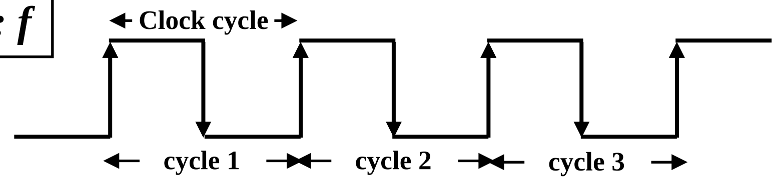
CPU Performance Evaluation: Cycles Per Instruction (CPI)

- **Most computers run synchronously utilizing a CPU clock running at a constant clock rate:** *Or clock frequency: f*



where: Clock rate = 1 / clock cycle

$f = 1/C$

- **The CPU clock rate <u>depends</u> on the <u>specific CPU organization (design)</u> and hardware <u>implementation technology (VLSI)</u> used.**
- **A computer machine (ISA) instruction is comprised of a number of elementary or <u>micro operations</u> which vary in number and complexity depending on the the <u>instruction</u> and the <u>exact CPU organization (Design).</u>**
  - **A <u>micro operation</u> 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.**
- **<u>Thus:</u> A single machine instruction may take one or more CPU cycles to complete termed as the <u>Cycles Per Instruction (CPI).</u>**

Instructions Per Cycle = IPC = 1/CPI

- **<u>Average (or effective) CPI of a program:</u> The average CPI of all instructions executed in the program on a given CPU design.**

4th Edition: Chapter 1 (1.4, 1.7, 1.8)
3rd Edition: Chapter 4

Cycles/sec = Hertz = Hz
MHz = 10^6 Hz GHz = 10^9 Hz

Generic CPU Machine Instruction Processing Steps

Instruction Fetch	**Obtain instruction from program memory** The Program Counter (PC) points to the instruction to be processed
Instruction Decode	**Determine required actions and instruction size**
Operand Fetch	**Locate and obtain operand data** From data memory or registers
Execute	**Compute result value or status**
Result Store	**Deposit results in storage (data memory or register) for later use**
Next Instruction	**Determine successor or next instruction** (i.e Update PC to fetch next instruction to be processed)

CPI = Cycles per instruction

Computer Performance Measures: Program Execution Time

- For a specific program compiled to run on a specific machine (CPU) “A”, has the following parameters:
  - The total executed instruction count of the program. $\boxed{I}$
  - The average number of cycles per instruction (average CPI). $\boxed{CPI}$
  - Clock cycle of machine “A” $\boxed{C}$

  Or effective CPI
- How can one measure the performance of this machine (CPU) 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:

Seconds/program

Programs/second

$$\boxed{\text{Performance}_A = 1 \; / \; \text{Execution Time}_A}$$

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 <u>a given specific program</u>:

$$\text{Performance}_A = 1 \;/\; \text{Execution Time}_A$$
$$\text{Performance}_B = 1 \;/\; \text{Execution Time}_B$$

- Machine A is n times faster than machine B means (or slower? if n < 1) :

$$\boxed{\text{Speedup} = n = \frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution Time}_B}{\text{Execution Time}_A}}$$

- Example:

(i.e Speedup is ratio of performance, no units)

For a given program:

Execution time on machine A: $\text{Execution}_A = 1$ second

Execution time on machine B: $\text{Execution}_B = 10$ seconds

Speedup= $\text{Performance}_A \;/\; \text{Performance}_B = \text{Execution Time}_B \;/\; \text{Execution Time}_A$

$= 10 / 1 = 10$

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.

The two CPUs may target different ISAs provided the program is written in a high level language (HLL)

CPU Execution Time: The CPU Equation

- **A program is comprised of <u>a number of instructions executed , I</u>**
  - **Measured in: instructions/program**

AKA Dynamic Instruction Count

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

Or Instructions Per Cycle (IPC):
IPC = 1/CPI

- **CPU has a fixed clock cycle time <u>C = 1/clock rate</u>**
  - **Measured in: seconds/cycle**

C = 1 / *f*

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

Executed

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

$$T = I \times CPI \times C$$

execution Time per program in seconds | **Number of instructions executed** | **Average CPI for program** | **CPU Clock Cycle**

(This equation is commonly known as the <u>CPU performance equation</u>)

CPU Average CPI/Execution Time

For a given program executed on a given machine (CPU):

CPI = Total program execution cycles / Instructions count Executed (I)

(i.e average or effective CPI)

$\rightarrow$ **CPU clock cycles = Instruction count x CPI**

(executed, I)

CPU execution time =

= CPU clock cycles x Clock cycle

= Instruction count x CPI x Clock cycle

$$T = I \times CPI \times C$$

- T: execution Time per program in seconds
- I: Number of instructions executed
- CPI: Average or effective CPI for program
- C: CPU Clock Cycle

(This equation is commonly known as the CPU performance equation)

CPI = Cycles Per Instruction

CPU Execution Time: Example

- A Program is running on a specific machine (CPU) with the following parameters: I
  - Total executed instruction count: 10,000,000 instructions
  - Average CPI for the program: 2.5 cycles/instruction.
  - CPU clock rate: 200 MHz. (clock cycle = C = 5×10^{-9} seconds)
    i.e 5 nanoseconds
- What is the execution time for this program:

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

CPU time = Instruction count x CPI x Clock cycle

= 10,000,000 x 2.5 x 1 / clock rate

= 10,000,000 x 2.5 x 5×10^{-9}

= 0.125 seconds

Nanosecond = nsec =ns = 10^{-9} second
MHz = 10^{6} Hz

T = I x CPI x C

Factors Affecting CPU Performance

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

$$T = I \times \underset{\text{Average}}{CPI} \times C$$

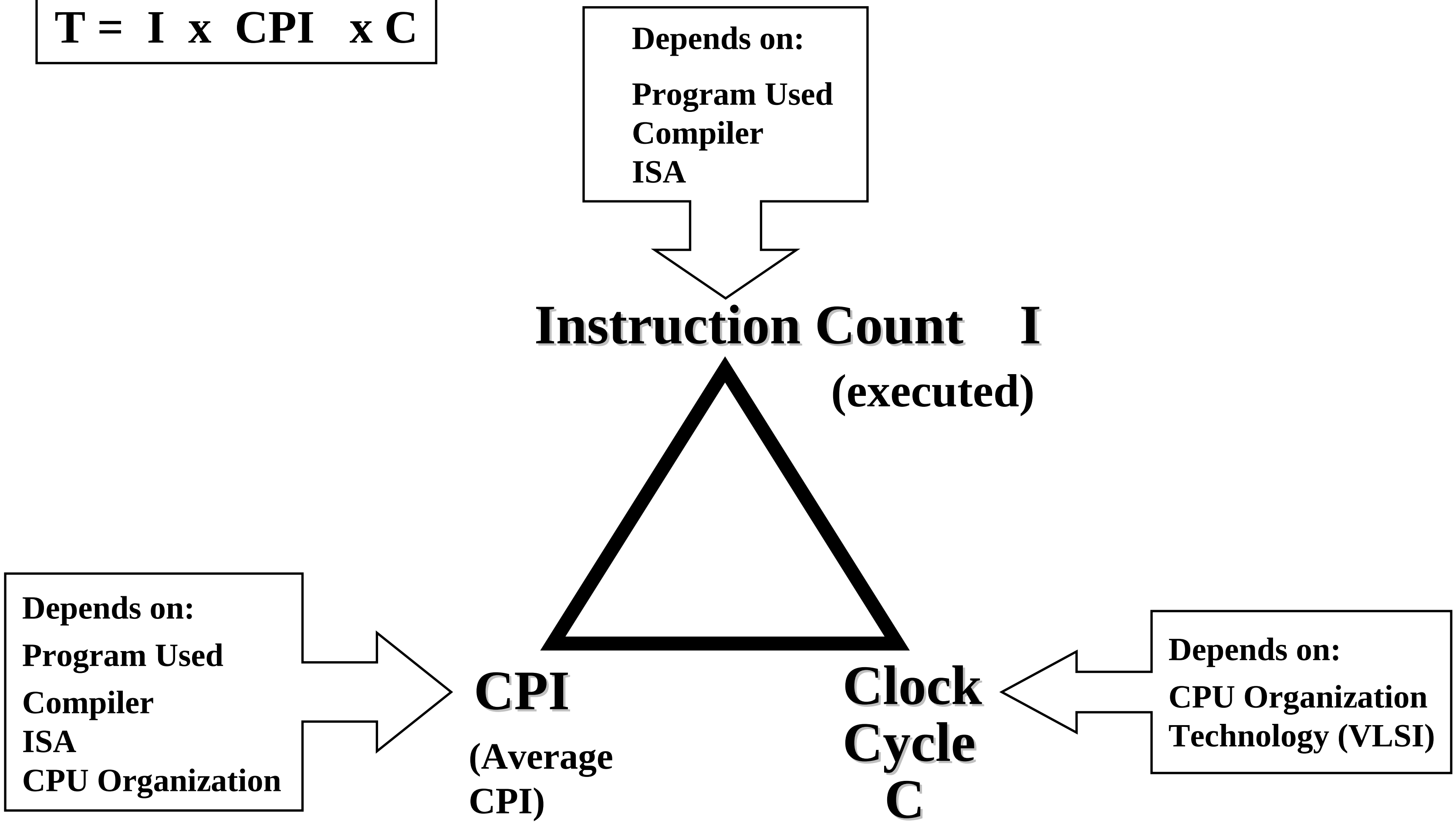
	Instruction Count	Cycles per Instruction	Clock Rate (1/C)
Program			
Compiler			
Instruction Set Architecture (ISA)			
Organization (CPU Design)			
Technology (VLSI)			

$T = I \times CPI \times C$

Aspects of CPU Execution Time

CPU Time = Instruction count executed x CPI x Clock cycle

T = I x CPI x C



Performance Comparison: Example

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

Thus: $C = 1/(200 \times 10^6) = 5 \times 10^{-9}$ seconds

- Using the same program with these changes:
  - A new compiler used: New executed instruction count, I: 9,500,000

    New CPI: 3.0
  - Faster CPU implementation: New clock rate = 300 MHz

Thus: $C = 1/(300 \times 10^6) = 3.33 \times 10^{-9}$ seconds

- What is the speedup with the changes?

$$\text{Speedup} = \frac{\text{Old Execution Time}}{\text{New Execution Time}} = \frac{I_{old} \times CPI_{old} \times \text{Clock cycle}_{old}}{I_{new} \times CPI_{new} \times \text{Clock Cycle}_{new}}$$

$$\text{Speedup} = (10{,}000{,}000 \times 2.5 \times 5 \times 10^{-9}) / (9{,}500{,}000 \times 3 \times 3.33 \times 10^{-9})$$

$$= .125 / .095 = 1.32$$

or 32 % faster after changes.

Clock Cycle = C = 1/ Clock Rate

T = I x CPI x C

Instruction Types & CPI

- **Given a program with n types or classes of instructions executed on a given CPU with the following characteristics:**

e.g ALU, Branch etc.

C_i = **Count of instructions of type_i executed**

CPI_i = **Cycles per instruction for type_i**

$i = 1, 2, \ldots. n$

Depends on CPU Design

Then:

CPI = CPU Clock Cycles / Instruction Count I

i.e average or effective CPI

Executed

Where:

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

$$\text{Executed Instruction Count I} = \Sigma C_i$$

T = I x CPI x C

Instruction Types & CPI: An Example

- **An instruction set has three instruction classes:**

Instruction class	CPI
A	1
B	2
C	3

e.g ALU, Branch etc.

For a specific CPU design

- **Two code sequences have the following instruction counts:**

Program Code Sequence	Instruction counts for instruction class A	B	C
1	2	1	2
2	4	1	1

- **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

i.e average or effective CPI

- **CPU cycles for sequence 2 = 4 x 1 + 1 x 2 + 1 x 3 = 9 cycles**

CPI for sequence 2 = 9 / 6 = 1.5

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

CPI = CPU Cycles / I

Instruction Frequency & CPI

- **Given a program with *n* types or classes of instructions with the following characteristics:**

i = 1, 2, …. n

C_i = Count of instructions of type_i executed

CPI_i = Average cycles per instruction of type_i

F_i = Frequency or fraction of instruction type_i executed

= C_i/ total executed instruction count = C_i/ I

Where: Executed Instruction Count $I = \Sigma C_i$

Then:

$$\underbrace{CPI}_{\text{i.e average or effective CPI}} = \sum_{i=1}^{n} \left(\underbrace{CPI_i \times F_i}_{} \right)$$

Fraction of total execution time for instructions of type i = $\dfrac{CPI_i \times F_i}{CPI}$

T = I x CPI x C

Instruction Type Frequency & CPI: A RISC Example

$$\frac{CPI_i \times F_i}{CPI}$$

Program Profile or Executed Instructions Mix

Base Machine (Reg / Reg)

Depends on CPU Design

Given

Op	Freq, F_i	CPI_i	$CPI_i \times F_i$	% Time
ALU	50%	1	.5	23% = .5/2.2
Load	20%	5	1.0	45% = 1/2.2
Store	10%	3	.3	14% = .3/2.2
Branch	20%	2	.4	18% = .4/2.2

Sum = 2.2

Typical Mix

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

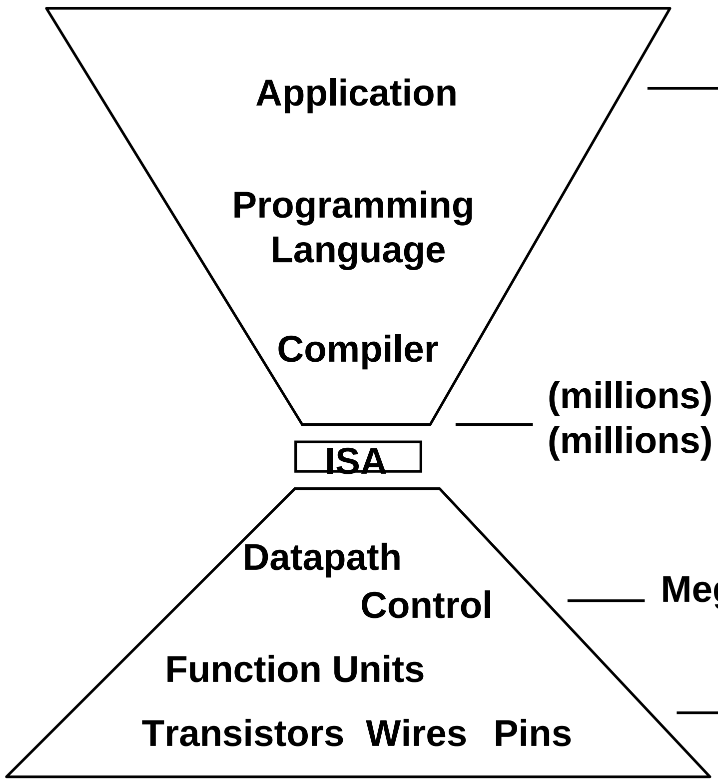
i.e average or effective CPI

$$CPI = .5 \times 1 + .2 \times 5 + .1 \times 3 + .2 \times 2 = 2.2$$
$$= .5 + 1 + .3 + .4$$

$T = I \times CPI \times C$

Metrics of Computer Performance

(Measures)



Execution time: Target workload, SPEC, etc.

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

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, SPEC CPU2000).**
- **Small "Kernel" Benchmarks:** **Also called synthetic 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	Benchmark	Cons
• Representative	Actual Target Workload	• Very specific. • Non-portable. • Complex: Difficult to run, or measure.
• Portable. • Widely used. • Measurements useful in reality.	Full Application Benchmarks	• Less representative than actual workload.
• Easy to run, early in the design cycle.	Small “Kernel” Benchmarks	• Easy to “fool” by designing hardware to run them well.
• Identify peak performance and potential bottlenecks.	Microbenchmarks	• Peak performance results may be a long way from real application performance

SPEC: System Performance Evaluation Corporation

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

Programs application domain: Engineering and scientific computation

- **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
- **SPEC CPU2006, 2006:**
  - CINT2006 (12 integer programs). CFP2006 (17 floating-point intensive programs)
  - Performance relative to a Sun Ultra Enterprise 2 workstation with a 296-MHz UltraSPARC II processor which is given a score of SPECint2006 = SPECfp2006 = 1

All based on execution time and give speedup over a reference CPU

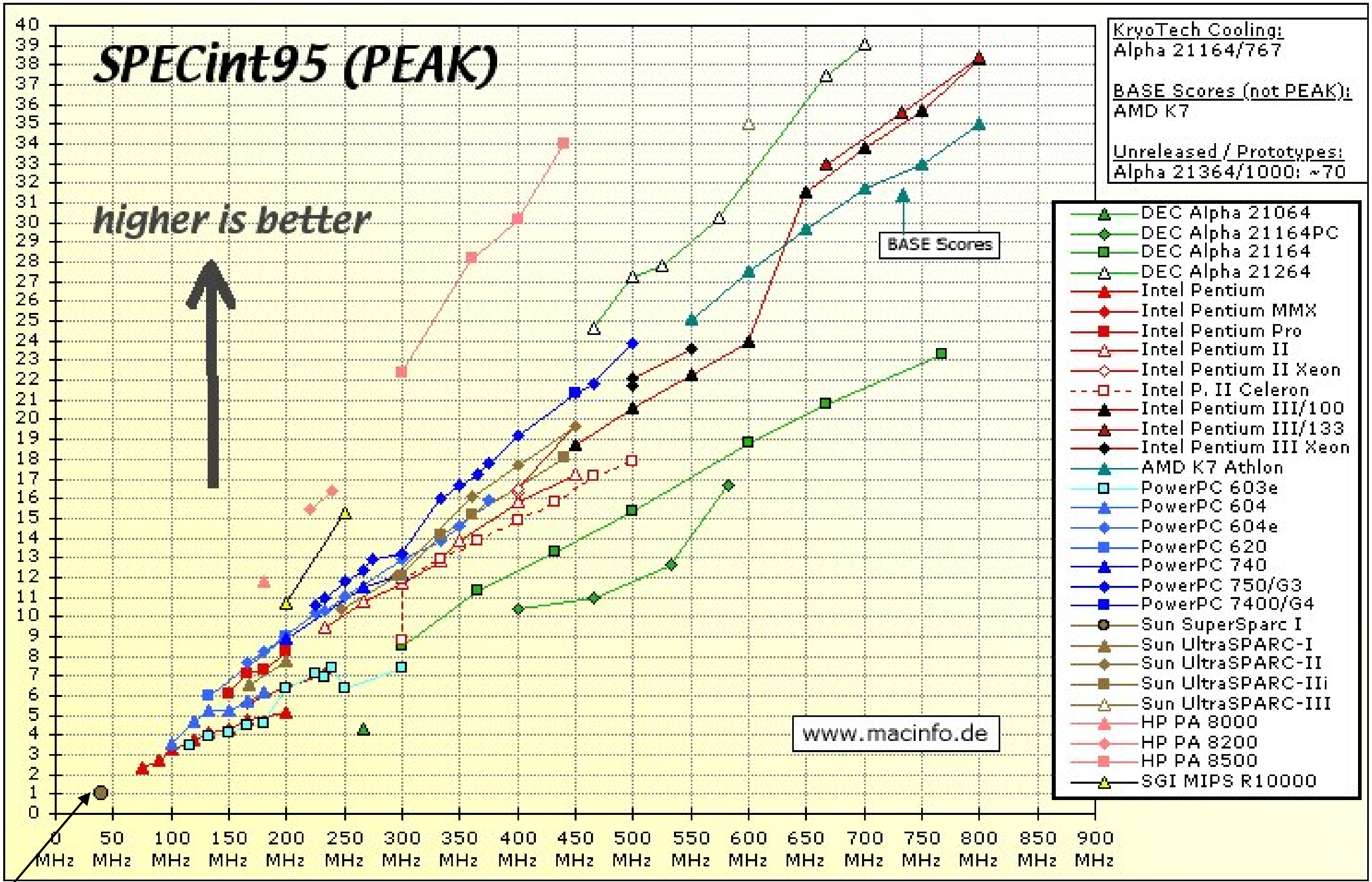
SPEC95 Programs

Programs application domain: Engineering and scientific computation

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

Resulting Performance relative to a Sun SuperSpark I (50 MHz) which is given a score of SPECint95 = SPECfp95 = 1

Sample SPECint95 (Integer) Results

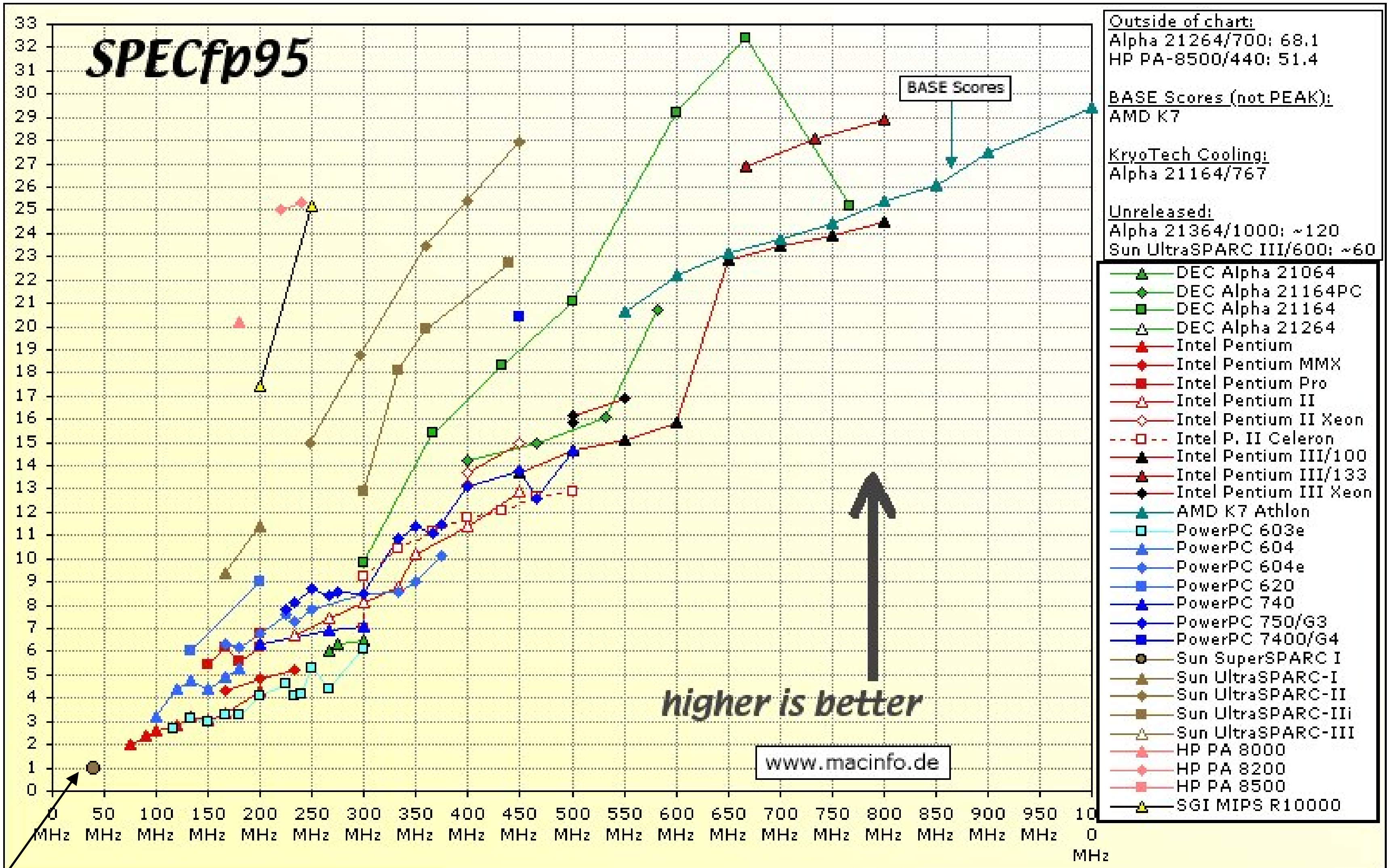


Source URL: http://www.macinfo.de/bench/specmark.html

Sun SuperSpark I (50 MHz) score = 1

T = I x CPI x C

Sample SPECfp95 (Floating Point) Results



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

Sun SuperSpark I (50 MHz) score = 1

T = I x CPI x C

SPEC CPU2000 Programs

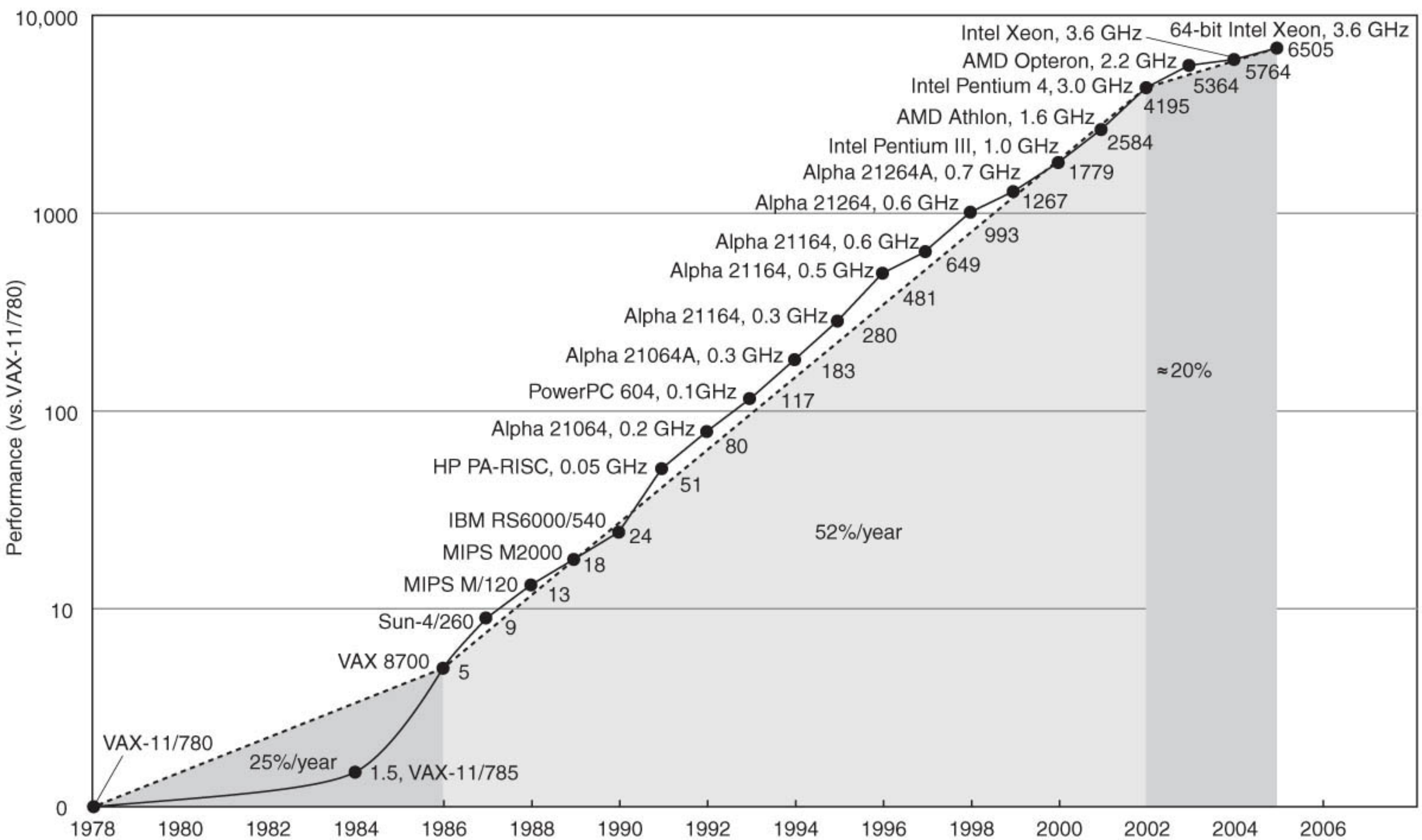
	Benchmark	Language	Descriptions
CINT2000 (Integer) 11 programs	**164.gzip**	**C**	**Compression**
	175.vpr	**C**	**FPGA Circuit Placement and Routing**
	176.gcc	**C**	**C Programming Language Compiler**
	181.mcf	**C**	**Combinatorial Optimization**
	186.crafty	**C**	**Game Playing: Chess**
	197.parser	**C**	**Word Processing**
	252.eon	**C++**	**Computer Visualization**
	253.perlbmk	**C**	**PERL Programming Language**
	254.gap	**C**	**Group Theory, Interpreter**
	255.vortex	**C**	**Object-oriented Database**
	256.bzip2	**C**	**Compression**
	300.twolf	**C**	**Place and Route Simulator**
CFP2000 (Floating Point) 14 programs	**168.wupwise**	**Fortran 77**	**Physics / Quantum Chromodynamics**
	171.swim	**Fortran 77**	**Shallow Water Modeling**
	172.mgrid	**Fortran 77**	**Multi-grid Solver: 3D Potential Field**
	173.applu	**Fortran 77**	**Parabolic / Elliptic Partial Differential Equations**
	177.mesa	**C**	**3-D Graphics Library**
	178.galgel	**Fortran 90**	**Computational Fluid Dynamics**
	179.art	**C**	**Image Recognition / Neural Networks**
	183.equake	**C**	**Seismic Wave Propagation Simulation**
	187.facerec	**Fortran 90**	**Image Processing: Face Recognition**
	188.ammp	**C**	**Computational Chemistry**
	189.lucas	**Fortran 90**	**Number Theory / Primality Testing**
	191.fma3d	**Fortran 90**	**Finite-element Crash Simulation**
	200.sixtrack	**Fortran 77**	**High Energy Nuclear Physics Accelerator Design**
	301.apsi	**Fortran 77**	**Meteorology: Pollutant Distribution**

Programs application domain: Engineering and scientific computation

Source: http://www.spec.org/cpu2000/

Integer SPEC CPU2000 Microprocessor Performance 1978-2006

Performance relative to VAX 11/780 (given a score = 1)



Top 20 SPEC CPU2000 Results (As of March 2002)

Top 20 SPECint2000

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

Top 20 SPECfp2000

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

Performance relative to a Sun Ultra5_10 (300 MHz) which is given a score of SPECint2000 = SPECfp2000 = 100

Source: http://www.aceshardware.com/SPECmine/top.jsp

Top 20 SPEC CPU2000 Results (As of October 2006)

	Top 20 SPECint2000				Top 20 SPECfp2000			
#	MHz	Processor	int peak	int base	MHz	Processor	fp peak	fp base
1	2933	Core 2 Duo EE	3119	3108	2300	POWER5+	3642	3369
2	3000	Xeon 51xx	3102	3089	1600	DC Itanium 2	3098	3098
3	2666	Core 2 Duo	2848	2844	3000	Xeon 51xx	3056	2811
4	2660	Xeon 30xx	2835	2826	2933	Core 2 Duo EE	3050	3048
5	3000	Opteron	2119	1942	2660	Xeon 30xx	3044	2763
6	2800	Athlon 64 FX	2061	1923	1600	Itanium 2	3017	3017
7	2800	Opteron AM2	1960	1749	2667	Core 2 Duo	2850	2847
8	2300	POWER5+	1900	1820	1900	POWER5	2796	2585
9	3733	Pentium 4 E	1872	1870	3000	Opteron	2497	2260
10	3800	Pentium 4 Xeon	1856	1854	2800	Opteron AM2	2462	2230
11	2260	Pentium M	1839	1812	3733	Pentium 4 E	2283	2280
12	3600	Pentium D	1814	1810	2800	Athlon 64 FX	2261	2086
13	2167	Core Duo	1804	1796	2700	PowerPC 970MP	2259	2060
14	3600	Pentium 4	1774	1772	2160	SPARC64 V	2236	2094
15	3466	Pentium 4 EE	1772	1701	3730	Pentium 4 Xeon	2150	2063
16	2700	PowerPC 970MP	1706	1623	3600	Pentium D	2077	2073
17	2600	Athlon 64	1706	1612	3600	Pentium 4	2015	2009
18	2000	Pentium 4 Xeon LV	1668	1663	2600	Athlon 64	1829	1700
19	2160	SPARC64 V	1620	1501	1700	POWER4+	1776	1642
20	1600	Itanium 2	1590	1590	3466	Pentium 4 EE	1724	1719

Performance relative to a Sun Ultra5_10 (300 MHz) which is given a score of SPECint2000 = SPECfp2000 = 100

Source: http://www.aceshardware.com/SPECmine/top.jsp

SPEC CPU2006 Programs

	Benchmark	Language	Descriptions
CINT2006 (Integer) 12 programs	**400.perlbench**	**C**	**PERL Programming Language**
	401.bzip2	**C**	**Compression**
	403.gcc	**C**	**C Compiler**
	429.mcf	**C**	**Combinatorial Optimization**
	445.gobmk	**C**	**Artificial Intelligence: go**
	456.hmmer	**C**	**Search Gene Sequence**
	458.sjeng	**C**	**Artificial Intelligence: chess**
	462.libquantum	**C**	**Physics: Quantum Computing**
	464.h264ref	**C**	**Video Compression**
	471.omnetpp	**C++**	**Discrete Event Simulation**
	473.astar	**C++**	**Path-finding Algorithms**
	483.Xalancbmk	**C++**	**XML Processing**
CFP2006 (Floating Point) 17 programs	**410.bwaves**	**Fortran**	**Fluid Dynamics**
	416.gamess	**Fortran**	**Quantum Chemistry**
	433.milc	**C**	**Physics: Quantum Chromodynamics**
	434.zeusmp	**Fortran**	**Physics/CFD**
	435.gromacs	**C/Fortran**	**Biochemistry/Molecular Dynamics**
	436.cactusADM	**C/Fortran**	**Physics/General Relativity**
	437.leslie3d	**Fortran**	**Fluid Dynamics**
	444.namd	**C++**	**Biology/Molecular Dynamics**
	447.dealII	**C++**	**Finite Element Analysis**
	450.soplex	**C++**	**Linear Programming, Optimization**
	453.povray	**C++**	**Image Ray-tracing**
	454.calculix	**C/Fortran**	**Structural Mechanics**
	459.GemsFDTD	**Fortran**	**Computational Electromagnetics**
	465.tonto	**Fortran**	**Quantum Chemistry**
	470.lbm	**C**	**Fluid Dynamics**
	481.wrf	**C/Fortran**	**Weather Prediction**
	482.sphinx3	**C**	**Speech recognition**

Programs application domain: Engineering and scientific computation

Source: http://www.spec.org/cpu2006/

Example Integer SPEC CPU2006 Performance Results

For 2.5 GHz AMD Opteron X4 model 2356 (Barcelona)

		I	CPI	C	T		Score (speedup)
Description	Name	Instruction Count × 10⁹	CPI	Clock cycle time (seconds × 10⁻⁹)	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2,118	0.75	0.4	637	9,770	15.3
Block-sorting compression	bzip2	2,389	0.85	0.4	817	9,650	11.8
GNU C compiler	gcc	1,050	1.72	0.4	724	8,050	11.1
Combinatorial optimization	mcf	336	10.00	0.4	1,345	9,120	6.8
Go game (AI)	go	1,658	1.09	0.4	721	10,490	14.6
Search gene sequence	hmmer	2,783	0.80	0.4	890	9,330	10.5
Chess game (AI)	sjeng	2,176	0.96	0.4	837	12,100	14.5
Quantum computer simulation	libquantum	1,623	1.61	0.4	1,047	20,720	19.8
Video compression	h264avc	3,102	0.80	0.4	993	22,130	22.3
Discrete event simulation library	omnetpp	587	2.94	0.4	690	6,250	9.1
Games/path finding	astar	1,082	1.79	0.4	773	7,020	9.1
XML parsing	xalancbmk	1,058	2.70	0.4	1,143	6,900	6.0
Geometric Mean							11.7

T on base processor

Performance relative to Base Processor a 296-MHz UltraSPARC II which is given a score of SPECint2006 = SPECfp2006 = 1

Computer Performance Measures : MIPS (Million Instructions Per Second) Rating

- **For a specific program running on a specific CPU the MIPS rating is a measure of how many millions of instructions are executed per second:**

MIPS Rating = Instruction count / (Execution Time x 10^6)

= Instruction count / (CPU clocks x Cycle time x 10^6)

= (Instruction count x Clock rate) / (Instruction count x CPI x 10^6)

= Clock rate / (CPI x 10^6)

- **<u>Major problem with MIPS rating:</u> As shown above the MIPS rating does not account for the count of instructions executed (I).**
  - **A higher MIPS rating in many cases may not mean higher performance or better execution time. i.e. due to compiler design variations.**
- **In addition the MIPS rating:**
  - **Does not account for the instruction set architecture (ISA) used.**
    - **Thus it cannot be used to compare computers/CPUs with different instruction sets.**
  - **<u>Easy to abuse:</u> Program used to get the MIPS rating is often omitted.**
    - **Often the <u>Peak MIPS rating</u> is provided for a given CPU which is obtained using a program comprised entirely of <u>instructions with the lowest CPI</u> for the given CPU design which <u>does not represent real programs.</u>**

T = I x CPI x C

Computer Performance Measures : MIPS (Million Instructions Per Second) Rating

- **Under what conditions can the MIPS rating be used to compare performance of different CPUs?**

- **The MIPS rating is <u>only valid</u> to compare the performance of different CPUs <u>provided that the following conditions are satisfied</u>:**

  1 **<u>The same program is used</u>**

  (actually this applies to all performance metrics)

  2 **<u>The same ISA is used</u>**

  3 **<u>The same compiler is used</u>**

  $\Rightarrow$ **(Thus the resulting programs used to run on the CPUs and obtain the MIPS rating are <u>identical</u> at the machine code level including the <u>same instruction count</u>)** **(binary)**

Compiler Variations, MIPS & Performance: An Example

- **For a machine (CPU) with instruction classes:**

Instruction class	CPI
A	1
B	2
C	3

e.g ALU, Branch etc.

For a specific CPU design

- **For a given high-level language program, two compilers produced the following executed instruction counts:**

Code from:	Instruction counts (in millions) for each instruction class: A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

- **The machine is assumed to run at a clock rate of 100 MHz.**

Compiler Variations, MIPS & Performance: An Example (Continued)

MIPS = Clock rate / (CPI x 10^6) = 100 MHz / (CPI x 10^6)

CPI = CPU execution cycles / Instructions count

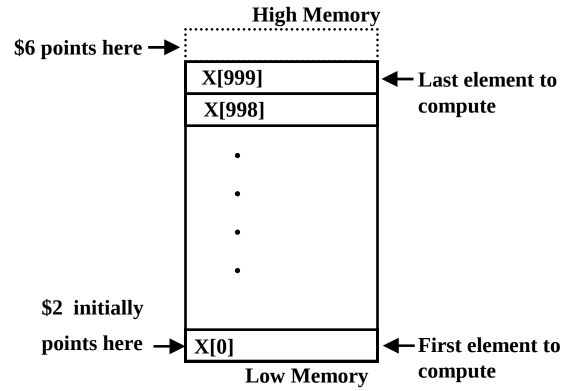
$$CPU\ clock\ cycles = \sum_{i=1}^{n}\left(CPI_i \times C_i\right)$$

CPU time = Instruction count x CPI / Clock rate

- **For compiler 1:**
  - **CPI_1 = (5 x 1 + 1 x 2 + 1 x 3) / (5 + 1 + 1) = 10 / 7 = 1.43**
  - **MIPS $Rating_1$ = 100 / (1.428 x 10^6) = 70.0 MIPS**
  - **CPU $time_1$ = ((5 + 1 + 1) x 10^6 x 1.43) / (100 x 10^6) = 0.10 seconds**
- **For compiler 2:**
  - **CPI_2 = (10 x 1 + 1 x 2 + 1 x 3) / (10 + 1 + 1) = 15 / 12 = 1.25**
  - **MIPS $Rating_2$ = 100 / (1.25 x 10^6) = 80.0 MIPS**
  - **CPU $time_2$ = ((10 + 1 + 1) x 10^6 x 1.25) / (100 x 10^6) = 0.15 seconds**

MIPS rating indicates that compiler 2 is better while in reality the code produced by compiler 1 is faster

MIPS (The ISA not the metric) Loop Performance Example



For the loop:

for (i=0; i<1000; i=i+1){
x[i] = x[i] + s; }

MIPS assembly code is given by:

```
        lw      $3,  0($1)        ; load s in $3
        addi    $6,  $2,  4000    ; $6 = address of last element + 4
loop:   lw      $4,  0($2)        ; load x[i] in $4
        add     $5,  $4, $3       ; $5 has x[i] + s
        sw      $5,  0($2)        ; store computed x[i]
        addi    $2,  $2,  4       ; increment $2 to point to next x[ ] element
        bne     $6,  $2,  loop    ; last loop iteration reached?
```

The MIPS code is executed on a specific CPU that runs at 500 MHz (C = clock cycle = 2ns = 2×10^{-9} seconds) with following instruction type CPIs :

Instruction type	CPI
ALU	4
Load	5
Store	7
Branch	3

For this MIPS code running on this CPU find:

1- Fraction of total instructions executed for each instruction type
2- Total number of CPU cycles
3- Average CPI
4- Fraction of total execution time for each instructions type
5- Execution time
6- MIPS rating , peak MIPS rating for this CPU

X[ ] array of words in memory, base address in $2 ,
s a constant word value in memory, address in $1

MIPS (The ISA) Loop Performance Example (continued)

- The code has 2 instructions before the loop and 5 instructions in the body of the loop which iterates 1000 times,
- Thus: Total instructions executed, I = 5x1000 + 2 = 5002 instructions

1 **Number of instructions executed/fraction F_i for each instruction type:**
   - ALU instructions = 1 + 2x1000 = 2001 $CPI_{ALU} = 4$ $Fraction_{ALU} = F_{ALU}$ = 2001/5002 = 0.4 = 40%
   - Load instructions = 1 + 1x1000 = 1001 $CPI_{Load} = 5$ $Fraction_{Load} = F_{Load}$ = 1001/5002= 0.2 = 20%
   - Store instructions = 1000 $CPI_{Store} = 7$ $Fraction_{Store} = F_{Store}$ = 1000/5002 = 0.2 = 20%
   - Branch instructions = 1000 $CPI_{Branch} = 3$ $Fraction_{Branch}= F_{Branch}$ = 1000/5002= 0.2 = 20%

2 $$CPU\ clock\ cycles = \sum_{i=1}^{n} \left(CPI_i \times C_i \right)$$

= 2001x4 + 1001x5 + 1000x7 + 1000x3 = 23009 cycles

3 **Average CPI = CPU clock cycles / I = 23009/5002 = 4.6**

4 **Fraction of execution time for each instruction type:**
   - Fraction of time for ALU instructions = CPI_{ALU} x F_{ALU} / CPI= 4x0.4/4.6 = 0.348 = 34.8%
   - Fraction of time for load instructions = CPI_{load} x F_{load} / CPI= 5x0.2/4.6 = 0.217 = 21.7%
   - Fraction of time for store instructions = CPI_{store} x F_{store} / CPI= 7x0.2/4.6 = 0.304 = 30.4%
   - Fraction of time for branch instructions = CPI_{branch} x F_{branch} / CPI= 3x0.2/4.6 = 0.13 = 13%

Instruction type	CPI
ALU	4
Load	5
Store	7
Branch	3

5 **Execution time = I x CPI x C = CPU cycles x C = 23009 x 2×10^{-9} =**

= 4.6×10^{-5} seconds = 0.046 msec = 46 usec

6 **MIPS rating = Clock rate / (CPI x 10^6) = 500 / 4.6 = 108.7 MIPS**
   - The CPU achieves its peak MIPS rating when executing a program that only has instructions of the type with the lowest CPI. In this case branches with $CPI_{Branch} = 3$
   - Peak MIPS rating = Clock rate / (CPI_{Branch} x 10^6) = 500/3 = 166.67 MIPS

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:

MFLOPS = Number of floating-point operations / (Execution time x 10^6)

- MFLOPS rating is a better comparison measure between different machines (applies even if ISAs are different) than the MIPS rating.
  - Applicable even if ISAs are different
- 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.
  - <u>Peak MFLOPS rating for a CPU:</u> Obtained using a program comprised entirely of the <u>simplest floating point instructions</u> (with the lowest CPI) for the given CPU design which <u>does not represent real floating point programs.</u>

Current peak MFLOPS rating: 8,000-20,000 MFLOPS (8-20 GFLOPS) per processor core

Quantitative Principles of Computer Design

- **Amdahl's Law:**

The performance gain from improving some portion of a computer is calculated by:

i.e using some enhancement

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement}}{\text{Performance for the entire task without using the enhancement}}$$

Before Enhancement

$$\text{or Speedup} = \frac{\text{Execution time without the enhancement}}{\text{Execution time for entire task using the enhancement}}$$

After Enhancement

Here: Task = Program

Recall: Performance = 1 /Execution Time

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:

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

original

- **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:**

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

Hence speedup is given by:

$$\text{Speedup(E)} = \frac{\cancel{\text{Execution Time without E}}}{((1 - F) + F/S) \times \cancel{\text{Execution Time without E}}} = \frac{1}{(1 - F) + F/S}$$

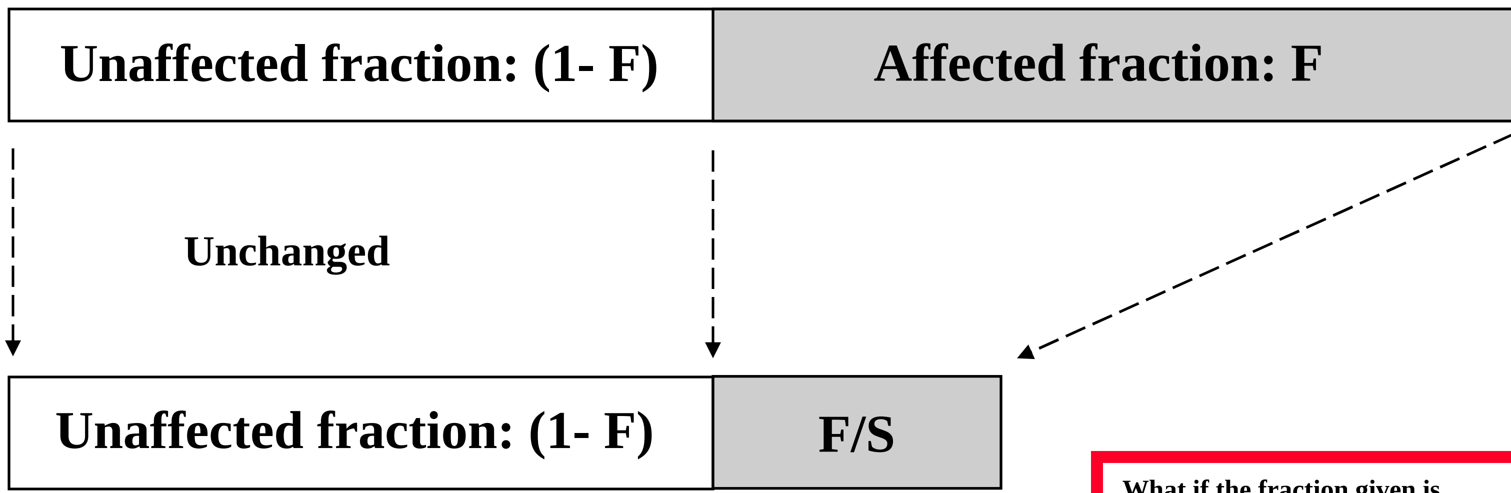
F (Fraction of execution time enhanced) refers to original execution time before the enhancement is applied

Pictorial Depiction of Amdahl's Law

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

Before:
Execution Time without enhancement E: (Before enhancement is applied)

- **shown normalized to 1 = (1-F) + F =1**



After:
Execution Time with enhancement E:

What if the fraction given is after the enhancement has been applied? How would you solve the problem? (i.e find expression for speedup)

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

Performance Enhancement Example

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

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%

CPI = 2.2

From a previous example

- **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 = 1- F = 100% - 45% = 55% or .55

Factor of enhancement = S = 5/2 = 2.5

Using Amdahl's Law:

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

An Alternative Solution Using CPU Equation

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%

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 of load is now 2 instead of 5

New CPI = .5 x 1 + .2 x 2 + .1 x 3 + .2 x 2 = 1.6

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

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

Which is the same speedup obtained from Amdahl's Law in the first solution.

T = I x CPI x C

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 = 100/4 = 25 seconds**

25 seconds = (100 - 80 seconds) + 80 seconds / S

25 seconds = 20 seconds + 80 seconds / S

→ **5 = 80 seconds / S**

→ **S = 80/5 = 16**

Alternatively, it can also be solved by finding enhanced fraction of execution time:

F = 80/100 = .8

and then solving Amdahl's speedup equation for desired enhancement factor S

$$\text{Speedup(E)} = \frac{1}{(1 - F) + F/S} = 4 = \frac{1}{(1 - .8) + .8/S} = \frac{1}{.2 + .8/s}$$

Solving for S gives S= 16

Hence multiplication should be 16 times faster to get an overall speedup of 4.

Machine = CPU

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?

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

$\rightarrow$ Execution time with enhancement = 100/5 = 20 seconds

20 seconds = (100 - 80 seconds) + 80 seconds / s

20 seconds = 20 seconds + 80 seconds / s

$\rightarrow$ 0 = 80 seconds / s

No amount of multiplication speed improvement can achieve this.

Extending Amdahl's Law To Multiple Enhancements

n enhancements each affecting a different portion of execution time

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

i = 1, 2, …. n

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

Unaffected fraction

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

What if the fractions given are after the enhancements were applied? How would you solve the problem? (i.e find expression for speedup)

Note: All fractions F_i refer to original execution time before the enhancements are applied.

Amdahl's Law With Multiple Enhancements: Example

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

$$\begin{array}{ll} \text{Speedup}_1 = S_1 = 10 & \text{Percentage}_1 = F_1 = 20\% \\ \text{Speedup}_2 = S_2 = 15 & \text{Percentage}_1 = F_2 = 15\% \\ \text{Speedup}_3 = S_3 = 30 & \text{Percentage}_1 = F_3 = 10\% \end{array}$$

- **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?**

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

- **Speedup = 1 / [(1 - .2 - .15 - .1) + .2/10 + .15/15 + .1/30)]**

  = 1 / [.55 + .0333]

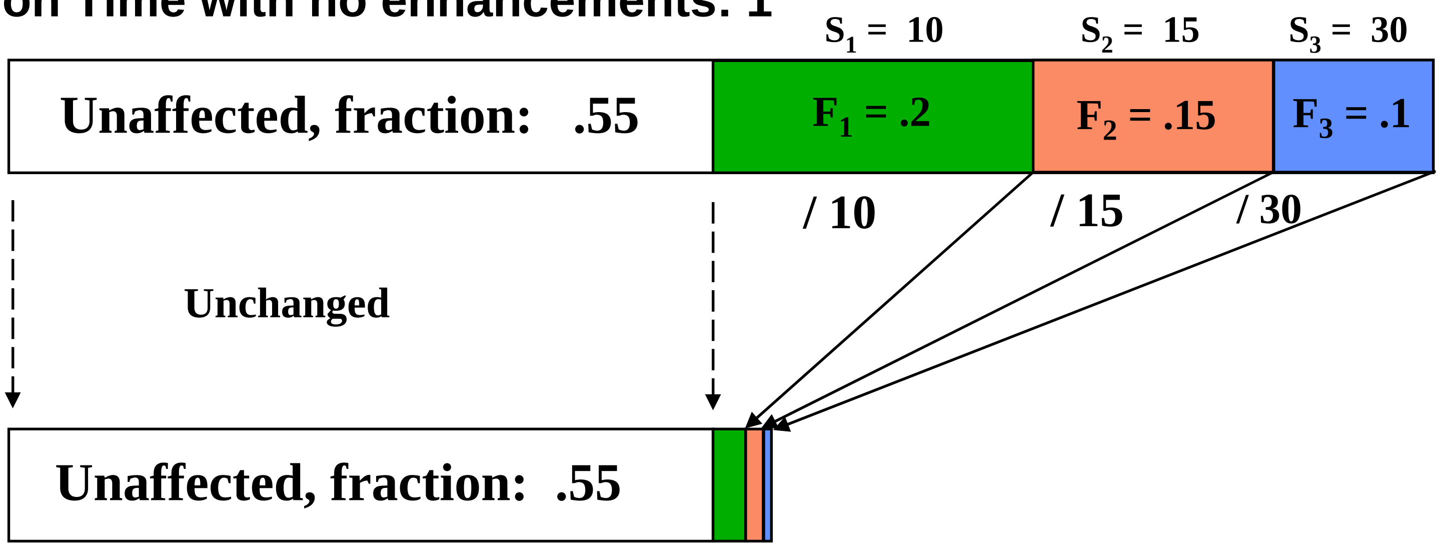
  = 1 / .5833 = 1.71

Pictorial Depiction of Example

Before:

Execution Time with no enhancements: 1

i.e normalized to 1



After:

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

Speedup = 1 / .5833 = 1.71

What if the fractions given are after the enhancements were applied? How would you solve the problem?

Note: All fractions F_i refer to original execution time.

"Reverse" Multiple Enhancements Amdahl's Law

- **Multiple Enhancements Amdahl's Law assumes that the fractions given refer to original execution time.**
- **If for each enhancement S_i the fraction F_i it affects is given as a fraction of the resulting execution time after the enhancements were applied then:**

$$Speedup = \frac{\left(\left(1 - \sum_i F_i\right) + \sum_i F_i \times S_i\right) X \cancel{\text{Resulting ExecutionTime}}}{\cancel{\text{Resulting ExecutionTime}}}$$

Unaffected fraction

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

i.e as if resulting execution time is normalized to 1

- **For the previous example assuming fractions given refer to resulting execution time after the enhancements were applied (not the original execution time), then:**

Speedup = (1 - .2 - .15 - .1) + .2 x10 + .15 x15 + .1x30
= .55 + 2 + 2.25 + 3
= 7.8

Find original fractions?