Microarchitectural Analysis with Intel® VTune™ Amplifier XE

Michael Klemm
Software & Services Group
Developer Relations Division
Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm
## Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the “Intel® Compiler User and Reference Guides” under “Compiler Options.” Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools 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 Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® 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.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101
Agenda

• Introduction to Parallel Studio XE
• About PMUs, EBS, counting
• VTune Amplifier XE: User Interface
• VTune Amplifier XE: Analysis Types
• Get your hands dirty (lab exercises)…
Agenda

• Introduction to Parallel Studio XE
• About PMUs, EBS, counting
• VTune Amplifier XE: User Interface
• VTune Amplifier XE: Analysis Types
• Get your hands dirty (lab exercises)…
Three Product Lines for Diverse Needs

Essential Performance
C/C++ developers
Microsoft Visual Studio*
Take advantage of multicore

Advanced Performance
C++ and Fortran developers
Windows* and Linux*
High performance, cross platform apps

Distributed Performance
C++ and Fortran developers
on Windows* and Linux*
High performance MPI clusters

http://www.intel.com/software/products
Software Tools to Drive Multicore

For Microsoft Visual Studio* C++ architects, developers, and software innovators creating parallel Windows* applications.

- Microsoft Visual Studio* plug-ins
- End-to-end product suite for parallelism
- Forward scaling to many-core

- Parallel Advisor
- Parallel Composer
- Parallel Inspector
- Parallel Amplifier
Intel® Parallel Studio

- The ultimate all-in-one toolset for the software development lifecycle
  - DESIGN
  - CODE & DEBUG
  - VERIFY
  - TUNE
## Intel® Parallel Studio XE 2011
Powerful tools to create fast, reliable and secure code

<table>
<thead>
<tr>
<th>Phase</th>
<th>Productivity Tool</th>
<th>Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advanced Build &amp; Debug</strong></td>
<td><strong>Intel® Composer XE</strong></td>
<td>C/C++ and Fortran compilers, performance libraries, and parallel models</td>
<td>Application performance, scalability and quality for current multicore and future many-core systems.</td>
</tr>
<tr>
<td><strong>Advanced Verify</strong></td>
<td><strong>Intel® Inspector XE</strong></td>
<td>Memory &amp; threading error checking tool for higher code reliability &amp; quality</td>
<td>Increases productivity and lowers cost, by catching memory and threading defects early</td>
</tr>
<tr>
<td><strong>Advanced Tune</strong></td>
<td><strong>Intel® VTune™ Amplifier XE</strong></td>
<td>Performance Profiler to optimize performance and scalability</td>
<td>Removes guesswork, saves time, makes it easier to find performance and scalability bottlenecks Combines ease of use with deeper insights.</td>
</tr>
</tbody>
</table>

+ Extra Feature in Parallel Studio XE 2011 only:
Static Security Analyzer
Windows & Linux Versions Available

• **Microsoft Windows* OS**
  - Windows XP*, Windows Vista*, Windows 7*
  - Integration with Microsoft Visual Studio* 2005, 2008 and 2010
  - Standalone GUI and command line
  - IA32 and Intel® 64

• **Linux* OS**
  - Red Hat Enterprise Linux* 4 (Update 8), 5 (Updates 4, 5), 6
  - SUSE LINUX Enterprise Server* 10, 11
  - Fedora* 12, 13
  - Additional distributions may also work – Some components have been tested on additional distributions
  - Standalone GUI and command line
  - IA32 and Intel® 64

• Latest Details in System Requirements within the Release notes

• **Single user and floating licenses available**
Agenda

• Introduction to Parallel Studio XE
• About PMUs, EBS, counting
• VTune Amplifier XE: User Interface
• VTune Amplifier XE: Analysis Types
• Get your hands dirty (lab exercises)...
Performance Monitoring Unit (PMU)

• Introduced in Pentium with a set of model-specific performance monitoring counter MSRs

• Performance monitoring mechanism and performance events for Pentium, P6, and P4 family are not architectural
  – All model specific

• Two classes of performance monitoring capabilities
  – Architectural and non-architectural
  – With Core Solo and Duo architectures, both architectural and non-architectural performance events are introduced
  – Non architectural events cannot be enumerated via CPUID
  – Architectural events can be enumerated via CPUID.0AH
Performance Monitoring Unit

• Architectural events
  – Smaller set of available events
  – Events are architectural when they behave consistently across microarchitectures
  – Configuring an architectural performance monitoring event involves programming performance event select registers
  – Architectural performance events can be collected using general-purpose performance counters

• Non-architectural events
  – Bigger set of available events
  – Non-architectural performance events use event select values that are model specific.
  – Non-architectural performance events can be collected using general-purpose performance counters, or fixed-function performance counters
Performance Monitoring Unit

• Three categories of events
  – Fixed events: clockticks, instructions retired, reference clockticks
  – Programmable events
    – Many events, up to 4/8 can be counted simultaneously when HT on/off
    – May skid depending on penalty and distance from retirement
  – Precise events
    – Sub group of the programmable events
    – Point precisely to the instruction executed after the one that caused the event
    – up to 4 can be counted simultaneously
Event Based Performance Analysis
Intel Xeon processor 5500 series

- The performance monitoring events on Core i7 and follow on series can be used to analyze the interaction between software (code and data) and microarchitectural units.
Event Based Performance Analysis
Intel Xeon processor 5500 series

- **Per-core PMU**: Each core provides 4 programmable counters and 3 fixed counters.
  - The programmable per-core counters can be configured to investigate front-end/micro-op flow issues, stalls inside a processor core.
  - Additionally, a subset of per-core PMU events support precise event-based sampling (PEBS).
  - Load latency measurement facility is new in Intel Core i7 processor and Intel Xeon processor 5500.

- **Uncore PMU**: Provides 8 programmable counters and 1 fixed counter.
  - The programmable per-core counters can be configured to characterize L3 and Intel QPI operations, local and remote data memory accesses.
Event Based Performance Analysis
Event Based Sampling (EBS)

- Both architectural and non-architectural processor events can be monitored using sampling and counting technologies

**Sampling:** Allows to profile all active software on the system, including operating system, device driver, and application software.

- Event-based samples are collected periodically after a specific number of processor events have occurred while the program is running
- The program is interrupted, allowing the interrupt handling driver to collect the Instruction Pointer (IP), load module, thread and process ID's
- Instruction pointer is then used to derive the function and source line number from the debug information created at compile time
How Event Based Sampling (EBS) Works

Select Event Signal → Count Down "Sample After" Value → Underflow to Zero → Internal Interrupt Controller

- A performance counter increments on the CPU every time an event occurs
- A sample of the execution context is recorded every time a performance counter overflows

Events = samples * sample after value
Event Based Performance analysis
Event Based Counting (EBC)

**Counting:** allows counting of the PMU events periodically

- Core and Uncore PMU events are *counted* using *time-based sampling*
- Sampling is not supported for the uncore PMU events, hence counting is used to monitor uncore events

- This means that you can monitor these events for the whole system over a designated time range, but you won’t be able to see how many of the certain events came from various functions/processes/modules.
  - You can only see the total count for the entire system
Agenda

• Introduction to Parallel Studio XE
• About PMUs, EBS, counting
• VTune Amplifier XE: User Interface
• VTune Amplifier XE: Analysis Types
• Get your hands dirty (lab exercises)
**Intel® VTune™ Amplifier XE Performance Profiler**

**Where is my application...**

**Spending Time?**
- Focus tuning on functions taking time
- See call stacks
- See time on source

**Wasting Time?**
- See cache misses on your source
- See functions sorted by # of cache misses

**Waiting Too Long?**
- See locks by wait time
- Red/Green for CPU utilization during wait

**Windows & Linux**
- Low overhead
- No special recompiles
Intel® VTune™ Amplifier XE
Tune Applications for Scalable Multicore Performance

- Fast, Accurate Performance Profiles
  - Hotspot (Statistical call tree)
  - Hardware-Event Based Sampling

- Thread Profiling
  - Visualize thread interactions on timeline
  - Balance workloads

- Easy set-up
  - Pre-defined performance profiles
  - Use a normal production build

- Compatible
  - Microsoft, GCC, Intel compilers
  - C/C++, Fortran, Assembly, .NET
  - Latest Intel® processors and compatible processors

- Find Answers Fast
  - Filter extraneous data
  - View results on the source / assembly
  - Event multiplexing

- Windows or Linux
  - Visual Studio Integration (Windows)
  - Standalone user i/f and command line
  - 32 and 64-bit

1 IA32 and Intel® 64 architectures. Many features work with compatible processors. Event based sampling requires a genuine Intel® Processor.
VTune™ Amplifier XE

GUI Layout

- Menu and tool bars
- Analysis Type
- Viewpoint currently being used
- Tabs within each result
- Grid area
- Stack Pane
- Filter area
- Timeline area

Current grouping

- Analyze type
- Grid area
- Stack Pane
- Filter area
- Timeline area
VTune™ Amplifier XE
GUI Layout

Adjust Data Grouping
- Function - Call Stack
- Module - Function - Call Stack
- Source File - Function - Call Stack
- Thread - Function - Call Stack
... (Partial list shown)

Click [+] for Call Stack

Double Click Function to View Source

Filter by Timeline Selection (or by Grid Selection)
- Zoom In And Filter On Selection
- Filter In by Selection
- Remove All Filters

Filter by Module & Other Controls

No filters are applied
Module: [All]
VTune™ Amplifier XE

GUI Layout

**Time on Source / Asm**

Quick Asm navigation:
Select source to highlight Asm

Quickly scroll to hot spots.
Scroll Bar “Heat Map” is an overview of hot spots

Right click for instruction reference manual

Click jump to scroll Asm
VTune™ Amplifier XE

GUI Layout

- Optional: Use API to mark frames and user tasks
- Optional: Add a mark during collection
Profile an Application

- Path to executable
- Command line arguments
- Where to execute the app
- Environment variables (e.g. OMP_NUM_THREADS)
Profile a Running Application
No need to stop and re-launch the app when profiling

- Attach to Process:
  - Hotspot
  - Concurrency
  - Locks & Waits

- Profile System:
  - Lightweight Hotspots
  - Advanced & Custom EBS
  - Optional: Filter by process after collection

(Attach to process is currently only available for Windows)
Agenda

• Introduction to Parallel Studio XE
• About PMUs, EBS, counting
• VTune Amplifier XE: User Interface
• VTune Amplifier XE: Analysis Types
• Get your hands dirty (lab exercises)
Performance Analysis Methodology
Optimization: A Top-down Approach

• Use top down approach
• Understand application and system characteristics
  – Use appropriate tools at each level

- System Config, BIOS, OS, Network I/O, Disk I/O, Database Tuning, etc.
- Application Design, Algorithmic Tuning, Driver Tuning, Parallelization
- Cache/Memory Instructions, SIMD, others
Performance Analysis Methodology
Optimization: A Top-down Approach

- Use top down approach
- Understand application and system characteristics
- Use appropriate tools at each level

- System Config, BIOS, OS
- Network I/O, Disk I/O, Database Tuning, etc.

- Application Design
- Algorithmic Tuning
- Driver Turning
- Parallelization
- Cache/Memory
- Instructions
- SIMD others

- Repeatable
- Representative
- Easy to run
- Verifiable
- Measure elapsed time
- Reasonable coverage
- Precision

- 1. Create a Benchmark
- 2. Collect Data
- 3. Analyze Data and Identify Performance Problems
- 4. Fix the problems in your code or system
- 5. Is Problem Fixed?
- 6. Are performance requirements met?

- System
- Application
- Processor
Kinds of Collection

• **User Mode Sampling and Tracing Analysis**
  - Hotspots, Concurrency, & Lock and Waits
  - Dynamically instruments binary
    - Minimal = Hotspots
    - More = Concurrency, & Locks and Waits
  - Uses OS Timer Service for each thread to collect a sample
  - Collects Call Stack
    - For OS Sample (Configurable)
    - For Instrumented APIs (Configurable)

• **Hardware Event-based Sampling Analysis**
  - Lightweight Hotspots, Advanced Processor Analysis
  - Uses installed driver to configure and collect interrupts from the Performance Monitoring Unit of each Intel CPU Core.
Analysis Types

*Hotspots*

- For each sample, capture execution context, time passed since previous sample and thread CPU time
- Allows time spent in system calls to be attributed to user functions making the call
- Provides additional knobs:
  - The defaults for Hotspot analysis are configurable and can be done so by creating a custom analysis type inherited from Hotspots
Analysis Types

**Lightweight Hotspots**

- New for Amplifier XE
- Similar to Hotspot Analysis
  - Sampling is performed with the SEP collector
  - Driver is required
- Stack walking is not performed
  - Only hotspots are reported
- Samples are taken more frequently, but may have less accurate timing information
- Analysis may be performed for a single application or for the entire system
Lightweight Hotspots vs. Hotspots

• Lightweight Hotspots
  – Uses EBS
  – Has no stack information
  – Lower overhead
  – Faster finalization
  – Requires a driver/privileged mode

• Hotspots
  – Uses TPSS collector
  – Stackwalk is done post process, hence slower finalization times
  – Provides stack information and a statistical call graph
Lightweight Hotspots vs. Hotspots

Smoke

- Mostly correlates, however the default attribution of system time in Hotspots is to the user function making the system call.
### Lightweight Hotspots vs. Hotspots

**Smoke**

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU_CLK_UNCORE</th>
<th>Function - Call Stack</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FireObject::checkCollision</strong></td>
<td>13.3%</td>
<td><strong>FireObject::checkCollision</strong></td>
<td>13.1%</td>
</tr>
<tr>
<td>FireObject::ProcessFireCollisionsRange</td>
<td>7.6%</td>
<td>FireObject::ProcessFireCollisionsRange</td>
<td>6.9%</td>
</tr>
<tr>
<td>D3DXSHProjectCubeMap</td>
<td>3.7%</td>
<td>D3DXSHProjectCubeMap</td>
<td>3.6%</td>
</tr>
<tr>
<td>tbb::internal::task_scheduler_observer_v3::observers::D3DDisassemble10Effect</td>
<td>3.1%</td>
<td>D3DDisassemble10Effect</td>
<td>3.1%</td>
</tr>
<tr>
<td>std::vector&lt; class FireObject *, class std::allocator&lt; FireObject &gt; &gt;</td>
<td>2.9%</td>
<td>std::vector&lt; class FireObject *, class std::allocator&lt; FireObject &gt; &gt;</td>
<td>2.5%</td>
</tr>
<tr>
<td>Ogre::TextureUnitState::getDesiredFormat</td>
<td>2.8%</td>
<td>Ogre::TextureUnitState::getDesiredFormat</td>
<td>2.5%</td>
</tr>
<tr>
<td>RtlInitUnicodeString</td>
<td>2.3%</td>
<td>RtlInitUnicodeString</td>
<td>2.4%</td>
</tr>
<tr>
<td>modf</td>
<td>2.3%</td>
<td>modf</td>
<td>2.4%</td>
</tr>
<tr>
<td>ParticleEmitter::ParticleSystem::update</td>
<td>1.5%</td>
<td>ParticleEmitter::ParticleSystem::update</td>
<td>1.5%</td>
</tr>
<tr>
<td>Animal::UpdateFear</td>
<td>1.5%</td>
<td>Animal::UpdateFear</td>
<td>1.2%</td>
</tr>
<tr>
<td>[fmodex.dll]</td>
<td>1.4%</td>
<td>[fmodex.dll]</td>
<td>1.4%</td>
</tr>
<tr>
<td>[nvd3dum.dll]</td>
<td>1.3%</td>
<td>[nvd3dum.dll]</td>
<td>1.3%</td>
</tr>
<tr>
<td>LdrUnlockLoaderLock</td>
<td>1.3%</td>
<td>LdrUnlockLoaderLock</td>
<td>1.3%</td>
</tr>
<tr>
<td>hkpMoppObbVirtualMachine::queryAabbOnTread</td>
<td>1.1%</td>
<td>hkpMoppObbVirtualMachine::queryAabbOnTread</td>
<td>1.1%</td>
</tr>
<tr>
<td>[nvlddmkm.sys]</td>
<td>1.1%</td>
<td>[nvlddmkm.sys]</td>
<td>1.1%</td>
</tr>
<tr>
<td>ParticleEmitter::FirePatch::initParticle</td>
<td>1.1%</td>
<td>ParticleEmitter::FirePatch::initParticle</td>
<td>1.1%</td>
</tr>
<tr>
<td>KeSynchronizeExecution</td>
<td>1.1%</td>
<td>KeSynchronizeExecution</td>
<td>1.1%</td>
</tr>
</tbody>
</table>

- Setting gives you better correlation to the hotspot report by Lightweight Hotspot analysis type.
Analysis Types

Concurrency

• Similar to the one in Amplifier
• Has the ability to also display utilization based on CPU Usage
• Has newer metrics such as “overhead” and “spin” time
• Has the same kind of knobs available under “Hotspots” and can be configured by creating a custom analysis type
• Overheads are higher than Hotspots as thread waits are monitored to get accurate thread concurrency
Analysis Types

Concurrency
Analysis Types

Lock and Waits

• Similar to the one in Amplifier
• Has newer metrics such as “spin” time
• Has the same kind of knobs available under “Hotspots” and can be configured by creating a custom analysis type
• Overheads are higher than Hotspots analysis type
### Analysis Types

#### Lock and Waits

![Table Diagram]

<table>
<thead>
<tr>
<th>Sync Object</th>
<th>Wait Time</th>
<th>Wait Count</th>
<th>Spin Time</th>
<th>Module</th>
<th>Object Type</th>
<th>Object Creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Objects</td>
<td>112.517s</td>
<td>5,325</td>
<td>0s</td>
<td>[Unknown]</td>
<td>Constant</td>
<td>[Unknown]</td>
</tr>
<tr>
<td>Manual Reset Event 0xae37</td>
<td>109.238s</td>
<td>41</td>
<td>0s</td>
<td>[Unknown]</td>
<td>Manual Reset Event</td>
<td>dllStopPlugin</td>
</tr>
<tr>
<td>Manual Reset Event 0xfa6d</td>
<td>74.068s</td>
<td>26</td>
<td>0s</td>
<td>[Unknown]</td>
<td>Manual Reset Event</td>
<td>LdrGetProcedureAddressEx</td>
</tr>
<tr>
<td>Thread Pool</td>
<td>57.628s</td>
<td>235</td>
<td>0s</td>
<td>[Unknown]</td>
<td>Constant</td>
<td>[Unknown]</td>
</tr>
<tr>
<td>Sleep</td>
<td>57.371s</td>
<td>5,234</td>
<td>0.193s</td>
<td>[Unknown]</td>
<td>Constant</td>
<td>[Unknown]</td>
</tr>
<tr>
<td>Unknown 0x991c9877</td>
<td>56.974s</td>
<td>6,337</td>
<td>0s</td>
<td>[Unknown]</td>
<td>Unknown</td>
<td>LdrGetProcedureAddressEx</td>
</tr>
<tr>
<td>TBB Scheduler</td>
<td>41.457s</td>
<td>2,200</td>
<td>11.301s</td>
<td>[Unknown]</td>
<td>Constant</td>
<td>TaskManagerTBB::Init</td>
</tr>
<tr>
<td>[Unknown]</td>
<td>17.061s</td>
<td>865</td>
<td>0s</td>
<td>[Unknown]</td>
<td>[Unknown]</td>
<td>[Unknown]</td>
</tr>
<tr>
<td>Stream ../../media/graphics/</td>
<td>0.457s</td>
<td>183</td>
<td>0.057s</td>
<td>[Unknown]</td>
<td>Stream</td>
<td>Ogre::FileSystemArchive::open</td>
</tr>
<tr>
<td>Stream ../../media/sounds/h</td>
<td>0.440s</td>
<td>171</td>
<td>0.063s</td>
<td>[Unknown]</td>
<td>Stream</td>
<td>Framework::GDFParser::EndEler</td>
</tr>
<tr>
<td>Stream Ogre.log 0x501382c</td>
<td>0.397s</td>
<td>193</td>
<td>0.059s</td>
<td>[Unknown]</td>
<td>Stream</td>
<td>Ogre::Log::Log</td>
</tr>
<tr>
<td>Stream Smoke.gdf 0xF2b92</td>
<td>0.386s</td>
<td>11</td>
<td>0.006s</td>
<td>[Unknown]</td>
<td>Stream</td>
<td>PlatformManager::FileSystem::S</td>
</tr>
<tr>
<td>Stream ../../media/sounds/M</td>
<td>0.306s</td>
<td>119</td>
<td>0.037s</td>
<td>[Unknown]</td>
<td>Stream</td>
<td>Framework::GDFParser::EndEler</td>
</tr>
<tr>
<td>Stream ../../media/physics/D</td>
<td>0.247s</td>
<td>5</td>
<td>0.011s</td>
<td>[Unknown]</td>
<td>Stream</td>
<td>hkStdioStreamReader::hkStdioS</td>
</tr>
<tr>
<td>Stream ../../media/graphics/</td>
<td>0.136s</td>
<td>41</td>
<td>0.022s</td>
<td>[Unknown]</td>
<td>Stream</td>
<td>Ogre::FileSystemArchive::open</td>
</tr>
<tr>
<td>Stream ../../media/sounds/h</td>
<td>0.134s</td>
<td>13</td>
<td>0.018s</td>
<td>[Unknown]</td>
<td>Stream</td>
<td>TaskManagerTBB::ParallelFor</td>
</tr>
</tbody>
</table>
Analysis Types

*Event based sampling*

- Captures program performance information in terms of hardware specific (PMU) events.
- Uses event multiplexing to collect as much information possible during a single run
- Two primary modes:
  - **Event Based Sampling (EBS)**
    Captures the CPU state periodically, when a configured hardware event counter exceeds configured Sample-after Value (SAV)
  - **Event Based Counting (EBC)**
    Counts the configured events and writes the counts to the trace file
    - EBC doesn't capture CPU state information
    - EBC data cannot be attributed to the program flow
    - EBC mode has lower overhead and collects smaller trace files (tb5 / tb6)
Analysis Types

*Event based sampling*

- Pre-defined event sets for supported processors
  - Cache analysis and false sharing
  - Branching issues
  - Memory access
  - Memory bandwidth

- Event multiplexing
- Pre-defined displays (viewpoints) transform data into information
- Cycle accounting analysis methodology
Analysis Types
Event based sampling

Core 2 family
Pre-defined event groups

Core i7 family
Pre-defined event groups
Analysis Types
Event based sampling

![Cycles and uOps - Where are hardware issues affecting my application’s performance?](image)

**Events and Ratios**

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU_CLK_UNHALTED\ CORE</th>
<th>INST_RETIRED\ ANY</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>FireObjects:checkCollision</td>
<td>21,736,000,000</td>
<td>13,986,000,000</td>
<td>1.567 systemproceduralfire.dll</td>
</tr>
<tr>
<td>FireObjects:ProcessFireCollisionsRange</td>
<td>10,770,000,000</td>
<td>20,338,000,000</td>
<td>0.530 systemproceduralfire.dll</td>
</tr>
<tr>
<td>D3DXSHProjectCubeMop</td>
<td>5,320,000,000</td>
<td>7,190,000,000</td>
<td>0.768 d3d8_42.dll</td>
</tr>
<tr>
<td>D3DDisassembleLODEffect</td>
<td>4,326,000,000</td>
<td>5,828,000,000</td>
<td>0.742 d3dcompiler_42.dll</td>
</tr>
<tr>
<td>modf</td>
<td>4,076,000,000</td>
<td>4,202,000,000</td>
<td>0.970 msvc90.dll</td>
</tr>
<tr>
<td>Ogre:TextureUnitStates:Texture compressed</td>
<td>4,038,000,000</td>
<td>6,454,000,000</td>
<td>0.626 ogresmipperl.dll</td>
</tr>
<tr>
<td>tbs:internal:task_scheduler_observery:observe</td>
<td>3,890,000,000</td>
<td>1,406,000,000</td>
<td>2.767 tbb.dll</td>
</tr>
<tr>
<td>std::vector::class FireObject, class std::allocator&lt;class FireObject&gt; &gt; &amp;::set</td>
<td>3,844,000,000</td>
<td>6,984,000,000</td>
<td>0.556 systemproceduralfire.dll</td>
</tr>
<tr>
<td>RtUIntUnicodeString</td>
<td>3,206,000,000</td>
<td>3,778,000,000</td>
<td>0.849 ntdll.dll</td>
</tr>
<tr>
<td>[fmodex.dll]</td>
<td>2,650,000,000</td>
<td>2,516,000,000</td>
<td>1.014 fmodex.dll</td>
</tr>
<tr>
<td>Ldr!UnlockLoaderLock</td>
<td>2,566,000,000</td>
<td>2,904,000,000</td>
<td>0.884 ntdll.dll</td>
</tr>
</tbody>
</table>

Selected 1 row(s):

<table>
<thead>
<tr>
<th>Thread (0x1b54)</th>
<th>PMU Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread (0x5290)</td>
<td>PMU Events</td>
</tr>
<tr>
<td>Thread (0x4e68)</td>
<td>PMU Events</td>
</tr>
<tr>
<td>Thread (0x5314)</td>
<td>PMU Events</td>
</tr>
<tr>
<td>Thread (0x2528)</td>
<td>PMU Events</td>
</tr>
<tr>
<td>Thread (0x5188)</td>
<td>PMU Events</td>
</tr>
</tbody>
</table>

**Ruler Area**

- Frame
- Thread
- Running
- PMU Sampling
- PMU Events
- PMU Sampling
- Frames over Time
- Frame Rate

**No filters are applied**

**Module:** [All]  **Thread:** [All]  **Process:** [All]  **Timeline PMU Event:** UOPS_REFERD_FUSEL
Analysis Types

Custom Analysis

• New analysis types may be created or customized with your favorite PMU events to match your workflow.

• There are two ways to create a custom analysis type
  – Inherit from an already existing analysis type
  – Create a new one from either TPSS or SEP based collectors
Creating a New Custom Analysis Type

- Select the “New Analysis” button

- In the Amplifier XE Result tab, the bottom has two buttons
  - Clicking on “New” will give you the option of creating a new EBS or TPSS based analysis type
Creating a New Custom Analysis Type

- Selecting Stack Sampling based collector will open this “in-place” dialog for customizing and renaming.
- Rename the analysis type to something that is meaningful and comments on what it is useful for.
- Adjust the knobs to create your custom stack sampling analysis type.
- Click on “ok” after all the changes are made.
Command Line Interface

- amplxe-cl is the command line.
  - Windows: C:\Program Files\Intel\Inspector XE
    \bin[32|64]\amplxe-cl.exe
  - Linux: /opt/intel/inspector_xe/bin[32|64]/amplxe-cl

- To get detailed help:
  - amplxe-cl -help

- Get Command Line from GUI
  - Command examples:
    1. amplxe-cl -collect-list
    2. amplxe-cl -knob-list=hotspots
    3. amplxe-cl -collect=hotspot - myapp.exe [MyParams]
    4. amplxe-cl -report hotspots

More Help is available with the Online Documentation
Remote Data Collection
Conveniently analyze data collected on remote systems

1. Setup the experiment using GUI locally
2. Copy command line instructions to paste buffer
3. Open remote shell on target machine
4. Paste command line, run collection
5. Copy result file to your local system
6. Open file using local GUI

- Minimal “performance footprint” during collection
- Easy setup using GUI
- Easy analysis of results
Compare Results Quickly - Sort By Difference

- Quickly identify cause of regressions.
  - Run a command line analysis daily
  - Identify the function responsible so you know who to alert
- Compare 2 optimizations – What improved?
- Compare 2 systems – What didn’t speed up as much?
Readying Your Application for Intel VTune Amplifier XE

• You should run Amplifier XE on a “Released/Optimized” build.
• Symbols Allow you to view the Source (Not Just the assembly)
  - Windows: /Zi
  - Linux: -g
• Intel Threading Runtimes need instrumented runtimes
  - TBB: Define TBB_USE_THREADING_TOOLS
  - OpenMP: Use Intel Dynamic Version of OpenMP
• Call Stack Mode – Requires use of the dynamic version of the C Runtime library to properly attribute System Calls
  - Windows use:/MD(d)
  - Linux do not use: -static
Readying Your Application

- Better Benchmarks for Intel® VTune Amplifier XE are:
  - Typical
  - Repeatable
  - Timely? ->

![Advanced settings for VTune Amplifier XE](image)
Agenda

- Introduction to Parallel Studio XE
- About PMUs, EBS, counting
- VTune Amplifier XE: User Interface
- VTune Amplifier XE: Analysis Types
- Get your hands dirty (lab exercises)
Get your hands dirty...

- Please unpack amplifier-labs.tar.gz with:
  
tar xzf amplifier-labs.tar.gz

- Have look at the workbook for instructions

- Labs:
  - Lab 1: Lightweight Hotspots
  - Lab 2: Determining Efficiency
  - Lab 3.1: Finding Cache Misses

- Feel free to also try the analysis on your code!