Intel VTune Amplifier XE

Seminar Software Profiling
Lena Herscheid
Supervisor: Dr. Peter Tröger
May 27th 2013
Agenda

• **Overview**
  • Sampling Approach

• **Features**
  • Algorithm Analysis
  • Hardware Profiling

• **How It Works**
  • User mode Sampling
  • Hardware Event Sampling

• **Evaluation**
Overview

• Commercial profiling tool
  • part of Intel Parallel Studio or standalone
  • provides CLI and GUI
  • Visual Studio and Eclipse plugins

• Fortran, C, C++, Java, .NET, Assembly

• Linux and Windows
The Sampling Approach

an ever-present trade-off: Overhead ↔ Information

idea: collect information sporadically

• time-driven: uniform time period between samples
• event-driven: uniform number of events between samples

+ less intrusive → no side effects introduced by profiling
+ small impact on execution speed → detect timing issues accurately
+ low overhead
  + user-mode sampling: about 5% (default interval of 10ms)
  + hardware event-based sampling: about 2% (1ms sampling interval)
Sampling Mistakes

• **the entire sampling interval** before an event is attributed to the code context
  • 10-2000k events depending on sampling interval
  • negligible for many samples and frequent events

• per-user filtering switched off by default
  • (and cannot be switched on again after installation…)
  • samples from **all processes** on the system are collected!

• call graphs are approximated and can be misleading
  • with infrequently asymmetric call patterns
Profiling Workflow

applications are profiled under different configurations
• collects a subset of supported metrics in one analysis run
• re-configure and re-run multiple times
  • to keep overhead low for each run
• debug symbols needed
  • for source-code-line granularity
Agenda

• Overview
  • Sampling Approach

• Features
  • Algorithm Analysis
  • Hardware Profiling

• How It Works
  • User mode Sampling
  • Hardware Event Sampling

• Evaluation
Algorithm Analysis

Choose viewpoint depending on profiling objective:

• Where are significant portions of time spent?
  → Hotspot Analysis

• How well is CPU time utilized? How well are threads scheduled?
  → Concurrency Analysis

• What are causes of ineffective CPU utilization?
  → Waits and Locks Analysis
VTune Examples

• CPU/Memory benchmarks ([http://www.roylongbottom.org.uk](http://www.roylongbottom.org.uk))
  • Whetstone: floating point computations
  • RandMem: memory read/write

• System configuration:

  Processor: Intel(R) Core(TM) i3-2310M CPU @ 2.10GHz  2.10 GHz
  Installed memory (RAM): 6.00 GB (5.84 GB usable)
  System type: 64-bit Operating System, x64-based processor
Hotspot Analysis (Whetstone)

### Top-Down View (Call Tree)

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU Time by Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Idle</td>
</tr>
<tr>
<td></td>
<td>5.509s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU Time</th>
<th>Source File</th>
<th>Start Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>whetstone</td>
<td>5.509s</td>
<td>whetstones.cpp</td>
<td>0x140027b0</td>
</tr>
<tr>
<td>main</td>
<td>5.509s</td>
<td>whetstones.cpp</td>
<td>0x140003c0</td>
</tr>
</tbody>
</table>

### Bottom-Up View

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU Time by Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Idle</td>
</tr>
<tr>
<td></td>
<td>5.509s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU Time</th>
<th>Source File</th>
<th>Start Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>whetstone</td>
<td>5.509s</td>
<td>whetstones.cpp</td>
<td>0x140027b0</td>
</tr>
<tr>
<td>main</td>
<td>5.509s</td>
<td>whetstones.cpp</td>
<td>0x140003c0</td>
</tr>
</tbody>
</table>

Top Hotspots:
- `whetstone`: 5.509s
- `whetstone2`: 5.198s

Other functions:
- `pa2`: 0.370s
- `pa`: 0.370s
- `calculateMhz`: 0.071s
- [Others]: 0.113s

**Intel VTune**
Lena Herscheid
Hotspot Analysis (Whetstone)

```c
if (calibrate == 0)
    ResumeThread(hThreadHandle);
    SetThreadPriority(hThreadHandle, THREAD_PRIORITY BELOW_NORMAL);
    mainCount = mainCount + 1;
for(i=i; i<n5; i++)
    {
        x = (SPDP)(t*atan(t2*sin(x)*cos(x)/(cos(x-y)+cos(x-y)-1.0)));
        y = (SPDP)(t*atan(t2*sin(y)*cos(y)/(cos(x-y)+cos(x-y)-1.0)));
    }
    t = (SPDP)1.0 - t;
if (calibrate == 0)
    {
        SuspendThread(hThreadHandle);
    }
    while ((mainCount + threadCount) < (xtrc));
t = t0;
end_time();
timed = secs;
```
Concurrency Analysis

- **CPU Usage**: threads consuming CPU time
- **Thread Concurrency**: threads in runnable state

- thread concurrency > CPU usage: thread oversubscription
- CPU usage > thread concurrency: spinning threads
Concurrency Analysis (Whetstone)
Concurrency Analysis (Whetstone)

```c
421    threadCount = 0;
422
423    do
424    {
425        if (calibrate == 0)
426            {
427                ResumeThread(hThreadHandle);
428                SetThreadPriority(hThreadHandle, THREAD_PRIORITY BELOW);
429            }
430            mainCount = mainCount + 1;
431            for(i=0; i<1*nImgT; i++)
432                {
433                    e1[0] = (e1[0] + e1[1] + e1[2] - e1[3]) * t;
437                }
438             t = (SPDP)1.0 - t;
439        if (calibrate == 0)
440            {
441                SuspendThread(hThreadHandle);
442             tt = t0;
```
Locks and Waits Analysis (Whetstone)

Top Waiting Objects

This section lists the objects that spent the most time waiting in your application. Objects can wait on specific calls, such as sleep0 or I/O, or on contended synchronizations. A significant amount of Wait time associated with a synchronization object reflects high contention for that object and, thus, reduced parallelism.

<table>
<thead>
<tr>
<th>Sync Object</th>
<th>Wait Time</th>
<th>Wait Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resume Thread</td>
<td>0.349s</td>
<td>5,397</td>
</tr>
<tr>
<td>Stream 0x5772d99f</td>
<td>0.012s</td>
<td>39</td>
</tr>
<tr>
<td>Stream Whetstone 0x45b11562</td>
<td>0.000s</td>
<td>4</td>
</tr>
<tr>
<td>Sleep</td>
<td>1.000s</td>
<td>1</td>
</tr>
</tbody>
</table>
Locks and Waits Analysis (Whetstone)

<table>
<thead>
<tr>
<th>Source</th>
<th>Line</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>370</td>
<td>if (errors == 0)</td>
</tr>
<tr>
<td></td>
<td>371</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>372</td>
<td>fprintf(outfile, &quot; Numeric results were as expected\n&quot;);</td>
</tr>
<tr>
<td></td>
<td>373</td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>374</td>
<td>fprintf(outfile, \n&quot;);</td>
</tr>
<tr>
<td></td>
<td>375</td>
<td>ResumeThread(hThreadHandle);</td>
</tr>
<tr>
<td></td>
<td>376</td>
<td>goThread = FALSE;</td>
</tr>
<tr>
<td></td>
<td>377</td>
<td>CloseHandle(hThreadHandle);</td>
</tr>
<tr>
<td></td>
<td>378</td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>379</td>
<td>for (i=0; i&lt;9; i++)</td>
</tr>
<tr>
<td></td>
<td>380</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>381</td>
<td>fprintf(outfile, &quot;$%\n&quot;, configdata[i]);</td>
</tr>
<tr>
<td></td>
<td>382</td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>383</td>
<td>fprintf(outfile, &quot;$%\n&quot;);</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>385</td>
<td>close (outfile);</td>
</tr>
<tr>
<td></td>
<td>386</td>
<td>Sleep(1000);</td>
</tr>
<tr>
<td></td>
<td>387</td>
<td>return 0;</td>
</tr>
</tbody>
</table>

Intel VTune Lena Herscheid
Hardware Profiling

- CPU-specific events and metrics
- profiling mode defines presets for Sample After Value and Events

General Exploration

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Sample After</th>
<th>LBR Filter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARITH.FPU_DIV_ACTIVE</td>
<td>2000000</td>
<td>None</td>
<td>Cycles when divider is busy executing divide operations</td>
</tr>
<tr>
<td>BR_MISP_RETIRED.ALL_BRANCHES_PS</td>
<td>400000</td>
<td>None</td>
<td>Mispredicted macro branch instructions retired. (Precise Event - PEBS)</td>
</tr>
<tr>
<td>CPU_CLK_UNHALTED.JEF_ TSC</td>
<td>2000000</td>
<td>None</td>
<td>Reference cycles when the core is not in halt state.</td>
</tr>
<tr>
<td>CPU_CLK_UNHALTED.THREAD</td>
<td>2000000</td>
<td>None</td>
<td>Core cycles when the core is not in halt state.</td>
</tr>
<tr>
<td>DSE2MTE_SWITCHES.PENALTY_CYCLES</td>
<td>2000000</td>
<td>None</td>
<td>Decode Stream Buffer (DSB)-to-MTE switch true penalty cycles</td>
</tr>
<tr>
<td>DTLB_LOAD_MISSES_STLB_HIT</td>
<td>100000</td>
<td>None</td>
<td>Load operations that miss the first DTLB level but hit the second and do not miss the second DTLB level.</td>
</tr>
<tr>
<td>DTLB_LOAD_MISSES.WALK_DURATION</td>
<td>2000000</td>
<td>None</td>
<td>Cycles when PMH is busy with page walks</td>
</tr>
<tr>
<td>ICACHE_MISS</td>
<td>2000000</td>
<td>None</td>
<td>Instruction cache, streaming buffer and victim cache misses</td>
</tr>
<tr>
<td>IDQ_MISC_CYCLES</td>
<td>2000000</td>
<td>None</td>
<td>Cycles when uops are being delivered to instruction Decode Queue (IDQ)</td>
</tr>
<tr>
<td>IPCORE_HIT_DELIVERED_CODE</td>
<td>2000000</td>
<td>None</td>
<td>Cycles when uops are being delivered to instruction Decode Queue (IDQ)</td>
</tr>
</tbody>
</table>

NOTE: For analysis purposes, Intel VTune Amplifier XE 2013 may adjust the Sample After values in the table below by a multiplier. The multiplier depends on the value of the Duration time estimate option specified in the Project Properties dialog.
Hardware Profiling: General Exploration (RandMem)
# Hardware Profiling: Memory Analysis

## RandMem

### Hardware Events

<table>
<thead>
<tr>
<th>Hardware Event Type</th>
<th>Hardware Event Count</th>
<th>Hardware Event Sample Count</th>
<th>Events Per Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_CLK_UNHALTED.THREAD</td>
<td>52,524,000,000</td>
<td>26,262</td>
<td>200000</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS</td>
<td>129,840,000</td>
<td>2,164</td>
<td>20000</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT_PS</td>
<td>14,100,000</td>
<td>235</td>
<td>20000</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS_PS</td>
<td>9,480,000</td>
<td>158</td>
<td>20000</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE_PS</td>
<td>374,100,000</td>
<td>1,247</td>
<td>10000</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_MISC_RETIRED.LLC_MISS_PS</td>
<td>125,810,000</td>
<td>4,227</td>
<td>10000</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_RETIRED_HIT_LFB_PS</td>
<td>2,693,800,000</td>
<td>8,996</td>
<td>100000</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_RETIRED_L1_HIT_PS</td>
<td>22,740,000,000</td>
<td>3,790</td>
<td>2000000</td>
</tr>
<tr>
<td>MEM_LOAD_UOPS_RETIRED_L2_HIT_PS</td>
<td>638,600,000</td>
<td>2,112</td>
<td>100000</td>
</tr>
<tr>
<td>MEM_UOPS_RETIRED_ALL LOADS_PS</td>
<td>27,210,000,000</td>
<td>4,533</td>
<td>2000000</td>
</tr>
<tr>
<td>MEM_UOPS_RETIRED_ALL STORES_PS</td>
<td>4,990,000,000</td>
<td>832</td>
<td>2000000</td>
</tr>
</tbody>
</table>

### Call Stack

<table>
<thead>
<tr>
<th>Call Stack</th>
<th>CPU_CLK_UNHALTED.THREAD</th>
<th>MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS</th>
<th>MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT_PS</th>
<th>MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS_PS</th>
<th>MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE_PS</th>
<th>MEM_LOAD_UOPS_MISC_RETIRED.LLC_MISS_PS</th>
<th>MEM_LOAD_UOPS_RETIRED_HIT_LFB_PS</th>
<th>MEM_LOAD_UOPS_RETIRED_L1_HIT_PS</th>
<th>MEM_LOAD_UOPS_RETIRED_L2_HIT_PS</th>
<th>MEM_UOPS_RETIRED_ALL LOADS_PS</th>
<th>MEM_UOPS_RETIRED_ALL STORES_PS</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread1TestRW</td>
<td>12,600,000,000</td>
<td>40,740,000</td>
<td>7,140,000</td>
<td>4,020,000</td>
<td>72,600,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thread1TestRD</td>
<td>1,472,000,000</td>
<td>35,622,000</td>
<td>420,000</td>
<td>2,400,000</td>
<td>188,600,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thread2TestRW</td>
<td>7,400,000,000</td>
<td>32,100,000</td>
<td>6,100,000</td>
<td>1,880,000</td>
<td>25,200,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thread2TestRD</td>
<td>5,873,000,000</td>
<td>17,700,000</td>
<td>360,000</td>
<td>1,140,000</td>
<td>83,400,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>runTests</td>
<td>4,040,000,000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initptd</td>
<td>1,948,000,000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FitzGetVal</td>
<td>1,713,000,000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rand</td>
<td>1,498,000,000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getpidt</td>
<td>1,136,000,000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Intel VTune** Lena Herscheid 19
Other Features

- graphical tool for comparing analysis results
- user task API
  - annotate source code with user tasks
- remote agents for Linux
  - profile Linux applications remotely, using Windows GUI
- MPI Profiling (with Intel’s MPI implementation)
- JIT Profiling
  - application must link against jitprofiling library and issue special events
  - LLVM supports the VTune API
- GPU Profiling (for OpenCL running on Intel GPUs)
Agenda

• Overview
  • Sampling Approach

• Features
  • Algorithm Analysis
  • Hardware Profiling

• How It Works
  • User mode Sampling
  • Hardware Event Sampling

• Evaluation
User Mode Sampling

• time-driven approach
• embeds sampling library using **LD_PRELOAD**
  • setup timer for each thread
  • upon timer expiration, issue **SIGPROF**
  • interrupt process, collector handles signal by sampling stack information
Correlating Event-based sampling with thread quanta

Collected information (depending on configuration)

- call stack information
- branching information
- event counter values
- timestamp
- power consumption values
Hardware Event Based Sampling

• event-driven

• implemented in a driver

• Performance Monitoring Unit (PMU) periodically interrupts processor
  • collect information after **SAV** (Sample After Value) events
    • automatic SAV calibration: needs additional run
    • “If you set the Sample After value, specify a number that is sufficiently large. A value that is too low increases the sample rate causing unpredictable results.”

• hardware limit on the number of simultaneously sampled events (4)
  • for more events: round-robin multiplexing
Performance Monitoring Unit

- one PMU per core
- one PMU in uncore region

- elapsed cycles
- L1, L2 cache events
- processed instructions
- ...

- uncore bound
- QPI events
- L3 cache events
- memory controller events
- ...

Intel VTune         Lena Herscheid
ITLB misses

front-end bound
cannot fetch & decode fast enough

back-end bound
no more operations accepted by back-end

FP assists
denormal values

branch mispredictions

L1/L2-cache bound
stalls due to cache misses
Agenda

• Overview
  • Sampling Approach

• Features
  • Algorithm Analysis
  • Hardware Profiling

• How It Works
  • User mode Sampling
  • Hardware Event Sampling

• Evaluation
Evaluation

+ detailed Hardware information
+ tool-supported low-level performance tuning
+ low overhead during application run
+ good visualizations

– long post-processing
– the absolute numbers are not reliable (sampling errors)
  – influenced by system state and randomness

– single user license: 900$
– confusingly many features
Evaluation (ctd.)

Use VTune...
+ if you target an Intel architecture
  + and are willing to optimize for it
+ if the application is compute-intensive
+ if hardware awareness counts
  + strict performance requirements
  + low-level programming language

Otherwise, use a simpler alternative
• Sampling Profilers:
  Very Sleepy for Windows, OProfile/gprof for Linux, AMD CodeAnalyst
• for just accessing PMUs: Intel Performance Counter (BSD license)
Sources


• Benchmarks: http://www.roylongbottom.org.uk

• http://www.realworldtech.com/vtune/

• Survey of Software Monitoring and Profiling Tools
  http://www.cse.wustl.edu/~jain/cse567-06/ftp/sw_monitors2.pdf