Reminders

• Project report due Dec 7th
  – Bring hard copy to class
  – Email soft copy to Patrick & Shakir

• Course evaluation due Dec 10th

• Final Exam due Dec 14th
  – Email to Prof. Crowley by 4PM on Dec 14th
Today’s Discussion

• Performance Monitoring*
  – Types of Monitors
  – Program Monitors
    • Instrumentation
    • Valgrind/Cachegrind (an example)

Monitoring

“If you can not measure it, you can not improve it.” - Lord Kelvin

• First, key step in performance measurement
  – observe the execution of system

• Used for
  – improving software
  – finding bottleneck
  – modeling
Types of Monitors

• Based on implementation
  – Hardware
    • probes, counters, timers (Oprofile & Vtune)
  – Software
    • Program monitors/profilers (Valgrind)
  – Firmware

• Key tradeoff: resolution vs. overhead
Program Monitors - 1

• Observe the execution of a program

• Used for
  – Tracing
  – Timing
  – Tuning
  – Asserting checking
  – Coverage analysis
Program Monitors - 2

• Basic steps
Program Instrumentation

• Modify source
  – Static (\textit{Gprof})
  – Dynamic (\textit{Dtrace})

• Rewrite binary
  – Static (\textit{Etch})
  – Dynamic (\textit{Valgrind})
Valgrind

• Dynamic binary instrumentation (DBI)
  – Instruments precompiled binary
  – Framework for building other tools

• Mainly used for debugging
  – memory errors (Memcheck tool)
Valgrind Limitations

- Kernel behavior not counted
- Other processes not counted
- Thread scheduling variations
Valgrind Tools

- Memory error detectors (*Memcheck*)
- 2 thread error detectors (*Helgrind & DRD*)
- Call-graph profiler (*Callgrind*)
- Heap profiler (*Massif*)
- Cache and branch-prediction profiler
  - *Cachegrind*
Cacheegrind

• Simulates program behavior
  – cache hierarchy
  – branch prediction (optional)
Cachegrind: Simulated Cache

• Separate L1 $: instruction (I1) and data (D1)
• Unified, inclusive L2:
• Write allocate
• Cache sizes can be manually set
  – default are automatically determined
CacheGrind: Cache Config.

- `--I1=<size>,<associativity>,<line size>`
- `--D1=<size>,<associativity>,<line size>`
- `--L2=<size>,<associativity>,<line size>`
CacheGrind: Branch Predictor

• Correlating 2-bit counter
  – Array of 16384 2-bit saturating counters
  – Array index computed from the branch address and the taken/not-taken behavior of the last few branches

• Target prediction- same as last time
Cacheegrind Limitations

• No TLB misses
• No virtual to physical address mappings
Cachegrind Usage

• Compile program
  – Preferably using --g to include debugging info

• Run valgrind
  – valgrind --tool=cachegrind --branch-sim=yes
  – Outputs to file cachegrind.out.<pid>

• Annotated source files
  – cg_annotate --<pid> <sourcefile(s)>
Cachegrind Example

- **fast.c**
  ```c
  int main(void)
  {
    int h, i, j, a[1024][1024];
    for (h=0; h<10; h++)
      for (i=0; i<1024; i++)
        for (j=0; j<1024; j++)
          a[i][j]=0
    return 0;
  }
  ```

- **slow.c**
  ```c
  int main(void)
  {
    int h, i, j, a[1024][1024];
    for (h=0; h<10; h++)
      for (i=0; i<1024; i++)
        for (j=0; j<1024; j++)
          a[j][i]=0
    return 0;
  }
  ```
Valgrind --
tool=cachegrind fast

Output for fast.c:

<table>
<thead>
<tr>
<th>Type</th>
<th>Refs</th>
<th>Misses</th>
<th>L1 Misses</th>
<th>L2 Misses</th>
<th>L1 Miss Rate</th>
<th>L2 Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>94,566,217</td>
<td>566</td>
<td>560</td>
<td>0.0%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>52,518,512</td>
<td>856,324</td>
<td>801</td>
<td>566</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td>0.0%</td>
<td>6.2%</td>
<td></td>
</tr>
<tr>
<td>L2d</td>
<td></td>
<td></td>
<td></td>
<td>0.0%</td>
<td>6.2%</td>
<td></td>
</tr>
</tbody>
</table>

Output for slow.c:

<table>
<thead>
<tr>
<th>Type</th>
<th>Refs</th>
<th>Misses</th>
<th>L1 Misses</th>
<th>L2 Misses</th>
<th>L1 Miss Rate</th>
<th>L2 Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>94,566,217</td>
<td>566</td>
<td>560</td>
<td>0.0%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>52,518,512</td>
<td>10,486,724</td>
<td>801</td>
<td>10,485,923</td>
<td>99.7%</td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td>0.0%</td>
<td>99.7%</td>
<td></td>
</tr>
<tr>
<td>L2d</td>
<td></td>
<td></td>
<td></td>
<td>0.0%</td>
<td>99.7%</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Refs</th>
<th>Misses</th>
<th>L1 Misses</th>
<th>L2 Misses</th>
<th>L1 Miss Rate</th>
<th>L2 Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>10,487,280</td>
<td>1,366</td>
<td>1,312</td>
<td>0.0%</td>
<td>99.7%</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>10,487,280</td>
<td></td>
<td></td>
<td>0.0%</td>
<td>99.7%</td>
<td></td>
</tr>
</tbody>
</table>
Cacheegrind Example (cont’d)

- `cg_annotate --<pid> fast.c`
Annotated `fast.c`:

<table>
<thead>
<tr>
<th>Ir</th>
<th>I1mr</th>
<th>I2mr</th>
<th>Dr</th>
<th>D1mr</th>
<th>D2mr</th>
<th>Dw</th>
<th>D1mw</th>
<th>D2mw</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>0</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>51,240</td>
<td>0</td>
<td>0</td>
<td>20,490</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>52,469,760</td>
<td>0</td>
<td>0</td>
<td>20,981,760</td>
<td>0</td>
<td>0</td>
<td>10,240</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>41,943,040</td>
<td>0</td>
<td>0</td>
<td>20,971,520</td>
<td>0</td>
<td>0</td>
<td>10,485,760</td>
<td>655,360</td>
<td>655,360</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

int main(void) {
  int h, i, j, a[i...]
  for (h = 0; h < ...
  for (i = 0; i...
  for (j = 0...
  return 0;
}

Annotated `slow.c`:

<table>
<thead>
<tr>
<th>Ir</th>
<th>I1mr</th>
<th>I2mr</th>
<th>Dr</th>
<th>D1mr</th>
<th>D2mr</th>
<th>Dw</th>
<th>D1mw</th>
<th>D2mw</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>0</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>51,240</td>
<td>0</td>
<td>0</td>
<td>20,490</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>52,469,760</td>
<td>0</td>
<td>0</td>
<td>20,981,760</td>
<td>0</td>
<td>0</td>
<td>10,240</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>41,943,040</td>
<td>0</td>
<td>0</td>
<td>20,971,520</td>
<td>0</td>
<td>0</td>
<td>10,485,760</td>
<td>10,485,760</td>
<td>10,485,760</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

int main(void) {
  int h, i, ...
  for (h = 0...
  for (i ... for (i...
  return 0;
}
Assignment

• For Monday
  – Turn in final report hard copy
  – Pick up final exam