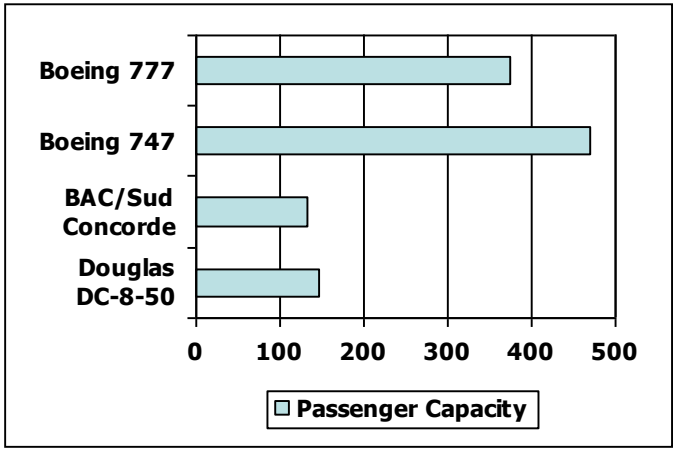
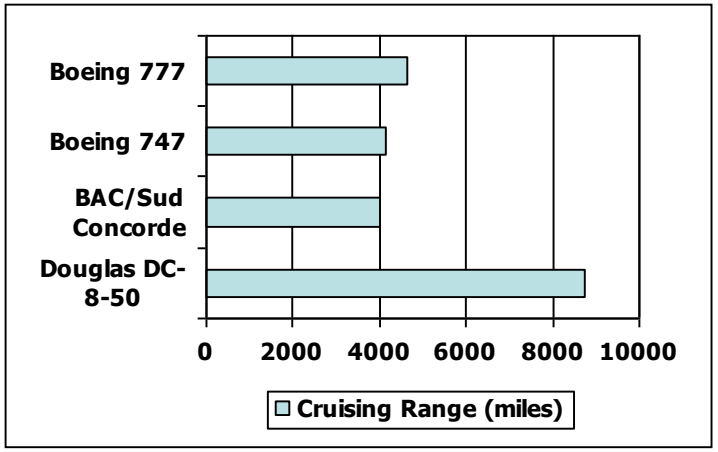
EE3009 Computer Architecture

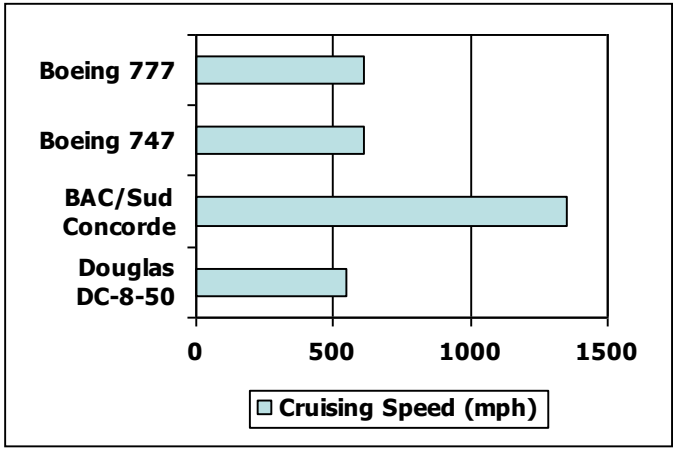
Lecture # 02 - Measuring Performance I

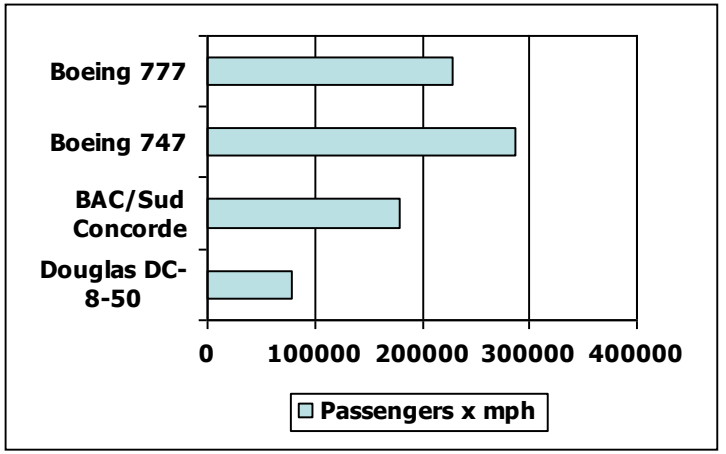
Defining Performance

- Which airplane has the best performance?









response time

Also called **execution time**. The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, and so on.

throughput

Also called **bandwidth**. Another measure of performance, it is the number of tasks completed per unit time.

Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

Performance

- For some program running on machine X,

$$\text{Performance}_X = 1 / \text{Execution time}_X$$

- Comparative (relative) Performance:

"X is n times faster than Y"

$$\text{Performance}_X \ / \ \text{Performance}_Y = n$$

CPU execution time

Also called **CPU time**. The actual time the CPU spends computing for a specific task.

user CPU time

The CPU time spent in a program itself.

system CPU time

The CPU time spent in the operating system performing tasks on behalf of the program.

Review: Time & Clock Metrics

- Determine effect of design change on performance
- CPU execution time = CPU clock cycles X clock cycle time
- CPU execution time = CPU clock cycles/clock rate

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

CPU Performance Equation

- # of clock cycles is not same for each instr.

- An alternative to "CPU clock cycles" is:
  - "number of instructions executed" or the Instruction Count (IC) of the program.
  - multiplied by the average time it takes for the instructions or the average Clocks Per Instruction (CPI).

- CPU Clock cycles = IC x CPI

$$\text{CPU time} = \text{IC} \times \text{CPI} \times \text{Clock cycle time} = \frac{\text{IC} \times \text{CPI}}{\text{Clock rate}}$$

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{instructio ns}}{\text{program}} \times \frac{\text{cycles}}{\text{instructio n}} \times \frac{\text{seconds}}{\text{cycle}}$$

Measuring performance parameters

- CPU execution time?
- Clock cycle time (or clock rate)?
- Instruction count?
- CPI?

CPI Example

- Suppose we have two implementations of the same instruction set architecture. Machine A has a clock cycle time of 1 ns and a CPI of 2.0 for some program, and machine B has a clock cycle time of 2 ns and a CPI of 1.2 for the same program.
- Which machine is faster for this program, and how much faster is it?

Reduce the CPU Time 1/2

$$\text{CPU time} = \frac{\text{IC} \times \text{CPI}}{\text{Clock rate}}$$

- Reduce instruction count
  - Which depends on instruction set architecture + compiler technology + algorithm + language selection.
- Reduce CPI
  - Which depends on system organization (pipelined or not) and instruction set architecture.
- Reduce clock cycle time (or increase rate)
  - Which depends on technology (VLSI/ULSI), implementation (hardwired vs. μ-programmed) and instruction set architecture.

Reduce the CPU Time 2/2

- Notice that reducing one measure may increase another.
- For example, one can reduce instruction count by the introduction of complex instructions which carry out several basic operations in a single instruction.
  - But, this will likely increase the clock cycles and CPI.

Variation of CPU Performance Equation

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

- where IC_i represents the number of times instruction i is executed in a program and CPI_i represents the number of clock cycles for instruction i.

- Why isn't CPI_i a constant ?

of Instructions Example

- A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions:

	Instruction Class		
	A	B	C
CPI	1	2	3

- The compiler writer is considering the following 2 code sequences:

	Instruction counts		
Code Sequence	A	B	C
1	2	1	2
2	4	1	1

- Which sequence executes less instructions? Which will be faster? What is the CPI for each sequence?

MIPS as a Performance Measure

- MIPS: Millions Instructions Per Second
- Sometimes used as performance metric
  - Faster machine $\Rightarrow$ larger MIPS
- MIPS specifies instruction execution rate

$$\text{MIPS} = \frac{\text{Instruction Count}}{\text{Execution Time} \times 10^6} = \frac{\text{Clock Rate}}{\text{CPI} \times 10^6}$$

- We can also relate execution time to MIPS

$$\text{Execution Time} = \frac{\text{Inst Count}}{\text{MIPS} \times 10^6} = \frac{\text{Inst Count} \times \text{CPI}}{\text{Clock Rate}}$$

Drawbacks of MIPS

Three problems using MIPS as a performance metric

1. Does not take into account the capability of instructions
   - Cannot use MIPS to compare computers with different instruction sets because the instruction count will differ
2. MIPS varies between programs on the same computer
   - A computer cannot have a single MIPS rating for all programs
3. MIPS can vary inversely with performance
   - A higher MIPS rating does not always mean better performance
   - Example in next slide shows this anomalous behavior

MIPS example

- Two different compilers are being tested on the same program for a **4 GHz** machine with three different classes of instructions: *Class A*, *Class B*, and *Class C*, which require 1, 2, and 3 cycles, respectively.
- The **instruction count** produced by the **first compiler** is 5 billion Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.
- The **second compiler** produces 10 billion Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.
- Which compiler produces a higher MIPS?
- Which compiler produces a better execution time?

Solution to MIPS Example

- First, we find the CPU cycles for both compilers
  - CPU cycles (compiler 1) = $(5\times1 + 1\times2 + 1\times3)\times10^9 = 10\times10^9$
  - CPU cycles (compiler 2) = $(10\times1 + 1\times2 + 1\times3)\times10^9 = 15\times10^9$
- Next, we find the execution time for both compilers
  - Execution time (compiler 1) = 10×10^9 cycles / 4×10^9 Hz = 2.5 sec
  - Execution time (compiler 2) = 15×10^9 cycles / 4×10^9 Hz = 3.75 sec
- Compiler1 generates faster program (less execution time)
- Now, we compute MIPS rate for both compilers
  - MIPS = Instruction Count / (Execution Time × 10^6)
  - MIPS (compiler 1) = $(5+1+1) \times 10^9 / (2.5 \times 10^6) = 2800$
  - MIPS (compiler 2) = $(10+1+1) \times 10^9 / (3.75 \times 10^6) = 3200$
- So, code from compiler 2 has a higher MIPS rating !!!

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)

Computer Performance Measures :

MFOLPS (Million FLOating-Point Operations Per Second)

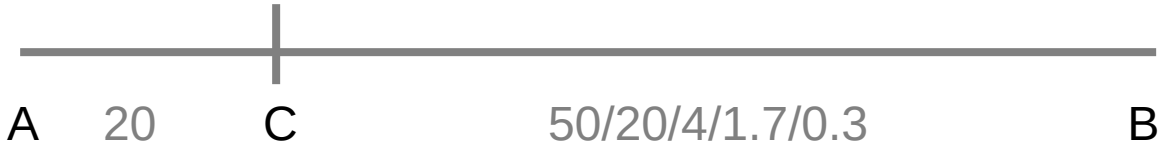
- MFLOPS is a better comparison measure between different machines than MIPS.
- Program-dependent: Different programs have different percentages of floating-point operations present. i.e compilers have no floating- point operations and yield a MFLOPS rating of zero.
- Dependent on the type of floating-point operations present in the program.

Evaluating Performance Gain

- **Common pitfall**: Expecting the improvement of one aspect of a computer to increase performance by an amount proportional to the size of the improvement.

Speedup Example 1

- Trip from point A to point B in two parts



A-C Trip	C-B Trip	Total Time	C-B Speedup	Overall Speedup
20	50	70	1	1
20	20	40	2.5	1.75
20	4	24	12.5	2.9
20	1.7	21.7	29.4	3.2
20	0.3	20.3	166.66	3.4

Speed Up= Execution Time Old/Execution Time New

Speedup Example 2

- A program runs in 100 secs on a computer with multiply operations responsible for 80 secs of this time.
- How much should we improve the speed of multiplication if we want the program to run 5times as fast?

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

For this problem:

$$\text{Execution time after improvement} = \frac{80 \text{ seconds}}{n} + (100 - 80 \text{ seconds})$$

$$20 \text{ seconds} = \frac{80 \text{ seconds}}{n} + 20 \text{ seconds}$$

$$0 = \frac{80 \text{ seconds}}{n}$$

Amdahl's Law

- The performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.
- This implies that the time consumed by events whose performance is not improved limits the effect of any improvement.
  - Lowest performer restricts all others.

Amdahl's Law….

The speedup of a feature is:

$$\text{Speedup} = \frac{\text{Performanc e using enhancemen t}}{\text{Performanc e without using enhancemen t}}$$

Amdahl's Law....

Overall Speedup Depends on two factors:

1. The fraction of the computation time in the original machine that can be converted to take advantage of the enhancement

$$\text{Fraction}_{\text{enhanced}} \leq 1$$

2. The feature improvement gained by the enhanced execution mode

$$\text{Speedup}_{\text{enhanced}} \geq 1$$

Amdahl's Law….

Exec time$_{new}$ = execution time after some enhancement

Exec time$_{old}$ = execution time before any enhancement

Fraction$_{enhanced}$ = fraction of work using the enhancement

Speedup$_{enhanced}$ = speedup of enhanced mode

$$\text{Exec time}_{new} = \text{Exec time}_{old} \times \left(\underbrace{\left(1 - Fraction_{enhanced}\right)}_{\text{Time spent using UNenhanced mode}} + \underbrace{\frac{Fraction_{enhanced}}{Speedup_{enhanced}}}_{\text{Time spent using enhancement}} \right)$$

4) Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left[\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{\left(1 - Fraction_{enhanced}\right)}$$

Amdahl's Law Example 1

- Suppose that we are considering an enhancement to the processor of a server system used for web serving. The new CPU is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original CPU is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?

Amdahl's Law example

- **New CPU 10X faster**
- **I/O bound server, so 60% time waiting for I/O**

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$= \frac{1}{\left(1 - 0.4\right) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

- **Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster**

Amdahl's Law Corollary

- Make the common case fast
  - Favor the frequent case over the infrequent case.
  - The frequent case is usually simpler anyway.
  - For example, given that overflow in addition is infrequent, favor optimizing the case when no overflow occurs.
- Objective
  - Determine the frequent case.
  - Determine possible feature improvement to make it fast.

Amdahl's Law can be used to quantify the improvement given that we have information concerning the feature needing enhancement.

Corollary

- If an enhancement is only usable for a fraction of a task, we can't speed up the task by more than the reciprocal of 1 minus that fraction.

$$\text{Performance Improvement Limit} \leq \frac{1}{1 - \text{Fraction}_{\text{enhanced}}}$$

Amdahl's Law Example 2

- Consider an enhancement that takes 20ns on a machine with enhancement and 100ns on a machine without. Assume enhancement can only be used 30% of the time.
- What is the overall speedup?
- Assume the same enhancement was made to a feature that was used 70% of the time.
- What is the overall speedup now?

Amdahl's Law Example 2

- **Case 1**
- $Fraction_{enhanced}$= 0.3
- $Speedup_{enhanced}$= 100/20=5

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

- $Speedup_{overall}$=1/ { (1-0.3)+ 0.3/5} = 1.32
- **Case 2:**
- $Speedup_{overall}$=1/ { (1-0.7)+ 0.7/5} = 2.27

Example

- Frequency of FP instructions : 25%
- Average CPI of FP instructions : 4.0
- Average CPI of other instructions : 1.33
- Frequency of FPSQR = 2%
- CPI of FPSQR = 20
- Design Alternative 1: Reduce CPI of FPSQR from 20 to 2.
- Design Alternative 2: Reduce average CPI of all FP instruction to 2.
- Compare these two design alternatives using CPU Performance equation.

Case 1- using FP to implement FPSQR

25% (2% FPSQR included)

Case 2- Add a FPSQR Module that responsible of 2% overall performance

Solution

$$CPI_{original} = \frac{\left(\sum_{i=1}^{n} CPI_i \times IC_i\right)}{Instruction\_Count} = \sum_{i=1}^{n} CPI_i \times \left(\frac{IC_i}{Instruction\_Count}\right)$$

$CPI_{original} = (4 \times 25\%) + (1.33 \times 75\%) = 2.0$

$CPI_{new\ FP} = (2.0 \times 25\%) + (1.33 \times 75\%) = 1.5$

CPI enhanced

When add an enhanced FPSQR

$CPI_{new\ FPSQR} = CPI_{original} - (CPI_{old\ FPSQR} - CPI_{of\ new\ FPSQR}) \times 2\%$

$= 2.0 - (20-2) \times 2\% = 1.64$

$Speedup_{new\ FP} = CPU_{time\ original} / CPU_{time\ new\ FP}$

$= IC \times Clock\ cycle \times CPI_{original} / IC \times Clock\ cycle \times CPI_{new\ FP}$

$= 2.0/1.5 = 1.33$

$Speedup_{new\ FPSQ} = 2.0/1.64 = 1.22$

Summary: Performance

- CPU Performance Metric: TIME

  $\text{Performance}_X = 1 / \text{Execution time}_X$

- Instead of using seconds to measure execution time, often we use clock cycles:

  CPU exec time = CPU clock cycles/clock rate

- Instead of using CPU clock cycles we use Instruction count and CPI:

  CPU exec time = IC x CPI /clock rate

- Effect of hardware/software changes can be tracked through above equation.
- Amdahl's Law: Lowest performer restricts all others
- Corollary of Amdahl's Law: Make the common case fast! It is simpler to speed up anyway!
- Other considerations?

Performance Summary

- **KEY IDEA: Performance determined by execution time**
- Do any of the other variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?
  - MFLOPS?
  - MIPS? {see example on p. 268 of text}

- **Common pitfall**: thinking one of the variables is indicative of performance when it really isn't.

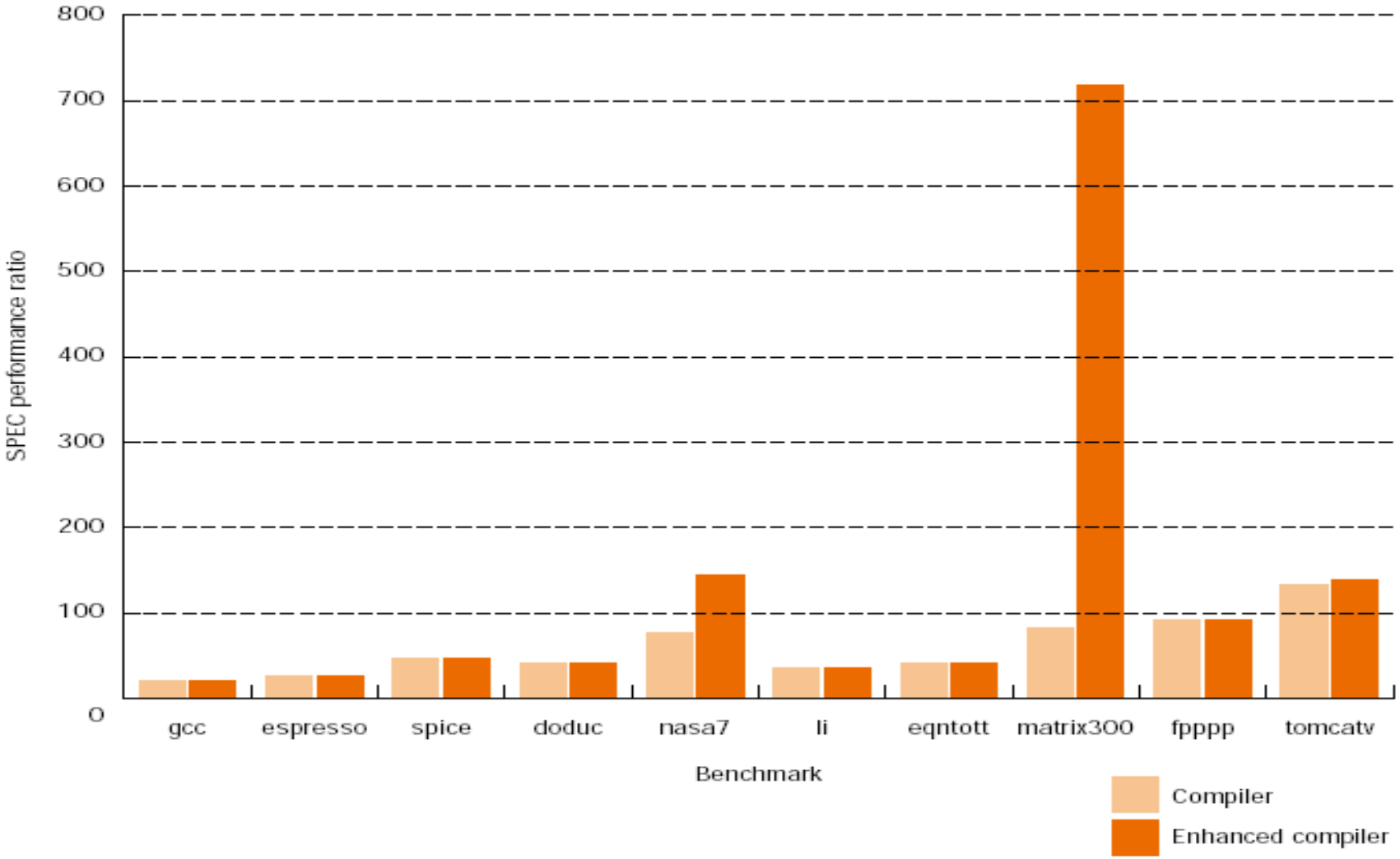
Evaluating Performance

- How to measure Performance?
- How to report and summarize Performance?
- Workload (real applications)

  vs.
- Benchmarks (programs specifically chosen to measure performance)

SPEC Benchmarks

(System Performance Evaluation Cooperative) suite of benchmarks



Performance Report

- Key idea is Reproducibility
- Report consists of:
  - System description or machine configuration (CPU, FPU, # of CPU, cache and memory size, etc.)
  - Operating System and Compiler (OS type and rev, compiler rev, file system)
  - Workload & benchmarks
  - Choice of data input