



# Intel® VTune™ Amplifier XE Generics

Rev.: 22/09/2015



# Agenda

Introduction to Intel® VTune™ Amplifier XE profiler

High-level Features

Types of Analysis

Hotspot analysis

- Basic Hotspots
- Advanced Hotspots

Concurrency Analysis

Locks and Waits Analysis

User and Synchronization API, Frame/Task Analysis

Command Line Interface, Installation, Remote Collection

Conclusion

# Intel® VTune™ Amplifier XE

## Performance Profiler

### Where is my application...

#### Spending Time?

Function	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 = (AP	30,000
477	float param2 = (AP	
478	bool neg = (rz < 0	

- See cache misses on your source
- See functions sorted by # of cache misses

#### Waiting Too Long?

	Wait Time	Wait Count
176.504s	Idle Poor Ok Ideal	18,277
84.681s	Idle Poor Ok Ideal	5,499
84.612s	Idle Poor Ok Ideal	5,489

- 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

# 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

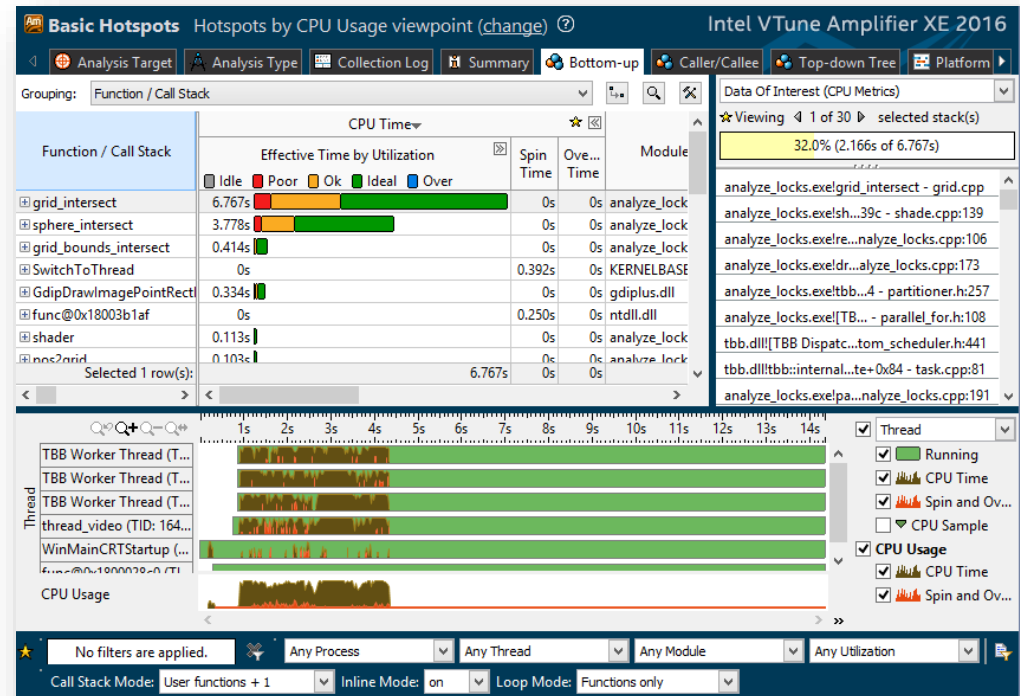
- Filter extraneous data
- View results on the source / assembly

### 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.

# A set of instruments to identify performance problems

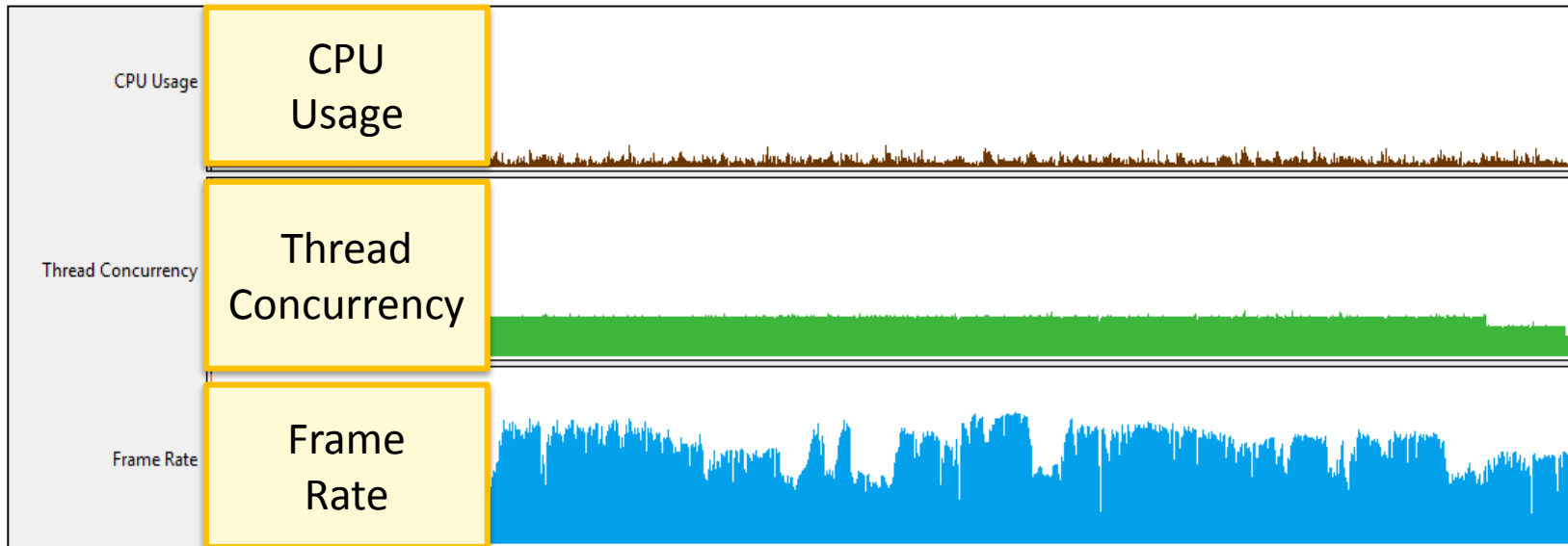
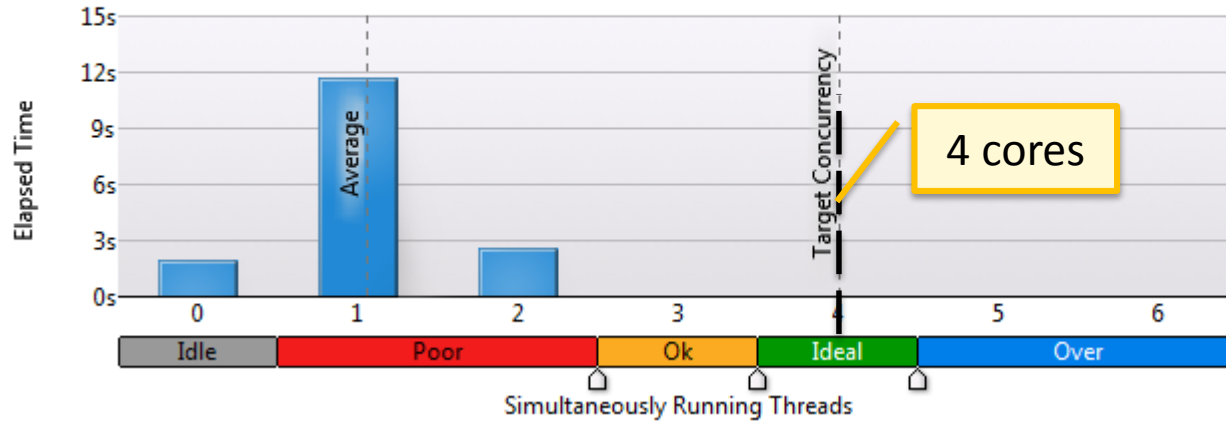
Quick Overview

# Intel® VTune™ Amplifier XE

## Get a quick snapshot

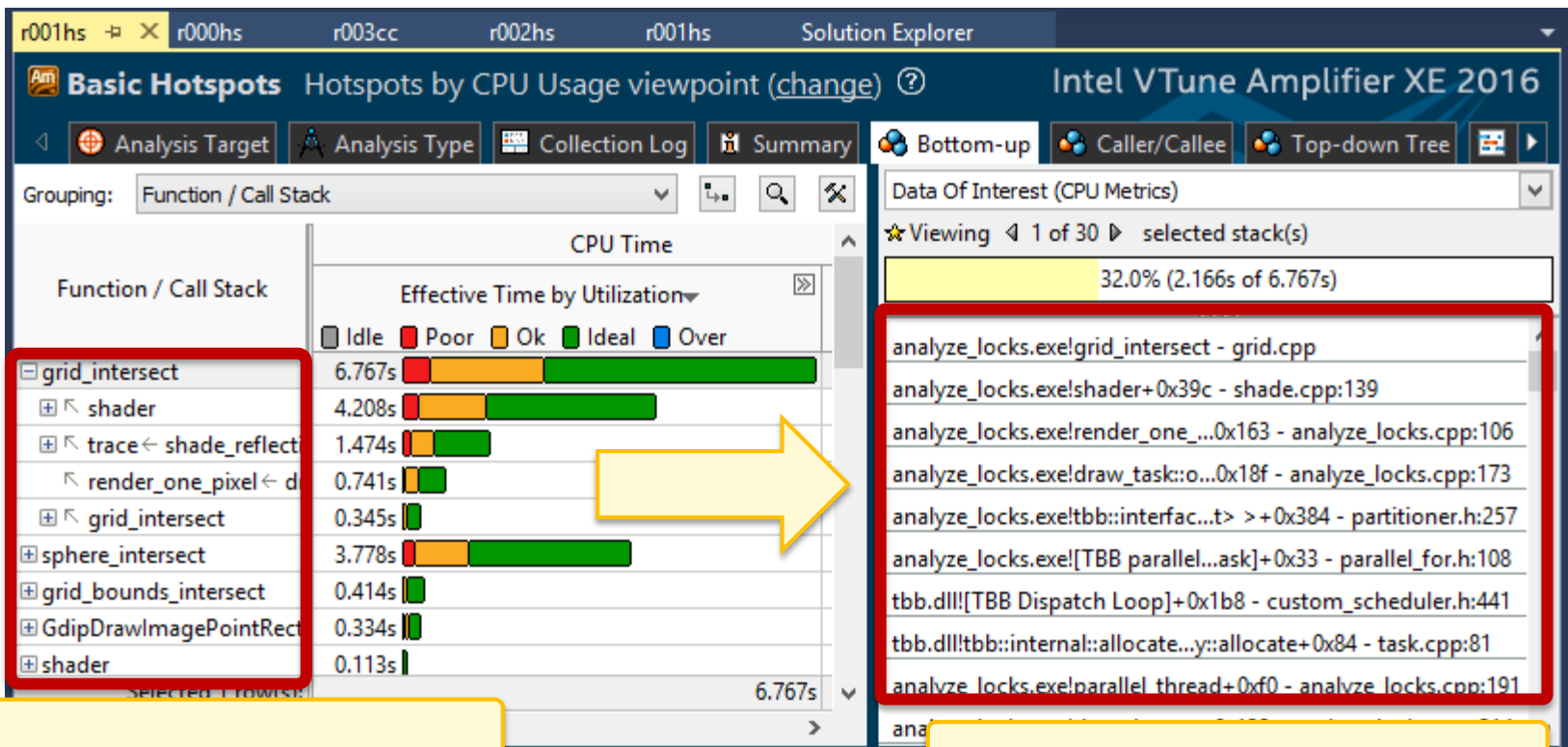
### Thread Concurrency Histogram

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.



# Intel® VTune™ Amplifier XE

Identify hotspots



Hottest Functions

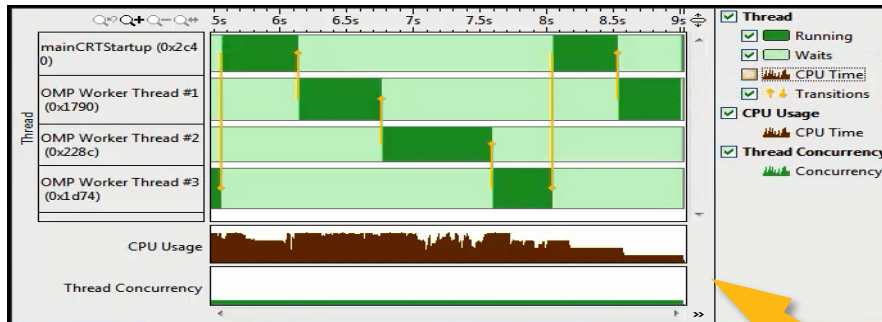
Hottest Call Stack

Quickly identify what is important

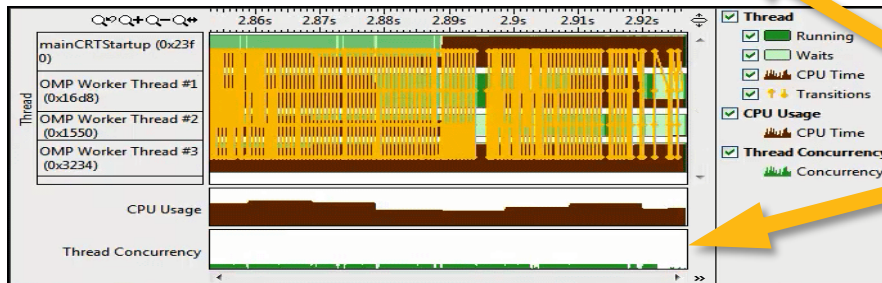
# Intel® VTune™ Amplifier XE

## Look for Common Patterns

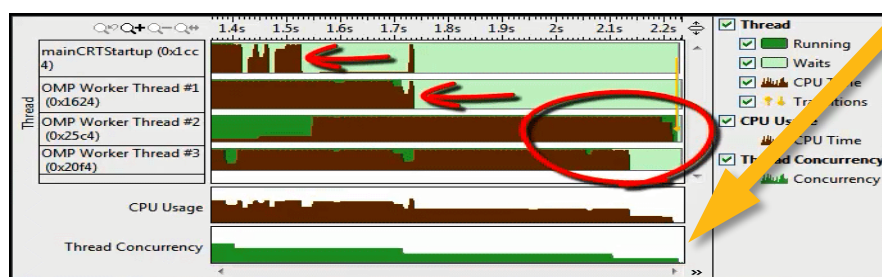
Coarse Grain Locks



High Lock Contention



Load Imbalance



Low  
Concurrency



# Intel® VTune™ Amplifier XE

## Find Answers Fast

Adjust Data Grouping

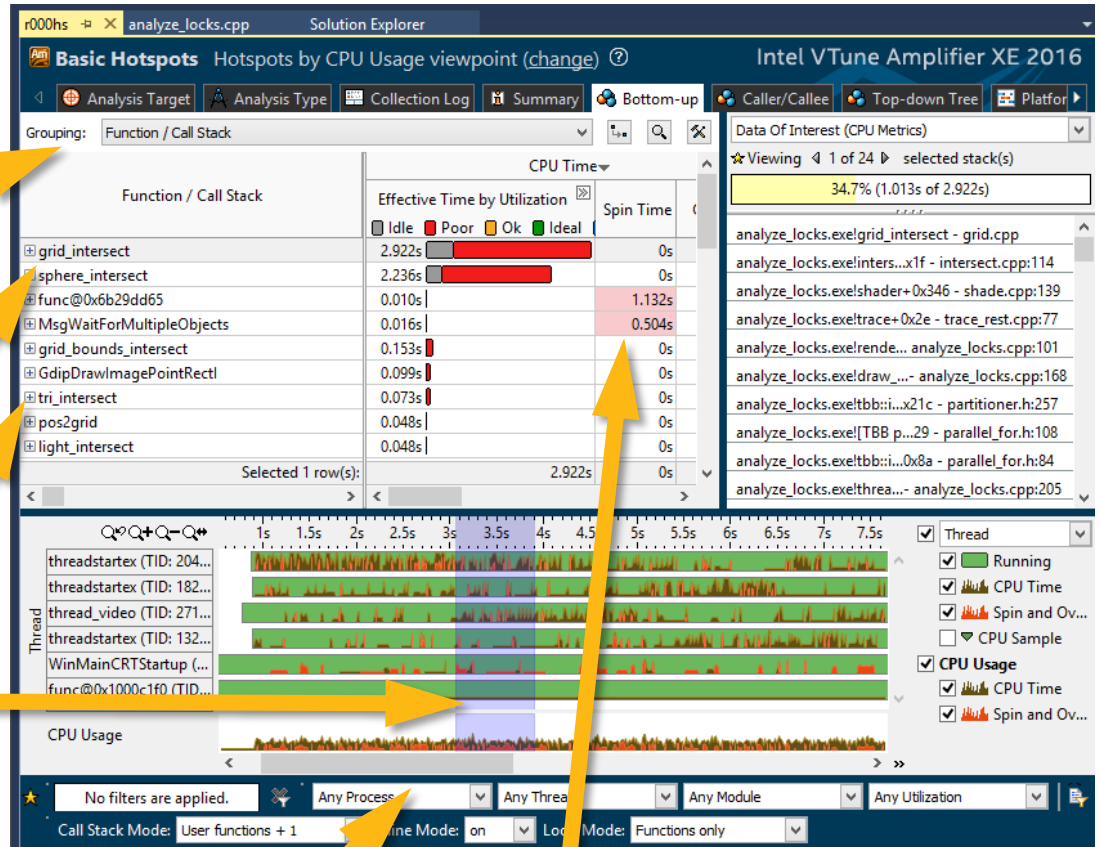
- Function - Call Stack
- Module - Function - Call Stack
- Source File - Function - Call Stack
- Thread - Function - Call Stack
- ... (Partial list shown)

Double Click Function 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

# Intel® VTune™ Amplifier XE

## Timeline Visualizes Thread Behavior

**Transitions**  
Locks & Waits

**CPU Time**  
Basic Hotspots | Advanced Hotspots

**Thread List:** wWinMainCRTstartu..., Thread (0x1364), Thread (0x136c), Thread (0x1374), Thread (0x137c), Thread (0x1384)

**Ruler Area:** 29.86s, 29.87s, 29.88s, 29.89s, 29.9s

**Thread Concurrency:** [Bar chart showing concurrency over time]

**Frames over Time:** [Bar chart showing frame rate over time]

**Frame Details:**  
Frame  
Start: 29.858s Duration: 0.017s  
Frame: 72  
Frame Domain: Smoke::Framework::execute()  
Frame Type: Good  
Frame Rate: 59.8242179

**Transition Details:**  
Transition  
wWinMainCRTStartup (0x12d4) to Thread (0x138c) (29.899s to 29.899s)  
Sync Object: TBB Scheduler  
Object Creation File: taskmanagertbb.cpp  
Object Creation Line: 318

**User Task Details:**  
User Task  
Start: 29.958s Duration: 0.018s  
Task Type: Smoke::Framework::execute():Other  
Task End Call Stack: Framework::Execute  
CPU Time: 94.233472%

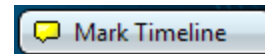
**Optional: Use API to mark frames and user tasks** [Frame icon] [User Task icon]

**Optional: Add a mark during collection** [Mark Timeline button]

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

View Source / Asm or both

CPU Time

Right click for instruction reference manual

The screenshot displays the Intel VTune Amplifier XE 2016 interface. The main window is titled "Basic Hotspots Hotspots by CPU Usage viewpoint (change)". It features a toolbar with options like "Analysis Target", "Analysis Type", "Collection Log", "Summary", "Bottom-up", "Caller/Callee", "Top-down", "Tree", and "Platform". The "grid.cpp" file is open. The interface is split into two panes: "Source" on the left and "Assembly" on the right. The "Source" pane shows C++ code with a "Heat Map" bar indicating CPU usage for each line. The "Assembly" pane shows the corresponding assembly instructions. A "Data Of Interest (CPU Metrics)" sidebar on the right shows "32.0% (2.166s of 6.767s)".

So. Li.	Source	Effective Time	Spin Time	Address	Sour... Line	Assembly	CPU Tim
579	cur = g->cells[voxindex]	1.6%	0.0s	0x1400112d0	581	Block 38:	
580	while (cur != NULL) {	0.1%	0.0s	0x1400112d4	581	mov rax, qword ptr [rdi+0x8]	2.2%
581	if (ry->mbox[cur->obj-]	25.1%	0.0s	0x1400112d7	581	mov edx, dword ptr [rbx+0x10]	8.1%
582	ry->mbox[cur->obj->i	8.5%	0.0s	0x1400112d9	581	mov ecx, dword ptr [rax]	0.2%
583	cur->obj->methods->i	6.1%	0.0s	0x1400112dd	581	mov rax, qword ptr [rbx+0x18]	14.1%
584	}			0x1400112dd	581	cmp dword ptr [rax+rcx*4], edx	0.6%
585	cur = cur->next;	5.8%	0.0s	0x1400112e0	581	jz 0x1400112f2 <Block 40>	
586	}			0x1400112e2	582	Block 39:	
587	curvox.z += step.z;	0.1%	0.0s	0x1400112e2	582	mov dword ptr [rax+rcx*4], edx	8.5%
588	if (ry->maxdist < tmax.z	0.3%	0.0s	0x1400112e5	583	mov rcx, qword ptr [rdi+0x8]	1.6%
589	break;			0x1400112e9	583	mov rdx, rbx	0.3%
590	voxindex += step.z*g->xs	0.1%	0.0s	0x1400112ec	583	mov rax, qword ptr [rcx+0x10]	
	Selected 1 row(s):	25.1%	0.0s	0x1400112f0	583	call qword ptr [rax]	4.3%
				0x1400112f2	585	Block 40:	
				0x1400112f2	585	mov rdi, qword ptr [rdi]	

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

# High-level Features

# Intel® VTune™ Amplifier XE

## Feature Highlights

### Basic 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

# 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 (Android, Tizen, Yocto – in the ISS)
- 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



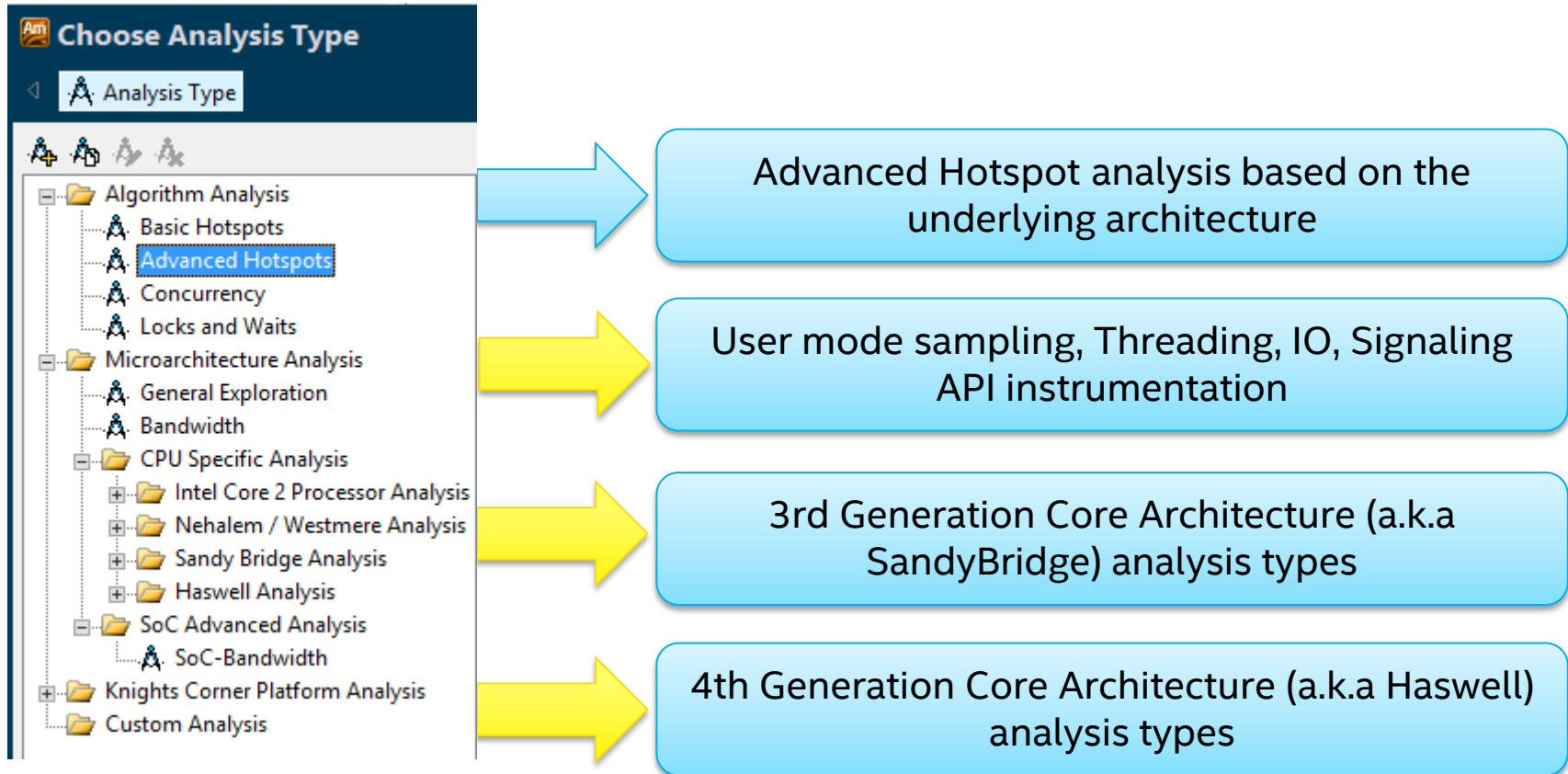
# Intel® VTune™ Amplifier XE

## Analysis Types (based on technology)

<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.

# Intel® VTune™ Amplifier XE

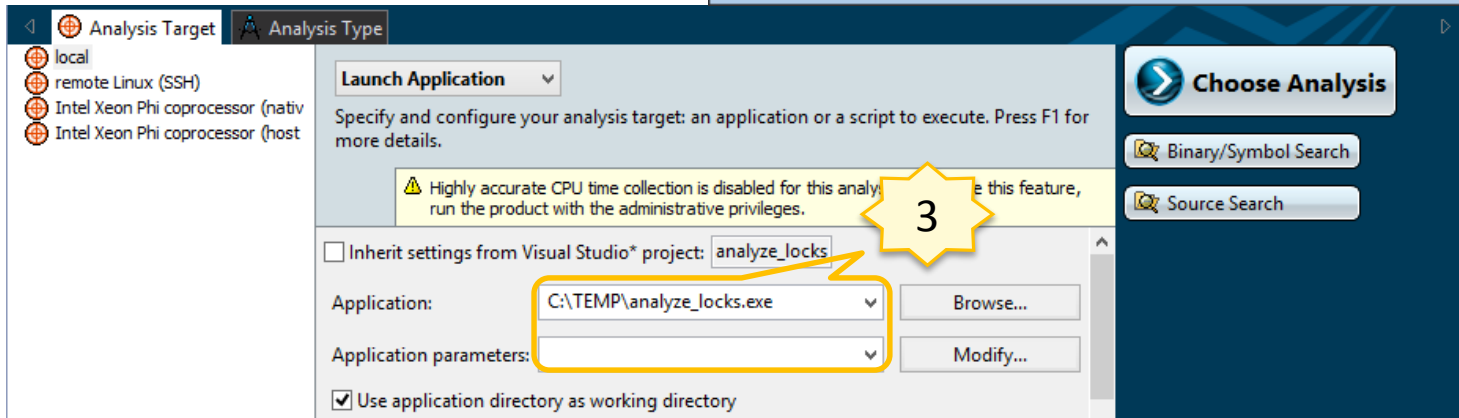
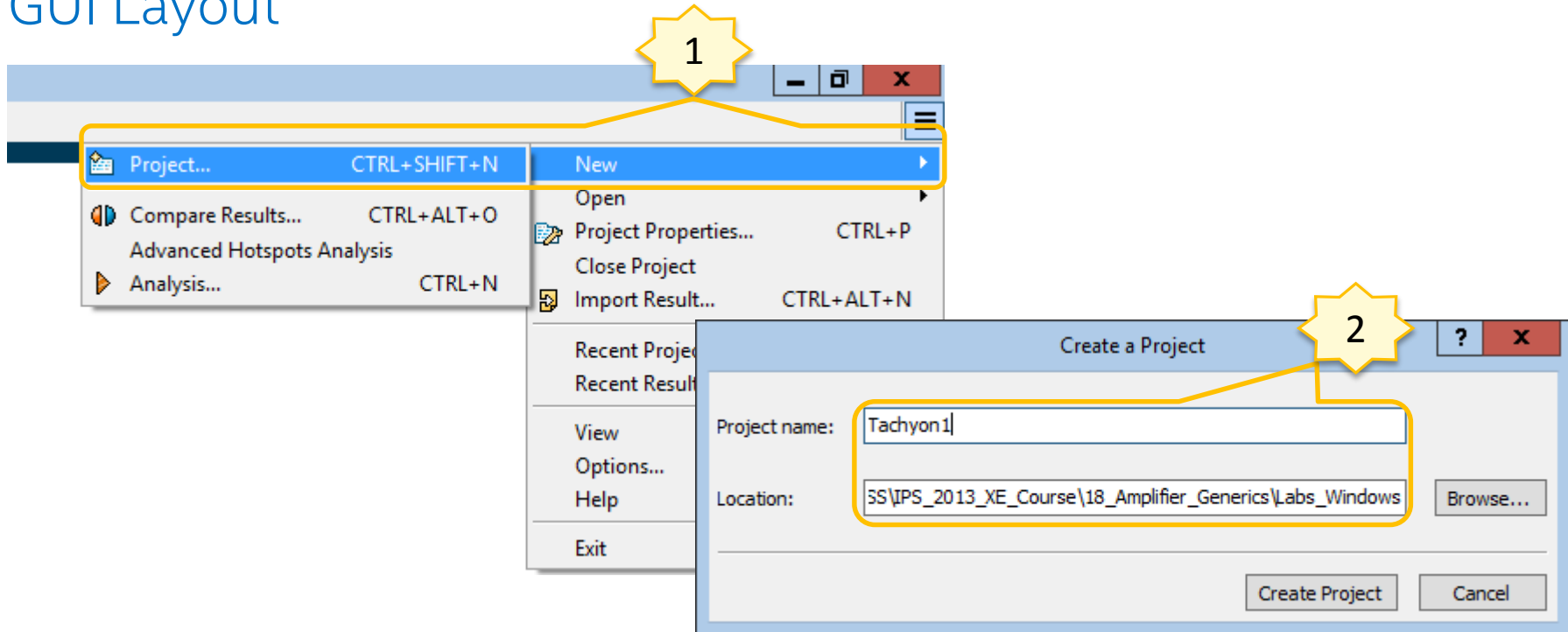
## Pre-defined Analysis Types



# GUI Layout

# Creating a Project

## GUI Layout



# Selecting type of data collection

## GUI Layout

The screenshot shows the Intel VTune Amplifier XE 2016 interface. The title bar reads "Choose Target and Analysis" and "Intel VTune Amplifier XE 2016". The main window is divided into several sections:

- Analysis Target:** A tree view on the left showing various analysis types. A yellow callout bubble points to this area with the text "All available analysis types".
- My comments...:** A text area for adding comments. A yellow callout bubble points to this area with the text "Different ways to start the analysis".
- Configuration:** A list of settings for the analysis, such as "Analyze GPU usage:", "CPU sampling interval, ms:", "Collect highly accurate CPU time:", "Collect CPU...", "Ma...", "Co...", "Collect s...", "Collect I... data:". A yellow callout bubble points to this area with the text "Helps creating new analysis types".
- Buttons:** On the right side, there are buttons for "Start", "Start Paused", and "Choose Target". A yellow callout bubble points to the "Start" button with the text "Copy the command line to clipboard".
- Context Menu:** A context menu is open over the "My comments..." area, showing options like "Copy from Current", "New Hardware Event-based Sampling Analysis", and "New User-mode Sampling and Tracing Analysis". A yellow callout bubble points to the "Copy from Current" option with the text "Copy the command line to clipboard".
- Command Line:** A "Command Line..." button is located at the bottom right of the configuration area.

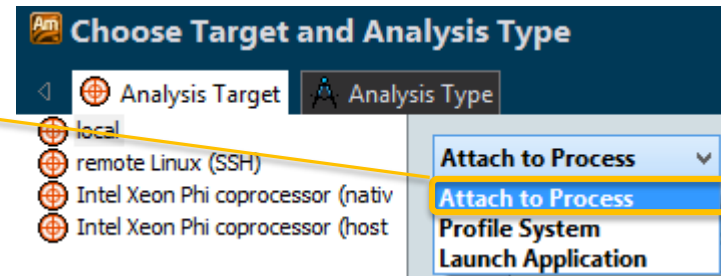
# Profile a Running Application

No need to stop and re-launch the app when profiling

## Two Techniques:

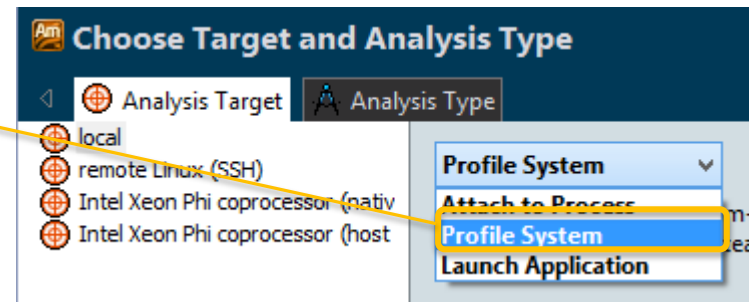
### Attach to Process:

- Any type of analysis



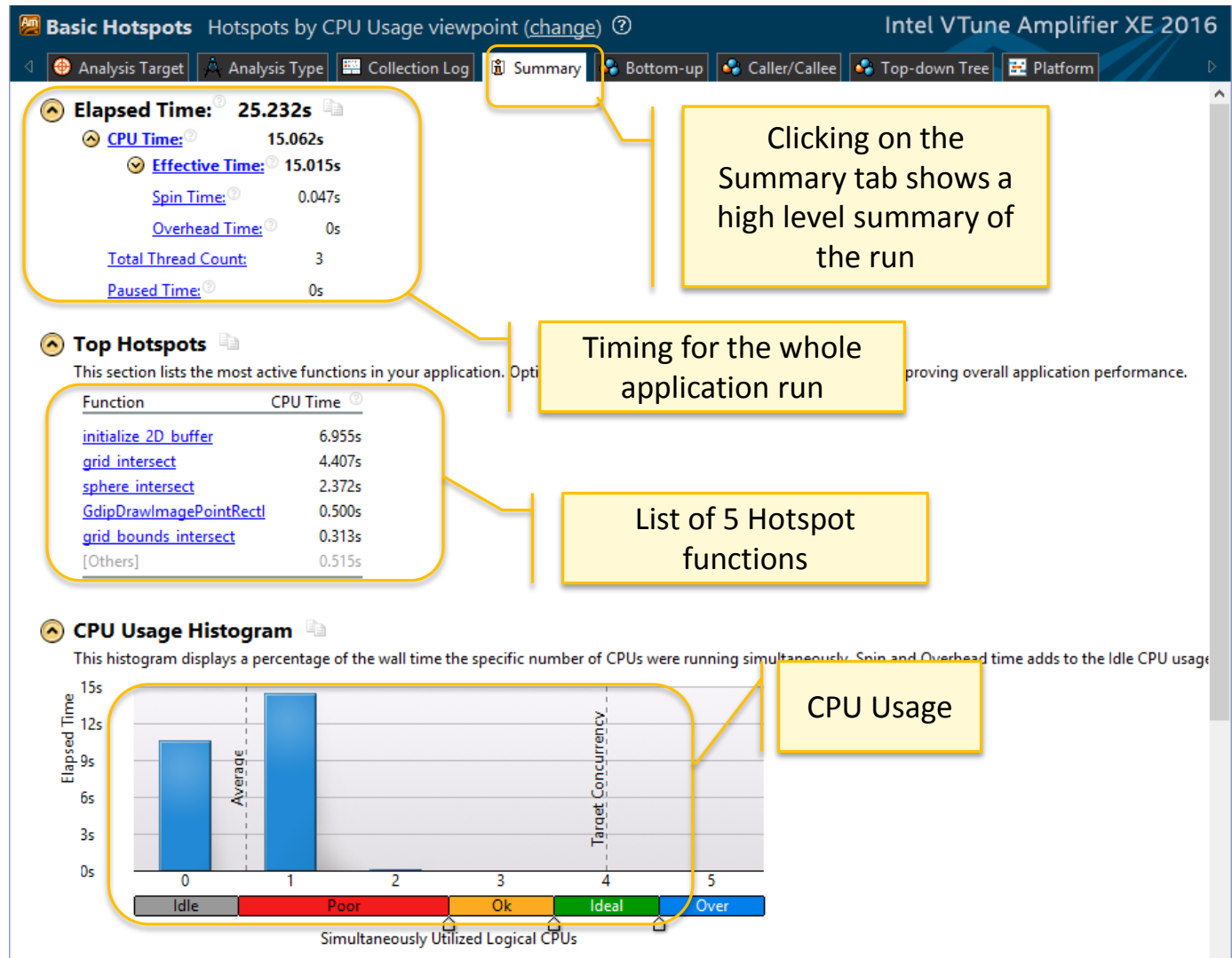
### Profile System:

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



# Summary View

## GUI Layout



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

The screenshot displays the Intel VTune Amplifier XE 2016 interface in the Bottom-Up View. The main window title is "Basic Hotspots" and the subtitle is "Hotspots by CPU Usage viewpoint (change)". The current analysis target is "grid.cpp".

**Table of Hotspots:**

Function / Call Stack	Idle	Ok	Ideal	Effective Time by Utilization
grid_intersect	0s	0.64s	4.440s	0s
sphere_intersect	0s	0.096s	2.662s	0s
grid_bounds_intersect	0s	0.022s	0.085s	0.307s
GdipDrawImagePointRect	0s	0.042s	0.077s	0.215s
shader	0s	0s	0.037s	0.076s
Selected 1 row(s):	0s	0.463s	1.864s	4.440s

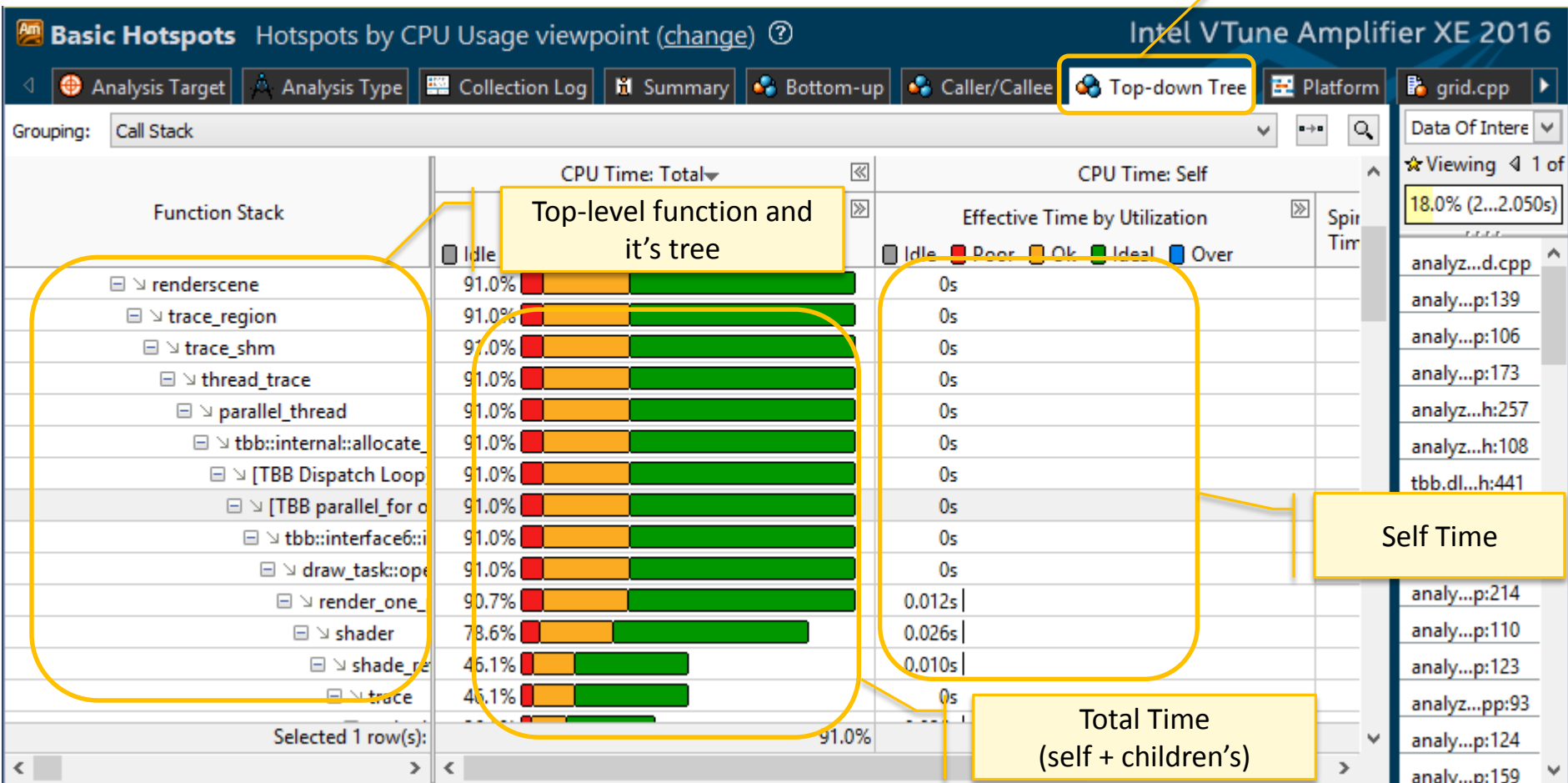
**Callouts and UI Elements:**

- Menu and Analysis Type:** Located at the top left, showing "Analysis Target" and "Analysis Type".
- Viewpoint currently being used:** Points to the "Bottom-up" tab in the top navigation bar.
- Current grouping:** Points to the "Function / Call Stack" grouping option.
- Grid area:** Points to the table of hotspots.
- Stack Pane:** Points to the "Data Of Interest (CPU Metrics)" pane on the right, showing a stack of function calls.
- Filter area:** Points to the filter controls at the bottom, including "No filters are applied", "Any Process", "Any Thread", "Any Module", and "Utilization: A".
- Timeline area:** Points to the "Thread" and "CPU Usage" visualizations at the bottom.



# Top-Down View GUI Layout

Clicking on the Top-Down Tree tab changes stack representation in the Grid



Top-level function and its tree

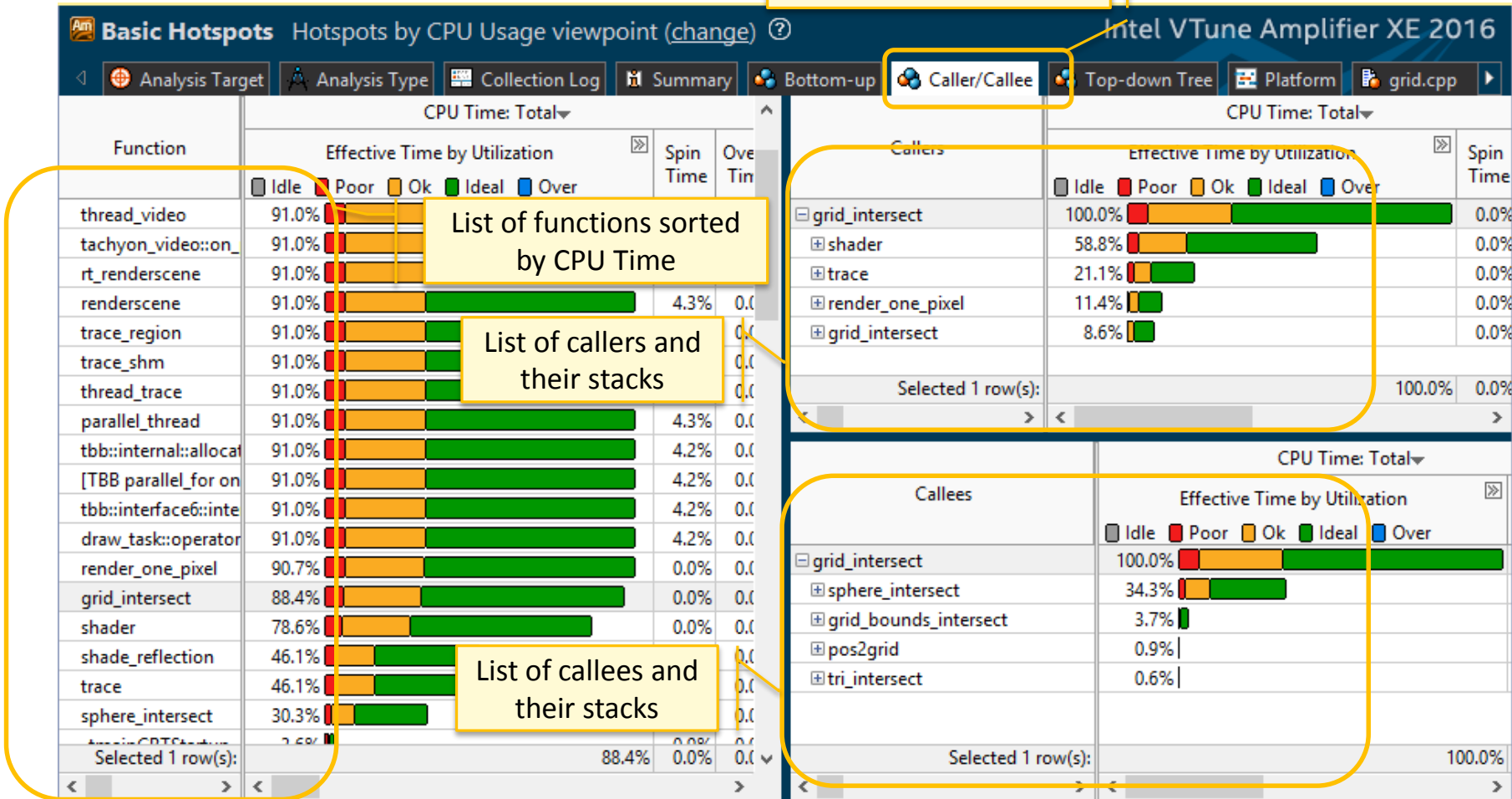
Self Time

Total Time (self + children's)

# 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

The image shows the Intel VTune GUI interface. On the left, a control panel contains buttons for 'Start', 'Start Paused', and 'Project Properties'. The 'Start Paused' button is highlighted with a yellow box. To the right, another control panel contains buttons for 'Resume', 'Pause', 'Stop', 'Cancel', and 'Mark Timeline'. The 'Resume' button is highlighted with a yellow box. Below these panels is a timeline view. The timeline axis is labeled with time intervals: 5s, 10s, 15s, 20s, 25s, 30s, 35s, and 40.538s. The timeline shows several threads: 'wWinMainCRTStartu...', 'Thread (0x1598)', and multiple 'TBB Worker Thread ...' threads. A yellow vertical bar is placed on the timeline at approximately 40.538s, representing a user mark. A yellow callout box points to this mark with the text 'Observe the mark on the Time Line'. Another yellow callout box points to the 'Mark Timeline' button with the text 'Click "Mark Timeline" during collection'. A third yellow callout box points to the 'Resume' button with the text 'Resume data collection when needed'. A fourth yellow callout box points to a yellow shaded region on the timeline with the text 'Observe paused region on the Time Line'. A fifth yellow callout box points to the 'Start Paused' button with the text 'Start application without data collection'.

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

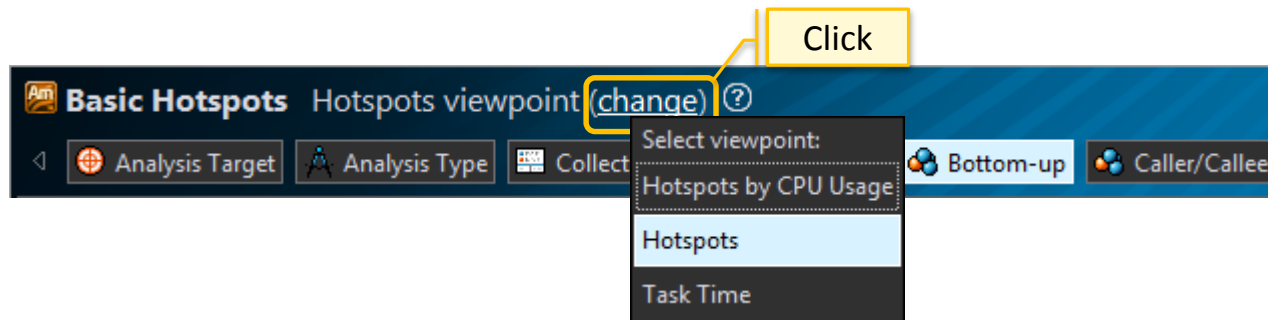
# Key Result Analysis and GUI Concepts

# Result Analysis

## 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 view points
- To change viewpoints, select a viewpoint by clicking on

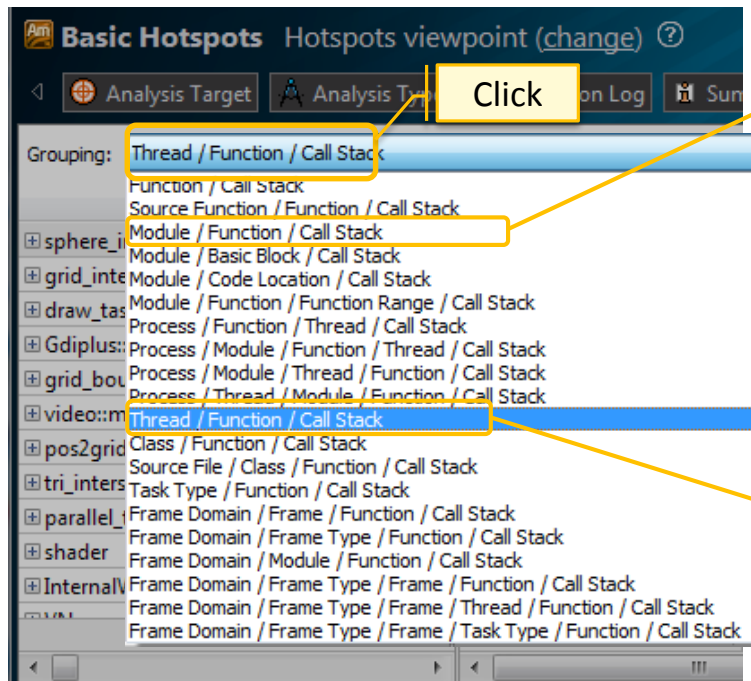


# Result Analysis

## GUI Concepts

### Groupings

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



Grouping: Module / Function / Call Stack

Module / Function / Call Stack	CPU Time
tachyon_analyze_locks.exe	13.367s
sphere_intersect	5.674s
grid_intersect	5.674s
grid_intersect	4.467s
intersect_objects	4.053s
grid_intersect	0.414s
draw_task::operator()	1.640s

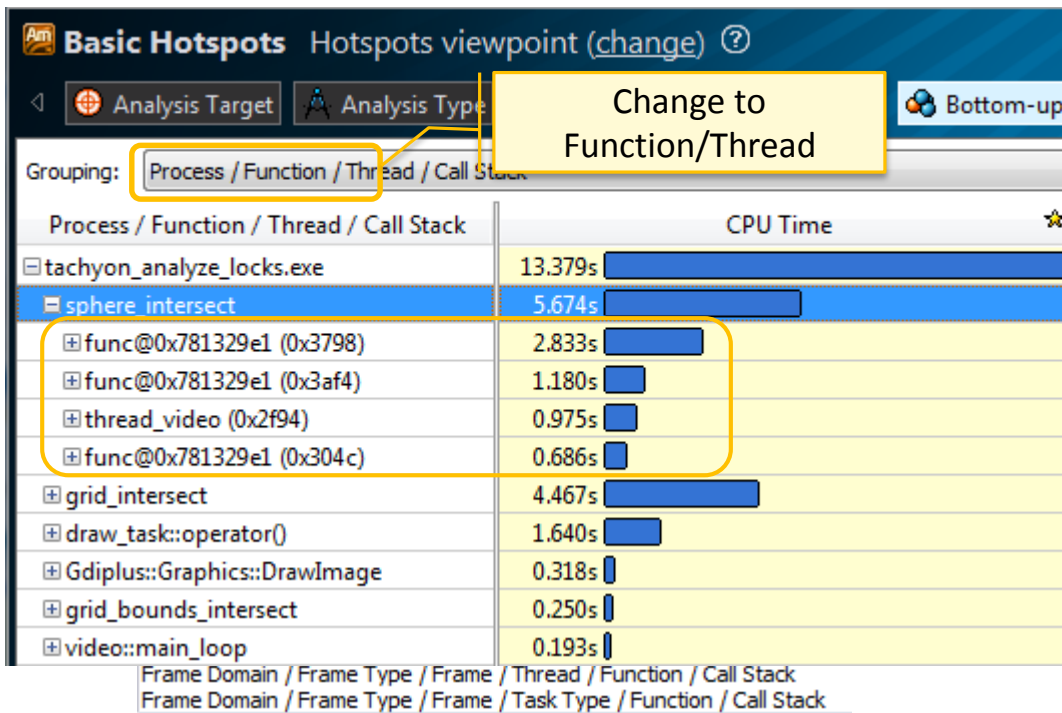
Grouping: Thread / Function / Call Stack

Thread / Function / Call Stack	CPU Time
func@0x781329e1 (0x3798)	5.983s
func@0x781329e1 (0x3af4)	2.598s
thread_video (0x2f94)	2.384s
sphere_intersect	0.975s
grid_intersect	0.975s
intersect_objects	0.957s
shader	0.625s
trace	0.333s



# Viewpoints and Groupings

For example, pre-defined groupings can be used to determine load imbalance



# Key Concepts

## Results Comparison

VTune™ Amplifier XE allows comparison of two similar runs

Extremely useful for:

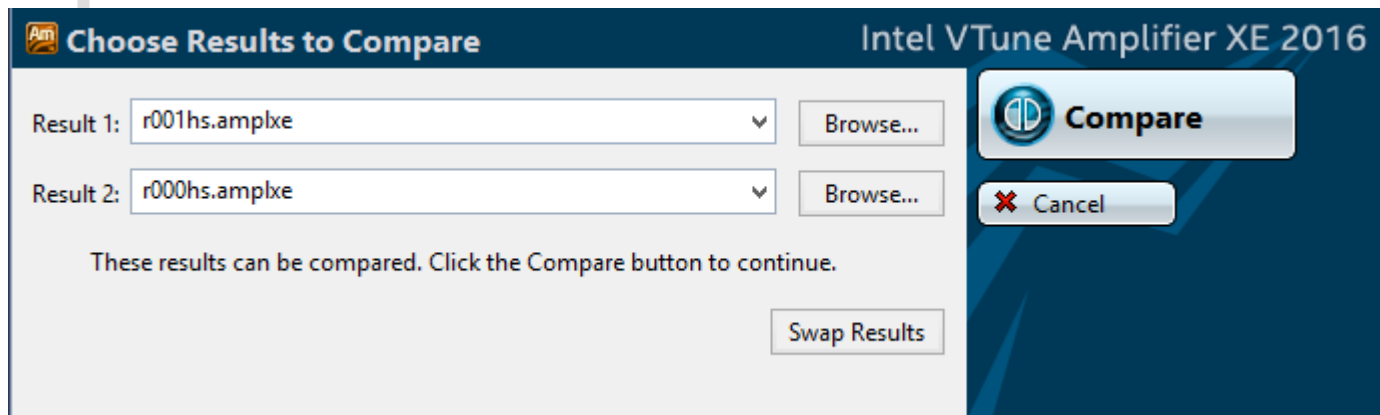
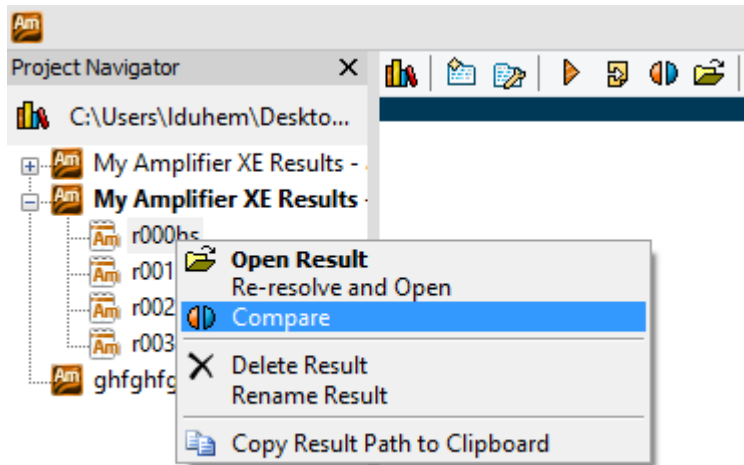
- Benchmarking
- Regression analysis
- Testing

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



# Reminding the 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

# Performance profiling

## 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**

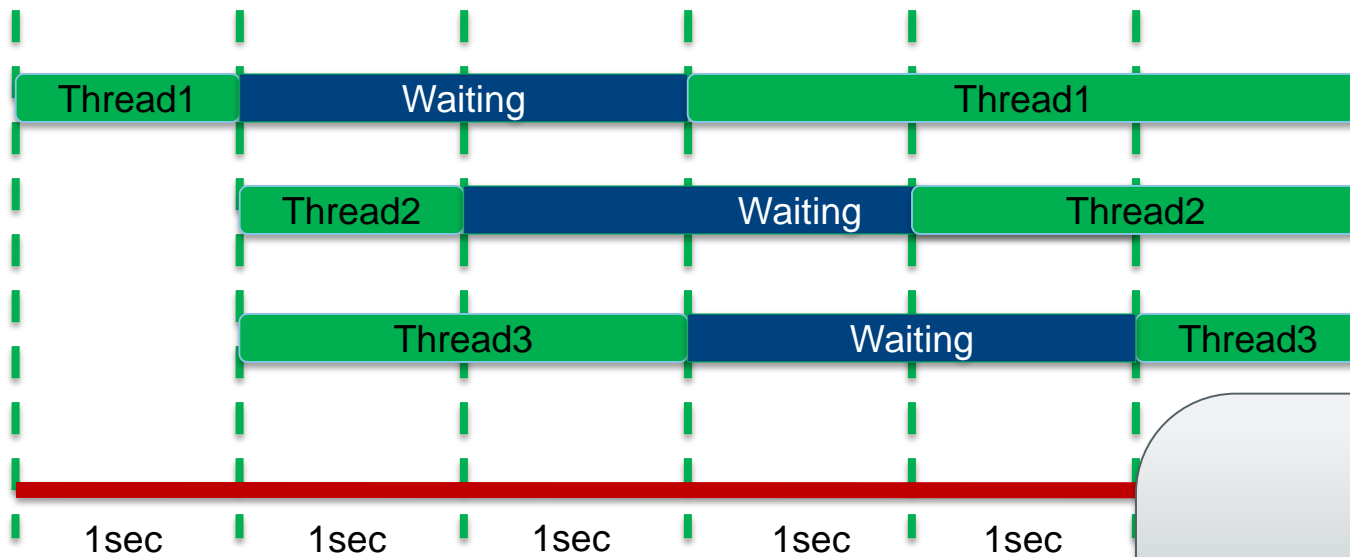
The amount of time a thread spends executing on a logical processor. For multiple threads, the CPU time of the threads is summed.

### **Wait Time**

The amount of time that a given thread waited for some event to occur, such as: synchronization waits and I/O waits

# Performance profiling

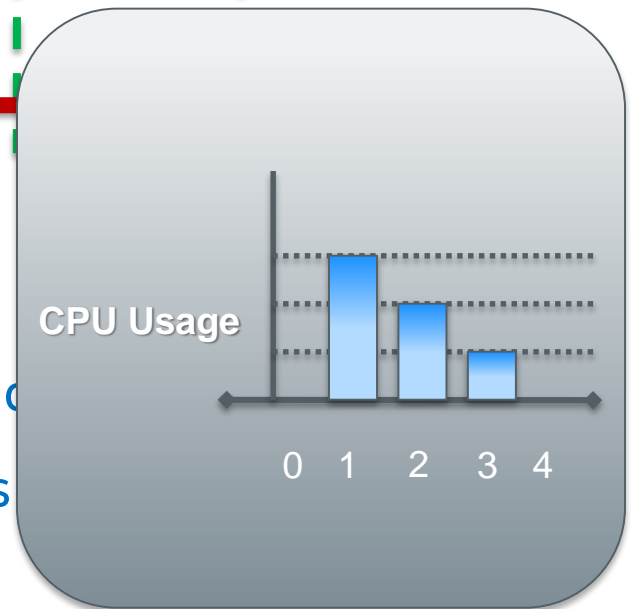
## CPU Usage



**Elapsed Time:** 6 seconds

**CPU Time:**  $T1 (4s) + T2 (3s) + T3 (3s) = 10$  seconds

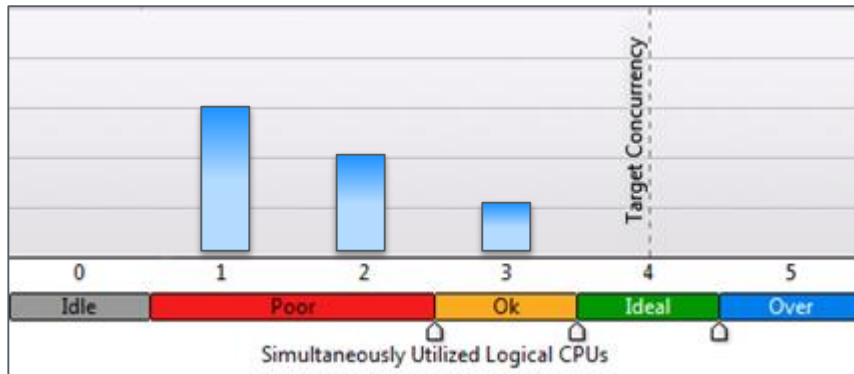
**Wait Time:**  $T1(2s) + T2(2s) + T3 (2s) = 6$  seconds



# CPU Usage

How it's presented by VTune Amplifier

## Summary View: CPU Usage Histogram



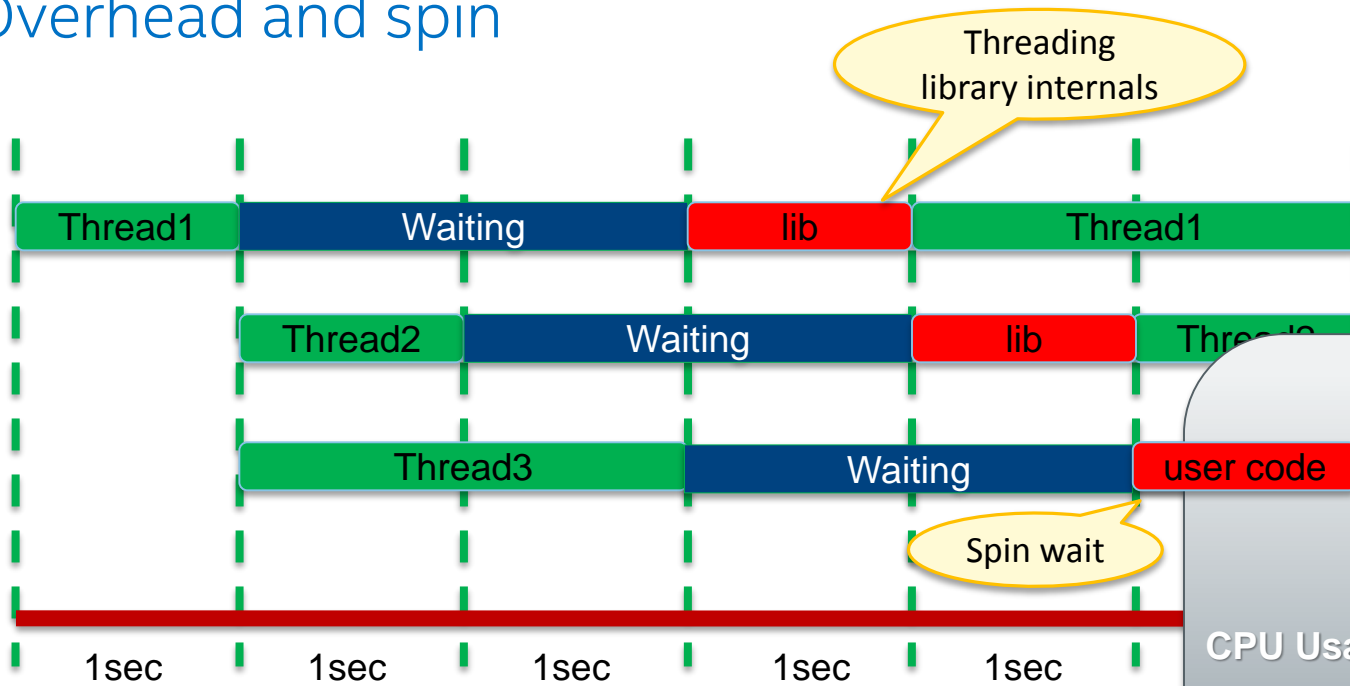
Only CPU Time measured  
Wait Time is not counted  
in Hotspots

## Bottom-Up View: CPU Time

Function	CPU Time	By CPU Utilization
My_Func()	10 s	

# Performance profiling

## Overhead and spin

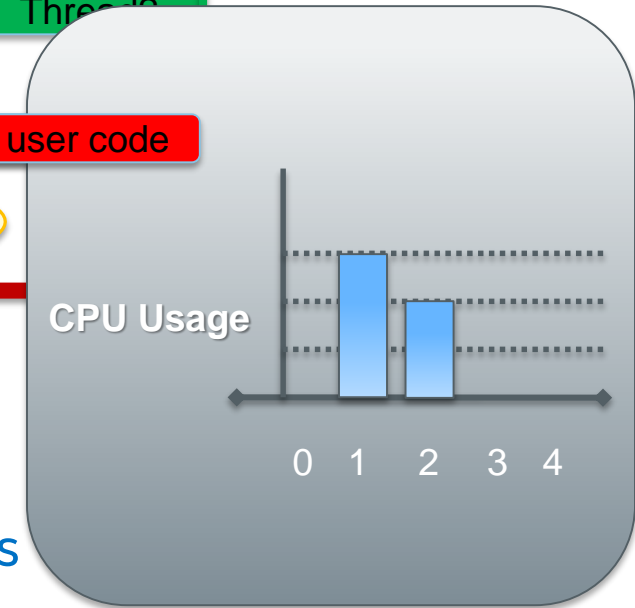


**Elapsed Time:** 6 seconds

**CPU Time:** T1 (4s) + T2 (3s) + T3 (3s) = 10 seconds

**Wait Time:** T1(2s) + T2(2s) + T3 (2s) = 6 seconds

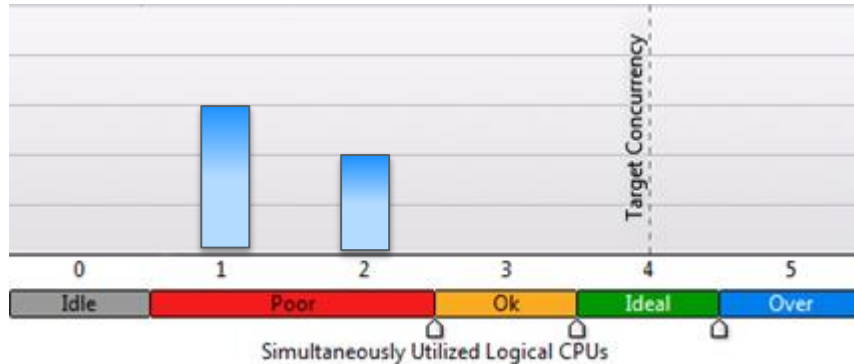
**Overhead and spin Time:** T1(1s) + T2(1s) + T2(1s) = 3 s



# CPU Usage

How it's presented by VTune Amplifier

## Summary View: CPU Usage Histogram



Overhead and Spin Time is not counted for CPU Usage

## Bottom-Up View: CPU Time

Function	CPU Time	By CPU Utilization	Overhead and Spin Time
My_Func()	10 s		4 s

# Hotspots analysis

## Hotspot viewpoint

**Basic Hotspots** Hotspots viewpoint (change) ? Intel VTune Amplifier XE 2016

Analysis Target Analysis Type **Select viewpoint:** Bottom-up Caller/Callee Top-down Tree Platform

Grouping: Function / Call Stack

**Adjust Data Grouping**

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

**Hotspots by CPU Usage**

Function (Full)	Source File	CPU Time
grid_intersect	grid.cpp	34.3% (1.273s of 3.710s)
sphere_intersect	sphere.cpp	
func@0x18003...		
USER32.dll	MsgWaitForM...	
analyze_locks.exe	grid_bounds_i...	
gdi32.dll	GdiDrawlma...	
analyze...	grid.cpp	
shade.cpp	shade.cpp	

**Hotspot Functions**

**Function CPU time**

**Call stack**

Viewing 1 of 22 selected stack(s)

- analyze\_locks.exe!grid\_intersect - grid.cpp
- analyze\_locks.exe!shader+0x39c - shade.cpp:139
- analyze\_locks.exe!render\_line\_pixel+0x163 - analyze\_locks.cpp:101
- analyze\_locks.exe!draw\_task::operator()+0x13f - analyze\_locks.cpp:168
- analyze\_locks.exe!tbb::interface6::intern...d\_range<int> +0x384 - partitioner.h:257
- analyze\_locks.exe!parallel\_for\_on class draw\_task+0x33 - parallel\_for.h:108
- tbb.dll!TBB Dis...loop+0x1b8 - custom\_scheduler.h:441
- tbb.dll!t...state\_root\_with\_context\_proxy::allocate+0x84 - task.cpp:81
- 0x8c - analyze\_locks.cpp:184

**Click [+] for Call Stack**

**Thread timeline**

**Filter by Module & Other Controls**

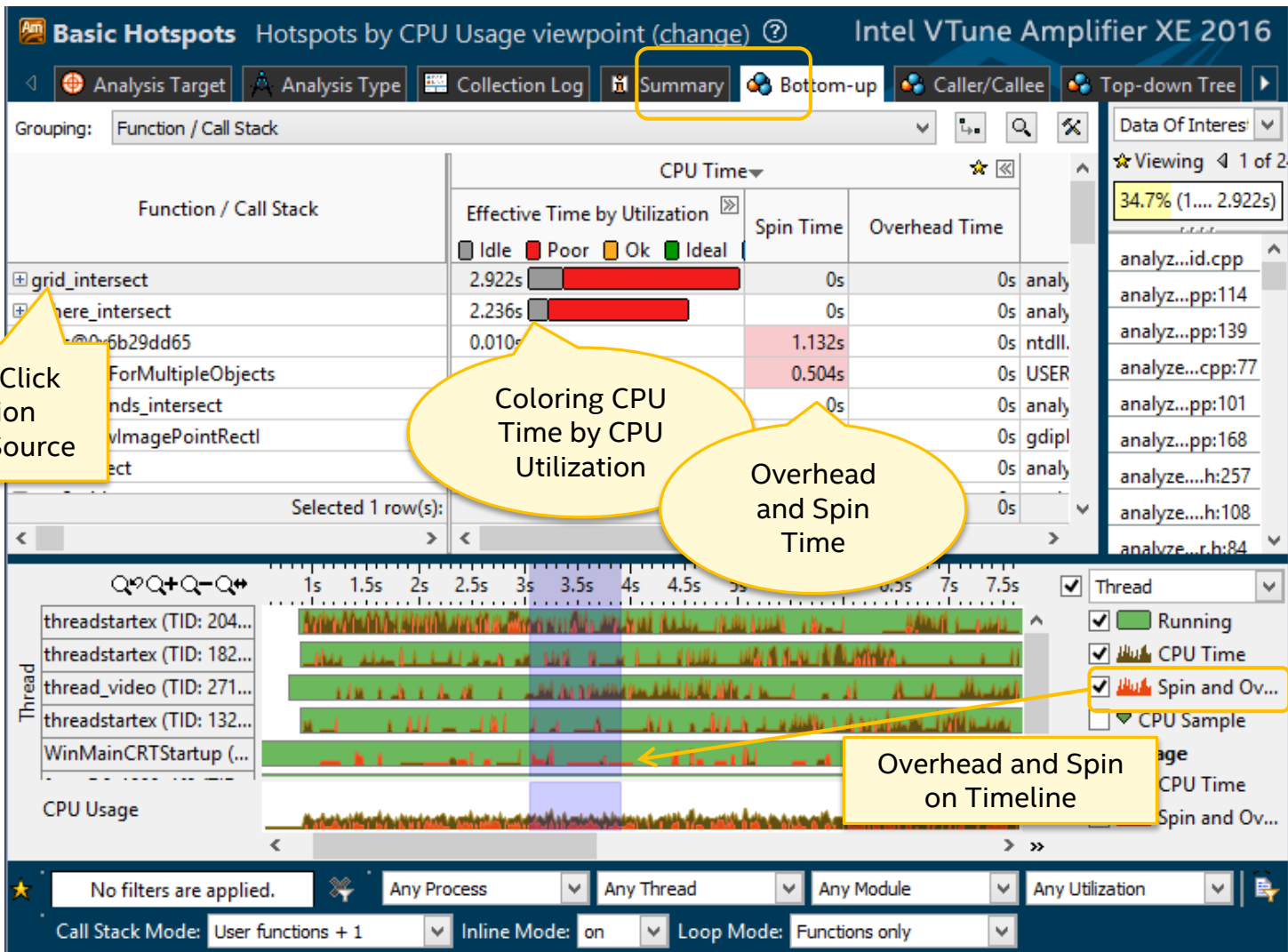
**Filter by Timeline Selection (or by Grid Selection)**

No filters are applied. Any Process Any Thread Any Module User functions + 1 on Functions only



# 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



# Advanced Hotspot analysis

Uses Intel's CPU hardware performance collectors

Higher resolution of sampling (~1 /ms)

Capable for system wide analysis (all processes running in a system)

OS modules and drivers profiling (ring 0 level)

OS context switches and threads synchronization issues

**Choose Analysis Type**

Analysis Type

- Algorithm Analysis
  - Basic Hotspots
  - Advanced Hotspots**
  - Concurrency
- CPU specific Analysis
- SoC Advanced Analysis
- Knights Corner Platform Analysis
- Custom

**Advanced Hotspots** Copy

Identify time-consuming code in your application. Advanced Hotspots analysis (formerly, Lightweight Hotspots) uses the OS kernel support or VTune Amplifier kernel driver to extend the Hotspots analysis by collecting call stacks, context switch and statistical call count data as well as analyzing the CPI (Cycles Per Instruction) metric. By default, this analysis uses higher frequency sampling at lower overhead compared to t...

CPU sampling interval, ms:

Select a level of details provided with event-based sampling collection. Higher collection levels cause higher overhead.

Hotspots

Hotspots, stacks and context switches

Hotspots, call counts, stacks and context switches

**Start**

**Start Paused**

Project Properties

Start the Analysis

Advanced Hotspot Analysis

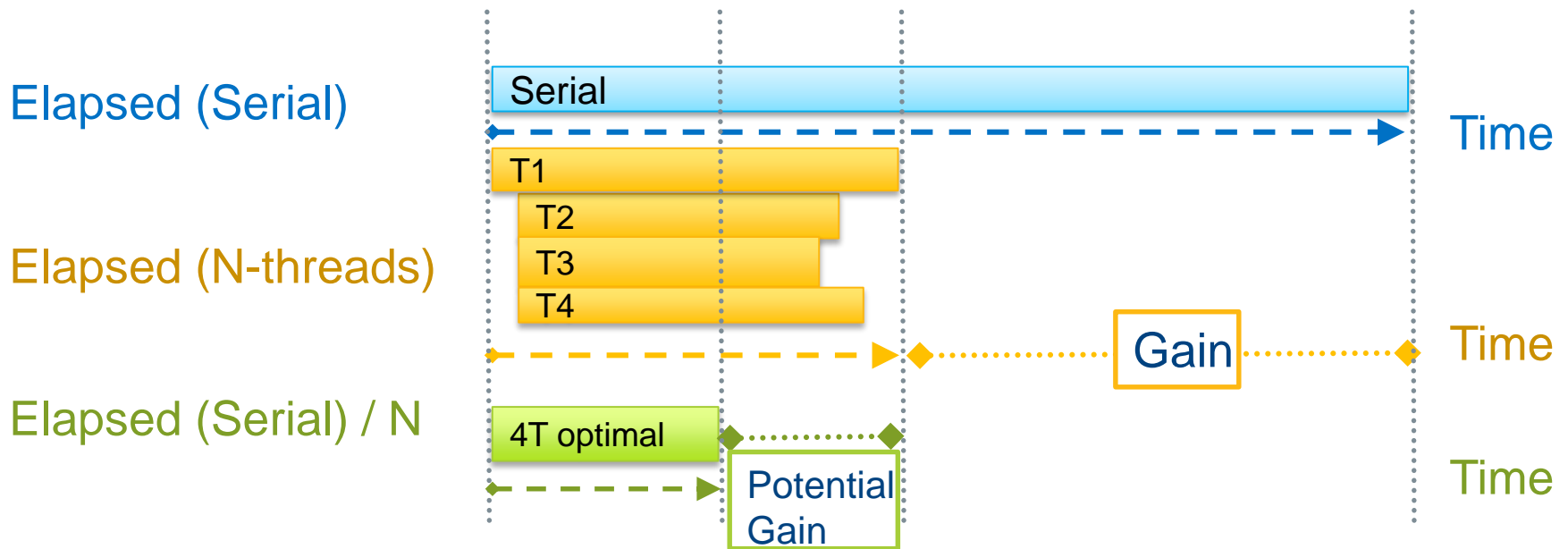
Select level of data collected

Select sampling interval

# 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)

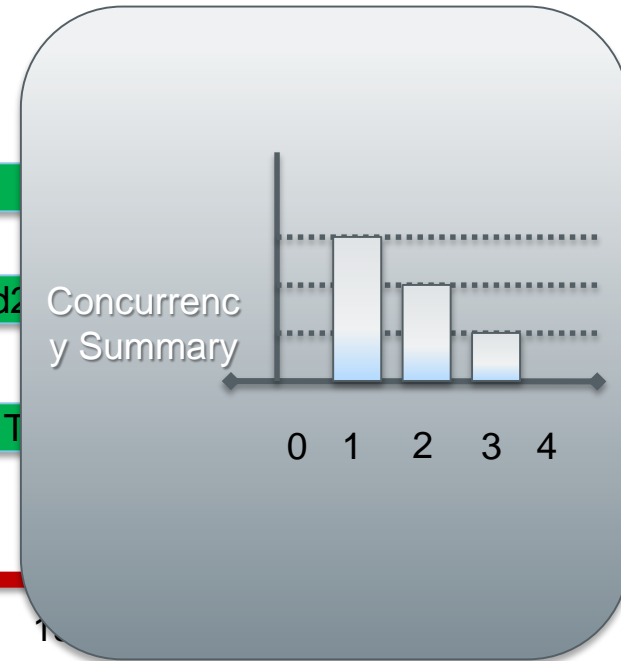
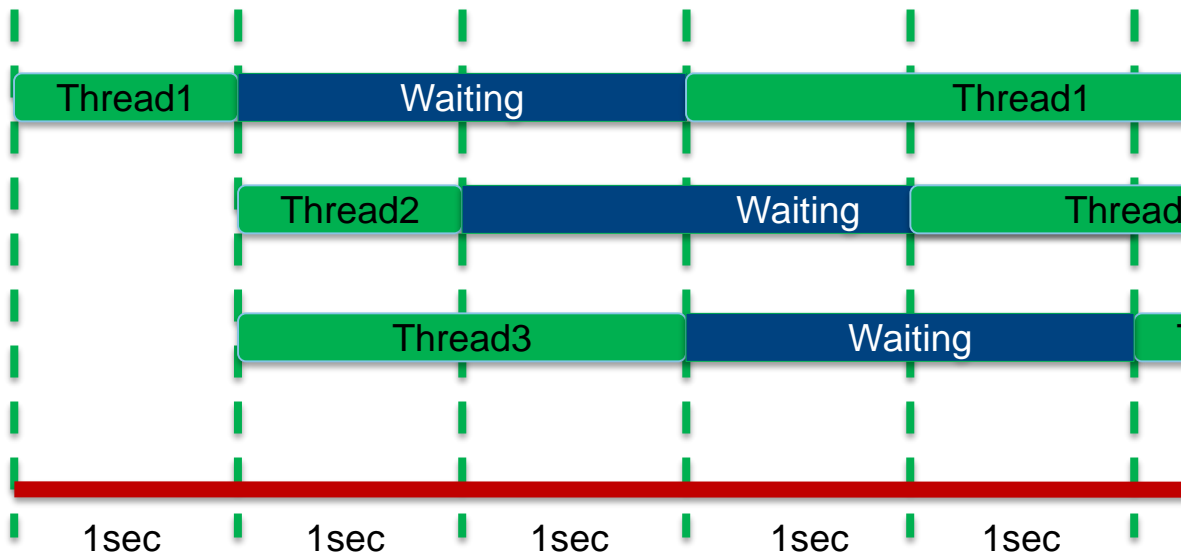


# Performance profiling

## Concurrency

Concurrency -

Is a measurement of the number of active threads



# Parallelism/Concurrency Analysis

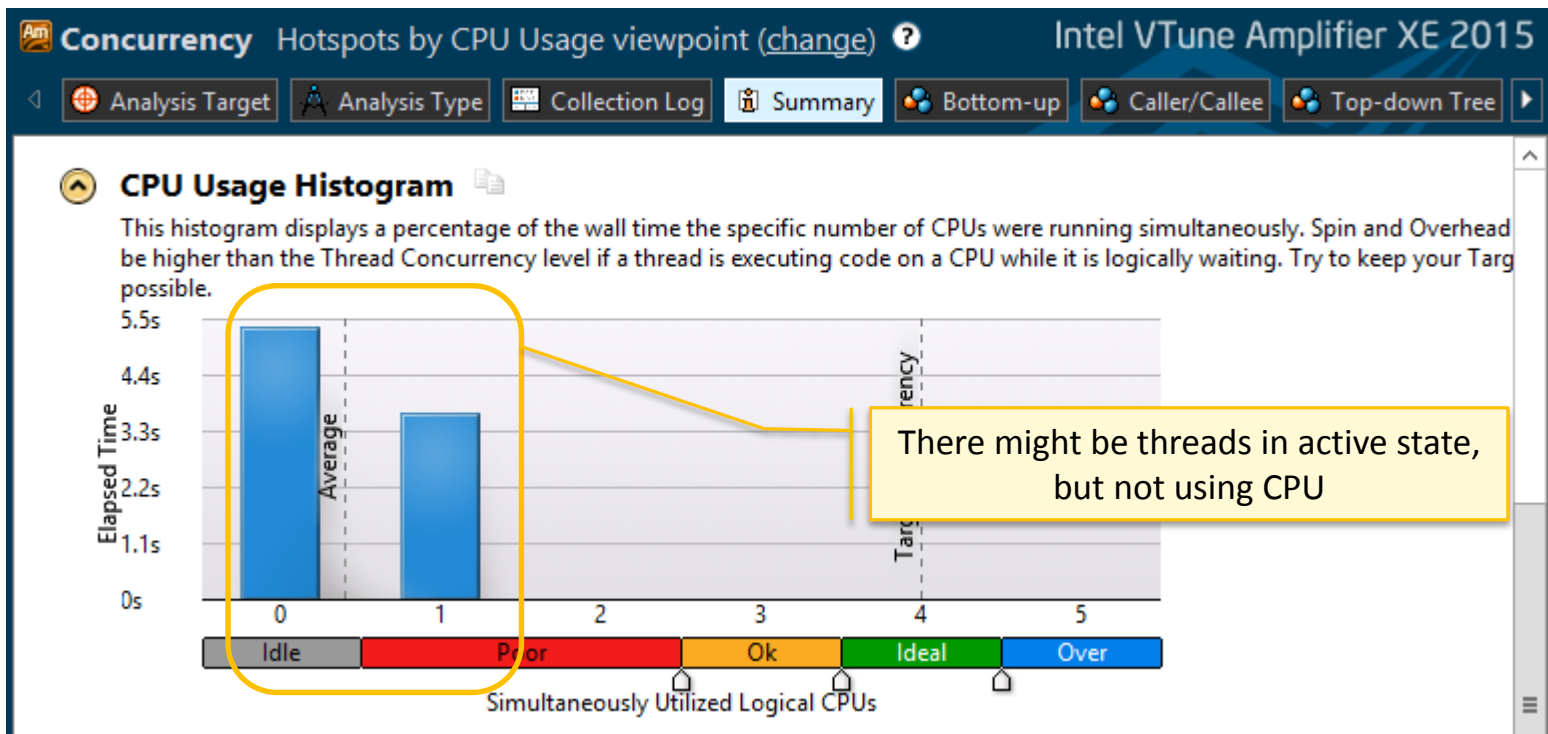
For Parallelism / Concurrency analysis,

- 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)

The screenshot shows the 'Choose Analysis Type' dialog in Intel VTune. The left sidebar lists analysis types, with 'Concurrency' selected. The main panel displays the 'Concurrency' configuration, including a 'Start' button. A yellow callout bubble points to the 'Start' button with the text 'Start the Analysis'. Another yellow callout bubble points to the 'Concurrency' item in the sidebar with the text 'Concurrency Analysis'. The 'Concurrency' configuration includes a 'CPU sampling interval, ms' set to 10, and several checkboxes for analyzing user tasks, Intel runtimes, GPU usage, and processor graphics hardware events.

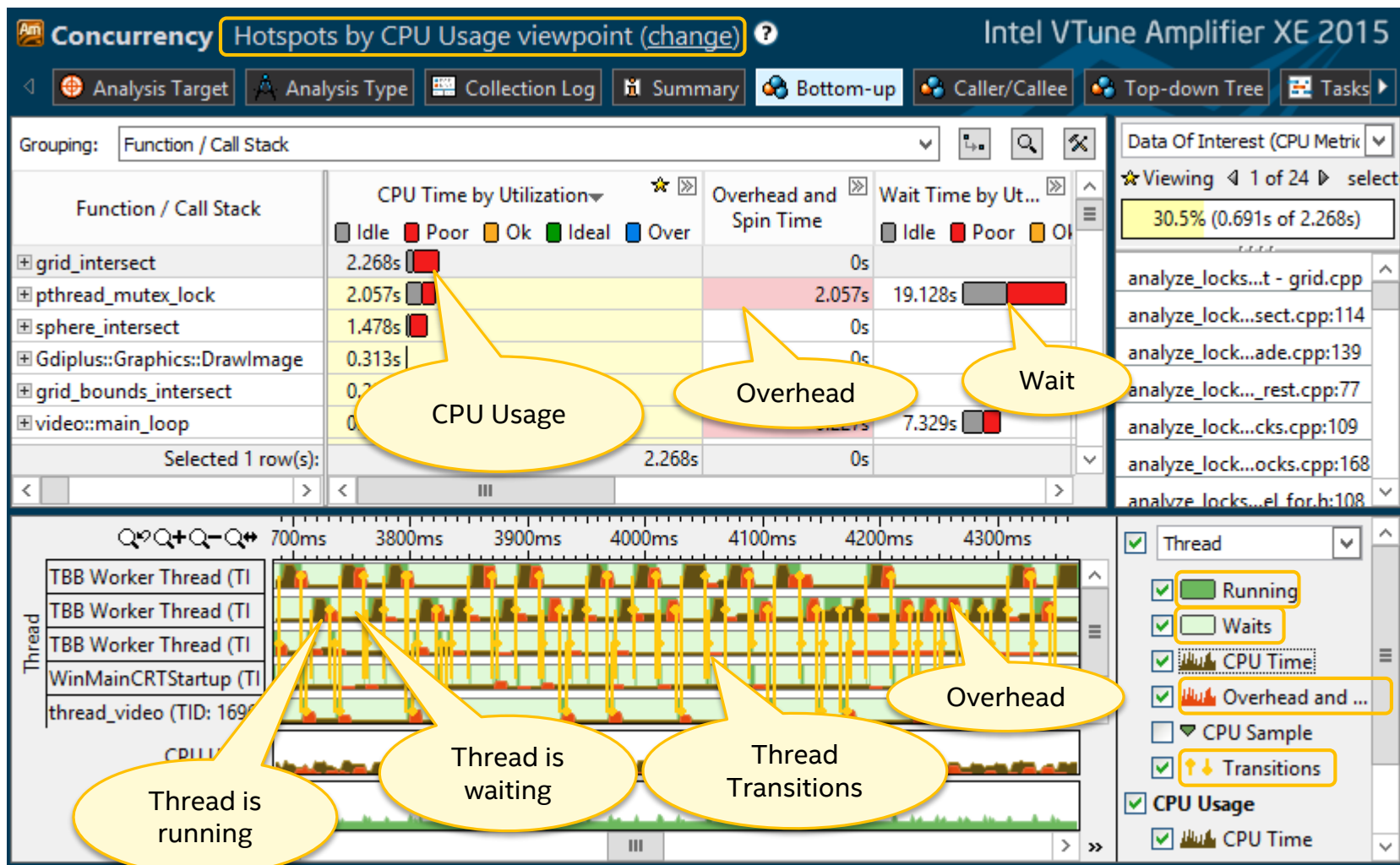
# Concurrency Analysis

## Summary view. CPU Usage Histogram



# Concurrency Analysis

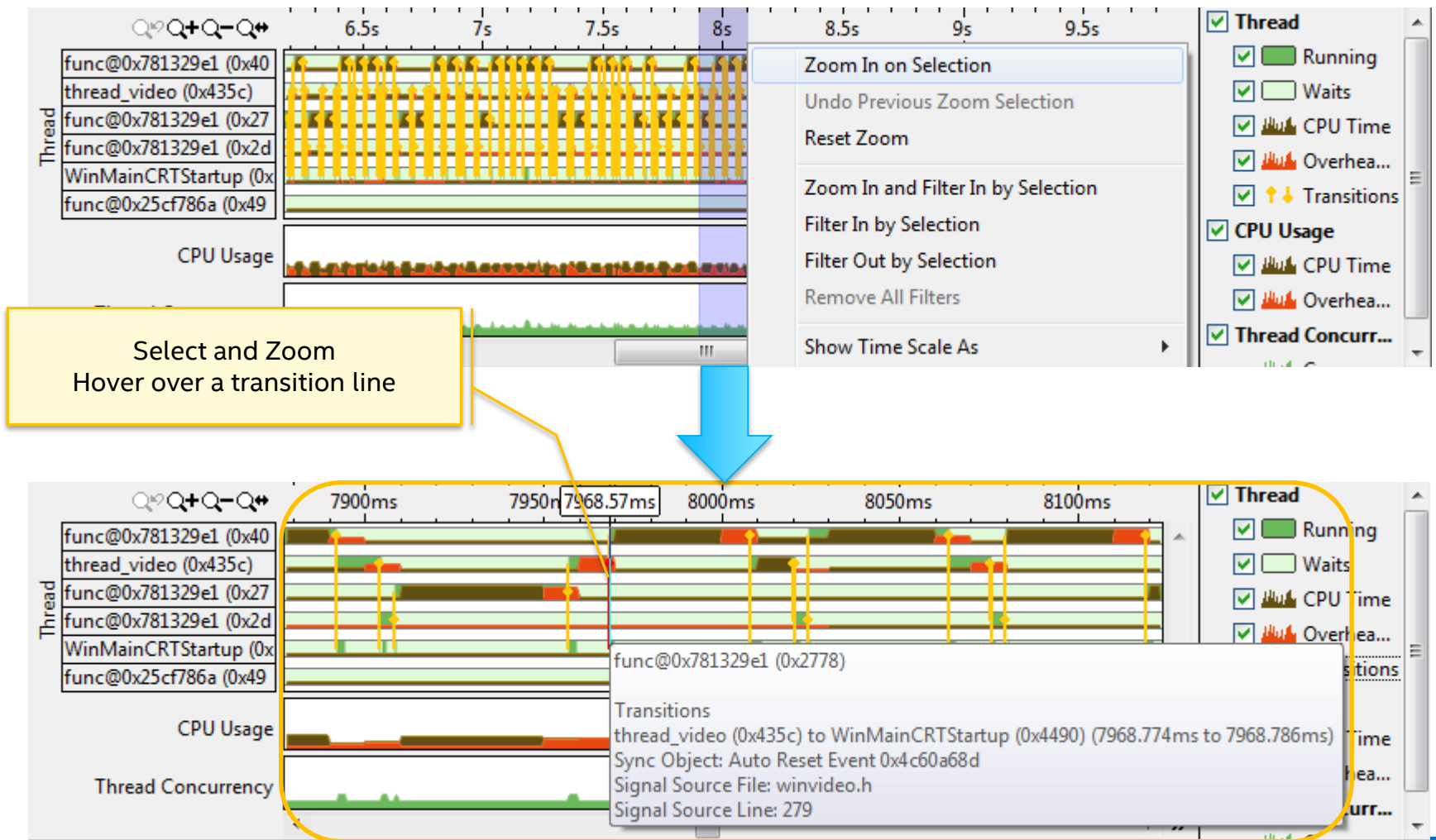
## Bottom-Up view. CPU Usage





# Concurrency Timeline

## Investigate reasons for transitions



# Concurrency Analysis

## Source Code View

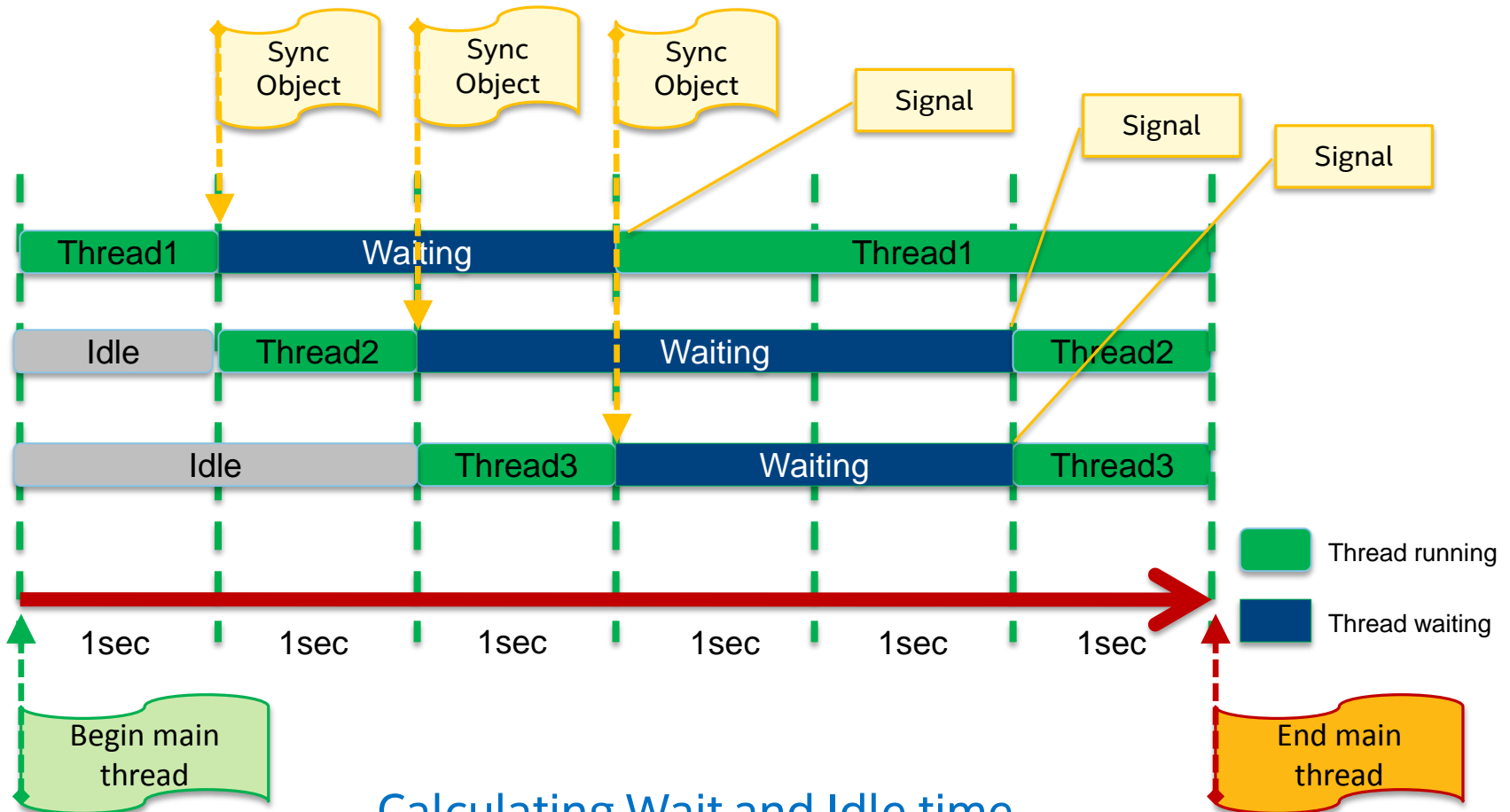
The screenshot displays the Intel VTune Amplifier XE 2015 interface. The title bar reads "Concurrency Hotspots by CPU Usage viewpoint (change) ? Intel VTune Amplifier XE 2015". The menu bar includes "Collection Log", "Summary", "Bottom-up", "Caller/Callee", "Top-down Tree", "Tasks and Frames", and "grid.cpp". The main window shows a source code view of "grid.cpp" with a table of CPU usage data. The table has columns for "So. Li.", "Source", and "CPU Time: Total by Utilization". The CPU time is represented by a bar chart with segments for Idle (grey), Poor (red), Ok (yellow), Ideal (green), and Over (blue). A callout bubble points to the bar for line 581, stating "CPU Usage coloring against source lines".

So. Li.	Source	CPU Time: Total by Utilization
572	tmax.x += tdelta.x;	
573	curpos = nXp;	
574	nXp.x += pdeltaX.x;	
575	nXp.y += pdeltaX.y;	
576	nXp.z += pdeltaX.z;	
577	}	
578	else if (tmax.z < tmax.y) {	9.865ms
579	cur = g->cells[voxindex];	109.361ms
580	while (cur != NULL) {	10.186ms
581	if (ry->mbox[cur->obj->id] != ry->serial) {	741.193ms
582	ry->mbox[cur->obj->id] = ry->serial;	504.642ms
583	cur->obj->methods->intersect(cur->obj, ry);	354.290ms
584	}	
585	cur = cur->next;	196.915ms
586	}	
587	curvox.z += step.z;	6.534ms
588	if (ry->maxdist < tmax.z    curvox.z == out.z)	23.929ms
589	break;	
590	voxindex += step.z*g->xsize*g->ysize;	10.203ms
591	tmax.z += tdelta.z;	16.985ms

Selected 1 row(s): 741.193ms

# Performance profiling

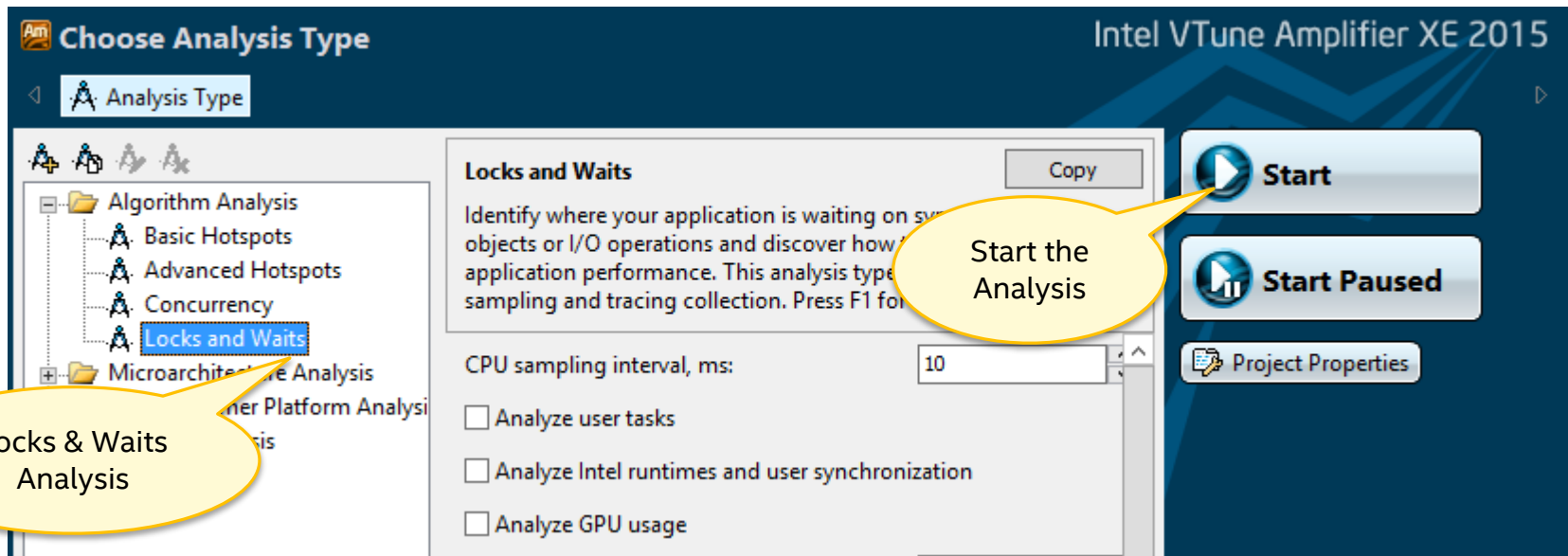
## Waiting on locks



# Locks and Waits Analysis

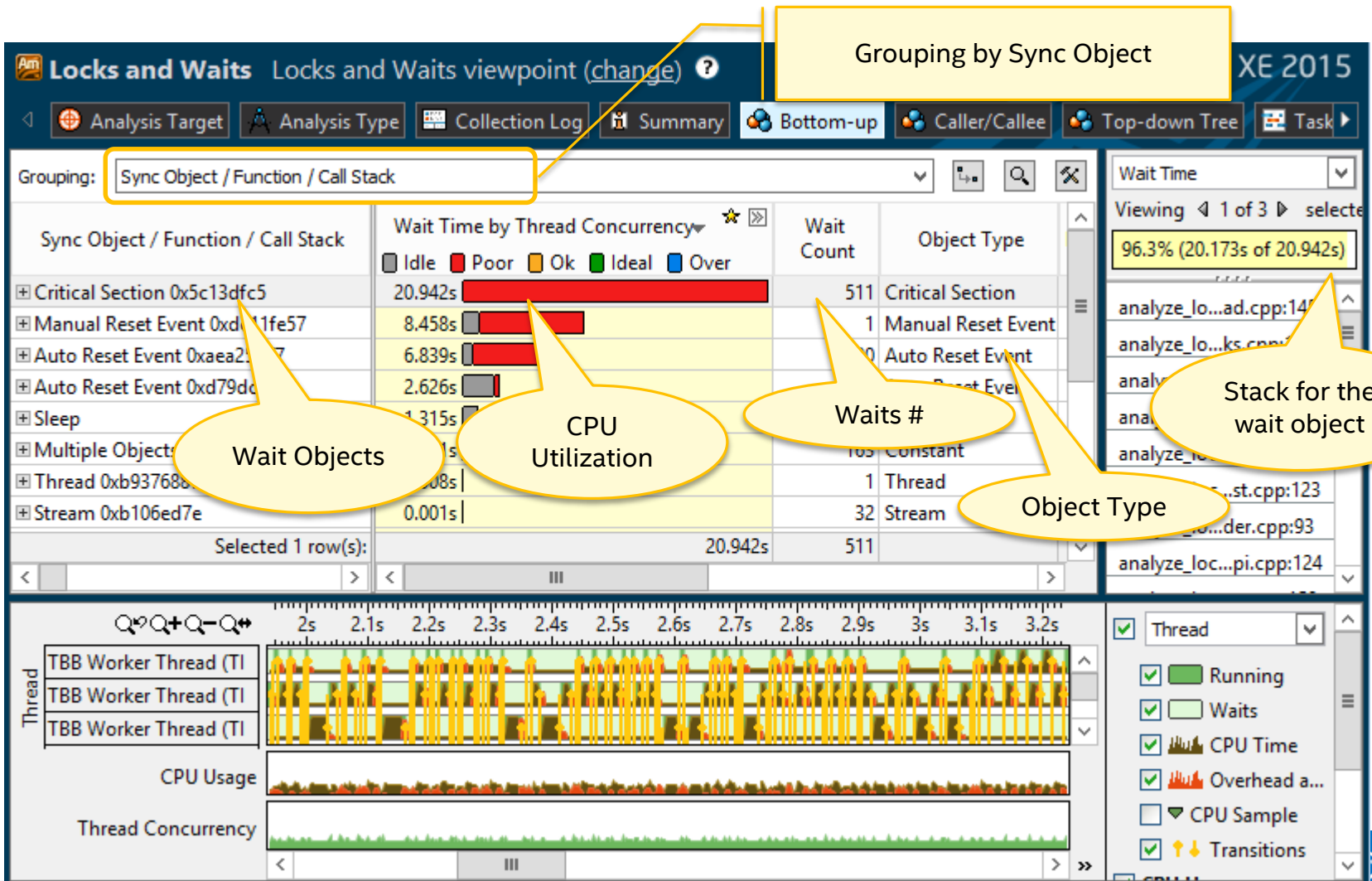
Identifies those threading items that are causing the most thread block time

- Synchronization locks
- Threading APIs
- I/O



# Locks and Waits Analysis

## Sync/Wait objects



# Locks and Waits Analysis

## Source View

The screenshot displays the Intel VTune Amplifier XE 2015 interface for Locks and Waits analysis. The main window is titled "Locks and Waits" and "Locks and Waits viewpoint (change)". The top navigation bar includes "Analysis Target", "Analysis Type", "Collection Log", "Summary", "Bottom-up", "Caller/Callee", "Top-down Tree", and "Task". The "Source" tab is active, showing a list of source code lines. Line 166 is highlighted, corresponding to the function `pthread_mutex_lock (&rgb_mutex);`. The "Wait Time: Tot..." column shows a total wait time of 20.173s, and the "Wait Count: T..." column shows a total wait count of 420. A legend indicates that the red bar represents "Poor" performance. The right-hand pane shows a "Wait Time" summary for the selected stack, indicating that 96.3% (20.173s of 20.942s) of the time is spent waiting. The stack list includes `analyze_locks.ex...pthread.cpp:145`, `analyze_locks.ex...ze_locks.cpp:166` (highlighted), `analyze_locks.ex...parallel_for.h:108`, `analyze_locks.ex...ace_rest.cpp:110`, `analyze_locks.exe...ace_rest.cpp:123`, `analyze_locks.ex... - render.cpp:93`, `analyze_locks.ex...19 - api.cpp:124`, `analyze_locks.ex... - video.cpp:159`, `video.h:190`, and several `[Unknown]` entries.

So. Li.	Source	Wait Time: Tot...	Wait Count: T...
157	<code>unsigned int serial = 1;</code>		
158	<code>unsigned int mboxsize = sizeof(unsigned int);</code>		
159	<code>unsigned int * local_mbox = (unsigned int *) malloc(mboxsize);</code>		
160	<code>memset(local_mbox, 0, mboxsize);</code>		
161			
162	<code>for (int y=r.begin(); y!=r.end(); ++y)</code>		
163	<code>drawing_area drawing(startx, totalx, totaly);</code>		
164			
165	<code>// Acquire mutex to protect pixel color</code>		
166	<code>pthread_mutex_lock (&amp;rgb_mutex);</code>	20.173s	420
167	<code>for (int x = startx; x &lt; stopx; x++)</code>		
168	<code>draw_area_t c = render_one_pixel (x, y);</code>		
169	<code>draw_area_t output_pixel (c);</code>		
170			
171			

Annotations:

- Upper function (pointing to `pthread_mutex_lock (&rgb_mutex);`)
- Waiting time on the object (pointing to 20.173s)
- Wait count (pointing to 420)
- Critical Section object upper on the stack (pointing to the stack list)

# Intel® VTune™ Amplifier XE

## User APIs

### User APIs

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

# User 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



# User API

## Collection control and threads naming

### 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.

# User API

## Collection Control Example

```
int main(int argc, char* argv[])
{
    doSomeInitializationWork();

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

    doSomeFinalizationWork();
    return 0;
}
```

# Command Line Interface

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

# Command Line Interface

## Examples

Display a list of available analysis types and preset configuration levels

```
amplxe-cl -collect-list
```

Run Hot Spot analysis on target *myApp* and store result in default-named directory, such as *r000hs*

```
amplxe-cl -c hotspots -- myApp
```

Run the Cuncurrency analysis, store the result in directory *r001par*

```
amplxe-cl -c concurrency -result-dir r001par -- myApp
```

# Command Line Interface

## Reporting

```
$> ampxe-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
```

# Command Line Interface

## Gropof-like output

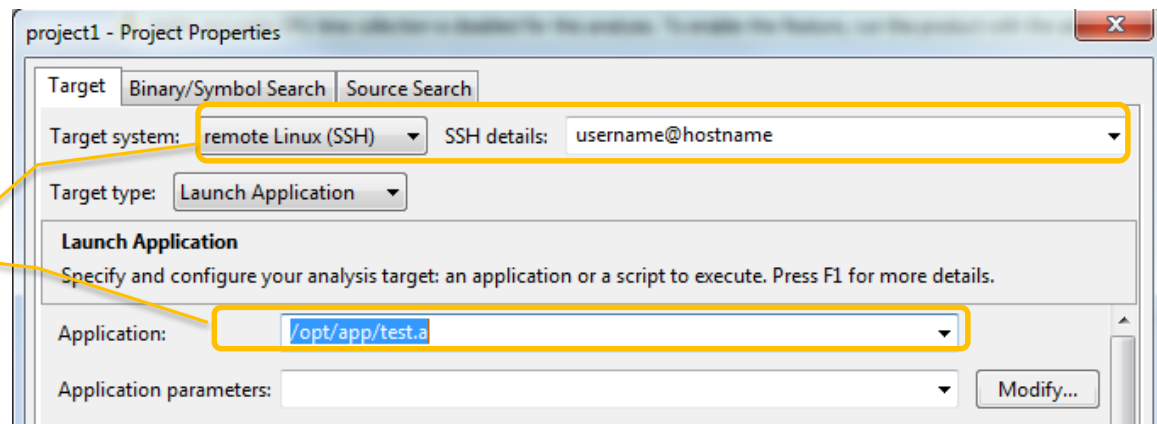
```
[levent@hlnasnb AXE_lab3]$ ampxe-cl -report gprof-cc -r /home/levent/examples/cern/labs/AXE_lab3/r003cc
Using result path ~/home/levent/examples/cern/labs/AXE_lab3/r003cc'
Executing actions 50 % Generating a report
```

Index	% Wait	Time:Total	Wait Time:Self	Children	Name	Index
[0]	99.88	190.104	190.104	0.0	G4RunManager::BeamOn ParRunManager::DoEventLoop	[23] [0]
[1]	0.1	0.162	0.162	0.001	operator<< G4RunManagerKernel::G4RunManagerKernel RunAction::EndOfRunAction	[17] [11] [30]
[2]	83.08	0.001	0.001	0.001	G4strstreambuf::sync G4MycoutDestination::ReceiveG4cout	[1] [5]
[3]	0.0	0.033	0.033	158.141	func@0x416c28 main G4_main	[7] [2] [18]
[4]	0.0	0.002	0.002	0.0	CLHEP::HepRandom::showEngineStatus CLHEP::RanecuEngine::showStatus	[22] [3]
[5]	0.0	0.001	0.001	0.001	G4_main G4MycoutDestination::G4MycoutDestination	[18] [4]
[6]	0.0	0.001	0.001	0.0	G4strstreambuf::sync G4MycoutDestination::ReceiveG4cout	[1] [5]
[7]	0.0	0	0	0	G4UImanager::ExecuteMacroFile <cycle 1> G4UIbatch::G4UIbatch	[28] [6]
[8]	99.88	0.0	0.033	158.141	func@0x416c28 main G4_main <cycle 1 as a whole>	[7] [2] [18] [8]

# Remote Data Collection



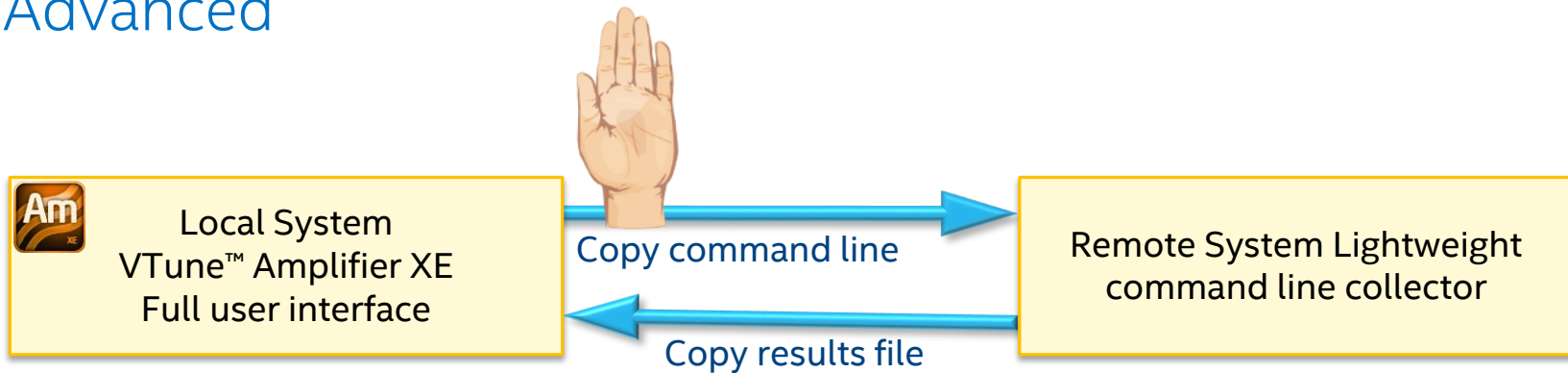
1. Setup the experiment using GUI locally
2. Configure remote target connection\*
3. Specify application to run
4. Run analysis and get results copied to the Host automatically.



\*Need to establish a passwordless ssh-connection

# Remote Data Collection

## Advanced



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?



# 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

