Performance of Computers

Which computer is fastest?
Not so simple
• scientific simulation - FP performance
• program development - Integer performance
• commercial work - I/O

Want to buy the fastest computer for what you want to do
• workload is important
Want to design the fastest computer for what they want to pay
• BUT cost is an important criterion

Forecast

Time and performance
Iron law
MIPS and MFLOPS
Which programs and how to average
Amdahl's law

Defining Performance

What is important to who
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
- with overlap, throughput > 1/av.response time
- e.g., a lunch buffet - assume 5 entrees
- each person takes 2 minutes at every 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?
  because there are 5 people (each at 1 entree), simultaneously; if there is no such overlap throughput = $1/10$

### Improve Performance

Improve (a) response time or (b) throughput?
- faster CPU
  - both (a) and (b)
- Add more CPUs
  - (b) but (a) may be improved due to less queueing

### What is Performance for us?

For computer architects
- CPU execution time = time spent running a program
  Because people like faster to be bigger to match intuition
  - performance = 1/X time
  - where X = response, CPU execution, etc.

Elasped time = CPU execution time + I/O wait
We will concentrate mostly on CPU execution time

### Performance Comparison

Machine A is n times faster than machine B iff
$$\text{perf}(A)/\text{perf}(B) = \text{time}(B)/\text{time}(A) = n$$

Machine A is x% faster than machine B iff
$$\text{perf}(A)/\text{perf}(B) = \text{time}(B)/\text{time}(A) = 1 + x/100$$

E.g., A 10s, B 15s
- $15/10 = 1.5 \Rightarrow A$ is 1.5 times faster than B
- $15/10 = 1 + 50/100 \Rightarrow A$ 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

E.g., 500 MHz PentiumIII runs 500M cycles/sec, 1 cycle = 2 ns
E.g., 2 GHz PentiumX will run 2G cycles/sec, 1 cycle = 0.5 ns

Our Goal

Minimize time which is the product, NOT isolated terms
  • E.g., ISA change to decrease instruction count
  • BUT leads to CPU organization which makes clock slower

Iron law

Time/program = instrs/program x cycles/instr x sec/cycle
sec/cycle (a.k.a. cycle time, clock time) - ‘heartbeat’ of computer
  • mostly determined by technology and CPU organization

cycles/instr (a.k.a. CPI)
  • mostly determined by ISA and CPU organization
  • overlap among instructions makes this smaller

instr/program (a.k.a. instruction count)
  • instrs executed NOT static code
  • mostly determined by program, compiler, ISA

Other Metrics

MIPS and MFLOPS

MIPS = instruction count/(execution time x 10^6 )

= clock rate/(CPI x 10^6 )

BUT MIPS has problems
Problems with MIPS

E.g., without FP H/W, an FP op may take 50 single-cycle instrs
with FP H/W only one 2-cycle instr

Thus adding FP H/W

- CPI increases (why?) **The FP op goes from 50/50 to 2/1**
- but instrs/prog decreases more (why?) each of the
  FP op reduces from 50 to 1, factor of 50
- total execution time decreases

- For MIPS
  - instrs/prog ignored
  - MIPS gets worse!

Other Metrics

\[
\text{MFLOPS} = \frac{\text{FP ops in program}}{\text{(execution time x } 10^6)}
\]

Assuming FP ops independent of compiler and ISA

- Assumption not true
- may not have divide in ISA
- optimizing compilers

Relative MIPS and normalized MFLOPS

- adds to confusion! (see book)

Problems with MIPS

Ignore program

Usually used to quote peak performance

- ideal conditions => guarantee not to exceed!!

When is MIPS ok?

- same compiler and same ISA
- e.g., same binary running on Pentium Pro and Pentium
- why? **instrs/prog is constant and may be ignored**

Rules

- Use ONLY Time
- Beware when reading, especially if details are omitted
- Beware of Peak
Iron Law Example

Machine A: clock 1 ns, CPI 2.0, for a program
Machine B: clock 2 ns, CPI 1.2, for same program

Which is faster and how much
Time/program = instrs/program x cycles/instr x sec/cycle
Time(A): $N \times 2.0 \times 1 = 2N$
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 CPI of A 2.0 and CPI of B 1.2
For equal performance, if clock of B is 2 ns, what is clock of A?
$\frac{\text{Time}(B)}{\text{Time}(A)} = 1 = \frac{(N \times 2.0 \times \text{clock}(A))}{(N \times 1.2 \times 2)}$
clock(A) = 1.2 ns

Which Programs

Execution time of what
Best case - you run the same set of programs everyday
  • port them and time the whole “workload”
In reality, use benchmarks
  • programs chosen to measure performance
  • predict performance of actual workload (hopefully)
    + saves effort and money
    – representative? honest?
How to average

Example (page 70)

<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: total execution time, then B is how much faster than A? \(9.1\)

How to average

Another: arithmetic mean (same result)

Arithmetic mean of times: \(\frac{\sum_{i=1}^{n} \text{time}(i)}{n}\) for \(n\) programs

AM(A) = 1001/2 = 500.5
AM(B) = 110/2 = 55
500.5/55 = 9.1

Valid only if programs run equally often, so use “weight” factors

Weighted arithmetic mean: \(\frac{\sum_{i=1}^{n} (\text{weight}(i) \times \text{time}(i))}{n}\)

Other Averages

E.g., 30 mph for first 10 miles
90 mph for next 10 miles. average speed?

Average speed = \(\frac{\text{total distance}}{\text{total time}}\)

- (\(20 / (10/30+10/90)\))
- 45 mph

Harmonic Mean

Harmonic mean of rates = \(\frac{1}{\frac{1}{\sum_{i=1}^{n} \text{time}(i)}}\)

Use HM if forced to start and end with rates

Trick to do arithmetic mean of times but using rates and not times
### Dealing with Ratios

E.g.,

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

If we take ratios, with respect to Machine A:

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

### Geometric Mean

Geometric mean of ratios is independent of reference machine (math property):

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

Use **GM** if forced to use ratios

In the example, GM for machine A is 1, for machine B is also 1

- normalized with respect to either machine

### But...

Geometric mean 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

Better yet, use unnormalized numbers to compute time

---

**SPEC95**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>AI, plays go</td>
</tr>
<tr>
<td>m88ksim</td>
<td>Motorola 88K chip simulator</td>
</tr>
<tr>
<td>gcc</td>
<td>GNU compiler</td>
</tr>
<tr>
<td>compress</td>
<td>Unix utility compresses files</td>
</tr>
<tr>
<td>li</td>
<td>Lisp Interpreter</td>
</tr>
<tr>
<td>jpeg</td>
<td>Graphic (de)compression</td>
</tr>
<tr>
<td>perl</td>
<td>Unix utility text processor</td>
</tr>
<tr>
<td>vortex</td>
<td>Database program</td>
</tr>
</tbody>
</table>

---

**Benchmarks: SPEC95**

System Performance Evaluation Cooperative

Latest is SPEC2K but text uses SPEC95

8 integer and 10 floating point programs

- normalize run time with a SPARCstation 10/40
- GM of the normalized times

---

**Some SPEC95 Programs**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>INT/FP</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m88ksim</td>
<td>Integer</td>
<td>Motorola 88K chip simulator</td>
</tr>
<tr>
<td>gcc</td>
<td>Integer</td>
<td>GNU compiler</td>
</tr>
<tr>
<td>compress</td>
<td>Integer</td>
<td>Unix utility compresses files</td>
</tr>
<tr>
<td>vortex</td>
<td>Integer</td>
<td>Database program</td>
</tr>
<tr>
<td>su2cor</td>
<td>FP</td>
<td>Quantum physics; Monte carlo</td>
</tr>
<tr>
<td>hydro2d</td>
<td>FP</td>
<td>Navier Stokes equations</td>
</tr>
<tr>
<td>mgrid</td>
<td>FP</td>
<td>3-D potential field</td>
</tr>
<tr>
<td>wave5</td>
<td>FP</td>
<td>Electromagnetic particle simulation</td>
</tr>
</tbody>
</table>
Amdahl’s Law

Why does the common case matter the most?

Speedup = old time/new time = new rate/old rate

Let an optimization speed f fraction of time by a factor of s

\[ \text{Spdup} = \frac{[(1-f) + f \times \text{oldtime}]}{[(1-f) \times \text{oldtime}] + f/s \times \text{oldtime}} = \frac{1}{1 - f + f/s} \]

Amdahl’s Law Example

Your boss asks you to improve Pentium Posterior performance by

- improve the ALU used 95% of time, by 10%
- improve the memory pipeline used 5%, by a factor of 10

Let f = fraction sped up and s = the speedup on that fraction

\[
\text{new_time} = (1-f) \times \text{old_time} + \frac{f}{s} \times \text{old_time}
\]

\[ \text{Speedup} = \frac{\text{new_rate}}{\text{old_rate}} = \frac{\text{old_time}}{\text{new_time}} = \frac{1}{1 - f + \frac{f}{s}} \]

Amdahl’s Law: Speedup = \( \frac{1}{1 - f + \frac{f}{s}} \)

<table>
<thead>
<tr>
<th>f</th>
<th>s</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>95%</td>
<td>1.10</td>
<td>1.094</td>
</tr>
<tr>
<td>5%</td>
<td>10</td>
<td>1.047</td>
</tr>
<tr>
<td>5%</td>
<td>∞</td>
<td>1.052</td>
</tr>
</tbody>
</table>

Amdahl’s Law: Limit

\[
\lim_{s \to \infty} \frac{1}{1 - f + \frac{f}{s}} = \frac{1}{1 - f} = \text{Make common case fast}
\]
Summary of Chapter 2

Time and performance: Machine A n times faster than Machine B
  • iff Time(B)/Time(A) = n

Iron Law: Time/prog = Instr count x CPI x Cycle time

Other Metrics: MIPS and MFLOPS
  • Beware of peak and omitted details

Benchmarks: SPEC95

Summarize performance: AM for time, HM for rate, GM for ratio

Amdahl's Law: Speedup = $1/(1 - f + f/s)$ - common case fast