Computer Performance Evaluation: Cycles Per Instruction (CPI)

- Most computers run synchronously utilizing a CPU clock running at a constant clock rate:
  
  \[ \text{Clock rate} = \frac{1}{\text{clock cycle}} \]

- 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 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).
Computer Performance Measures: Program Execution Time

• For a specific program compiled to run on a specific machine “A”, the following parameters are provided:
  – The total 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 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 “A”, “B” running a given 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:
  \[
  n = \frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution Time}_B}{\text{Execution Time}_A}
  \]

- Example:
  For a given program:
  \[
  \text{Execution time on machine A: } \quad \text{Execution}_A = 1 \text{ second} \\
  \text{Execution time on machine B: } \quad \text{Execution}_B = 10 \text{ seconds} \\
  \text{Performance}_A / \text{Performance}_B = \frac{\text{Execution Time}_B}{\text{Execution Time}_A} = \frac{10}{1} = 10
  \]

  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
  - Measured in: instructions/program

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

- CPU has a fixed clock cycle time = 1/clock rate
  - Measured in: seconds/cycle

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

\[
T = \frac{I \times CPI \times C}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]
CPU Execution Time

For a given program and machine:

\[
CPI = \frac{\text{Total program execution cycles}}{\text{Instructions count}}
\]

\[
\rightarrow \quad \text{CPU clock cycles} = \text{Instruction count} \times CPI
\]

CPU execution time =

\[
= \text{CPU clock cycles} \times \text{Clock cycle}
\]

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

\[
= I \times CPI \times C
\]
CPU Execution Time: Example

- A Program is running on a specific machine with the following parameters:
  - Total 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:

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

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

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

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

<table>
<thead>
<tr>
<th>CPU time</th>
<th>Seconds</th>
<th>= Instructions x Cycles x Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Equation:**

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

**Table:**

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

CPU Time = Instruction count \times CPI \times Clock cycle

- Depends on:
  - Program Used
  - Compiler
  - ISA

- Instruction Count

- CPI

- Clock Cycle

- Depends on:
  - CPU Organization
  - Technology
Performance Comparison: Example

• From the previous example: A Program is running on a specific machine with the following parameters:
  – Total 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 instruction count 9,500,000
    New CPI: 3.0
  – Faster CPU implementation: New clock rate = 300 MHZ
• What is the speedup with the changes?

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

or 32% faster after changes.
Instruction Types & CPI

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

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

Then:

\[
\text{CPU clock cycles} = \sum_{i=1}^{n} (CPI_i \times C_i)
\]
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>Instruction counts for instruction class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</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:

\[
CPI = \sum_{i=1}^{n} (CPI_i \times F_i)
\]
Instruction Type Frequency & CPI: A RISC Example

Base Machine (Reg / Reg)

<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>

Typical Mix

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

CPI = .5 x 1 + .2 x 5 + .1 x 3 + .2 x 2 = 2.2
Metrics of Computer Performance

Application
Programming Language
Compiler

Execution time: Target workload, SPEC95, etc.

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

Datapath
Control
Function Units
Transistors Wires Pins

Megabytes per second.
Cycles per second (clock rate).

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

– 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.

**Actual Target Workload**

**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.

**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:**
  - Eighteen new application benchmarks selected (with given inputs) reflecting a technical computing workload.
  - 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
  - Source code must be compiled with standard compiler flags.
## SPEC95

<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>fpopp</td>
<td>Quantum chemistry</td>
</tr>
<tr>
<td>wave5</td>
<td>Plasma physics; electromagnetic particle simulation</td>
</tr>
</tbody>
</table>
Sample SPECint95 Results

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

Source URL:  http://www.macinfo.de/bench/specmark.html
SPEC95 For High-End CPUs First Quarter 2000
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:
  
  \[
  \text{MIPS} = \frac{\text{Instruction count}}{(\text{Execution Time} \times 10^6)}
  \]
  
  \[
  = \frac{\text{Instruction count}}{(\text{CPU clocks} \times \text{Cycle time} \times 10^6)}
  \]
  
  \[
  = \frac{(\text{Instruction count} \times \text{Clock rate})}{(\text{Instruction count} \times \text{CPI} \times 10^6)}
  \]
  
  \[
  = \frac{\text{Clock rate}}{(\text{CPI} \times 10^6)}
  \]

• Faster execution time usually means faster MIPS rating.

• 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 some 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)

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

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

\[
\text{CPU clock cycles} = \sum_{i=1}^{n} \left( CPI_i \times C_i \right)
\]

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

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

• For compiler 2:
  – \(\text{CPI}_2 = \frac{(10 \times 1 + 1 \times 2 + 1 \times 3)}{(10 + 1 + 1)} = \frac{15}{12} = 1.25\)
  – \(\text{MIP}_2 = \frac{100}{(1.25 \times 10^6)} = 80.0\)
  – \(\text{CPU time}_2 = \frac{((10 + 1 + 1) \times 10^6 \times 1.25)}{(100 \times 10^6)} = 0.15 \text{ seconds}\)
Computer Performance Measures:
**MFOLPS** (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 MIPS.
- 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 $E$:

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

  - 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:

- Unaffected, fraction: \((1 - F)\)
- Affected fraction: \(F\)

After:
Execution Time with enhancement E:

- Unaffected, fraction: \((1 - F)\)
- Affected fraction: \(\frac{F}{S}\)

Unchanged

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

\[
\text{Fraction enhanced} = F = 45\% \quad \text{or} \quad .45
\]

\[
\text{Unaffected fraction} = 100\% - 45\% = 55\% \quad \text{or} \quad .55
\]

\[
\text{Factor of enhancement} = \frac{5}{2} = 2.5
\]

Using Amdahl’s Law:

\[
\text{Speedup(E)} = \frac{1}{(1 - F) + \frac{F}{S}} = \frac{1}{.55 + .45/2.5} = 1.37
\]
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?

\[
\begin{align*}
\text{Desired speedup} &= 4 \\
\text{Execution Time with enhancement} &= \frac{100}{4} \\
\text{Execution time with enhancement} &= 25 \text{ seconds}
\end{align*}
\]

\[
\begin{align*}
25 \text{ seconds} &= (100 - 80 \text{ seconds}) + \frac{80 \text{ seconds}}{n} \\
25 \text{ seconds} &= 20 \text{ seconds} + \frac{80 \text{ seconds}}{n}
\end{align*}
\]

\[
\begin{align*}
5 &= \frac{80 \text{ seconds}}{n} \\
n &= \frac{80}{5} = 16
\end{align*}
\]

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?

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

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

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

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

\[
0 = \frac{80 \text{ seconds}}{n}
\]

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:

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

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

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:
  
  \[
  \begin{align*}
  \text{Speedup}_1 &= S_1 = 10 & \text{Percentage}_1 &= F_1 = 20\% \\
  \text{Speedup}_2 &= S_2 = 15 & \text{Percentage}_1 &= F_2 = 15\% \\
  \text{Speedup}_3 &= S_3 = 30 & \text{Percentage}_1 &= F_3 = 10\%
  \end{align*}
  \]

- 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?

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

- Speedup = \( \frac{1}{\left(1 - .2 - .15 - .1\right) + .2/10 + .15/15 + .1/30} \)
  
  = \( \frac{1}{\left[ .55 + .0333 \right]} \)
  
  = \( \frac{1}{.5833} \) = 1.71
Before:
Execution Time with no enhancements: 1

Unaffected, fraction: .55

<table>
<thead>
<tr>
<th>Unchanged</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 = .2</td>
</tr>
<tr>
<td>S1 = 10</td>
</tr>
<tr>
<td>F2 = .15</td>
</tr>
<tr>
<td>S2 = 15</td>
</tr>
<tr>
<td>F3 = .1</td>
</tr>
<tr>
<td>S3 = 30</td>
</tr>
</tbody>
</table>

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

Speedup = 1 / .5833 = 1.71

Note: All fractions refer to original execution time.