Performance Analysis

Topics

- Measuring performance of systems
- Reasoning about performance
- Amdahl’s law
Evaluation Tools

Benchmarks, traces, & mixes
- macrobenchmarks & suites
  - application execution time
- microbenchmarks
  - measure one aspect of performance
- traces
  - replay recorded accesses
    - cache, branch, register

Simulation at many levels
- ISA, cycle accurate, RTL, gate, circuit
  - trade fidelity for simulation rate

Area and delay estimation

Analysis
- instructions, throughput, Amdahl’s law
- e.g., queuing theory
Metrics of Evaluation

Level of design $\Rightarrow$ performance metric

Examples

- Applications perspective
  - Time to run task (Response Time)
  - Tasks run per second (Throughput)

- Systems perspective
  - Millions of instructions per second (MIPS)
  - Millions of FP operations per second (MFLOPS)

- Bus/network bandwidth: megabytes per second
- Function Units: cycles per instruction (CPI)
- Fundamental elements (transistors, wires, pins): clock rate
Basis of Evaluation

Pros
- representative
- portable
- widely used
- improvements useful in reality
- easy to run, early in design cycle
- identify peak capability and potential bottlenecks

Cons
- very specific
- non-portable
- difficult to run, or measure
- hard to identify cause
- less representative
- easy to “fool”
- “peak” may be a long way from application performance

Slide courtesy of D. Patterson
Some Warnings about Benchmarks

Benchmarks measure the whole system
- application
- compiler
- operating system
- architecture
- implementation

Popular benchmarks typically reflect yesterday’s programs
- what about the programs people are running today?
- need to design for tomorrow’s problems

Benchmark timings are sensitive
- alignment in cache
- location of data on disk
- values of data

Danger of inbreeding or positive feedback
- if you make an operation fast (slow) it will be used more (less) often
  - therefore you make it faster (slower)
    - and so on, and so on…
- the optimized NOP
Know what you are measuring!

Compare apples to apples

Example

- Wall clock execution time:
  - User CPU time
  - System CPU time
  - Idle time (multitasking, I/O)

```bash
csh> time latex lecture2.tex
```
```
csh> 0.68u 0.05s 0:01.60 45.6%
csh>```

user → system → elapsed → % CPU time
Two notions of “performance”

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput (pmph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 747</td>
<td>6.5 hours</td>
<td>610 mph</td>
<td>470</td>
<td>286,700</td>
</tr>
<tr>
<td>Concorde</td>
<td>3 hours</td>
<td>1350 mph</td>
<td>132</td>
<td>178,200</td>
</tr>
</tbody>
</table>

Which has higher performance?

- **Time to do the task** (Execution Time)
  - execution time, response time, latency
- **Tasks per day, hour, week, sec, ns. ..** (Performance)
  - throughput, bandwidth

Response time and throughput often are in opposition

Slide courtesy of D. Patterson
Brief History of Benchmarking

Early days (1960s)
- Single instruction execution time
- Average instruction time [Gibson 1970]
- Pure MIPS (1/AIT)

Simple programs (early 70s)
- Synthetic benchmarks (Whetstone, etc.)
- Kernels (Livermore Loops)

Relative Performance (late 70s)
- VAX 11/780 \(\equiv\) 1-MIPS
  - but was it?
- MFLOPs

“Real” Applications (late 80s–now)
- SPEC
  - Desktop
  - Scientific
  - Java
  - Media
  - Parallel
  - etc.
- TPC
  - Transaction Processing
- Graphics
  - 3D-Mark
  - Real games (Assassin’s Creed, Call of Duty, Flight Simulator, etc.)
SPEC: Standard Performance Evaluation Corporation (www.spec.org)

System Performance and Evaluation Cooperative
- HP, DEC, Mips, Sun
- Portable O/S and high level languages

Spec89 ⇒ Spec92 ⇒ Spec95 ⇒ Spec2000 ⇒ SPEC2006....

Categories
- CPU (most popular)
- JVM, JBB
- SpecWeb - web server performance
- SFS - file server performance

Benchmarks change with the times and technology
- Elimination of Matrix 300
- Compiler restrictions
How to Compromise a Benchmark
The compiler reorganized the code!

**Change the memory system performance**
- Matrix multiply cache blocking
- You will see this later in “performance programming”

**Before**

![Before Matrix Multiplication](image1)

**After**

![After Matrix Multiplication](image2)
Spec2006 Suite

12 Integer benchmarks (C/C++)
- compression
- C compiler
- Perl interpreter
- Database
- Chess
- Bioinformatics

17 FP applications (Fortran/C)
- Shallow water model
- 3D graphics
- Quantum chromodynamics
- Computer vision
- Speech recognition

Characteristics
- Computationally intensive
- Little I/O
- Relatively small code size
- Variable data set sizes
Improving Performance: Fundamentals

Suppose we have a machine with two instructions

- Instruction A executes in 100 cycles
- Instruction B executes in 2 cycles

We want better performance…. 

- Which instruction do we improve?
CPU Performance Equation

3 components to execution time:

\[
\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \ast \frac{\text{Cycles}}{\text{Instruction}} \ast \frac{\text{Seconds}}{\text{Cycle}}
\]

Factors affecting CPU execution time:

<table>
<thead>
<tr>
<th></th>
<th>Inst. Count</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>Inst. Set</td>
<td>X</td>
<td>X</td>
<td>(X)</td>
</tr>
<tr>
<td>Organization</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>MicroArch</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

- Consider all three elements when optimizing
- Workloads change!
Cycles Per Instruction (CPI)

Depends on the instruction

\[ CPI_i = \text{Execution time of instruction } i \times \text{Clock Rate} \]

Average cycles per instruction

\[ CPI = \sum_{i=1}^{n} CPI_i \times F_i \quad \text{where} \quad F_i = \frac{IC_i}{IC_{tot}} \]

Example:

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>CPI(i)</th>
<th>%time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>0.5</td>
<td>33%</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>2</td>
<td>0.4</td>
<td>27%</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>2</td>
<td>0.2</td>
<td>13%</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>0.4</td>
<td>27%</td>
</tr>
<tr>
<td>CPI(total)</td>
<td></td>
<td></td>
<td>1.5</td>
<td></td>
</tr>
</tbody>
</table>
Amdahl’s Law

How much performance could you get if you could speed up some part of your program?

Performance improvements depend on:
- how good is enhancement
- how often is it used

Speedup due to enhancement E (fraction $p$ sped up by factor $S$):

$$\text{Speedup}(E) = \frac{\text{ExTime}_\text{old}}{\text{ExTime}_\text{new}} = \frac{1}{(1 - p) + \frac{p}{S}}$$

$$\text{ExTime}_\text{new} = \text{ExTime}_\text{old} \times \left[(1 - p) + \frac{p}{S}\right]$$
Amdahl’s Law: Example

FP instructions improved by 2x
But….only 10% of instructions are FP

\[
ExTime_{new} = ExTime_{old} \left( 0.9 + \frac{0.1}{2} \right) = 0.95 \times ExTime_{old}
\]

\[
Speedup_{total} = \frac{1}{0.95} = 1.053
\]

Speedup bounded by

\[
\frac{1}{\text{fraction of time not enhanced}}
\]
Amdahl’s Law: Example 2

• Parallelize (vectorize) some portion of your program
  • Make it 100x faster?
  • How much faster does the whole program get?

\[ T_1 = T_0 \left[ (1 - p) + \frac{p}{S} \right] \]
Amdahl’s Law: Summary message

Make the Common Case fast

Examples:

- All instructions require instruction fetch, only fraction require data
  ⇒ optimize instruction access first

- Data locality (spatial, temporal), small memories faster
  ⇒ storage hierarchy: most frequent accesses to small, local memory
Is Speed the Last Word in Performance?

Depends on the application!

Cost
- Not just processor, but other components (ie. memory)

Power consumption
- Trade power for performance in many applications

Capacity
- Many database applications are I/O bound and disk bandwidth is the precious commodity

Throughput (a form of speed)
- An individual program isn’t faster, but many more programs can be completed per unit time
- Example: Google search (processes many, many searches simultaneously)
Summary

Today
- Performance analysis overview
- Amdahl’s law

Next Time
- Making the processor faster: pipelining