Performance Evaluation

[Ch. 1] What is performance?

... of a car?

... of a car wash?

... of a TV?

How should we measure the performance of a computer?

• The response time (or wall-clock time) it takes to complete a task? Why isn’t this a good measure?

• The CPU time it takes to complete a task?

• The user CPU time it takes? How is this different from “CPU time”?

H&P use

• system performance to refer to elapsed time on an unloaded system, and

• CPU performance to refer to user CPU time on an unloaded system.

Amdahl’s Law

The improvement in performance (“speedup”) is limited by the part you cannot improve.

In the last class, we looked at improving performance by pipelining operations. What was the part we could not improve?
Speedup = \frac{\text{Performance of task with gizmo}}{\text{Performance of task without gizmo}}

or

Speedup = \frac{\text{Execution time of task without gizmo}}{\text{Execution time of task with gizmo}}

However, usually we can’t speed up (or “enhance”) the whole task. So we have to speed up only a part.

Speedup_{\text{enhanced}} = \text{Best-case speedup from gizmo alone}

Fraction_{\text{enhanced}} = \text{Fraction of task that gizmo can enhance}

Speedup_{\text{overall}} = \frac{\text{Execution time of task without gizmo}}{\text{Execution time of task with gizmo}}

= \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{old}} \times (1 - \text{Fraction}_{\text{enhanced}}) + \text{Execution time}_{\text{old}} \times \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}

= \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}

= \frac{1}{(1 - f) + \frac{f}{s}}

_Amdahl’s Law example_

You do simulation of jet plane wings. One run takes one week on your fastest computer.

You get this ad in your mailbox:

The _Acme Hyperbole_ is the largest supercomputer ever built, it has 100,000 processors (great!)
It costs $1 bazillion (not so great)

Now, 1 week is 600,000 sec, so

- You could run a simulation in 6 seconds, right?
- Well, not all of a program can be done at the same time

Data dependencies:
\[ x = (\ldots), \text{ followed by } (\ldots) = x \times y \]

Control dependencies:
\[
\text{if } xxx \text{ then } yyy \text{ else } zzz
\]

Say 80% of your program is parallelizable (pretty good).

How fast would your program finish?

\[
\text{Speedup}_{\text{enhanced}} = \frac{1}{1 - \text{Fraction}_{\text{enhanced}}} + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}
\]

\[
= \frac{1}{1 - 0.8} + \frac{0.8}{100000}
\]

\[ \approx \frac{1}{0.2} = 5. \]

So the program runs approximately 5 times faster, finishing in ______ - not quite as great as one would hope.

Worth a bazillion dollars?

Let’s take another look at Amdahl’s law, from the perspective that not all work is parallelizable.

Recall—

- Speedup is limited by the part you cannot improve.
- The common case matters most.
Case 1: Suppose $f = 0.95$ and $s = 1.1$. What kind of speedup can we get?

$$s_{overall} = \frac{1}{(1-0.95) + \frac{0.95}{1.10}} =$$

Case 2: Suppose $f = 0.05$ and $s = 10$.

$$s_{overall} = \frac{1}{(1-0.05) + \frac{0.05}{10}} =$$

Case 3: Suppose $f = 0.05$, $s \rightarrow \infty$.

$$s_{overall} = \frac{1}{(1-0.05) + \varepsilon} \approx$$

Workload selection

What workloads do we use to evaluate performance?

Observations

- A database search does “different things” from an FFT
- Hardware good for searching databases isn’t good for an FFT.

Solutions

- Ask the users
- Guess—

- Standards:

Benchmark suites

- SPEC89, SPEC92, SPEC95, SPEC2K (for workstations)
  - Includes code and inputs
- Transaction Process Council TPC-A, B, C, D
  (web/database servers)
  - No code, just a “specification”
• Perfect Club (for supercomputers)
  Code, but you can rewrite it as long as results are same

**Graveyard of failed metrics**

MIPS (million instructions per second)

\[
\text{MIPS} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{program time}}{10^6}
\]

• Instruction sets are not the same across different vendors’ machines

• MIPS can be inversely proportional to performance!
  (consider FP hw vs. software emulation)

MFLOPS (Million floating-point operations per second)

• The set of FP instructions is not consistent across machines (A Pentium has a divide, Cray C90 supercomputer does not)

• Integer-only code (e.g., a compiler) has a zero MFLOPS rating

“Peak performance” (maximum performance for a synthetic string of instructions)

• Example: The DEC Alpha microprocessor has a peak performance of 1.2 “BIPS”

• When compared using benchmarks, the actual rate is closer to 360 to 750 (DEC Alpha) MIPS

So, what metric **should** we use?

**Run time**

Run time—the only unimpeachable measure of performance for processors.

But, there are many ways to interpret run time.
What we care about:

- Single program compute time
- Fraction of system time due to single program

Compute time = “CPU time”

CPU time = clock cycle count $\times$ cycle time

$$CPU \text{ time} = \frac{\text{clock cycle count}}{\text{clock rate (MHz)}}$$

Cycles per instruction = CPI = $\frac{\text{clock cycle count}}{\text{instruction count}}$

So, clock cycle count =

And, CPU time =

We can improve CPU time in three ways.

- Decrease instruction count (IC)
  
  Good compiler
  
  Better software algorithms

- Decrease CPI (increase “IPC”, a.k.a. inst. level parallelism, or ILP)
  
  Fancy hardware (e.g., caches, branch prediction, pipelining, superscalar)
Good compiler
• Decrease CT
  Deeper pipelining & really good circuit design
  Technology scaling
  Simple ISA; Less aggressive ILP (simple microarchitecture)

The real story on RISC vs. CISC

RISC: Simple instructions
• Takes a lot of them to do anything: Increases
• Easier to build hardware:
• Easier to parallelize:
Net effect:
  Need a lot of memory to hold program, but…
  Runs faster if Inc(IC) < Dec(CT) and Dec(CPI).

CISC: Big honking complex instructions
• Takes very few to do anything: Decreases IC
• Easier to program by hand
• Harder to build fast hardware: Incr. CT
• Harder to parallelize: Increases CPI

Net effect:
• Memory efficient
• Runs faster if Dec(IC) > Inc(CT) and Inc(CPI)

Retrospective
If memory is expensive, people hand code machines, and compilers are terrible ⇒ Use

If memory is inexpensive, no one hand codes, and compilers are terrific ⇒ Use

**Which computer is faster?**

<table>
<thead>
<tr>
<th></th>
<th>Computer A</th>
<th>Computer B</th>
<th>Computer C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program P1 (sec)</td>
<td>1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Program P2 (sec)</td>
<td>1000</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>Total time (sec)</td>
<td>1001</td>
<td>110</td>
<td>40</td>
</tr>
</tbody>
</table>

- A is 10x faster than B for P1
- B is 10x faster than A for P2
- A is 20x faster than C for P1
- C is 50x faster than A for P2
- etc.

Total execution time gives the clearest picture:

- B is 1001/110 = 9.1x faster than A for both programs
- C is 25x faster than A for both programs
- C is 2.75x faster than B for both programs

Which would you buy? (Answer: C is fastest, overall)

The *arithmetic mean* of times is a good measure too

**Example of means**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prog 1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Prog 2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Harmonic mean</td>
<td>4</td>
<td>3.11</td>
</tr>
<tr>
<td>Arithmetic mean</td>
<td>4</td>
<td>4.5</td>
</tr>
</tbody>
</table>

(Rates given in instructions per second)

Which is faster, A or B?
Consider running an average instruction from Prog. 1 followed by one from Prog. 2:

for A: \((1/4 + 1/4) = 1/2\)
for B: \((1/2 + 1/7) = 9/14\)

A runs the two instructions faster \((1/2 < 9/14)\), thus A is better.

Now look at the harmonic mean (Hmean) vs. the arithmetic mean (Amean).

Hmean says A has a higher rate than B (4 vs. 3.11) so Amean says B has a higher rate than A (4.5 vs. 4) so B is better, but that’s wrong!

If you used the wrong method to combine the numbers, you would buy the slower machine!

Note also that the definition of harmonic mean is just the average of the rates converted to times, then converted back to rates.

**Rules**

Use arithmetic mean to combine run times.

\[
\bar{x} = \sum_{i=1}^{N} \text{weight}_i \times \text{time}_i = \frac{1}{N} \sum_{i=1}^{N} \text{time}_i
\]

Use harmonic mean to combine rates (e.g., IPC), because it actually combines them as times then converts back to a rate

\[
\bar{H} = \left( \sum_{i=1}^{N} \text{weight}_i / \text{rate}_i \right)^{-1} = \frac{N}{\sum_{i=1}^{N} 1 / \text{rate}_i}
\]