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 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:

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

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:

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

- Machine A is n times faster than machine B means:

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

- 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

$$\text{Performance}_A / \text{Performance}_B = \text{Execution Time}_B / \text{Execution Time}_A$$
$$= 10 / 1 = 10$$

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.

CPU Execution Time: The CPU Equation

- **A program is comprised of a number of instructions, 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 rate**
  - **Measured in: seconds/cycle**
- **CPU execution time is the product of the above three parameters as follows:**

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

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

CPU Execution Time

For a given program and machine:

CPI = Total program execution cycles / Instructions count

$\rightarrow$ 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 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:**

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

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}

= .125 seconds

Factors Affecting CPU Performance

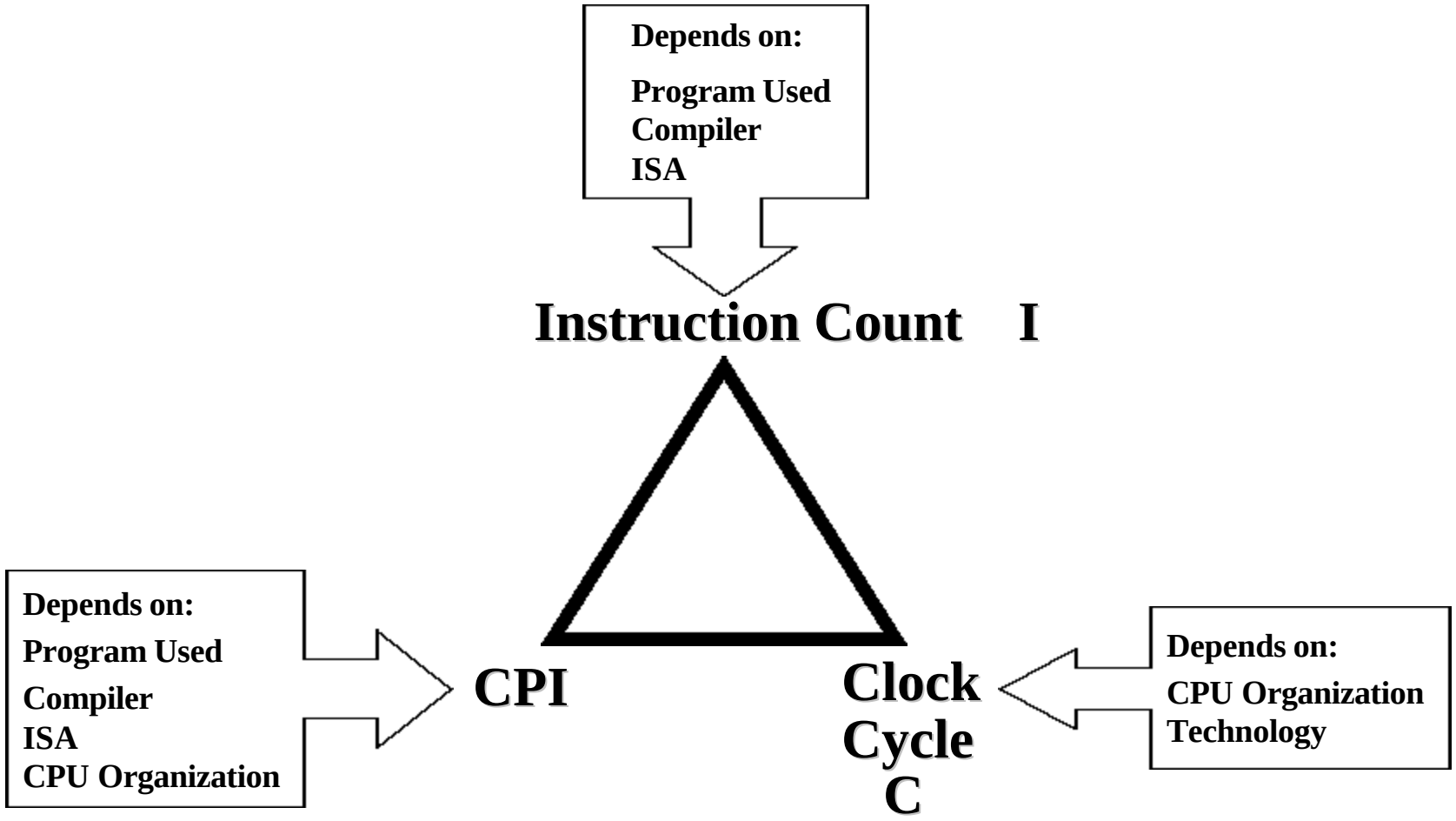
$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

T = I x CPI x C

	Instruction Count	Cycles per Instruction	Clock Rate
Program			
Compiler			
Instruction Set Architecture (ISA)			
Organization			
Technology			

Aspects of CPU Execution Time

CPU Time = Instruction count x CPI x Clock cycle



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?

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

$$\text{Speedup} = (10{,}000{,}000 \times 2.5 \times 5\times10^{-9}) / (9{,}500{,}000 \times 3 \times 3.33\times10^{-9})$$

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

or 32 % faster after changes.

Instruction Types & CPI

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

C_i = **Count of instructions of type$_i$**

CPI_i = **Cycles per instruction for type$_i$**

Then:

CPI = CPU Clock Cycles / Instruction Count I

Where:

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

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

Instruction Types & CPI: An Example

- An instruction set has three instruction classes:

Instruction class	CPI
A	1
B	2
C	3

- Two code sequences have the following instruction counts:

	Instruction counts for instruction class		
Code Sequence	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

- 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 = Count of instructions of type$_i$

  CPI_i = Average cycles per instruction of type$_i$

  F_i = Frequency of instruction type$_i$

  = C_i/ total instruction count

Then:

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

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

Instruction Type Frequency & CPI: A RISC Example

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

Base Machine (Reg / Reg)

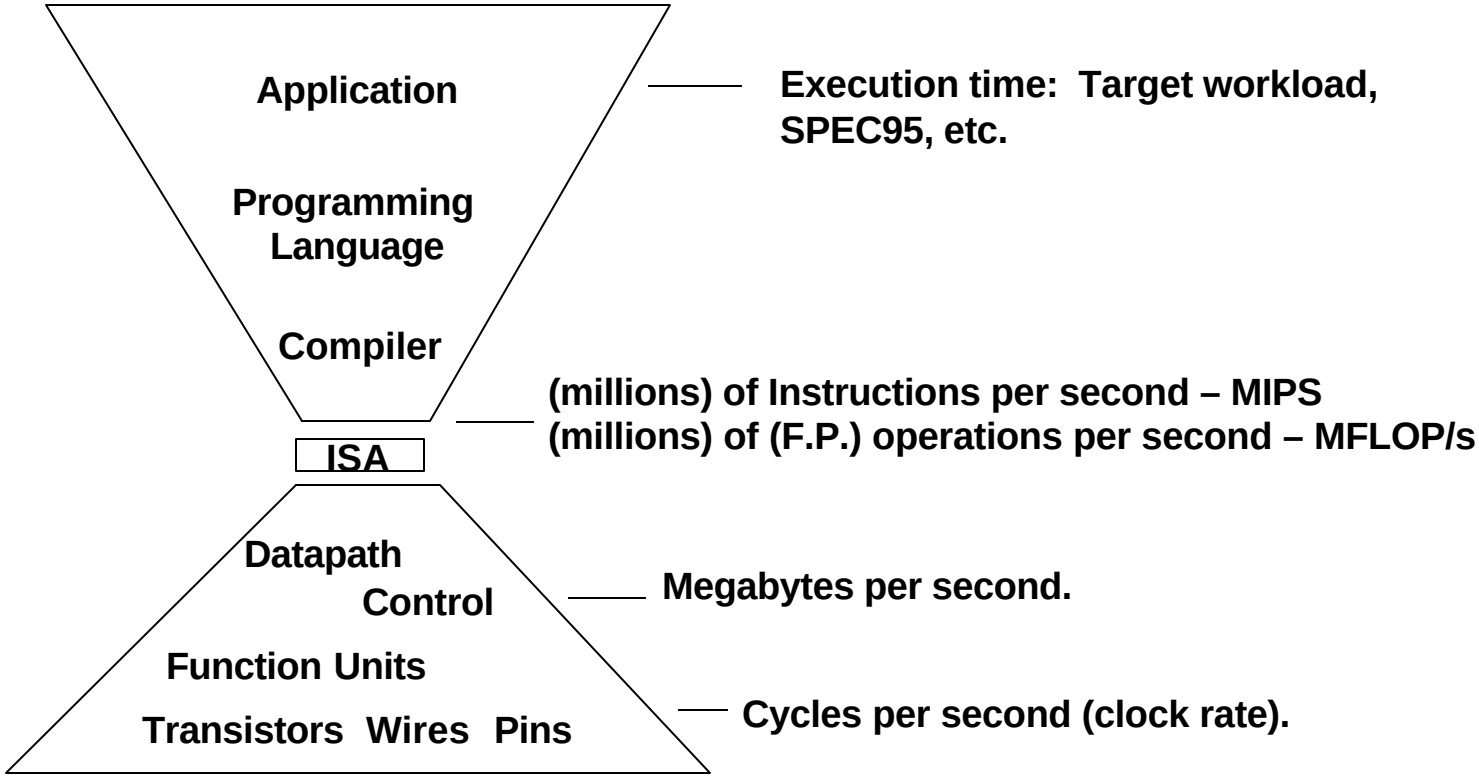
Op	Freq, F_i	CPI_i	$CPI_i \times F_i$	% Time
ALU	50%	1	.5	23% = .5/2.2
Load	20%	5	1.0	45% = 1/2.2
Store	10%	3	.3	14% = .3/2.2
Branch	20%	2	.4	18% = .4/2.2

Typical Mix

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

CPI = .5 x 1 + .2 x 5 + .1 x 3 + .2 x 2 = 2.2

Metrics of Computer Performance



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:**
  - **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		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 Cooperative

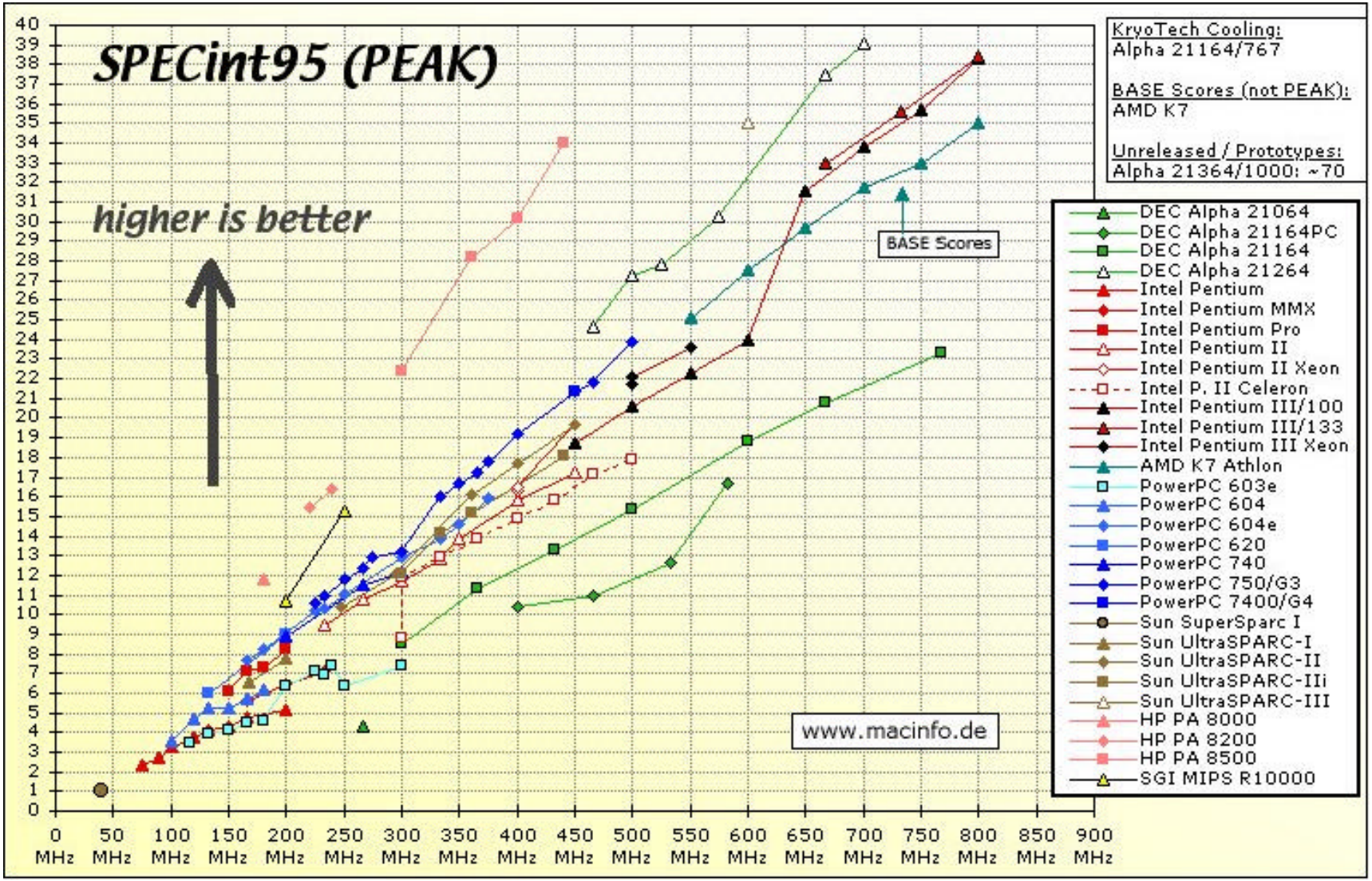
The most popular and industry-standard set of CPU benchmarks.

- **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

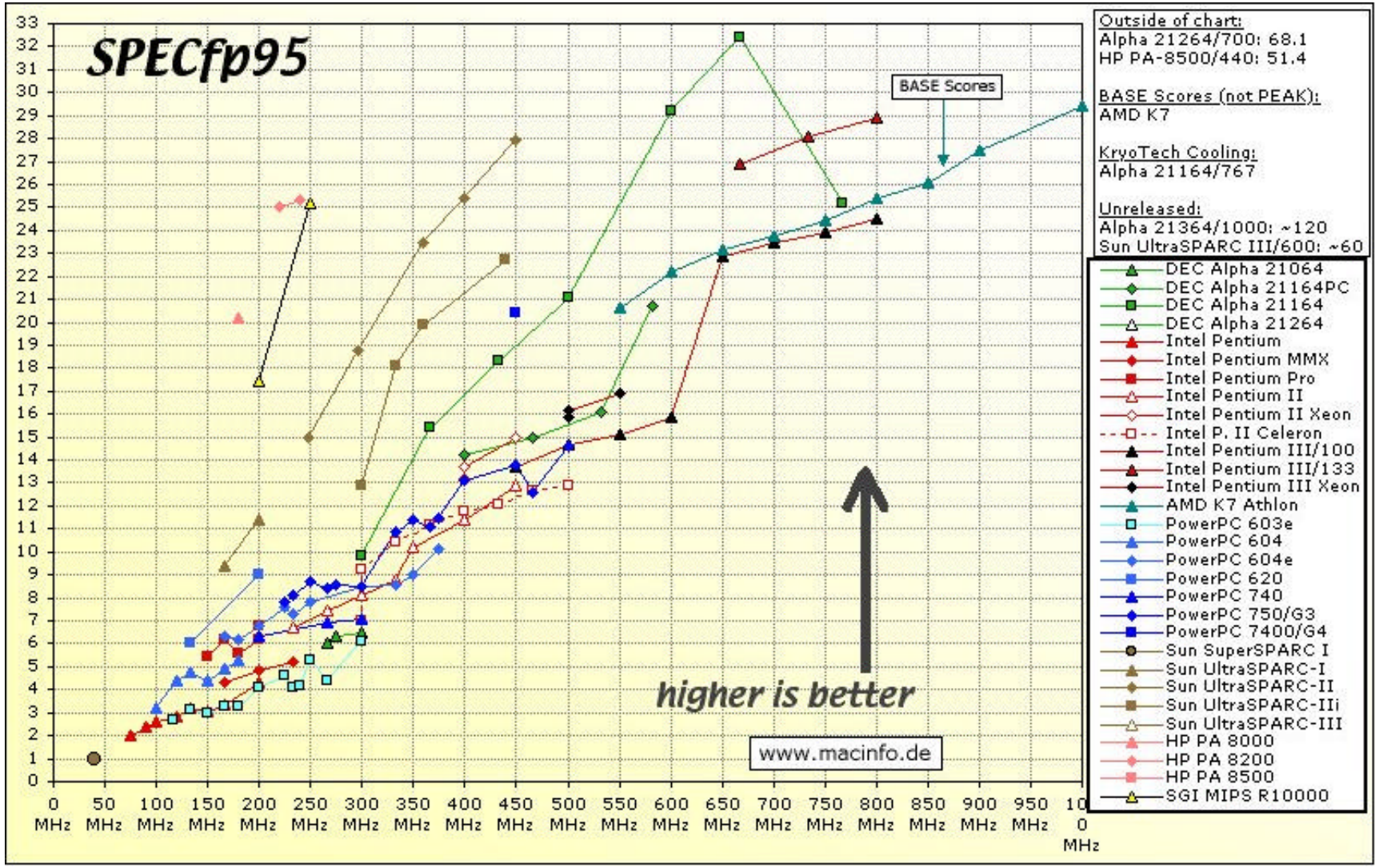
	Benchmark	Description
Integer	go	Artificial intelligence; plays the game of Go
	m88ksim	Motorola 88k chip simulator; runs test program
	gcc	The Gnu C compiler generating SPARC code
	compress	Compresses and decompresses file in memory
	li	Lisp interpreter
	ijpeg	Graphic compression and decompression
	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

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

	Benchmark	Language	Descriptions
CINT2000 (Integer)	**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)	**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**

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

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

Source: 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:**

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

- **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:

Instruction class	CPI
A	1
B	2
C	3

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

	Instruction counts (in millions) for each instruction class		
Code from:	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**
  - **MIP_1 = 100 / (1.428 x 10^6) = 70.0**
  - **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**
  - **MIP_2 = 100 / (1.25 x 10^6) = 80.0**
  - **CPU $time_2$ = ((10 + 1 + 1) x 10^6 x 1.25) / (100 x 10^6) = 0.15 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:**

MFLOPS = Number of floating-point operations / (Execution time x 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:

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

  - 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:

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

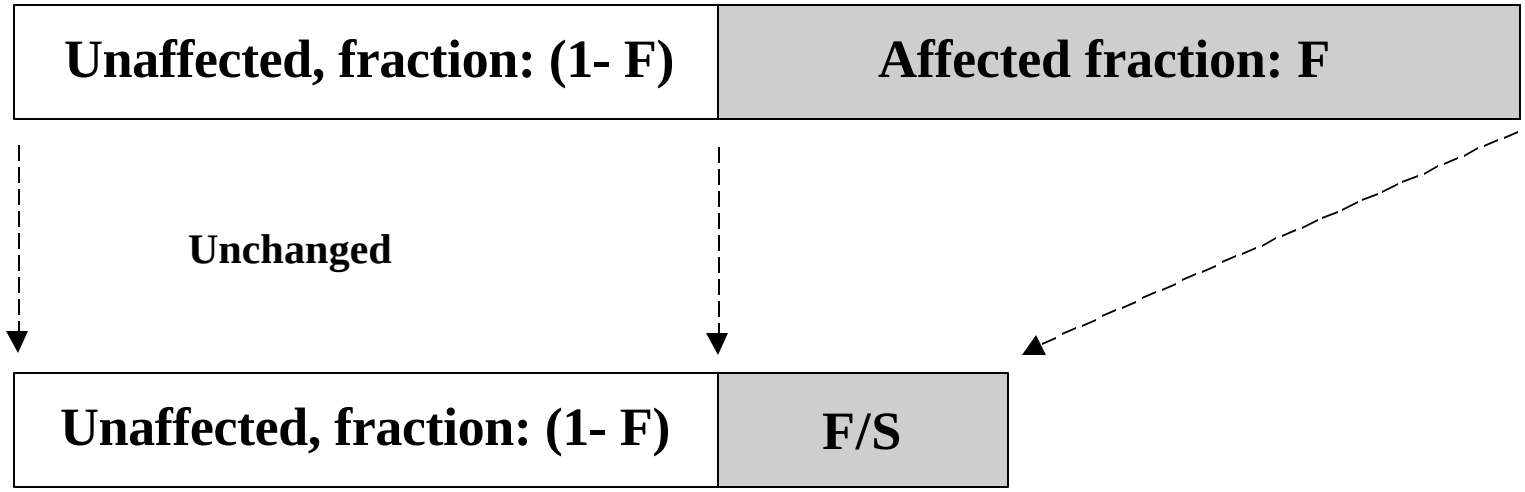
Hence speedup is given by:

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

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:



After:
Execution Time with enhancement E:

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

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

- **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:

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

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

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

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

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?**

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

$\rightarrow$ **Execution time with enhancement = 25 seconds**

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

25 seconds = 20 seconds + 80 seconds / n

$\rightarrow$ **5 = 80 seconds / n**

$\rightarrow$ **n = 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?**

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

$\rightarrow$ **Execution time with enhancement = 20 seconds**

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

20 seconds = 20 seconds + 80 seconds / n

$\rightarrow$ **0 = 80 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:**

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

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

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:**

  $Speedup_1 = S_1 = 10$ $Percentage_1 = F_1 = 20\%$
  $Speedup_2 = S_2 = 15$ $Percentage_1 = F_2 = 15\%$
  $Speedup_3 = S_3 = 30$ $Percentage_1 = F_3 = 10\%$

- **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?**

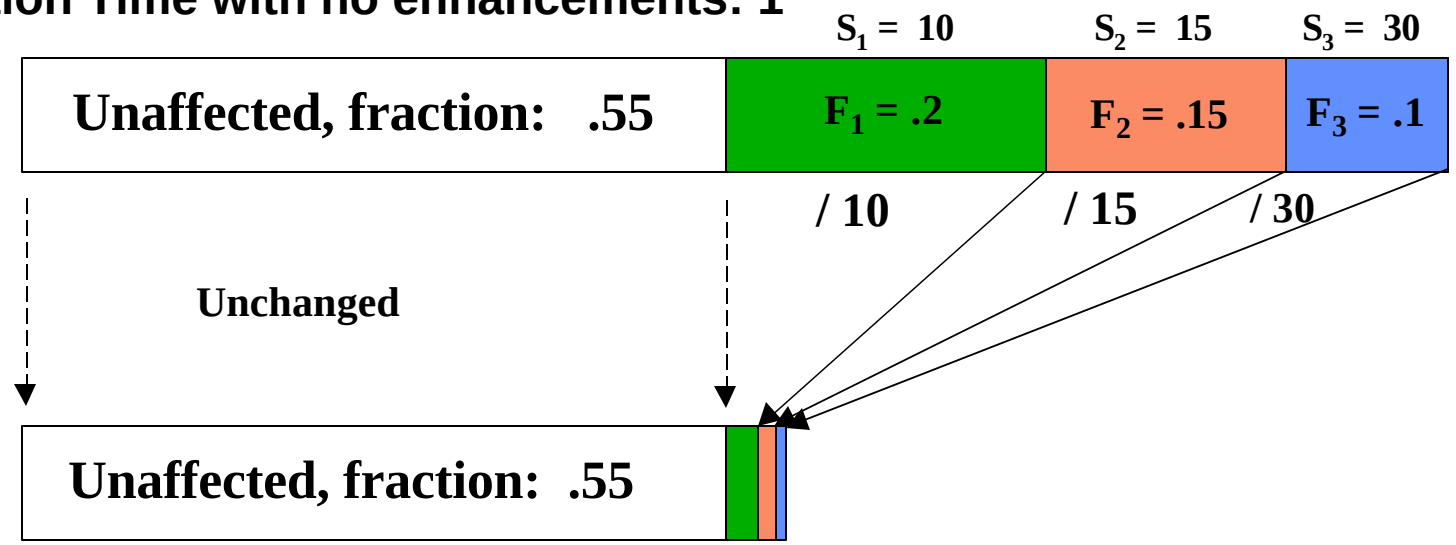
$$Speedup = \frac{1}{\left((1 - \sum_i F_i) + \sum_i \frac{F_i}{S_i}\right)}$$

- **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



After:

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

Speedup = 1 / .5833 = 1.71

Note: All fractions refer to original execution time.