

# Computer Performance

CIT 595  
Spring 2008

## Motivation for Examining Performance

- Hardware performance is often key to the effectiveness of an entire system of hardware and software
- Why certain piece of software performs the way it does?
- Why one instruction set can be implemented to perform better than another?

CIT 595

2

## What defines Performance?

- If you are running a program on 2 different workstations
  - Then faster one is the one that gets the job done first
  - As a user you are interested reducing the **response time**
    - > i.e. time from start to completion of task (a.k.a. execution time)
- If you are running a computer center with 2 timeshared computers running jobs submitted by many users
  - Then faster computer is the one that completed most jobs during the day
  - As system administrator you are interested increasing **throughput**
    - > i.e. the total amount of work done in a given time

CIT 595

3

## Measuring Response Time

- **Response Time** is one of the measure of computer performance
- So how do we measure the time?
  - We could just time from started our program till the time it was done executing (i.e. elapsed or response time)
  - However modern computers are often timeshared and may simultaneously work on several programs
    - So we want distinguish between the time the processor is working on our behalf vs. the time spent waiting

CIT 595

4

### Execution Time or Latency

- Response Time = CPU Execution Time + Wait Time
- CPU (processor) execution time *i.e.* time spent computing for a particular program
  - Not running other programs or waiting on I/O
  - Also called as *Latency*

CIT 595

5

### CPU Execution Time or Latency

- CPU execution time of program depends on:
  - Dependent on the total number of instructions executed
  - Each instruction performance is dependent on the number of clock cycles it takes to complete the instruction cycle (Fetch thru Store) called as *cycles per instruction (CPI)*
  - One clock cycle is time interval with clock ticks *i.e.* *clock cycle time*

$$\text{CPU Time} = \frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

↓ Clock Cycle time    ↓ CPI    ↓ Instruction Count

CIT 595

6

### CPU Execution Time (contd..)

- Clock cycle time = 1/Clock Rate or 1/Frequency
- Clock rate is given in Hertz (Hz) = cycles/sec

$$\text{CPU Time} = \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

CIT 595

7

### Cycles Per Instruction (CPI)

- CPI is an average of all the instructions executed in the program
- Depends on
  - Instruction Set Architecture (ISA)
  - Design details (micro-architecture)
    - CPI varies among different implementations of the same ISA
- CPI is obtained by performing a detailed simulation of an implementation

CIT 595

8

### Example: Calculating CPI

■ Given:

Instruction	# cycles	% Instr. Count
Load	5	25
Store	5	15
ALU	5	40
Branch/Jump	3	15
Others	6	5

The average CPI is  
 $= 0.25 \times 5 + 0.15 \times 5 + 0.40 \times 5 + 0.15 \times 3 + 0.05 \times 5$   
 $= 4.75$  cycles

### # Instructions & Clock Cycle Time

- Number of Instructions depends on:
  - Algorithm
  - Compiler
  - ISA
- Clock Cycle Time depends on:
  - Hardware Technology (transistor technology)
  - Design (Micro-architecture)
  - Simulate the longest propagation path of your circuit

### Maximizing Performance in relation to Execution Time

- One, way to maximize performance, we want to minimize execution time
- Thus we can relate performance and execution time for a machine X as:
 
$$\text{Performance}_x = 1 / \text{Execution Time}_x$$
- For low latency we want to minimize all 3 factors – CPI, # instructions and Clock cycle time
  - This is difficult as trying to decrease one often leads to increasing the other parameter
  - E.g. Two ISA Philosophies
    - > RISC – low CPI/clock period, high inst count
    - > CISC – low insn count, high CPI/clock period

### Performance Ratio (Comparing Performance)

- To claim that Machine X performs better than Machine Y:

$$\text{Performance}_x > \text{Performance}_y \text{ i.e.}$$

$$\frac{1}{\text{Execution Time}_x} > \frac{1}{\text{Execution Time}_y}$$

$$\text{Performance Ratio} = \frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{Execution Time}_y}{\text{Execution Time}_x}$$

## Relative Performance Example

	A	B
Clock Cycle Time (ns)	1	2
CPI	2.0	1.2

- 2 implementations of the same ISA
- Which machine (A or B) is faster and by how much?

Let "I" be the number instructions per program

Ex  $\text{Time}_A = 1 \times 2.0 \times I = 2 \times I \text{ ns}$

Ex  $\text{Time}_B = 2 \times 1.2 \times I = 2.4 \times I \text{ ns}$

$$\text{Performance Ratio} = \frac{\text{Time}_B}{\text{Time}_A} = \frac{2.4 \times I \text{ ns}}{2.0 \times I \text{ ns}} = 1.2$$

Machine A is 1.2 times faster than B

CIT 595

13

## Misleading Metrics

- Clock
  - Which is faster?
    - $\text{Clock}_A = 5 \text{ GHz}$ ,  $\text{Clock}_B = 3 \text{ GHz}$
  - Probably A, but make this assumption only if the same ISA & compiler
    - If stated that  $\text{CPI}_A = 2$  and  $\text{CPI}_B = 1$ , then which is faster?
- MIPS – Millions of instructions per second
  - $\text{instr/sec} = 1 / [(\text{cycles/instr}) * (\text{seconds/cycle})]$ 
    - Higher the MIPS, faster the machine
  - Different ISAs/Compilers have incomparable MIPS

CIT 595

14

## Maximizing Performance in relation to Throughput

- Throughput is amount of work that can be done in a given time
- Measured in tasks per time unit
  - E.g. x tasks per hour
- So higher the throughput, better the performance
- Also, *Throughput* and *Response Time* are inversely related
  - If system carries out a task in k seconds then its throughput is 1/k tasks per second
- Throughput can be increased:
  - Pipelining at instruction level (chp 5)
  - Timesharing a computer system (chp 8 - OS)
  - Achieving Parallelism (chp 9)
    - Superscalar Processor
    - Use multiprocessors (Shared Memory, Distributed, Chip Multiprocessors)

CIT 595

15

## Evaluating Performance

- Step I – Choose a workload
  - Workload: set of programs whose performance you care about
  - Use standardized workload (known as benchmarks)
- Step II – Decide the metric
  - Execution Time or Throughput
- Step III - Evaluate data gathered using statistical analysis
  - Statistical method use depends on
    - Metric
    - Distribution of the data

CIT 595

16

## Standard Performance Evaluation Corporation (SPEC)

- Is consortium that collects, distributes and standardizes benchmarks
- Produces benchmark suites for various classes of CPU, Java, I/O, Web, Multithreaded etc.
- E.g. CPU 2006 consists of 29 CPU intensive C/C++, Fortran programs
  - integer: perl, gcc,bzip2(compression),sjeng(AI: chess)
  - floating point: povray(ray tracing), wrf (weather prediction),sphynx3 (speech recognition)
- Like SPEC, there is Transaction Processing Council (TPC)
  - Used for web/database server workloads
  - Programs are I/O or network intensive rather than CPU intensive

CIT 595

17

## Statistical Analysis: Arithmetic Mean

$$\text{Arithmetic Mean} = \frac{\sum_{i=1}^n x_i}{n}$$

- Used for units that proportional to time
- The arithmetic mean can be misleading if the data are skewed or scattered

Program	System A Execution Time	System B Execution Time	System C Execution Time
v	50	100	500
w	200	400	600
x	250	500	500
y	400	800	800
z	5000	4100	3500
Average	1180	1180	1180

CIT 595

18

## Geometric Mean

- Represented as:

$$G = \left[ x_1 \cdot x_2 \cdot x_3 \cdot \dots \cdot x_n \right]^{\frac{1}{n}}$$

- Unlike an arithmetic mean, tends to dampen the effect of skew
- Used for unit less quantities e.g. performance ratio/speedup
- Performance results are stated in relation to the performance of a common machine used as reference

CIT 595

19

## Geometric Mean Example

Program	System A Execution Time	Execution Time Normalized to B	System B Execution Time	Execution Time Normalized to B	System C Execution Time	Execution Time Normalized to B
v	50	2	100	1	500	0.2
w	200	2	400	1	600	0.6667
x	250	2	500	1	500	1
y	400	2	800	1	800	1
z	5000	0.82	4100	1	3500	1.1714
Geometric Mean		1.6733		1		0.6898

Ratio = Ex (Machine Reference)/ Ex(Your Machine)

Geometric Mean for System A normalized with respect to System B:

$$= (100/50 \times 400/200 \times 500/250 \times 800/400 \times 4100/5000)^{1/5}$$

$$= 1.6733$$

CIT 595

20

## Geometric Mean Consistency

- The results that we got when using System B and System C as reference machines are given below
- We find that Geometric Mean of A to Geometric Mean of B to be consistent regardless which system is taken as reference
  - $1.6733/1 = 2.4258/1.4497$

System A Execution Time	Execution Time Normalized to B	System B Execution Time	Execution Time Normalized to B	System C Execution Time	Execution Time Normalized to B
Geometric Mean	1.6733		1		0.6898

  

System A Execution Time	Execution Time Normalized to C	System B Execution Time	Execution Time Normalized to C	System C Execution Time	Execution Time Normalized to C
Geometric Mean	2.4258		1.4497		1

CIT 595

21

## Harmonic Mean

- Used to units that inversely proportional to time i.e. throughput
- Unlike latencies, throughput cannot be added
  - E.g. 1<sup>st</sup> - 10 mile @ 30 mph, 2<sup>nd</sup> - 10 miles @ 40 mph and last 10 miles @ 60 mph
    - Average is not 43.3 mph
- Harmonic Mean

$$H = n \div (1/x_1 + 1/x_2 + 1/x_3 + \dots + 1/x_n)$$

CIT 595

22

## Reporting Performance Results

- Idea is that performance results must be reproducible
- E.g. SPEC reports
  - Complete description of machine (HW + SW)
    - Clock speed, Memory Organization, Operating System etc
  - Compiler flags
  - Choice of Inputs
  - Baseline and optimized results

CIT 595

23

## Performance Improvement & Amdahl's Law

- CPU Time can be improved by optimizing CPU
  - i.e. maximize speed and efficiency of operations
- But overall system performance is also dependent on I/O and memory
- Amdahl's law
  - Speedup limited by non-accelerated part
  - Speedup =  $1 / ((1-f) + f/k)$ 
    - f – fraction of worked performed by new component (enhancement)
    - k – speedup of the new component
  - Make the Common Case fast

CIT 595

24

## Programmer Tools

- On Unix, command called “time” provides the following:
  - First do: gcc myprog.c -o progname
  - Then do: time progname
  - 90.7u 12.9s 2:39 65%
    - > User CPU time is 90.7 u
    - > system CPU time is 12.9 s
    - > elapsed time is 159 s (2mins 39 sec)
    - > % of elapsed time that is CPU time
  - So more than 1/3 of elapsed time was spent waiting for I/O, running other programs or both

CIT 595

25

## Programmer Tools (contd..)

Drawbacks of UNIX *time* command

- Has poor resolution only milliseconds
- Sometimes one wants a higher precision
  - esp. if **performance** improvements are in the 1-2% range
- Times the whole code
  - Sometimes one is only interested in timing some part of the code
  - Sometimes one wants to compare the **execution time** of different sections of the code
    - > To figure out bottlenecks

CIT 595

26

## Programmer Tools (contd..)

- Profiler
  - Performance analysis tool that measures the behavior of a program as it runs
    - > Particularly the frequency and duration of function calls
  - The output is a statistical summary of the events observed (a **profile**)
  - E.g. GProf for C programs

CIT 595

27