MEASURING COMPUTER PERFORMANCE

Necessity of evaluation computer performance

- For comparing different computer performances
- User: Interested in reducing the execution time (response time) of a task.
- Computer centre administrator: throughput
  - The computer user may say a computer is faster when a program runs in less time, while
  - The computer centre manager may say a computer is faster when it completes more jobs in an hour

TIME

- Time is the measure of computer performance
  - The computer that performs the same amount of work in the least time is the fastest.
  - Program execution time is measured in seconds per program.
- But, since with multiprogramming the CPU works on another program while waiting for I/O…
- CPU_time - the time CPU is computing (not including extra-time; e.g. time waiting for I/O or running other programs)

A computer faster than another?

- when a program runs in less time (will say a computer user)
  - the computer user is interested in reducing response time also referred to as execution time or latency.
- when computer completes more jobs in an hour (will say computer centre manager)
  - the computer centre manager is interested in increasing throughput - the total amount of work done in a given time - sometimes called bandwidth
USE accurate terms:

- **Response time**, **execution time** and **throughput** utilized for evaluate an entire computing work
  - **Response time** (execution time, latency): how long the computer responds to user input or finishes a program; the sooner the better
    - A user sees the result in 5 second after inputting the keyword to a library database system
    - A simulation program finishes in one hour
  - **Throughput**: Amount of work done in a given time. Throughput = how much work a computer can finish for a given time; the higher the better
    - A web server serves up to 5 million requests per second
    - Why throughput: a system runs multiple jobs simultaneously
  - **Bandwidth** and **latency** for memory performance

"A is faster than B"

- "A is n times faster than B" will mean
  \[
  n = \frac{\text{Performance}_B}{\text{Performance}_A} = \frac{1/\text{Execution Time}_B}{1/\text{Execution Time}_A} = \frac{\text{Performance}_A}{\text{Performance}_B}
  \]

**Example 1**

- If machine A runs a program in 10 seconds and machine B runs the same program in 15 seconds, which of the following statements is true?
  a. A is 50% faster than B
  b. A is 33% faster than B
Example 2

- Machine A is \( n\% \) faster than machine B can be expressed as:

\[
\frac{\text{Execution Time}_B}{\text{Execution Time}_A} = 1 + \frac{n}{100}
\]

\[
n = \left( \frac{\text{Execution time}_B - \text{Execution time}_A}{\text{Execution time}_A} \right) \times 100
\]

\[
\frac{15 - 10}{10} \times 100 = 50
\]

A is therefore 50\% faster than B.

Using Cycles to Compute CPU Time

Clock Cycle?

- Logic signal used to determine when “state” should be updated
- Ex: When does “Register” latch output of the adder?
  - It takes a certain amount of time for the values (11, 22) to propagate through the adder and the result appear at the Output.
  - The register cannot latch (store) the output of the adder before the value has been computed.

CPU time is the time a processor spends executing a piece of software.

CPU execution time can be computed as:

\[
\text{CPU}_{\text{time}} = N_r C_k P \times T_{\text{CLK}} = \frac{N_r C_k P}{f_{\text{CK}}}
\]

where:
- \( N_r C_k P \) = CPU clock cycles for a program
- \( T_{\text{CLK}} = \) Clock period (clock cycle time)
- \( f_{\text{CK}} = \) Clock frequency (clock rate)
- CPI = average clock cycles per instruction
- NrI = Number of instructions per program

Using Cycles to Compute CPU Time

CPU execution time can be computed as:

\[
\text{CPU}_{\text{time}} = N_r I \times CPI \times T_{\text{CLK}}
\]

Clock Cycles Per Instruction (CPI)

- CPI = average clock cycles per instruction

\[
\text{CPI} = \frac{N_r C_k P}{N_r I}
\]

- CPI is an average of all the instructions executed in a program
- CPI is useful in comparing two different implementations of the same architecture.
Example 3

- CPU clock frequency is 1 Megahertz (clock cycle time is $10^{-6}$ sec.)
- Program takes 4.5 million cycles to execute
- What’s the CPU time?

- $4,500,000 \times 10^{-6} = 4.5$ seconds

Example 4

- CPU clock frequency is 500 MHz
- Program takes 45 million cycles to execute
- What’s the CPU time?

- $45,000,000 \times \frac{1}{500,000,000} = 0.09$ seconds

Influences on CPU Performance

\[ CPU_{time} = NrI \times CPI \times T_{CLK} \]

<table>
<thead>
<tr>
<th>Inst Count (NrI)</th>
<th>CPI</th>
<th>Clock Rate ($f_{CLK}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>√</td>
<td>(√)</td>
</tr>
<tr>
<td>Instr. Set</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Organisation</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>

NrI ⇒ Program type
⇒ Compiler quality
⇒ ISA (low no. of complex instructions)

CPI ⇒ ISA (high no. of simple instructions)
⇒ Compiler quality
⇒ organisation (parallel execution, hardwired control, memory bandwidth)

$T_{CLK} ⇒$ hardware technology
⇒ organisation
Cycles Per Instruction

- CPI = “Average Clock Cycles per Instruction”
  \[ CPI = \frac{N_r C_k_p}{N_r I} \]

- but we can write
  \[ N_r C_k_p = \sum_{i=1}^{n} (CPI_i \times I_i) \]

- where \( I_i \) represents number of times instruction \( i \) is executed in a program and \( CPI_i \) represents the number of clock cycles for instruction \( i \).

Cycles Per Instruction

- This form can be used to express CPU time as
  \[ CPU_{time} = T_{CLK} \times \sum_{i=1}^{n} CPI_i \times I_i \]

- Instruction Frequency” - what is the percentage of type \( i \) instructions from the total number of instructions per program?
  \[ F_i = \frac{I_i}{N_r I} \]

Overall CPI

- The later form of the CPI calculation multiplies each individual \( CPI_i \) by the fraction of occurrence in a program
  \[ CPI = \frac{N_r C_k_p}{N_r I} = \frac{\sum_{i=1}^{n} (CPI_i \times I_i)}{N_r I} = \sum_{i=1}^{n} \left( \frac{CPI_i \times I_i}{N_r I} \right) \]

\[ CPI = \sum_{i=1}^{n} CPI_i \times F_i \]

⇒ Invest Resources where time is Spent!

Example 5

- A benchmark has 100 instructions:
  - 25 instructions are loads/stores (each takes 2 cycles)
  - 50 instructions are adds (each takes 1 cycle)
  - 25 instructions are square root (each takes 100 cycles)

\[ CPI = \frac{(25 \times 2) + (50 \times 1) + (25 \times 100)}{100} = \frac{2600}{100} = 26.0 \]
**Example 6: Calculating CPI**

<table>
<thead>
<tr>
<th>OP</th>
<th>Freq</th>
<th>CPI&lt;sub&gt;i&lt;/sub&gt;</th>
<th>CPI&lt;sub&gt;i&lt;/sub&gt; * F&lt;sub&gt;i&lt;/sub&gt; (%)time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>0.5 (33%)</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>2</td>
<td>0.4 (27%)</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>2</td>
<td>0.2 (13%)</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>0.4 (27%)</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>1.5</td>
<td></td>
</tr>
</tbody>
</table>

**MIPS**

\[
MIPS = \frac{NI}{CPU_{times} * 10^6} = \frac{NI}{CPU_{times} * CPI * T_{CLK} * 10^6} = \frac{f_{CLK}}{CPI * 10^6}
\]

**Good news**
- The rightmost form is very convenient since clock rate is fixed for a machine and CPI is usually a small number.

**Bad news**
- MIPS is dependent on the instruction set, making it difficult to compare MIPS of computers with different instruction sets.
- MIPS varies between programs on the same computer; and most importantly,
- MIPS can vary inversely to performance.

**Example: MIPS as a meaningless indicator of performance**

- Machine A has a special instruction for performing square root calculations
  - It takes 100 cycles to execute
- Machine B doesn’t have the special instruction -- must perform square root calculations in software using simple instructions (e.g., Add, Mult, Shift) that each take 1 cycle to execute
- Only for square root instructions, if \(f_{CLK} = 1\) MHz:
  - Machine A: 1/100 MIPS = 0.01 MIPS
  - Machine B: 1 MIPS

**MFLOPS**

\[
MFLOPS = \frac{NI \_of \_Floating \_Point \_Operations \_in \_a \_Program}{Execution \_time \times 10^6}
\]

- Since MFLOPS were intended to measure floating-point performance, they are not applicable outside the range.
- Based on operations rather than instructions, MFLOPS is intended to be a fair comparison between different machines.
**MFLOPS criticism**

- MFLOPS is not dependable because the set of floating point operations is not consistent across machines.
- MFLOPS rating is dependent on the machine (FP instructions implemented).
  - Some machines have no divide instruction, or trigonometric functions.
- MFLOPS rating is dependent on the program.
  - For example, a program with 100% floating-points adds will have a higher rating than a program with 100% floating-point divides.

**BENCHMARKS**

Programs to Evaluate Performance

- Benchmarks can focus on specific aspects of a system.
  - Floating point & integer ALU
  - Memory system
  - I/O
  - Operating System

**BENCHMARKS - Real applications**

- Patterson & Hennessy describe five types of benchmark programs, listed below in decreasing order of accuracy of prediction:
  - Real applications
  - Modified (or scripted) applications
  - Kernels
  - Toy benchmarks
  - Synthetic benchmarks

- Examples are compilers for C, text-processing software like Word, and other applications like Photoshop.
- Real applications have input, output, and options that a user can select when running the program.
- Real applications often encounter **portability problems** arising from **dependences on the operating system or compiler**.
  - Enhancing portability often means modifying the source and sometimes eliminating some important activity, such as interactive graphics, which tends to be more system dependent.
**BENCHMARKS - Modified applications**

- Real applications are used as the *building blocks for a benchmark*, either with modifications to the application or with a *script that acts as stimulus* to the application.
- Applications are modified for one of two primary reasons: *to enhance portability* or *to focus on one particular aspect of system performance*.
  - For example, to create a *CPU-oriented benchmark*, I/O may be removed or restructured to minimize its impact on execution time.

**BENCHMARKS - Kernels**

- **Kernels** are constructed by *extracting small, key pieces from real programs*.
- Unlike real programs, no user would run kernel programs. They exist solely to evaluate performance.
- Kernels are best used to isolate performance of individual features of a machine to explain the reasons for differences in performance of real programs.

**BENCHMARKS - Synthetic benchmarks**

- Similar in philosophy to kernels, synthetic benchmarks try to match the average frequency of operations and operands of a large set of programs.
- Synthetic benchmarks are even further removed from reality than kernels because kernel code is extracted from real programs, while synthetic code is created artificially to match an average execution profile.
- The most popular synthetic benchmarks:
  - Whetstone (floating point operations)
  - Dhrystone (fixed point operations)

**BENCHMARKS - Toy benchmarks**

- Toy benchmarks are typically between 10 and 100 lines of code and produce a result the user already knows before running the toy program.
- Examples: Sieve of Eratosthenes, Puzzle, Quicksort.
**SPEC - the Standard Performance Evaluation Corporation**

- SPEC is a non-profit corporation formed to "establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers"
- http://www.spec.org
- SPEC develops suites of benchmarks intended to measure computer performance
  - These suites are packaged with source code and tools and are extensively tested for portability before release
  - They are available to the public for a fee covering development and administrative costs
  - By license agreement, SPEC members and customers agree to run and report results as specified in each benchmark suite's documentation.

**SPEC - Benchmark Suite**

- Main categories of benchmark suites:
  - Desktop benchmarks: CPU, memory, and graphics performance
  - Server benchmarks: throughput-oriented, I/O and OS intensive
  - Embedded benchmarks: measuring the ability to meet deadline and save power

**Amdahl’s Law**

- The performance gain that can be obtained by improving some portion of a computer can be calculated using Amdahl’s Law
- Amdahl’s Law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used
- Amdahl's Law defines the speedup that can be gained by using a particular feature

\[
\text{Speedup} = \frac{\text{Performance}_\text{old}}{\text{Performance}_\text{new}}
\]

\[
\text{Speedup} = \frac{\text{Entire}_\text{old}}{\text{Entire}_\text{new}} \times \frac{\text{Performance}_\text{new}}{\text{Performance}_\text{old}}
\]
Amdahl’s Law

### Example 1
- Program runs for 100 seconds on a uniprocessor
- 10% of the program can be parallelized on a multiprocessor (F=0.1)
- Assume a multiprocessor with 10 processors (n=10)

\[
\text{Speedup} = \frac{1}{(1 - 0.1) + \frac{0.1}{10}} = \frac{1}{0.91} = 1.1
\]

### Example 2
- Floating point instructions improved to run 2X; but only 10% of actual instructions are FP

\[
\begin{align*}
\text{ExTime}_{new} &= \text{ExTime}_{old} \times \left(0.9 + \frac{0.1}{2}\right) = 0.95 \times \text{ExTime}_{old} \\
\text{Speedup}_{overall} &= \frac{1}{0.95} = 1.053
\end{align*}
\]

Amdahl’s Law – Example 1

- Program runs for 100 seconds on a uniprocessor
- 10% of the program can be parallelized on a multiprocessor (F=0.1)
- Assume a multiprocessor with 10 processors (n=10)

\[
\text{Speedup} = \frac{1}{(1 - 0.1) + \frac{0.1}{10}} = \frac{1}{0.91} = 1.1
\]

Amdahl’s Law – Example 2

- Floating point instructions improved to run 2X; but only 10% of actual instructions are FP

\[
\begin{align*}
\text{ExTime}_{new} &= \text{ExTime}_{old} \times \left(0.9 + \frac{0.1}{2}\right) = 0.95 \times \text{ExTime}_{old} \\
\text{Speedup}_{overall} &= \frac{1}{0.95} = 1.053
\end{align*}
\]
Amdahl’s Law – Example 3

• Suppose we could improve the speed of the CPU in our machine by a factor of five (without affecting I/O performance) for five times the cost. Also assume that the CPU is used 50% of the time, and the rest of the time the CPU is waiting for I/O. If the CPU is one-third of the total cost of the computer, is increasing the CPU speed by a factor of five a good investment from the cost / performance viewpoint?

• Answer: The speedup obtained is:

\[
\text{speedup} = \frac{1}{0.5 + \frac{0.5}{5}} = \frac{1}{0.6} = 1.67
\]

• The new machine will cost 2.33 times the original machine:

\[
\frac{2}{3} \times 1 + \frac{1}{3} \times 5 = 2.33
\]

• Since the cost increase is larger than the performance improvement, this change does not improve cost / performance