CPU Performance

Performance Assessment

- In evaluating processor hardware and setting requirements for new systems, performance is one of the key parameters to consider, along with cost, size, security, reliability, and, in some cases power consumption.
- Today's lecture focuses on the measuring of the performance of the computer machines

Performance Assessment

- How do we define performance?
  - response (or execution) time
    - the time between the start and the finish of a task
  - throughput
    - total amount of work done in a given time

➢ Almost just an inverse relationship, but throughput can be increased with *parallelism*

Cont.

- Response Time (latency)
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?
- Throughput
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done?

- *If we upgrade a machine with a new processor what do we increase?*
- *If we add a new machine to the lab what do we increase?*

Execution Time

- *Elapsed Time*
  - counts everything (*disk and memory accesses, waiting for I/O, running other programs, etc.*) from start to finish
  - a useful number, but often not good for comparison purposes

    elapsed time = CPU time + wait time (I/O, other programs, etc.)

- *CPU time*
  - doesn't count waiting for I/O or time spent running other programs
  - can be divided into *user CPU time* and *system CPU time* (OS calls)

    CPU time = user CPU time + system CPU time

    $\Rightarrow$ elapsed time = user CPU time + system CPU time + wait time

- Our focus: *user CPU time* (*CPU execution time* or, simply, *execution time*)
  - time spent executing the lines of code that are *in our program*

Cont.

- Factors that affect performance:
  - How well the program uses the instructions of the machine ?
  - How well the underlying hardware implements the instructions ?
  - How well the memory and I/O systems perform ?

Performance and Speed

- For some program running on machine X:

$$\text{Performance}_X = 1 / \text{Execution time}_X$$

- *X is n times faster than Y* means:

$$\text{Performance}_X \ / \ \text{Performance}_Y = n$$

Measuring Time

- Execution time is the amount of time it takes the program to execute in seconds.

Computer Clock Times

- Computers run according to a clock that runs at a steady rate
- The time interval is called a *clock cycle time* (e.g., 10ns).
- The *clock rate* is the reciprocal of clock cycle time - a frequency, how many cycles per sec (e.g., 100MHz).
  - 10 ns = 1/100,000,000 (clock time), same as:-
  - 1/10ns = 100,000,000 = 100MHz (clock rate).

Purchasing Decision

- Computer A has a 100MHz processor
- Computer B has a 300MHz processor
- So, B is faster, right?

- WRONG!

- Now, let's get it right.....

Measuring Performance

- The only important question: "HOW FAST WILL MY PROGRAM RUN?"
- CPU execution time for a program

  = CPU clock cycles * clock cycle time

  (= CPU clock cycles/Clock rate)
- In computer design, trade-off between:
  - clock cycle time, and
  - number of cycles required for a program

Performance Equation I

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

equivalently

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

- So, to improve performance one can either:
  - reduce the number of cycles for a program, or
  - reduce the clock cycle time (CPU improvement), or, equivalently,
  - increase the clock rate (CPU improvement)
- CPU improvement (Basic eq skip mem/IO optimization)
  - RISC machines try to reduce the number of cycles per instruction, and CISC machines try to reduce the number of instructions per program

Amdahl’ s Law

- The most dramatic improvement in system performance is realized when the performance of the most frequently used components is improved.
  - In short, our efforts at improving performance reap the greatest rewards by making the common case faster.
- Knowing whether a system or application is CPU bound→ our focus, memory bound, or I/O bound is the first step toward improving system performance.

Cycles Per Instruction

- The execution time of a program clearly must depend on the number of instructions
  - but different instructions take different times
- An expression that includes this is:-
  - CPU clock cycles = N * CPI
    - N = number of instructions
    - CPI = average clock cycles per instruction

Example

- Machine A
  - clock cycle time
    - 10ns/cycle
  - CPI = 0.5 for prog X

- Machine B
  - clock cycle time
    - 3ns/cycle (Approx.)
  - CPI = 2.0 for prog X

Let I = number of instructions in the program.

CPU clock cycles (A) = I * 0.5
CPU time (A) = CPU clock cycles * clock cycle time
= I * 0.5 * 10
= I * 5 ns

CPU clock cycles (B) = I * 2.0
CPU time (B) = CPU clock cycles * clock cycle time
= I * 2 * 3
= I * 6 ns

$$\frac{Performance(A)}{Performance(B)} = \frac{Execution(B)}{Execution(A)} = 1.2$$

Basic Performance Equation

- CPU Time = I * CPI * T
  - I = number of instructions in program
  - CPI = average cycles per instruction
  - T = clock cycle time
- CPU Time = I * CPI / R
  - R = 1/T the clock rate
    - T or R are usually published as performance measures for a processor

CPU Performance Calculation

$$\text{Performance} = \frac{1}{\text{Time}}$$

$$\text{Cycle Time (Sec)} = \frac{1}{\text{Clock Rate (Hz)}}$$

$$\text{Time} = (\#\text{Cycles}) \times (\text{Cycle Time in Sec})$$

$$\#\text{Cycles} = (\text{CPI}) \times (\text{Instruction count})$$

$$\therefore \text{Time} = (\text{CPI}) \times (\text{Instruction Count}) \times (\text{Cycle Time})$$

$$\text{CPI} = \sum \text{CPI}_i \times \text{Freq}_i = \frac{\sum \text{CPI}_i \times \text{Instruction Count}_i}{\text{Total Instruction Count}}$$

Examples

- Example 1:

  Machine A does a task in 20s, machine B does the same task in 25s.

  a) What is the performance of each machine?
     - Answer: PA = 1/20,PB = 1/25

  b) How much faster is A than B? (what is the speedup?)
     - Answer: 5/4

Examples

- Example 2:
  Machine A executes a program in 10s.
  a) If machine B is 1.3x faster than A, what is the execution time on machine B?
     - Answer: 1.3 = PB/PA = TA/TB: TB = 10/1.3
  b) If machine C is 1.5x slower than A, what is the execution time on machine C?
     - Answer: 1.5 = PA/PC = TC/TA: TC = 15

Examples

- Example 3: Machine A runs at 500 MHz. Machine B runs at 650 MHz. Program1 requires 100 x 10^6 clock cycles on machine A and 1.2 times that many on machine B. Which machine is faster? By how much?
- Answer:

  Exec(A) = $100 \times 10^6 / (500 \times 10^{6)} = .2$ seconds

  OR $100 \times 10^6 \times 2 \times 10^{-9} = 200 \times 10^{-3} = .2$ s

  Exec(B) = $120 \times 10^6 / (650 \times 10^6) = .18$ seconds

  Machine B is .2/.18 = 1.11 times faster than A

  Compare: 650/500 = 1.3 times clock rate

Examples

- Example 4: Assume 3 types of instructions:
  - Arithmetic (=,+,-,*,/) takes 4 cycles
  - Conditional (if) takes 3 cycles
  - I/O takes 5 cycles

Consider the following code segment:

```
cin >> num1;
cin >> num2;
num3 = num1 + num2;
if (num3 > 10)
 cout << "yes";
else
 cout << "no";
```

a) How many cycles to complete? (5+5+8+3+5=26 cycles)
b) What's the average number of cycles per instruction?(26/5 = 5.2 cycles)

Examples

- <u>CPI Calculation with Instruction Count:</u>
  Assume CPI = CPU Clock Cycles/Instruction Count
  then overall program CPU Clock Cycles = $\sum(CPI_i \times Count_i)$
  so that CPI = Overall Program Cycles/#Instructions

- <u>Example 5:</u> Assume Class A CPI=1, Class B CPI=2, Class C CPI=3 . Program requires 5 A, 3 B, 2 C instructions. What is the CPI?
  # of CPU Cycles = $5 \times 1 + 3 \times 2 + 2 \times 3 = 17$
  # of Instructions = $5 + 3 + 2 = 10$
  Therefore CPI = 17 cycles/10 instructions = 1.7 cycles/instruction

Examples

Execution Time = #Cycles × cycle time = (CPI × Instr. Count) × cycle time

= Instruction Count × CPI × cycle time

= (Instruction Count × CPI)/Clock Rate

Example 6: How long would it take to execute a program with 100×10^6 instructions if CPI is 3 and clock rate is 500 MHz?

Answer: Time = $100 \times 10^6 \times 3/(500 \times 10^6) = 3/5 = 0.6$ sec

Examples

- Example 7: Machine 1 and Machine 2 both have clock speeds of 500 MHz
  On Machine 1, program P requires 100×10^6 instructions & has a CPI of 2.5
  On Machine 2, program P requires 90×10^6 instructions & has a CPI of 3

  Which machine is faster? By how much?

Answer: T1 = 0.5 sec, T2 = 0.54 sec

Machine 1 is 1.08 times faster

Instruction Pipelining (1 of 7)

- Some CPUs divide the fetch-decode-execute cycle into smaller steps.
- These smaller steps can often be executed in parallel to increase throughput.
- Such parallel execution is called *instruction pipelining*.
- Instruction pipelining provides for *instruction level parallelism* (ILP)

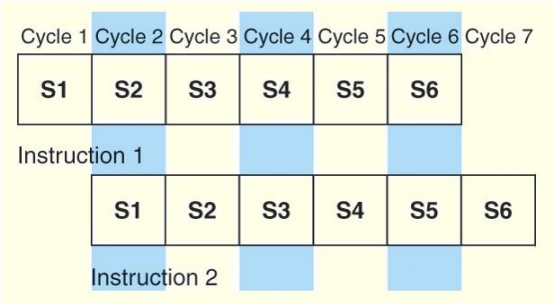
The next slide shows an example of instruction pipelining.

Instruction Pipelining (2 of 7)

- Suppose a fetch-decode-execute cycle were broken into the following smaller steps:
  1. Fetch instruction
  2. Decode opcode
  3. Calculate effective address of operands
  4. Fetch operands
  5. Execute instruction
  6. Store result
- Suppose we have a six-stage pipeline. S1 fetches the instruction, S2 decodes it, S3 determines the address of the operands, S4 fetches them, S5 executes the instruction, and S6 stores the result.

Instruction Pipelining (3 of 7)

- For every clock cycle, one small step is carried out, and the stages are overlapped.



S1. Fetch instruction.
S2. Decode opcode.
S3. Calculate effective address of operands.
S4. Fetch operands.
S5. Execute.
S6. Store result.

Instruction Pipelining (4 of 7)

- The theoretical speedup offered by a pipeline can be determined as follows:
  - Let t_p be the time per stage. Each instruction represents a task, T, in the pipeline.
  - The first task (instruction) requires $k \times t_p$ time to complete in a k-stage pipeline. The remaining $(n - 1)$ tasks emerge from the pipeline one per cycle. So the total time to complete the remaining tasks is $(n - 1)t_p$.
  - Thus, to complete n tasks using a k-stage pipeline requires:

$$(k \times t_p) + (n - 1)t_p = (k + n - 1)t_p.$$

Instruction Pipelining (5 of 7)

- If we take the time required to complete *n* tasks without a pipeline and divide it by the time it takes to complete n tasks using a pipeline, we find:

$$\text{Speedup } S = \frac{n t_n}{(k + n - 1)\, t_p}$$

- If we take the limit as n approaches infinity, $(k + n - 1)$ approaches n, which results in a theoretical speedup of:

$$\text{Speedup } S = \frac{k t_p}{t_p} = k$$

Instruction Pipelining (6 of 7)

- Our neat equations take a number of things for granted.
- First, we have to assume that the architecture supports fetching instructions and data in parallel.
- Second, we assume that the pipeline can be kept filled at all times. This is not always the case. Pipeline *hazards* arise that cause pipeline conflicts and stalls.

Instruction Pipelining (7 of 7)

- An instruction pipeline may stall, or be flushed for any of the following reasons:
  - Resource conflicts.
  - Data dependencies.
  - Conditional branching.
- Measures can be taken at the software level as well as at the hardware level to reduce the effects of these hazards, but they cannot be totally eliminated.

Thanks!