Performance and Cost

- Which of the following airplanes has the best performance?

<table>
<thead>
<tr>
<th>Airplane</th>
<th>Passengers</th>
<th>Range (mi)</th>
<th>Speed (mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 737-100</td>
<td>101</td>
<td>630</td>
<td>598</td>
</tr>
<tr>
<td>Boeing 747</td>
<td>470</td>
<td>4150</td>
<td>610</td>
</tr>
<tr>
<td>BAC/Sud Concorde</td>
<td>132</td>
<td>4000</td>
<td>1350</td>
</tr>
<tr>
<td>Douglas DC-8-50</td>
<td>146</td>
<td>8720</td>
<td>544</td>
</tr>
</tbody>
</table>

- How much faster is the Concorde vs. the 747
- How much bigger is the 747 vs. DC-8?
Performance and Cost

• Which computer is fastest?
• Not so simple
  – Scientific simulation – FP performance
  – Program development – Integer performance
  – Database workload – Memory, I/O
Performance of Computers

• Want to buy the fastest computer for what you want to do?
  – Workload is all-important
  – Correct measurement and analysis

• Want to design the fastest computer for what the customer wants to pay?
  – Cost is an important criterion
Defining Performance

• What is important to whom?
• Computer system user
  – Minimize elapsed time for program = time_end – time_start
  – Called response time
• Computer center manager
  – Maximize completion rate = #jobs/second
  – Called throughput
Response Time vs. Throughput

• Is throughput = 1/av. response time?
  – Only if NO overlap
  – Otherwise, throughput > 1/av. response time
  – E.g. a lunch buffet – assume 5 entrees
  – Each person takes 2 minutes/entree
  – Throughput is 1 person every 2 minutes
  – BUT time to fill up tray is 10 minutes
  – Why and what would the throughput be otherwise?
    • 5 people simultaneously filling tray (overlap)
    • Without overlap, throughput = 1/10
What is Performance for us?

• For computer architects
  – CPU time = time spent running a program

• Intuitively, bigger should be faster, so:
  – Performance = 1/X time, where X is response, CPU execution, etc.

• Elapsed time = CPU time + I/O wait

• We will concentrate on CPU time
Improve Performance

• Improve (a) response time or (b) throughput?
  – Faster CPU
    • Helps both (a) and (b)
  – Add more CPUs
    • Helps (b) and perhaps (a) due to less queueing
Performance Comparison

- Machine A is n times faster than machine B iff
  \[ \frac{\text{perf}(A)}{\text{perf}(B)} = \frac{\text{time}(B)}{\text{time}(A)} = n \]

- Machine A is x% faster than machine B iff
  \[ \frac{\text{perf}(A)}{\text{perf}(B)} = \frac{\text{time}(B)}{\text{time}(A)} = 1 + \frac{x}{100} \]

- E.g. time(A) = 10s, time(B) = 15s
  \[ \frac{15}{10} = 1.5 \Rightarrow \text{A is 1.5 times faster than B} \]
  \[ \frac{15}{10} = 1.5 \Rightarrow \text{A is 50% faster than B} \]
Breaking Down Performance

• A program is broken into instructions
  – H/W is aware of instructions, not programs
• At lower level, H/W breaks instructions into cycles
  – Lower level state machines change state every cycle
• For example:
  – 1GHz Snapdragon runs 1000M cycles/sec, 1 cycle = 1ns
  – 2.5GHz Core i7 runs 2.5G cycles/sec, 1 cycle = 0.25ns
Iron Law

Processor Performance = \frac{\text{Time}}{\text{Program}}

= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}

= \text{(code size)} \times \text{(CPI)} \times \text{(cycle time)}

Architecture --> Implementation --> Realization

Compiler Designer      Processor Designer         Chip Designer
Iron Law

• Instructions/Program
  – Instructions executed, not static code size
  – Determined by algorithm, compiler, ISA

• Cycles/Instruction
  – Determined by ISA and CPU organization
  – Overlap among instructions reduces this term

• Time/cycle
  – Determined by technology, organization, clever circuit design
Our Goal

• Minimize time which is the product, NOT isolated terms

• Common error to miss terms while devising optimizations
  – e.g. ISA change to decrease instruction count
  – BUT leads to CPU organization which makes clock slower

• Bottom line: terms are inter-related
Other Metrics

• MIPS and MFLOPS
• MIPS = instruction count/(execution time x 10^6)
  = clock rate/(CPI x 10^6)
• But MIPS has serious shortcomings
Problems with MIPS

- E.g. without FP hardware, an FP op may take 50 single-cycle instructions
- With FP hardware, only one 2-cycle instruction

Thus, adding FP hardware:
- CPI increases (why?)
- Instructions/program decreases (why?)
- Total execution time decreases

BUT, MIPS gets worse!

50/50 => 2/1
50 => 1
50 => 2
50 MIPS => 2 MIPS
Problems with MIPS

• Ignores program
• Usually used to quote peak performance
  – Ideal conditions => guaranteed not to exceed!
• When is MIPS ok?
  – Same compiler, same ISA
  – E.g. same binary running on AMD Phenom, Intel Core i7
  – Why? Instr/program is constant and can be ignored
Other Metrics

• MFLOPS = FP ops in program/(execution time x 10^6)

• Assuming FP ops independent of compiler and ISA
  – Often safe for numeric codes: matrix size determines # of FP ops/program
  – However, not always safe:
    • Missing instructions (e.g. FP divide)
    • Optimizing compilers

• Relative MIPS and normalized MFLOPS
  – Adds to confusion
Rules

• Use ONLY Time
• Beware when reading, especially if details are omitted
• Beware of Peak
  – “Guaranteed not to exceed”
Iron Law Example

• Machine A: clock 1ns, CPI 2.0, for program x
• Machine B: clock 2ns, CPI 1.2, for program x
• Which is faster and how much?

\[
\text{Time/Program} = \text{instr/program} \times \text{cycles/instr} \times \text{sec/cycle}
\]

\[
\text{Time(A)} = N \times 2.0 \times 1 = 2N
\]

\[
\text{Time(B)} = N \times 1.2 \times 2 = 2.4N
\]

Compare: \[
\frac{\text{Time(B)}}{\text{Time(A)}} = \frac{2.4N}{2N} = 1.2
\]

• So, Machine A is 20% faster than Machine B for this program
Iron Law Example

Keep clock(A) @ 1ns and clock(B) @ 2ns
For equal performance, if CPI(B)=1.2, what is CPI(A)?

\[
\frac{\text{Time}(B)}{\text{Time}(A)} = 1 = \frac{(N \times 2 \times 1.2)}{(N \times 1 \times \text{CPI}(A))}
\]

\[
\text{CPI}(A) = 2.4
\]
Iron Law Example

- Keep CPI(A)=2.0 and CPI(B)=1.2
- For equal performance, if clock(B)=2\text{ns}, what is clock(A)?

\[
\frac{\text{Time(B)}}{\text{Time(A)}} = 1 = \frac{N \times 2.0 \times \text{clock(A)}}{N \times 1.2 \times 2}
\]

\[
\text{clock(A)} = 1.2\text{ns}
\]
Which Programs

• Execution time of what program?
• Best case – your always run the same set of programs
  – Port them and time the whole workload
• In reality, use benchmarks
  – Programs chosen to measure performance
  – Predict performance of actual workload
  – Saves effort and money
  – Representative? Honest? Benchmarketing...
How to Average

<table>
<thead>
<tr>
<th></th>
<th>Machine A</th>
<th>Machine B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Program 2</td>
<td>1000</td>
<td>100</td>
</tr>
<tr>
<td>Total</td>
<td>1001</td>
<td>110</td>
</tr>
</tbody>
</table>

• One answer: for total execution time, how much faster is B? 9.1x
How to Average

• Another: arithmetic mean (same result)
• Arithmetic mean of times:
  • AM(A) = 1001/2 = 500.5
  • AM(B) = 110/2 = 55
  • 500.5/55 = 9.1x
• Valid only if programs run equally often, so use weighted arithmetic mean:

\[
\left\{ \sum_{i=1}^{n} \frac{\text{time}(i)}{\text{weight}(i) \times \text{time}(i)} \right\} \times \frac{1}{n}
\]
Other Averages

• E.g., 30 mph for first 10 miles, then 90 mph for next 10 miles, what is average speed?
  • Average speed = (30+90)/2 \text{ WRONG}
  • Average speed = total distance / total time
    = (20 / (10/30 + 10/90))
    = 45 \text{ mph}
Harmonic Mean

• Harmonic mean of rates =

\[
\frac{n}{\sum_{i=1}^{n} \frac{1}{rate(n)}}
\]

• Use HM if forced to start and end with rates (e.g. reporting MIPS or MFLOPS)

• Why?
  – Rate has time in denominator
  – Mean should be proportional to inverse of sums of time (not sum of inverses)
Dealing with Ratios

<table>
<thead>
<tr>
<th></th>
<th>Machine A</th>
<th>Machine B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Program 2</td>
<td>1000</td>
<td>100</td>
</tr>
<tr>
<td>Total</td>
<td>1001</td>
<td>110</td>
</tr>
</tbody>
</table>

- If we take ratios with respect to machine A

<table>
<thead>
<tr>
<th></th>
<th>Machine A</th>
<th>Machine B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Program 2</td>
<td>1</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Dealing with Ratios

- Average for machine A is 1, average for machine B is 5.05
- If we take ratios with respect to machine B

<table>
<thead>
<tr>
<th>Program</th>
<th>Machine A</th>
<th>Machine B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>Program 2</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Average</td>
<td>5.05</td>
<td>1</td>
</tr>
</tbody>
</table>

- Can’t both be true!!!
- Don’t use arithmetic mean on ratios!
Geometric Mean

• Use geometric mean for ratios
• Geometric mean of ratios = \( \sqrt[n]{\prod_{i=1}^{n} \text{ratio}(i)} \)

• Independent of reference machine
• In the example, GM for machine a is 1, for machine B is also 1
  – Normalized with respect to either machine
But…

- GM of ratios is not proportional to total time
- AM in example says machine B is 9.1 times faster
- GM says they are equal
- If we took total execution time, A and B are equal only if
  - Program 1 is run 100 times more often than program 2
- Generally, GM will mispredict for three or more machines
Summary

• Use AM for times
• Use HM if forced to use rates
• Use GM if forced to use ratios

• Best of all, use unnormalized numbers to compute time
Benchmarks: SPEC2000

• System Performance Evaluation Cooperative
  – Formed in 80s to combat benchmarking
  – SPEC89, SPEC92, SPEC95, SPEC2000

• 12 integer and 14 floating-point programs
  – Sun Ultra-5 300MHz reference machine has score of 100
  – Report GM of ratios to reference machine
## Benchmarks: SPEC CINT2000

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>164.gzip</td>
<td>Compression</td>
</tr>
<tr>
<td>175.vpr</td>
<td>FPGA place and route</td>
</tr>
<tr>
<td>176.gcc</td>
<td>C compiler</td>
</tr>
<tr>
<td>181.mcf</td>
<td>Combinatorial optimization</td>
</tr>
<tr>
<td>186.crafty</td>
<td>Chess</td>
</tr>
<tr>
<td>197.parser</td>
<td>Word processing, grammatical analysis</td>
</tr>
<tr>
<td>252.eon</td>
<td>Visualization (ray tracing)</td>
</tr>
<tr>
<td>253.perlbmk</td>
<td>PERL script execution</td>
</tr>
<tr>
<td>254.gap</td>
<td>Group theory interpreter</td>
</tr>
<tr>
<td>255.vortex</td>
<td>Object-oriented database</td>
</tr>
<tr>
<td>256.bzip2</td>
<td>Compression</td>
</tr>
<tr>
<td>300.twolf</td>
<td>Place and route simulator</td>
</tr>
</tbody>
</table>
## Benchmarks: SPEC CFP2000

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>168.wupwise</td>
<td>Physics/Quantum Chromodynamics</td>
</tr>
<tr>
<td>171.swim</td>
<td>Shallow water modeling</td>
</tr>
<tr>
<td>172.mgrid</td>
<td>Multi-grid solver: 3D potential field</td>
</tr>
<tr>
<td>173.applu</td>
<td>Parabolic/elliptic PDE</td>
</tr>
<tr>
<td>177.mesa</td>
<td>3-D graphics library</td>
</tr>
<tr>
<td>178.galgel</td>
<td>Computational Fluid Dynamics</td>
</tr>
<tr>
<td>179.art</td>
<td>Image Recognition/Neural Networks</td>
</tr>
<tr>
<td>183.equake</td>
<td>Seismic Wave Propagation Simulation</td>
</tr>
<tr>
<td>187.facerec</td>
<td>Image processing: face recognition</td>
</tr>
<tr>
<td>188.ammp</td>
<td>Computational chemistry</td>
</tr>
<tr>
<td>189.lucas</td>
<td>Number theory/primality testing</td>
</tr>
<tr>
<td>191.fma3d</td>
<td>Finite-element Crash Simulation</td>
</tr>
<tr>
<td>200.sixtrack</td>
<td>High energy nuclear physics accelerator design</td>
</tr>
<tr>
<td>301.apsi</td>
<td>Meteorology: Pollutant distribution</td>
</tr>
</tbody>
</table>
Thank You