Parallelism IV

John Cavazos

Dept of Computer & Information Sciences

University of Delaware

Lecture Overview

- Profiling
    - Intel® ISAT
    - Intel® Vtune™ Amplifier XE Evolution

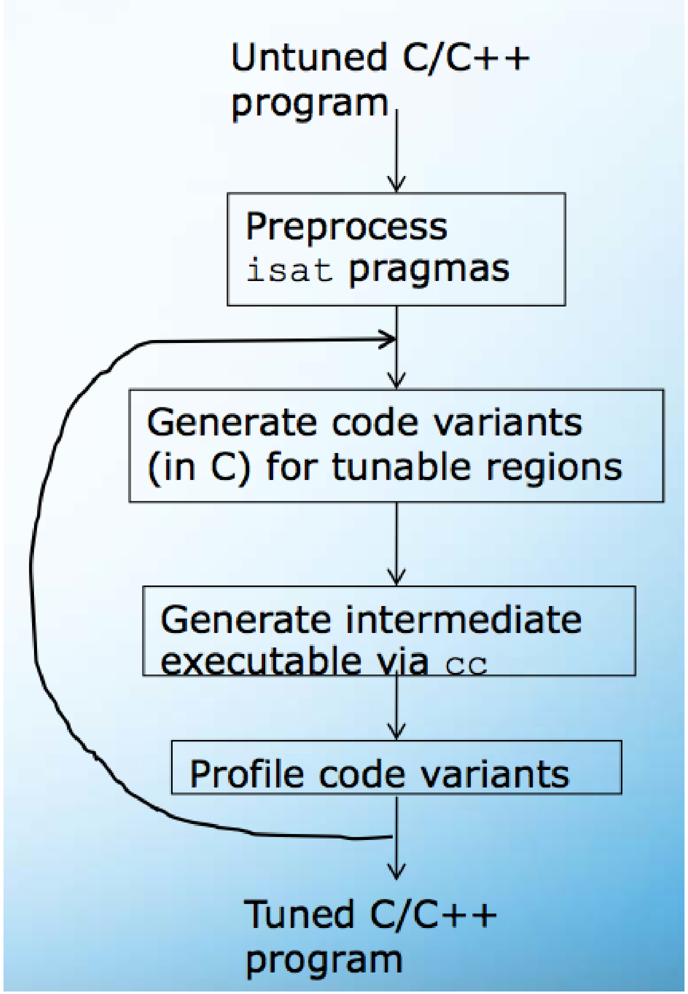
ISAT

- Intel® Software Autotuning Tool (ISAT)
  - http://software.intel.com/en-us/articles/intel-software-autotuning-tool/
  - Supports automatic search of the near-optimal values of important program parameters
    - Examples: OpenMP, Intel TBB (Threading Building Blocks)
  - Visualizes tuning results
  - Identifies regions that need tuning
  - Compiler independent

source: http://www.ckluk.org/ck/talks/cilkplus-tutorial-ppopp11.pdf

Quick Overview of ISAT

- Implemented in Python
  - Source-to-source translation
- Work with any C/C++ compilers
- Thread-safe implementation


Untuned C/C++ program
Preprocess isat pragmas
Generate code variants (in C) for tunable regions
Generate intermediate executable via cc
Profile code variants
Tuned C/C++ program

source: http://www.ckluk.org/ck/talks/cilkplus-tutorial-ppopp11.pdf

Example: OpenMP

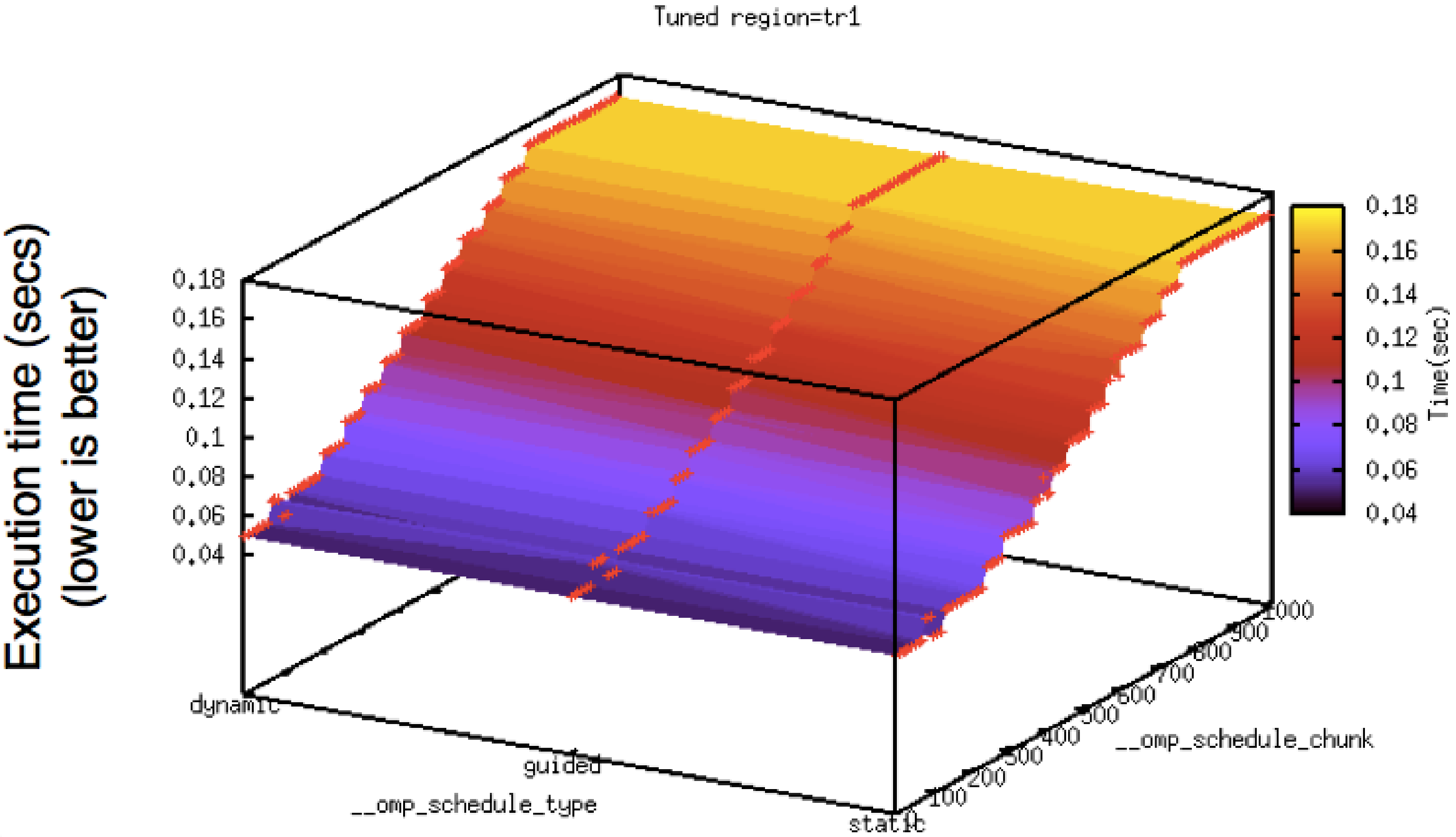
```
#pragma isat tuning
variable(@omp_schedule_type, [static, dynamic, guided])
variable(@omp_schedule_chunk, range(0, 1000, 100))
search(dependent)

#pragma omp parallel for
for(i = 0; i < N; i++)
    C[i] = A[i] * B[i];
```

- ISAT supports the tuning of the scheduling parameters in OpenMP loops (`parallel for` and `parallel do`)
- Two tunable parameters: schedule type and chunk size
- Specification of values
  - Explicit list using [ ] or range(begin, end, stride)
- Search scheme: dependent or independent

source: http://www.ckluk.org/ck/talks/cilkplus-tutorial-ppopp11.pdf

Visualization of Result

Tuning the schedule for `parellel_for`



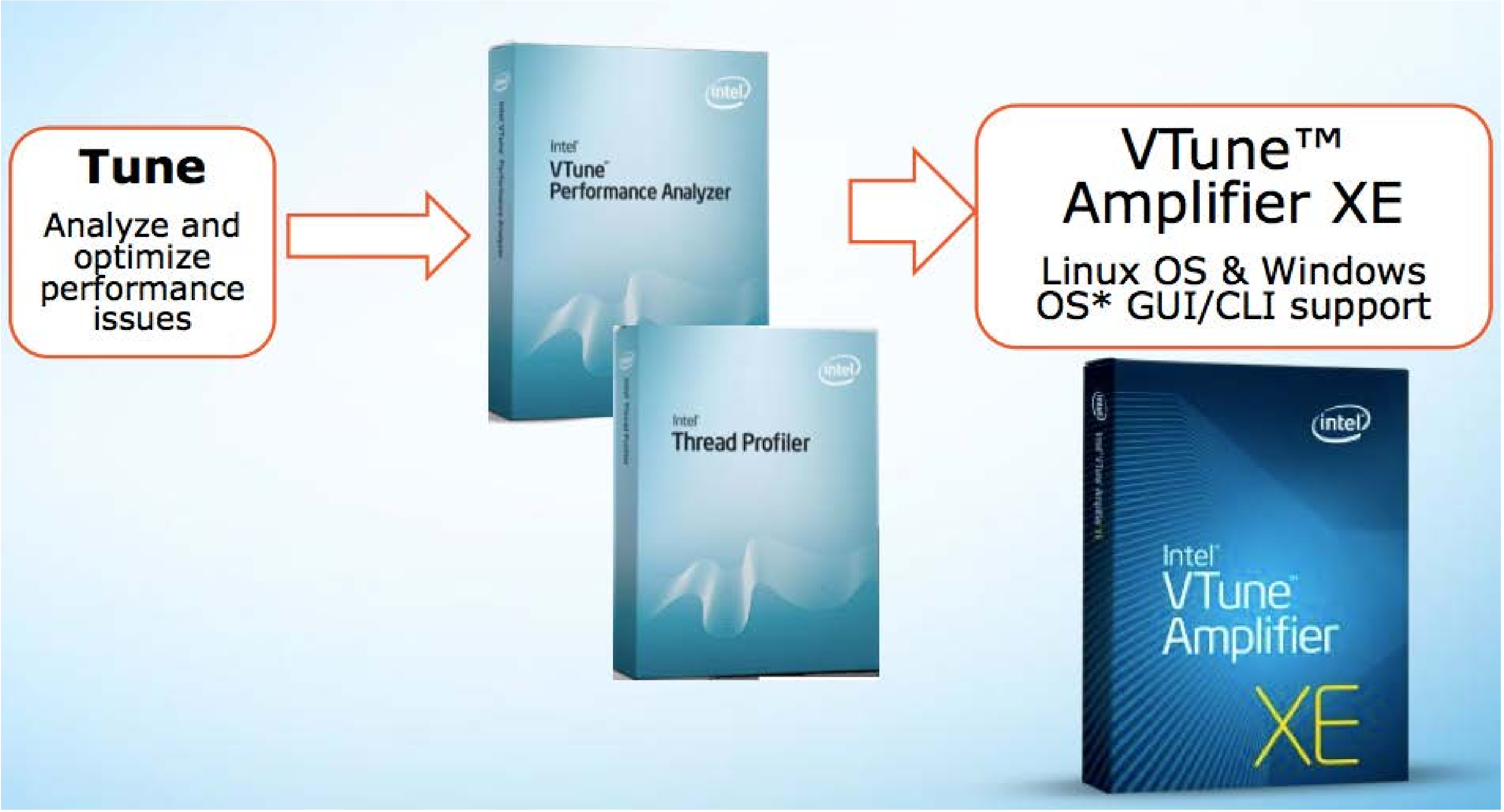
source: http://www.ckluk.org/ck/talks/cilkplus-tutorial-ppopp11.pdf

Future work for ISAT

- Support for tuning MPI programs
- Tune for Energy
  - Feasible on SandyBridge with its energy counters
- Advanced search strategies
  - Machine learning
- Search for the best permutations of compiler optimizations

source: http://www.ckluk.org/ck/talks/cilkplus-tutorial-ppopp11.pdf

Vtune™



Tune

Analyze and optimize performance issues

VTune™ Amplifier XE

Linux OS & Windows OS* GUI/CLI support

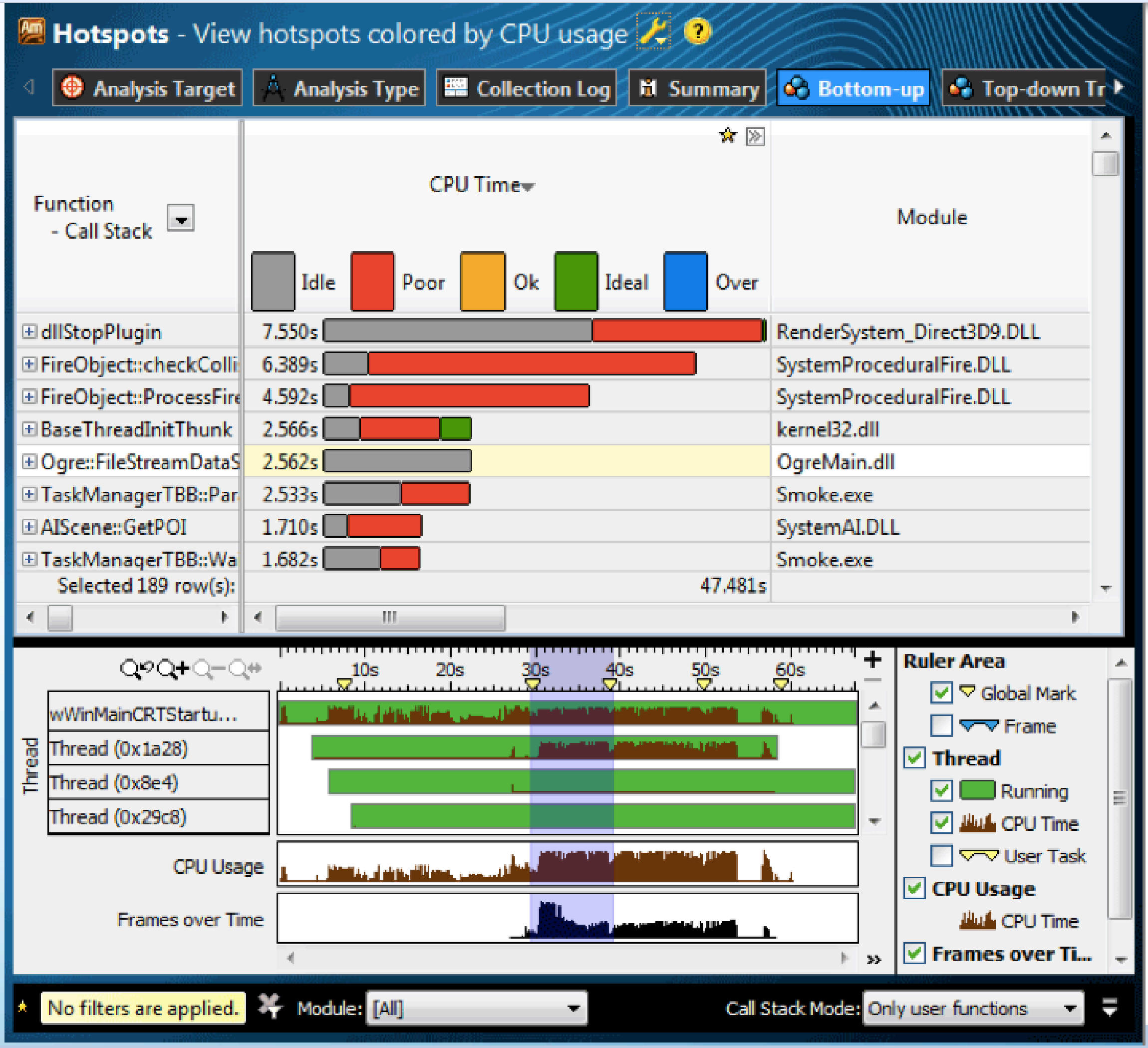
source: http://tools.zih.tu-dresden.de/2011/downloads/Intel_Using_VTuneAmplifierXE_for_HighPerformanceComputing_r0.9.pdf

Quick Overview of VTune™

- Performance Profiles
  - Hotspot
  - Hardware-Event Based Sampling (EBS)
- Locks and waits analysis
- Thread Profiling
  - Visualize thread interactions on timeline
  - Balance workloads
- Hardware event sampling

source: http://tools.zih.tu-dresden.de/2011/downloads/Intel_Using_VTuneAmplifierXE_for_HighPerformanceComputing_r0.9.pdf

Example: Hotspot Analysis



CPU Time legend: ▨ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over

Function - Call Stack	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	

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

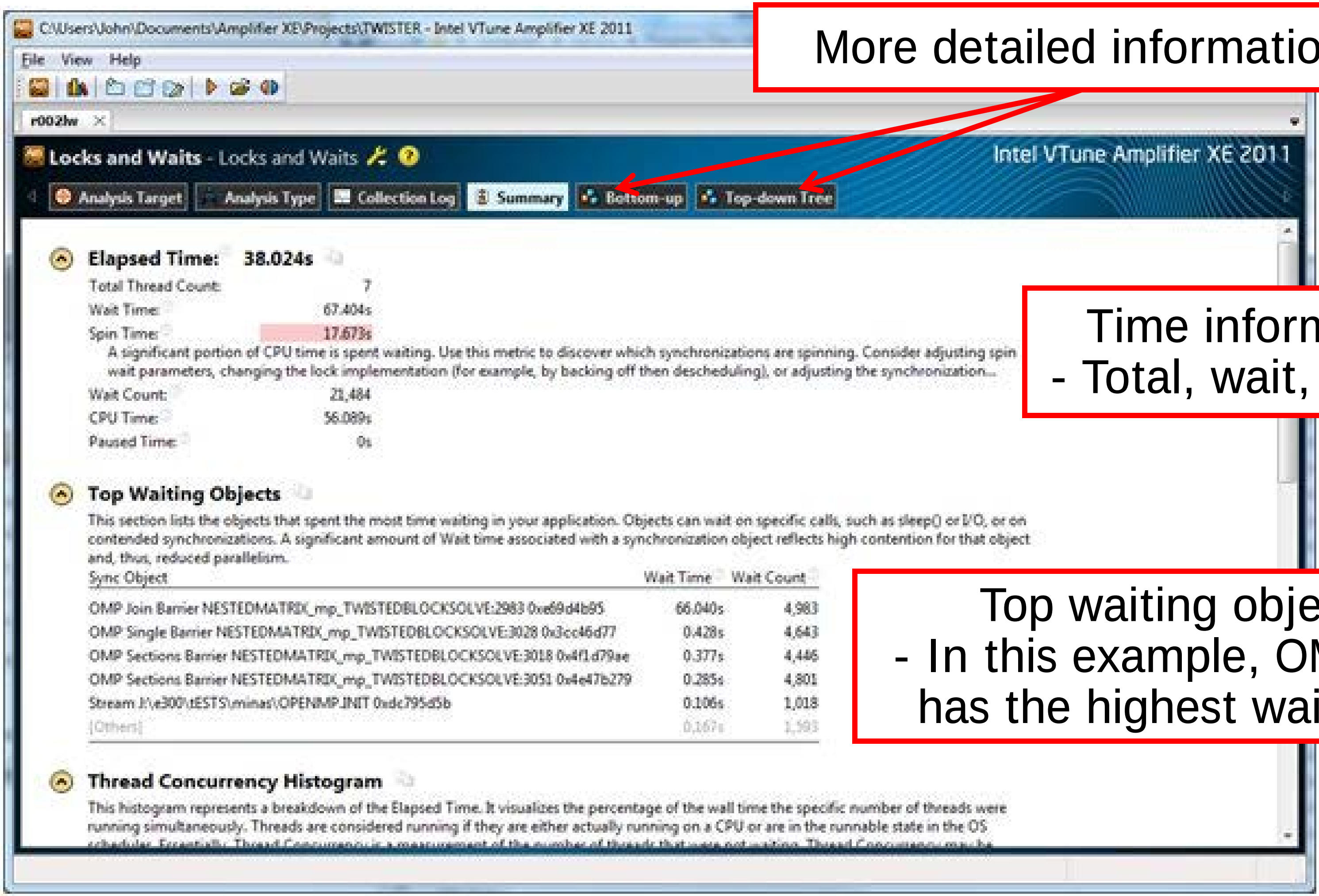
source: http://tools.zih.tu-dresden.de/2011/downloads/Intel_Using_VTuneAmplifierXE_for_HighPerformanceComputing_r0.9.pdf

Example: OpenMP Analysis

- Identifies where your application is waiting on synchronization objects or I/O operations
- Discovers how these waits effect your application performance

source: http://www.polyhedron.com/plain-VTune

Example: OpenMP Analysis


C:\Users\John\Documents\Amplifier XE\Projects\TWISTER - Intel VTune Amplifier XE 2011
File View Help
r002lw
Locks and Waits - Locks and Waits
Intel VTune Amplifier XE 2011
Analysis Target
Analysis Type
Collection Log
Summary
Bottom-up
Top-down Tree

Elapsed Time: 38.024s

Total Thread Count:	7
Wait Time:	67.404s
Spin Time:	17.673s

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

Wait Count:	21,484
CPU Time:	56.089s
Paused Time:	0s

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
OMP Join Barrier NESTEDMATRIX_mp_TWISTEDBLOCKSOLVE:2983 0xe69d4b95	66.040s	4,983
OMP Single Barrier NESTEDMATRIX_mp_TWISTEDBLOCKSOLVE:3028 0x3cc46d77	0.428s	4,643
OMP Sections Barrier NESTEDMATRIX_mp_TWISTEDBLOCKSOLVE:3018 0x4f1d79ae	0.377s	4,446
OMP Sections Barrier NESTEDMATRIX_mp_TWISTEDBLOCKSOLVE:3051 0x4e47b279	0.285s	4,801
Stream J:\e300\tESTS\minas\OPENMP.INIT 0xdc795d5b	0.106s	1,018
[Others]	0.167s	1,593

Thread Concurrency Histogram

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

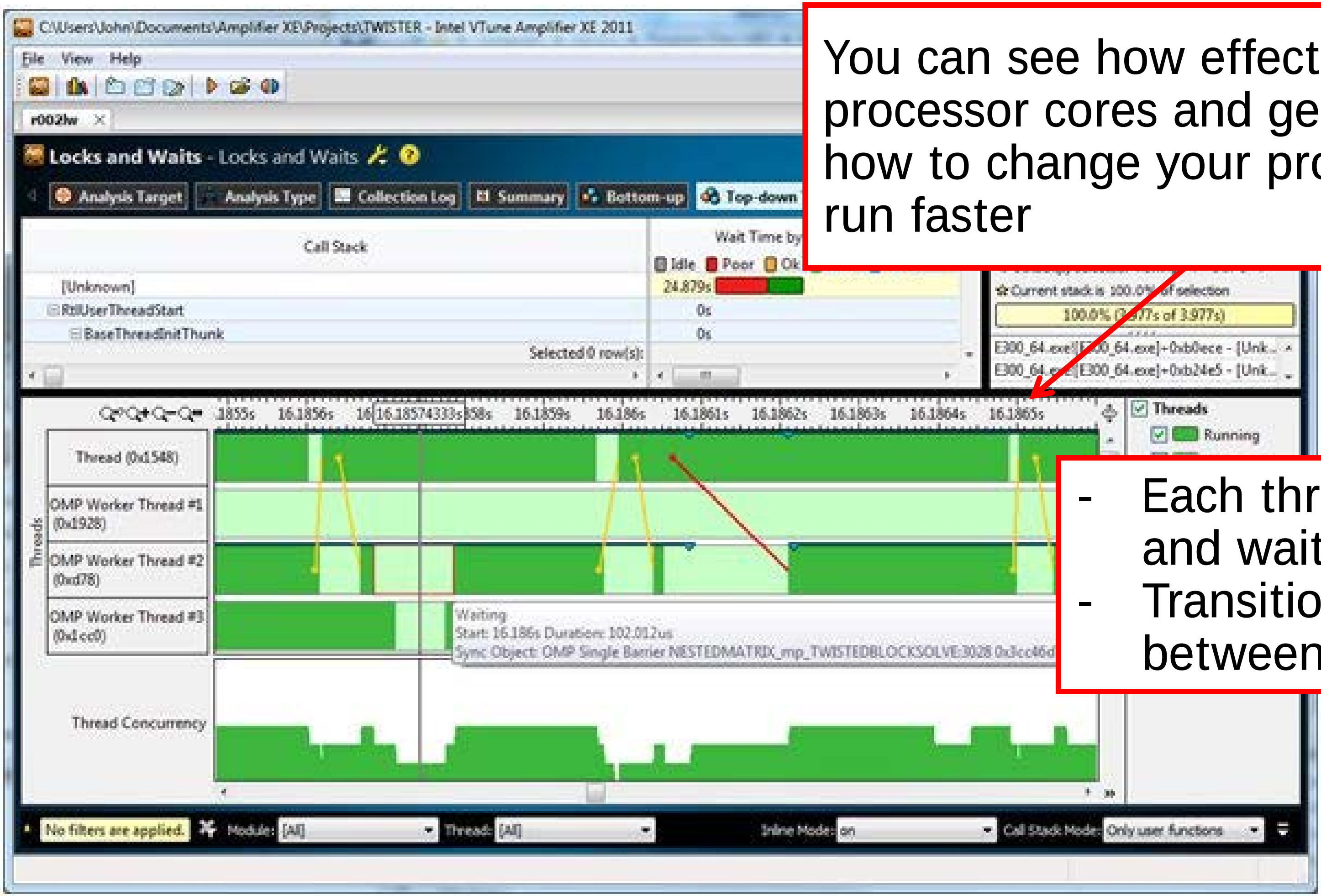
More detailed information

Time information
- Total, wait, spin, ...

Top waiting objects
- In this example, OMP Join has the highest wait time

source: http://www.polyhedron.com/plain-VTune

Example: OpenMP Analysis



C:\Users\John\Documents\Amplifier XE\Projects\TWISTER - Intel VTune Amplifier XE 2011

Call Stack	Wait Time by
[Unknown]	24.879s
RtlUserThreadStart	0s
BaseThreadInitThunk	0s

Selected 0 row(s):

Current stack is 100.0% of selection

100.0% (3.977s of 3.977s)

E300_64.exe![E300_64.exe]+0xb0ece - [Unk...

E300_64.exe![E300_64.exe]+0xb24e5 - [Unk...

Waiting
Start: 16.186s Duration: 102.012us
Sync Object: OMP Single Barrier NESTEDMATRIX_mp_TWISTEDBLOCKSOLVE:3028 0x3cc46d

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

You can see how effectively using processor cores and get insights how to change your programs to run faster

- Each thread's run and wait
- Transitions between threads

source: http://www.polyhedron.com/plain-VTune

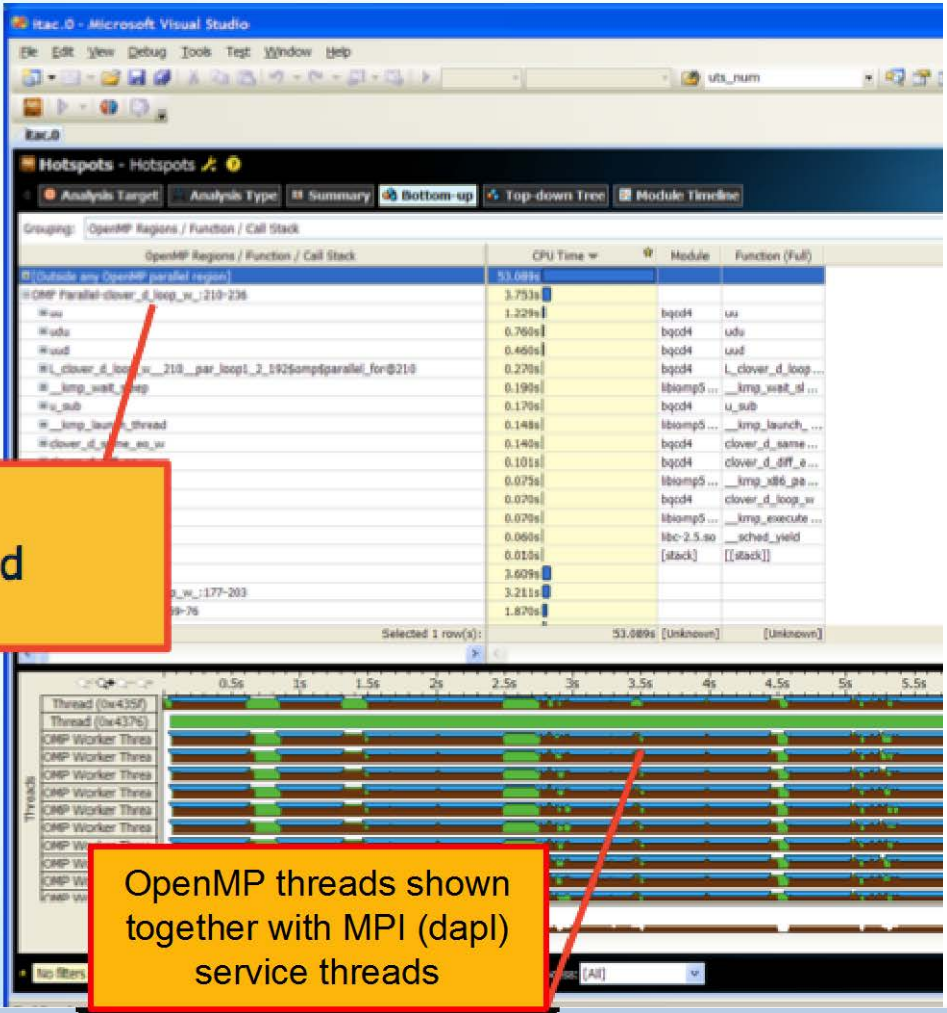
Example: MPI Analysis

- Use `amplex-cl` command to collect data and post-process results
  - Individual result directory will be created for each spawned process
- Example

  -
    ```
    > mpiexec -n 4 amplex-cl -r my_result  -collect hotspot -- my_app [my_app options]
    ```
  - This will create result directory for 4 processes named my_result.1 to my_result.4
- Using GUI viewer to analyze each result

source: http://www.polyhedron.com/plain-VTune

Example: MPI Analysis



Grouping: OpenMP Regions / Function / Call Stack

OpenMP Regions / Function / Call Stack	CPU Time	Module	Function (Full)
[Outside any OpenMP parallel region]	53.089s		
OMP Parallel-clover_d_loop_w_:210-236	3.753s		
uu	1.229s	bqcd4	uu
udu	0.760s	bqcd4	udu
uud	0.460s	bqcd4	uud
L_clover_d_loop_w__210__par_loop1_2_192$omp$parallel_for@210	0.270s	bqcd4	L_clover_d_loop...
__kmp_wait_sleep	0.190s	libiomp5 ...	__kmp_wait_sl ...
u_sub	0.170s	bqcd4	u_sub
__kmp_launch_thread	0.148s	libiomp5 ...	__kmp_launch_ ...
clover_d_same_eo_w	0.140s	bqcd4	clover_d_same ...
	0.101s	bqcd4	clover_d_diff_e ...
	0.075s	libiomp5 ...	__kmp_x86_pa ...
	0.070s	bqcd4	clover_d_loop_w
	0.070s	libiomp5 ...	__kmp_execute ...
	0.060s	libc-2.5.so	__sched_yield
	0.010s	[stack]	[[stack]]
	3.609s		
p_w_:177-203	3.211s		
69-76	1.870s		
Selected 1 row(s):	53.089s	[Unknown]	[Unknown]

OpenMP regions with routines called inside

OpenMP threads shown together with MPI (dapl) service threads

source: http://tools.zih.tu-dresden.de/2011/downloads/Intel_Using_VTuneAmplifierXE_for_HighPerformanceComputing_r0.9.pdf

Future Work for Vtune™

- Upcoming updates
  - Will be part of Intel® Cluster Studio XE
  - Will be supported on clusters

source: http://tools.zih.tu-dresden.de/2011/downloads/Intel_Using_VTuneAmplifierXE_for_HighPerformanceComputing_r0.9.pdf

Performance Counters

- Special purpose registers built into the processor microarchitecture
  - Monitor hardware-related activity within the computer

- Useful for performance analysis and tuning
  - Provide low-level information that can not be obtained with software profilers

source: http://pcsostres.ac.upc.edu/eitm/lib/exe/fetch.php/slides:pmc.pdf

Performance Counters

- Example of Events
  - Cache Line Access
  - Cache Access in Different Level (1,2, and 3)
  - Branch Prediction Related Events
  - Number of Total Cycles and Instructions
  - Number of Interrupt and Stall Cycles
  - …