Performance
9 ways to fool the public
1.5-1.6

#1 – Reporting Results

#2 - Choosing Applications

#3 - Choosing Applications

Benchmark suites
• SPEC-fp – scientific codes
• SPEC-int – integer codes (irregular, small codes)
• TPC – database benchmarks
• EEMBC – embedded proc benchmarks

#4 - Running Applications

• Solution:
  • www.tomshardware.com
#5 – Reporting Results

- Solution:

<table>
<thead>
<tr>
<th>Speedup</th>
<th>Comp A</th>
<th>Comp B</th>
<th>Comp C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>P2</td>
<td>1000</td>
<td>100</td>
<td>20</td>
</tr>
</tbody>
</table>

- On P1, A is ____ x’s as fast as B
- On P1, B is ____ x’s as fast as C
- On P2, C is ____ x’s as fast as A
- On P1, B is ____ x’s as fast as A
- Which is fastest?!?

#6 – Floating 0 on graph

#6 – Normalized speedup, 0 to 1…

All execution times are divided by North’s execution time to obtain speedup

#7 – Reporting Results

- Manipulating average results

- Solution:

<table>
<thead>
<tr>
<th>Speedup</th>
<th>Comp A</th>
<th>Comp B</th>
<th>Comp C</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>P2</td>
<td>1000</td>
<td>100</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arithmetic Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric Mean</td>
</tr>
<tr>
<td>Weighted Mean</td>
</tr>
<tr>
<td>(60/40)</td>
</tr>
</tbody>
</table>
#8 – Reporting Results

Amdahl’s Law

• Speedup =

Solution:

Amdahl’s Law

• Helps target ______________

• Optimize the ______________

• Allow ______________ to suffer

Amdahl’s Law Example

Suppose multiplication operations constitute 80% of the execution of a program. How much improvement do we need in the multiply to get a 3x speedup in the program?

What is the most possible speedup by improving only the multiply?

#9 – Misusing alternate metrics

CPU time

• IPC = Instructions per Cycle
• CPI = Cycles per Instruction
• MIPS = Millions of Instructions per Second
• Clock Rate = # clock ticks / unit time
• Cycle Time = time between clock ticks
• Clock Rate = 1 / Cycle Time
• IPC = 1 / CPI

• = #Insts * Clock cycle time * CPI
• = #Inst * CPI / Clock rate
• = #Inst * Clock cycle time / IPC
• = #Inst / (Clock rate * IPC)

• Comparing two machines – if clock rate is equal, and programs are identical, compare only with IPC!
Using CPI

We have to code sequences involving recalculation of the same values. The compiler writer can choose between having a code sequence of 7 instructions to recalculate three values, or temporarily storing them in the stack. What are the total cycles of the code sequences?

<table>
<thead>
<tr>
<th>Code Sequence</th>
<th>Loads</th>
<th>Stores</th>
<th>ALU Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

Which code sequence should the compiler choose?

Using Execution Time

Our old 1GHz machine runs our program in 20 seconds. We are designing a new machine, and the target execution time is 10 seconds. Through various architectural innovations, we can increase the clock rate. Unfortunately, this increase comes at a price – we can pump the clock rate up to 2GHz, but the new execution takes 1.2 times as many cycles. Is this enough? What is the clock rate necessary for our performance target?

Cheating MIPS

Compiler 1 is a conventional compiler. We compare it against Compiler 2, which randomly inserts nops into the code. What are their relative MIPS ratings? What are their relative execution times?

<table>
<thead>
<tr>
<th>Instruction Class -</th>
<th>A (1 CPI)</th>
<th>B (2 CPI)</th>
<th>C (3 CPI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler 1</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Compiler 2</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• Exec Time 1:

• Exec Time 2:

• MIPS 1:

• MIPS 2: