

Performance Tuning Methodology (Intel® VTune™ Amplifier XE)

Dr.-Ing. Michael Klemm
Software and Services Group
Intel Corporation
(michael.klemm@intel.com)

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.

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

Copyright © 2013, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, Phi, VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries. *Other names and brands may be claimed as the property of others.

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

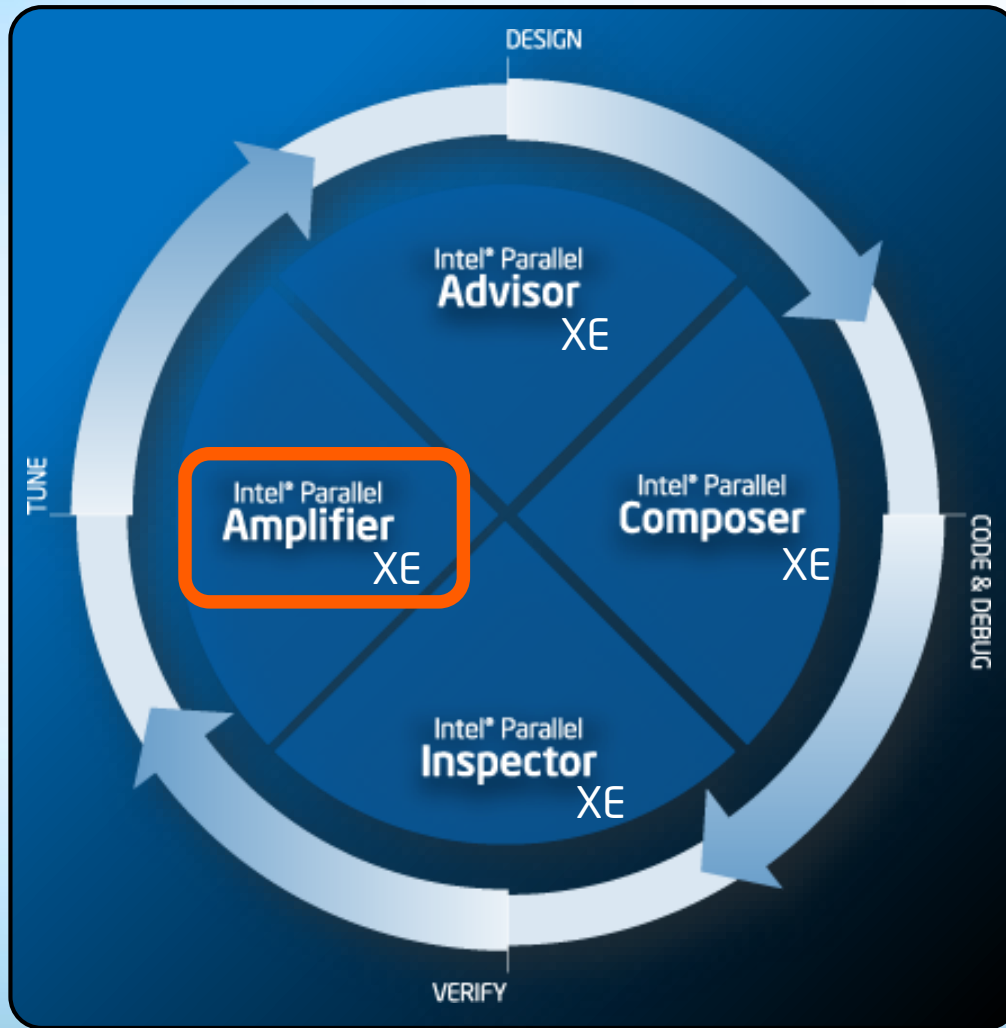
Agenda

- Performance Tuning Methodology
- Intel® VTune™ Amplifier XE: User Interface
- Fundamental Analysis: Hotspots
- Finding Issues in Parallel Applications
- Using the Performance Monitoring Unit

Agenda

- Performance Tuning Methodology
- Intel® VTune™ Amplifier XE: User Interface
- Fundamental Analysis: Hotspots
- Finding Issues in Parallel Applications
- Using the Performance Monitoring Unit

Intel® Parallel Studio XE



Holistic toolset for the parallel software development lifecycle

- DESIGN
- CODE & DEBUG
- VERIFY
- TUNE

Intel® Cluster Studio XE adds:

- Intel® MPI
- Intel® Trace Analyzer and Collector





The Software Optimization Process

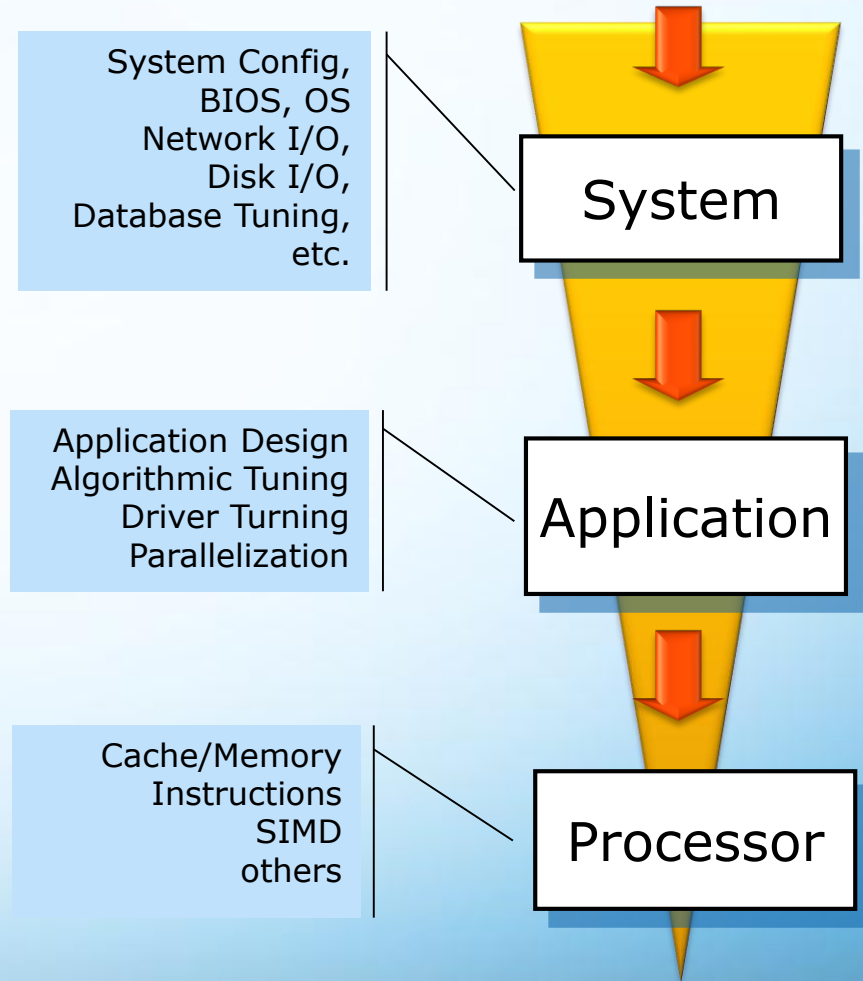
- The process of improving the software by eliminating bottlenecks so that it operates more efficiently on a given hardware and uses resources optimally
 - Identifying bottlenecks in the target application and eliminating them appropriately is the key to an efficient optimization
- There are many optimization methodologies, which help developers answer the questions of
 - Why to optimize?
 - What to optimize?
 - To what to optimize?
 - How to optimize?

These methods aid developers to reach their performance requirements.

Performance Analysis Methodology

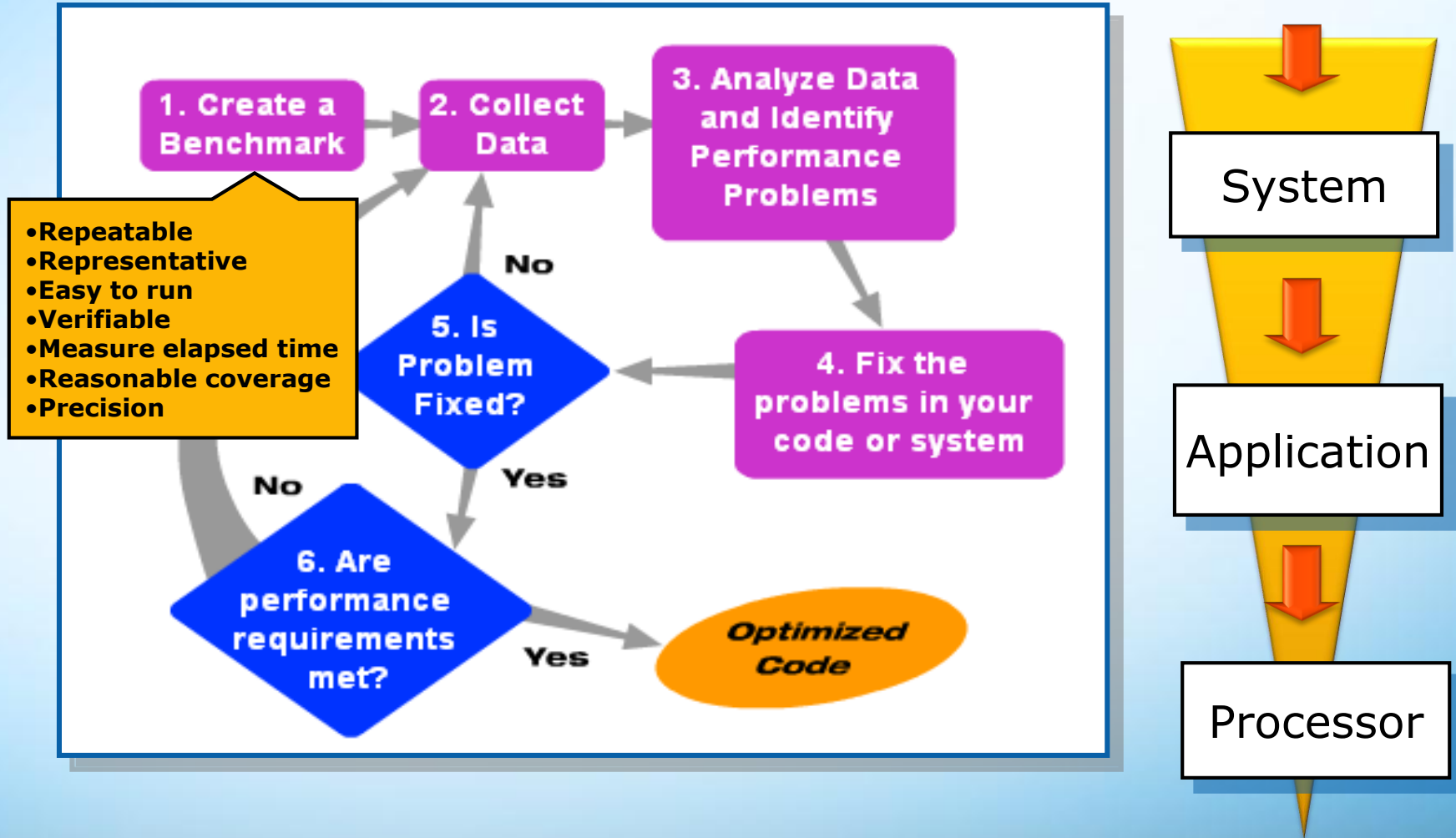
Optimization: A Top-down Approach

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



Performance Analysis Methodology

Optimization: A Top-down Approach



When to Stop

- Is architecture at maximum efficiency?
 - What this means: calculating theoretical maximum.
 - Know about best values for CPI or IPC.
 - Know the maximum FLOPS for the data type used.
- Is the performance requirement fulfilled?
 - What are the performance requirements?
 - Incrementally complete optimizations until done.
 - Key question: Are you “happy” with it?

CPI: Cycles per Instructions

IPC: Instructions per Cycle

FLOPS: Floating-Point Oper. Per Sec.

Questions to Ask Yourself

*"We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil.**"*

— Donald Knuth

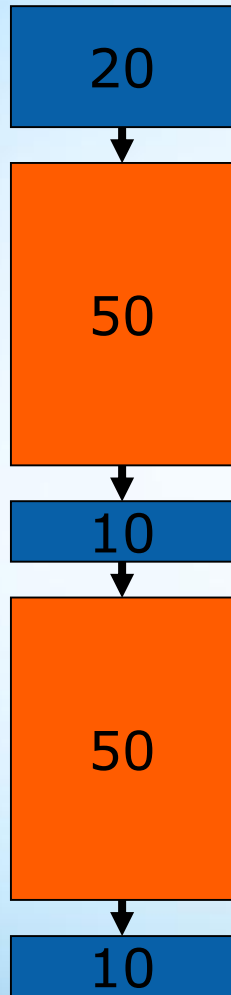
Quality code is:

- Portable
- Readable
- Maintainable
- Reliable

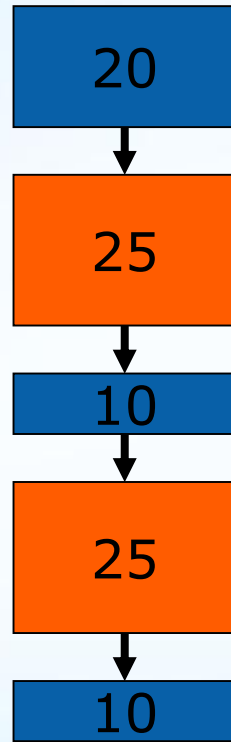
Intelligently sacrifice quality for performance

Amdahl's Law Slightly Reinterpreted

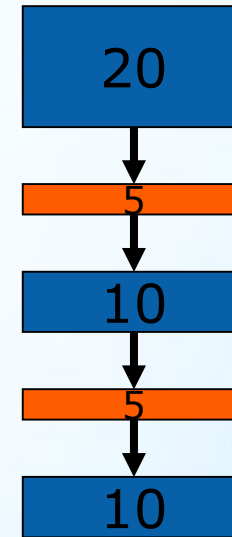
1 core, t=140



1 core, optimized



1 cores, optimized even more



Hotspot

A 2x improvement in the hotspots overall leads to 1.5x

A 10x improvement in the hotspots leads to 2.8x

Agenda

- Performance Tuning Methodology
- **Intel® VTune™ Amplifier XE: User Interface**
- Fundamental Analysis: Hotspots
- Finding Issues in Parallel Applications
- Using the Performance Monitoring Unit

Selecting type of data collection

The screenshot shows the Intel VTune Amplifier XE interface within Microsoft Visual Studio. The main window is titled "Choose Analysis Type" and displays a tree view of analysis types on the left, a detailed description of the selected "Weight Hotspots" type in the center, and control buttons on the right. A red box highlights the "Run" button in the Visual Studio toolbar. A red circle highlights the "Start" button, and another red circle highlights the "Start Paused" button. A red circle highlights the "New ..." button, which has a context menu open showing options like "Copy from current", "New CPU Event Collector analysis", and "New Stack-sampling Collector analysis".

All available analysis types

Different ways to start the analysis

Helps creating new analysis types

Copy correct command line syntax to clipboard

Weight Hotspots
Identify your most time-consuming source code. Unlike Hotspots, Lightweight Hotspots does not collect stack information but can be used system-wide. This analysis type is based on the hardware event-based sampling collection. Press F1 for more details.

Events configured for CPU: 45nm Intel(R) Core(TM) microarchi

After	Event
100000	Core cycles when
20000000	Reference cycles
20000000	Instructions retire

Copy from current
New CPU Event Collector analysis
New Stack-sampling Collector analysis

VTune™ Amplifier XE

GUI Layout

Adjust Data Grouping

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

Click [+] for Call Stack

Double Click Function to View Source

Filter by Timeline Selection (or by Grid Selection)

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

Filter by Module & Other Controls

Hotspots - View hotspots colored by CPU usage

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Top-down Tr

Function - Call Stack

CPU Time

Legend: Idle, Poor, Ok, Ideal, Over

Function	CPU Time	Module
+ dllStopPlugin	7.550s	RenderSystem_Direct3D9.DLL
+ FireObject::checkColli	6.389s	SystemProceduralFire.DLL
+ FireObject::ProcessFire	4.592s	SystemProceduralFire.DLL
+ BaseThreadInitThunk	2.566s	kernel32.dll
+ Ogre::FileStreamDataS	2.562s	OgreMain.dll
+ TaskManagerTBB::Par	2.533s	Smoke.exe
+ AIScene::GetPOI	1.710s	SystemAI.DLL
+ TaskManagerTBB::Wa	1.682s	Smoke.exe
Selected 189 row(s):		47.481s

Timeline View:

- Thread: wWinMainCRTStart, Thread (0x1a28), Thread (0x8e4), Thread (0x29c8)
- CPU Usage
- Frames over Time

Ruler Area:

- Global Mark
- Frame
- Thread
 - Running
 - CPU Time
 - User Task
- CPU Usage
 - CPU Time
- Frames over Ti...

Bottom Controls:

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

VTune™ Amplifier XE

GUI Layout

Hotspots - View CPU time hotspots and stacks Intel VTune Amplifier XE 2011

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

Line	Source	CPU Time	Address	Assembly	CPU Time
469	FireObject::checkCollision(V3 pos, V3 pre	0.476s	0x388c	fld st0, dword ptr [esp+0xc]	0.004s
472	#define FMax std::max<float>	0.561s	0x3890	fld st0, st0	0.993s
	- prev	6.846s	0x3892	fmulp st2, st0	0.787s
	es	3.593s	0x3894	fxch st0, st1	1.465s
477	float param2 = (AABB.zMax - prevPos.	0.830s	0x3896	fstp dword ptr [esp+0x8], st0	0.325s
478	bool neg = (rz < 0.f);	0.615s	0x389a	fld st0, dword ptr [esp+0x40]	0.014s
479	minP = FMax(neg? param2 : param1, mi	3.008s	0x389e	fsubrp st2, st0	0.010s
480	maxP = FMin(neg? param1 : param2, ma	1.875s	0x38a0	fld st0, st0	0.010s
481	if(maxP > minP) {	0.972s	0x38a2	fmulp st2, st0	0.233s
482	rx = 1.f/(pos.x - prevPos.x);	0.252s	0x38a4	fxch st0, st1	0.247s
483	param1 = (AABB.xMin - prevPos.x)	0.264s	0x38a6	fstp dword ptr [esp+0xc], st0	0.326s
	Pos.x)	0.040s	0x38aa	fcomp st0, st2	0.032s
	param1	0.047s	0x38ac	fnstsw stx	
	param2	0.274s	0x38ae	test ah, 0x5	
		0.164s	0x38b1	jp 0x100038c1	
		0.612s	0x38b3	Block 2:	
		0.830s	0x38b3	mov dl, 0x1	0.000s
			0x38b5	lea ecx, ptr [esp+0xc]	0.024s
			0x38b9	jmp 0x100038c1 <Block 4>	
			0x38bb	Block 3:	
			0x38bb	xor edi, dl	0.159s

Selected 1 row(s): 0.830s | Highlighted 6 row(s): 0.830s

Intel® VTune™ Amplifier XE

Time on Source / Asm

Quick Asm navigation:
Select source to highlight Asm

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

Right click for instruction
reference manual

Click jump to scroll Asm



VTune™ Amplifier XE


GUI Layout

The screenshot displays the VTune Amplifier XE GUI layout. The main window is divided into several panels:

- Transitions Locks & Waits:** Shows a timeline of threads (wWinMainCRTStartu..., Thread (0x1364), Thread (0x136c), Thread (0x1374), Thread (0x137c), Thread (0x1384)) and their states (Running, Waits, User Task, Transition). A ruler area on the right allows filtering by Frame, Thread, Thread Concurrency, and Frames over Time.
- CPU Time Hotspots:** Shows a timeline of CPU usage across different threads, with a ruler area for time selection.
- Lightweight Hotspots:** Shows a timeline of lightweight hotspots across different threads, with a ruler area for time selection.

Three yellow arrows point from the main timeline to three hover boxes:

- Frame:**
 - Frame
 - Start: 29.858s Duration: 0.017s
 - Frame: 72
 - Frame Domain: Smoke::Framework::execute()
 - Frame Type: Good
 - Frame Rate: 59.8242179
- Transition:**
 - 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:**
 - 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
- Optional: Add a mark during collection 

Agenda

- Performance Tuning Methodology
- Intel® VTune™ Amplifier XE: User Interface
- **Fundamental Analysis: Hotspots**
- Finding Issues in Parallel Applications
- Using the Performance Monitoring Unit

Readying Your Application for Intel VTune Amplifier XE

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

Analysis Types

Hotspots

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

User-mode Sampling and Tracing Settings

CPU sampling interval, ms:	10	▲ ▼
Collect CPU sampling data:	With stacks	▼
Collect signalling API data:	No	▼
Collect synchronization API data:	No	▼
Collect I/O API data:	No	▼
<input checked="" type="checkbox"/> Collect timeline data		

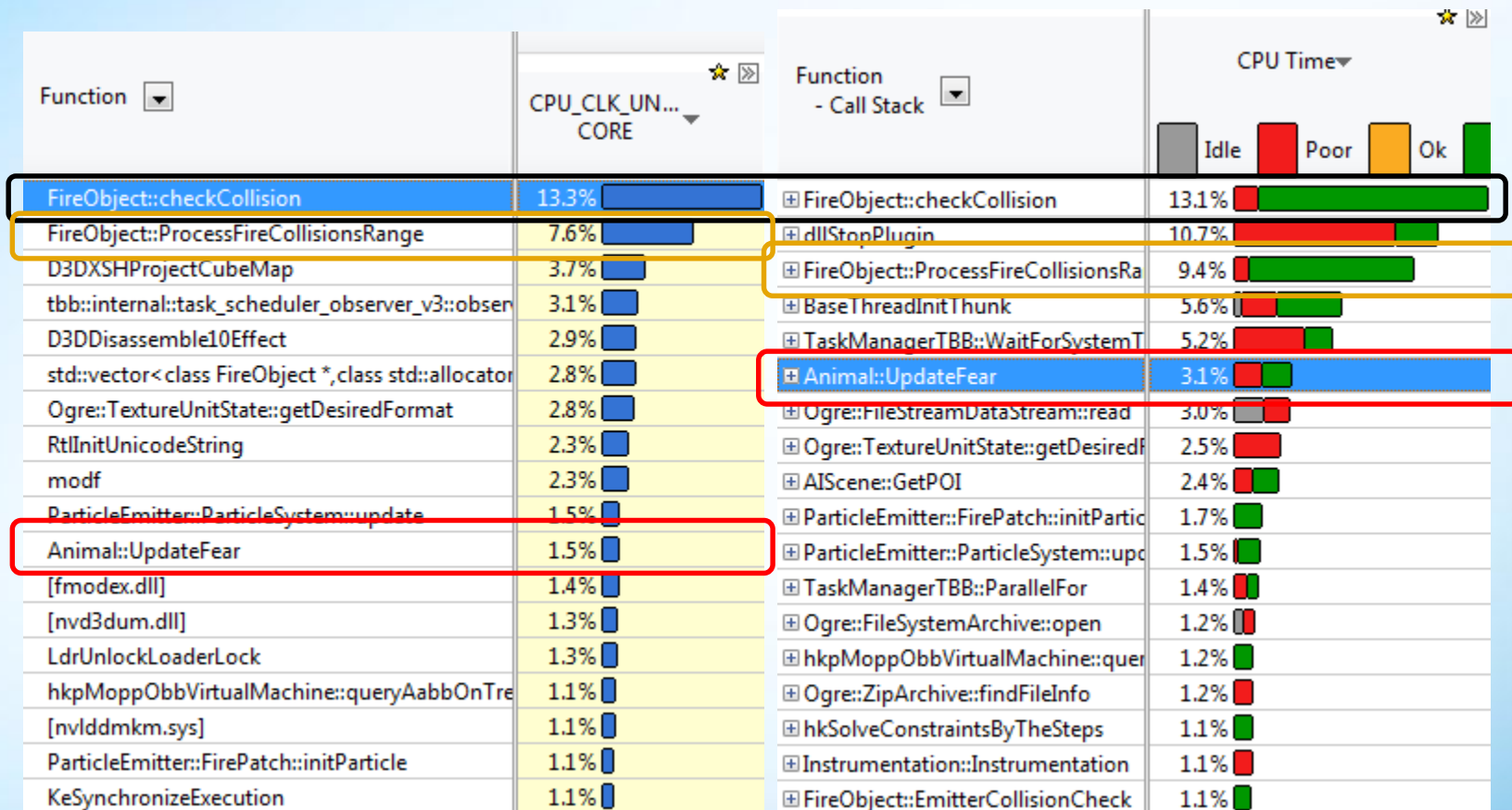
Analysis Types

Lightweight Hotspots

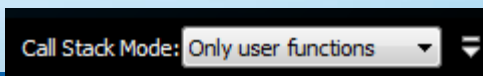
- Similar to Hotspot Analysis
 - Sampling is performed with the SEP collector
 - Driver is required
- Stack walking is not performed
 - Only hotspots are reported
- Samples are taken more frequently, but may have less accurate timing information
- Analysis may be performed for a single application or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

Lightweight Hotspots vs. Hotspots

Smoke



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



Lightweight Hotspots vs. Hotspots

Smoke

Function	CPU_CLK_UN... CORE	Function - Call Stack	CPU Time
FireObject::checkCollision	13.3%	FireObject::checkCollision	13.1%
FireObject::ProcessFireCollisionsRange	7.6%	FireObject::ProcessFireCollisionsRange	6.9%
D3DXSHProjectCubeMap	3.7%	D3DXSHProjectCubeMap	3.6%
tbb::internal::task_scheduler_observer_v3::obs	3.1%	NtYieldExecution	3.1%
D3DDisassemble10Effect	2.9%	NtReadFile	3.1%
std::vector<class FireObject *,class std::alloc	2.8%	D3DDisassemble10Effect	3.0%
Ogre::TextureUnitState::getDesiredFormat	2.8%	tbb::internal::task_scheduler_observer_v3	2.5%
RtlInitUnicodeString	2.3%	Ogre::TextureUnitState::getDesiredForm	2.5%
modf	2.3%	std::vector<class FireObject *,class std::a	2.5%
ParticleEmitter::ParticleSystem::update	1.5%	WaitForSingleObject	2.5%
Animal::UpdateFear	1.5%	RtlInitUnicodeString	2.4%
[fmodex.dll]	1.4%	modf	2.4%
[nvd3dum.dll]	1.3%	tbb::captured_exception::~~captured_exc	1.9%
LdrUnlockLoaderLock	1.3%	Animal::UpdateFear	1.4%
hkpMoppObbVirtualMachine::queryAabbOnTre	1.1%	NtWaitForMultipleObjects	1.4%
[nvlddmkm.sys]	1.1%	ParticleEmitter::ParticleSystem::update	1.4%
ParticleEmitter::FirePatch::initParticle	1.1%	NtCreateFile	1.3%
KeSynchronizeExecution	1.1%	hkpMoppObbVirtualMachine::queryAab	1.2%

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

Agenda

- Performance Tuning Methodology
- Intel® VTune™ Amplifier XE: User Interface
- Fundamental Analysis: Hotspots
- **Finding Issues in Parallel Applications**
- Using the Performance Monitoring Unit

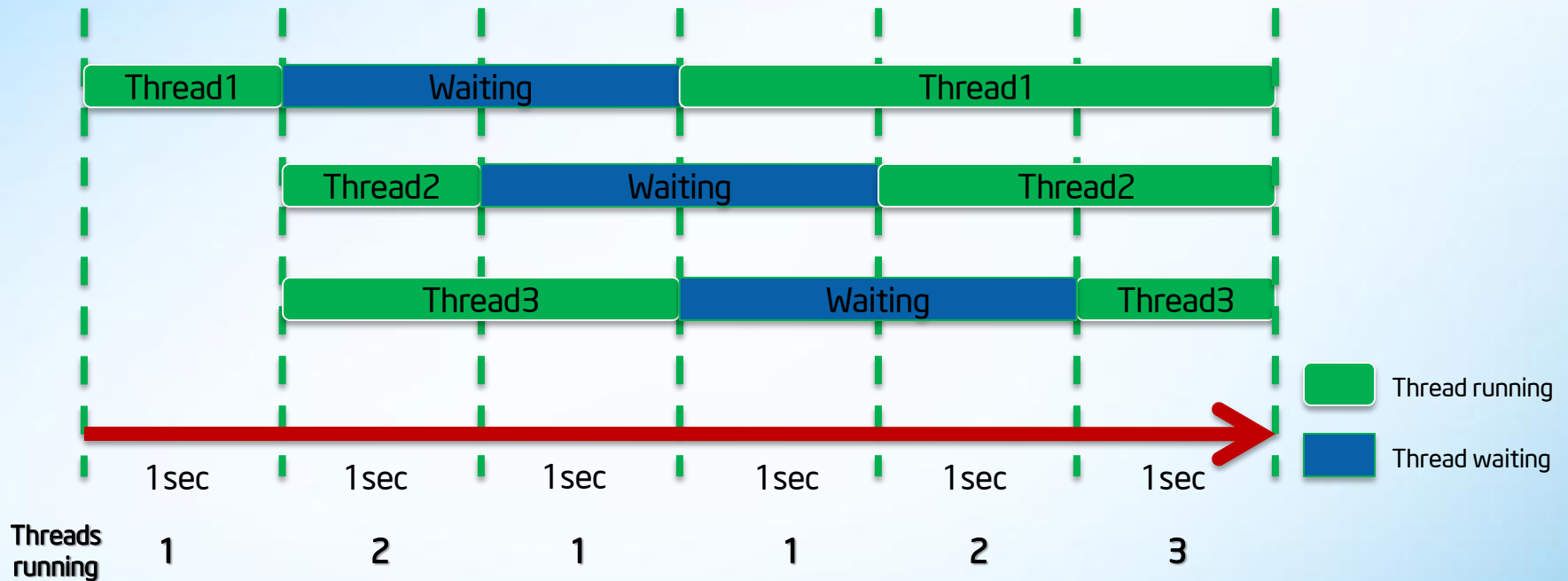
Issues in Parallel Applications

- Load imbalance
 - Work distribution is not optimal
 - Some threads are heavily loaded, while others idle
 - Slowest thread determines total speed-up
- Locking issues
 - Locks prohibit threads to concurrently enter code regions
 - Effectively serialize execution
- Parallelization overhead
 - With large no. of threads, data partition get smaller
 - Overhead might get significant (e.g. OpenMP startup time)

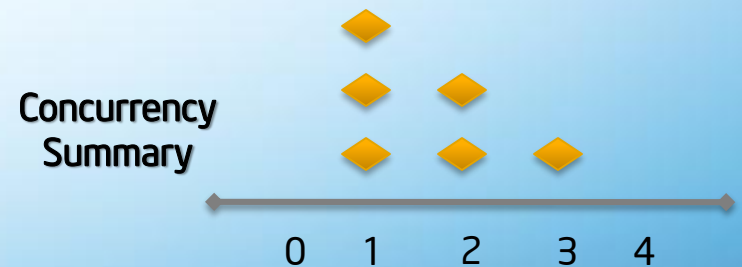
Issues in Parallel Applications

- Load imbalance
 - Work distribution is not optimal
 - Some threads are heavily loaded, while others idle
 - Slowest thread determines total speed-up
- Locking issues
 - Locks prohibit threads to concurrently enter code regions
 - Effectively serialize execution
- Parallelization overhead
 - With large no. of threads, data partition get smaller
 - Overhead might get significant (e.g. OpenMP startup time)

Threading Analysis Terminology

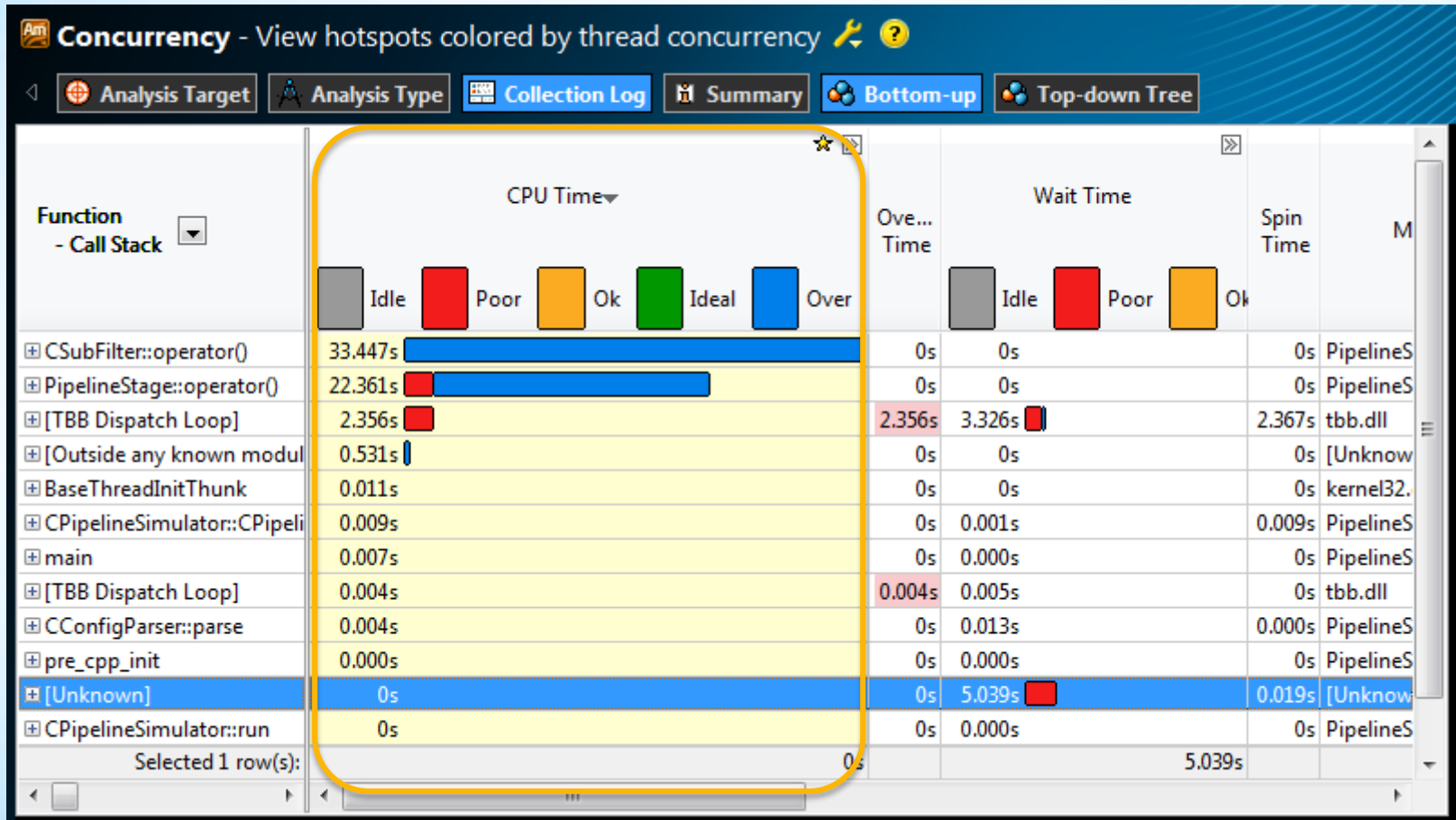


- **Elapsed Time:** 6 seconds
- **CPU Time:** T1 (4s) + T2 (3s) + T3 (3s) = 10 seconds
- **Wait Time:** T1(2s) + T2(2s) + T3 (2s) = 6 seconds



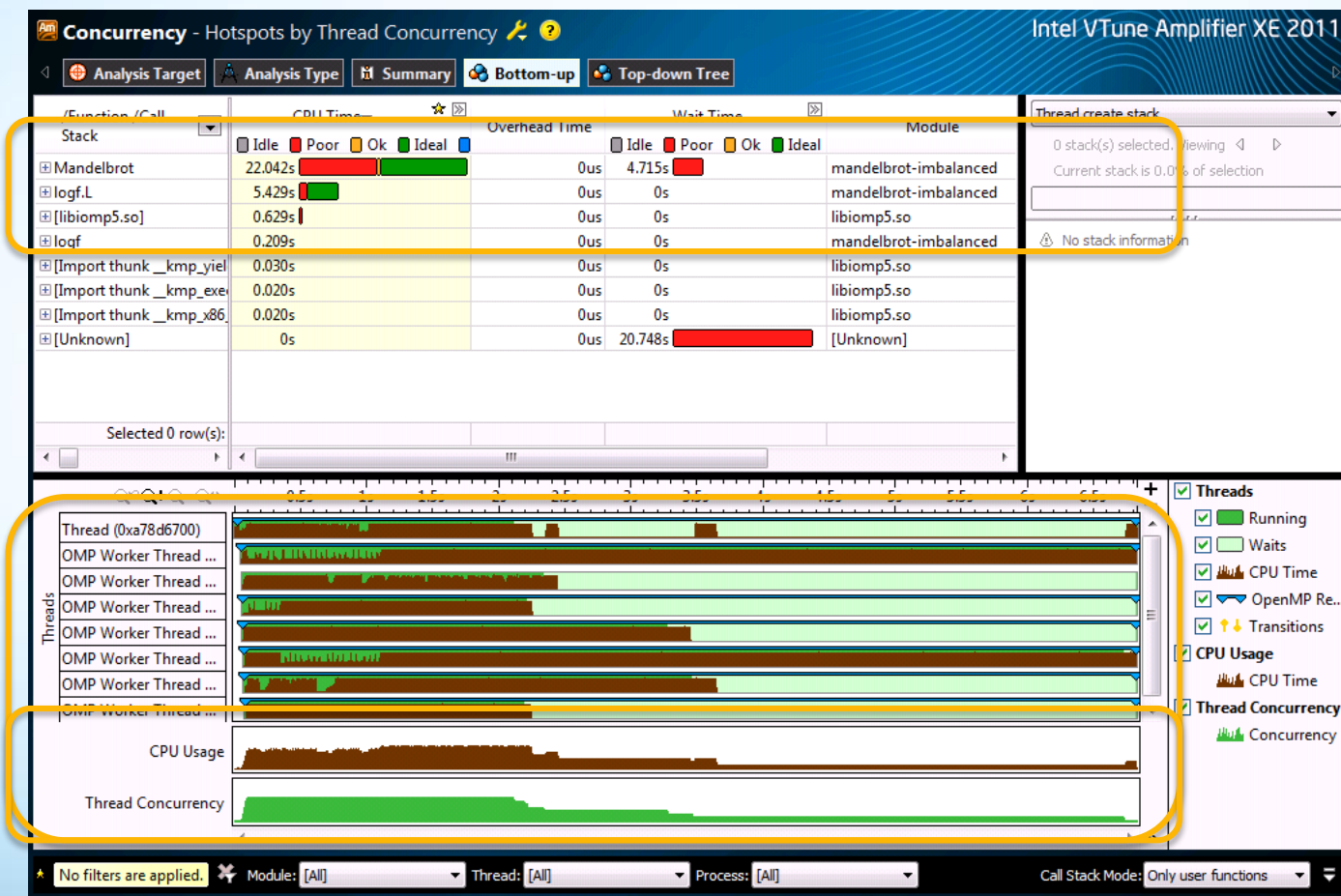
Analysis Types

Concurrency

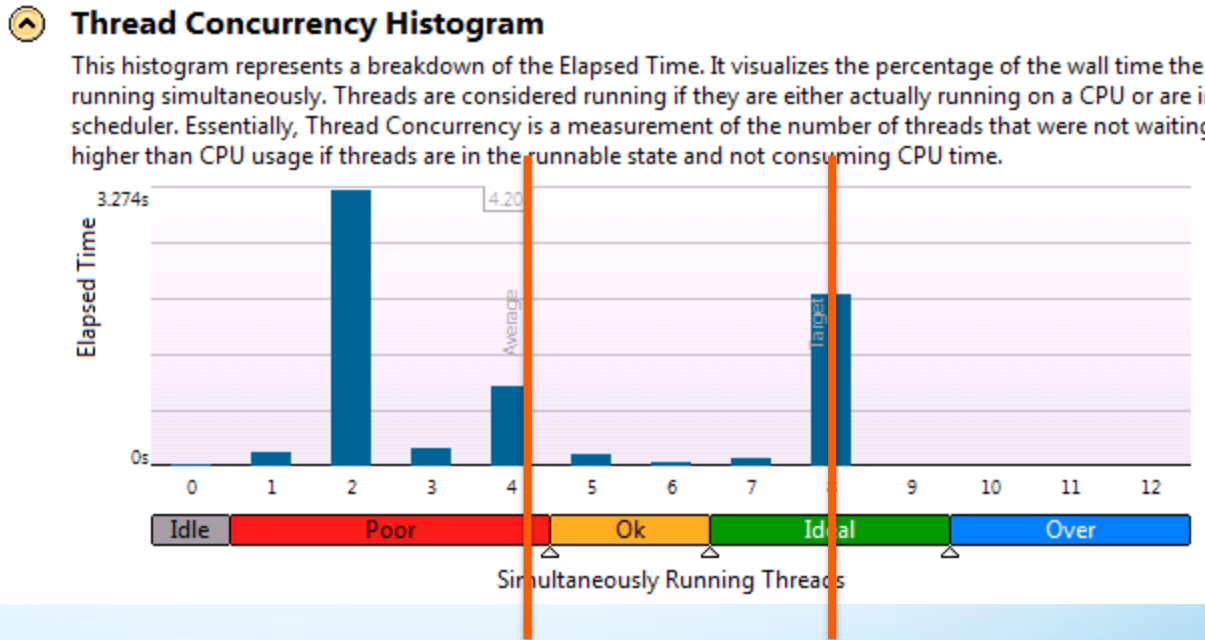


Hotspots Analysis vs. Concurrency Analysis

- Hotspot Analysis and Concurrency Analysis are similar:



Thread Concurrency Histogram



Average Target

Issues in Parallel Applications

- Load imbalance
 - Work distribution is not optimal
 - Some threads are heavily loaded, while others idle
 - Slowest thread determines total speed-up
- Locking issues
 - Locks prohibit threads to concurrently enter code regions
 - Effectively serialize execution
- Parallelization overhead
 - With large no. of threads, data partition get smaller
 - Overhead might get significant (e.g. OpenMP startup time)

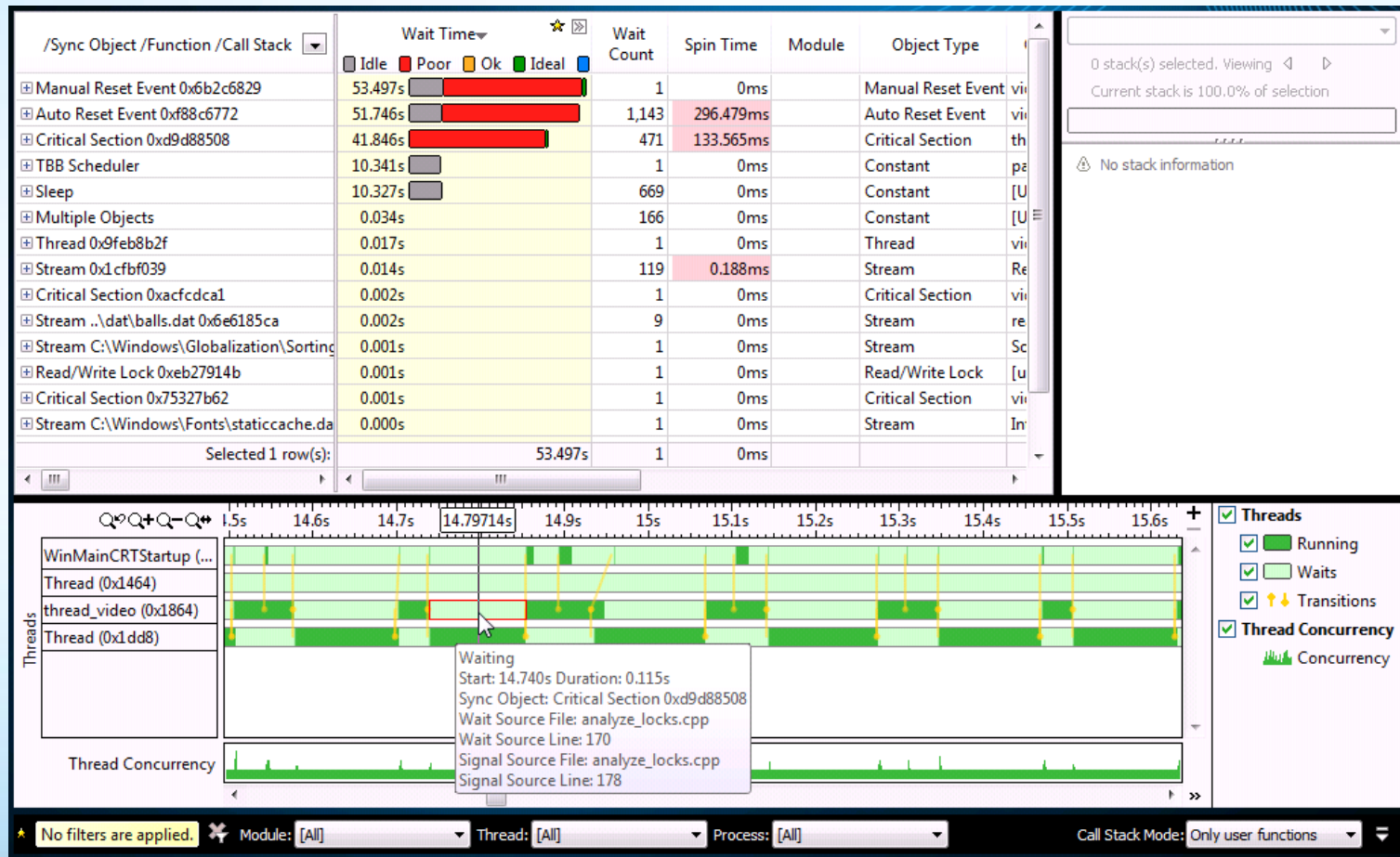
Analysis Types

Lock and Waits

Sync Object - Function - Call Stack	Wait Time	Wait Count	Spin Time	Module	Object Type	Object Creation
+ Multiple Objects	112.517s	5,325	0s	[Unknown]	Constant	[Unknown]
+ Manual Reset Event 0xae37	109.238s	41	0s	[Unknown]	Manual Reset Event	dllStopPlugin
+ Manual Reset Event 0xfa6d	74.068s	26	0s	[Unknown]	Manual Reset Event	LdrGetProcedureAddressEx
+ Thread Pool	57.628s	235	0s	[Unknown]	Constant	[Unknown]
+ Sleep	57.371s	5,234	0.193s	[Unknown]	Constant	[Unknown]
+ Unknown 0x991c9877	56.974s	6,337	0s	[Unknown]	Unknown	LdrGetProcedureAddressEx
+ TBB Scheduler	41.457s	2,200	11.301s	[Unknown]	Constant	TaskManagerTBB::Init
+ [Unknown]	17.061s	865	0s	[Unknown]	[Unknown]	[Unknown]
+ Stream ../media/graphics/	0.457s	183	0.057s	[Unknown]	Stream	Ogre::FileSystemArchive::open
+ Stream ../media/sounds/h	0.440s	171	0.063s	[Unknown]	Stream	Framework::GDParser::EndEler
+ Stream Ogre.log 0x501382c	0.397s	193	0.059s	[Unknown]	Stream	Ogre::Log::Log
+ Stream Smoke.gdf 0xf2b92	0.386s	11	0.006s	[Unknown]	Stream	PlatformManager::FileSystem::S
+ Stream ../media/sounds/M	0.306s	119	0.037s	[Unknown]	Stream	Framework::GDParser::EndEler
+ Stream ..\media\physics\D	0.247s	5	0.011s	[Unknown]	Stream	hkStdioStreamReader::hkStdioS
+ Stream ../media/graphics/	0.136s	41	0.022s	[Unknown]	Stream	Ogre::FileSystemArchive::open
+ Stream ../media/sounds/h	0.134s	13	0.018s	[Unknown]	Stream	TaskManagerTBB::ParallelFor
Selected 1 row(s):	112.517s	5,325				

Timeline Visualizes Thread Behavior

- Retrieve additional information about waiting threads



Timeline Visualizes Thread Behavior

- Retrieve additional information on thread transitions

The screenshot displays the Intel Thread Analyzer interface. The top section is a table listing thread wait events. The bottom section is a timeline visualization showing thread execution and transitions over time.

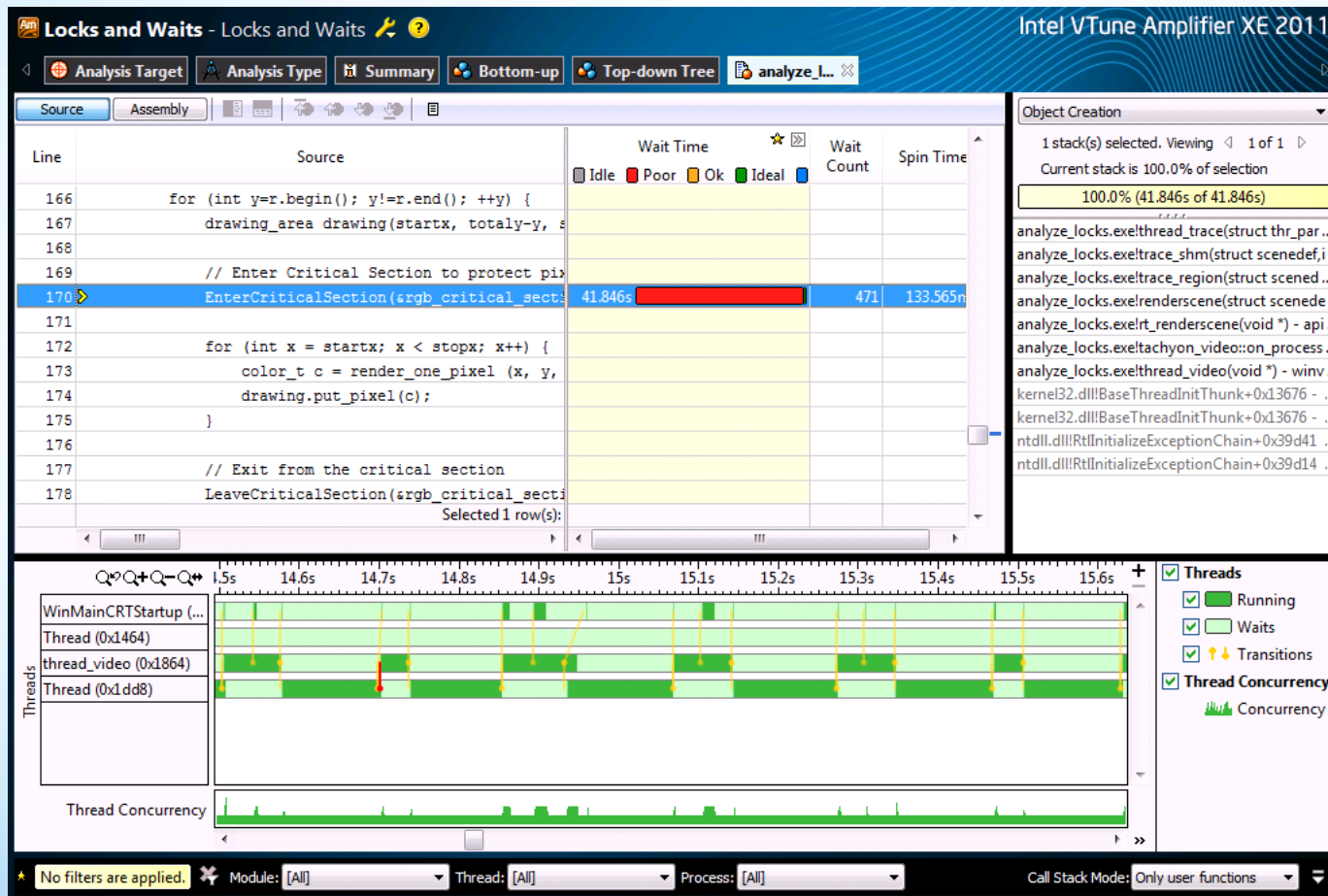
/Sync Object /Function /Call Stack	Wait Time	Wait Count	Spin Time	Module	Object Type
Manual Reset Event 0x6b2c6829	53.497s	1	0ms		Manual Reset Event
Auto Reset Event 0xf88c6772	51.746s	1,143	296.479ms		Auto Reset Event
Critical Section 0xd9d88508	41.846s	471	133.565ms		Critical Section
TBB Scheduler	10.341s	1	0ms		Constant
Sleep	10.327s	669	0ms		Constant
Multiple Objects	0.034s	166	0ms		Constant
Thread 0x9feb8b2f	0.017s	1	0ms		Thread
Stream 0x1cfbf039	0.014s	119	0.188ms		Stream
Critical Section 0xacfdca1	0.002s	1	0ms		Critical Section
Stream ..\dat\balls.dat 0x6e6185ca	0.002s	9	0ms		Stream
Stream C:\Windows\Globalization\Sorting	0.001s	1	0ms		Stream
Read/Write Lock 0xeb27914b	0.001s	1	0ms		Read/Write Lock
Critical Section 0x75327b62	0.001s	1	0ms		Critical Section
Stream C:\Windows\Fonts\staticcache.da	0.000s	1	0ms		Stream

Selected 1 row(s): 53.497s, 1, 0ms

The timeline visualization shows threads: WinMainCRTStartup (...), Thread (0x1464), thread_video (0x1864), and Thread (0x1dd8). A tooltip for a transition shows: Thread (0x1dd8) to thread_video (0x1864) (14.703s to 14.703s), Sync Object: Critical Section 0xd9d88508, Wait Source File: analyze_locks.cpp, Wait Source Line: 170, Signal Source File: analyze_locks.cpp, Signal Source Line: 178.

Drilling down into Thread Behavior

- Reveal source code that causes thread transitions



Agenda

- Performance Tuning Methodology
- Intel® VTune™ Amplifier XE: User Interface
- Fundamental Analysis: Hotspots
- Finding Issues in Parallel Applications
- Using the Performance Monitoring Unit

Performance Monitoring Unit (PMU)

- **Per-core PMU:**

- Each core provides 2 programmable counters and 1 fixed counters.
- The programmable per-core counters can be configured to investigate front-end/micro-op flow issues, stalls inside a processor core.

- **Uncore PMU:**

- Uncore of the coprocessor has four counters to monitor uncore events
- Can be used to investigate memory behavior and global on-chip issues

Event Based Performance Analysis

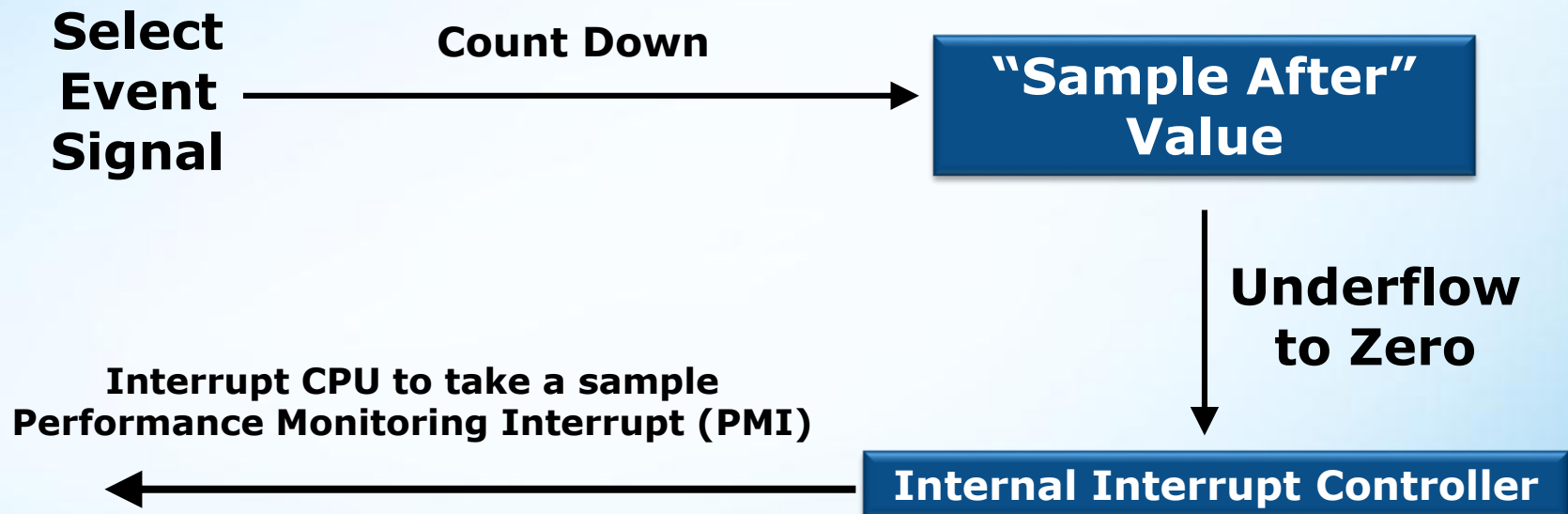
Event Based Sampling(EBS)

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

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

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

How Event Based Sampling (EBS) Works



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

$$\text{Events} = \text{samples} * \text{sample after value}$$

Native Launch configuration

Target Search Directories

Target type Launch Application

Launch Application
Specify and configure the application executable (target) to analyze. Press F1 for more details.

Application: ssh Browse...

Application parameters: mic0 /tmp/diffusionMIC tiled Modify...

Working directory: /home/michome/rreed/projects/MICtest/perfMIC Browse...

- Application settings:

- Application: ssh
- Parameters: mic0 "<app startup>"
- Working directory: Usually does not matter
- Don't forget to set search directories under "All files"

Target Search Directories

Search directories for: All files

Search Directories	Search Subdirectories
/home/michome/rreed/projects/MICtest	<input type="checkbox"/>
/lib/firmware/mic	<input checked="" type="checkbox"/>

Application Configuration

File View Help

r000lh r001lh r002lh r003lh New Amplifier XE Result

Choose Analysis Type

Analysis Type

- Access Contention
- Branch Analysis
- Client Analysis
- Core Port Saturation
- Cycles and uOps
- Loop Analysis
- Memory Access
- Port Saturation
- Intel Atom Processor Analysis
 - General Exploration
- Knights Corner Platform Analysis
 - Lightweight Hotspots**
- Power Analysis
 - CPU Sleep States
 - CPU Frequency
- Custom Analysis
 - Branch Mispredicts
 - Cache Misses
 - Execution Stalls
 - False Sharing

Knights Corner Platform - Lightweight Hotspots Copy

Identify your most time-consuming source code. Unlike Hotspots, Lightweight Hotspots has lower overhead because it does not collect stack information. It can also be used to sample all processes on a system. This analysis type uses hardware event-based sampling collection. Press F1 for more details.

List of MIC cards (e.g. 0,1,2,3):

Details

Events configured for CPU: Intel(R) Xeon(R) / Core i7 980X 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 De
CPU_CLK_UNHALTED	10000000	Number of cycles during which
INSTRUCTIONS_EXECUTED	10000000	Number of instructions execut

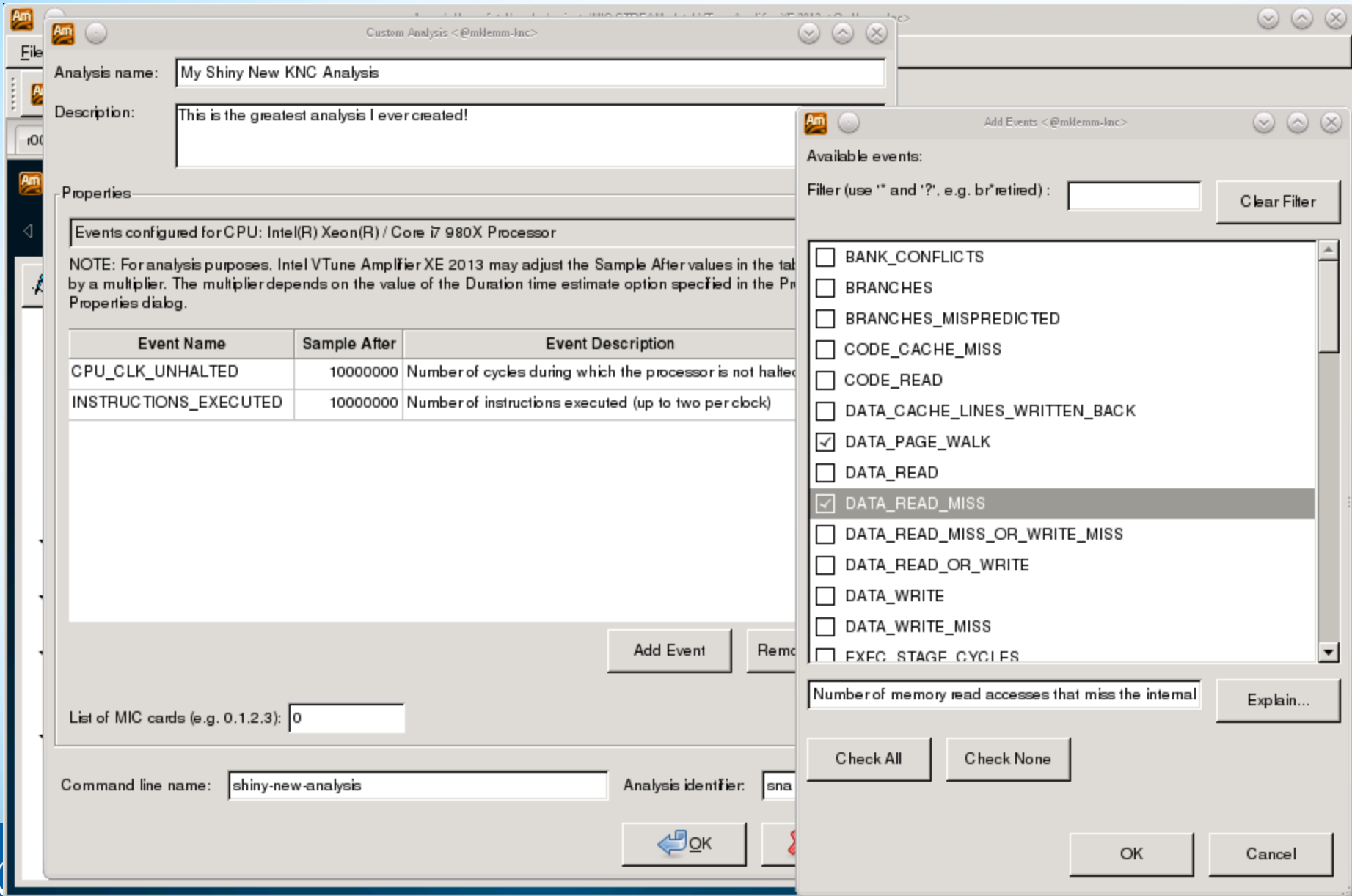
Start

Start Paused

Project Properties

Command Line...

Configuring a User-defined Analysis



Some useful events and metrics

Scenario	Event name(s)
Wall-clock profiling	CPU_CLK_UNHALTED, INSTRUCTIONS_EXECUTED (or EXEC_STAGE_CYCLES)
Main memory bandwidth	L2_DATA_READ_MISS_MEM_FILL, L2_DATA_WRITE_MISS_MEM_FILL
L1 Cache misses	DATA_READ_MISS_OR_WRITE_MISS
TLB misses and page faults	DATA_PAGE_WALK, LONG_DATA_PAGE_WALK, DATA_PAGE_FAULT
Vectorized code execution	VPU_INSTRUCTIONS_EXECUTED, VPU_ELEMENTS_ACTIVE
Various hazards	BRANCHES_MISPREDICTED
Cycles per instruction	CPU_CLK_UNHALTED / INSTRUCTIONS_EXECUTED
Memory Bandwidth (used by all cores at once)	$(L2_DATA_READ_MISS_MEM_FILL + L2_DATA_WRITE_MISS_MEM_FILL) * 64 / CPU_CLK_UNHALTED / \text{Frequency}$

Example: Hotspots of OpenFOAM

