Role of Performance

Introduction

- Performance dictates the effectiveness of an entire system, including hardware and software
- Performance measurement is one of the most important and difficult problems in computers
  - Consider the code to initialize a million integers using a loop vs using a system call
- Different aspects of performance may require different performance metrics
- Our goal for understanding performance
  - Effect of software on performance (see the above example)
  - Effect of instruction set architecture
  - Hardware features
- Defining performance
  - Needs and desires, buying a car
  - Response time
  - Execution time
  - Clock time is dependent on computer load, I/O wait, and OS overhead
  - Throughput
  - For our purpose,

    $$\text{Performance} = \frac{c}{\text{Execution time}}$$

    where c is a constant
  - For two machines, performance (p_i) and execution time (e_i) obey the relation

    $$\frac{p_i}{p_j} = \frac{e_j}{e_i} = n$$

    and we say that machine i is n times faster than machine j

Measuring performance

- Amount of work and amount of time
- Simplest time definition is the real clock time
  - System time, user time, I/O time, overhead
- *System performance* – Elapsed time on unloaded system
- CPU *performance* – CPU time
- *Clock cycles*
  - Constant time interval for the clock within the system
  - Dictates how fast a CPU can execute each instruction
- Clock rate
  - Inverse of clock cycle

  - 500 MHz
  - Clock cycle for 500 MHz is 2ns

Performance metrics

- CPU execution time is given by the product of CPU clock cycles for program and clock cycle time
- It can also be measured by

$$\frac{\text{CPU clock cycles for program}}{\text{Clock rate}}$$

- Improving performance
  - Current system
    * Execution time – 10 sec
    * Clock speed – 400 MHz
  - New system
    * Execution time – 6 sec
    * Clock speed – ?
    * Number of clock cycles – 1.2 times current system
  - Compute the number of clock cycles for current system
    *

$$\begin{aligned} \text{CPU time} &= \frac{\text{CPU clock cycles for program}}{\text{Clock rate}} \\ 10sec &= \frac{\text{CPU clock cycles for program}}{400 \times 10^6 \text{cps}} \end{aligned}$$

    * CPU clock cycles for program $= 4000 \times 10^6$
  - Compute the clock speed for new system
    *

$$\begin{aligned} \text{CPU time} &= \frac{\text{CPU clock cycles for program}}{\text{Clock rate}} \\ 6sec &= \frac{1.2 \times 4000 \times 10^6}{\text{Clock rate}} \end{aligned}$$

    *

$$\begin{aligned} \text{Clock rate} &= \frac{1.2 \times 4000 \times 10^6}{6} \\ &= 800 \times 10^6 \\ &= 800\text{MHz} \end{aligned}$$

- *Clock cycles per instruction*, or CPI
  - Average number of cycles for all instructions for the program being executed
  - CPU clock cycles is given by the product of number of instrcutions and CPI
- Using performance equation
  - Two implementations of the same ISA – machines M_a and M_b
  - M_a clock cycle time 1ns and CPI 2.0 for some code p
  - M_b clock cycle time 2ns and CPI 1.2 for p

- Identify faster machine
  - Let total clock cycles for the program on respective machines be c_a and c_b, and number of instructions be I
  - $$\begin{aligned} c_a &= T \times 2.0 \\ c_b &= T \times 1.2 \end{aligned}$$
  - CPU time t = CPU clock cycles $\times$ Clock cycle time
  - $t_a = I \times 2.0 \times 1 = 2I$ ns
  - $t_b = I \times 1.2 \times 2 = 2.4I$ ns
  - Machine m_a is faster; since performance is inversely proportional to time, the performance gain is given by
    $$\frac{t_b}{t_a} = \frac{2.4}{2} = 1.2$$

- Basic performance equation

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

or

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

- Measuring the performance factors
  - Measure CPU time by actually running the program
  - Clock cycle time is usually available as part of documentation
  - Instruction count and CPI are more difficult to obtain
  - Instruction count can be measured by using profiling tools, for example, `gprof(1)` in Unix

```
$ gcc -pg -o foobar foobar.c
$ foobar
$ gprof > foobar.profile
```

  - CPI can be obtained by detailed simulation of an implementation or by combining hardware counters and simulation
  - You may be able to compute CPU clock cycles by looking at different types of instructions and using their individual clock cycle counts
    $$\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times C_i)$$
    - C_i is the number of instructions of class i
    - CPI_i is the average number of cycles per instruction for class i
    - n is the number of instruction classes

- Comparing code segments – deciding ow to write efficient code for a given machine by selecting a set of instructions
  - Instruction classes

Instruction class	CPI
A	1
B	2
C	3

  - Instruction count for different code sequences

Code sequence	Number of instructions		
	A	B	C
c_1	2	1	2
c_2	4	1	1

- Find out the number of instructions for each code sequence, the faster code sequence, and CPI for each code sequence
  * Number of instructions in sequence $c_1 = 2 + 1 + 2 = 5$
  * Number of instructions in sequence $c_2 = 4 + 1 + 1 = 6$
  * Obviously, sequence c_1 executes fewer instructions
  * CPU clock cycles$_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10$
  * CPU clock cycles$_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9$
  * Code sequence c_2 is faster
  * $\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$
  * $\text{CPI}_1 = \frac{10}{5} = 2$
  * $\text{CPI}_2 = \frac{9}{6} = 1.5$

Benchmarks for performance evaluation

- *Workload*
  - Typical set of programs run in day-to-day work
  - Compare the execution time of workload on two computers to evaluate their relative performance
  - Not always feasible for real world
    * Too expensive (taking machines to prospective buyers' sites)
    * Proprietory issues (sending code and data to vendor sites)
- Benchmarks
  - Programs specifically chosen to simulate the actual workload performance
  - Selection of programs based on expected usage environment
  - Compiler optimization to beat benchmarks
    * Compiler may beat the benchmark but not guaranteed to produce correct working code at similar performance level
    * Code optimization to beat benchmark, especially if the benchmark is skewed towards some code
  - Benchmarks are used for
    * Easy coding and simulation
    * Simplicity
    * More easily standardized than large code
- Reproducibility
  - Most important component of a benchmark
  - Contains everything required to simulate a benchmark

Comparing and summarizing performance

- Summarizing implies loss of information but ease of understanding

- Should not cause confusion with contradictory but true statements
  - *Machine A is 10 times faster than machine B for program 1*
  - *Machine B is 10 times faster than machine A for program 2*

- Total execution time
  - Compare total execution time of a set of programs taken together
  - If P_i is performance of machine i and E_i is execution time of machine i, then,

$$\frac{P_a}{P_b} = \frac{E_b}{E_a} = \frac{1001}{110} = 9.1$$

- Average execution time
  - Computed over a number of small benchmarks
  - Arithmetic mean $AM = \frac{1}{n}\sum_{i=1}^{n} E_i$
  - Smaller mean implies smaller execution time

- Weighted average execution time
  - Applies a weight to each task such that sum of all weights w_i is 1
  - Weighted arithmetic mean $WAM = \frac{1}{n}\sum_{i=1}^{n} w_i \times E_i$, with the condition that $\sum_{i=1}^{n} w_i = 1$

SPEC95 Benchmark

- SPEC – *System Performance Evaluation Cooperative*
- Most comprehensive and popular set of CPU benchmarks
- 8 integer programs written in C and 10 floating point programs written in Fortran 77
- Separate time measurement for each set
  - Measurement normalized by dividing the execution time of a Sun SPARCstation 10/40 by the execution time on measured machine, yielding SPEC ratio
  - SPECint95 or SPECfp95 – Summary measurement by taking the geometric mean of the SPEC ratios
- For a given ISA, performance improvement comes from
  1. Increase in clock rate
  2. Improvements in processor organization to lower the CPI
  3. Compiler enhancements to lower the instruction count, or generate instructions with a lower average CPI
- In Figure 2.7, we see that Pentium Pro is 1.4 to 1.5 times faster on SPECint95 and 1.6 to 1.7 times faster on SPECfp95, *at the same clock rate*
- Increasing clock speed (Figure 2.8) does not increase the SPEC performance by the same level because of memory speed bottleneck

Fallacies and pitfalls

Pitfall 1 *Expecting the improvement of one aspect of a machine to increase performance by an amount proportional to the size of the improvement.*

- A program runs in 100 sec on a machine, with multiply operations taking up 80 seconds of this time. How much does the speed of multiplication need to improve to get a five-fold increase in code execution?

$$\begin{aligned}\text{Execution time after improvement} &= \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}\\ \frac{100}{5} &= \frac{80}{n} + (100 - 80)\\ 20 &= \frac{80}{n} + 20\\ 0 &= \frac{80}{n}\end{aligned}$$

There is no amount by which we can improve the performance of multiply to realize a five-fold increase in overall performance

  - This is *Amdahl's Law* in computing, or the law of diminishing returns in everyday life
  - Opportunity of improvement is affected by how many time the event occurs

- Common theme (Corollary of Amdahl's law) – make the common case fast

Fallacy 1 *Hardware-independent metrics predict performance.*

- Code size as a measure of speed
- ISA with smallest instruction set is the fastest

Pitfall 2 *Using MIPS as a performance metric.*

- MIPS $= \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$
- Intuitive, as more MIPS implies faster execution
- Problems
  1. MIPS does not account for capabilities of instructions
  2. A machine cannot have same MIPS rating for all programs
  3. MIPS can vary inversely with performance
- Consider the machine with three instruction classes and CPI measurements as follows:
  - Instruction classes

Instruction class	CPI
A	1
B	2
C	3

  - Instruction count (in billions of instructions for each class) for same program from two different compilers

Code from	Instruction count		
	A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

  - Machine clock rate – 500 MHz
  - Which code sequence executes faster according to MIPS? According to execution time?

- Solution
  - Find the execution time on each compiler using the equation

$$\text{Execution time} = \frac{\text{CPU clock cycles}}{\text{Clock rate}}$$

  - If C_i is the number of instructions of class i executed

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times C_i)$$

$$\begin{aligned}\text{CPU clock cycles}_1 &= (5 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 10 \times 10^9 \\ \text{CPU clock cycles}_2 &= (10 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 15 \times 10^9\end{aligned}$$

  - Execution time for two compilers

$$\begin{aligned}\text{Execution time}_1 &= \frac{10 \times 10^9}{500 \times 10^6} = 20s \\ \text{Execution time}_2 &= \frac{15 \times 10^9}{500 \times 10^6} = 30s\end{aligned}$$

  - MIPS rate

$$\begin{aligned}\text{MIPS} &= \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} \\ \text{MIPS}_1 &= \frac{(5 + 1 + 1) \times 10^9}{20 \times 10^6} = 350 \\ \text{MIPS}_2 &= \frac{(10 + 1 + 1) \times 10^9}{30 \times 10^6} = 400\end{aligned}$$

- Conclusion – Code from compiler 1 runs faster but code from compiler 2 has higher MIPS

Fallacy 2 *Synthetic benchmarks predict performance.*

- Goal to create a benchmark where execution frequency of a synthetic benchmark matches the characteristics of a large set of programs
- Most popular synthetic benchmarks – Whetstone and Dhrystone
- Whetstone – Measurement of Algol programs in a scientific/engineering environment (converted to Fortran)
- Dhrystone – Systems programming environments, originally in Ada and later converted to C

Pitfall 3 *Using arithmetic mean of normalized execution times to predict performance.*

- Normalized arithmetic mean is dependent on the machine used for normalization
- Better way is to use geometric mean given by

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

  where Execution time ratio$_i$ is the execution time, normalized to the reference machine, for the ith program of a total of n in the total workload

- Geometric mean is independent of the data series used for normalization because of the property

$$\frac{\text{Geometric mean}(X_i)}{\text{Geometric mean}(Y_i)} = \text{Geometric mean}\left(\frac{X_i}{Y_i}\right)$$

implying that mean of ratios, or ratio of means, is equal

Fallacy 3 *The geometric mean of execution time ratios is proportional to total execution time.*

- Geometric mean does not predict execution time