Intel® VTune™ Amplifier XE 2013
Intel® VTune™ Amplifier XE 2013
Second Generation VTune™ Analyzer

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, Java*
- 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

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

First create a project

Welcome to Intel VTune Amplifier XE 2013

Recent Projects:
- Test
- Tachyon2013
- Java Mandelbroi
- Office 13
- Projects

Recent Results:
- r090hs
- r002hs
- r010hs
- r011hs
- r012hs

To start an analysis, click the toolbar button New Analysis.
Specify optional app to launch

Indicate if you want to start an app

- Automatically resume collection after (sec):
- Automatically stop collection after (sec):
- Store result in the project directory: C:\Users\gcarlet\Documents\Amplifier XE\Projects\samp1
- Store result in (and create link file to) another directory
- Result location:
- Advanced
  - Duration time estimate: Between 1 and 15 minutes
  - Collection data limit, MB: 100
  - Slow frames threshold, frames/s: 40
  - Fast frames threshold, frames/s: 100
  - CPU mask:
Indicate type of profiling (ex: Lightweight Hotspots)

1. Click New analysis button
2. Select profiling type
3. Click "Start" to begin profiling
Demo: Hotspot Collector
Hotspots analysis

The image depicts a screenshot of Intel VTune Amplifier XE 2013, a tool used for performance analysis of software applications. The interface shows a table with columns for Function, CPU time, and Module. The table highlights various functions and their associated CPU usage and modules. The diagram also includes annotations for Function hotspot, Call stack, Thread timeline, and CPU time.
Hotspots analysis – Source View
Types of Performance Data Collection

Hardware Event-based Sampling Analysis

• Collection Types: Lightweight Hotspots, Advanced Processor Analysis
• System wide analysis
  – Kernel mode code, device drivers, OS, ...
• Measures cache misses, branch mispredictions, ...
• Uses the Performance Monitoring Unit of each Intel CPU Core.

User Mode Sampling and Tracing Analysis

• Collection Types: Hotspots, Concurrency, & Lock and Waits
  – Dynamically instruments binary
    – Minimal = Hotspots
    – More = Concurrency, & Locks and Waits
• 1 process only
• User mode SW only
• Uses OS Timer Service for each thread to collects a sample
Advanced Processor Analysis – General Exploration

• Predefined Analysis Type that collects different types of CPU performance events

• Good for first look at whether any CPU event categories are affecting performance

• GUI highlights those events and functions that have performance problems
Demo – General Exploration Collector
Running the General Exploration collector

1. Click “New Analysis” button

2. Select “General Exploration” for your CPU architecture

3. Click “Start” to begin profiling
CPU HW Sampling results

<table>
<thead>
<tr>
<th>/Function</th>
<th>PMU Event Count</th>
<th>CPI</th>
<th>Retire Stalls</th>
<th>LLC Miss</th>
<th>LLC Load</th>
<th>Conten Acces</th>
<th>Instru Stalls</th>
<th>Branch Mispr</th>
<th>Execu Stalls</th>
<th>Data Sharing</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize_2D_buffer</td>
<td>42,564,000,000</td>
<td>0.669</td>
<td>0.530</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>sphere_intersect</td>
<td>20,652,000,000</td>
<td>1.077</td>
<td>0.815</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>grid_intersect</td>
<td>11,816,000,000</td>
<td>1.461</td>
<td>0.847</td>
<td>0.015</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>grid_bounds_intersect</td>
<td>1,700,000,000</td>
<td>1.710</td>
<td>0.688</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GdipCreateSolidFill</td>
<td>528,000,000</td>
<td>0.610</td>
<td>0.295</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>shader</td>
<td>302,000,000</td>
<td>1.641</td>
<td>0.603</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Selected 1 row(s):</td>
<td>90,000,000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hovering the mouse over a highlighted problem displays a tooltip with a problem definition and high level suggestions for fixes or analysis next steps.

Performance problems are highlighted.
Intel® VTune™ Amplifier XE

Frame Analysis

Frame: a region executed repeatedly (non-overlapping).

API marks start and finish

Examples:

• Game – Compute next graphics frame
• Database – Query response time
• Computation – Convergence loop

Application

```c
void algorithm_1();
void algorithm_2(int myid);
double GetSeconds();
DWORD WINAPI do_xform (void * lpmyid);
bool checkResults();
__itt_frame =
    __itt_frame_createA("myDomain");

while( gRunning ) {
    __itt_frame_begin(itt_frame);
    //Do Work
    __itt_frame_end(itt_frame);
}  
```
Intel® VTune™ Amplifier XE
Find Slow Frames With One Click

(1) Regroup Data

Function - Call Stack
Module - Function - Call Stack
Source File - Function - Call Stack
Thread - Function - Call Stack
Function - Thread - Call Stack
OpenMP Region - Function - Call Stack
Task Type - Function - Call Stack
Frame Domain - Function - Call Stack
Frame Domain - Frame Type - Function - Call Stack

Result:

Table showing CPU Time and Frame Time for different categories of frames.

... (Partial list shown)
Locks and Waits Collection

Identifies those threading items that are causing the most thread block time

- Synchronization locks
- Threading APIs
- I/O
Demo: Locks-And-Waits Collector
Running the “Locks and Waits” collector

1. Click “New Analysis” button

2. Select “Locks and Waits”

3. Click “Start” to begin profiling
Locks-and-Waits View

![Screen capture of Locks-and-Waits View in Intel VTune Amplifier XE 2011 interface showing the Locks and Waits panel with detailed wait time and module information for a selected mutex and thread.](image-url)
Locks-and-Waits Source View

```
162   drawing_area drawing(startx, totaly-y, stopx
163   // Acquire mutex to protect pixel calculation
164   pthread_mutex_lock (&rgb_mutex);
165   for (int x = startx; x < stopx; x++) {
166     color_t c = render_one_pixel (x, y, local_
167     drawing.put_pixel(c);
169   }
```

```
Address          Assembly
0x323a           call 0x804ae00 <
0x323f           Block 5:
0x323f           add $0x14, %esp
0x3242           pushl $0x805d54
0x3247           call 0x8049b10 <
0x324c           Block 6:
0x324c           movl $0x805d574,
0x3252           add $0x10, %esp
```

Thread:
- Running
- Waits
- Transition
- CPU Time

CPU Usage

Thread Concurrency

No filters are applied.

Module: [All]   Threads: [All]   Call Stack Model: Only user functions
Java* Support

- Analyze CPU usage and HW events (cache misses, ...)
- Drill down to Java source code
- See how much time the Java Interpreter is consuming
- Mixed Mode Profiling: Java* and C/C++ code
Intel® VTune™ Amplifier XE 2013

Power profiling
• Profile sleep state and dynamic frequency speeds
• Identify causes of wake-ups
• See sleep state and frequency transitions in timeline view
• Determine best BIOS power settings
• Currently Linux* only

Intel® Xeon Phi™ Coprocessor support
• Hardware CPU event profiling (CPU usage, cache misses,...)
• Correlate data events across multiple cards
Intel® VTune™ Amplifier XE 2013

Analyze User Tasks via APIs
• Timeline is marked with start and stop times of your tasks
• Tasks can correspond to functions
• Supported by all collection modes
• Can be Nested

User Defined Performance Metrics
• Define new columns in sampling results displays
• Define formulas based on sampling counters
• Define tooltip text/descriptions for the new metric also
• Implemented using Python scripts
• See help under “user-defined metrics”
Summary

- The Intel® VTune Amplifier XE can be used to find:
  - Source code for performance bottlenecks
  - Characterize the amount of parallelism in an application
  - Determine which synchronization locks or APIs are limiting the parallelism in an application
  - Easily find CPU performance events that are causing additional CPU clocks
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 #20110228
Legal Disclaimer

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.

Performance tests and ratings are measured using specific computer systems and/or components
and reflect the approximate performance of Intel products as measured by those tests. Any
difference in system hardware or software design or configuration may affect actual performance.
Buyers should consult other sources of information to evaluate the performance of systems or
components they are considering purchasing. For more information on performance tests and on

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino
Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386,
Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside,
NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel
XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium
Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, VPro Inside, VTune, Xeon, and Xeon
Inside are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2013. Intel Corporation.

http://intel.com/software/products
AutoDetect DirectX* Frames

- Find occasional slow video frames
  - Identify causes of intermittently slow frames by comparing slow frame functions to fast frames
  - Definition of “slow” is user configurable

2.6% of DirectX frames were too slow

- Expand “Slow” and “Fast” nodes to see the differences and identify slow frame causes
“JIT” APIs

Profiling Runtime generated code

APIs to indicate attributes of code
• Code memory address
• Symbol information
  – Function names, Line Numbers

Drill down to source code when viewing profiling analysis

APIs are defined in jitprofiling.h