Revealing the performance aspects in your code

Intel® VTune™ Amplifier XE Generics

Rev.: Sep 1, 2013
Agenda

- Introduction to Intel® VTune™ Amplifier XE profiler
- High-level Features
- Types of Analysis
- Hotspot analysis
  - Statistical Call Graph
  - Lab 1: Find the Performance Hotspot
- Concurrency Analysis
  - Lab 2: Analyzing Parallelism
- Locks and Waits Analysis
  - Lab 3: Identifying Parallelism issues
- User and Synchronization API, Frame/Task Analysis
  - Lab 4: Instrumenting user source code
- Command Line Interface, Installation, Remote Collection
- Conclusion
**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

**Advanced Profiling For Scalable Multicore Performance**

Software & Services Group, Developer Products Division

Copyright © 2013, Intel Corporation. All rights reserved.

*Other brands and names are the property of their respective owners.
Fast, Accurate Performance Profiles
- Hotspot (Statistical call tree)
- Call counts (Statistical)
- Hardware-Event Sampling

Thread Profiling
- Visualize thread interactions on timeline
- Balance workloads

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

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

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

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.
A set of instruments to identify performance problems

Quick Overview
Intel® VTune™ Amplifier XE
Get a quick snapshot

Thread Concurrency Histogram
This histogram represents a breakdown of the Elapsed Time. It visualizes the percentage of the wall time the specific number of threads were considered running if they are either actually running on a CPU or are in the runnable state in the OS scheduler. Essentially, Thread Concurrency that were not waiting. Thread Concurrency may be higher than CPU usage if threads are in the runnable state and not consuming CPU time.

Elapsed Time
0s  1s  2s  3s  4s  5s  6s  7s  8s  9s  10s  11s  12s  13s  14s  15s

- Idle
- Poor
- Ok
- Ideal
- Over

Simultaneously Running Threads

4 cores
Intel® VTune™ Amplifier XE
Identify hotspots

Hottest Functions

Hottest Call Stack

Quickly identify what is important
Intel® VTune™ Amplifier XE
Identify threading inefficiency

- **Coarse Grain Locks**
- **High Lock Contention**
- **Load Imbalance**

Low Concurrency
Intel® VTune™ Amplifier XE
Find Answers Fast

Adjust Data Grouping:
- Function - Call Stack
- Module - Function - Call Stack
- Source File - Function - Call Stack
- Thread - Function - Call Stack

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]
Call Stack Mode: Only user functions
Intel® VTune™ Amplifier XE
Timeline Visualizes Thread Behavior

- Optional: Use API to mark frames and user tasks
- Optional: Add a mark during collection
**Intel® VTune™ Amplifier XE**  
See Profile Data On Source / Asm

<table>
<thead>
<tr>
<th>Line</th>
<th>Source</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>468</td>
<td>FireObject::checkCollision(V3 pos, V3 pre)</td>
<td>0.476s</td>
</tr>
<tr>
<td>472</td>
<td>#define FMax std::max&lt;float&gt;</td>
<td>0.561s</td>
</tr>
<tr>
<td>473</td>
<td>float param2 = (AABB::xMax - prevPos.x);</td>
<td>0.830s</td>
</tr>
<tr>
<td>477</td>
<td>bool neg = (rx &lt; 0.f);</td>
<td>0.615s</td>
</tr>
<tr>
<td>478</td>
<td>minP = FMax/neg? param2 : param1, mi</td>
<td>3.008s</td>
</tr>
<tr>
<td>479</td>
<td>maxP = FMin/neg? param1 : param2, ma</td>
<td>1.875s</td>
</tr>
<tr>
<td>481</td>
<td>if(maxP &gt; minP) {</td>
<td>0.972s</td>
</tr>
<tr>
<td>482</td>
<td>rx = 1.f/(pos.x - prevPos.x);</td>
<td>0.252s</td>
</tr>
<tr>
<td>483</td>
<td>param1 = (AABB::xMin - prevPos.x)</td>
<td>0.264s</td>
</tr>
<tr>
<td>484</td>
<td>param2 = (AABB::xMax - prevPos.x)</td>
<td>0.040s</td>
</tr>
<tr>
<td>486</td>
<td>prex = (rx &lt; 0.f);</td>
<td>0.047s</td>
</tr>
<tr>
<td>487</td>
<td>if(maxP &gt; minP) {</td>
<td>0.274s</td>
</tr>
<tr>
<td>488</td>
<td>param1 = (AABB::xMin - prevPos.x)</td>
<td>0.164s</td>
</tr>
<tr>
<td>489</td>
<td>if(maxP &gt; minP) {</td>
<td>0.612s</td>
</tr>
</tbody>
</table>

**Quick Asm navigation:**  
Select source to highlight Asm

**Quickly scroll to hot spots.**

**Time on Source / Asm**

**Right click for instruction reference manual**

**Click jump to scroll Asm**
High-level Features
Intel® VTune™ Amplifier XE
Feature Highlights

• Hot Spot Analysis (Statistical Call Graph)
  - Locates the time consuming regions of your application
  - Provides associated call-stacks that let you know how you got to these time consuming regions
  - Call-tree built using these call stacks

• Advanced Hotspot and architecture analysis
  - Based on Hardware Event-based Sampling (EBS)
  - Pre-defined tuning experiments

• Thread Profiling
  - Visualize thread activity and lock transitions in the timeline
  - Provides lock profiling capability
  - Shows CPU/Core utilization and concurrency information

• GPU Compute Performance Analysis
  - Collect GPU data for tuning OpenCL applications. Correlate GPU and CPU activities

• CPU Power Efficiency Analysis
  - Wake-up rate and frequency measurement per Core
Intel® VTune™ Amplifier XE
Feature Highlights

• **Attach to running processes**
  – Hotspot and Concurrency analysis modes can attach to running processes

• **System wide data collection**
  – EBS modes allows system wide data collection and the tool provides the ability to filter this data

• **GUI**
  – Standalone GUI available on Windows* and Linux
  – Microsoft* Visual Studio integration

• **Command Line**
  – Comprehensive support for regression analysis and remote collection

• **Platform & application support**
  – Windows* and Linux
  – Microsoft*.NET/C# applications
  – Java* and mixed applications
  – Fortran applications
Intel® VTune™ Amplifier XE
Feature Highlights

• **Event multiplexing**
  - Gather more information with each profiling run

• **Timeline correlation of thread and event data**
  - Populates thread active time with event data collected for that thread
  - Ability to filter regions on the timeline

• **Advanced Source / Assembler View**
  - See event data graphed on the source / assembler
  - View and analyze assembly as basic blocks
  - Review the quality of vectorization in the assembly code display of your hot spot

• **Provides pre-defined tuning experiments**
  - Predefined profiles for quick analysis configuration
  - A user profile can be created on a basis of a predefined profile

• **User API**
  - Rich set of user API for collection control, events highlighting, code instrumentation, and visualization enhancing.
Data Collectors and Analysis Types
## Intel® VTune™ Amplifier XE

### Analysis Types (based on technology)

<table>
<thead>
<tr>
<th>Software Collector</th>
<th>Hardware Collector</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software Collector</strong>&lt;br&gt;Any IA86 processor, any virtual, no driver</td>
<td><strong>Hardware Collector</strong>&lt;br&gt;Higher res., lower overhead, system wide</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Basic Hotspots</th>
<th>Advanced Hotspots</th>
</tr>
</thead>
<tbody>
<tr>
<td>Which functions use the most time?</td>
<td>Which functions use the most time?&lt;br&gt;Where to inline? – Statistical call counts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concurrency</th>
<th>General Exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune parallelism.&lt;br&gt;Colors show number of cores used.</td>
<td>Where is the biggest opportunity?&lt;br&gt;Cache misses? Branch mispredictions?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Locks and Waits</th>
<th>Advanced Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune the #1 cause of slow threaded performance – waiting with idle cores.</td>
<td>Dig deep to tune bandwidth, cache misses, access contention, etc.</td>
</tr>
</tbody>
</table>
Intel® VTune™ Amplifier XE
Pre-defined Analysis Types

Advanced Hotspot analysis based on the underlying architecture

User mode sampling, Threading, IO, Signaling API instrumentation

3rd Generation Core Architecture (a.k.a SandyBridge) analysis types

4th Generation Core Architecture (a.k.a Haswell) analysis types
Creating a Project

GUI Layout

1. Open the Intel VTune Amplifier and select "New Project..." using the keyboard shortcut CTRL+SHIFT+N.
2. In the "Create a Project" dialog box, enter the project name "tachyon1" and select the location "C:\Intel\TechInfo\SS\IPS_2013_XE_Course\15_Amplifier_Generics\Labs_".
3. In the "Project Properties" dialog box, set the "Launch Application" target to "Launch Application" and specify the application path as previously selected.

*Other brands and names are the property of their respective owners.
Selecting type of data collection

GUI Layout

All available analysis types

Different ways to start the analysis

Helps creating new analysis types

Copy the commandline to clipboard
Profile a Running Application
No need to stop and relaunch the app when profiling

Two Techniques:

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

- **Profile System:**
  - Advanced Hotspots & Custom EBS
  - Optional: Filter by process after collection
Clicking on the Summary tab shows a high level summary of the run.

Timing for the whole application run.

List of 5 Hotspot functions.

CPU Usage.
Bottom-Up View
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
Clicking on the Top-Down Tree tab changes stack representation in the Grid.

Top-level function and its tree

Total Time (self + children’s)

Self Time
Caller/Callee View

GUI Layout

Select a function in the Bottom-Up and find the caller/callee

List of functions sorted by CPU Time

List of callers and their stacks

List of callees and their stacks
Adding User Marks to the Timeline

GUI Controls

- Start application without data collection
- Resume data collection when needed
- Observe paused region on the Time Line
- Click “Mark Timeline” during collection
- Observe the mark on the Time Line
Key Result Analysis and GUI Concepts
• **Viewpoints**
  - It is a pre-defined view that determines what needs to be displayed in the grid and timeline for a given analysis type.
  - An analysis type may support more than one viewpoints.
  - To change viewpoints, select a viewpoint by clicking on.
**Groupings**

- Each analysis type has many viewpoints
- Each viewpoint has pre-defined groupings
- Allows you to analyze the data in different hierarchies and granularities
Viewpoints and Groupings

- For example, pre-defined groupings can be used to determine load imbalance.
VTune™ Amplifier XE allows comparison of two similar runs

Extremely useful for

- Benchmarking
- Regression analysis
- Testing

During performance optimization work source code may change

- Binary recompiled: compare based on source function
- Inside a function: compare based on functions level
- Functions changed: group by source files and compare
- Source files changed: compare by modules
### Results Comparison

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU Time: Difference by Utilization</th>
<th>CPU Time: r001cc</th>
<th>CPU Time: r003cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>sphere_intersect</td>
<td>-0.617s</td>
<td>5.688s</td>
<td>6.304s</td>
</tr>
<tr>
<td>grid_intersect</td>
<td>-1.212s</td>
<td>4.193s</td>
<td>5.406s</td>
</tr>
<tr>
<td>draw_task::operator()</td>
<td>-0.088s</td>
<td>2.229s</td>
<td>2.318s</td>
</tr>
<tr>
<td>parallel_thread</td>
<td>0.904s</td>
<td>1.458s</td>
<td>0.554s</td>
</tr>
<tr>
<td>grid_bounds_intersect</td>
<td>0.129s</td>
<td>0.385s</td>
<td>0.256s</td>
</tr>
<tr>
<td>Gdiplus::Graphics::DrawImage</td>
<td>-2.328s</td>
<td>0.352s</td>
<td>2.680s</td>
</tr>
</tbody>
</table>
Lab Activities
Analysis Types Revisited
Reminding methodology of performance profiling and tuning

The Goal: minimize the time it takes your program / module / function to execute

- Identify Hotspots and focus on them
- It’s just a few functions (20% of code does 80% of job)
- Optimize them (with compiler or hand optimizations)
- Check for hotspots again, and find new ones

How to optimize the Hotspots?

- Maximize CPU utilization and minimize elapsed time
- Ensure CPU is busy all the time
- All Cores busy – parallelism
- Busy with useful tasks
- Optimize tasks execution
Intel® VTune™ Amplifier XE

Terminology

• **Elapsed Time**
  The total time your target application ran. Wall clock time at end of application – Wall clock time at start of application

• **CPU Time**
  The amount of time a thread spends executing on a logical processor. For multiple threads, the CPU time of the threads is summed.

• **Wait Time**
  The amount of time that a given thread waited for some event to occur, such as: synchronization waits and I/O waits
**Elapsed Time**: 6 seconds

**CPU Time**: T1 (4s) + T2 (2s) + T3 (2s) = 8 seconds

**Wait Time**: T1(2s) + T2(3s) + T3 (2s) = 7 seconds
Hotspot analysis

- Displays hot functions in your application
- Shows most time consuming call sequences
  - Statistical Call Graph
- Include timeline view of threads in your application
Hotspot analysis
Summary

Note Elapsed Time and CPU Time
Hotspot analysis
Summary (Continued)

**CPU Usage Histogram**
This histogram represents a breakdown of the Elapsed Time. It visualizes what percentage of the wall time the specific number of CPUs were higher than the thread concurrency if a thread is executing code on a CPU while it is logically waiting.

- Elapsed Time
- Average
- Poor
- Ok
- Ideal
- Over

Simultaneously Utilized Logical CPUs

**Collection and Platform Info**
This section provides information about this collection, including result set size and collection platform data.

- Application Command Line: C:\Intel\VTech Info\SS1PS_2013_XE_Course15_Amplifier_Generics\Labs_Windows\VTune Amplifier XE\ta
- Operating System: Microsoft Windows 7
- Computer Name: VTSYMBAL-MOBL3.ccr.corp.intel.com
- Result Size: 1 MB

**CPU**
- Name: Intel(R) Core(TM) Processor 2xxx Series
- Frequency: 2.5 GHz
- Logical CPU Count: 4

Note overall CPU Usage
Note # of CPUs Available on the platform
Hotspots analysis
Hotspot functions

Adjust Data Grouping

Call stack

Function CPU time

Filter by Module & Other Controls

Filter by Timeline Selection (or by Grid Selection)

Thread timeline

Click [+] for Call Stack

Hotspot Functions

Change Viewpoint

Filter by Timeline Selection (or by Grid Selection)
Hotspots analysis
Hotspot functions by CPU usage

Double Click Function to View Source

Coloring CPU Time by CPU Utilization

Overhead and Spin Time

Overhead and Spin on Timeline
Hotspots analysis
Source View

Source View

Assembly View

Self and Total Time on Source / Asm

Right click for instruction reference manual

Quick Asm navigation:
Select source to highlight Asm

Click jump to scroll Asm

Quickly scroll to hot spots.
Scroll Bar “Heat Map” is an overview of hot spots.
Lab 1

• Find the Performance Hotspot
Reminding methodology of performance profiling and tuning

How to optimize the Hotspots?

- Maximize CPU utilization and minimize elapsed time
  - Ensure CPU is busy all the time
  - All Cores busy – parallelism (high concurrency)
Concurrency -

Is a measurement of the number of active threads
Intel® VTune™ Amplifier XE
Parallelism/Concurrency Analysis

- For Parallelism / Concurrency analysis,
  - Stack sampling is done just like in Hotspots analysis
  - Wait functions are instrumented (e.g. WaitForSingleObject, EnterCriticalSection)
  - Signal functions are instrumented (e.g. SetEvent, LeaveCriticalSection)
  - I/O functions are instrumented (e.g. ReadFile, socket)

Concurrency Analysis

Start the Analysis
Concurrency Analysis

Summary

Concurrency Levels

Adjustable Metrics

Thread Concurrency Histogram
This histogram represents a breakdown of the Elapsed Time. It visualizes the percentage of the wall time the specific number of threads were running simultaneously. Threads are considered running if they are either actually running or in the runnable state in the OS scheduler. Essentially, Thread Concurrency is a measurement of the number of threads that were not waiting. Thread Concurrency may vary depending on the number of threads that are in the runnable state and not consuming CPU time.

Elapsed Time
0s  1s  2s  3s  4s  5s

Simultaneously Running Threads
Idle Poor Ok Ideal

Concurrent Analysis
Concurrency Analysis
Summary: Concurrency vs. CPU Usage Histogram

Threads might be in active state, but not using CPU
Concurrency View

Concurrent Level

Overhead

Wait

Thread is running

Thread waiting

Thread Transitions

Overhead

Software & Services Group, Developer Products Division

Copyright © 2013, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Concurrency Timeline
Investigate reasons for transitions

Select and Zoom
Hover over a transition line
Source Code View by Concurrency

Concurrency coloring for CPU Time against source lines
Lab 2

• Analyzing Parallelism
Waiting on locks

Calculating Wait and Idle time
Intel® VTune™ Amplifier XE
Locks and Waits Analysis

- Identifies those threading items that are causing the most thread block time
  - Synchronization locks
  - Threading APIs
  - I/O

Start the Analysis

Details
- CPU sampling interval, ms: 10
- Collect highly accurate CPU time: Yes
- Collect CPU sampling data: With stacks
- Collect signalling API data: With stacks
- Collect synchronization API data: With stacks
- Collect I/O API data: With stacks
Locks and Waits View

- **Grouping by Sync Object**
- **Wait Objects**
- **CPU Utilization**
- **Waits #**
- **Spinning**
- **Stack for the wait object**

Intel VTune Amplifier XE 2013

Software & Services Group, Developer Products Division

Copyright © 2013, Intel Corporation. All rights reserved.

*Other brands and names are the property of their respective owners.*
Locks-and-Waits Source View

Critical Section object
Waiting time on the Critical Section
Wait count
Lab 3

• Finding Parallelism Issues
User APIs

- Collection Control API
- Thread Naming API
- User-Defined Synchronization API
- Task API
- User Event API
- Frame API
- JIT Profiling API
Enable you to

- control collection
- set marks during the execution of the specific code
- specify custom synchronization primitives implemented without standard system APIs

To use the user APIs, do the following:

- Include `ittnotify.h`, located at `<install_dir>/include`
- Insert `__itt_*` notifications in your code
- Link to the `libittnotify.lib` file located at `<install_dir>/lib`
Intel® VTune™ Amplifier XE
User APIs

Collection Control APIs

void __itt_pause (void)
Run the application without collecting data. VTune™ Amplifier XE reduces the overhead of collection, by collecting only critical information, such as thread and process creation.

void __itt_resume (void)
Resume data collection. VTune™ Amplifier XE resumes collecting all data.

Thread naming APIs

void __itt_thread_set_name (const __itt_char *name)
Set thread name using char or Unicode string, where name is the thread name.

void __itt_thread_ignore (void)
Indicate that this thread should be ignored from analysis. It will not affect the concurrency of the application. It will not be visible in the Timeline pane.
int main(int argc, char* argv[]) {  
    doSomeInitializationWork();

    __itt_resume();
    while(gRunning) {
        doSomeDataParallelWork();

    }

    __itt_pause();

    doSomeFinalizationWork();
    return 0;
}  

long spin = 1;

. . . .

. . . .

__itt_sync_prepare((void *) &spin );
while(ResourceBusy);

// spin wait;
__itt_sync_acquired((void *) &spin );

// Use shared resource
__itt_sync_releasing((void *) &spin );

// Code here should free the resource
Intel® VTune™ Amplifier XE
User Event APIs

• Useful to observe when certain events occur in your application or identify how long certain regions of code take to execute

• Event APIs enables you to annotate an application when certain events occur

```c
__itt_event __itt_event_create(char *, int);
__itt_event_start(__itt_event);
__itt_event_end(__itt_event);
```
Create a user event type with the specified name. This API returns a handle to the user event type that should be passed into the following APIs as a parameter. The namelen parameter refers to the number of characters, not the number of bytes.

Call this API with an already created user event handle to register an instance of that event. This event appears in the Timeline pane display as a tick mark.

Call this API following a call to `__itt_event_start()` to show the user event as a tick mark with a duration line from start to end. If this API is not called, the user event appears in the Timeline pane as a single tick mark.
DWORD WINAPI aiWork(LPVOID lpArg)
{
    int tid = *((int*)lpArg);
    __itt_event aiEvent;
    aiEvent = __itt_event_create("AI Thread Work",14);

    while(gRunning) {
        WaitForSingleObject(bSignal[tid], INFINITE);
        __itt_event_start(aiEvent);
        doSomeDataParallelWork();
        __itt_event_end(aiEvent);
        SetEvent(eSignal[tid]);
    }
    return 0;
}
Intel® VTune™ Amplifier XE
Visualizing Events in the Timeline View

User defined task

User Tasks
Start: 29.651ms Duration: 231.248us
Task Type: F2

User Tasks
Start: 29.882ms Duration: 217.747us
Task Type: F2

User Tasks
Start: 30.101ms Duration: 218.356us
Task Type: F2

User Tasks
Start: 30.319ms Duration: 223.247us
Task Type: F2

4 out of 8 User Tasks Item(s) shown
Frame Analysis – Analyze Long Latency Activity

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

- API marks start and finish
- Auto detect DirectX frames

Examples:
- Game – Compute next graphics frame
- Simulator – Time step loop
- Computation – Convergence loop
__itt_domain*
__itt_domain_create( const
__itt_char *name );

Create a domain with a domain name.
Since the domain is expected to be static over the
application's execution time, there is no mechanism to
destroy a domain. Any domain can be accessed by any
thread in the process, regardless of which thread created the
domain. This call is thread-safe.

void
__itt_frame_begin_v3(const
__itt_domain *domain, __itt_id
*id);

Define the beginning of the frame instance.
A __itt_frame_begin_v3 call must be paired with a
__itt_frame_end_v3 call. Successive calls to __itt_frame_begin_v3 with the same ID
are ignored until a call to __itt_frame_end_v3 with the same
ID.
• domain is the domain for this frame instance.
• id is the instance ID for this frame instance, or NULL.

void __itt_frame_end_v3(const
__itt_domain *domain, __itt_id
*id);

Define the end of the frame instance.
A __itt_frame_end_v3 call must be paired with a
__itt_frame_begin_v3 call. The first call to
__itt_frame_end_v3 with a given ID ends the frame.
Successive calls with the same ID are ignored, as are calls
that do not have a matching __itt_frame_begin_v3 call.
• domain - The domain for this frame instance
• id - The instance ID for this frame instance, or NULL for the
current instance.
Frame Analysis
Using APIs

__itt_domain* pD = __itt_domain_create ("SimDomain");

while(gRunning) {
    __itt_frame_begin_v3(pD, NULL);

    start = clock();
    //Wait all threads before moving into the next frame
    WaitForMultipleObjects(FUNCTIONAL_DOMAINS, eSignal, TRUE, INFINITE);
    stop = clock();
    //Give all threads the "go" signal
    for (int i = 0; i < FUNCTIONAL_DOMAINS; i++)
        SetEvent(bSignal[i]);
    if (frame % NETWORKCONNECTION_FREQ == 0) {
        //Start network thread
        SetEvent(bNetSignal);
    }
    __itt_frame_end_v3(pD, NULL);
}
Frame Rate Chart

This histogram shows the total number of frames in your application executed with a specific frame rate. High number of slow or fast frames signals a performance bottleneck. Explore the data provided in the Bottom-up, Top-down Tree, and Timeline panes to identify code regions with the high/slow frame rate. Try to optimize your code to keep the frame rate constant (for example, from 30 to 60 frames per second).

Adjust the frame rate then Apply changes
Frame Analysis
Find Slow Frames With One Click

(1) Regroup Data

<table>
<thead>
<tr>
<th>Grouping</th>
<th>Function - Call Stack</th>
<th>Module - Function - Call Stack</th>
<th>Source File - Function - Call Stack</th>
<th>Thread - Function - Call Stack</th>
<th>Function - Thread - Call Stack</th>
<th>OpenMP Region - Function - Call Stack</th>
<th>Task Type - Function - Call Stack</th>
<th>Frame Domain - Frame - Function - Call Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Function - Call Stack</td>
<td>Module - Function - Call Stack</td>
<td>Source File - Function - Call Stack</td>
<td>Thread - Function - Call Stack</td>
<td>Function - Thread - Call Stack</td>
<td>OpenMP Region - Function - Call Stack</td>
<td>Task Type - Function - Call Stack</td>
<td>Frame Domain - Frame - Function - Call Stack</td>
</tr>
<tr>
<td></td>
<td>Function - Call Stack</td>
<td>Module - Function - Call Stack</td>
<td>Source File - Function - Call Stack</td>
<td>Thread - Function - Call Stack</td>
<td>Function - Thread - Call Stack</td>
<td>OpenMP Region - Function - Call Stack</td>
<td>Task Type - Function - Call Stack</td>
<td>Frame Domain - Frame - Function - Call Stack</td>
</tr>
<tr>
<td></td>
<td>Function - Call Stack</td>
<td>Module - Function - Call Stack</td>
<td>Source File - Function - Call Stack</td>
<td>Thread - Function - Call Stack</td>
<td>Function - Thread - Call Stack</td>
<td>OpenMP Region - Function - Call Stack</td>
<td>Task Type - Function - Call Stack</td>
<td>Frame Domain - Frame - Function - Call Stack</td>
</tr>
<tr>
<td></td>
<td>Function - Call Stack</td>
<td>Module - Function - Call Stack</td>
<td>Source File - Function - Call Stack</td>
<td>Thread - Function - Call Stack</td>
<td>Function - Thread - Call Stack</td>
<td>OpenMP Region - Function - Call Stack</td>
<td>Task Type - Function - Call Stack</td>
<td>Frame Domain - Frame - Function - Call Stack</td>
</tr>
<tr>
<td></td>
<td>Function - Call Stack</td>
<td>Module - Function - Call Stack</td>
<td>Source File - Function - Call Stack</td>
<td>Thread - Function - Call Stack</td>
<td>Function - Thread - Call Stack</td>
<td>OpenMP Region - Function - Call Stack</td>
<td>Task Type - Function - Call Stack</td>
<td>Frame Domain - Frame - Function - Call Stack</td>
</tr>
</tbody>
</table>

... (Partial list shown)

Before: List of Functions Taking Time

After: List of Slow Frames
Just 2 more clicks shows where to focus tuning...

Slow functions in slow frames

(1) Only show slow frames

Result:
Functions taking a lot of time in slow frames

(2) Regroup: Show functions
A task is a logical unit of work performed by a particular thread.

Tasks can be nested.

You can use task APIs to assign tasks to threads.

One thread executes one task at a given time.

Tasks may correspond to functions, scopes, or a case block in a switch statement.
### Task API primitives

<table>
<thead>
<tr>
<th>Use This Primitive</th>
<th>To Do This</th>
</tr>
</thead>
<tbody>
<tr>
<td>void ITTAPI_itt_task_begin (const _itt_domain *domain, _itt_id taskid, _itt_id parentid, _itt_string_handle *name)</td>
<td>Create a task instance on a thread. This becomes the current task instance for that thread. A call to _itt_task_end() on the same thread ends the current task instance.</td>
</tr>
<tr>
<td>void ITTAPI_itt_task_begin_fn (const _itt_domain *domain, _itt_id taskid, _itt_id parentid, void *fn)</td>
<td>Begin a task instance on a thread.</td>
</tr>
<tr>
<td>void ITTAPI_itt_task_end (const _itt_domain *domain)</td>
<td>End a task instance on a thread.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_itt_domain</td>
<td>The domain of the task.</td>
</tr>
<tr>
<td>_itt_id taskid</td>
<td>This is a reserved parameter.</td>
</tr>
<tr>
<td>_itt_id parentid</td>
<td>This is a reserved parameter.</td>
</tr>
<tr>
<td>_itt_string_handle</td>
<td>The task string handle.</td>
</tr>
<tr>
<td>*fn</td>
<td>This is a reserved parameter.</td>
</tr>
</tbody>
</table>
Task API usage

__itt_domain* domain = __itt_domain_create(L"Task Domain");
__itt_string_handle* UserTask = __itt_string_handle_create(L"UserTask");
__itt_string_handle* UserSubTask = __itt_string_handle_create(L"UserSubTask");

int main(int argc, char* argv[]) {
    ...
    __itt_task_begin (domain, __itt_null, __itt_null, UserTask);
    //create many threads to call work()
    __itt_task_end (domain);
    ...
}

work() {
    __itt_task_begin (domain, __itt_null, __itt_null, UserSubTask);
    do_foo();
    __itt_task_end (domain);
    return 0;
}
Using Task API
Hotspots analysis – Bottom-up pane
Using Task API
Hotspots analysis – Tasks pane
Lab 4

• Instrumenting user source code
Windows & Linux Versions Available
Stand-alone GUI, Command line, Visual Studio Integration

Microsoft Windows* OS
- Windows XP*, Windows 7*, Windows 8 Desktop*
- Standalone GUI and command line
- IA32 and Intel® 64

Linux* OS
- RHEL*, Fedora*, SUSE*, CentOS*, Ubuntu*
- Additional distributions may also work
- Standalone GUI and command line
- IA32 and Intel® 64

Single user and floating licenses available

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

• Windows
  – Integrated into Microsoft Visual Studio, or Standalone
  – Administrative privileges required for full package
  – GUI and command line versions are both installed

• Linux
  – Standalone GUI and command line versions
  – Root access not required but won’t install “Event-based sampling” collectors
    – Hotspot collector with calling sequences will work
  – Data collection-only installation option
    – Enables collection with no license
    – Collection results then copied to system with license for viewing
  – Driver for event-based sampling is built at install time and can be “insmod’d” at install time and boot time
• Command line (CLI) versions exist on Linux* and Windows*
  – **CLI use cases:**
    – Test code changes for performance regressions
    – Automate execution of performance analyses
  – **CLI features:**
    – Fine-grained control of all analysis types and options
    – Text-based analysis reports
    – Analysis results can be opened in the graphical user interface
Intel® VTune™ Amplifier XE
Command Line Interface - Examples

• Display a list of available analysis types and preset configuration levels
  
  \texttt{amplxe-cl -collect-list}

• Run Hot Spot analysis on target \texttt{myApp} and store result in default-named directory, such as \texttt{r000hs}
  
  \texttt{amplxe-cl -c hotspots -- myApp}

• Run the Parallelism analysis, store the result in directory \texttt{r001par}
  
  \texttt{amplxe-cl -c parallelism -result-dir r001par -- myApp}
>$ amplxe-cl -report summary -r /home/user1/examples/lab2/r003cc

Summary
-------
Average Concurrency: 9.762
Elapsed Time: 158.749
CPU Time: 561.030
Wait Time: 190.342
CPU Usage: 3.636
Executing actions 100 % done
### Intel® VTune™ Amplifier XE

#### Command Line Interface – Gropof-like output

```bash
$ amplxe-cl -report gprof-cc -r /home/levent/examples/cern/labs/AXE_lab3/r003cc
```

```bash
Using result path /home/levent/examples/cern/labs/AXE_lab3/r003cc.
```

<table>
<thead>
<tr>
<th>Index</th>
<th>% Wait</th>
<th>Time:Total</th>
<th>Time:Self</th>
<th>Children</th>
<th>Name</th>
<th>Name</th>
<th>Name</th>
<th>Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>99.88</td>
<td>190.104</td>
<td>190.104</td>
<td>G4RunManager::BeamOn</td>
<td>[23]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>ParRunManager::DoEventLoop</td>
<td>[0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.162</td>
<td>0.162</td>
<td>operator&lt;&lt;</td>
<td>[17]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.025</td>
<td>0.025</td>
<td>G4RunManagerKernel::G4RunManagerKernel</td>
<td>[11]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>0.1</td>
<td>0.186</td>
<td>0.001</td>
<td>G4strstreambuf::sync</td>
<td>[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.001</td>
<td>0.001</td>
<td>G4MycoutDestination::ReceiveG4cout</td>
<td>[5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td>83.08</td>
<td>0.033</td>
<td>158.141</td>
<td>func@0x416c28</td>
<td>[7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>158.108</td>
<td>158.108</td>
<td>main</td>
<td>[2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>G4_main</td>
<td>[18]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td>0.0</td>
<td>0.002</td>
<td>0.002</td>
<td>CLHEP::HepRandom::showEngineStatus</td>
<td>[22]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>CLHEP::RanecuEngine::showStatus</td>
<td>[3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[4]</td>
<td>0.0</td>
<td>0.001</td>
<td>0.001</td>
<td>G4_main</td>
<td>[18]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[5]</td>
<td>0.0</td>
<td>0.001</td>
<td>0.001</td>
<td>G4MycoutDestination::G4MycoutDestination</td>
<td>[4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[6]</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>G4UImanager::ExecuteMacroFile &lt;cycle 1&gt;</td>
<td>[28]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[7]</td>
<td>83.08</td>
<td>0.033</td>
<td>158.141</td>
<td>func@0x416c28</td>
<td>[7]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>158.141</td>
<td>158.141</td>
<td>main</td>
<td>[2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[8]</td>
<td>99.88</td>
<td>0.0</td>
<td>190.107</td>
<td>G4_main</td>
<td>[18]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>190.107</td>
<td>190.107</td>
<td>&lt;cycle 1 as a whole&gt;</td>
<td>[8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Example:

$> amplxe-cl -report hotspots -csv-delimiter=comma -format=csv -report-out=testing111 -r r003cc

Function,Module,CPU Time,Idle:CPU Time,Poor:CPU Time,Ok:CPU Time,Ideal:CPU Time,Over:CPU Time
CLHEP::RanecuEngine::flat,test40,50.751,0,0.050,0.081,0.080,50.541
G4UniversalFluctuation::SampleFluctuations,test40,32.730,0,0.030,0.070,0.010,32.620
sqrt,test40,19.060,0,0.010,0.070,0.030,18.951
G4Track::GetVelocity,test40,15.330,0,0.030,0.030,0.040,15.230
G4VoxelNavigation::LevelLocate,test40,14.460,0,0.020,0.010,0.040,14.390
G4Step::UpdateTrack,test40,14.090,0,0.030,0.020,14.040
G4NavigationLevelRep::G4NavigationLevelRep,test40,13.721,0,0.030,0.020,0.040,13.631
exp,test40,13.438,0,0.038,0.010,0.060,13.330
log,test40,13.340,0,0.180,0.020,0.110,13.030
G4PhysicsVector::GetValue,test40,11.970,0,0.020,0.020,0.050,11.880
Remote Data Collection

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

**One typical model**
- Collect on Linux, analyze and display on Windows
  - The Linux machine is target
- Collect data on Linux system using command line tool
  - Doesn’t require a license
- Copy the resulting performance data files to a Windows* system
- Analyze and display results on the Windows* system
  - Requires a license
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
  – Understand problems limiting CPU instruction level parallelism
  – Instrument user code for better understanding of execution flow defined by threading runtimes
Questions?
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