Evaluating Performance Profilers

CIT 595
Spring 2010

Evaluating Computer Performance

Step I – Choose a workload
- Workload: set of programs whose performance you care about
- Use standardized workload (known as benchmarks)

Step II – Decide the metric
- Execution Time or Throughput

Step III - Evaluate data gathered using statistical analysis
- Statistical method use depends on
  - Metric
  - Distribution of the data

Standard Performance Evaluation Corporation (SPEC)
- Is consortium that collects, distributes and standardizes benchmarks
- Produces benchmark suites for various classes of CPU, Java, I/O, Web, Multithreaded etc.
- E.g. CPU 2006 consists of 29 CPU intensive C/C++, Fortran programs
  - integer: perl, gcc,bzip2(compression),sjeng(AI: chess)
  - floating point: povray(ray tracing), wrf (weather prediction),sphynx3 (speech recognition)
- Like SPEC, there is Transaction Processing Council (TPC)
  - Used for web/database server workloads
  - Programs are I/O or network intensive rather than CPU intensive

Statistical Analysis: Arithmetic Mean
- Used for units that proportional to time
- The arithmetic mean can be misleading if the data are skewed or scattered

<table>
<thead>
<tr>
<th>Program</th>
<th>System A Execution Time</th>
<th>System B Execution Time</th>
<th>System C Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>50</td>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>w</td>
<td>200</td>
<td>400</td>
<td>600</td>
</tr>
<tr>
<td>x</td>
<td>250</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>y</td>
<td>400</td>
<td>800</td>
<td>800</td>
</tr>
<tr>
<td>z</td>
<td>5000</td>
<td>4100</td>
<td>3500</td>
</tr>
<tr>
<td>Average</td>
<td>1180</td>
<td>1180</td>
<td>1180</td>
</tr>
</tbody>
</table>
Geometric Mean

- Represented as:
  \[ G = \left( x_1 \cdot x_2 \cdot x_3 \cdot \ldots \cdot x_n \right)^{1/n} \]

- Unlike an arithmetic mean, tends to dampen the effect of skew

- Used for unit less quantities e.g. performance ratio or speedup

- Performance results are stated in relation to the performance of a common machine used as reference

Geometric Mean Example

<table>
<thead>
<tr>
<th>Program</th>
<th>System A Execution Time</th>
<th>System B Execution Time</th>
<th>System C Execution Time</th>
<th>Execution Time Normalized to B</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>50</td>
<td>100</td>
<td>800</td>
<td>0.5</td>
</tr>
<tr>
<td>w</td>
<td>200</td>
<td>400</td>
<td>500</td>
<td>0.8667</td>
</tr>
<tr>
<td>x</td>
<td>250</td>
<td>500</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>400</td>
<td>800</td>
<td>800</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>5000</td>
<td>4100</td>
<td>3500</td>
<td>1.1714</td>
</tr>
</tbody>
</table>

Geometric Mean = \( \left( \frac{100}{50} \times \frac{400}{200} \times \frac{500}{250} \times \frac{800}{400} \times \frac{4100}{5000} \right)^{1/5} = 1.6733 \)

Harmonic Mean

- Used to units that inversely proportional to time i.e. throughput

- Unlike latencies, throughput cannot be added

- E.g. 1st - 10 mile @ 30 mph, 2nd - 10 miles @ 40 mph and last 10 miles @ 60 mph
  > Average is not 43.3 mph

- Harmonic Mean
  \[ H = n ÷ (1/x_1+1/x_2+1/x_3+\ldots+1/x_n) \]
Reporting Performance Results

- Idea is that performance results must be reproducible

- E.g. SPEC reports
  - Complete description of machine (HW + SW)
    - Clock speed, Memory Organization, Operating System etc
  - Compiler flags
  - Choice of Inputs
  - Baseline and optimized results

Programmer Tools: Timing

- On Unix, command called “time” provides the following:
  - First do: gcc myprog.c –o progname
  - Then do: time progname
  - 90.7u 12.9s 2:39 65%
    - User CPU time is 90.7 u
    - system CPU time is 12.9 s
    - elapsed time is 159 s (2mins 39 sec)
    - % of elapsed time that is CPU time

- But, which parts of the code are the most time consuming?

Programmer Tools: Timing

- Many O.Ss provide a way to get the time
  - e.g., UNIX `gettimeofday` command (time since Jan 1, 1970)
  ```c
  #include <sys/time.h>
  struct timeval start_time, end_time;
  gettimeofday(&start_time, NULL);
  <execute some code here>
  gettimeofday(&end_time, NULL);
  float seconds = end_time.tv_sec - start_time.tv_sec + 1.0E-6f * (end_time.tv_usec - start_time.tv_usec);
  ```

- `tv_sec` field: This represents the number of whole seconds of elapsed time
- `tv_usec` is the rest of the elapsed time (a fraction of a second), represented as the number of microseconds. It is always less than one million.

Programmer Tool: Profiler

- Profiler
  - Performance analysis tool that measures the behavior of a program as it runs
    - Particularly the frequency and duration of function calls
  - The output is a statistical summary of the events observed (a profile)
  - E.g. GProf is GNU Profiler
Gprof (GNU Performance Profiler)

- Instrumenting the code
  - `gcc -Wall -pg prog.c -o prog`

- Running the code (e.g., `prog`)
  - Produces output file `gmon.out` containing statistics

- Printing a human-readable report from `gmon.out`
  - `gprof ./prog > gprofreport`

Two Main Outputs of Gprof

- Call graph profile: detailed information per function
  - Which functions called it, and how much time was consumed?
  - Which functions it calls, how many times, and for how long?

- Flat profile: one line per function
  - name: name of the function
  - %time: percentage of time spent executing this function
  - cumulative seconds: time spent in this function + others
  - self seconds: time spent executing this function
  - calls: number of times function was called (excluding recursive)
  - self ms/call: average time per execution (excluding descendents)
  - total ms/call: average time per execution (including descendents)

Example

Flat profile:

<table>
<thead>
<tr>
<th>Each sample counts as 0.01 seconds.</th>
<th>name</th>
<th>cumulative seconds</th>
<th>self seconds</th>
<th>total seconds</th>
<th>calls</th>
<th>ms/call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time: 58.83 0.58 0.58 main</td>
<td></td>
<td>271.11 271.11 foo0</td>
<td>7.10 0.92 0.07 1 70.29 70.29 foo1</td>
<td>7.10 0.99 0.07 1 70.29 70.29 foo2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Limitation of Gprof

- Timing is not very precise
- Based on interval counting
  - A counter for each function is maintained to record the time spent executing that function
  - OS causes program to be interrupted at some regular time interval $\delta$ (range 1-10ms)
  - On interrupt: sees what line is running, & what function it's in; the counter for that function is increased by $\delta$
- Run the program multiple times so that profiler can gather information
### Amdahl's Law

- **Ex Time After Improvement** = Ex. Time Unaffected + Ex Time Affected/Amount of Improvement
- **Speedup** = Ex time (old)/Ex Time (new)
  - Speedup = \( \frac{1}{(1-f) + \frac{f}{k}} \)
    - \( f \) – fraction of work performed by new component (enhancement)
    - \( k \) – speedup of the new component
  - Speedup limited by non-accelerated part

E.g. Suppose a program runs 100 sec on a machine with MUL operation responsible 80 sec of this time. How much do we have improve the speed of multiplication if we want the program to run 4 times faster?

### Other Profiler Examples

- **Linux Trace Toolkit**
  - Collects data on processes blocking, context switches, and execution time.
  - This helps identify performance problems over multiple processes or threads

- **VTune**
  - Performance Analyzer is Intel's tool for call graph or analyzing a set of tuning events
  - It works with C/C++/Fortran/.NET/Java and other applications on Linux or Windows, but only when running on selected Intel hardware