



# Intel® VTune™ Amplifier XE

**Jackson Maruszczak – Technical Consulting Engineer**



# Agenda

Intro to Intel® VTune™ Amplifier XE

Types of Analysis

Visualizing Report Data

Advanced Features

# Agenda

- Intro to Intel® VTune™ Amplifier XE

Types of Analysis

Visualizing Report Data

Advanced Features

# Intel® VTune™ Amplifier XE

## Performance Profiler

### Where is my application...

#### Spending Time?

Function - Call Stack ▾	CPU Time ▾
algorithm_2	3.560s
do_xform ←	3.560s
algorithm_1	1.412s
BaseThreadInitTh	0.000s

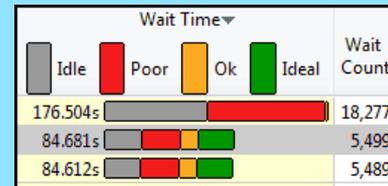
- Focus tuning on functions taking time
- See call stacks
- See time on source

#### Wasting Time?

Line		MEM_LOAD... LLC_MISS
475	float rx, ry, rz =	
476	float param1 = (A2	30,000
477	float param2 = (A2	
478	bool neg = (rz < 0	

- Find cache misses, branch mis-predictions and other inefficiencies
- Tune bandwidth

#### Waiting Too Long?



- See locks by wait time
- Red/Green for CPU utilization during wait

- Windows & Linux
- Low overhead
- No special recompiles

*We improved the performance of the latest run 3 fold. We wouldn't have found the problem without something like Intel® VTune™ Amplifier XE.*

Claire Cates  
Principal Developer, SAS Institute Inc.

## Advanced Profiling For Scalable Multicore Performance

# Intel® VTune™ Amplifier XE

## Tune Applications for Scalable Multicore Performance

### 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

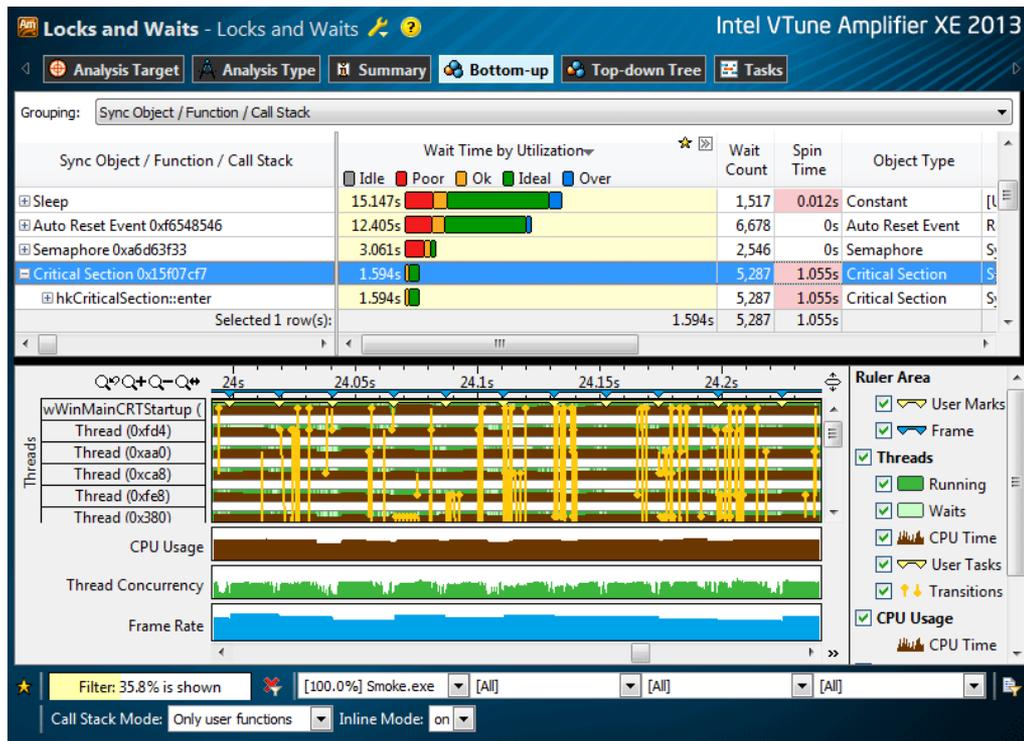
### Compatible

- Microsoft, GCC, Intel compilers
- C/C++, Fortran, Assembly, .NET, Java

- Latest Intel® processors and compatible processors<sup>1</sup>

### Windows or Linux

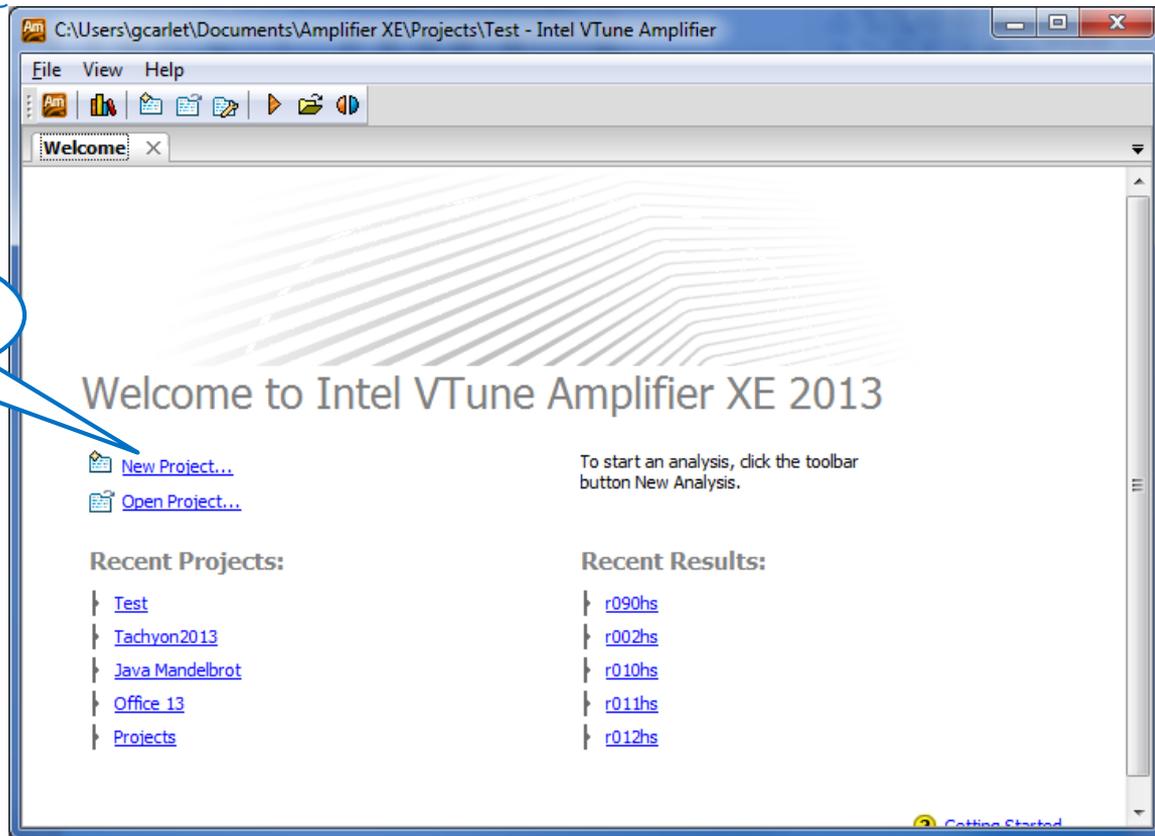
- Visual Studio Integration (Windows)
- Standalone user i/f and command line
- 32 and 64-bit



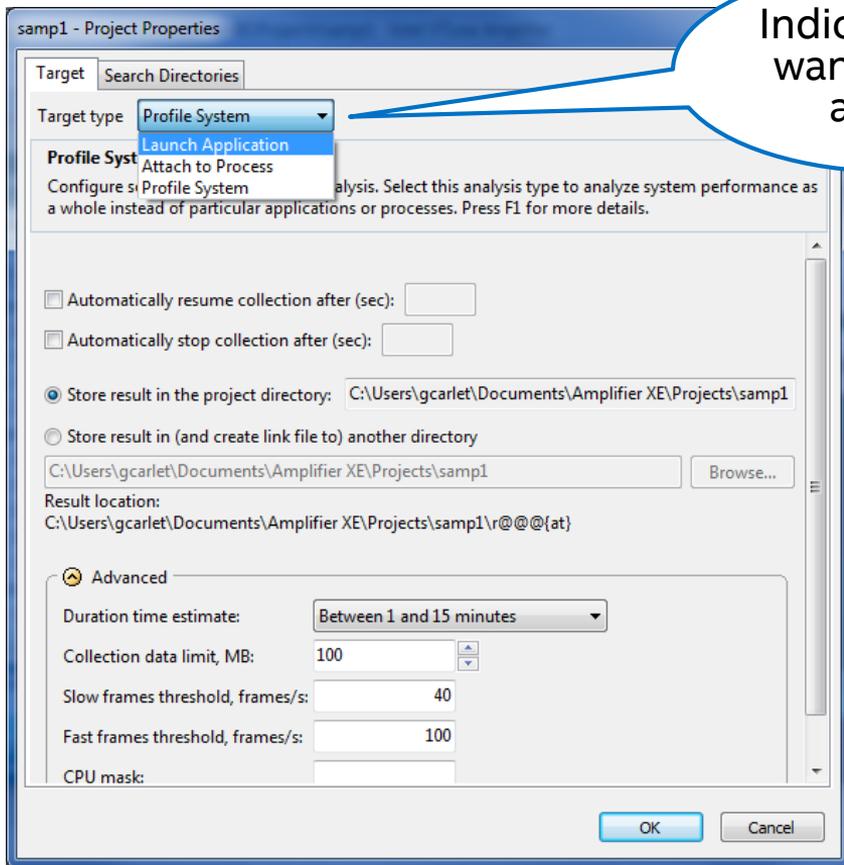
<sup>1</sup> IA32 and Intel® 64 architectures. Many features work with compatible processors. Event based sampling requires a genuine Intel® Processor.

# Starting VTune™ Amplifier XE - The First Time

First create  
a project



# Specify optional app to launch



Indicate if you want to start an app

# Indicate type of profiling

1. Click New analysis button

2. Select profiling type

3. Click "Start" to begin profiling

# Agenda

## Intro to Intel® VTune™ Amplifier XE

- Types of Analysis

## Visualizing Report Data

## Advanced Features

# Two Ways to Collect Data

<b>Software Collector</b> Hotspots, Concurrency, Locks & Waits	<b>Hardware Collector</b> Lightweight Hotspots, Advanced Analysis
Uses OS interrupts	Uses the on chip Performance Monitoring Unit (PMU)
Collects from a single process tree	Collect system wide or from a single process tree.
~10ms default resolution	~1ms default resolution (finer granularity - finds small functions)
Collect on both Intel® and compatible processors	Requires a genuine Intel® processor for collection
Call stacks show calling sequence	New! Optionally collect call stacks
Works in virtual environments	Works in virtual environments only when supported by the VM (e.g., vSphere* 5.1)
No driver required	Requires a driver
<b>No special recompiles - C, C++, C#, Fortran, Java, Assembly</b>	

# Predefined Performance Profiles

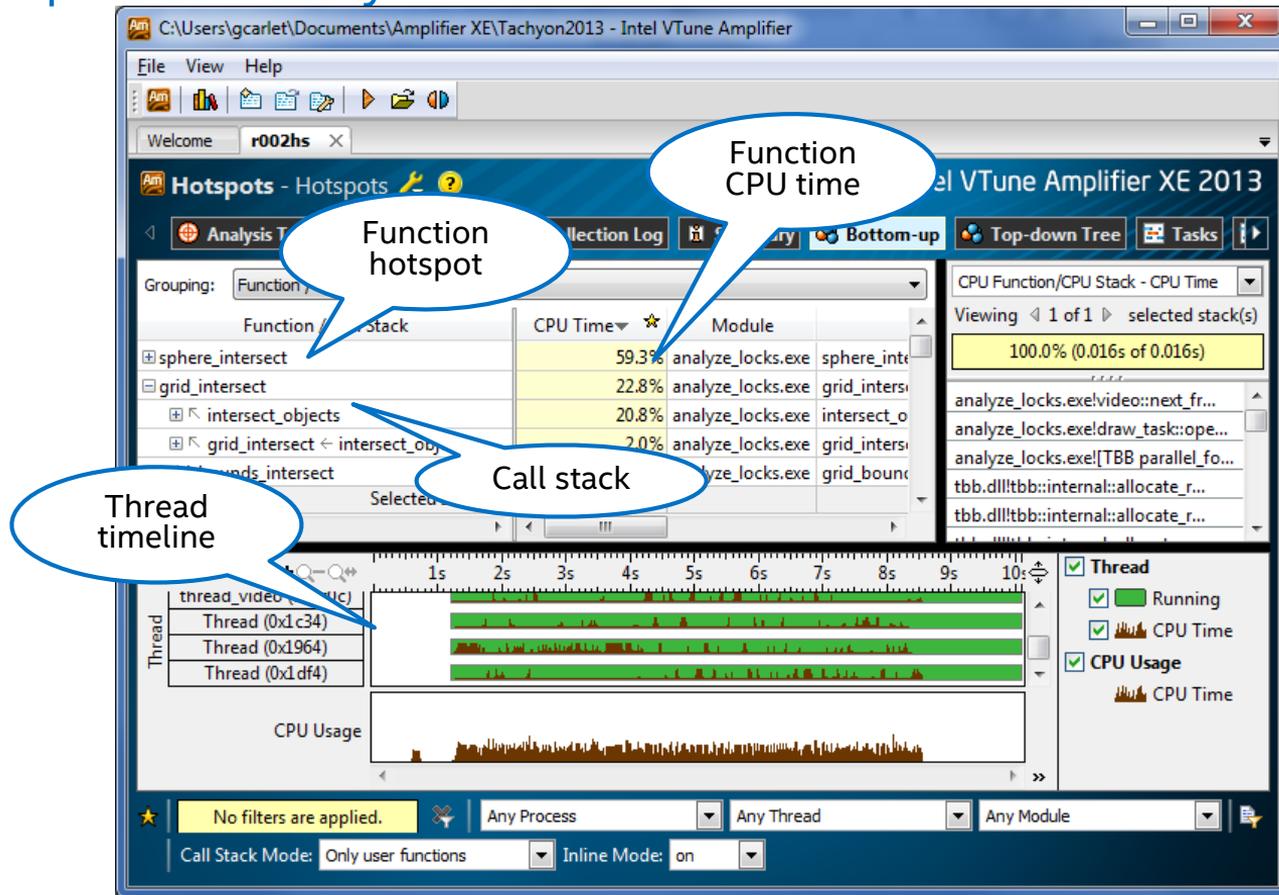
<b>Software Collector</b> Any x86 processor, any virtual, no driver	<b>Hardware Collector</b> Higher res., lower overhead, system wide
<b>Basic Hotspots</b> Which functions use the most time?	<b>Advanced Hotspots</b> Which functions use the most time? Where to inline? – Statistical call counts
<b>Concurrency</b> Tune parallelism. Colors show number of cores used.	<b>General Exploration</b> Where is the biggest opportunity? Cache misses? Branch mispredictions?
<b>Locks and Waits</b> Tune the #1 cause of slow threaded performance – waiting with idle cores.	<b>Advanced Analysis</b> Dig deep to tune bandwidth, cache misses, access contention, etc.

# Hotspots Analysis

Identifies where your application is spending the most time

- Top functions/modules etc...
- Includes call stack information
- Drill down to source lines

# Hotspots Analysis



# Hotspots Analysis – Source View

The screenshot displays the Intel VTune Amplifier XE 2013 interface in Source View. The main window title is "Hotspots - Hotspots" and the application title bar shows "Intel VTune Amplifier XE 2013". The interface includes a menu bar (File, View, Help), a toolbar, and a navigation pane with buttons for Analysis Target, Analysis Type, Collection Log, Summary, Bottom-up, Top-down Tree, and Tasks. The Source View pane shows a table of source code with CPU Time percentages. The selected row (line 581) is highlighted in blue and shows a CPU Time of 5.9%.

Source	CPU Time
578 else if (tmax.z < tmax.y) {	0.0%
579 cur = g->cells[voxindex];	1.2%
580 while (cur != NULL) {	0.0%
581 if (ry->mbox[cur->obj->id] != ry->serial) {	5.9%
582 ry->mbox[cur->obj->id] = ry->serial;	5.5%

The bottom section of the interface shows a timeline view with a "Thread" panel and a "CPU Usage" panel. The Thread panel displays four threads: thread\_video (0x1c0c), Thread (0x1c34), Thread (0x1964), and Thread (0x1df4). The CPU Usage panel shows a bar chart of CPU usage over time. The bottom status bar indicates "No filters are applied." and shows filter settings for Process, Thread, and Module.

# Concurrency Analysis

Identifies how well your application is taking advantage of available processing power

- Determine how much parallelism occurred during a run
- Identify locations of high and low concurrency
- Timeline view helps visualize application concurrency

# Concurrency Analysis

## Summary View

Intel VTune Amplifier XE 2014

Concurrency Hotspots by CPU Usage viewpoint (change)

Analysis Target Analysis Type Collection Log Summary Bottom-up Caller/Callee Top-down Tree Tasks and Frames

### Elapsed Time: 6.107s

Total Thread Count:	6
Overhead Time:	0s
Spin Time:	1.909s
CPU Time:	12.029s
Paused Time:	0s

A significant portion of CPU time is spent waiting. Use this metric to discover which synchronizations are spinning. Consider adjusting spin wait parameters, changing the lock implementation (for example, by backing off then descheduling), or adjusting the synchronization granularity.

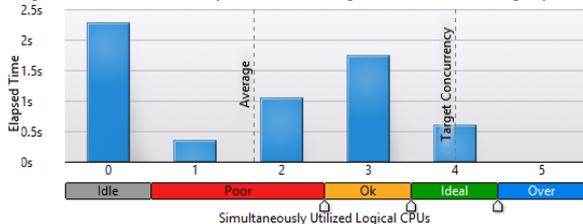
### Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	CPU Time
<a href="#">grid_intersect</a>	5.360s
<a href="#">sphere_intersect</a>	3.542s
<a href="#">SwitchToThread</a>	0.986s
<a href="#">_kmp_launch_thread</a>	0.874s
<a href="#">grid_bounds_intersect</a>	0.297s
[Others]	0.969s

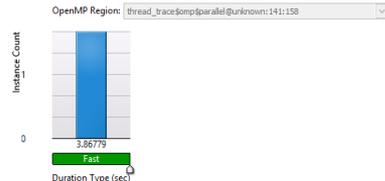
### CPU Usage Histogram

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



### OpenMP Region Duration Histogram

This histogram shows the total number of region instances in your application executed with a specific duration. High number of slow instances may signal a performance bottleneck. Explore the data provided in the Bottom-up, Top-down Tree, and Timeline panes to identify code regions with the slow duration.



### Collection and Platform Info

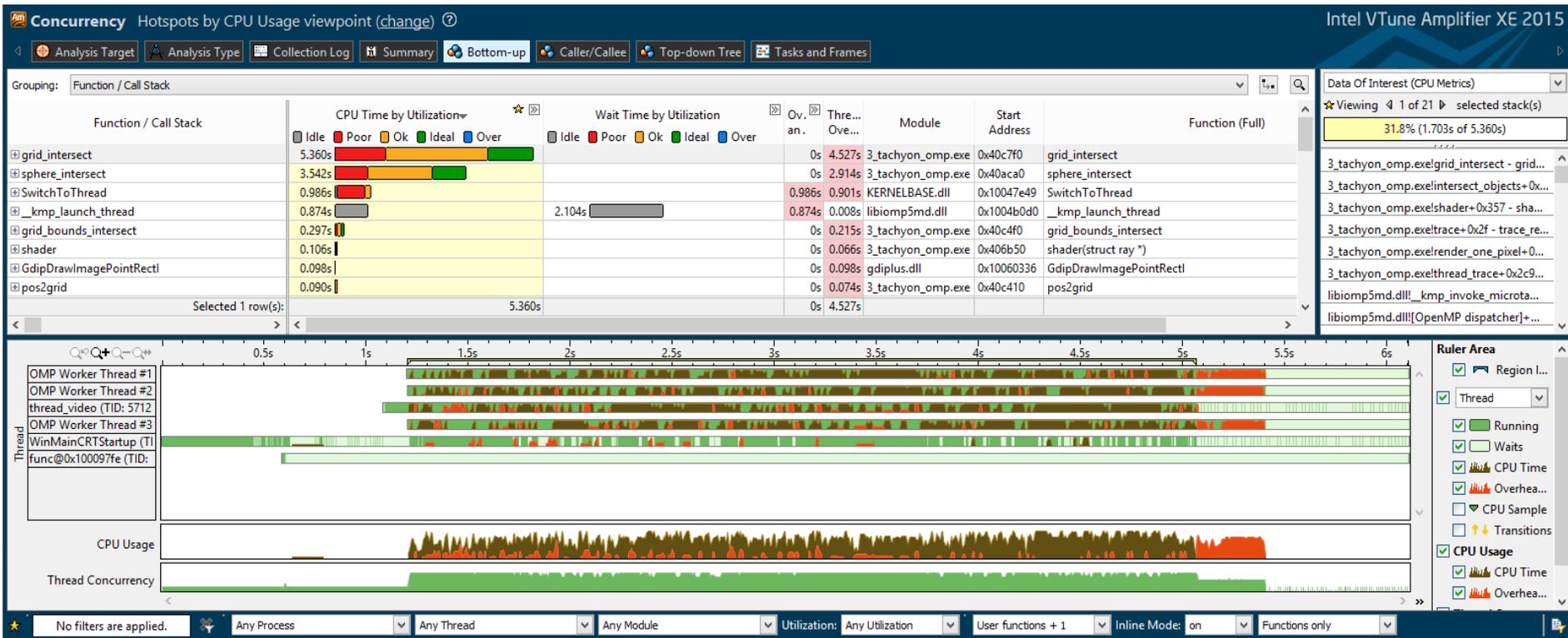
This section provides information about this collection, including result set size and collection platform data.

Application Command Line:	C:\Users\jmarusa\Desktop\tachyon_Advisor\Release3_tachyon_omp.exe
Operating System:	Microsoft Windows
Computer Name:	JMARUSAR-MOBL2.amr.corp.intel.com
Result Size:	3 MB
Collection start time:	17:10:13 21/07/2014 UTC
Collection stop time:	17:10:20 21/07/2014 UTC
<b>CPU</b>	
Name:	4th generation Intel(R) Core(TM) Processor family
Frequency:	2.5 GHz
Logical CPU Count:	4



# Concurrency Analysis

## Bottom-up View



# Locks and Waits Analysis

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

- Synchronization locks
- Threading APIs
- I/O

# Locks and Waits View

Intel VTune Amplifier XE 2011

Analysis Type | Collection Log | Summary | Bottom-up | Top-down Tree

Sync Object	Wait Time	Wait Cou...	Module	Object Type	Object Creation F
Mutex 0x3de2bb60	25.316s	492	[Unknown]	Mutex	thread_trace
draw_task::operator	25.316s	492	tachyon...	Mutex	[Unknown]
TBB Scheduler	10.735s	1	[Unknown]	Constant	tbb::parallel_for<tbb::blocked_ra
Stream 0xc287898b	9.950s	828	[Unknown]	Stream	tbb::parallel_for<tbb::blocked_ra
Stream 0xbbb17798	0.664s	327	[Unknown]	Stream	[libX11.so.6.2]
Socket 0x7b5836b3	0.013s	134	[Unknown]	Socket	[libX11.so.6.2]
Mutex 0x76509717	0.010s	3	[Unknown]	Mutex	drawing_area::~drawing_area
...	...	...	...	...	...
Selected (1 row(s))	25.316s	492			

Object Creation (Use) | 1 stack(s) selected. Viewing 1 of 1 | Current stack is 100.0% of selection | 100.0% (0.014s of 0.014s)

Thread (0x7fffff) | tbb::internal::rml... | CPU Usage | Thread Concurr...

No filters are applied. | Module: [All] | Thread: [All] | Call Stack Mode: Only user functions

# Locks and Waits Source View

The screenshot shows the Intel VTune Amplifier XE 2011 interface. The main window is titled "Locks and Waits - Locks and Waits" and displays a table of source code with wait times and assembly instructions. The table has columns for Line, Source, Wait Time, Address, and Assembly. A red bar indicates a wait time of 25.316s for the pthread\_mutex\_lock call at line 165. Below the table is a timeline view showing thread execution and CPU usage.

Line	Source	Wait Time	Address	Assembly
162	drawing_area drawing(startx, totaly-y, stopx		0x323a	calll 0x804ae00 <
163			0x323f	<b>Block 5:</b>
164	// Acquire mutex to protect pixel calculation		0x323f	add \$0x14, %esp
165	pthread_mutex_lock (&rgb_mutex);	25.316s	0x3242	pushl \$0x805d544
166	for (int x = startx; x < stopx; x++) {		0x3247	calll 0x8049b10 <
167	color_t c = render_one_pixel (x, y, local_r		0x324c	<b>Block 6:</b>
168	drawing.put_pixel(c);		0x324c	movl 0x805d574,
169	}		0x3252	add \$0x10, %esp

Selected (1 row(s)): 25.316s  
Highlighted 2 row(s):

Thread (0x7fffff) tbb::internal::rm...  
CPU Usage  
Thread Concurr...

No filters are applied. Module: [All] Thread: [All] Call Stack Mode: Only user functions

# 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

# Running the General Exploration collector

The screenshot shows the Intel VTune Amplifier XE 2011 interface. The main window is titled "New Amplifier XE Result" and displays the "Choose Analysis Type" dialog. The "Analysis Type" tree view is expanded to show "General Exploration" selected under "Advanced Intel(R) Core(TM) 2 Processor Family Analysis". The "Start" button is highlighted with a callout. The "Project Navigator" on the left shows a project named "Tachyon".

1. Click "New Analysis" button

2. Select "General Exploration"

3. Click "Start" to begin profiling

# CPU HW Sampling results

General Exploration - General Exploration

Analysis Target Analysis Type Summary Bottom-up Top-down Tree Tasks and Frames

Grouping: Function / Call Stack

Function / Call Stack	Hardware Ev ... CPU_CL... THREAD	Hardware Ev ... INST_RETIRED. ANY	CPI Rate	Filled Pipeline Slots		Un...	Ba. Bo.	Bo	Ba.	Bo	analyze_locks.exe	grid_inter
				Re.	Bad Speculation							
				Branch Mispredict	Machine Clears							
grid_intersect	8,130,000,000	10,560,000,000	0.770	0.328	0.162	0.006	0.458	0.125			analyze_locks.exe	grid_inter
sphere_intersect	7,512,000,000	10,326,000,000	0.727	0.372	0.016	0.027	0.478	0.038			analyze_locks.exe	sphere_ir
grid_bounds_intersect	1,196,000,000	862,000,000	1.387	0.192	0.100	0.000	0.607	0.088			analyze_locks.exe	grid_bour
GdipCreateSolidFill	652,000,000	564,000,000	1.156	0.276	0.000	0.000	0.628	0.073			gdiplus.dll	GdipCreat
pos2grid	246,000,000	236,000,000	1.042	0.193	0.000	0.000	0.563	0.071			analyze_locks.exe	pos2grid
[rdpdd.dll]	226,000,000	476,000,000	0.475	0.431	0.531	0.000	0.226	0.232			rdpdd.dll	[rdpdd.dll]
shader	208,000,000	160,000,000	1.300	0.252	0.000	0.000	0.760	0.108			analyze_locks.exe	shader(st
[TBB Scheduler Internals]	170,000,000	46,000,000	3.696	0.544	0.000	0.000	0.441	0.000			tbb.dll	tbb::inter
Raypnt											analyze_locks.exe	Raypnt(sl

Performance problems are highlighted

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

# Agenda

Intro to Intel® VTune™ Amplifier XE

Types of Analysis

- Visualizing Report Data

Advanced Features

# Find Answers Fast

## Adjust Data Grouping

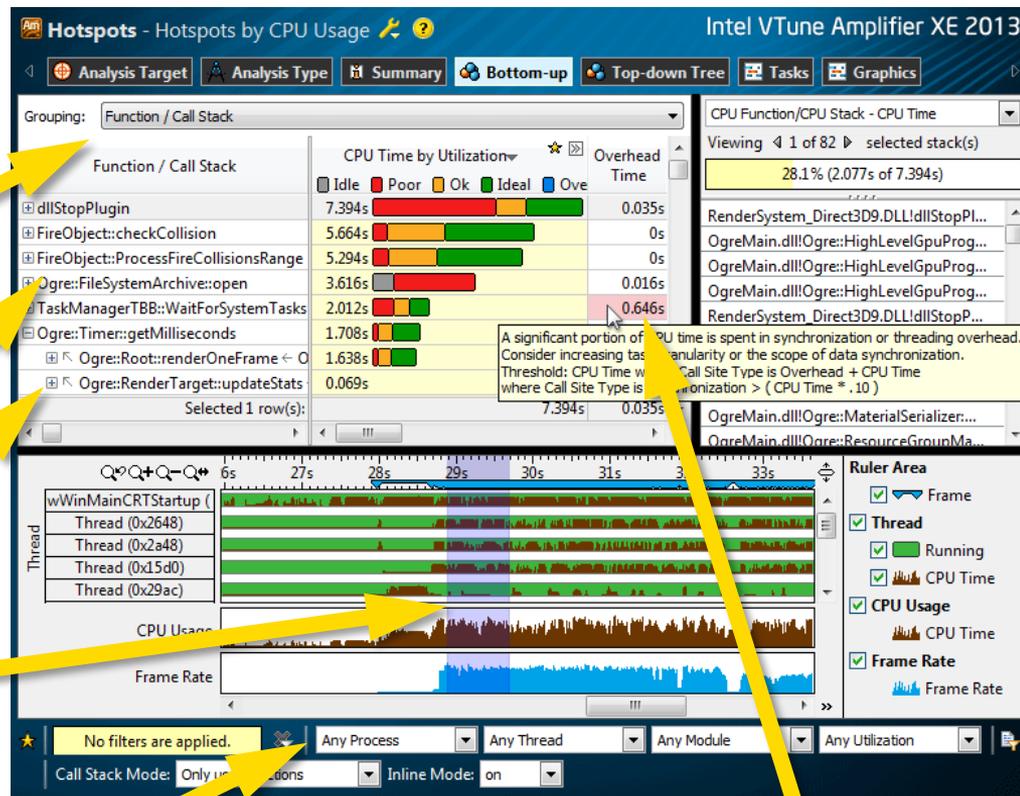
- Function - Call Stack
- Module - Function - Call Stack
- Source File - Function - Call Stack
- Thread - Function - Call Stack
- ... (Partial list shown)

## Double Click to View Source

## Click [+] for Call Stack

## Filter by Timeline Selection (or by Grid Selection)

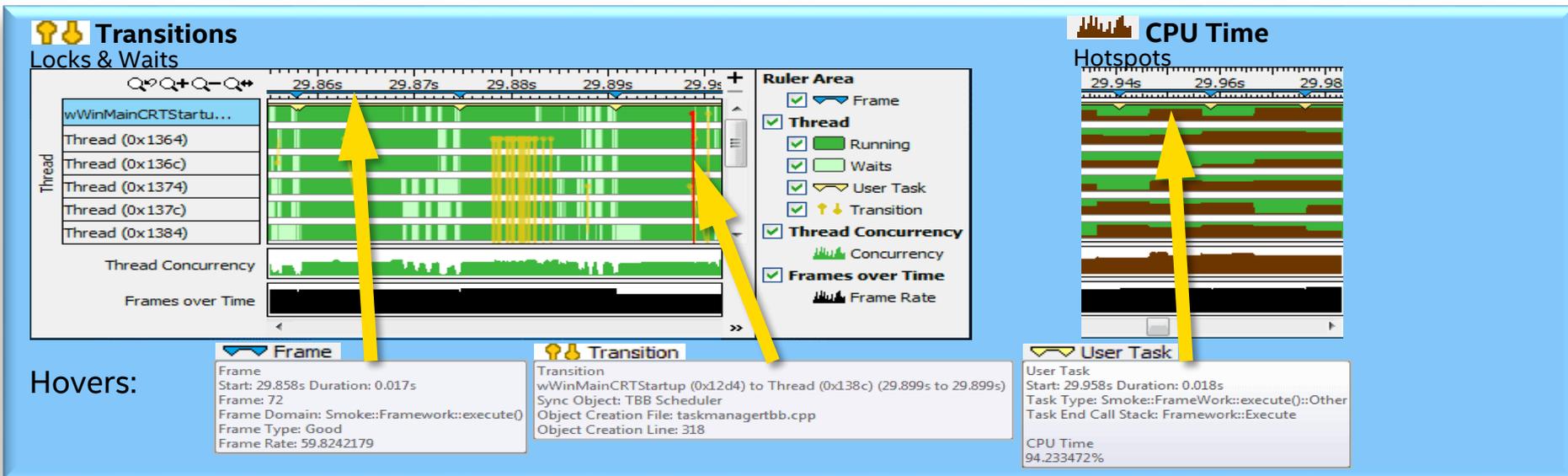
- Zoom In And Filter On Selection
- Filter In by Selection
- Remove All Filters



Filter by Process & Other Controls

Tuning Opportunities Shown in Pink. Hover for Tips

# Timeline Visualizes Thread Behavior



Optional: Use API to mark frames and user tasks

Optional: Add a mark during collection



# See Event Data On Source / Asm

Double Click from Grid or Timeline

View Source / Asm or both

Events

The screenshot displays the Intel VTune Amplifier XE 2013 interface. The top bar shows the application name and various toolbars. The main area is split into two panes: Source and Assembly. The Source pane shows C++ code with performance metrics (CPI Rate, Reti) for each line. The Assembly pane shows the corresponding assembly instructions with their own performance metrics. A blue box highlights source line 1403, and a yellow arrow points from this box to the corresponding assembly instruction. Another yellow arrow points from the assembly instruction back to the source code. A third yellow arrow points from the assembly instruction to a blue box that says 'Click jump to scroll Asm'. A fourth yellow arrow points from the assembly instruction to a blue box that says 'Right click for instruction reference manual'. The bottom of the interface shows filter settings for Process, Thread, Module, and Timeline Hardware Event.

Source Line	Source	CPI Rate		
		CPU_CLK... THREAD	INSTRU... ANY	UOI RET
1402	#else /* !NO_MUTUAL_FIRE_COLLISION_C...			
1403	u32 rangeEnd = (u32)m_pFireO...	24,000,000	16,000,000	16,0
1404	u32 rangeBegin = 0;			
1405	#endif /* !NO_MUTUAL_FIRE_COLLISION_C...			
1406				

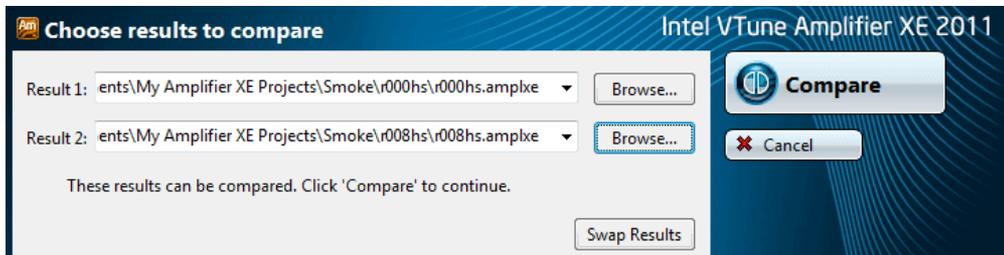
Code Location	Source Line	Assembly	CPI Rate	
			CPU_CL... THREAD	INSTRU... ANY
0x100060e7	1383	jmp 0x100060f0 <Block 4>		
		Block 3:		
0x100060e9	1383	lea esp, pt...		
		Block 4:		
0x100060f0	1403	mov esi, dword ptr [ebp+0x8]	4,000,000	2,000,000
0x100060f3	1403	mov eax, dword ptr [esi+0x344]	4,000,000	0
0x100060f9	1403	mov ebx, dword ptr [eax+0x10]	8,000,000	4,000,000
0x100060fc	1403	sub ebx, dword ptr [eax+0xc]	8,000,000	10,000,000
0x100060ff	1408	mov edi, dword ptr [ebp+0x10]	8,000,000	0

Quick Asm navigation:  
Select source to highlight Asm

Click jump to scroll Asm

Right click for instruction  
reference manual

# Compare Results Quickly - Sort By Difference



Function - Call Stack	CPU Time: r000hs	CPU Time: r008hs	CPU Time:Difference	Module
FireObject::checkCollision	22.317s	18.897s	3.420s	SystemProceduralI
BaseThreadInitThunk	29.797s	26.508s	3.289s	kernel32.dll
FireObject::ProcessFireCollisionsRange	18.892s	17.169s	1.723s	SystemProceduralI

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?

# Agenda

Intro to Intel® VTune™ Amplifier XE

Types of Analysis

Visualizing Report Data

- Advanced Features

# Command Line Interface

amplxe-cl is the command line.

**Windows:** C:\Program Files\Intel\Amplifier XE \bin[32|64]\amplxe-cl.exe

**Linux:** /opt/intel/amplifier\_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`

 **Get Command Line**

CLI is very useful for running The Amplifier XE as part of Performance regression testing

# Regression Testing

Create Baseline:

```
$> ampxe-cl -collect  
hotspots -r BaseLinePerf -- myapp.exe  
$> ampxe-cl -report hotspots -r BaseLinePerf
```

Nightly Performance Regression Testing:

```
$> ampxe-cl -collect  
hotspots -r nightlyresults -- myapp.exe  
$> ampxe-cl -report hotspots -r BaseLinePerf -r  
NightlyResults
```

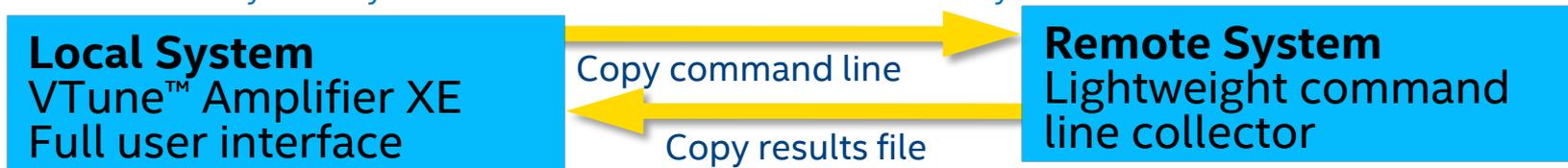
[...]

Module	Process	Result 1:CPU Time	Result 2:CPU Time	Difference:CPU Time
myapp.exe	myapp.exe	23.141	61.531	-38.391

...

# 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 the target system
4. Paste command line, run collection
5. Copy result to your system
6. Open file using local GUI

- Minimal “performance footprint” during collection
- Easy setup using GUI
- Easy analysis of results

# Tune MPI Apps Single Node Threading

## Intel® VTune™ Amplifier XE Performance Profiler

### Launch Intel® VTune™ Amplifier XE

- Use mpirun or mpiexec
- List your app as a parameter

Results organized by MPI rank

### Review results

- Graphical user interface
- Command line report

Tune for Scalable Multicore Performance

# Using VTune Amplifier XE with Hybrid MPI + Threads

Use the command-line tool under the MPI run scripts to gather report data

```
$ mpirun -n 4 amplxe-cl --result-dir ampl_results -collect hotspots --  
./example.exe
```

A results directory is created for each MPI rank

- Can use arg sets to filter on a subset of ranks

Launch the GUI and view the results for each particular rank

```
$ amplxe-gui ampl_results.<rank#>
```

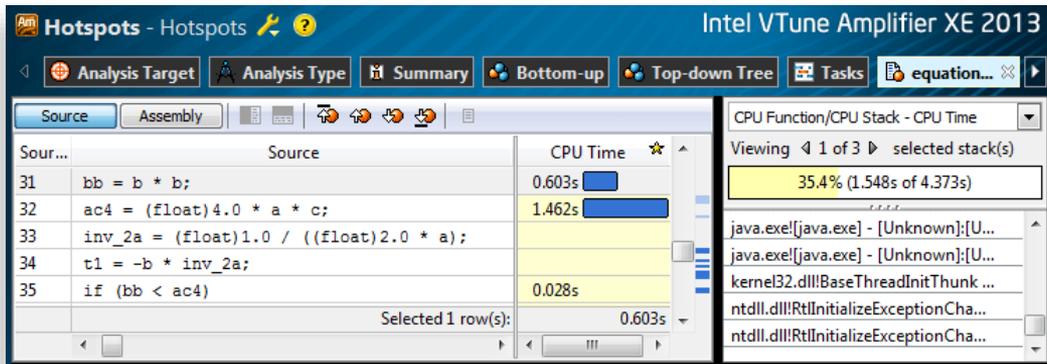
# Low Overhead Java\* Profiling

## Low Overhead & Precise

- Sampling is fast / unobtrusive
- Hardware sampling even faster (Now with optional stacks!)
- Advanced profiles are unique (cache misses, bandwidth...)

## Versatile & Easy to Use

- Multiple simultaneous JVMs
- Mixed Java / C++ / Fortran
- See results on the Java source



**Better data, lower overhead, easier to use**

# Intel® Xeon Phi™ coprocessor support

Hardware CPU event profiling (CPU usage, cache misses,...)

The screenshot shows the Intel VTune Amplifier XE 2013 interface. The main window title is "/home/michome/rreed/projects/matrix/linux/matrix-mic - Intel VTune Amplifier". The interface is titled "Choose Analysis Type" and shows a tree view on the left with "Lightweight Hotspots" selected under "Knights Corner Platform Analysis". The main panel displays the configuration for "Lightweight Hotspots - Knights Corner Platform".

**Lightweight Hotspots - Knights Corner Platform** Copy

Identify your most time-consuming source code. Unlike Hotspots, Lightweight Hotspots has lower overhead when stack collection is disabled. Reduced overhead makes it possible to set a lower sampling interval than Hotspots (as low as 1ms without stacks), which is useful for locating small functions that are called frequently. This analysis type can also be used to sample all processes on a system. Press F1 for more details. Press F1 for more details.

List of Intel Xeon Phi coprocessor cards:

Analyze user tasks

**Details**

Events configured for CPU: Intel(R) Xeon(R) E5 processor

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.

Event Name	Sample After	Event Description
CPU_CLK_UNHALTED	10000000	
INSTRUCTIONS_EXECUTED	10000000	

On the right side of the interface, there are buttons for "Start", "Start Paused", and "Project Properties".

# 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 cycles

# Legal Disclaimer & Optimization Notice

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 © 2014, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo 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

