A Performance Tuning Methodology: From the System Down to the Hardware - Introduction

Jackson Marusarz
Intel Corporation
ATPESC 2014
Why performance profiling?

Project performance tuning for:

- Reducing direct compute time costs
- Decreasing indirect costs
- Better user/customer experience

If you are not in that business, don’t bother
Project development cycle and performance analysis

1. **Design**
   - Think performance wise (app/sys level)

2. **Prototyping**
   - Choose perf. effective solutions

3. **Implementation**
   - Apply perf. optimization and check results

4. **Testing**
   - Add perf. regression phase to test stage

5. **Release**
   - Collect and analyze perf. related complaints from users of your product
Optimization: A Top-down Approach

**System**
- H/W tuning:
  - BIOS (TB, HT)
  - Memory
  - Network I/O
  - Disk I/O
- OS tuning:
  - Page size
  - Swap file
  - RAM Disk
  - Power settings
  - Network protocols

**Application**
- Better application design:
  - Parallelization
  - Fast algorithms / data bases
  - Programming language and RT libs
  - Performance libraries
  - Driver tuning

**Processor**
- Tuning for Microarchitecture:
  - Compiler settings/Vectorization
  - Memory/Cache usage
  - CPU pitfalls

## Performance profiling tools
### Level wise selection

<table>
<thead>
<tr>
<th>System</th>
<th>System profiler</th>
<th>OS embedded</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Universal (for OS, HW)</td>
<td>Windows: Perf mon, Proc mon</td>
</tr>
<tr>
<td></td>
<td>Proprietary (OS+HW)</td>
<td>Linux: top, vmstat, OProfile</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>Supported languages</th>
<th>Windows: WPT, Xperf, VTune</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE based</td>
<td>.Net/C#, Java</td>
<td>Managed: .Net, Java tools, VTune</td>
</tr>
<tr>
<td>Command Line</td>
<td>Python, Java Script, HTML</td>
<td>Linux: gprof, Valgrind, Google perftools, Crxprof, VTune</td>
</tr>
<tr>
<td></td>
<td>C, C++, Fortran</td>
<td></td>
</tr>
</tbody>
</table>

### Microarchitcture
- Provided by CPU/Platform manufacturer
Performance profiling tools
Level wise selection

<table>
<thead>
<tr>
<th>System</th>
<th>System profiler</th>
<th>OS embedded</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Universal (for OS, HW)</td>
<td>Windows: Perf mon, Proc mon</td>
</tr>
<tr>
<td></td>
<td>Proprietary (OS+HW)</td>
<td>Linux: top, vmstat, OProfile</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>Supported languages</th>
<th>Windows: WPT, Xperf, VTune</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE based</td>
<td>.Net/C#, Java</td>
<td>Managed: .Net, Java tools, VTune</td>
</tr>
<tr>
<td>Command Line</td>
<td>Python, Java Script, HTML</td>
<td>Linux: gprof, Valgrind, Google perftools, Crxprof, VTune</td>
</tr>
<tr>
<td></td>
<td>C, C++, Fortran</td>
<td></td>
</tr>
</tbody>
</table>

Microarchitecture
Provided by CPU/Platform manufacturer

Tools are essential for efficient performance analysis.
Optimization: A Top-down Approach

**H/W tuning:**
- BIOS (TB, HT)
- Memory
- Network I/O
- Disk I/O

**OS tuning:**
- Page size
- Swap file
- RAM Disk
- Power settings
- Network protocols

*Other brands and names are the property of their respective owners.*
System Tuning

Who: System Administrators, Performance Engineers, Machine Owners, etc...

How:

• Benchmarks
  • Numerous FLOPS benchmarks
  • Network/MPI Benchmarks: [www.intel.com/go/imb](http://www.intel.com/go/imb)
  • <insert your favorite here>

• Tools
  • `vmstat`, `top`, `sysprof`, `iostat`, `sar`, Task Manager, etc...
  • Many vendor/platform specific tools

• Fixes
  • Upgrade Hardware - $$$
  • Check BIOS and OS configurations
    • Prefetchers, NUMA, Memory Configuration, Power Management, SMT
System Tuning

Who: System Administrators, Performance Engineers, Machine Owners, etc...

How:
• Benchmarks
  • Numerous FLOPS benchmarks
  • Network/MPI Benchmarks: [www.intel.com/go/imb](http://www.intel.com/go/imb)
  • <insert your favorite here>

• Tools
  • vmstat, top, sysprof, iostat, sar, Task Manager, etc...
  • Many vendor/platform specific tools

• Fixes
  • Upgrade Hardware - $$$
  • Check BIOS and OS configurations
    • Prefetchers, NUMA, Memory Configuration, Power Management, SMT

This is often outside the capabilities of most users
Optimization: A Top-down Approach

**H/W tuning:**
- BIOS (TB, HT)
- Memory
- Network I/O
- Disk I/O

**OS tuning:**
- Page size
- Swap file
- RAM Disk
- Power settings
- Network protocols

**Better application design:**
- Parallelization
- Fast algorithms / data bases
- Programming language and RT libs
- Performance libraries
- Driver tuning
Application Tuning

Who: Software Developers, Performance Engineers, Domain Experts

How:
• Workload selection
  • Repeatable results
  • Steady state

• Define Metrics and Collect Baseline
  • Wall-clock time, FLOPS, FPS
  • <insert your metric here>

• Identify Hotspots
  • Focus effort where it counts
  • Use Tools

• Determine inefficiencies
  • Is there parallelism?
  • Are you memory bound?
  • Will better algorithms or programming languages help?

This step often requires some knowledge of the application and its algorithms
Application Tuning
Find Hotspots

• This could be at the module, function, or source code level
• Determine your own granularity

$ oreport --exclude-dependent --demangle=smart --symbols 'which lyx'

CPU: PIII, speed 863.195 MHz (estimated)
Counted CPU_CLK_UNHALTED events (clocks processor is not halted) with a unit mask of 0x00 (No unit mask)

<table>
<thead>
<tr>
<th>uma</th>
<th>samples</th>
<th>%</th>
<th>symbol name</th>
</tr>
</thead>
<tbody>
<tr>
<td>081ec974</td>
<td>5016</td>
<td>8.5096</td>
<td>_Rb_tree&lt;unsigned short, pair&lt;unsigned short const, int&gt;&gt;, unsigned short</td>
</tr>
<tr>
<td>0810c4ec</td>
<td>3323</td>
<td>5.6375</td>
<td>Paragraph::getFontSettings(BufferParams const&amp;, int) const</td>
</tr>
<tr>
<td>081319d8</td>
<td>3220</td>
<td>5.4627</td>
<td>LyXText::setFont(Buffer const*, Paragraph*, int) const</td>
</tr>
<tr>
<td>080e45d8</td>
<td>3011</td>
<td>5.1082</td>
<td>LyXFont::realize(LyXFont const&amp;)</td>
</tr>
<tr>
<td>080e3d78</td>
<td>2623</td>
<td>4.4499</td>
<td>LyXFont::LyXFont()</td>
</tr>
<tr>
<td>081255a4</td>
<td>1823</td>
<td>3.0927</td>
<td>LyXText::singleWidth(BufferView*, Paragraph*, int, char) const</td>
</tr>
<tr>
<td>080e3cf0</td>
<td>1804</td>
<td>3.0605</td>
<td>operator==(LyXFont::FontBits const&amp;, LyXFont::FontBits const&amp;)</td>
</tr>
<tr>
<td>081120e0</td>
<td>1729</td>
<td>2.9332</td>
<td>Paragraph::Pimpl::getChar(int) const</td>
</tr>
<tr>
<td>081ed020</td>
<td>1380</td>
<td>2.3412</td>
<td>font_metrics::width(char const*, unsigned, LyXFont const&amp;)</td>
</tr>
<tr>
<td>08110d60</td>
<td>1310</td>
<td>2.2224</td>
<td>Paragraph::getChar(int) const</td>
</tr>
<tr>
<td>081ebc94</td>
<td>1227</td>
<td>2.0816</td>
<td>qfont_loader::getfontinfo(LyXFont const&amp;)</td>
</tr>
</tbody>
</table>

Application Tuning
Find Hotspots

- This could be at the module, function, or source code level
- Determine your own granularity

sysprof: http://sysprof.com
Application Tuning

Find Hotspots

- This could be at the module, function, or source code level
- Determine your own granularity

Application Tuning
Find Hotspots

- This could be at the module, function, or source code level
- Determine your own granularity

Application Tuning
Find Hotspots

- This could be at the module, function, or source code level
- Determine your own granularity

This may reinforce your understanding of the application but often reveals surprises
Application Tuning
Resource Utilization

- Is the application parallel?
- Multi-thread vs. Multi-process
- Memory Bound?

```
last pid: 86494: load averages: 0.83, 0.65, 0.69 up 67+22:48:43 14:44:15
227 processes: 1 running, 224 sleeping, 2 zombie
CPU: 20.2% user, 0.0% nice, 6.5% system, 0.2% interrupt, 73.1% idle
Mem: 1657M Active, 1866M Inact, 273M Wired, 190M Cache, 112M Buf, 11M Free
Swap: 4500M Total, 249M Used, 4251M Free, 5% Inuse

<table>
<thead>
<tr>
<th>PID</th>
<th>USERNAME</th>
<th>THR</th>
<th>PRI</th>
<th>NICE</th>
<th>SIZE</th>
<th>RES</th>
<th>STATE</th>
<th>%C</th>
<th>TIME</th>
<th>WCPU</th>
<th>COMMAND</th>
<th>%use</th>
</tr>
</thead>
<tbody>
<tr>
<td>86460</td>
<td>www</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>150M</td>
<td>30204K</td>
<td>accept 1</td>
<td>0:02</td>
<td>11.18%</td>
<td>php-cgi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86458</td>
<td>www</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>150M</td>
<td>29912K</td>
<td>accept 0</td>
<td>0:02</td>
<td>8.98%</td>
<td>php-cgi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86463</td>
<td>psql</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>949M</td>
<td>99M</td>
<td>sbwait 1</td>
<td>0:01</td>
<td>7.96%</td>
<td>postgres</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85685</td>
<td>www</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>150M</td>
<td>35204K</td>
<td>accept 0</td>
<td>0:07</td>
<td>7.57%</td>
<td>php-cgi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85274</td>
<td>www</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>149M</td>
<td>40868K</td>
<td>sbwait 3</td>
<td>0:27</td>
<td>5.18%</td>
<td>php-cgi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85267</td>
<td>www</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>151M</td>
<td>40044K</td>
<td>sbwait 2</td>
<td>0:33</td>
<td>4.59%</td>
<td>php-cgi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85884</td>
<td>www</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>150M</td>
<td>41584K</td>
<td>accept 2</td>
<td>0:14</td>
<td>4.59%</td>
<td>php-cgi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85887</td>
<td>psql</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>951M</td>
<td>128M</td>
<td>sbwait 1</td>
<td>0:04</td>
<td>4.20%</td>
<td>postgres</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85886</td>
<td>psql</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>949M</td>
<td>161M</td>
<td>sbwait 0</td>
<td>0:08</td>
<td>3.37%</td>
<td>postgres</td>
<td></td>
<td></td>
</tr>
<tr>
<td>86459</td>
<td>psql</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>949M</td>
<td>75960K</td>
<td>sbwait 2</td>
<td>0:01</td>
<td>3.37%</td>
<td>postgres</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85279</td>
<td>psql</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>950M</td>
<td>192M</td>
<td>sbwait 2</td>
<td>0:14</td>
<td>2.39%</td>
<td>postgres</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85269</td>
<td>psql</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>950M</td>
<td>199M</td>
<td>sbwait 1</td>
<td>0:19</td>
<td>2.20%</td>
<td>postgres</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85268</td>
<td>www</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>152M</td>
<td>44356K</td>
<td>sbwait 2</td>
<td>0:32</td>
<td>1.17%</td>
<td>php-cgi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85273</td>
<td>psql</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>950M</td>
<td>215M</td>
<td>sbwait 0</td>
<td>0:19</td>
<td>1.17%</td>
<td>postgres</td>
<td></td>
<td></td>
</tr>
<tr>
<td>97082</td>
<td>psql</td>
<td>1</td>
<td>44</td>
<td>0</td>
<td>26020K</td>
<td>6832K</td>
<td>select 0</td>
<td>46:55</td>
<td>0.00%</td>
<td>postgres</td>
<td></td>
<td></td>
</tr>
<tr>
<td>892</td>
<td>root</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>3160K</td>
<td>8K</td>
<td>-</td>
<td>2</td>
<td>13:33</td>
<td>0.00%</td>
<td>nfsd</td>
<td></td>
</tr>
<tr>
<td>1796</td>
<td>root</td>
<td>1</td>
<td>44</td>
<td>0</td>
<td>19780K</td>
<td>13660K</td>
<td>select 3</td>
<td>12:43</td>
<td>0.00%</td>
<td>Xvfb</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Is the application parallel?
Application Tuning
Resource Utilization

- Memory Bound?

- Know your max theoretical memory bandwidth
MPI applications have added communication complexity

Application Tuning
What’s Next?

• If your Hotspots are common algorithms:
  • Look for optimized libraries

• If your Hotspots are uncommon:
  • Compiler optimizations
  • Expert analysis and refactoring of an algorithm
    • The opposite of “low-hanging fruit”
  • Deeper analysis of hardware performance
    • More on this later

• If the system is underutilized:
  • Add parallelism - multi-thread or multi-process
    • OpenMP, TBB, Cilk, MPI, etc...

- Tools can help you determine where to look and may identify some issues.
- Some tools may provide suggestions for fixes.
- In the end – the developer and/or expert has to make the changes and decisions – there is no silver bullet.
Optimization: A Top-down Approach

**Hardware Tuning:**
- BIOS (TB, HT)
- Memory
- Network I/O
- Disk I/O

**OS Tuning:**
- Page size
- Swap file
- RAM Disk
- Power settings
- Network protocols

**Better Application Design:**
- Parallelization
- Fast algorithms / data bases
- Programming language and RT libs
- Performance libraries
- Driver tuning

**Tuning for Microarchitecture:**
- Compiler settings/Vectorization
- Memory/Cache usage
- CPU pitfalls
Microarchitecture Tuning

Who: **Architecture Experts**
Software Developers, Performance Engineers, Domain Experts

How:
- Use architecture specific hardware events
- Use predefined metrics and best known methods
  - Often hardware specific
  - (Hopefully) provided by the vendor
- Tools make this possible for the non-expert
  - Linux perf
  - Intel® VTune™ Amplifier XE
- Follow the Top-Down Characterization
  - Locate the hardware bottlenecks

Now we’re getting into Intel specific tuning
Introduction to Performance Monitoring Unit (PMU)

- Registers on Intel CPUs to count architectural events
  - E.g. Instructions, Cache Misses, Branch Mispredict

- Events can be counted or sampled
  - Sampled events include Instruction Pointer

- Raw event counts are difficult to interpret
  - Use a tool like VTune or Perf with predefined metrics
### Raw PMU Event Counts vs Metrics

#### Grouping: Function / Call Stack

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU_CLK_UNHALTED</th>
<th>CPU_CLK_UNHALTED</th>
<th>LPD_PEND</th>
<th>OFFS</th>
<th>BR_MISP</th>
<th>CPU_CLK_UNHALTED</th>
<th>CYCLE_AC</th>
<th>CYCLE_AC</th>
<th>DTL</th>
<th>DTLB_L1</th>
<th>DTLB_L2</th>
<th>DTLB_L3</th>
<th>DTLB_L4</th>
<th>DTLB_L5</th>
<th>ICACHE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere_intersect</td>
<td>13,604,002,456</td>
<td>14,118,021,177</td>
<td>12,572,018,658</td>
<td>6,344,009,516</td>
<td>52,001,128</td>
<td>14,924,022,386</td>
<td>5,402,003,516</td>
<td>4,374,006,396</td>
<td>0</td>
<td>0</td>
<td>234,000,351</td>
<td>25,000,039</td>
<td>0</td>
<td>7,800,234</td>
<td>0</td>
</tr>
<tr>
<td>Grid_bounds_intersect</td>
<td>8,706,013,059</td>
<td>9,134,013,701</td>
<td>8,494,012,741</td>
<td>2,428,006,357</td>
<td>0</td>
<td>15,600,351</td>
<td>9,464,014,196</td>
<td>3,016,004,524</td>
<td>2,838,004,212</td>
<td>0</td>
<td>104,000,156</td>
<td>25,000,039</td>
<td>0</td>
<td>10,400,312</td>
<td>0</td>
</tr>
<tr>
<td>__lkm_end_split_barrier</td>
<td>984,001,476</td>
<td>1,004,001,506</td>
<td>672,001,003</td>
<td>1,494</td>
<td>0</td>
<td>15,600,351</td>
<td>962,001,443</td>
<td>312,000,468</td>
<td>286,000,429</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2,400,006,357</td>
</tr>
<tr>
<td>__kmp_y86_pause</td>
<td>228,000,342</td>
<td>224,000,336</td>
<td>122,000,182</td>
<td>0</td>
<td>10,400,312</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2,400,006,357</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>__ladder</td>
<td>216,000,312</td>
<td>242,000,336</td>
<td>142,000,213</td>
<td>104,000,156</td>
<td>0</td>
<td>208,000,312</td>
<td>104,000,156</td>
<td>52,000,078</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>__Raypnt</td>
<td>206,000,309</td>
<td>210,000,315</td>
<td>208,000,312</td>
<td>0</td>
<td>0</td>
<td>234,000,351</td>
<td>52,000,078</td>
<td>78,000,117</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>__pos2grid</td>
<td>204,000,306</td>
<td>248,000,372</td>
<td>180,000,270</td>
<td>26,000,039</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>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>__tri_intersect</td>
<td>168,000,232</td>
<td>200,000,312</td>
<td>180,000,270</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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>__VScales</td>
<td>124,000,185</td>
<td>126,000,189</td>
<td>164,000,264</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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>__lkm_yield</td>
<td>36,000,144</td>
<td>38,000,147</td>
<td>200,000,300</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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Selected 1 row(s):

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU_CLK_UNHALTED</th>
<th>CPU_CLK_UNHALTED</th>
<th>LPD_PEND</th>
<th>OFFS</th>
<th>BR_MISP</th>
<th>CPU_CLK_UNHALTED</th>
<th>CYCLE_AC</th>
<th>CYCLE_AC</th>
<th>DTL</th>
<th>DTLB_L</th>
<th>DTLB_L</th>
<th>DTLB_L</th>
<th>DTLB_L</th>
<th>DTLB_L</th>
<th>ICACHE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere_intersect</td>
<td>13,604,002,456</td>
<td>14,118,021,177</td>
<td>12,572,018,658</td>
<td>6,344,009,516</td>
<td>52,001,128</td>
<td>14,924,022,386</td>
<td>5,402,003,516</td>
<td>4,374,006,396</td>
<td>0</td>
<td>0</td>
<td>234,000,351</td>
<td>25,000,039</td>
<td>0</td>
<td>7,800,234</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Grouping: Function / Call Stack

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>Clockticks Retired</th>
<th>Instruction Retired</th>
<th>CPI Rate</th>
<th>MUX Reliability</th>
<th>Filled Pipeline Slots</th>
<th>Unfilled Pipeline Slots (Stalls)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>__grid_intersect</td>
<td>14,118,021,177</td>
<td>12,572,018,658</td>
<td>1.123</td>
<td>0.046</td>
<td>0.246</td>
<td>0.033</td>
</tr>
<tr>
<td>Sphere_intersect</td>
<td>9,134,013,701</td>
<td>8,494,012,741</td>
<td>1.075</td>
<td>0.955</td>
<td>0.250</td>
<td>0.065</td>
</tr>
<tr>
<td>Grid_bounds_intersect</td>
<td>1,004,001,506</td>
<td>672,001,003</td>
<td>1.494</td>
<td>0.958</td>
<td>0.227</td>
<td>0.000</td>
</tr>
<tr>
<td>__lkm_end_split_barrier</td>
<td>624,000,936</td>
<td>460,000,690</td>
<td>1.357</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Pos2grid</td>
<td>248,000,372</td>
<td>180,000,270</td>
<td>1.378</td>
<td>0.636</td>
<td>0.357</td>
<td>0.000</td>
</tr>
<tr>
<td>Shader</td>
<td>242,000,363</td>
<td>142,000,213</td>
<td>1.794</td>
<td>0.860</td>
<td>0.322</td>
<td>0.000</td>
</tr>
<tr>
<td>__kmp_y86_pause</td>
<td>224,000,336</td>
<td>122,000,183</td>
<td>1.836</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>__Raypnt</td>
<td>210,000,315</td>
<td>208,000,312</td>
<td>1.010</td>
<td>0.877</td>
<td>0.093</td>
<td>0.279</td>
</tr>
</tbody>
</table>

Selected 1 row(s):

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>Clockticks Retired</th>
<th>Instruction Retired</th>
<th>CPI Rate</th>
<th>MUX Reliability</th>
<th>Filled Pipeline Slots</th>
<th>Unfilled Pipeline Slots (Stalls)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>__grid_intersect</td>
<td>14,118,021,177</td>
<td>12,572,018,658</td>
<td>1.123</td>
<td>0.046</td>
<td>0.246</td>
<td>0.033</td>
</tr>
</tbody>
</table>
Adding Regression Tests for Performance

• Regression testing isn’t just for bugs

1. Create a baseline performance characterization
2. After each change or at a regular interval
   1. Compare new results to baseline
   2. Compare new results to previous results
   3. Evaluate the change
3. goto (1)

• Performance tuning is easier if it’s always on your mind and integrated into your development
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY
ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS
DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR
IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES
RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY
PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on
Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using
specific computer systems, components, software, operations and functions. Any change to any of
those factors may cause the results to vary. You should consult other information and performance
tests to assist you in fully evaluating your contemplated purchases, including the performance of that
product when combined with other products.

Copyright ©, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk
are trademarks of Intel Corporation in the U.S. and other countries.

**Optimization Notice**

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that
are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and
other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on
microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended
for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for
Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information
regarding the specific instruction sets covered by this notice.

Notice revision #20110804