Lecture Topics

- finish RigelSim example
- capturing more details (OProfile)
- subtle hazards (ProfileMe)

Administrivia

- midterm
  - one week from today, in class
  - old midterms available online

- constructor example code online; please take a look
[glossed over quickly last time to show data]

- call graph profile information
  - note: accounting for recursion is tricky; see notes in output or papers
  - tracks number of samples found in function and descendants
  - listed in decreasing order of frequency
  - function names annotated with rank information
  - for each parent function
    - time in function while within this parent
    - time in function’s children while within this parent
    - number of calls made from parent
    - total number of calls made non-recursively
  - for reported function:
    - % of time, self-time, children time, calls (recursion separately)
  - for each child function
    - time in child while within reported function
    - time in grandchildren while within reported function
    - number of calls made to child from reported function
    - total number of calls to child made non-recursively
[review]

- example from RigelSim
  - simulator for 1000-core chip
  - example uses 128 cores
  - names truncated for clarity
  - optimized -O2

- total time is 664.50 seconds

```plaintext
%      cum.     self
time   seconds   seconds calls        name
64.90    431.27   431.27 1133522621 std::map<std::string, ...
9.39    493.68    62.41     28629808 CacheModel::read_access_instr
7.48    543.40    49.72     29209531 CacheModel::read_access
7.22    591.37    47.97      8387008   Cluster::step
1.18    599.20     7.83     84413863   CoreSystem::execute

%      self          calls              name
0.00         176/1133522621   ProfileStat::init(_IO_FILE*) [124]
0.01       31224/1133522621   DRAMModel::SetDataBusBusy [108]
0.01       33026/1133522621   DRAMModel::SendCommand [96]
0.02       52069/1133522621   TileInterconnectHTree::PerCycle [18]
0.03       72078/1133522621   GlobalNetworkCrossbar::PerCycle [42]
1.51     3975785/1133522621   L2Cache::PerCycle [13]
1.99     5241880/1133522621   TileInterconnectBase::PerCycle [19]
427.69 1124116383/1133522621   Cluster::step() [2]

[3] 64.9 431.27             1133522621   std::map<std::string, ...
  0.00         178/354             std::_Rb_tree [301]
  0.00         178/191             std::_Rb_tree [303]
  0.00          89/89              std::_Rb_tree [314]
```

- flat profile indicates that STL map call is taking a large fraction of total time
- call graph profile indicates that single parent primarily responsible (cluster step function)
• after code fixed to use enum and array

• total time is 151.15 seconds (saved more than we might expect)
  – top call ordering is identical (except for map), as are counts
  – top call timing reduced by 17 to 58% of original (extra optimizations?)
  – savings on longer runs is about a factor of two

<table>
<thead>
<tr>
<th>%</th>
<th>self</th>
<th>cum.</th>
<th>time</th>
<th>seconds</th>
<th>seconds</th>
<th>calls</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>34.20</td>
<td>51.70</td>
<td>51.70</td>
<td>51.70</td>
<td>28629808</td>
<td></td>
<td></td>
<td>CacheModel::read_access_instr</td>
</tr>
<tr>
<td>24.68</td>
<td>89.00</td>
<td>37.30</td>
<td>37.30</td>
<td>29209531</td>
<td></td>
<td></td>
<td>CacheModel::read_access</td>
</tr>
<tr>
<td>13.42</td>
<td>109.28</td>
<td>20.28</td>
<td>20.28</td>
<td>8387008</td>
<td></td>
<td></td>
<td>Cluster::step</td>
</tr>
<tr>
<td>2.22</td>
<td>112.64</td>
<td>3.36</td>
<td>3.36</td>
<td>84413863</td>
<td></td>
<td></td>
<td>CoreSystem::execute</td>
</tr>
</tbody>
</table>

Capturing More Details

• only instrumented calls recorded with gprof; what about
  – library routines?
  – system calls?
  – interrupt handlers?

• OProfile tool supports system-wide data collection
  – oprofile.sourceforge.net
  – operates as a kernel daemon
  – driven by processor performance counter events
    • these are not virtualized automatically!
    • OProfile counts everything that executes
  – opens gaping security holes
    • e.g., freeze machine on demand
    • be careful about who can use it
• performance counters
  – wide range available (examples from AMD Athlon)
    • data cache accesses, misses, fills from L2/memory, writebacks
    • instruction cache fetches, misses
    • data/instruction TLB misses (two levels each)
    • misaligned data references
    • retired: instructions, operations, taken/mispredicted/all branches
    • number of interrupts, cycles spent with IF=0, interrupts pending with IF=0
  – use model
    • select event counter and choose counter value
    • use privileged instructions to inform processor
    • when reached, NMI generated (processor-specific, of course!)

• OProfile approach
  – install kernel module
  – use kernel daemon to track samples
  – minimize interference overhead
    • do not dump to disk (for example) on each NMI
    • accumulate into buffer, dump once in a while
    • high water mark used to make daemon runnable
  – use model
    • start the daemon
    • start profiling (don’t profile starting the daemon!)
    • execute your program
    • stop profiling
    • (when all done, tear down daemon)
    • allows sessions to be named (not all “gmon.out”)

• OProfile options
  – location of kernel image (or none)
  – buffer size, high water mark
  – call graph depth desired
  – separation of sample data
    • based on current task information and CPU ID
      – interrupts can thus be misattributed
      – e.g., my disk block can arrive while some other task runs
      – likely for interactive code, but probably not too important
    • options
      – lump everything together (default)
      – separate results per application
        (shared libraries or both libraries and kernel)
      – separate results per thread
      – separate results per CPU
    • data sets expand rapidly; can limit which images to track to reduce

Subtle Hazards

• two main problems
  – sampling rate may alias with program’s natural event frequency
  – events and instructions are not necessarily correlated
    (interrupts do not break most instructions)
    • floating point division
    • long-latency events (e.g., TLB misses)
    • still true today for some event types

• graph handed out
  – Intel CPU clock sampling (100,000 cycles)
  – note periodic behavior; superscalar issue/retirement?
placeholder for
OProfile linked list count graph

(handout distributed in lecture;
available to students as
event-output.pdf)