Intel VTune Amplifier XE

Seminar Software Profiling

Lena Herscheid

Supervisor: Dr. Peter Tröger

May 27th 2013

Agenda

- **Overview**
    - Sampling Approach
- Features
    - Algorithm Analysis
    - Hardware Profiling
- How It Works
    - User mode Sampling
    - Hardware Event Sampling
- Evaluation

Overview

- Commercial profiling tool
  - part of Intel Parallel Studio or standalone
  - provides CLI and GUI
  - Visual Studio and Eclipse plugins
- Fortran, C, C++, Java, .NET, Assembly
- Linux and Windows

The Sampling Approach

an ever-present trade-off: **Overhead ⇔ Information**

idea: collect information sporadically

- time-driven: uniform time period between samples
- event-driven: uniform number of events between samples

+ less intrusive → no side effects introduced by profiling

+ small impact on execution speed → detect timing issues accurately

+ low overhead

  + user-mode sampling: about 5% (default interval of 10ms)

  + hardware event-based sampling: about 2% (1ms sampling interval)

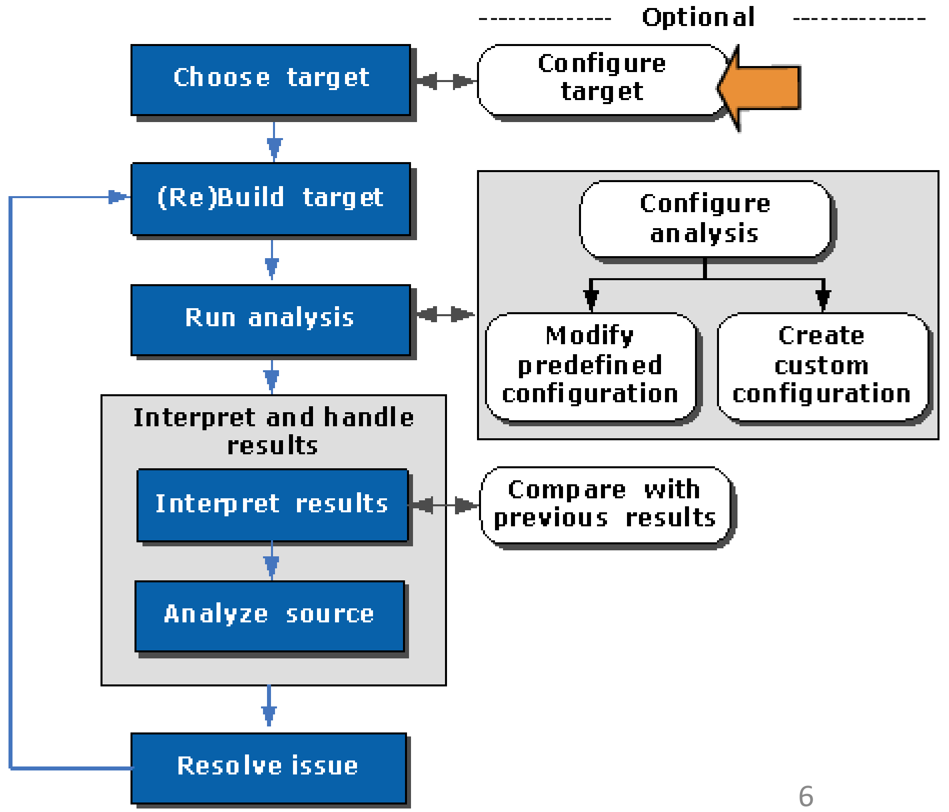
Sampling Mistakes

- **the entire sampling interval** before an event is attributed to the code context
  - 10-2000k events depending on sampling interval
  - negligible for many samples and frequent events
- per-user filtering switched off by default
  - (and cannot be switched on again after installation…)
  - samples from **all processes** on the system are collected!
- call graphs are approximated and can be misleading
  - with infrequently asymmetric call patterns

Profiling Workflow

applications are profiled under different configurations

- collects a subset of supported metrics in one analysis run
- re-configure and re-run multiple times
  - to keep overhead low for each run
- debug symbols needed
  - for source-code-line granularity



Agenda

- Overview
    - Sampling Approach
- **Features**
    - Algorithm Analysis
    - Hardware Profiling
- How It Works
    - User mode Sampling
    - Hardware Event Sampling
- Evaluation

Algorithm Analysis

Choose viewpoint depending on profiling objective:

- Where are significant portions of time spent?

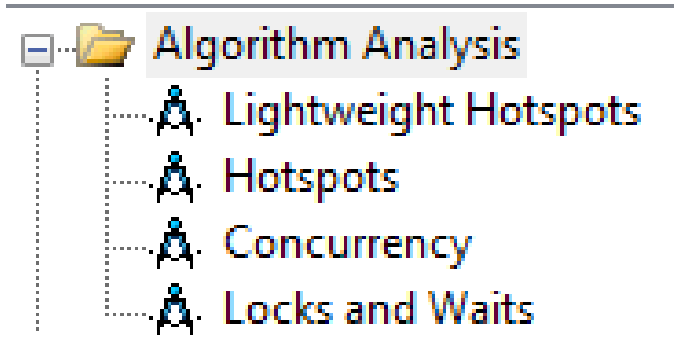
  → **Hotspot Analysis**

- How well is CPU time utilized? How well are threads scheduled?

  → **Concurrency Analysis**

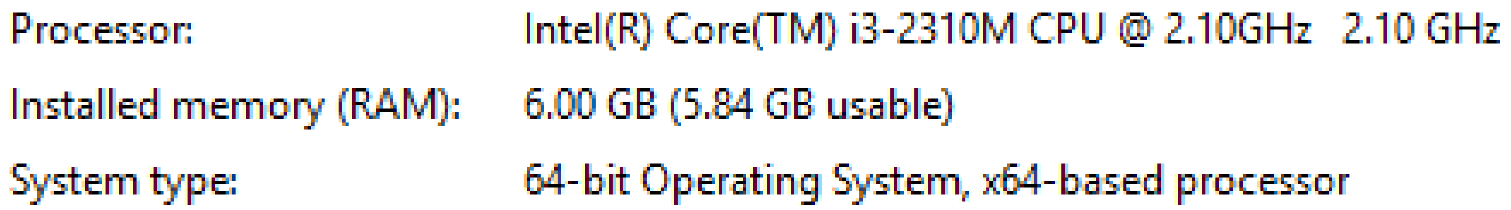
- What are causes of ineffective CPU utilization?

  → **Waits and Locks Analysis**



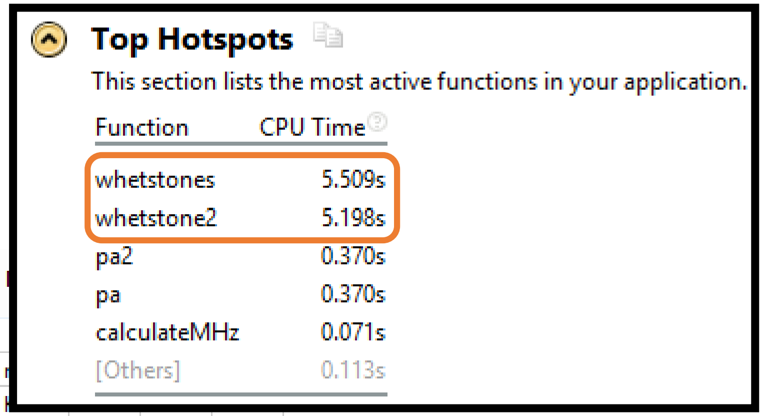
VTune Examples

- CPU/Memory benchmarks (http://www.roylongbottom.org.uk)
  - Whetstone: floating point computations
  - RandMem: memory read/write
- System configuration:



Processor:	Intel(R) Core(TM) i3-2310M CPU @ 2.10GHz 2.10 GHz
Installed memory (RAM):	6.00 GB (5.84 GB usable)
System type:	64-bit Operating System, x64-based processor

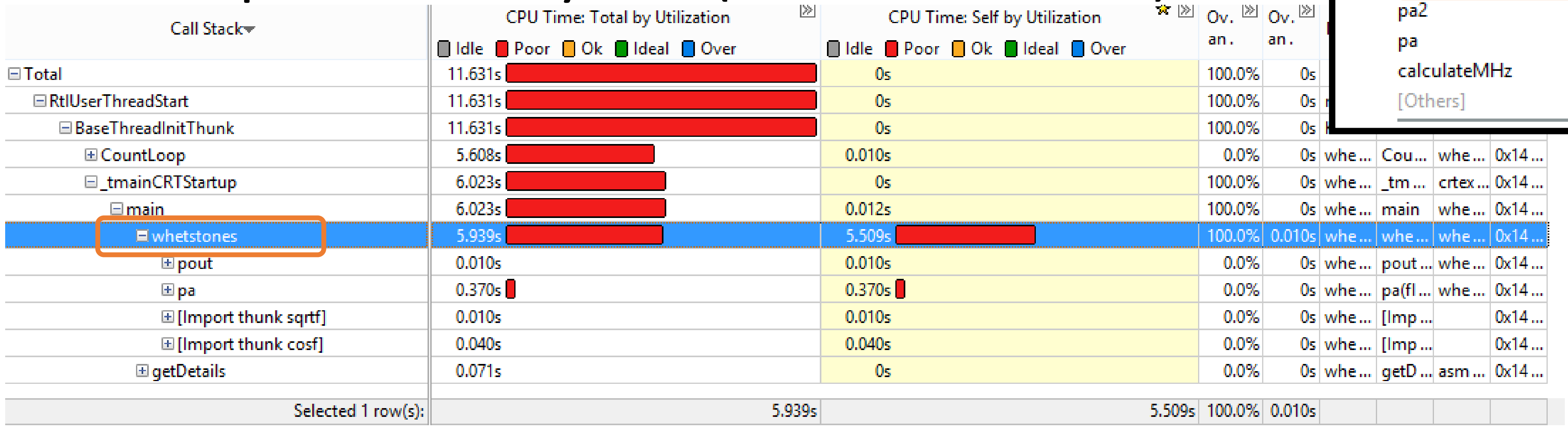
Hotspot Analysis (Whetstone)



Top Hotspots

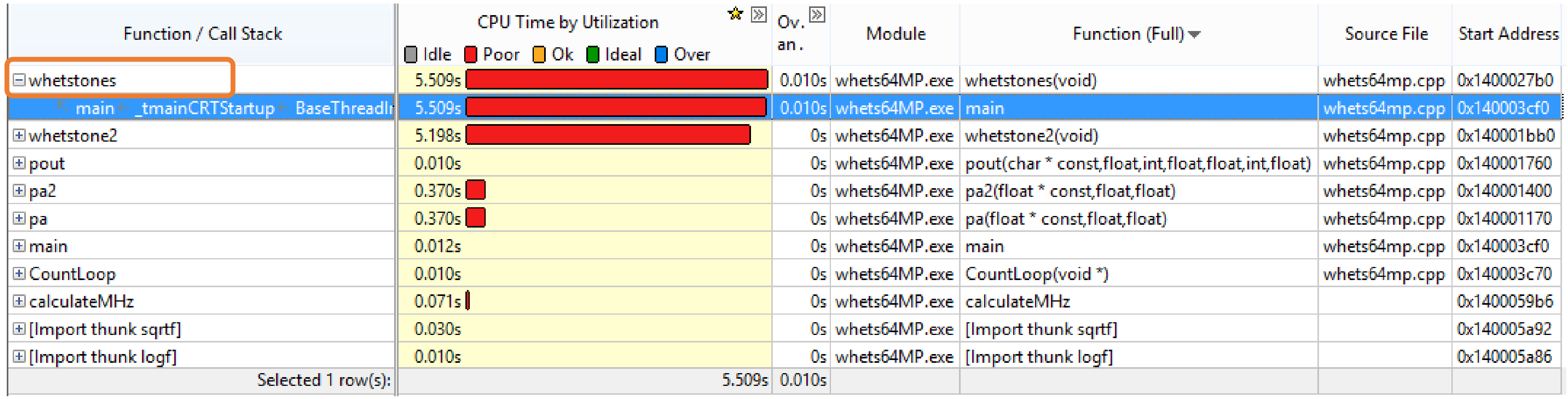
This section lists the most active functions in your application.

Function	CPU Time
whetstones	5.509s
whetstone2	5.198s
pa2	0.370s
pa	0.370s
calculateMHz	0.071s
[Others]	0.113s



Call Stack	CPU Time: Total by Utilization (▨ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over)	CPU Time: Self by Utilization (▨ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over)	Ov. an.	Ov. an.				
Total	11.631s	0s	100.0%	0s				
RtlUserThreadStart	11.631s	0s	100.0%	0s				
BaseThreadInitThunk	11.631s	0s	100.0%	0s				
CountLoop	5.608s	0.010s	0.0%	0s	whe…	Cou…	whe…	0x14…
_tmainCRTStartup	6.023s	0s	100.0%	0s	whe…	_tm…	crtex…	0x14…
main	6.023s	0.012s	100.0%	0s	whe…	main	whe…	0x14…
whetstones	5.939s	5.509s	100.0%	0.010s	whe…	whe…	whe…	0x14…
pout	0.010s	0.010s	0.0%	0s	whe…	pout…	whe…	0x14…
pa	0.370s	0.370s	0.0%	0s	whe…	pa(fl…	whe…	0x14…
[Import thunk sqrtf]	0.010s	0.010s	0.0%	0s	whe…	[Imp…		0x14…
[Import thunk cosf]	0.040s	0.040s	0.0%	0s	whe…	[Imp…		0x14…
getDetails	0.071s	0s	0.0%	0s	whe…	getD…	asm…	0x14…
Selected 1 row(s):	5.939s	5.509s	100.0%	0.010s				

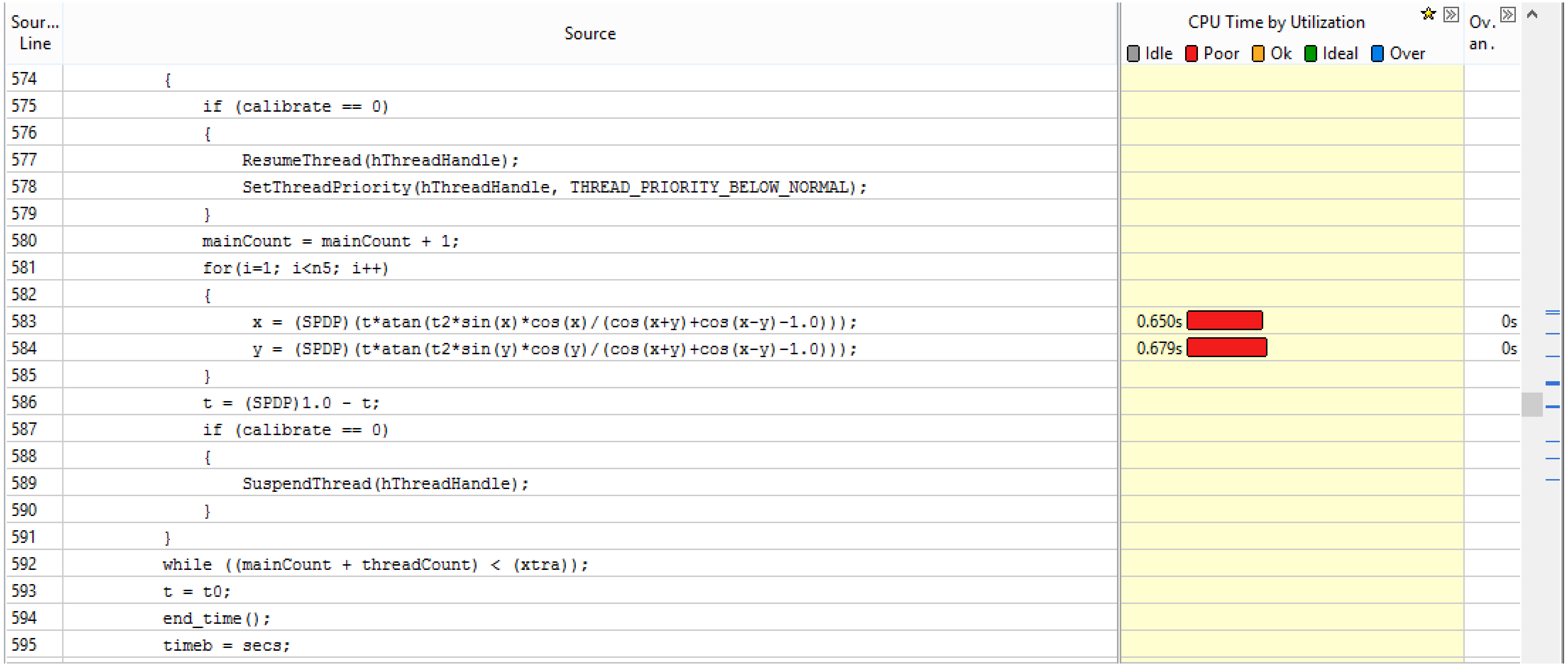
Top-Down View
(Call Tree)



Function / Call Stack	CPU Time by Utilization (▨ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over)	Ov. an.	Module	Function (Full)	Source File	Start Address
whetstones	5.509s	0.010s	whets64MP.exe	whetstones(void)	whets64mp.cpp	0x1400027b0
main ← _tmainCRTStartup ← BaseThreadIı	5.509s	0.010s	whets64MP.exe	main	whets64mp.cpp	0x140003cf0
whetstone2	5.198s	0s	whets64MP.exe	whetstone2(void)	whets64mp.cpp	0x140001bb0
pout	0.010s	0s	whets64MP.exe	pout(char * const,float,int,float,float,int,float)	whets64mp.cpp	0x140001760
pa2	0.370s	0s	whets64MP.exe	pa2(float * const,float,float)	whets64mp.cpp	0x140001400
pa	0.370s	0s	whets64MP.exe	pa(float * const,float,float)	whets64mp.cpp	0x140001170
main	0.012s	0s	whets64MP.exe	main	whets64mp.cpp	0x140003cf0
CountLoop	0.010s	0s	whets64MP.exe	CountLoop(void *)	whets64mp.cpp	0x140003c70
calculateMHz	0.071s	0s	whets64MP.exe	calculateMHz		0x1400059b6
[Import thunk sqrtf]	0.030s	0s	whets64MP.exe	[Import thunk sqrtf]		0x140005a92
[Import thunk logf]	0.010s	0s	whets64MP.exe	[Import thunk logf]		0x140005a86
Selected 1 row(s):	5.509s	0.010s				

Bottom-Up View

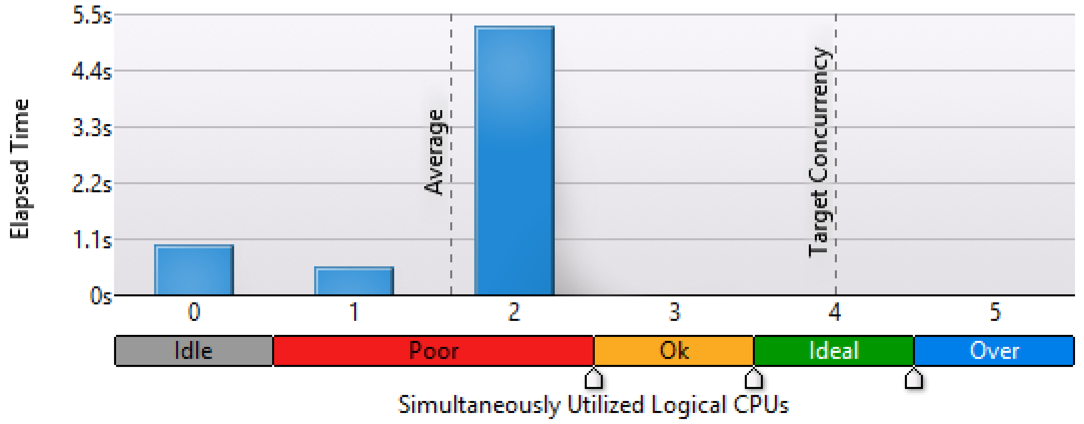
Hotspot Analysis (Whetstone)



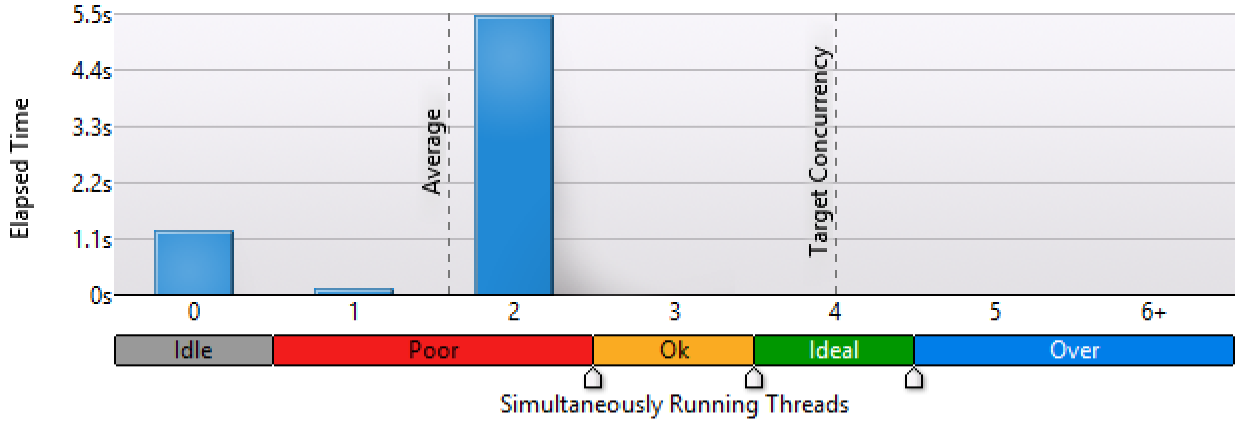
Sour... Line	Source	CPU Time by Utilization ⬜ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Ov. an.
574	`{`		
575	`if (calibrate == 0)`		
576	`{`		
577	`ResumeThread(hThreadHandle);`		
578	`SetThreadPriority(hThreadHandle, THREAD_PRIORITY_BELOW_NORMAL);`		
579	`}`		
580	`mainCount = mainCount + 1;`		
581	`for(i=1; i<n5; i++)`		
582	`{`		
583	`x = (SPDP)(t*atan(t2*sin(x)*cos(x)/(cos(x+y)+cos(x-y)-1.0)));`	0.650s 🟥	0s
584	`y = (SPDP)(t*atan(t2*sin(y)*cos(y)/(cos(x+y)+cos(x-y)-1.0)));`	0.679s 🟥	0s
585	`}`		
586	`t = (SPDP)1.0 - t;`		
587	`if (calibrate == 0)`		
588	`{`		
589	`SuspendThread(hThreadHandle);`		
590	`}`		
591	`}`		
592	`while ((mainCount + threadCount) < (xtra));`		
593	`t = t0;`		
594	`end_time();`		
595	`timeb = secs;`		

Concurrency Analysis

CPU Usage: threads consuming CPU time

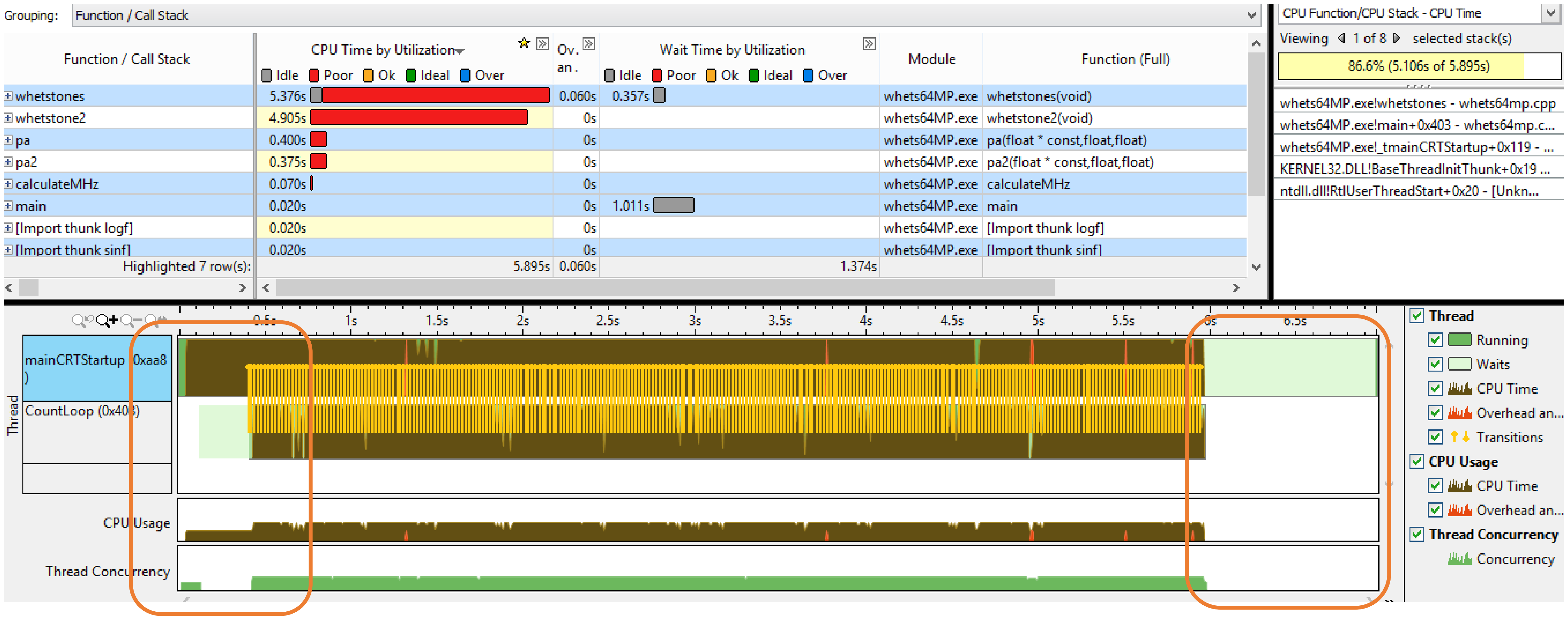


Thread Concurrency: threads in runnable state



- thread concurrency > CPU usage: thread oversubscription
- CPU usage > thread concurrency: spinning threads

Concurrency Analysis (Whetstone)



Grouping: Function / Call Stack

Function / Call Stack	CPU Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Ov. an.	Wait Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Module	Function (Full)
⊞ whetstones	5.376s	0.060s	0.357s	whets64MP.exe	whetstones(void)
⊞ whetstone2	4.905s	0s		whets64MP.exe	whetstone2(void)
⊞ pa	0.400s	0s		whets64MP.exe	pa(float * const,float,float)
⊞ pa2	0.375s	0s		whets64MP.exe	pa2(float * const,float,float)
⊞ calculateMHz	0.070s	0s		whets64MP.exe	calculateMHz
⊞ main	0.020s	0s	1.011s	whets64MP.exe	main
⊞ [Import thunk logf]	0.020s	0s		whets64MP.exe	[Import thunk logf]
⊞ [Import thunk sinf]	0.020s	0s		whets64MP.exe	[Import thunk sinf]
Highlighted 7 row(s):	5.895s	0.060s	1.374s		

CPU Function/CPU Stack - CPU Time

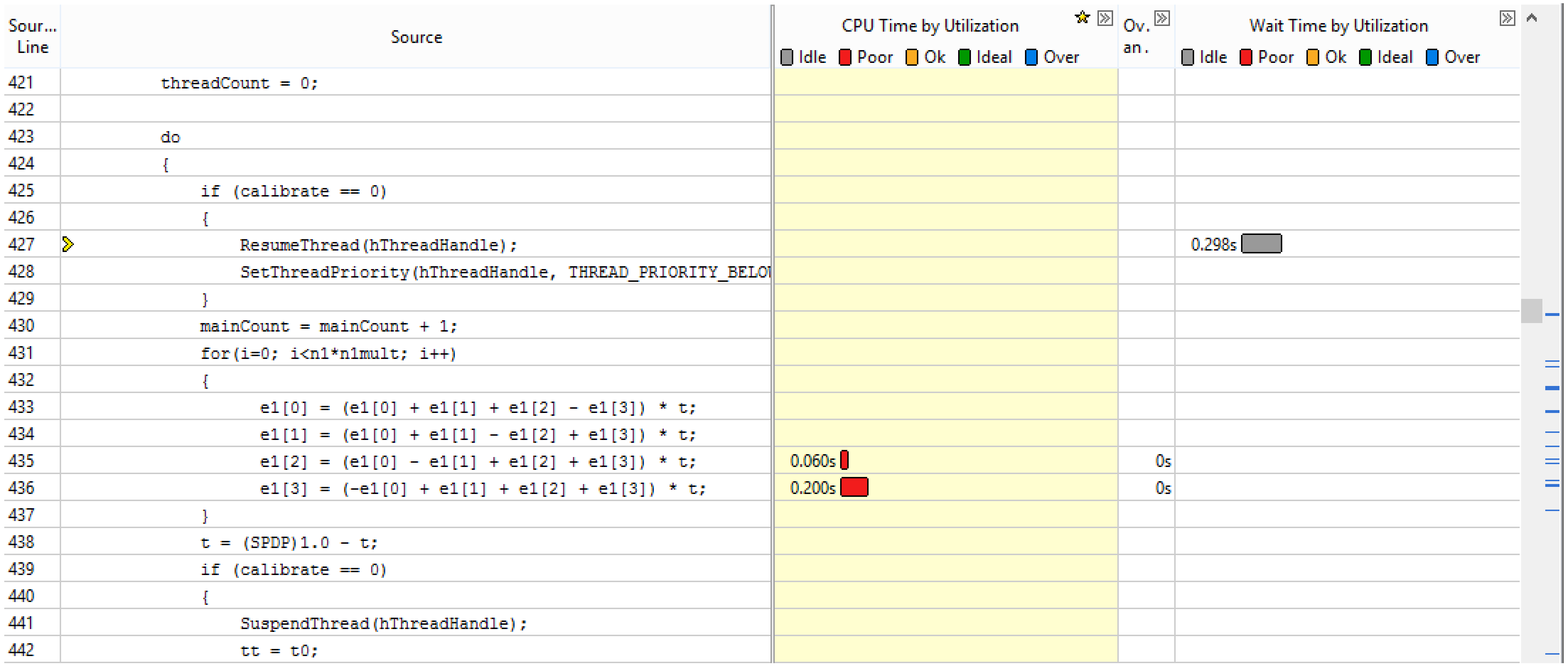
Viewing ◁ 1 of 8 ▷ selected stack(s)

86.6% (5.106s of 5.895s)

whets64MP.exe!whetstones - whets64mp.cpp
whets64MP.exe!main+0x403 - whets64mp.c...
whets64MP.exe!_tmainCRTStartup+0x119 - ...
KERNEL32.DLL!BaseThreadInitThunk+0x19 ...
ntdll.dll!RtlUserThreadStart+0x20 - [Unkn...

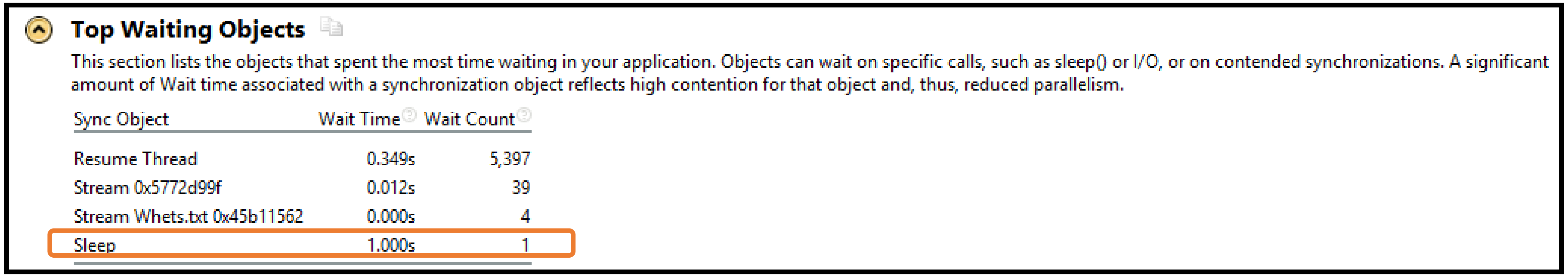
✓ Thread
✓ Running
✓ Waits
✓ CPU Time
✓ Overhead an...
✓ Transitions
✓ CPU Usage
✓ CPU Time
✓ Overhead an...
✓ Thread Concurrency
Concurrency

Concurrency Analysis (Whetstone)



Sour... Line	Source	CPU Time by Utilization ⬜ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Ov. an.	Wait Time by Utilization ⬜ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over
421	threadCount = 0;			
422				
423	do			
424	{			
425	if (calibrate == 0)			
426	{			
427	ResumeThread(hThreadHandle);			0.298s ⬜
428	SetThreadPriority(hThreadHandle, THREAD_PRIORITY_BELOW			
429	}			
430	mainCount = mainCount + 1;			
431	for(i=0; i<n1*n1mult; i++)			
432	{			
433	e1[0] = (e1[0] + e1[1] + e1[2] - e1[3]) * t;			
434	e1[1] = (e1[0] + e1[1] - e1[2] + e1[3]) * t;			
435	e1[2] = (e1[0] - e1[1] + e1[2] + e1[3]) * t;	0.060s 🟥	0s	
436	e1[3] = (-e1[0] + e1[1] + e1[2] + e1[3]) * t;	0.200s 🟥	0s	
437	}			
438	t = (SPDP)1.0 - t;			
439	if (calibrate == 0)			
440	{			
441	SuspendThread(hThreadHandle);			
442	tt = t0;			

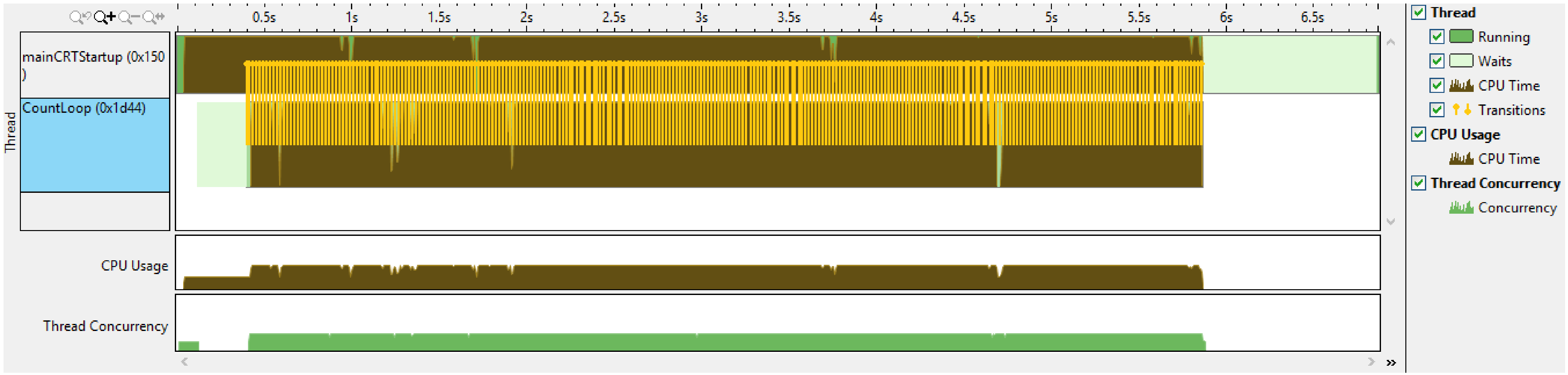
Locks and Waits Analysis (Whetstone)



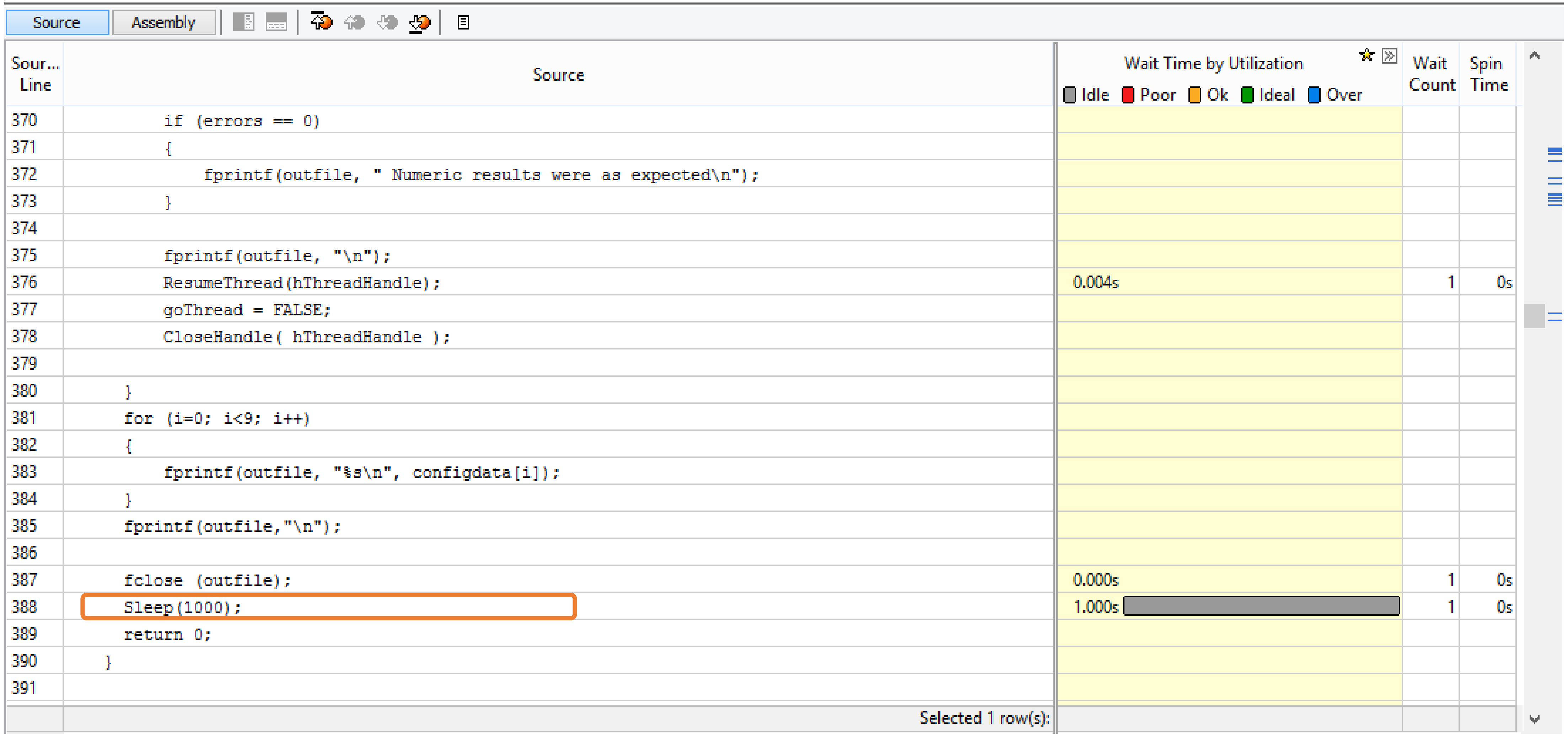
Top Waiting Objects

This section lists the objects that spent the most time waiting in your application. Objects can wait on specific calls, such as sleep() or I/O, or on contended synchronizations. A significant amount of Wait time associated with a synchronization object reflects high contention for that object and, thus, reduced parallelism.

Sync Object	Wait Time	Wait Count
Resume Thread	0.349s	5,397
Stream 0x5772d99f	0.012s	39
Stream Whets.txt 0x45b11562	0.000s	4
Sleep	1.000s	1



Locks and Waits Analysis (Whetstone)

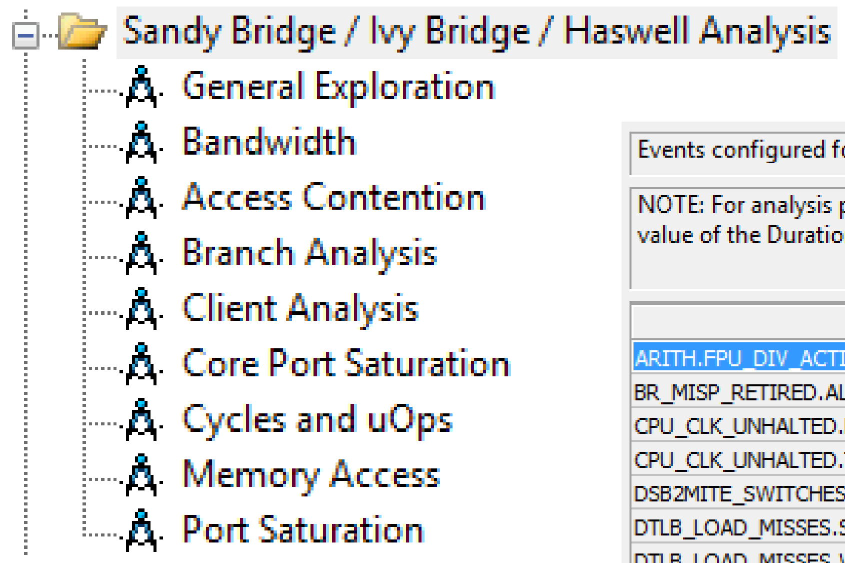


Sour... Line	Source	Wait Time by Utilization ▨ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over	Wait Count	Spin Time
370	`if (errors == 0)`			
371	`{`			
372	`fprintf(outfile, " Numeric results were as expected\n");`			
373	`}`			
374				
375	`fprintf(outfile, "\n");`			
376	`ResumeThread(hThreadHandle);`	0.004s	1	0s
377	`goThread = FALSE;`			
378	`CloseHandle( hThreadHandle );`			
379				
380	`}`			
381	`for (i=0; i<9; i++)`			
382	`{`			
383	`fprintf(outfile, "%s\n", configdata[i]);`			
384	`}`			
385	`fprintf(outfile,"\n");`			
386				
387	`fclose (outfile);`	0.000s	1	0s
388	`Sleep(1000);`	1.000s ▨	1	0s
389	`return 0;`			
390	`}`			
391				

Selected 1 row(s):

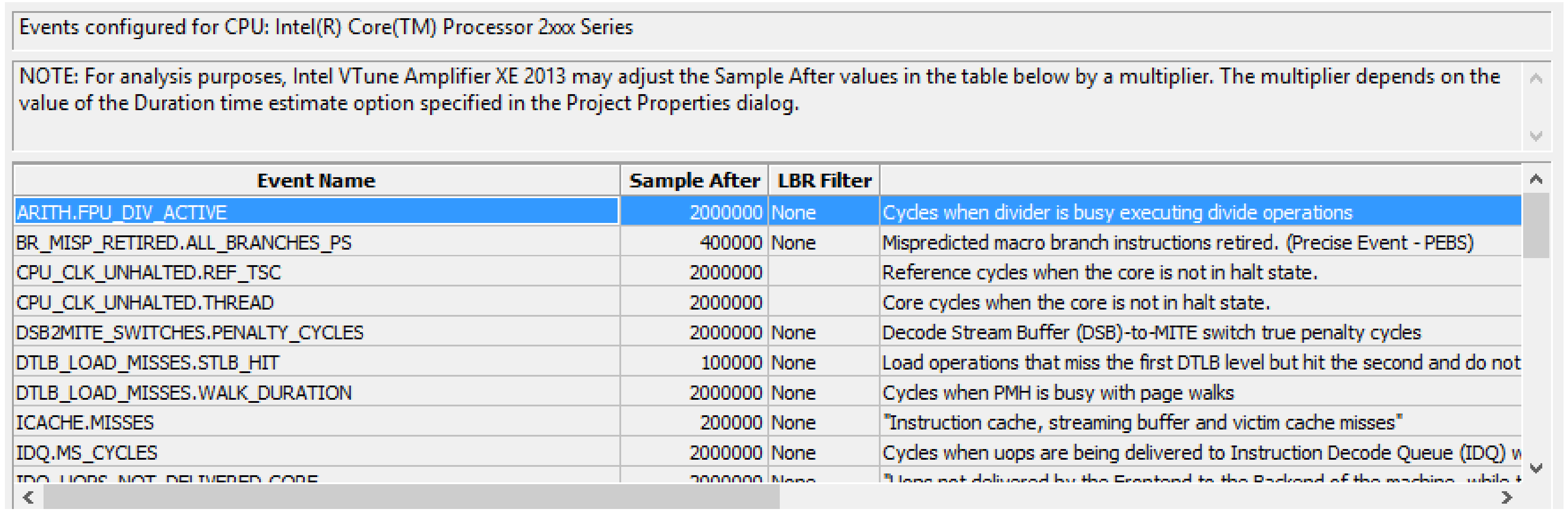
Hardware Profiling

- CPU-specific events and metrics
- profiling mode defines presets for Sample After Value and Events



Sandy Bridge / Ivy Bridge / Haswell Analysis
- General Exploration
- Bandwidth
- Access Contention
- Branch Analysis
- Client Analysis
- Core Port Saturation
- Cycles and uOps
- Memory Access
- Port Saturation

General Exploration

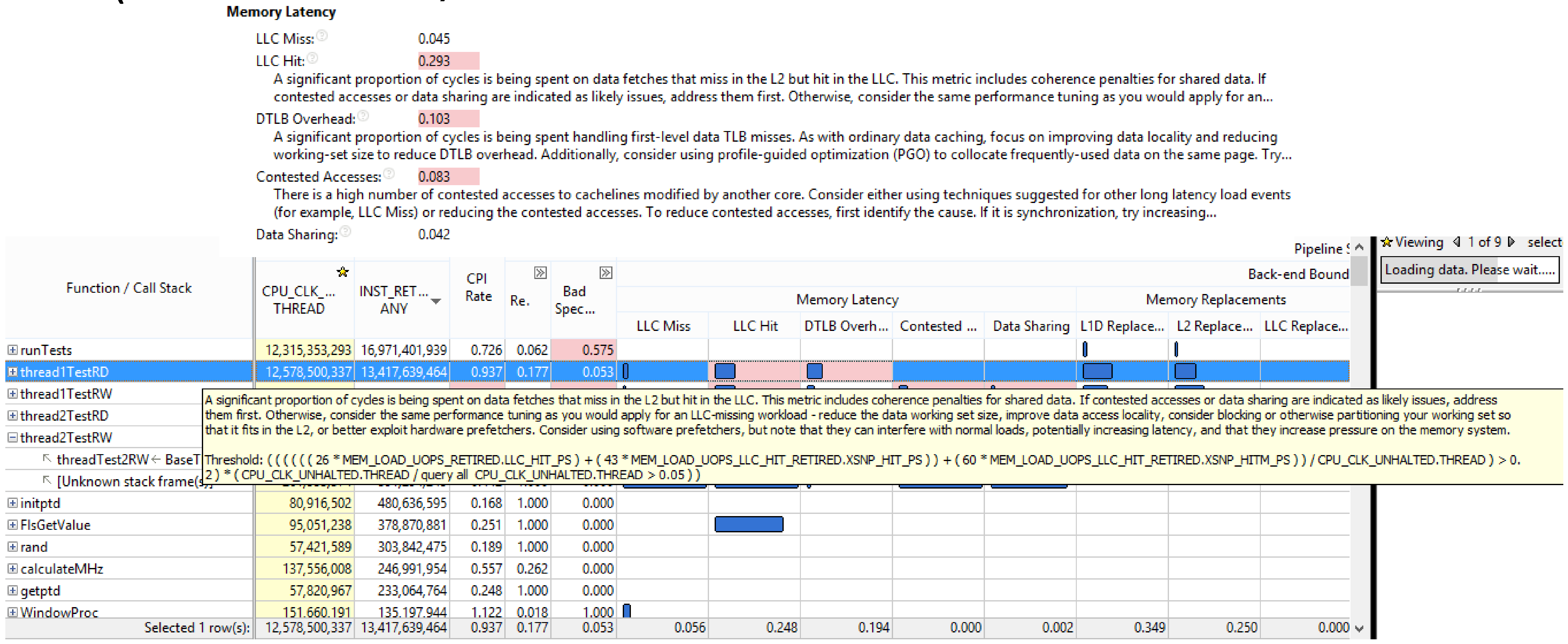


Events configured for CPU: Intel(R) Core(TM) Processor 2xxx Series

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	LBR Filter	
ARITH.FPU_DIV_ACTIVE	2000000	None	Cycles when divider is busy executing divide operations
BR_MISP_RETIRED.ALL_BRANCHES_PS	400000	None	Mispredicted macro branch instructions retired. (Precise Event - PEBS)
CPU_CLK_UNHALTED.REF_TSC	2000000		Reference cycles when the core is not in halt state.
CPU_CLK_UNHALTED.THREAD	2000000		Core cycles when the core is not in halt state.
DSB2MITE_SWITCHES.PENALTY_CYCLES	2000000	None	Decode Stream Buffer (DSB)-to-MITE switch true penalty cycles
DTLB_LOAD_MISSES.STLB_HIT	100000	None	Load operations that miss the first DTLB level but hit the second and do not
DTLB_LOAD_MISSES.WALK_DURATION	2000000	None	Cycles when PMH is busy with page walks
ICACHE.MISSES	200000	None	"Instruction cache, streaming buffer and victim cache misses"
IDQ.MS_CYCLES	2000000	None	Cycles when uops are being delivered to Instruction Decode Queue (IDQ) w
IDQ_UOPS_NOT_DELIVERED.CORE	2000000	None	"Uops not delivered by the Frontend to the Backend of the machine, while t

Hardware Profiling : General Exploration (RandMem)



Memory Latency

LLC Miss:	0.045
LLC Hit:	0.293

A significant proportion of cycles is being spent on data fetches that miss in the L2 but hit in the LLC. This metric includes coherence penalties for shared data. If contested accesses or data sharing are indicated as likely issues, address them first. Otherwise, consider the same performance tuning as you would apply for an...

DTLB Overhead:	0.103

A significant proportion of cycles is being spent handling first-level data TLB misses. As with ordinary data caching, focus on improving data locality and reducing working-set size to reduce DTLB overhead. Additionally, consider using profile-guided optimization (PGO) to collocate frequently-used data on the same page. Try...

Contested Accesses:	0.083

There is a high number of contested accesses to cachelines modified by another core. Consider either using techniques suggested for other long latency load events (for example, LLC Miss) or reducing the contested accesses. To reduce contested accesses, first identify the cause. If it is synchronization, try increasing...

Data Sharing:	0.042

Pipeline S

Viewing 1 of 9 select

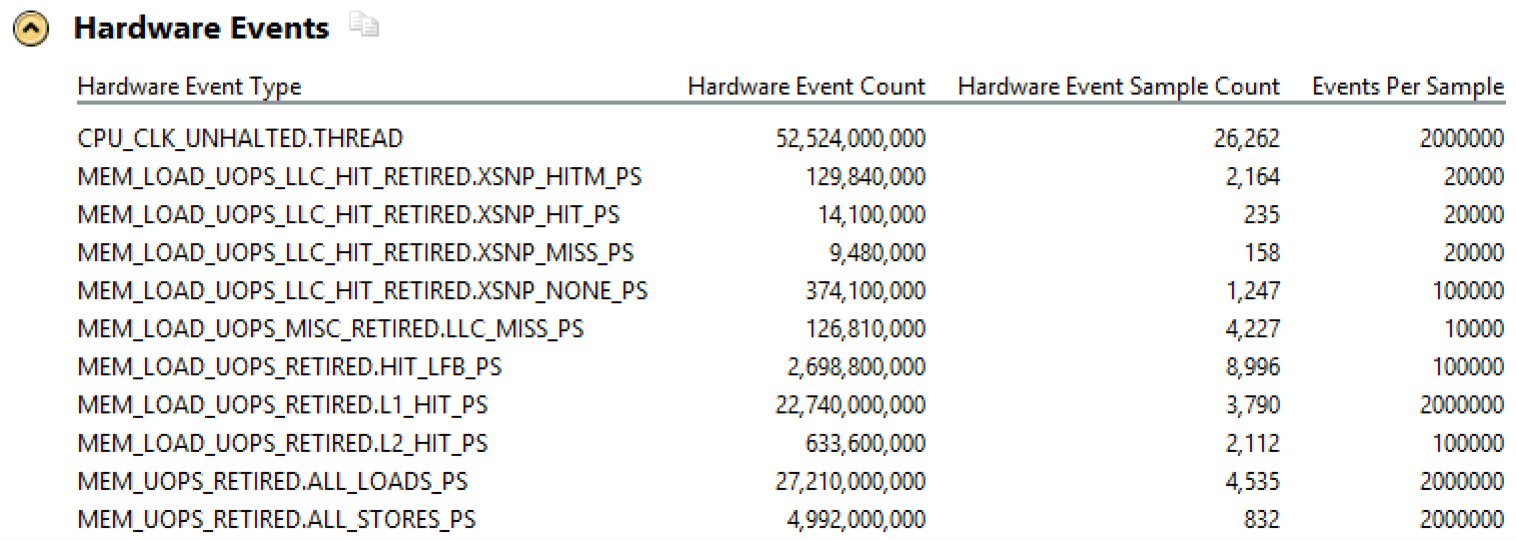
Loading data. Please wait.....

Function / Call Stack	CPU_CLK_... THREAD	INST_RET... ANY	CPI Rate	Re.	Bad Spec...	Back-end Bound							
						Memory Latency					Memory Replacements		
						LLC Miss	LLC Hit	DTLB Overh...	Contested ...	Data Sharing	L1D Replace...	L2 Replace...	LLC Replace...
runTests	12,315,353,293	16,971,401,939	0.726	0.062	0.575								
thread1TestRD	12,578,500,337	13,417,639,464	0.937	0.177	0.053								
thread1TestRW	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]								
thread2TestRD	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]								
thread2TestRW	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]								
threadTest2RW ← BaseT	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]								
[Unknown stack frame(s)]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]								
initptd	80,916,502	480,636,595	0.168	1.000	0.000								
FlsGetValue	95,051,238	378,870,881	0.251	1.000	0.000								
rand	57,421,589	303,842,475	0.189	1.000	0.000								
calculateMHz	137,556,008	246,991,954	0.557	0.262	0.000								
getptd	57,820,967	233,064,764	0.248	1.000	0.000								
WindowProc	151,660,191	135,197,944	1.122	0.018	1.000								
Selected 1 row(s):	12,578,500,337	13,417,639,464	0.937	0.177	0.053	0.056	0.248	0.194	0.000	0.002	0.349	0.250	0.000

A significant proportion of cycles is being spent on data fetches that miss in the L2 but hit in the LLC. This metric includes coherence penalties for shared data. If contested accesses or data sharing are indicated as likely issues, address them first. Otherwise, consider the same performance tuning as you would apply for an LLC-missing workload - reduce the data working set size, improve data access locality, consider blocking or otherwise partitioning your working set so that it fits in the L2, or better exploit hardware prefetchers. Consider using software prefetchers, but note that they can interfere with normal loads, potentially increasing latency, and that they increase pressure on the memory system.

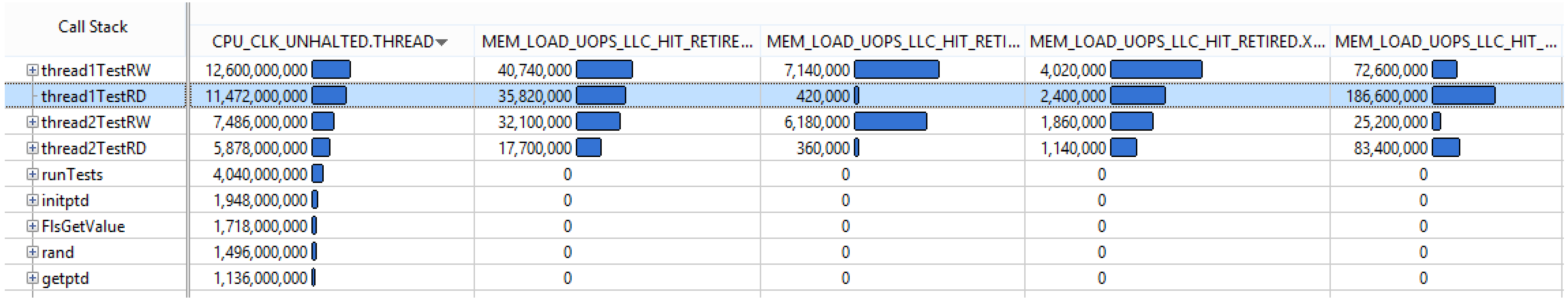
Threshold: ((((((26 * MEM_LOAD_UOPS_RETIRED.LLC_HIT_PS) + (43 * MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT_PS)) + (60 * MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS)) / CPU_CLK_UNHALTED.THREAD) > 0.2) * (CPU_CLK_UNHALTED.THREAD / query all CPU_CLK_UNHALTED.THREAD > 0.05))

Hardware Profiling : Memory Analysis (RandMem)



Hardware Events

Hardware Event Type	Hardware Event Count	Hardware Event Sample Count	Events Per Sample
CPU_CLK_UNHALTED.THREAD	52,524,000,000	26,262	2000000
MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HITM_PS	129,840,000	2,164	20000
MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_HIT_PS	14,100,000	235	20000
MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_MISS_PS	9,480,000	158	20000
MEM_LOAD_UOPS_LLC_HIT_RETIRED.XSNP_NONE_PS	374,100,000	1,247	100000
MEM_LOAD_UOPS_MISC_RETIRED.LLC_MISS_PS	126,810,000	4,227	10000
MEM_LOAD_UOPS_RETIRED.HIT_LFB_PS	2,698,800,000	8,996	100000
MEM_LOAD_UOPS_RETIRED.L1_HIT_PS	22,740,000,000	3,790	2000000
MEM_LOAD_UOPS_RETIRED.L2_HIT_PS	633,600,000	2,112	100000
MEM_UOPS_RETIRED.ALL_LOADS_PS	27,210,000,000	4,535	2000000
MEM_UOPS_RETIRED.ALL_STORES_PS	4,992,000,000	832	2000000



Call Stack	CPU_CLK_UNHALTED.THREAD	MEM_LOAD_UOPS_LLC_HIT_RETIRE...	MEM_LOAD_UOPS_LLC_HIT_RETI...	MEM_LOAD_UOPS_LLC_HIT_RETIRED.X...	MEM_LOAD_UOPS_LLC_HIT_...
⊞thread1TestRW	12,600,000,000	40,740,000	7,140,000	4,020,000	72,600,000
thread1TestRD	11,472,000,000	35,820,000	420,000	2,400,000	186,600,000
⊞thread2TestRW	7,486,000,000	32,100,000	6,180,000	1,860,000	25,200,000
⊞thread2TestRD	5,878,000,000	17,700,000	360,000	1,140,000	83,400,000
⊞runTests	4,040,000,000	0	0	0	0
⊞initptd	1,948,000,000	0	0	0	0
⊞FlsGetValue	1,718,000,000	0	0	0	0
⊞rand	1,496,000,000	0	0	0	0
⊞getptd	1,136,000,000	0	0	0	0

Other Features

- graphical tool for comparing analysis results
- user task API
    - annotate source code with user tasks
- remote agents for Linux
    - profile Linux applications remotely, using Windows GUI
- MPI Profiling (with Intel's MPI implementation)
- JIT Profiling
    - application must link against jitprofiling library and issue special events
    - LLVM supports the VTune API
- GPU Profiling (for OpenCL running on Intel GPUs)

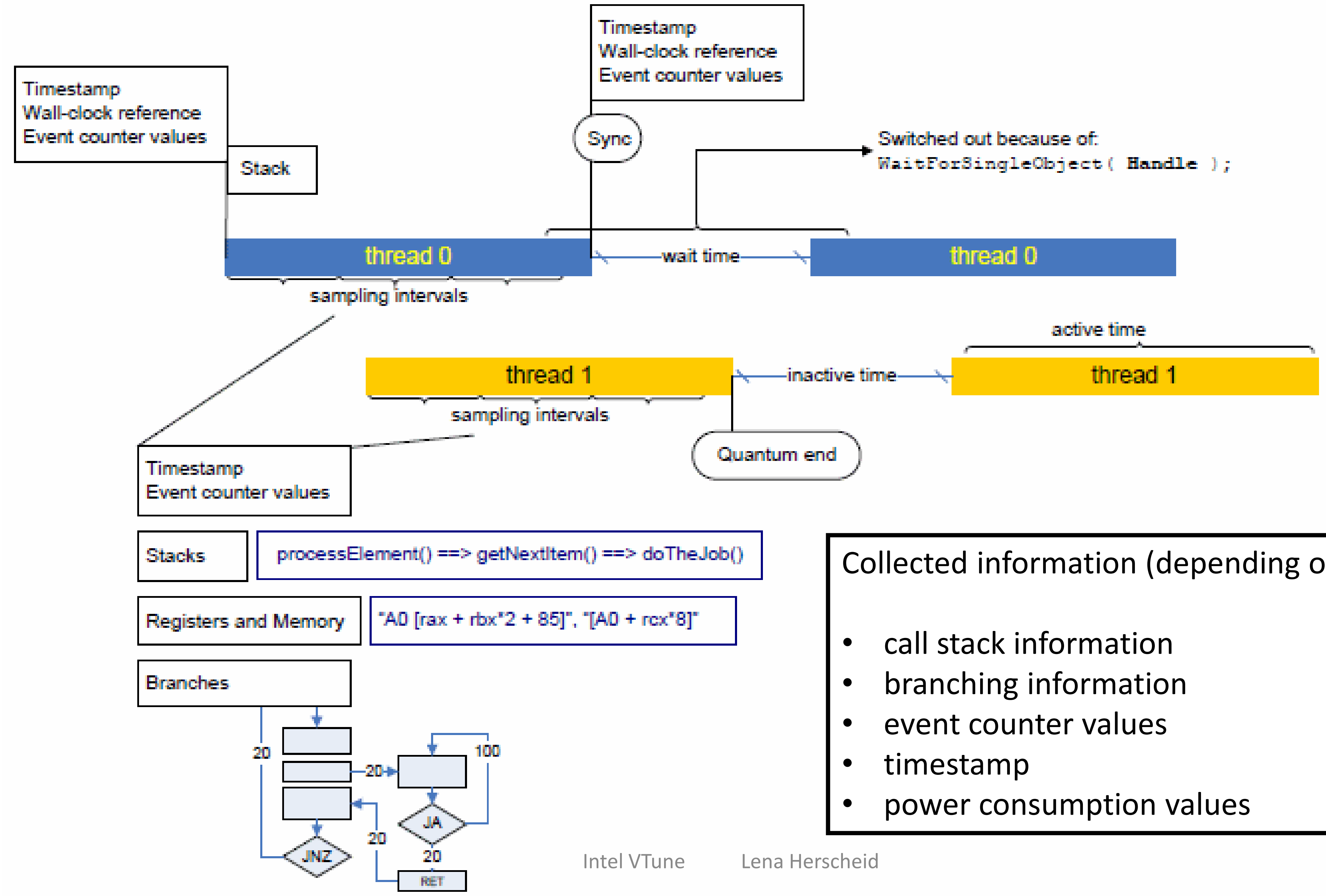
Agenda

- Overview
  - Sampling Approach
- Features
  - Algorithm Analysis
  - Hardware Profiling
- **How It Works**
  - User mode Sampling
  - Hardware Event Sampling
- Evaluation

User Mode Sampling

- time-driven approach
- embeds sampling library using **`LD_PRELOAD`**
  - setup timer for each thread
  - upon timer expiration, issue **`SIGPROF`**
  - interrupt process, collector handles signal by sampling stack information

Correlating Event-based sampling with thread quanta



Collected information (depending on configuration)

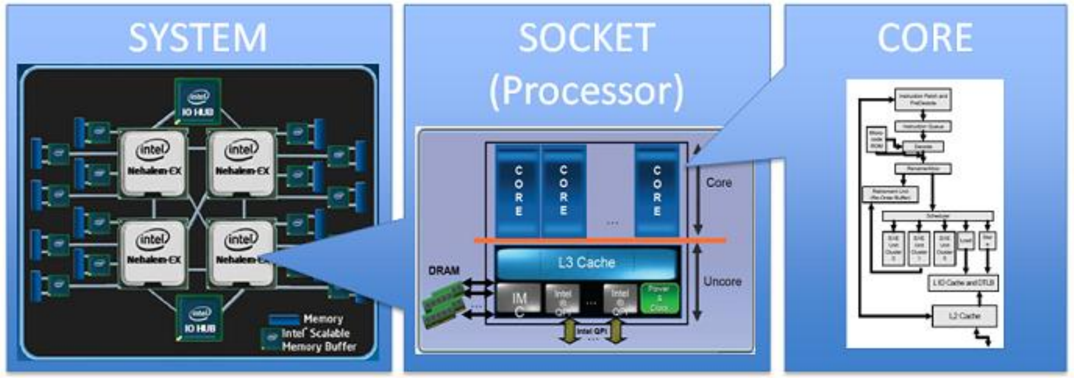
- call stack information
- branching information
- event counter values
- timestamp
- power consumption values

Hardware Event Based Sampling

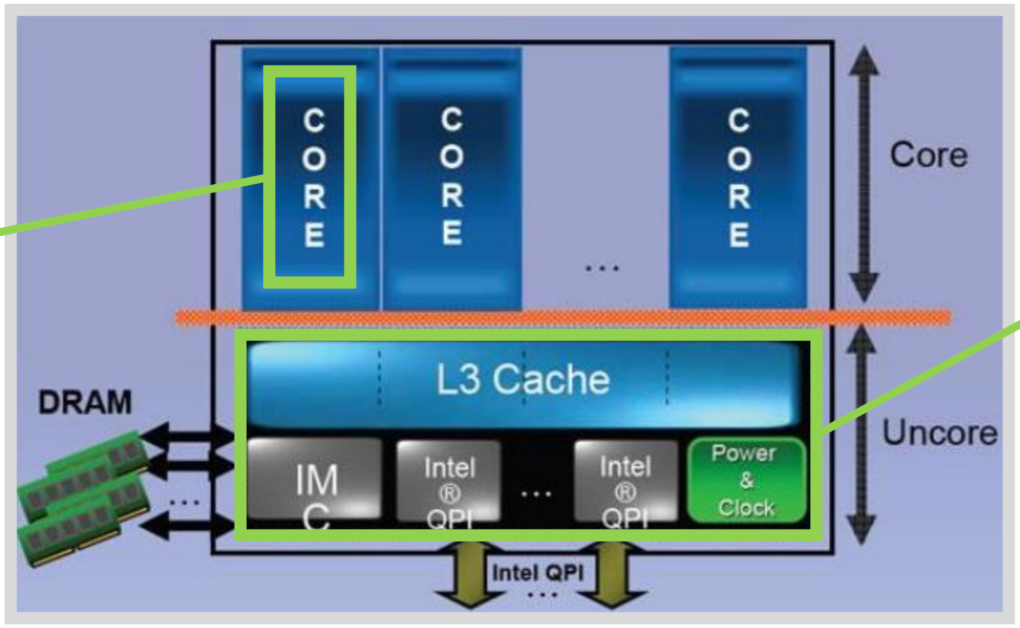
- event-driven
- implemented in a driver
- Performance Monitoring Unit (PMU) periodically interrupts processor
  - collect information after **SAV** (Sample After Value) events
    - automatic SAV calibration: needs additional run
    - *"If you set the Sample After value, specify a number that is sufficiently large. <u>A value that is too low increases the sample rate causing unpredictable results."</u>*
- hardware limit on the number of simultaneously sampled events (4)
  - for more events: round-robin multiplexing

Performance Monitoring Unit

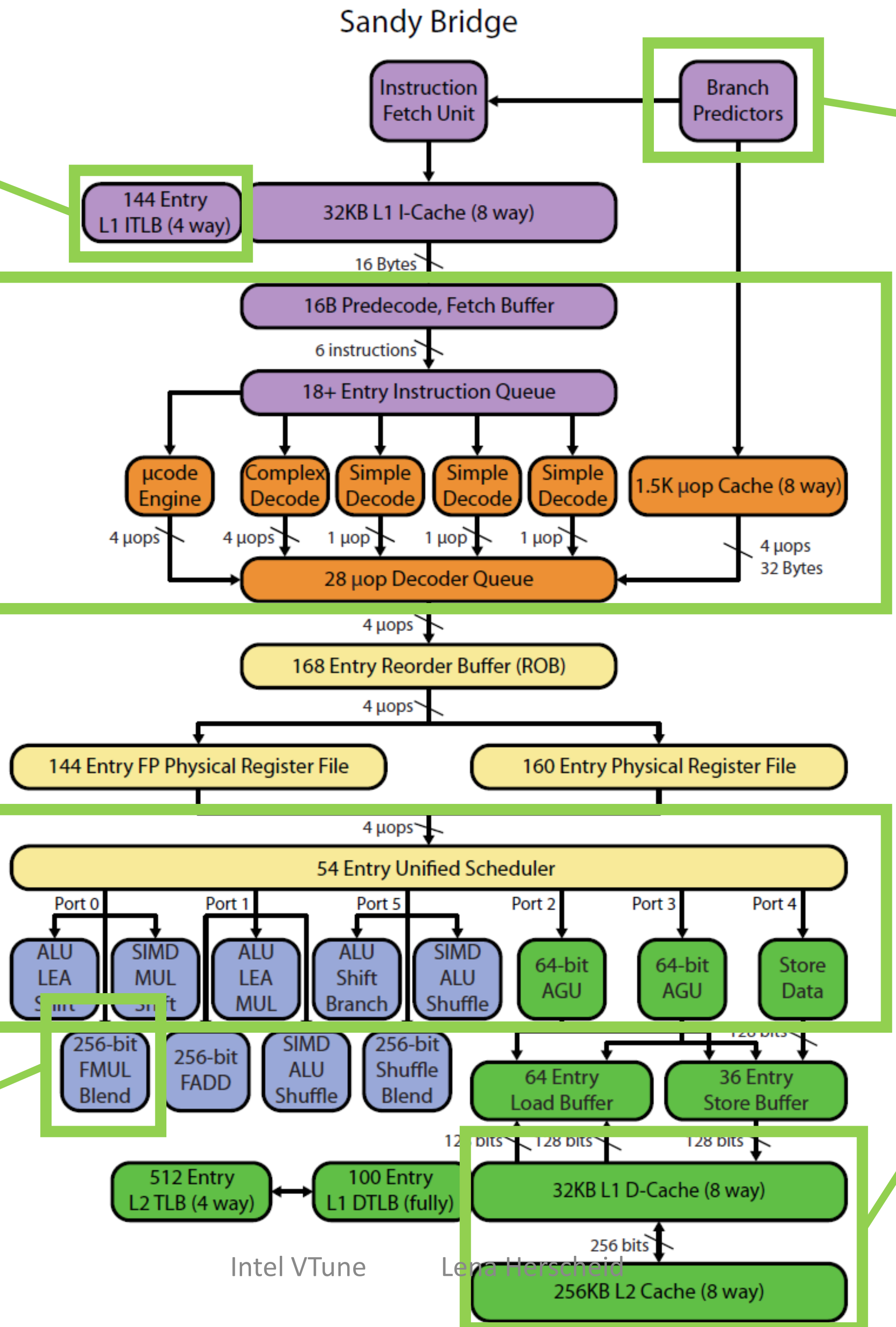
- one PMU per core
- one PMU in uncore region



- elapsed cycles
- L1, L2 cache events
- processed instructions
- …



- uncore bound
- QPI events
- L3 cache events
- memory controller events
- …



ITLB misses

branch mispredictions

front-end bound
cannot fetch & decode fast enough

back-end bound
no more operations accepted by back-end

FP assists
denormal values

L1/L2-cache bound
stalls due to cache misses

Agenda

- Overview
    - Sampling Approach
- Features
    - Algorithm Analysis
    - Hardware Profiling
- How It Works
    - User mode Sampling
    - Hardware Event Sampling
- **Evaluation**

Evaluation

Finalizing results

Loading 'tbs29524.tb6' file

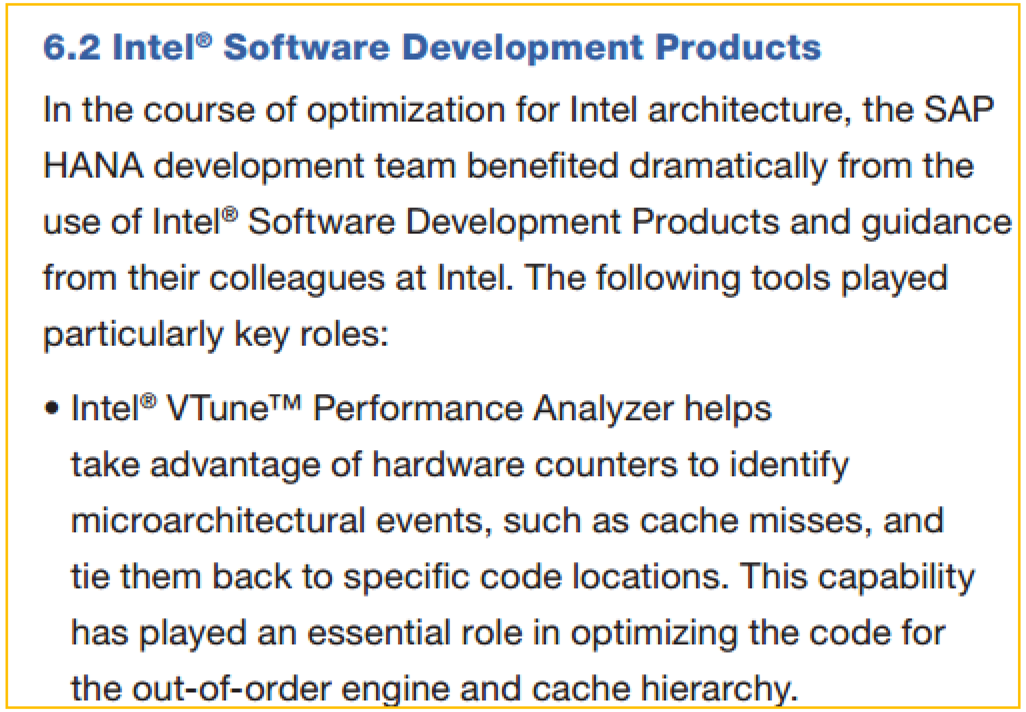
+ detailed Hardware information
+ tool-supported low-level performance tuning
+ low overhead during application run
+ good visualizations

– long post-processing
– the absolute numbers are not reliable (sampling errors)
  – influenced by system state and randomness
– single user license: 900$
– confusingly many features

Evaluation (ctd.)

Use VTune…

+ if you target an Intel architecture
    + and are willing to optimize for it
+ if the application is compute-intensive
+ if hardware awareness counts
    + strict performance requirements
    + low-level programming language



6.2 Intel® Software Development Products

In the course of optimization for Intel architecture, the SAP HANA development team benefited dramatically from the use of Intel® Software Development Products and guidance from their colleagues at Intel. The following tools played particularly key roles:

- Intel® VTune™ Performance Analyzer helps take advantage of hardware counters to identify microarchitectural events, such as cache misses, and tie them back to specific code locations. This capability has played an essential role in optimizing the code for the out-of-order engine and cache hierarchy.

Otherwise, use a simpler alternative

- Sampling Profilers:
  Very Sleepy for Windows, OProfile/gprof for Linux, AMD CodeAnalyst
- for just accessing PMUs: Intel Performance Counter (BSD license)
    - http://software.intel.com/en-us/articles/intel-performance-counter-monitor/#license

Sources

- Intel Materials: Documentation, Forum, Architecture Manual
    - http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe-2011-documentation
    - http://software.intel.com/en-us/forums/intel-vtune-amplifier-xe-and-vtune-performance-analyzer
    - http://www.intel.de/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf
- Benchmarks: http://www.roylongbottom.org.uk
- http://www.realworldtech.com/vtune/
- Survey of Software Monitoring and Profiling Tools http://www.cse.wustl.edu/~jain/cse567-06/ftp/sw_monitors2.pdf