COMPUTER PERFORMANCE

CMSC 411 - 1 1

Definition: Performance

- Performance is in units of things per second
  - 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

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

CMSC 411 - 3 (from Patterson) 2

The bottom line: Performance

Car	Time to Boston	mph	passengers	Throughput(person, mile/hour)
Ferrari	3	150	2	100
Greyhound	7	65	60	560

- **Time to do the task**
  - *execution time*, response time, latency
- **Tasks per day, hour, week, sec, ns. ..**
  - *throughput*, bandwidth

Comparing performance of two machines

- **Definition: Performance is equal to 1 divided by execution time**
- **Problem: How to measure execution time?**

What is time?

- **Unix time command example:**
  - **90.7u 12.9s 2:39 65%**
  - **The user used the CPU for 90.7 seconds (user CPU time)**
  - **The system used it for 12.9 seconds (system CPU time)**
  - **Elapsed time from the user's request to completion of the task was 2 minutes, 39 seconds (159 seconds)**
  - **And (90.7 + 12.9)/159 = 65%**
    - » **the rest of the time was spent waiting for I/O or running other programs**

Time (cont.)

- **Usual measurements of time:**
  - **system performance measures the elapsed time on unloaded (single user) system**
  - **CPU performance measures user CPU time on unloaded system**

Relative Performance

$$\text{Relative Performance}(X/Y) = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n$$

- X is n times faster than Y
- X is n times as fast as Y
- From Y to X, speedup is n

CMSC 411 - 3 (from Patterson) 7

What is time?

CPU Execution Time = CPU clock cycles * Clock cycle time

- Every conventional processor has a clock with an associated clock cycle time or clock rate
- Every program runs in an integral number of clock cycles

Cycle Time

- MHz = millions of cycles/second, GHz = billions of cycles/second
- X MHz = 1000/X nanoseconds cycle time
- Y GHz = 1/Y nanoseconds cycle time

CMSC 411 - 3 (from Patterson) 8

How many clock cycles?

Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)

Example: Computer A runs program C in 3.6 billion cycles. Program C consists of 2 billion dynamic instructions. What is the CPI?

3.6 billion cycles / 2 billion instruction = 1.8 CPI

How many clock cycles?

Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)

Example: A computer is running a program with CPI = 2.0, and executes 24 million instructions, how long will it run?

24 millions instruction * 2 CPI = 48 millions cycles

Example

$$\text{CPU Execution Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

- **IC = 1 billion, 500 MHz processor, execution time of 3 seconds. What is the CPI for this program?**
- **3 s = 1b * CPI * 1 / 500M s CPI = 1.5**
- **Suppose we reduce CPI to 1.2 (through an architectural improvement). What is the new execution time?**
- **Exe Time = 1.2 * 1b * 1/500 s = 2.4 s**
- **Performance$_{new}$ / performance$_{old}$ = 3/2.4 = 1.25**

Who Affects Performance?

$$\text{CPU Execution Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

- Programmer
- Compiler
- instruction-set architect
- hardware designer
- materials scientist/physicist/silicon engineer

Performance Variation

CPU Execution Time = Instruction Count X CPI X Clock Cycle Time

	Number of instructions	CPI	Clock Cycle Time
Same machine different programs			
Same program, different machine, same ISA			
Same program, different machines			

Performance Variation

CPU Execution Time = Instruction Count X CPI X Clock Cycle Time

	Number of instructions	CPI	Clock Cycle Time
Same machine different programs	yes	yes	no
Same program, different machine, same ISA			
Same program, different machines			

Performance Variation

$$\text{CPU Execution Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

	Number of instructions	CPI	Clock Cycle Time
Same machine different programs	yes	yes	no
Same program, different machine, same ISA	no	yes	yes
Same program, different machines			

CMSC 411 - 3 (from Patterson) 15

Performance Variation

$$\text{CPU Execution Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

	Number of instructions	CPI	Clock Cycle Time
Same machine different programs	yes	yes	no
Same program, different machine, same ISA	no	yes	yes
Same program, different machines	yes	yes	yes

CMSC 411 - 3 (from Patterson) 16

Other Performance Metrics

- MIPS
  - Millions of Instructions Per Second = instruction count/ (exe time * 10^6)
  - Program-independent?
  - Deceptive

- FLOPS
  - FLoating-point Operations Per Second
  - How does execution time depend on FLOPS?

Which Programs

- Real applications

- SPEC (System Performance Evaluation Cooperative)
  - Provides a common set of real applications along with strict guidelines for how to run them.
  - Provides a relatively unbiased means to compare machines.

COMPUTER BENCHMARKS

How to measure CPU performance

- Benchmark: a program used to measure performance
  - real programs - what is reality?
  - kernels - loops in which most of time is spent in a real program
  - toy programs
  - synthetic programs
- Fact: Computer manufacturers tune their product to the popular benchmarks
  - ``Your results may vary," unless you run benchmark programs and nothing else
  - See Figure 1.13 listing programs in the SPEC CPU2006 benchmark suite

Performance: What to measure

- Usually rely on benchmarks vs. real workloads
- To increase predictability, collections of benchmark applications, called benchmark suites, are popular
- SPECCPU: popular desktop benchmark suite
  - CPU only, split between integer and floating point programs
  - SPEC2006 has 12 integer, 17 floating-point C/C++/Fortran programs
  - 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-W a transactional web benchmark
  - TPC-App application server and web services benchmark

CMSC 411 - 3 (from Patterson) 21

SPEC CPU 2006 (Integer)

Benchmark	Language	Application Area
• 400.perlbench	C	Programming Language
• 401.bzip2	C	Compression
• 403.gcc	C	C Compiler
• 429.mcf	C	Combinatorial Optimization
• 445.gobmk	C	Artificial Intelligence: Go
• 456.hmmer	C	Search Gene Sequence
• 458.sjeng	C	Artificial Intelligence: chess
• 462.libquantum	C	Physics / Quantum Computing
• 464.h264ref	C	Video Compression
• 471.omnetpp	C++	Discrete Event Simulation
• 473.astar	C++	Path-finding Algorithms
• 483.xalancbmk	C++	XML Processing

SPEC CPU 2006 (Floating Point)

Benchmark	Language	Application Area
• 410.bwaves	Fortran	Fluid Dynamics
• 416.gamess	Fortran	Quantum Chemistry
• 433.milc	C	Physics / QCD
• 434.zeusmp	Fortran	Physics / CFD
• 435.gromacs	C,Fortran	Molecular Dynamics
• 436.cactusADM	C,Fortran	Physics
• 437.leslie3d	Fortran	Fluid Dynamics
• 444.namd	C++	Biology / Molecular Dynamics
• 447.dealII	C++	Finite Element Analysis
• 450.soplex	C++	Linear Programming
• 453.povray	C++	Image processing / ray-tracing
• 454.calculix	C,Fortran	Structural mechanics
• 459.GemsFDTD	Fortran	Computational E&M
• 465.tonto	Fortran	Quantum Chemistry
• 470.lbm	C	Fluid Dynamics
• 481.wrf	C,Fortran	Weather simulation
• 482.sphinx3	C	Speech recognition

SPEC CPU Change Over Time (Int)

		Benchmark name by SPEC generation			
SPEC2006 benchmark description	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			

SPEC CPU Change Over Time (Float)

SPEC2006 benchmark description	SPEC2006	Benchmark name by SPEC generation SPEC2000	SPEC95	SPEC92	SPEC89
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			

Reproducibility

- **Benchmarking is a laboratory experiment, and needs to be documented as fully as a well-run chemistry experiment**
  - **Identify each variable hardware component**
  - **Identify compiler flags and measure variability**
  - **Verify reproducibility and provide data for others to reproduce the benchmarking results**

IMPROVING COMPUTER PERFORMANCE – AMDAHL' S LAW

CMSC 411 - 1 27

How to make computers faster

- Make the common case faster!
- Example: Put more effort and funds into optimizing the hardware for addition than to optimize square root
- Amdahl's law quantifies this principle:
  - Define speedup as the time the task took originally divided by the time the task takes after improvement

CMSC 411 - 2 28

Amdahl's Law

- Suppose that the original task runs for 1 second so it takes *f* seconds in the critical piece, and *1-f* in other things
- Then the task on the improved machine will take only *f/s* seconds in the critical piece, but will still take *1-f* seconds in other things
- speedup = $\frac{old\_time}{new\_time} = \frac{1}{(1-f)+\frac{f}{s}}$

Example 1

- Suppose we work very hard improving the square root hardware, and that our task originally spends *1%* of its time doing square roots. Even if the improvement reduces the square root time to zero, the speedup is no better than
- speedup = $\frac{1}{1-f} = \frac{1}{.99}$ = 1.01
- ***And we might be better off putting our effort into a more important part of the task***

Example 2

- Suppose that, for the same cost, we can speed up integer arithmetic by a factor of 20, or speed up floating point arithmetic by a factor of 2. If our task spends *10%* of its time in integer arithmetic, and *40%* of its time in floating point arithmetic, which should we do?

Example 2 (cont.)

- Option 1:
  speedup = $\dfrac{1}{.9 + \dfrac{.1}{20}}$ = 1.105

- Option 2:
  speedup = $\dfrac{1}{.6 + \dfrac{.4}{2}}$ = 1.25

CPU Performance

- More jargon …
- Something new happens in a computer during every clock cycle
- The clock rate is what manufacturers usually advertise to indicate a chip's speed:
  - e.g., a 2.8GHz Pentium 4
- But how fast the machine is depends on the clock rate *and* the number of clock cycles per instruction (CPI)

CMSC 411 - 2 33

CPU Performance (cont.)

- So total CPU time = instruction count (IC) times CPI divided by clock rate (MHz/GHz)
  - IC * CPI / GHz
- There's an example on p. 50 that illustrates the overall effect
- Note: CPI is not a good measure of performance all by itself since it varies by type of instruction!

CMSC 411 - 2 34

How to design a fast computer

- **Amdahl's law says put your effort into optimizing instructions that are often used.**
- **The locality of reference rule-of-thumb says that a program spends *90%* of its time in *10%* of the code, so we should put effort into taking advantage of this, through a memory hierarchy**
- **For data accesses, 2 types of locality**
  - **temporal**
  - **spatial**

Take advantage of parallelism

- **At digital design level**
  - **e.g., set associative caches (Chapter 2)**
- **At processor level**
  - **pipelining (Appendix C)**
  - **among instructions (Chapter 3)**
- **At system level**
  - **multiple cores (Chapter 4)**
  - **multiple CPUs (Chapter 5)**
  - **disks (Appendix D)**

RELIABILITY

CMSC 411 - 1 37

Define and quantify reliability (1/3)

- How decide when a system is operating properly?
- Infrastructure providers now offer Service Level Agreements (SLA) to guarantee that their networking or power service would be dependable
- Systems alternate between 2 states of service with respect to an SLA:

1. Service accomplishment, where the service is delivered as specified in SLA
2. Service interruption, where the delivered service is different from the SLA

- Failure = transition from state 1 to state 2
- Restoration = transition from state 2 to state 1

CMSC 411 - 3 (from Patterson) 38

Define and quantify reliability (2/3)

- *Module reliability* = measure of continuous service accomplishment (or time to failure).
  2 metrics

1. *Mean Time To Failure* (*MTTF*) measures Reliability
2. *Failures In Time* (*FIT*) = 1/MTTF, the rate of failures
   - Traditionally reported as failures per billion hours of operation

- *Mean Time To Repair* (*MTTR*) measures Service Interruption
  - *Mean Time Between Failures* (*MTBF*) = MTTF+MTTR
- *Module availability* measures service as alternate between the 2 states of accomplishment and interruption (number between 0 and 1, e.g. 0.9)
- *Module availability = MTTF / (MTTF + MTTR)*

Example calculating reliability

- If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure), overall failure rate is the sum of failure rates of the modules
- Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF), and 1 power supply (0.2M hour MTTF):

$$
\begin{aligned}
FailureRate &= 10 \times (1/1{,}000{,}000) + 1/500{,}000 + 1/200{,}000 \\
&= (10 + 2 + 5)/1{,}000{,}000 \\
&= 17/1{,}000{,}000 \\
&= 17{,}000\,FIT \\
MTTF &= 1{,}000{,}000{,}000/17{,}000 \\
&\approx 59{,}000\,hours
\end{aligned}
$$