6 Measuring Performance

- **Key measure of performance for a computing system is speed**
  - ***Response time*** **or** ***execution time*** **or** ***latency*.**
  - ***Throughput*.**
- **Throughput is relevant to I/O, particularly in large systems which handle many jobs**
- **Reducing execution time will nearly always improve throughput; reverse is not true. ⇒ We concentrate on execution time.**
  - **Execution time can mean:**
    - » ***Elapsed time*** **-- includes all I/O, OS and time spent on other jobs**
    - » **CPU time -- time spent by processor on your job (no I/O)**

      CPU time can mean *user* CPU time or *System* CPU time
    - » **Unix format:** `90.7u   12.9s   2:39   65%`

      user CPU time is 90.7 sec; system CPU time is 12.9 sec; total elapsed time is 2 min., 39 sec; total CPU time is 65% of total elapsed time.

IBM
מ ב י

CPU Execution Time

We consider CPU execution time on an *unloaded system.*

machine X is *n* times faster than machine Y if

$$\frac{\text{CPU Time}_Y}{\text{CPU Time}_X} = n \quad \textbf{or} \quad \frac{\text{performance}_X}{\text{performance}_Y} = n$$

CPU Time = EXECUTION Time

$$\text{performance} = \frac{1}{\text{CPU Time}}$$

» Basic measure of performance:

- **CPU Time =** $\frac{\text{Clock cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{Clock cycle}}$ (= Cycles count **X** Clock cycle time)
- **The *clock* in a digital system creates a sequence of hardware signals; hardware events must occur at these predefined *clock ticks* . The *clock cycle time* or just *clock cycle,* or even *clock period* is the time between two clock ticks.**
- **Clock cycle time is measured in *nanoseconds* (10^{-9}sec) or *microseconds* (10^{-6}sec)**
- **Clock *rate* =** $\frac{1}{\text{Clock cycle time}}$ **is measured in *MegaHertz* (MHz) (10^{6} cycles/sec)**

IBM
מבי

Example

Program P runs on computer A in 10 seconds. Designer says clock rate can be increased significantly, but total cycle count will also increase by 20%. What clock rate do we need on computer B for P to run in 6 seconds? (Clock rate on A is 100 MHz).

The new machine is B. We want CPU Time_B = 6 seconds.

We know that Cycles count_B = 1.2 Cycles count_A. Calculate Cycles count_A.

$$\text{CPU Time}_A = 10 \text{ sec.} = \frac{\text{Cycles count}_A}{100 \times 10^6 \text{ cycles/sec}} \quad ; \text{Cycles count}_A = 1000 \times 10^6 \text{ cycles}$$

Calculate Clock rate_B:

$$\text{CPU Time}_B = 6 \text{ sec.} = \frac{1.2\text{Cycles count}_A}{\text{Clock rate}_B} \quad ; \text{Clock rate}_B = \frac{200 \times 10^6 \text{ cycles}}{\text{second}} = 200 \text{ MHz}$$

✓ Machine B must run at twice the clock rate of A to achieve the target execution time.

CPI (Cycles per Instruction)

$$\textbf{Cycles Count} = \frac{\text{Instructions}}{\text{program}} \times \frac{\text{Average clock cycles}}{\text{instruction}} \quad (= \text{IC} \times \text{CPI})$$

CPI is one way to compare different implementations of the same Instruction Set Architecture (ISA), since instruction count (IC) for a given program will be the same in both cases.

Example:

We have two machines with different implementations of the same ISA. Machine A has a clock cycle time of 10 ns and a CPI of 2.0 for program P; machine B has a clock cycle time of 20 ns and a CPI of 1.2 for the same program. Which machine is faster?

Let IC be the number of instructions to be executed. Then

Cycles count_A = **2.0 IC**

Cycles count_B = **1.2 IC**

calculate CPU Time for each machine:

CPU Time_A = **2.0 IC x 10 ns = 20.0 IC ns**

CPU Time_B = **1.2 IC x 20 ns = 24.0 IC ns**

» **Machine A is faster; in fact 24/20 = 20% faster.**

Composite Performance Measure

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{program}} \times \frac{\text{Average clock cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{clock cycle}}$$

$$\text{or} = \text{Instruction Count} \times \text{CPI} \times \text{clock cycle time}$$

$$\text{or} = \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock rate}}$$

- **These formulas show that performance is always a function of 3 distinct factors; 1 or 2 factors alone are not sufficient.**
- **IC (Instruction Count) was once the main factor advertised (VAX); today clock rate is in the headlines (700 MHz Pentiums; 600 MHz Alpha).**
- **CPI is more difficult to advertise.**
  - **Changing one factor often affects others.**
  - **Lower CPI means each instruction may be doing less; hence may increase IC.**
  - **Decreasing Instruction count means each instruction is doing more; hence either CPI or cycle time or both, may increase.**
  - **A smart compiler may decrease CPI by choosing the right *kind* of instructions, without a large increase in Instruction count.**

IBM
מ ב י

Compiler technology has a major impact on total performance

Example:

A compiler writer must choose between two code sequences for a certain high level language statement. Instruction counts for the two sequences are as follows:

	instruction counts per type		
sequence	A	B	C
1	2	1	2
2	4	1	1

- **Which sequence executes more instructions ?**
- **Which has lower CPI ?**
- **Which is faster ?**

example (continued)

Hardware specifications give the following CPI:

instruction type	CPI per instruction type
A	1
B	2
C	3

Use the formula:

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

- **Instruction count 1 = 2+1+2 = 5; Instruction count 2 = 4+1+1 = 6.**
- **Total cycles 1 = (2x1)+(1x2)+(2x3) = 10 ; total cycles 2 = (4x1)+(1x2)+(1x3) = 9.**
- ***Sequence 2 is faster* .**
- **CPI 1 = 10/5 = 2; CPI 2 = 9/6 = 1.5**

IBM
מ ב י

CPI

When calculating CPI from dynamic instruction count data, a useful formula is:

$$\text{CPI} = \sum_{i=1}^{T} w_i \, \text{CPI}_i$$

Where:

$$w_i = \frac{\text{Icount}_i}{\text{Icount}}$$

T = the number of instruction types

IBM
מ ב י

MIPS -- a popular performance metric

$$\textbf{MIPS} = \frac{\text{Instruction count}}{\text{CPU time X } \mathbf{10^6}} = \frac{\text{Instruction count}}{\text{IC X CPI X Clock cycle time X } \mathbf{10^6}} = \frac{\text{IC X Clock rate}}{\text{IC X CPI X } \mathbf{10^6}}$$

$$\textbf{so MIPS} = \frac{\text{Clock rate}}{\text{CPI X } \mathbf{10^6}}$$

also called *Native* MIPS.

In fact, MIPS can be very misleading because it leaves out one of the 3 key factors in performance -- IC (Instruction count).

- **Faster machines means bigger MIPS (Execution Time = IC / (MIPS X $10^{6)}$).**
- **MIPS cannot be used to compare machines with different instruction sets.**
- **MIPS seems like it is *native* to the machine, but in fact, you cannot count instructions without choosing some subset of the instruction set to execute. Thus MIPS just hides an arbitrary choice of instructions. *MIPS cannot compare different programs on the same computer.***
- **MIPS can vary inversely with performance (next slide).**

IBM
מבי

Example

We have the following instruction count data from two different compilers running the same program (clock rate = 100 MHz).

instruction counts (millions) each type

code from	A	B	C
compiler 1	5	1	1
compiler 2	10	1	1

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

- Which sequence executes more instructions ?
- Which has lower CPI ?
- Which is faster ?

CPI for each instruction type is the same as the previous example. To use our formula for MIPS, we need the CPI.

- $$\text{CPI 1} = \frac{((5x1)+(1x2)+(1x3)) \text{ X } 10^6}{(5+1+1) \text{ X } 10^6} = \frac{10}{7} = 1.428$$
- $$\text{CPI 2} = \frac{((10x1)+(1x2)+(1x3)) \text{ X } 10^6}{(10+1+1) \text{ X } 10^6} = \frac{15}{12} = 1.25$$

IBM
מבי

Example (continued)

- **MIPS 1 = 100 MHz / (1.428 X 10^6) = 70**
- **MIPS 2 = 100 MHz / (1.25 X 10^6) = 80**

But Compiler 1 is obviously faster,

because CPU time is $\frac{\text{Instruction count}}{\text{MIPS X } 10^6}$

so:

- **CPU time 1 =** $\frac{7 \text{ X } 10^6}{70 \text{ X } 10^6}$ **= 0.10 seconds**
- **CPU time 2 =** $\frac{12 \text{ X } 10^6}{80 \text{ X } 10^6}$ **= 0.15 seconds**

- **Peak MIPS → MIPS rating at minimal CPI; completely unrealistic**
- **Relative MIPS →** $\frac{\text{CPU Time}_{reference}}{\text{CPU Time}_{target}}$ **X** $\text{MIPS}_{reference}$

✡ **depends on program; needs reference machine**

IBM
יבמ

MegaFlops (MFLOPS)

$$\text{MFLOPS} = \frac{\text{FP operations in program}}{\text{CPU time X } 10^6}$$

- **Use *normalization* to achieve a fair measure of total work done.**
  - **different machines have different FP operations.**
  - **different FP ops take different amounts of time.**
- **e.g. add = 1 normalized FP operation; mult =2; div =4; func (sin, cos) = 8 etc.**
- **MFLOPs is only meaningful for certain programs;**
  - **compilers have an MFLOPs rating of near zero, for any machine.**

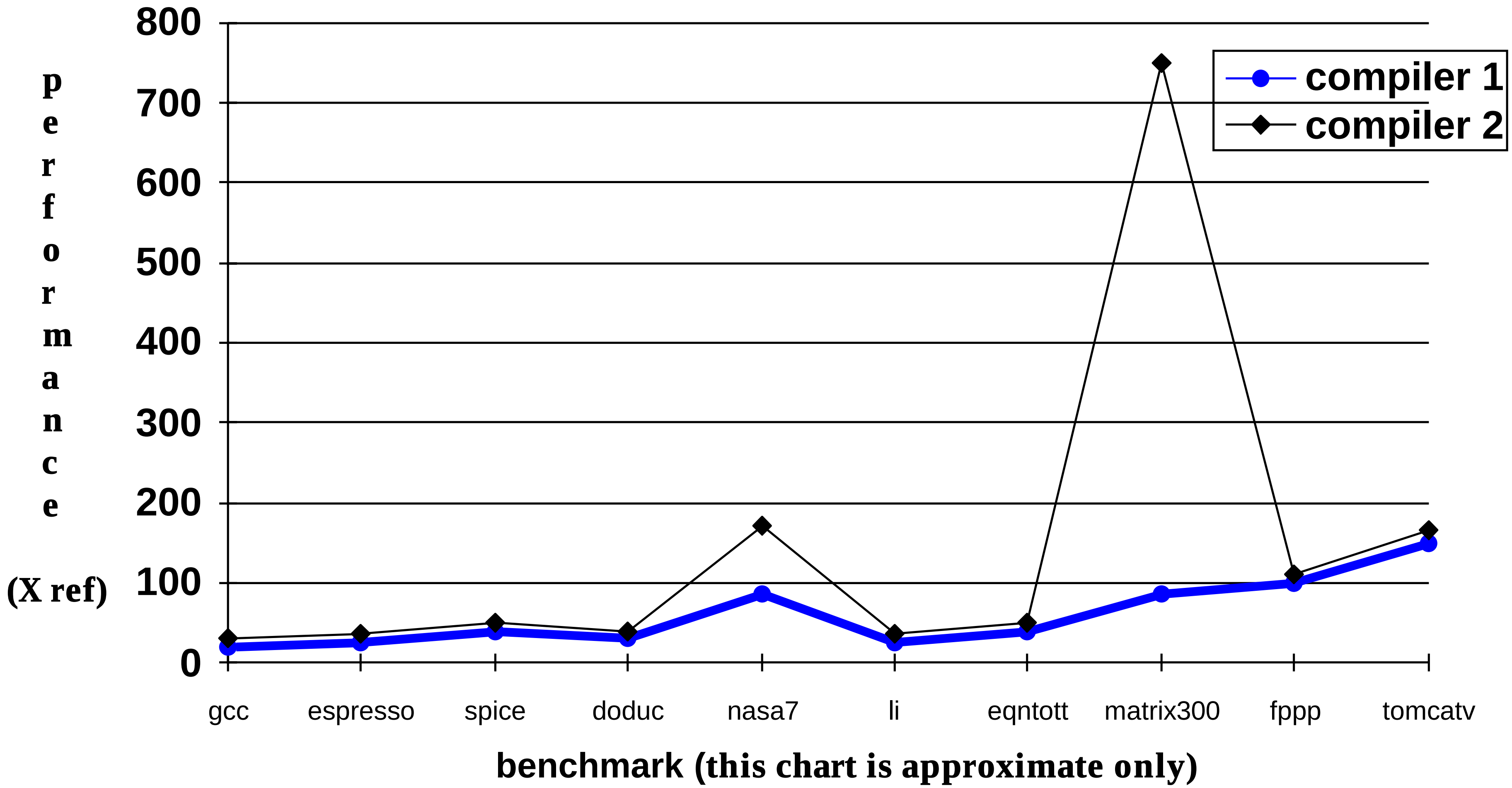
- **Best version of MFLOPs (normalized, program specified) is basically a measure of work per unit time.**
  - **Tempting to generalize to different programs, but this is false.**

IBM
מבי

SPEC

- **SPEC is Standard Performance Evaluation Corporation**
- **SPEC's mission: *To establish, maintain, and endorse a standardized set of relevant benchmarks and metrics for performance evaluation of modern computer systems*.**
- **User community can benefit greatly from an objective series of applications-oriented tests, which can serve as common reference points and be considered during the evaluation process.**
- **While no one benchmark can fully characterize the overall system performance, the results of a variety of realistic benchmarks can give valuable insight into expected real performance.**
- **Legally, SPEC is a non-profit corporation registered in California.**

SPEC Performance



SPEC performance ratios for IBM PowerStation 550 -- two compilers.

This result from SPEC reports illustrates how misleading a performance measure can be when based on a small, unrealistic program. Matrix300 is Matrix mult code which runs 99% of time in a single line. Compiler blocks the code to avoid memory accesses; effectiveness of technique will be much lower in real code. Also, reflects nothing about machine.

IBM
מ ב י

Amdahl's Law

make the common case fast -- why?

Denote part of system that was enhanced as the *enhanced fraction* or $F_{enhanced}$.

$$\text{Speedup} = \frac{\text{CPU Time}_{old}}{\text{CPU Time}_{new}} = \frac{\text{CPU Time}_{old}}{\text{CPU Time}_{old}\,(1 - F_{enhanced}) + \text{CPU Time}_{old}\,F_{enhanced}\,(1/\,\text{speedup}_{enhanced})} =$$

$$= \frac{1}{(1 - F_{enhanced}) + \frac{F_{enhanced}}{\text{speedup}_{enhanced}}}$$

Example

Suppose we have a technique for improving the performance of FP operations by a factor of 10. What fraction of the code must be floating point to achieve a 300% improvement in performance?

$$3 = \frac{1}{(1 - F_{enhanced}) + \frac{F_{enhanced}}{10}} \quad \Longrightarrow \quad F_{enhanced} = 20/27 = 74\%$$

Even dramatic enhancements make a limited contribution unless they relate to a very common case.

IBM
מ ב י