Computer Performance Evaluation: Cycles Per Instruction (CPI)

• Most computers run synchronously utilizing a CPU clock running at a constant clock rate:
  
  where: Clock rate = 1 / clock cycle

• A computer machine instruction is comprised of a number of elementary or micro operations which vary in number and complexity depending on the instruction and the exact CPU organization and implementation.
  – A micro operation is an elementary hardware operation that can be performed during one CPU clock cycle.
  – This corresponds to one micro-instruction in microprogrammed CPUs.
  – Examples: register operations: shift, load, clear, increment, ALU operations: add, subtract, etc.

• Thus a single machine instruction may take one or more cycles to complete termed as the Cycles Per Instruction (CPI).
CPU Machine Instruction Execution Steps

- **Instruction Fetch**: Obtain instruction from program storage
- **Instruction Decode**: Determine required actions and instruction size
- **Operand Fetch**: Locate and obtain operand data
- **Execute**: Compute result value or status
- **Result Store**: Deposit results in storage for later use
- **Next Instruction**: Determine successor or next instruction
Computer Performance Measures: Program Execution Time

- For a specific program compiled to run on a specific machine (CPU) "A", the following parameters are provided:
  - The total executed instruction count of the program.
  - The average number of cycles per instruction (average CPI).
  - Clock cycle of machine "A"

- How can one measure the performance of this machine running this program?
  - Intuitively the machine (or CPU) is said to be faster or has better performance running this program if the total execution time is shorter.
  - Thus the inverse of the total measured program execution time is a possible performance measure or metric:

\[
\text{Performance}_A = \frac{1}{\text{Execution Time}_A}
\]

How to compare performance of different machines?
What factors affect performance? How to improve performance?
Comparing Computer Performance Using Execution Time

• To compare the performance of two machines (or CPUs) “A”, “B” running a given specific program:

\[
\text{Performance}_A = \frac{1}{\text{Execution Time}_A} \\
\text{Performance}_B = \frac{1}{\text{Execution Time}_B}
\]

• Machine A is \( n \) times faster than machine B means:

\[
\text{Speedup} = n = \frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution Time}_B}{\text{Execution Time}_A}
\]

• Example:

For a given program:

Execution time on machine A: \( \text{Execution}_A = 1 \) second
Execution time on machine B: \( \text{Execution}_B = 10 \) seconds

\[
\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution Time}_B}{\text{Execution Time}_A} = 10 / 1 = 10
\]

The performance of machine A is 10 times the performance of machine B when running this program, or: Machine A is said to be 10 times faster than machine B when running this program.
CPU Execution Time: The CPU Equation

• A program is comprised of a number of instructions executed, \( I \)
  – Measured in: instructions/program

• The average instruction takes a number of cycles per instruction (CPI) to be completed.
  – Measured in: cycles/instruction, CPI

• CPU has a fixed clock cycle time \( C = 1/clock \text{ rate} \)
  – Measured in: seconds/cycle

• CPU execution time is the product of the above three parameters as follows:

\[
T = I \times CPI \times C
\]
CPU Execution Time

For a given program and machine (CPU):

CPI = Total program execution cycles / Instructions count

→ CPU clock cycles = Instruction count x CPI

CPU execution time =

= CPU clock cycles x Clock cycle
= Instruction count x CPI x Clock cycle
= I x CPI x C
CPU Execution Time: Example

• A Program is running on a specific machine (CPU) with the following parameters:
  – Total executed instruction count: 10,000,000 instructions
  – Average CPI for the program: 2.5 cycles/instruction.
  – CPU clock rate: 200 MHz.

• What is the execution time for this program:

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

\[
= 10,000,000 \times 2.5 \times \frac{1}{\text{clock rate}}
\]

\[
= 10,000,000 \times 2.5 \times 5 \times 10^{-9}
\]

\[
= .125 \text{ seconds}
\]
Factors Affecting CPU Performance

\[
\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

\[
T = I \times \text{CPI} \times C
\]

<table>
<thead>
<tr>
<th></th>
<th>Instruction Count</th>
<th>Cycles per Instruction</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruction Set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture (ISA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Aspects of CPU Execution Time

CPU Time = Instruction count × CPI × Clock cycle

Depends on:
- Program Used
- Compiler
- ISA

Instruction Count I

Depends on:
- CPU Organization
- Technology

CPI

Clock Cycle C

Depends on:
- Program Used
- Compiler
- ISA
- CPU Organization

EECC550 - Shaaban
Performance Comparison: Example

• From the previous example: A Program is running on a specific machine (CPU) with the following parameters:
  – Total executed instruction count: 10,000,000 instructions
  – Average CPI for the program: 2.5 cycles/instruction.
  – CPU clock rate: 200 MHz.

• Using the same program with these changes:
  – A new compiler used: New executed instruction count: 9,500,000
    New CPI: 3.0
  – Faster CPU implementation: New clock rate = 300 MHz

• What is the speedup with the changes?

\[
\text{Speedup} = \frac{\text{Old Execution Time} = I_{\text{old}} \times \text{CPI}_{\text{old}} \times \text{Clock cycle}_{\text{old}}}{\text{New Execution Time} = I_{\text{new}} \times \text{CPI}_{\text{new}} \times \text{Clock cycle}_{\text{new}}}
\]

\[
\text{Speedup} = \frac{(10,000,000 \times 2.5 \times 5 \times 10^{-9})}{(9,500,000 \times 3 \times 3.33 \times 10^{-9})} = \frac{.125}{.095} = 1.32
\]

or 32% faster after changes.
Instruction Types & CPI

• Given a program with \( n \) types or classes of instructions with the following characteristics:

\[ C_i = \text{Count of instructions of type}_i \]
\[ CPI_i = \text{Cycles per instruction for type}_i \]

Then:

\[ CPI = \frac{\text{CPU Clock Cycles}}{\text{Instruction Count}} \]

Where:

\[ CPU \text{ clock cycles} = \sum_{i=1}^{n} (CPI_i \times C_i) \]
\[ \text{Instruction Count} = \sum C_i \]
Instruction Types & CPI: An Example

• An instruction set has three instruction classes:

<table>
<thead>
<tr>
<th>Instruction class</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
</tbody>
</table>

• Two code sequences have the following instruction counts:

<table>
<thead>
<tr>
<th>Code Sequence</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• CPU cycles for sequence 1 = 2 x 1 + 1 x 2 + 2 x 3 = 10 cycles
  CPI for sequence 1 = clock cycles / instruction count
  = 10 /5 = 2

• CPU cycles for sequence 2 = 4 x 1 + 1 x 2 + 1 x 3 = 9 cycles
  CPI for sequence 2 = 9 / 6 = 1.5
Instruction Frequency & CPI

• Given a program with \( n \) types or classes of instructions with the following characteristics:

\[
C_i = \text{Count of instructions of type } i \\
CPI_i = \text{Average cycles per instruction of type } i \\
F_i = \text{Frequency of instruction type } i \\
= \frac{C_i}{\text{total instruction count}}
\]

Then:

\[
CPI = \sum_{i=1}^{n} \left( CPI_i \times F_i \right)
\]

Fraction of total execution time for instructions of type \( i \) = \( \frac{CPI_i \times F_i}{CPI} \)
### Instruction Type Frequency & CPI: A RISC Example

#### Base Machine (Reg / Reg)

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq, $F_i$</th>
<th>CPI$_i$</th>
<th>CPI$_i$ x $F_i$</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>.5</td>
<td>23%</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td>1.0</td>
<td>45%</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td>.3</td>
<td>14%</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
<td>18%</td>
</tr>
</tbody>
</table>

**Typical Mix**

$$CPI = \sum_{i=1}^{n} (CPI_i \times F_i)$$

$$CPI = .5 \times 1 + .2 \times 5 + .1 \times 3 + .2 \times 2 = 2.2$$
Metrics of Computer Performance

Each metric has a purpose, and each can be misused.

- Cycles per second (clock rate).
- Megabytes per second.
- Execution time: Target workload, SPEC95, etc.

(millions) of Instructions per second – MIPS
(millions) of (F.P.) operations per second – MFLOP/s

Application
Programming Language
Compiler
ISA
Datapath
Control
Function Units
Transistors Wires Pins
Choosing Programs To Evaluate Performance

Levels of programs or benchmarks that could be used to evaluate performance:

- **Actual Target Workload:** Full applications that run on the target machine.

- **Real Full Program-based Benchmarks:**
  - Select a specific mix or suite of programs that are typical of targeted applications or workload (e.g. SPEC95, SPEC CPU2000).

- **Small “Kernel” Benchmarks:**
  - Key computationally-intensive pieces extracted from real programs.
    - Examples: Matrix factorization, FFT, tree search, etc.
  - Best used to test specific aspects of the machine.

- **Microbenchmarks:**
  - Small, specially written programs to isolate a specific aspect of performance characteristics: Processing: integer, floating point, local memory, input/output, etc.
### Types of Benchmarks

#### Pros
- Representative
- Portable.
- Widely used.
- Measurements useful in reality.
- Easy to run, early in the design cycle.
- Identify peak performance and potential bottlenecks.

#### Cons
- Very specific.
- Non-portable.
- Complex: Difficult to run, or measure.
- Less representative than actual workload.
- Easy to “fool” by designing hardware to run them well.
- Peak performance results may be a long way from real application performance.

### Benchmarks
- **Actual Target Workload**
- **Full Application Benchmarks**
- **Small “Kernel” Benchmarks**
- **Microbenchmarks**
SPEC: System Performance Evaluation Cooperative

The most popular and industry-standard set of CPU benchmarks.

- **SPECmarks, 1989:**
  - 10 programs yielding a single number (“SPECmarks”).

- **SPEC92, 1992:**
  - SPECInt92 (6 integer programs) and SPECfp92 (14 floating point programs).

- **SPEC95, 1995:**
  - SPECInt95 (8 integer programs):
    - go, m88ksim, gcc, compress, li, ijpeg, perl, vortex
  - SPECfp95 (10 floating-point intensive programs):
    - tomcatv, swim, su2cor, hydro2d, mgrid, applu, turb3d, apsi, fppp, wave5
  - Performance relative to a Sun SuperSpark I (50 MHz) which is given a score of SPECInt95 = SPECfp95 = 1

- **SPEC CPU2000, 1999:**
  - CINT2000 (11 integer programs). CFP2000 (14 floating-point intensive programs)
  - Performance relative to a Sun Ultra5_10 (300 MHz) which is given a score of SPECInt2000 = SPECfp2000 = 100
## SPEC95 Programs

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>Artificial intelligence; plays the game of Go</td>
</tr>
<tr>
<td>m88ksim</td>
<td>Motorola 88k chip simulator; runs test program</td>
</tr>
<tr>
<td>gcc</td>
<td>The Gnu C compiler generating SPARC code</td>
</tr>
<tr>
<td>compress</td>
<td>Compresses and decompresses file in memory</td>
</tr>
<tr>
<td>li</td>
<td>Lisp interpreter</td>
</tr>
<tr>
<td>ijpeg</td>
<td>Graphic compression and decompression</td>
</tr>
<tr>
<td>perl</td>
<td>Manipulates strings and prime numbers in the special-purpose programming language Perl</td>
</tr>
<tr>
<td>vortex</td>
<td>A database program</td>
</tr>
<tr>
<td>tomcatv</td>
<td>A mesh generation program</td>
</tr>
<tr>
<td>swim</td>
<td>Shallow water model with 513 x 513 grid</td>
</tr>
<tr>
<td>su2cor</td>
<td>Quantum physics: Monte Carlo simulation</td>
</tr>
<tr>
<td>hydro2d</td>
<td>Astrophysics: Hydrodynamic Naiver Stokes equations</td>
</tr>
<tr>
<td>mgrid</td>
<td>Multigrid solver in 3-D potential field</td>
</tr>
<tr>
<td>applu</td>
<td>Parabolic/elliptic partial differential equations</td>
</tr>
<tr>
<td>trub3d</td>
<td>Simulates isotropic, homogeneous turbulence in a cube</td>
</tr>
<tr>
<td>apsi</td>
<td>Solves problems regarding temperature, wind velocity, and distribution of pollutant</td>
</tr>
<tr>
<td>fpppp</td>
<td>Quantum chemistry</td>
</tr>
<tr>
<td>wave5</td>
<td>Plasma physics; electromagnetic particle simulation</td>
</tr>
</tbody>
</table>

### Integer

### Floating Point
Sample SPECint95 Results

Source URL: http://www.macinfo.de/bench/specmark.html
Sample SPECfp95 Results

Source URL: http://www.macinfo.de/bench/specmark.html
## SPEC CPU2000 Programs

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Language</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>164.gzip</td>
<td>C</td>
<td>Compression</td>
</tr>
<tr>
<td>175.vpr</td>
<td>C</td>
<td>FPGA Circuit Placement and Routing</td>
</tr>
<tr>
<td>176.gcc</td>
<td>C</td>
<td>C Programming Language Compiler</td>
</tr>
<tr>
<td>181.mcf</td>
<td>C</td>
<td>Combinatorial Optimization</td>
</tr>
<tr>
<td>186.crafty</td>
<td>C</td>
<td>Game Playing: Chess</td>
</tr>
<tr>
<td>197.parser</td>
<td>C</td>
<td>Word Processing</td>
</tr>
<tr>
<td>252.eon</td>
<td>C++</td>
<td>Computer Visualization</td>
</tr>
<tr>
<td>253.perlbmk</td>
<td>C</td>
<td>PERL Programming Language</td>
</tr>
<tr>
<td>254.gap</td>
<td>C</td>
<td>Group Theory, Interpreter</td>
</tr>
<tr>
<td>255.vortex</td>
<td>C</td>
<td>Object-oriented Database</td>
</tr>
<tr>
<td>256.bzip2</td>
<td>C</td>
<td>Compression</td>
</tr>
<tr>
<td>300.twolf</td>
<td>C</td>
<td>Place and Route Simulator</td>
</tr>
<tr>
<td>168.wupwise</td>
<td>Fortran 77</td>
<td>Physics / Quantum Chromodynamics</td>
</tr>
<tr>
<td>171.swim</td>
<td>Fortran 77</td>
<td>Shallow Water Modeling</td>
</tr>
<tr>
<td>172.mgrid</td>
<td>Fortran 77</td>
<td>Multi-grid Solver: 3D Potential Field</td>
</tr>
<tr>
<td>173.applu</td>
<td>Fortran 77</td>
<td>Parabolic / Elliptic Partial Differential Equations</td>
</tr>
<tr>
<td>177.mesa</td>
<td>C</td>
<td>3-D Graphics Library</td>
</tr>
<tr>
<td>178.galgel</td>
<td>Fortran 90</td>
<td>Computational Fluid Dynamics</td>
</tr>
<tr>
<td>179.art</td>
<td>C</td>
<td>Image Recognition / Neural Networks</td>
</tr>
<tr>
<td>183.equake</td>
<td>C</td>
<td>Seismic Wave Propagation Simulation</td>
</tr>
<tr>
<td>187.facerec</td>
<td>Fortran 90</td>
<td>Image Processing: Face Recognition</td>
</tr>
<tr>
<td>188.ammp</td>
<td>C</td>
<td>Computational Chemistry</td>
</tr>
<tr>
<td>189.lucas</td>
<td>Fortran 90</td>
<td>Number Theory / Primality Testing</td>
</tr>
<tr>
<td>191.fma3d</td>
<td>Fortran 90</td>
<td>Finite-element Crash Simulation</td>
</tr>
<tr>
<td>200.sixtrack</td>
<td>Fortran 77</td>
<td>High Energy Nuclear Physics Accelerator Design</td>
</tr>
<tr>
<td>301.apsi</td>
<td>Fortran 77</td>
<td>Meteorology: Pollutant Distribution</td>
</tr>
</tbody>
</table>

## Top 20 SPEC CPU2000 Results (As of March 2002)

### Top 20 SPECint2000

<table>
<thead>
<tr>
<th>#</th>
<th>MHz</th>
<th>Processor</th>
<th>int peak</th>
<th>int base</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1300</td>
<td>POWER4</td>
<td>814</td>
<td>790</td>
</tr>
<tr>
<td>2</td>
<td>2200</td>
<td>Pentium 4</td>
<td>811</td>
<td>790</td>
</tr>
<tr>
<td>3</td>
<td>2200</td>
<td>Pentium 4 Xeon</td>
<td>810</td>
<td>788</td>
</tr>
<tr>
<td>4</td>
<td>1667</td>
<td>Athlon XP</td>
<td>724</td>
<td>697</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>Alpha 21264C</td>
<td>679</td>
<td>621</td>
</tr>
<tr>
<td>6</td>
<td>1400</td>
<td>Pentium III</td>
<td>664</td>
<td>648</td>
</tr>
<tr>
<td>7</td>
<td>1050</td>
<td>UltraSPARC-III Cu</td>
<td>610</td>
<td>537</td>
</tr>
<tr>
<td>8</td>
<td>1533</td>
<td>Athlon MP</td>
<td>609</td>
<td>587</td>
</tr>
<tr>
<td>9</td>
<td>750</td>
<td>PA-RISC 8700</td>
<td>604</td>
<td>568</td>
</tr>
<tr>
<td>10</td>
<td>833</td>
<td>Alpha 21264B</td>
<td>571</td>
<td>497</td>
</tr>
<tr>
<td>11</td>
<td>1400</td>
<td>Athlon</td>
<td>554</td>
<td>495</td>
</tr>
<tr>
<td>12</td>
<td>833</td>
<td>Alpha 21264A</td>
<td>533</td>
<td>511</td>
</tr>
<tr>
<td>13</td>
<td>600</td>
<td>MIPS R14000</td>
<td>500</td>
<td>483</td>
</tr>
<tr>
<td>14</td>
<td>675</td>
<td>SPARC64 GP</td>
<td>478</td>
<td>449</td>
</tr>
<tr>
<td>15</td>
<td>900</td>
<td>UltraSPARC-III</td>
<td>467</td>
<td>438</td>
</tr>
<tr>
<td>16</td>
<td>552</td>
<td>PA-RISC 8600</td>
<td>441</td>
<td>417</td>
</tr>
<tr>
<td>17</td>
<td>750</td>
<td>POWER RS64-IV</td>
<td>439</td>
<td>409</td>
</tr>
<tr>
<td>18</td>
<td>700</td>
<td>Pentium III Xeon</td>
<td>438</td>
<td>431</td>
</tr>
<tr>
<td>19</td>
<td>800</td>
<td>Itanium</td>
<td>365</td>
<td>358</td>
</tr>
<tr>
<td>20</td>
<td>400</td>
<td>MIPS R12000</td>
<td>353</td>
<td>328</td>
</tr>
</tbody>
</table>

### Top 20 SPECfp2000

<table>
<thead>
<tr>
<th>MHz</th>
<th>Processor</th>
<th>fp peak</th>
<th>fp base</th>
</tr>
</thead>
<tbody>
<tr>
<td>1300</td>
<td>POWER4</td>
<td>1169</td>
<td>1098</td>
</tr>
<tr>
<td>1000</td>
<td>Alpha 21264C</td>
<td>960</td>
<td>776</td>
</tr>
<tr>
<td>1050</td>
<td>UltraSPARC-III Cu</td>
<td>827</td>
<td>701</td>
</tr>
<tr>
<td>2200</td>
<td>Pentium 4 Xeon</td>
<td>802</td>
<td>779</td>
</tr>
<tr>
<td>2200</td>
<td>Pentium 4</td>
<td>801</td>
<td>779</td>
</tr>
<tr>
<td>833</td>
<td>Alpha 21264B</td>
<td>784</td>
<td>643</td>
</tr>
<tr>
<td>800</td>
<td>Itanium</td>
<td>701</td>
<td>701</td>
</tr>
<tr>
<td>833</td>
<td>Alpha 21264A</td>
<td>644</td>
<td>571</td>
</tr>
<tr>
<td>1667</td>
<td>Athlon XP</td>
<td>642</td>
<td>596</td>
</tr>
<tr>
<td>750</td>
<td>PA-RISC 8700</td>
<td>581</td>
<td>526</td>
</tr>
<tr>
<td>750</td>
<td>PA-RISC 8700</td>
<td>581</td>
<td>526</td>
</tr>
<tr>
<td>1533</td>
<td>Athlon MP</td>
<td>547</td>
<td>504</td>
</tr>
<tr>
<td>600</td>
<td>MIPS R14000</td>
<td>529</td>
<td>499</td>
</tr>
<tr>
<td>675</td>
<td>SPARC64 GP</td>
<td>509</td>
<td>371</td>
</tr>
<tr>
<td>900</td>
<td>UltraSPARC-III</td>
<td>482</td>
<td>427</td>
</tr>
<tr>
<td>1400</td>
<td>Athlon</td>
<td>458</td>
<td>426</td>
</tr>
<tr>
<td>1400</td>
<td>Pentium III</td>
<td>456</td>
<td>437</td>
</tr>
<tr>
<td>500</td>
<td>PA-RISC 8600</td>
<td>440</td>
<td>397</td>
</tr>
<tr>
<td>450</td>
<td>POWER3-II</td>
<td>433</td>
<td>426</td>
</tr>
<tr>
<td>500</td>
<td>Alpha 21264</td>
<td>422</td>
<td>383</td>
</tr>
<tr>
<td>400</td>
<td>MIPS R12000</td>
<td>407</td>
<td>382</td>
</tr>
</tbody>
</table>

Source: [http://www.aceshardware.com/SPECmine/top.jsp](http://www.aceshardware.com/SPECmine/top.jsp)
Computer Performance Measures: MIPS (Million Instructions Per Second)

- For a specific program running on a specific computer, MIPS is a measure of how many millions of instructions are executed per second:

  \[ \text{MIPS} = \frac{\text{Instruction count}}{(\text{Execution Time} \times 10^6)} \]
  \[ = \frac{\text{Instruction count}}{(\text{CPU clocks} \times \text{Cycle time} \times 10^6)} \]
  \[ = \frac{(\text{Instruction count} \times \text{Clock rate})}{(\text{Instruction count} \times \text{CPI} \times 10^6)} \]
  \[ = \frac{\text{Clock rate}}{(\text{CPI} \times 10^6)} \]

- Problems with MIPS rating:
  - No account for the instruction set used.
  - Program-dependent: A single machine does not have a single MIPS rating since the MIPS rating may depend on the program used.
  - Easy to abuse: Program used to get the MIPS rating is often omitted.
  - Cannot be used to compare computers with different instruction sets.
  - A higher MIPS rating in many cases may not mean higher performance or better execution time. i.e. due to compiler design variations.
Compiler Variations, MIPS & Performance:
An Example

• For a machine with instruction classes:

<table>
<thead>
<tr>
<th>Instruction class</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
</tbody>
</table>

• For a given program, two compilers produced the following instruction counts:

<table>
<thead>
<tr>
<th>Instruction counts (in millions) for each instruction class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code from:</td>
</tr>
<tr>
<td>Compiler 1</td>
</tr>
<tr>
<td>Compiler 2</td>
</tr>
</tbody>
</table>

• The machine is assumed to run at a clock rate of 100 MHz.
Compiler Variations, MIPS & Performance: 
An Example (Continued)

MIPS =  \frac{\text{Clock rate}}{\text{CPI} \times 10^6} = \frac{100 \text{ MHz}}{\text{CPI} \times 10^6}

\text{CPI} = \frac{\text{CPU execution cycles}}{\text{Instructions count}}

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

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

• For compiler 1:
  – \text{CPI}_1 = \frac{(5 \times 1 + 1 \times 2 + 1 \times 3)}{(5 + 1 + 1)} = \frac{10}{7} = 1.43
  – \text{MIPS}_1 = \frac{100}{(1.428 \times 10^6)} = 70.0
  – \text{CPU time}_1 = \frac{((5 + 1 + 1) \times 10^6 \times 1.43)}{(100 \times 10^6)} = 0.10 \text{ seconds}

• For compiler 2:
  – \text{CPI}_2 = \frac{(10 \times 1 + 1 \times 2 + 1 \times 3)}{(10 + 1 + 1)} = \frac{15}{12} = 1.25
  – \text{MIPS}_2 = \frac{100}{(1.25 \times 10^6)} = 80.0
  – \text{CPU time}_2 = \frac{((10 + 1 + 1) \times 10^6 \times 1.25)}{(100 \times 10^6)} = 0.15 \text{ seconds}
Computer Performance Measures:
MFLOPS (Million FLOating-Point Operations Per Second)

- A floating-point operation is an addition, subtraction, multiplication, or division operation applied to numbers represented by a single or a double precision floating-point representation.
- MFLOPS, for a specific program running on a specific computer, is a measure of millions of floating point-operation (megaflops) per second:
  \[
  \text{MFLOPS} = \frac{\text{Number of floating-point operations}}{\text{(Execution time} \times 10^6)}
  \]
- MFLOPS is a better comparison measure between different machines than the MIPS rating.
- Program-dependent: Different programs have different percentages of floating-point operations present. i.e compilers have no floating-point operations and yield a MFLOPS rating of zero.
- Dependent on the type of floating-point operations present in the program.
Performance Enhancement Calculations: Amdahl's Law

- The performance enhancement possible due to a given design improvement is limited by the amount that the improved feature is used.

- Amdahl’s Law:

  Performance improvement or speedup due to enhancement E:

  \[
  \text{Speedup}(E) = \frac{\text{Execution Time without } E}{\text{Execution Time with } E} = \frac{\text{Performance with } E}{\text{Performance without } E}
  \]

  Suppose that enhancement E accelerates a fraction \( F \) of the execution time by a factor \( S \) and the remainder of the time is unaffected then:

  \[
  \text{Execution Time with } E = ((1-F) + F/S) \times \text{Execution Time without } E
  \]

  Hence speedup is given by:

  \[
  \text{Speedup}(E) = \frac{1}{((1-F) + F/S) \times \text{Execution Time without } E} = \frac{1}{(1 - F) + F/S}
  \]
Pictorial Depiction of Amdahl’s Law

Enhancement E accelerates fraction F of execution time by a factor of S

Before:
Execution Time without enhancement E:

<table>
<thead>
<tr>
<th>Unaffected, fraction: (1 - F)</th>
<th>Affected fraction: F</th>
</tr>
</thead>
</table>

Unchanged

| Unaffected, fraction: (1 - F) | F/S |

After:
Execution Time with enhancement E:

\[
\text{Speedup}(E) = \frac{\text{Execution Time without enhancement E}}{\text{Execution Time with enhancement E}} = \frac{1}{(1 - F) + \frac{F}{S}}
\]
Performance Enhancement Example

For the RISC machine with the following instruction mix given earlier:

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>CPI(i)</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>.5</td>
<td>23%</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td>1.0</td>
<td>45%</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td>.3</td>
<td>14%</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
<td>18%</td>
</tr>
</tbody>
</table>

CPI = 2.2

If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement:

Fraction enhanced = F = 45% or .45
Unaffected fraction = 100% - 45% = 55% or .55
Factor of enhancement = 5/2 = 2.5

Using Amdahl’s Law:

\[
\text{Speedup}(E) = \frac{1}{(1 - F) + \frac{F}{S}} = \frac{1}{.55 + \frac{.45}{2.5}} = 1.37
\]
An Alternative Solution Using CPU Equation

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>CPI(i)</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>.5</td>
<td>23%</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td>1.0</td>
<td>45%</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td>.3</td>
<td>14%</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
<td>18%</td>
</tr>
</tbody>
</table>

CPI = 2.2

If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement:

Old CPI = 2.2

New CPI = \(0.5 \times 1 + 0.2 \times 2 + 0.1 \times 3 + 0.2 \times 2 = 1.6\)

\[
\text{Speedup}(E) = \frac{\text{Original Execution Time}}{\text{New Execution Time}} = \frac{\text{Instruction count} \times \text{old CPI} \times \text{clock cycle}}{\text{Instruction count} \times \text{new CPI} \times \text{clock cycle}}
\]

\[
\frac{\text{old CPI}}{\text{new CPI}} = \frac{2.2}{1.6} = 1.37
\]

Which is the same speedup obtained from Amdahl’s Law in the first solution.
Performance Enhancement Example

• A program runs in 100 seconds on a machine with multiply operations responsible for 80 seconds of this time. By how much must the speed of multiplication be improved to make the program four times faster?

\[
\text{Desired speedup} = 4 = \frac{100}{\text{Execution Time with enhancement}}
\]

→ Execution time with enhancement = 25 seconds

\[
25 \text{ seconds} = (100 - 80 \text{ seconds}) + \frac{80 \text{ seconds}}{n}
\]

\[
25 \text{ seconds} = 20 \text{ seconds} + \frac{80 \text{ seconds}}{n}
\]

→ \( \frac{5}{n} = 80 \text{ seconds} \)

→ \( n = \frac{80}{5} = 16 \)

Hence multiplication should be 16 times faster to get a speedup of 4.
Performance Enhancement Example

- For the previous example with a program running in 100 seconds on a machine with multiply operations responsible for 80 seconds of this time. By how much must the speed of multiplication be improved to make the program five times faster?

\[
\text{Desired speedup} = 5 = \frac{100}{\text{Execution Time with enhancement}}
\]

\[\Rightarrow \text{Execution time with enhancement} = 20 \text{ seconds}\]

\[20 \text{ seconds} = (100 - 80 \text{ seconds}) + \frac{80 \text{ seconds}}{n}\]

\[20 \text{ seconds} = 20 \text{ seconds} + \frac{80 \text{ seconds}}{n}\]

\[\Rightarrow 0 = 80 \text{ seconds} / n\]

No amount of multiplication speed improvement can achieve this.
Extending Amdahl's Law To Multiple Enhancements

- Suppose that enhancement $E_i$ accelerates a fraction $F_i$ of the execution time by a factor $S_i$ and the remainder of the time is unaffected then:

$$Speedup = \frac{\text{Original Execution Time}}{\left( (1 - \sum_i F_i) + \sum_i \frac{F_i}{S_i} \right) \times \text{Original Execution Time}}$$

$$Speedup = \frac{1}{\left( (1 - \sum_i F_i) + \sum_i \frac{F_i}{S_i} \right)}$$

Note: All fractions refer to original execution time.
Amdahl's Law With Multiple Enhancements: Example

• Three CPU performance enhancements are proposed with the following speedups and percentage of the code execution time affected:

<table>
<thead>
<tr>
<th>Speedup</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1 = 10$</td>
<td>$F_1 = 20%$</td>
</tr>
<tr>
<td>$S_2 = 15$</td>
<td>$F_2 = 15%$</td>
</tr>
<tr>
<td>$S_3 = 30$</td>
<td>$F_3 = 10%$</td>
</tr>
</tbody>
</table>

• While all three enhancements are in place in the new design, each enhancement affects a different portion of the code and only one enhancement can be used at a time.

• What is the resulting overall speedup?

$$Speedup = \frac{1}{\left(1 - \sum_i F_i \right) + \sum_i \frac{F_i}{S_i}}$$

• Speedup

$$= 1 / \left[(1 - .2 - .15 - .1) + .2/10 + .15/15 + .1/30\right]$$

$$= 1 / \left[ .55 + .0333 \right]$$

$$= 1 / .5833 = 1.71$$
Pictorial Depiction of Example

Before:
Execution Time with no enhancements: 1

Unaffected, fraction: .55

F1 = .2

F2 = .15

F3 = .1

/ 10 / 15 / 30

Unchanged

After:
Execution Time with enhancements: .55 + .02 + .01 + .00333 = .5833

Speedup = 1 / .5833 = 1.71

Note: All fractions refer to original execution time.