CpE 313: Microprocessor Systems Design

Handout 02
Performance Measurement

August 26, 2004

Shoukat Ali

shoukat@umr.edu

UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

Performance

- purchasing perspective
  - given a collection of machines, which has the
    - best performance ?
    - least cost ?
    - best performance / cost ?
- design perspective
  - faced with design options, which has the
    - best performance?
    - least cost ?
    - best performance / cost ?
- both require
  - basis for comparison
  - metric for evaluation
- our goal is to understand cost & performance implications of architectural choices

Two Notions of "Performance"

Plane	DC to Paris time	Speed	Passengers	Throughput pass./hour
Boeing 747	6.5 hours	610 mph	470	470/6.5 = 72.3
Concorde	3 hours	1350 mph	132	132/3 = 44

- which has higher performance?
- two notions of performance
  - time to do the task
    - execution time, response time, latency
  - tasks per day, hour, week, sec, ns. ..
    - throughput, bandwidth

Performance: Is It Throughput or Latency?

- latency of Concorde vs. Boeing 747?
  - Concord is 2.2 times better (2.2 = 6.5 hour / 3 hour)
- throughput of Concorde vs. Boeing 747 ?
  - Boeing is 1.6 times better
    (1.6 = 72.3 pass. per hour / 44 pass. per hour)
- Boeing is 1.6 times ("60%") faster in terms of throughput
- Concord is 2.2 times ("120%") faster in terms of flying time (or latency)
- we will focus primarily on latency or execution time for a single program
  - will also consider throughput

Definitions

- we always try to quote performance such that bigger is better
  - if you think that latency or execution time is more important than throughput, let performance = 1/execution time
  - if you think that throughput is more important than latency or execution time, let performance = throughput

- " X is n times faster than Y" means

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

Performance of a Computer

- performance of computers measured by
  -
  -
- which programs should we run on the computer to measure its performance?
  - the ones that the computer will have to run in real-life
  -
  - is workload for a computer likely to be known?
    - not for a general purpose computer!
  - even if known, is it going to be the same for other computers?
    - if not, how do I compare two computers?
- ANSWER: there should be a bunch of standard programs that should be run on to-be-compared computers
  - this set of standard programs called

Where Do I Get These Benchmarks?

- Transaction Processing Council (TPC)
  - TPC-A, TPC-B, TPC-C, TPC-H, TPC-R, TPC-W
- System Performance Evaluation Corporation (SPEC)
  - SPEC 92, SPEC 95, SPEC2002, SPECWeb99, ….
- NASA Benchmarks:
  - Scientific and Parallelizable Programs
- Ziff Davis (media company focused on the technology and game markets, owns PC Magazine, among others):
  - NetBench, ServerBench, WebBench,…..

- http://www.ideasinternational.com/ lists top 10 computers for a number of benchmarks, e.g., TPC, SPECxx, and many others

What Else Designers Use for Computer Evaluation?

- kernels
  - key pieces from
    - examples: Livermore Loops and Linpack
  - used to identify which part of computer (cache, pipeline, memory) is responsible for good or bad performance
- synthetic benchmarks
  - that are supposed to look like the real programs in terms of operations (floating point, integer) and memory accesses
  - examples: Whetstone, Dhrystone

SPEC CPU 2000 Benchmark Suite

- evolved from SPEC89, SPEC92, SPEC95
- designed to measure (as opposed to graphics)
  - for desktops
- (with inputs) reflecting a technical computing workload
  - eleven integer programs (called CINT2000)
  - fourteen floating-point intensive programs (called CFP2000)
  - see figure 1.12 in text
- SPEC insists that all programs be run with
  - ensures fair comparison among computers
- please see www.spec.org for more information

SPEC CPU 2000 Suite

Benchmark	Type	Source	Description
gzip	Integer	C	Compression using the Lempel-Ziv algorithm
vpr	Integer	C	FPGA circuit placement and routing
gcc	Integer	C	Consists of the GNU C compiler generating optimized machine code
mcf	Integer	C	Combinatorial optimization of public transit scheduling
crafty	Integer	C	Chess-playing program
parser	Integer	C	Syntactic English language parser
eon	Integer	C++	Graphics visualization using probabilistic ray tracing
perlmbk	Integer	C	Perl (an interpreted string-processing language) with four input scripts
gap	Integer	C	A group theory application package
vortex	Integer	C	An object-oriented database system
bzip2	Integer	C	A block-sorting compression algorithm
twolf	Integer	C	Timberwolf: a simulated annealing algorithm for VLSI place and route
wupwise	FP	F77	Lattice gauge theory model of quantum chromodynamics
swim	FP	F77	Solves shallow water equations using finite difference equations
mgrid	FP	F77	Multigrid solver over three-dimensional field
apply	FP	F77	Parabolic and elliptic partial differential equation solver
mesa	FP	C	Three-dimensional graphics library
galgel	FP	F90	Computational fluid dynamics
art	FP	C	Image recognition of a thermal image using neural networks
equake	FP	C	Simulation of seismic wave propagation
facerec	FP	C	Face recognition using wavelets and graph matching
ammp	FP	C	Molecular dynamics simulation of a protein in water
lucas	FP	F90	Performs primality testing for Mersenne primes
fma3d	FP	F90	Finite element modeling of crash simulation
sixtrack	FP	F77	High-energy physics accelerator design simulation
apsi	FP	F77	A meteorological simulation of pollution distribution

Other Benchmark Suites

- SPECapc
  - for measuring desktop
  - uses three graphics-intensive programs
- SPECWeb
  - for measuring performance of
  - web server task: provide web pages to multiple clients
  - perf. = pages served per second (a throughput measure)
- TPC-A, TPC-B, …
  - from Transaction Processing Council
  - transaction = database access + update
  - for measuring perf. of
    - airline reservation systems, ATMs
  - perf. = transactions/second (a throughput measure)

Reporting Performance Results

- list everything about your computer (HW + SW + compiler flags) that someone may need to know to repeat your performance calculation

Hardware		Software	
Model number	Precision WorkStation 410	O/S and version	Windows NT 4.0
CPU	700 MHz, Pentium III	Compilers and version	Intel C/C++ Compiler 4.5
Number of CPUs	1	Other software	See below
Primary cache	16KBI+16KBD on chip	File system type	NTFS
Secondary cache	256KB(I+D) on chip	System state	Default
Other cache	None		
Memory	256 MB ECC PC100 SDRAM		
Disk subsystem	SCSI		
Other hardware	None		

SPEC CINT2000 base tuning parameters/notes/summary of changes:

+FDO: PASS1=-Qprof_gen PASS2=-Qprof_use

Base tuning: -QxK -Qipo_wp shlW32M.lib +FDO

shlW32M.lib is the SmartHeap library V5.0 from MicroQuill www.microquill.com

Portability flags:

176.gcc: -Dalloca=_alloca /F10000000 -Op

186.crafy: -DNT_i386

253.perlbmk: -DSPEC_CPU2000_NTOS -DPERLDLL /MT

254.gap: -DSYS_HAS_CALLOC_PROTO -DSYS_HAS_MALLOC_PROTO

CPU Performance Equation

- two expressions for CPU time for a program

CPU Time = CPU Clock Cycles for a Program * Clock Cycle Time

CPU Time = CPU Clock Cycles for a Program / Clock Rate

- IC, instruction count = no. of assembly instructions in the program
  - depends on
    - the C program length
    - how you convert C into assembly
      - by hand or by compiler, which compiler?
    - instruction set (will discuss that more next week)
- CPI_{op} = no. of clock cycles needed to execute instruction “op”
  - CPI_{add} = no. of clock cycles needed to execute “add” instruction
- CPI = average clock cycles per instruction

CPI = no. of clock cycles for entire program / IC

Re-Writing CPU Performance Equation

CPU Time = IC * CPI * Clock Cycle Time

$$\text{CPU Time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instructions}} \times \frac{\text{Seconds}}{\text{Clock}}$$

More About CPI (Average Cycles/Instruction)

- how can we determine CPI, i.e., average cycles/instruction if given the cycles per instruction for all different instruction types?
- example: we have a particular processor with the following "instruction mix" for a given program

Oper	$Freq_{op}$	CPI_{op}
ALU	50%	1
Load	20%	5
Store	10%	3
Branch	20%	2

typical instruction mix

$$\text{CPI, average cycles/instruction} = \sum_{\text{all operations}} Freq_{op}\ CPI_{op}$$

$$= 0.5*1 + 0.2*5 + 0.1*3 + 0.2*2$$

$$= 2.2$$

CPI Example (continued)

Op	$Freq_{op}$	CPI_{op}	$Freq_{op}*CPI_{op}$	% time spent
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%
			(sum = 2.2)	

How much faster would the machine be if a better cache reduced the average load time to 2 cycles?

What if two ALU instructions could be executed at once?

CPI Example (continued)

Op	$Freq_{op}$	CPI_{op}	$Freq_{op}$*CPI_{op}	% time spent
ALU	50%	1	.5	
Load	20%			
Store	10%	3	.3	
Branch	20%	2	.4	
			(sum =)	

How much faster would the machine be if a better cache reduced the average load time to 2 cycles?

CPI Example (continued)

Op	$Freq_{op}$	CPI_{op}	$Freq_{op}$*CPI_{op}	% time spent
ALU	50%			
Load	20%	5	1.0	
Store	10%	3	.3	
Branch	20%	2	.4	
			(sum =)	

What if two ALU instructions could be executed at once?

Amdahl's Law

- determines the speedup that a proposed enhancement will bring
- one of the most fundamental laws of computer engineering
- let speedup(E) be the speedup due to enhancement E

$$\text{speedup(E)} = \frac{\text{excn time w/o E}}{\text{excn time with E}} = \frac{\text{perf with E}}{\text{perf w/o E}}$$

- suppose that enhancement E accelerates a fraction F of the task by a factor S and the remainder of the task is unaffected
  - F = fraction of the time that the enhancement can be used

$$\text{excn time with E} = ((1-F)+\frac{F}{S})\times\text{excn time w/o E}$$

$$\text{speedup(E)} = \frac{1}{(1-F)+\frac{F}{S}}$$

Note: F = fraction of the ***computation time*** in original machine

Corollary to Amdahl's Law

- even if S is infinite, speedup is no more than 1/(1-F)
- no use trying to make a fraction of the program too fast if that fraction is too small
- make the common case fast!