Differential Profiling

Andrew Lenharth
CS 378 Fall 2013

What if we want to address scaling?
- Input size or complexity
- Number of threads

How do we go from a bunch of numbers to a higher level understanding of performance?

Differential profiling

- General technique to find scaling bottlenecks
- Picks out significant changes even if small
- Applies to any profiling measurement
  - Printf style time or PAPI counters
  - Vtune style mass of numbers

Related to Qualitative Differences

- Take 2 profiles and compute a function over them, the heart of which is taking their difference
  - $F(P_1, P_2) = f(P_1) - g(P_2)$
- 3 Types
  - Ratio
  - Weighted Difference
  - projected-saturation

Ratio Differential Profiling

- We need at least 2 inputs: baseline and stressed
  - Size, threads, complexity, outside load, available resources
- Measure the system under each producing Mb and Ms
  - One or many measured quantities
- Take the ratio Ms/Mb
- Sort by the ratio

Ratio DP – Familiar

- New processor
  - $\text{Time}_{\text{old}} / \text{Time}_{\text{new}} = \text{speedup}$!
    - e.g. Apple A6 is twice as fast as A5
    - e.g. Ivy bridge is 20% faster than Sandy bridge
- Threads
  - My program is 3 times faster at 4 threads

But we can apply it to individual instructions
Ratio DP – Example

<table>
<thead>
<tr>
<th>Function</th>
<th>M1</th>
<th>M2</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malloc</td>
<td>10</td>
<td>11</td>
<td>1.1</td>
</tr>
<tr>
<td>Validate</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Format</td>
<td>100</td>
<td>101</td>
<td>1.01</td>
</tr>
<tr>
<td>Foo</td>
<td>34</td>
<td>65</td>
<td>1.9</td>
</tr>
<tr>
<td>Bar</td>
<td>345</td>
<td>543</td>
<td>1.57</td>
</tr>
</tbody>
</table>

- Validate grew the faster from M1 to M2, worth investigating even though seemingly insignificant
- Foo seems not so good either, though small

What are these lines?
- Padded Lock? – No context
- WorkListHelper.h:63 – part of pop: if (empty())
- UserContext.h:86 – A per worklist push check

Weighted Differential Profiling

- Consider 2 conditions: baseline and stressed
- Measure the system under each producing Mb and Ms
- Quantify work done as Wb and Ws
- Compute $D = \frac{W_b \times Ms - W_s \times Mb}{Mb}$
- Sort by D

Weighted DP Properties

- Filters out poor scaling but low cost/impact
- Must be able to quantify work done
  - If you can't, do you really know what the system is suppose to be doing?

Weighted DP – Example

<table>
<thead>
<tr>
<th>Function</th>
<th>M1</th>
<th>M2</th>
<th>Weighted diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malloc</td>
<td>10</td>
<td>11</td>
<td>-4</td>
</tr>
<tr>
<td>Validate</td>
<td>1</td>
<td>4</td>
<td>2.5</td>
</tr>
<tr>
<td>Format</td>
<td>100</td>
<td>101</td>
<td>-49</td>
</tr>
<tr>
<td>Foo</td>
<td>34</td>
<td>65</td>
<td>14</td>
</tr>
<tr>
<td>Bar</td>
<td>345</td>
<td>543</td>
<td>25.5</td>
</tr>
</tbody>
</table>

- Now assume load at M2 = 1.5x load M1
- Validate IS actually insignificant
- Foo seems not so good either, though small
- Bar had significant growth
Weighted DP – Parallel Special Case

• For a given load, you can vary the number of threads.
• The total cycles summed over all threads should remain constant
• Thus \( W_s = W_b \)
• Any metric should then be flat if achieving ideal scaling

Projected Saturation

• Ratio and Weighted Difference assume unbounded resources
  – Fine for cpu time
• Limited resources impose hard constraints
  – Bandwidth, memory, critical section hold time
• Project from the data how far until measured resource hits bound

Linear Projection

\[
S = (m_s - m_2)(l_2 - l_1)/(m_2 - m_1) + l_2
\]

where \( m_s \) is the saturation point
\( m_i \) is the measurement at the \( i \)th workload
\( l_i \) is the offered load at the \( i \)th workload
\( S \) is the workload in terms of \( I \) at which saturation occurs

Other projections

• For non-linear growing resources, use a different projection function
• May require more than 2 measurements

Saturation -- Example

• Database with a uniform load per client
  – 4 connections = IO 128kB / second
  – 12 connections = IO 164kB / second
  – Disk IO limit 2MB / second
• \( S = (m_s - m_2)(l_2 - l_1)/(m_2 - m_1) + l_2 \)
• \( S = (2048 - 164)(12 - 4)/(164 - 128) + 12 \)
• \( S = 430 \)
• 430 clients saturate the disk
Running Example: SSSP

```cpp
std::set<UpdateRequest, std::less<UpdateRequest> > initial;

//**SNIP** Cut initialization stuff

while (!initial.empty()) {
    UpdateRequest req = *initial.begin();
    initial.erase(initial.begin());
    SNode& data = graph.getData(req.n);
    if (req.w < data.dist) {
        data.dist = req.w;
        for (Graph::edge_iterator ii = graph.edge_begin(req.n),
            ee = graph.edge_end(req.n); ii != ee; ++ii) {
            GNode dst = graph.getEdgeDst(ii);
            int d = graph.getEdgeData(ii);
            unsigned int newDist = req.w + d;
            if (newDist < graph.getData(dst).dist)
                initial.insert(UpdateRequest(dst, newDist));
        }
    }
}
```

PAPI

- Standard way to collect performance counters
  - Mainly Linux
  - Cross platform
- Provides portable names for commonly measured quantities
  - e.g. PAPI_L2_TCM (L2 data + instruction misses)
- Can multiplex counters
- Handles overflow
- Handles virtualization

What are those Lines?

- 198: Reading a node for the first time
- 209: pushing a node onto the local worklist
- 202: iterating over the edges
- stl_map.h: 455, 457: map index
- 199: CAS
- WL: 420: push node to far sub-worklist
- WL: 426: push nodes

PAPI Example

- Initialization code is in the homework handout
- Running on road network of the US
VTune

- Intel tool for event based sampling
  - Similar tools include shark and oprofile
- Multiplexes counters
- Performs some analysis and optimization guidance
- Newest version is context sensitive
- Includes other analysis tools
  - e.g. lock analysis

Vtune Example

- Same SSSP, using CSR Graph
- Vtune Summary page

<table>
<thead>
<tr>
<th>Elapsed Time:</th>
<th>17.504s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Event Count:</td>
<td>48,032,000,000</td>
</tr>
<tr>
<td>CPU_CLK_UNHALTED.THREAD:</td>
<td>31318000000</td>
</tr>
<tr>
<td>INST_RETIRED.ANY:</td>
<td>16714000000</td>
</tr>
<tr>
<td>CPI Rate:</td>
<td>1.874</td>
</tr>
<tr>
<td>Retire Stalls:</td>
<td>0.77s</td>
</tr>
<tr>
<td>LLC Miss:</td>
<td>0.004</td>
</tr>
<tr>
<td>LLC Load Misses Serviced By Remote DRAM:</td>
<td>0s</td>
</tr>
<tr>
<td>Contended Accesses:</td>
<td>0s</td>
</tr>
<tr>
<td>Instruction Speculation</td>
<td>0.230s</td>
</tr>
<tr>
<td>Branch Mispredict:</td>
<td>0.234s</td>
</tr>
<tr>
<td>Execution Stalls:</td>
<td>0.317s</td>
</tr>
<tr>
<td>Data Sharing:</td>
<td>0s</td>
</tr>
<tr>
<td>Paused Time:</td>
<td>1.740s</td>
</tr>
</tbody>
</table>

Vtune Live Demo

SSSP has runtime problems

DMR – kernel bottleneck

Weighted DP - example