serial computing will ever catch up with parallel computing
- Software parallelism
  - Grids and clusters, networked computers
  - Two common ways to program:
    - Message Passing Interface (lower level)
    - MapReduce (higher level, more constrained)
- Parallelism is often difficult
  - Speedup is limited by serial portion of code and communication overhead