



# Intel® VTune™ Amplifier XE

Dr. Michael Klemm  
Software and Services Group  
Developer Relations Division

SOFTWARE AND SERVICES

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

# Shameless Plug...

**Authors:** Alexander Supalov, Andrey Semin, Michael Klemm, Chris Dahnken

## **Table of Contents:**

Foreword by Bronis de Supinski (CTO LLNL)

Preface

Chapter 1: No Time to Read this Book?

Chapter 2: Overview of Platform Architectures

Chapter 3: Top-Down Software Optimization

Chapter 4: Addressing System Bottlenecks

Chapter 5: Addressing Application Bottlenecks:  
Distributed Memory

Chapter 6: Addressing Application Bottlenecks:  
Shared Memory

Chapter 7: Addressing Microarchitecture Bottlenecks

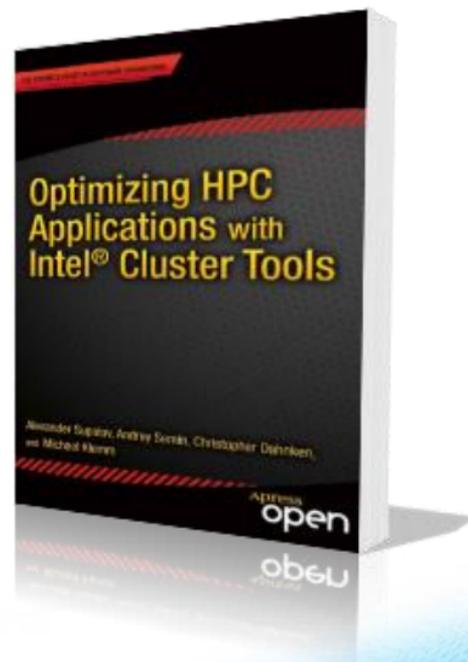
Chapter 8: Application Design Implications

ISBN-13 (pbk): 978-1-4302-6496-5

ISBN-13 (electronic): 978-1-4302-6497-2

**SOFTWARE AND SERVICES**

Apress  
**open**



**Order now at <http://www.clusterbook.info>**

# Contents

- Performance Analysis in a Nutshell
- Introduction to Intel® VTune™ Amplifier XE
  - Features and analysis types
  - Command Line Interface (CLI)
  - Graphical User Interface (GUI)
- VTune™ Typical HPC Workflow with MPI

# Performance Analysis in a Nutshell





**SOFTWARE AND SERVICES**



**SOFTWARE AND SERVICES**

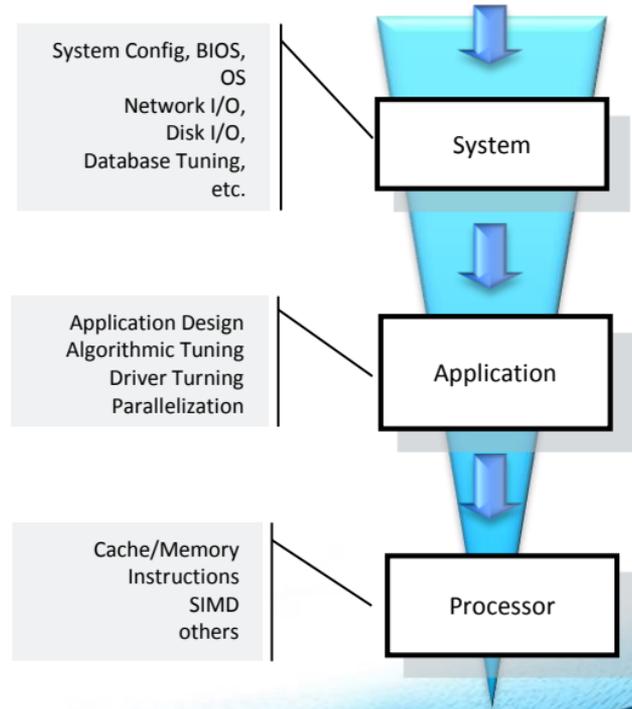
# The Software Optimization Process

- 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?

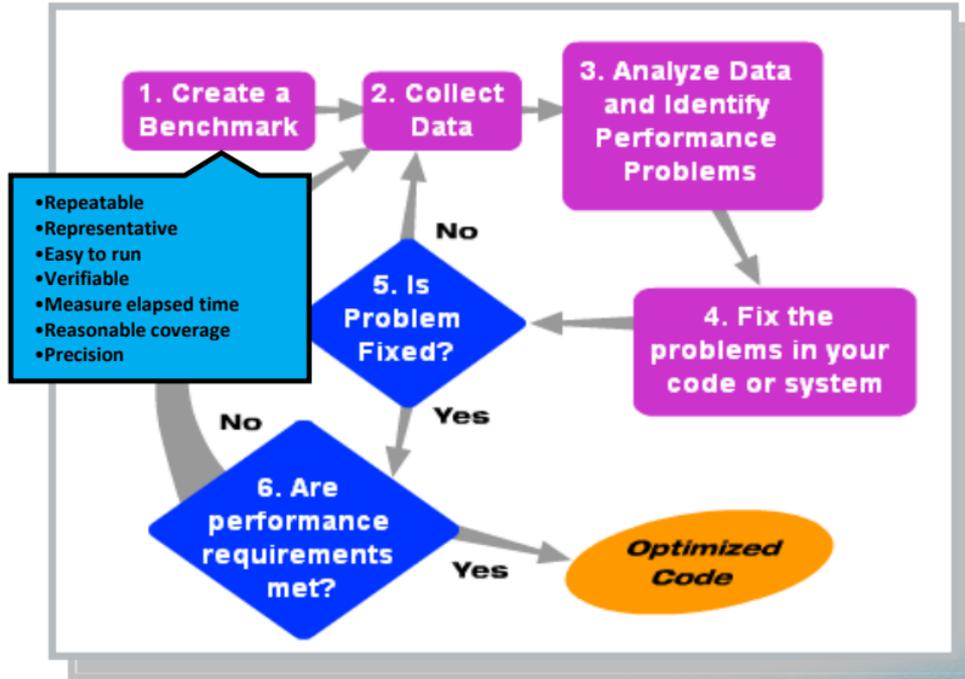
These methods aid developers to reach their performance requirements.

# Performance Analysis Methodology

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



# Performance Analysis Methodology

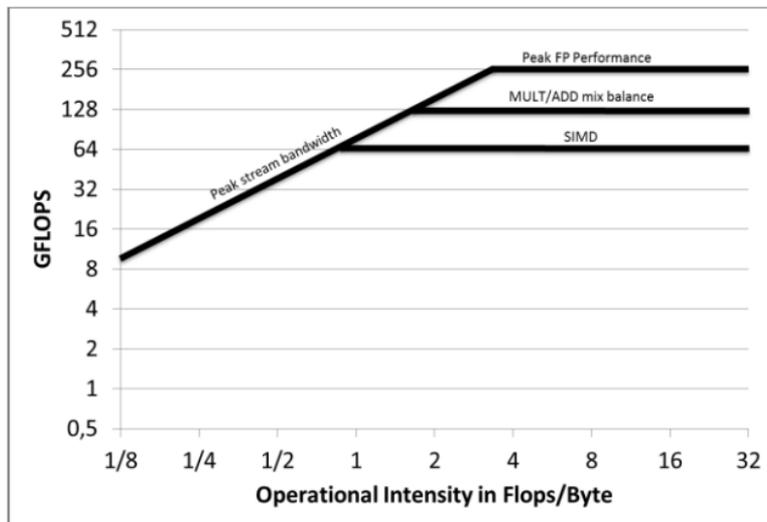


# When to Stop

- 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?

# Roofline Model

- The Roofline Model helps predict upper bounds for performance



# 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**

SOFTWARE AND SERVICES

# Intel® VTune™ Amplifier XE - Introduction



# Intel® VTune™ Amplifier

## Accurate data and meaningful analysis

- Accurate CPU, GPU and threading data
- OpenMP\* region efficiency analysis
- Powerful data analysis & filtering
- Data displayed on the source code
- Easy set-up, no special (re)compiles



<http://intel.ly/vtune-amplifier-xe>

# Intel® VTune™ Amplifier analysis

- Analysis separated into two steps
  - *Collect*: collection of analysis data
  - *Report*: compilation of reports based on the data collected
  - The use of GUI and/or CLI is supported in both steps
- Nonintrusive sampling based collection
  - No special (re)compiles needed
  - Statistical analysis to determine approximate behaviour

# Intel® VTune™ Amplifier preparations

- Run VTune on a “released/optimized” build
  - To view source code, compile with debugging symbols (i.e., `-g`)
- Use instrumented threading runtimes
  - OpenMP: Use Intel Dynamic Version of OpenMP
  - TBB: Define `TBB_USE_THREADING_TOOLS`
- For call stacks use a dynamic version of the C RTL to properly attribute system calls (i.e., do not use `-static`)

# Collecting data

Software Collector	Hardware Collector	
Uses OS interrupts	Uses the on chip Performance Monitoring Unit (PMU)	
Collects from a single process tree	Collect system wide or from a single process tree.	
~10ms default resolution	~1ms default resolution (finer granularity - finds small functions)	
Either an Intel® or a compatible processor	Requires a genuine Intel® processor for collection	
Call stacks show calling sequence	Optionally collect call stacks	
Works in virtual environments	Works in a VM only when supported by the VM (e.g., vSphere*, KVM)	
No driver required	Requires a driver	<ul style="list-style-type: none"><li>- Easy to install on Windows</li><li>- Linux requires root (or use default perf driver without stacks)</li></ul>

**No special recompiles - C, C++, C#, Fortran, Java, Assembly**

# A rich set of analysis types

Software Collector	Hardware Collector
<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 access contention, <b>memory analysis</b> , etc.
Any IA86 processor, any VM, no driver	Higher res., lower overhead, system wide

**No special recompiles - C, C++, C#, Fortran, Java, Assembly**

# VTune Linux\* improvements



## Previously added in 2015:

- Auto-rebuild Intel EBS driver
  - Does advanced analysis stop working when an OS update is installed?
  - The driver can be setup to auto-rebuild after an OS update.
- Auto-disable NMI watchdog
  - Tired of turning off NMI watchdog to run advanced EBS profiling?
  - NMI watchdog timer is automatically turned off and its state restored after the collection

## Added in 2016:

- Perf can collect stacks
- Use pre-installed perf driver
  - Intel EBS driver provides additional features not available in perf:
    - Uncore events
    - Multiple precise events
    - New events for the latest processors, even on an older OS

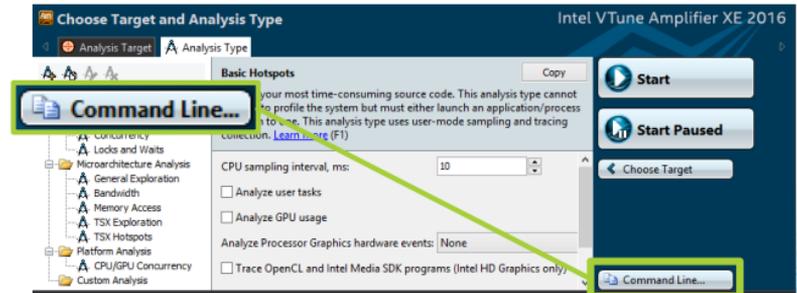
**Easier access to the on-chip PMU for advanced performance profiling**  
**SOFTWARE AND SERVICES**

# Intel® VTune™ Amplifier XE – Command Line Interface



# VTune command line interface (CLI)

- Command line tool **amplxe-cl**
  - Windows:  
C:\Program Files (x86)\Intel\VTune Amplifier XE \bin[32|64]\amplxe-cl.exe
  - Linux:  
/opt/intel/vtune\_amplifier\_xe/bin[32|64]/amplxe-cl
- Help: **amplxe-cl -help**
- GUI can be used to setup the command line
  - 1) Configure analysis in GUI
  - 2) Press "Command Line..."
  - 3) Copy & paste command



**Great for regression analysis – send results file to developer  
Command line results can also be opened in the GUI**

# VTune CLI: syntax

- VTune command line application `amplxe-cl`  
`amplxe-cl <-action> [-action-option] [-global-option]`  
`[[--] <target> [target-options]]`
  - `-action`: usually either *collect* or *report*
  - `-action-option`: modifies the behaviour of an action
  - `-global-option`: adjusts global settings
  - `<target>`: denotes the target application to profile

```
> amplxe-cl -collect advanced-hotspots -r result_dir -- ./app
```

# VTune CLI: collect

- Syntax:

`-c[collect] <analysis type> [-analysis-option]`

- The type of analysis defined with `analysis type`
  - Analysis type defines the set of available `analysis-option` modifiers or "knob"s
- Command line help with `-help` on each analysis type and available knobs

```
> ampxe-cl -help -collect # List analysis types available
> ampxe-cl -help -collect hotspots # List knobs for "hotspots"
```

# VTune CLI: collect - analysis types

- For HPC, the analysis types of interest are
  - **hotspots**: Identify hotspots, collect stacks and call tree information
  - **advanced-hotspots**: Identify hotspots, use hardware counters, do not collect stacks or call trees
  - **general-exploration**: Identify low-level hardware issues
  - **memory-access**: Identify memory access related issues and estimate bandwidth

# VTune CLI: collect - global modifiers

- A large number of global modifiers available
  - **[no-]auto-finalize**: [do not] finalize the result after the collection stops
  - **data-limit**: limit the amount of data collected. The default is 1GB, set to 0 for unlimited
  - **quiet**: limit the amount of information displayed
  - **-search-dir**: path where the binary and symbol files are stored
  - **-result-dir**: path where the result will be stored

# VTune CLI: report

- Syntax:

`-r[report] <report type> [-report-option]`

- The type of report defined with `report type`
- Report type defines the set of available `report-option` modifiers

- Command line help with `-help`

```
> ampxe-cl -help -report # List report types available
> ampxe-cl -help -report hotspots # Usage of "hotspots" report
```

- NOTE: using a GUI to view results is preferable

# VTune CLI: report - report types

- For HPC, the report types of interest are
  - **summary**: Report overall application performance
  - **hotspots**: Report CPU time for application
  - **hw-events**: Display the total number of hardware events
- A report is automatically based on the type of data collected!

# VTune CLI: report - global modifiers

- A large number of global modifiers available
  - **column**: Specify which columns to include or exclude
  - **filter**: Specify which data to include or exclude
  - **group-by**: Specify grouping in a report
  - **time-filter**: Specify which time range to include
  - **-source-search-dir**: path where the source code is stored
  - **-result-dir**: path where the result will be stored

# VTune CLI: example

- Collect **hotspots** of application **nbody**, store results to directory **nbody\_hs**

```
> amplxe-cl -collect hotspots -r nbody_hs -- ./nbody 262144
```

- View available columns in the result and then compile a **hotspots** report from specific columns

```
> amplxe-cl -report hotspots -r nbody_hs column=?  
> amplxe-cl -report hotspots -r nbody_hs -column="CPU  
Time:Self", "Source File"
```

# Intel® VTune™ Amplifier XE – Graphical User Interface



# VTune Graphical User Interface (GUI)

- Graphical tool **amplxe-gui**

- Windows:

- ```
C:\Program Files (x86)\Intel\VTune Amplifier XE \bin[32|64]\amplxe-gui.exe
```

- Linux:

- ```
/opt/intel/vtune_amplifier_xe/bin[32|64]/amplxe-gui
```

- Pure GUI workflow

- 1) Set up a project
- 2) Choose analysis
- 3) View analysis results



# VTune GUI: Welcome view

The screenshot shows the Intel VTune Amplifier XE 2016 Welcome view. The window title is "/home/mbycklin/intel/ampxe/projects/nbody - Intel VTune Amplifier". The main content area displays "Welcome to Intel VTune Amplifier XE 2016 Performance Profiler" with a "Getting Started" link. Below this, there are four main sections: "Current project: nbody", "Recent Projects:", "Recent Results:", and "Project / result management".

**Current project: nbody**

- ▶ [Basic Hotspots Analysis](#)
- ▶ [Advanced Hotspots Analysis](#)
- ▶ [Locks and Waits Analysis](#)
- ▶ [General Exploration Analysis](#)
- ▶ [New Analysis...](#)
- ▶ [Import Result...](#)
- ▶ [Configure Project](#)

**Recent Projects:**

- > [memkind\\_devel](#)
- > [ElmerFem\\_MPI](#)
- > [ElmerFem](#)
- > [Elmer\\_PoissonPFEM](#)

**Recent Results:**

- > [r004hs \[memkind\\_devel\]](#)
- > [r003hs \[memkind\\_devel\]](#)
- > [r002hs \[memkind\\_devel\]](#)
- > [r001hs \[memkind\\_devel\]](#)
- > [r000hs \[memkind\\_devel\]](#)

**Project / result management**

- ▶ [New Project...](#)
- ▶ [Open Project...](#)
- ▶ [Open Result](#)

**Callouts:**

- Open/close project panel:** Points to the project management icon in the top toolbar.
- Analysis types available for the current project:** Points to the "Current project: nbody" list.
- Recent projects:** Points to the "Recent Projects:" list.
- Recent results:** Points to the "Recent Results:" list.
- Project / result management:** Points to the "New Project...", "Open Project...", and "Open Result" buttons.

# VTune GUI: choose analysis view

Available analysis types

Choose target and Analysis Type

Analysis Target Analysis Type

**Basic Hotspots** Copy

Identify your most time-consuming source code. This analysis type cannot be used to profile the system but must either launch an application/process or attach to one. This analysis type uses user-mode sampling and tracing collection. [Learn more](#) (F1)

CPU sampling interval, ms: 1

Analyze user tasks

Analyze GPU usage (Intel HD Graphics only)

Analyze Processor Graphics hardware events: None

Trace OpenCL and Intel Media SDK programs (Intel HD Graphics only)

Details

Start

Start Paused

Choose Target

Algorithm Analysis

- Basic Hotspots
- Advanced Hotspots
- Concurrency
- Locks and Waits

Microarchitecture Analysis

- General Exploration
- Memory Access
- TSX Exploration
- TSX Hotspots

Platform Analysis

- CPU/GPU Concurrency
- Custom Analysis

Command Line...

Different ways to start the analysis

Get analysis command line

# VTune GUI: analysis view

Adjust data grouping

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

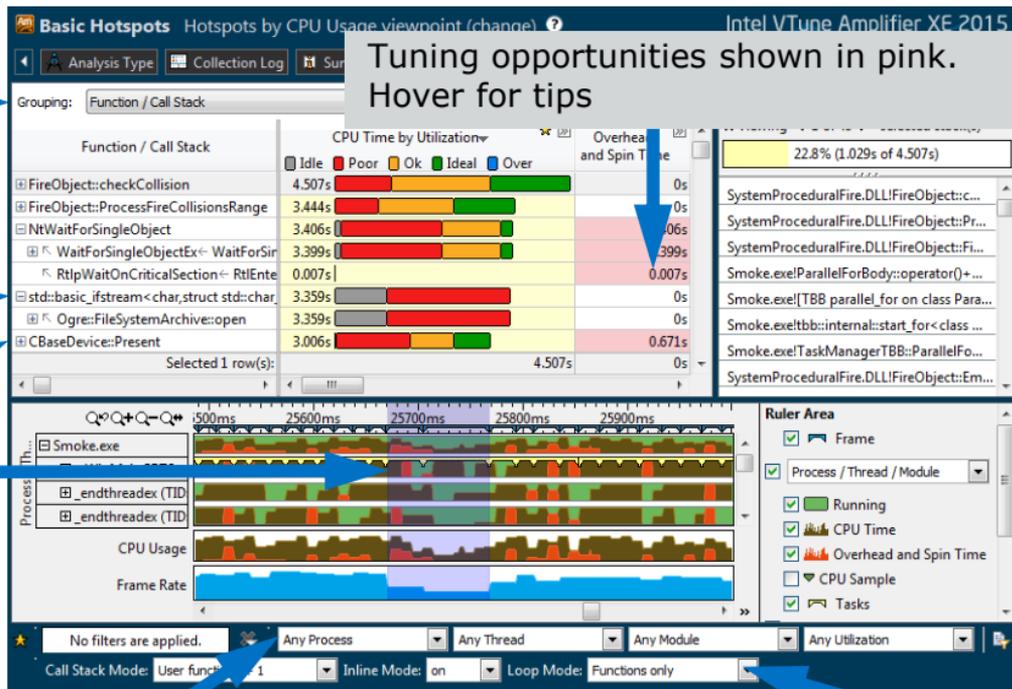
Double click function to view source

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

Tuning opportunities shown in pink. Hover for tips



SOFTWARE AND SERVICES

Filter by process & other controls

Function / loop mode

# VTune GUI: view source code profile

View source / assembly or both

CPU Time

Right click for instruction reference manual

The screenshot shows the Intel VTune Amplifier XE 2015 GUI. The main window is titled "Hotspots by CPU Usage viewpoint (change)". It has a menu bar with "Analysis Type", "Collection Log", "Summary", "Bottom Up", "Caller/Callee", "Top-down Tree", "Tasks and Frames", and "grid.cpp". Below the menu bar are tabs for "Source" and "Assembly". The "Source" tab is active, showing a C++ code snippet with a "CPU Time" column. The "Assembly" tab is also visible, showing a table of assembly instructions with columns for "Address", "Source Line", "Assembly", and "CPU Time". A vertical "Heat Map" bar is located between the source and assembly views. Annotations with blue arrows point to various parts of the GUI: one points to the "Source" tab, another to the "CPU Time" column, a third to the "Assembly" tab, a fourth to the "Heat Map" bar, and a fifth to the assembly instruction table. A text box on the left says "Quick assembly navigation: Select source to highlight assembly". A text box at the bottom right says "Click jump to scroll assembly".

Quick assembly navigation:  
Select source to highlight assembly

Source Line	Source	CPU Time: Total ...	Address	Sour... Line	Assembly	CPU Time: Total ...
579	cur = g->cells[voxindex];	0.310s	0x418b6d	580	cmp dword ptr [ebp-0x190], 0x	0.120s
580	while (cur != NULL) {	0.499s	0x418b74	580	jz 0x418b66 <Block 58>	0.379s
581	if (ry->mbox[cur->obj->id]	7.795s	0x418b76		Block 54:	
582	ry->mbox[cur->obj->id] = r	0.547s	0x418b76	581	mov edx, dword ptr [ebp-0x190]	0.090s
583	cur->obj->methods->interse	1.769s	0x418b7c	581	mov eax, dword ptr [edx+0x4]	0.020s
584	}		0x418b7f	581	mov ecx, dword ptr [eax]	3.853s
585	cur = cur->next;	0.568s	0x418b81	581	mov edx, dword ptr [ebp+0xc]	2.500s
586	}	0.070s	0x418b84	581	mov eax, dword ptr [edx+0x10]	0.030s
587	curvox.z += step.z;	0.070s	0x418b87	581	mov edx, dword ptr [ebp+0xc]	
588	if (ry->maxdist < tmax.z    cu	0.100s	0x418b8a	581	mov eax, dword ptr [eax+ecx*4]	0.040s
			0x418b8d	581	cmp eax, dword ptr [edx+0xc]	1.262s
			0x418b90	581	jz 0x418b66 <Block 57>	
			0x418b92		Block 55:	
			0x418b92	582	mov ecx, dword ptr [ebp-0x190]	0.331s
			0x418b98	582	mov edx, dword ptr [ebp+0x4]	0.116s

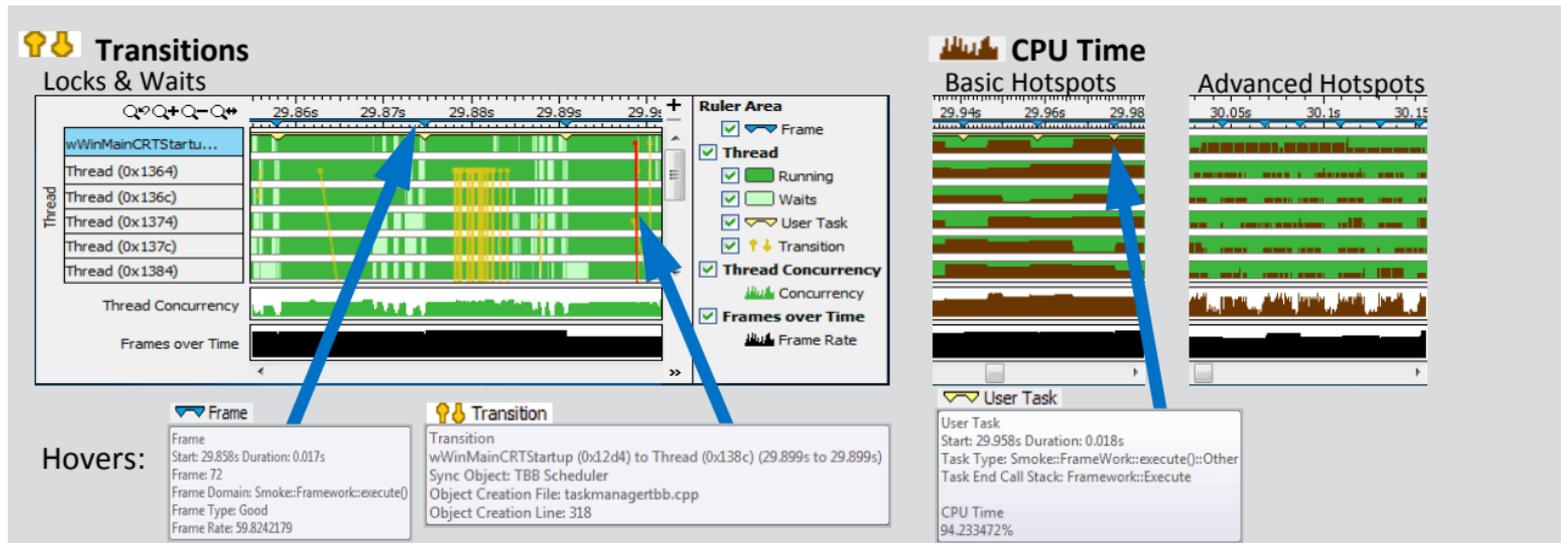
Selected 1 row(s): 7.795s

Highlighted 9 row(s): 7.795s

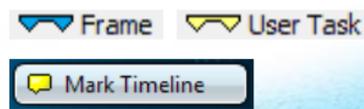
Scroll Bar "Heat Map" is an overview of hot spots

Click jump to scroll assembly

# VTune GUI: view thread timeline

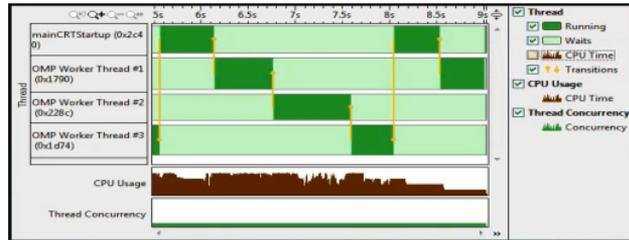


- Optional: Use API to mark frames and user tasks
- Optional: Add a mark during collection

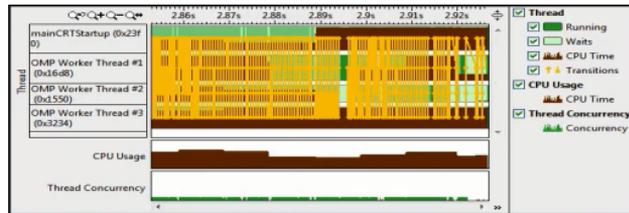


# VTune GUI: find concurrency issues

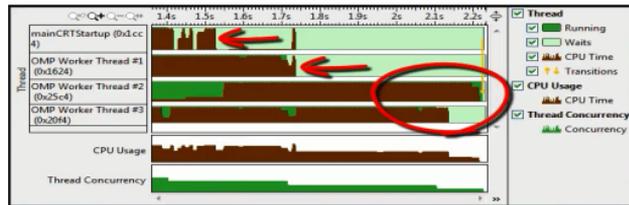
Coarse grain locks



High lock contention



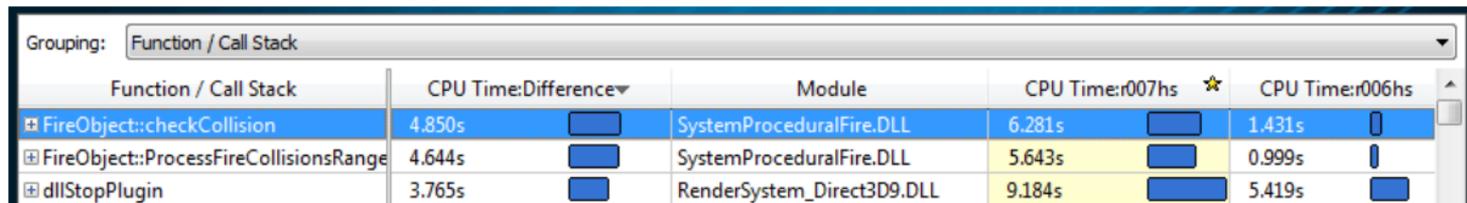
Load imbalance



Low concurrency

# VTune GUI: results comparison

- Quickly identify cause of regressions.
  - Run a command line analysis daily
  - Identify the function responsible so you know who to alert
- Compare 2 optimizations – What improved?
- Compare 2 systems – What didn't speed up as much?



The screenshot shows the VTune GUI interface with a table of CPU time differences. The table has five columns: Function / Call Stack, CPU Time:Difference, Module, CPU Time:r007hs, and CPU Time:r006hs. The 'Grouping' dropdown is set to 'Function / Call Stack'. The table contains three rows of data, with the first two rows highlighted in blue and the third row highlighted in yellow.

Function / Call Stack	CPU Time:Difference	Module	CPU Time:r007hs	CPU Time:r006hs
FireObject::checkCollision	4.850s	SystemProceduralFire.DLL	6.281s	1.431s
FireObject::ProcessFireCollisionsRange	4.644s	SystemProceduralFire.DLL	5.643s	0.999s
dllStopPlugin	3.765s	RenderSystem_Direct3D9.DLL	9.184s	5.419s

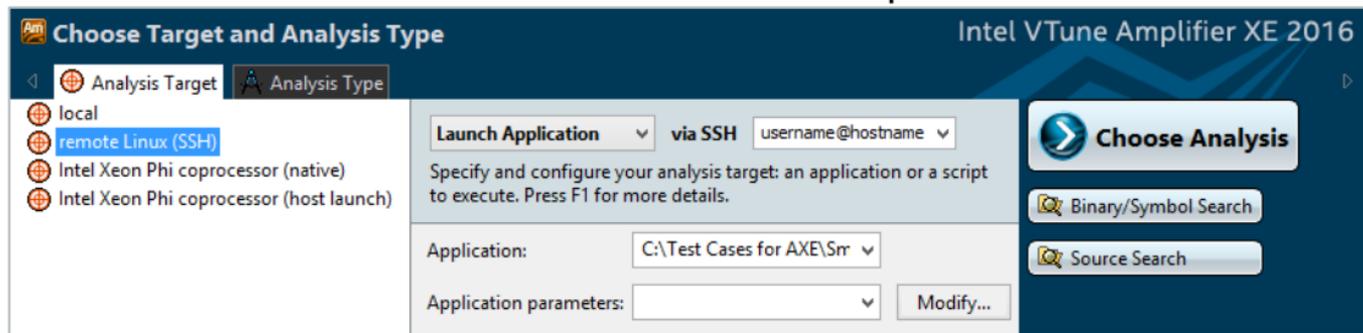
# VTune GUI: remote data collection

- Interactive analysis

- 1) Configure SSH to a remote Linux\* target
- 2) Choose and run analysis with the GUI

- Command line analysis

- 1) Run command line remotely on Windows\* or Linux\* target
- 2) Copy results back to host and open in GUI



SOFTWARE

Conveniently use your local UI to analyze remote systems

# VTune GUI: MPI+OpenMP analysis (1/3)

Advanced Hotspots Hotspots viewpoint (change) Intel VTune Amplifier XE 2015

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

### OpenMP Analysis. Collection Time: 14.490

Serial Time (outside any parallel region): 4.020s (27.7%)

Serial time of your application is high. It directly impacts application Elapsed Time and scalability. Explore options for parallelization, algorithm or microarchitecture tuning of the serial part of the application.

Parallel Region Time: 10.469s (72.3%)

Estimated Ideal Time: 7.115s (49.1%)

Potential Gain: 3.354s (23.1%)

The time wasted on load imbalance or parallel work arrangement is significant and negatively impacts the application performance and scalability. Explore OpenMP regions with the highest metric values. Make sure the workload of the regions is enough and the loop schedule is..

### Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	Potential Gain (%)	Elapsed Time
<a href="#">conj_grad_omp\$parallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:514:695</a>	3.294s 22.7%	10.208s
<a href="#">MAIN_omp\$parallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:185:231</a>	0.059s 0.4%	0.200s
<a href="#">MAIN_omp\$parallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:339:345</a>	0.001s 0.0%	0.001s
<a href="#">MAIN_omp\$parallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:361:365</a>	0.001s 0.0%	0.001s
<a href="#">MAIN_omp\$parallel:24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:263:269</a>	0.000s 0.0%	0.000s
[Others]	0.000s 0.0%	0.000s

- An overview of the "Summary" pane

SOFTWARE AND SERVICES

# VTune GUI: MPI+OpenMP analysis (2/3)

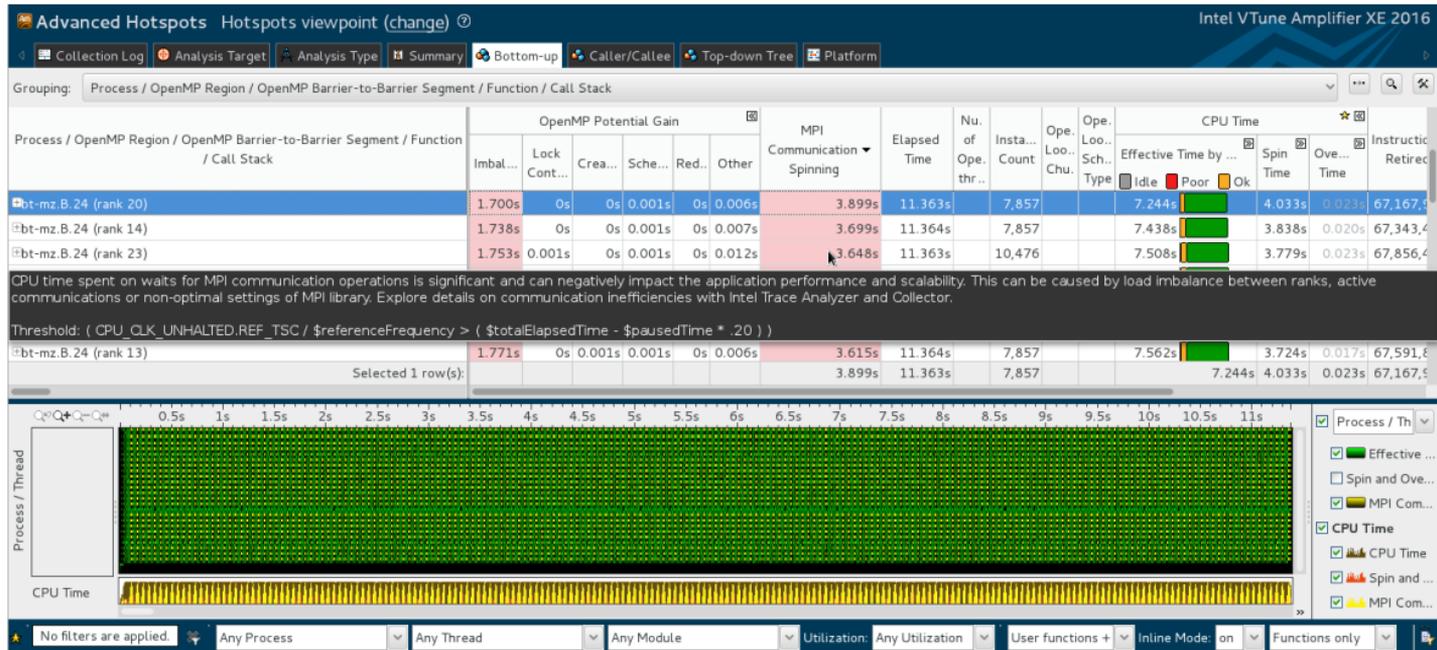
## 🔍 Top OpenMP Processes by MPI Communication Spin Time 📄

This section lists processes sorted by MPI Communication Spin time. The lower MPI Communication Spin time, the more a process was on a critical path of MPI application execution. Explore OpenMP efficiency metrics by MPI processes laying on the critical path

Process	PID	MPI Communication Spinning <sup>Ⓢ</sup>	(%) <sup>Ⓢ</sup>	OpenMP Potential Gain <sup>Ⓢ</sup>	(%) <sup>Ⓢ</sup>	Serial Time <sup>Ⓢ</sup>	(%) <sup>Ⓢ</sup>
<a href="#">bt-mz.B.4 (rank 2)</a>	37954	4.428s	16.8%	11.234s	42.7%	5.410s	20.6%
<a href="#">bt-mz.B.4 (rank 0)</a>	37948	5.236s	19.9%	9.953s	37.8%	6.542s	24.9%
<a href="#">bt-mz.B.4 (rank 3)</a>	37957	5.274s	20.0%	10.329s	39.3%	6.384s	24.3%
<a href="#">bt-mz.B.4 (rank 1)</a>	37951	6.196s	23.5%	9.183s	34.9%	7.513s	28.6%

- VTune reports contain MPI communication spinning metrics for Intel MPI
- Showing OpenMP metrics and serial time per process sorting by processes laying on critical path of MPI execution

# VTune GUI: MPI+OpenMP analysis (3/3)



- “MPI communication-aware” grid and Process/Thread scalable timeline view

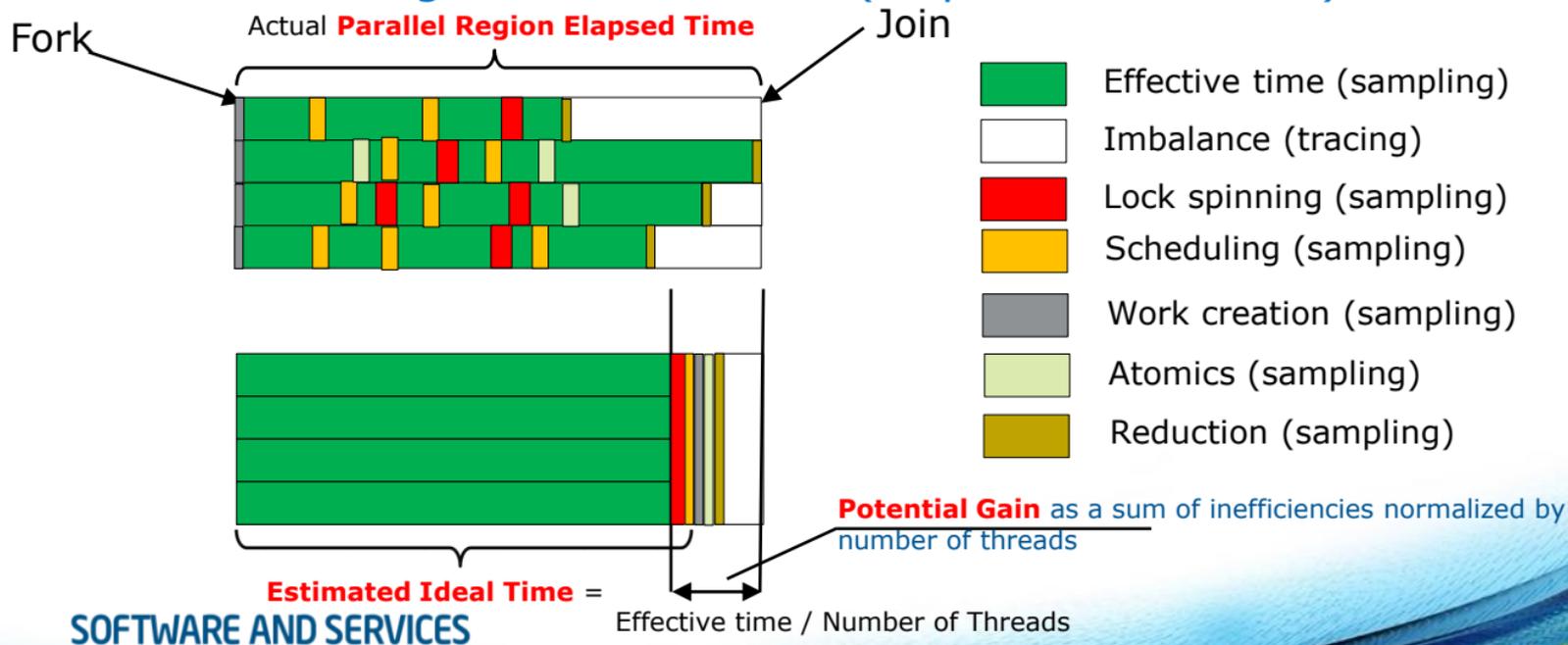
SOFTWARE AND SERVICES

# VTune GUI: OpenMP analysis

- **Tracing** of OpenMP constructs to provide region/work sharing context and imbalance on barriers
  - Advanced hotspots w/o stacks is recommended to make sampling representative for small regions
- VTune is provided with information by Intel OpenMP RTL
  - Fork-Join points of parallel regions with number of working threads (Intel Compilers version 14 and later)
  - OpenMP construct barrier points with imbalance info and OpenMP loop metadata
    - Embed source file name to an OpenMP region with `-parallel-source-info=2` compiler option

# VTune GUI: OpenMP region view (1/2)

## Definition of Region Potential Gain (elapsed time metric)



# VTune GUI: OpenMP region view (2/2)

Advanced Hotspots Hotspots viewpoint (change) Intel

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

Grouping: OpenMP Region / Function / Call Stack

OpenMP Region / Function / Call Stack	OpenMP Potential Gain						OpenMP Potential Gain (% of Collection Time)						Elapsed Time	Number of OpenMP threads	Inst... Count
	Imbalance	Lock Con...	Crea...	Sch...	Red...	Other	Imbalance (%)	Lock Con...	Crea... (%)	Sch... (%)	Red... (%)	Other (%)			
conj_grad_Somp\$parallel:24@/home/vtune/work/apps/NPB/NP3.3.1/NP3.3-OMP/CG/cg.f:514:695	3.944s	0s	0s	0s	0s	0.000s	34.6%	0.0%	0.0%	0.0%	0.1%	11.095s	24	76	
MAIN_Somp\$parallel:24@/home/vtune/work/apps/NPB/NP3.3.1/NP3.3-OMP/CG/cg.f:185:231	0.086s	0s	0s	0s	0s	0.000s	0.8%	0.0%	0.0%	0.0%	0.0%	0.286s	24	1	
[Serial - outside any region]						0s					0.0%	0.012s			
MAIN_Somp\$parallel:24@/home/vtune/work/apps/NPB/NP3.3.1/NP3.3-OMP/CG/cg.f:339:345	0.000s	0s	0s	0s	0s	0s	0.0%	0.0%	0.0%	0.0%	0.0%	0.001s	24	75	
MAIN_Somp\$parallel:24@/home/vtune/work/apps/NPB/NP3.3.1/NP3.3-OMP/CG/cg.f:361:365	0.000s	0s	0s	0s	0s	0s	0.0%	0.0%	0.0%	0.0%	0.0%	0.001s	24	75	
MAIN_Somp\$parallel:24@/home/vtune/work/apps/NPB/NP3.3.1/NP3.3-OMP/CG/cg.f:263:269	0.000s	0s	0s	0s	0s	0s	0.0%	0.0%	0.0%	0.0%	0.0%	0.000s	24	1	

Advanced Hotspots Hotspots viewpoint (change) Intel VTune AI

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

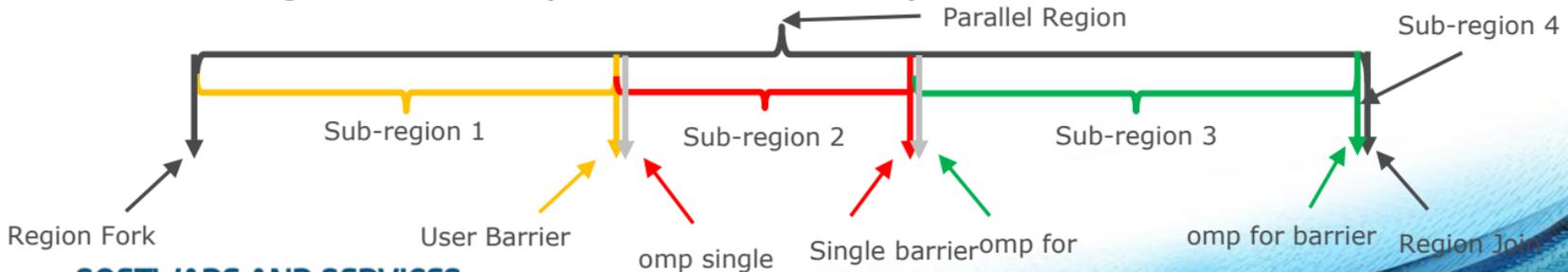
Grouping: OpenMP Region / Function / Call Stack

OpenMP Region / Function / Call Stack	OpenMP Potential Gain						OpenMP Potential Gain (% of Collection Time)						Elaps... Time	Num. of Ope... thre...	Inst... Count
	Imbalance	Lock Con...	Creation	Scheduling	Reduc...	Other	Imbalance (%)	Lock Con...	Crea... (%)	Scheduling (%)	Red... (%)	Other (%)			
conj_grad_Somp\$parallel:24@/home/vtune/work/apps/NPB/NP3.3.1/NP3.3-OMP/CG/cg.f:514:695	0.206s	0s	0.000s	3.128s	0s	0.000s	25.9%	0.0%	0.0%	0.0%	0.0%	11.758s	24	76	
MAIN_Somp\$parallel:24@/home/vtune/work/apps/NPB/NP3.3.1/NP3.3-OMP/CG/cg.f:185:231	0.075s	0.000s	0s	0s	0s	0.000s	0.6%	0.0%	0.0%	0.0%	0.0%	0.285s	24	1	
[Serial - outside any region]						0s					0.0%	0.013s			
MAIN_Somp\$parallel:24@/home/vtune/work/apps/NPB/NP3.3.1/NP3.3-OMP/CG/cg.f:339:345	0.000s	0s	0s	0s	0s	0.000s	0.0%	0.0%	0.0%	0.0%	0.0%	0.001s	24	75	
MAIN_Somp\$parallel:24@/home/vtune/work/apps/NPB/NP3.3.1/NP3.3-OMP/CG/cg.f:361:365	0.000s	0s	0s	0s	0s	0s	0.0%	0.0%	0.0%	0.0%	0.0%	0.001s	24	75	

# VTune GUI: OpenMP Barrier-to-Barrier view (1/2)

- Helps if there is more than 1 barrier (implicit or explicit) in a parallel region
  - Especially useful for the model with 1 parallel region and work-sharing constructs inside with pragma single to do sequential work
- OpenMP RTL allows tracing sub-region segments from region fork or previous barrier points

```
#pragma omp parallel
{
    .....
    #pragma omp barrier
    #pragma omp single
    {
        ....
    }
    #pragma omp for
    {
        ...
    }
}
```



# VTune GUI: OpenMP Barrier-to-Barrier view (2/2)

Advanced Hotspots Hotspots viewpoint (change) Intel VTune Amplifier XE 20

Collection Log Analysis Target Analysis Type Summary Bottom-up Caller/Callee Top-down Tree Platform

Grouping: OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack

OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack

Imbalance on a loop barrier

	Imbalance	Lock Con...	Creation	Schedu...	Redu...	Atomi...	Other	Elapsed Time	Number of OpenMP threads	Ins... Cou...	OpenMP Loop Schedule Type	OpenMP Loop Chunk	Avg OpenMP Loop Iteration Count
conj_grad_omp\$parallel.24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f.514:695	3.944s	0s	0.000s	0.002s	0.000s	0s	0.094s	11.095s	24	76	Static	3125	75,000
conj_grad_omp\$loop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f.683	3.725s	0s	0s	0s	0s	0s	0.004s	0.418s	24		Static	3125	75,000
conj_grad_omp\$loop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f.625	0.033s	0s	0s	0.002s	0.000s	0s	0.002s	0.068s	24		Static	3125	75,000
conj_grad_omp\$loop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f.650	0.015s	0s	0s	0.000s	0s	0s	0.001s	0.064s	24		Static	3125	75,000
conj_grad_omp\$loop_barrier_segment@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f.664	0.014s	0s	0s	0.000s	0s	0s	0.001s	0.079s	24		Static	3125	75,000

Advanced Hotspots Hotspots viewpoint (change) Intel VTune Amplifier XE 2015

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

Grouping: OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack

OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack

	Imba...	Lock Con...	Cre...	Scheduling	Red...	Oth.	Imba... (%)	Lock Cont...	Cre... (%)	Schedu... (%)	Red... (%)	Oth... (%)	Elap... Time	Nu. of Ope. thr...	Ins. Co.	Ope. Loop... Chu.	Open... Loop Sched... Type
conj_grad_omp\$parallel.24@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f.514:695	0.20...	0.0...	0.0...	3.127s	0.0...	0.0...	1.7%	0.0%	0.0%	25.9%	0.0%	0.0%	11.7...	24	76		Static
conj_grad_omp\$loop_barrier@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f.572:580	0.00...	0.0...	0.0...	3.125s	0.0...	0.0...	0.1%	0.0%	0.0%	25.9%	0.0%	0.0%	0.41...	24		1	Dynamic
conj_grad_omp\$loop_barrier@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f.675:683	0.12...	0s	0s	0s	0s	0s	1.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.41...	24			Static
conj_grad_omp\$loop_barrier@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f.621:625	0.02...	0s	0s	0.001s	0.0...	0.0...	0.2%	0.0%	0.0%	0.0%	0.0%	0.0%	0.07...	24			Static
conj_grad_omp\$loop_barrier@/home/vtune/work/apps/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f.637:650	0.02...	0s	0s	0.000s	0.0...	0.0...	0.2%	0.0%	0.0%	0.0%	0.0%	0.0%	0.07...	24			Static

Dynamic scheduling overhead on a parallel loop

- Imbalance distribution by loop, single, reduction, user, join barriers

# VTune GUI: OpenMP serial hotspots

**Advanced Hotspots** Hotspots viewpoint (change)

Collection Log Analysis Target Analysis Type Summary Bottom-up Caller/Call

OpenMP Analysis. Collection Time: 23.251s

Serial Time (outside any parallel region): **13.396s (57.6%)**

Serial Time of your application is high. It directly impacts application elapsed time and scalability. microarchitecture tuning of the serial part of the application.

Parallel Region Time: **9.855s (42.4%)**

Estimated Ideal Time: 3.943s (17.0%)

Potential Gain: **5.912s (25.4%)**

The time wasted on load imbalance or parallel work arrangement is significant and negatively impacts scalability. Explore OpenMP regions with the highest metric values. Make sure the workload

**Advanced Hotspots** Hotspots viewpoint (change)

Collection Log Analysis Target Analysis Type Summary Bottom-up

OpenMP Region / Thread / Function / Call Stack

OpenMP Region / Thread / Function / Call Stack	Poten... Gain	Elapsed Time	CPU Time	Instructions Retired
[Serial - outside any region]	0s 0.0%	<b>13.396s</b>	24,004s	79,223,400
OMP Master Thread #0 (TID: 227)	0s 0.0%	13.313s	55,703,700	
CPhysicalGeometry::FindFace	0s 0.0%	1.587s	6,488,100	
CTetrahedron::GetNode	0s 0.0%	1.216s	2,311,200	
rint_malloc	0s 0.0%	0.763s	5,175,900	
rtd:sort...,gnu_cxx:...normal	0s 0.0%	0.628s	6,633,300	
rint_free	0s 0.0%	0.602s	3,223,800	
CPhysicalGeometry::SetPoint_C	0s 0.0%	0.566s	899,100	
L_libc_malloc	0s 0.0%	0.479s	2,130,300	

**Serial hotspots under Master Thread**

**Advanced Hotspots** Hotspots viewpoint (change)

Collection Log Analysis Target Analysis Type Summary Bottom-up Caller/Call Top-down Tree Platforms

OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack

OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack	OpenMP Potential = Gain	Elapsed Time	Num. of Ops. Exec.	Inst. Count	Op. Loop Type	Avg Op. Loop Iter.	Max Op. Loop Iter.	Min Op. Loop Iter.	Effective Time by U
main_read_SomeSource636@/home/stune/work/apps/NPB/NPB3.3	1.228s	3.401s	24	23				37.66%	Poor
MAIN_SompParallel24@/home/stune/work/apps/NPB/NPB3.3	0.000s	0.000s	24	23					
MAIN_SompParallel24@/home/stune/work/apps/NPB/NPB3.3	0	0	23					0.003s	
[Serial - outside any region]	0s	<b>0.00s</b>						13.472s	

Selected 1 row(s)

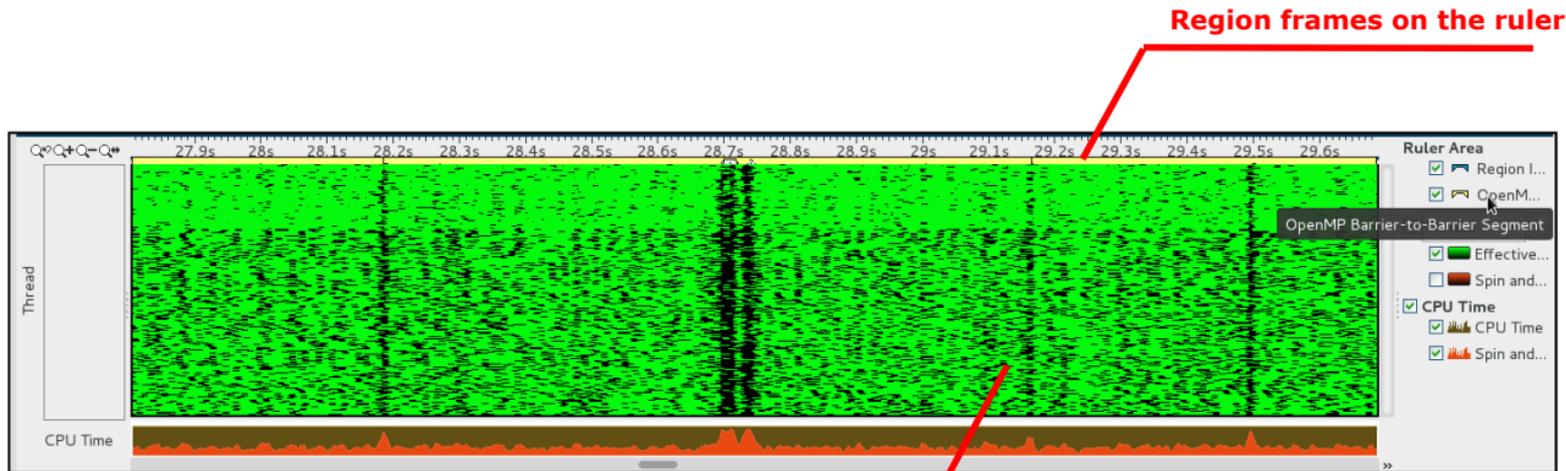
Time Filter: 0.00s

Ruler Area

- Region L...
- OpenMP...
- Running
- CPU Time
- Spin and...
- Hardware E...

**Time Filter to exclude initialization phase**

# VTune GUI: OpenMP scalable timeline



Intel® Xeon Phi™ profiling result  
with 288 threads

# VTune GUI: OpenMP region source view

**Advanced Hotspots Hotspots viewpoint (change)** Intel VTune AM

Collection Log Analysis Target Analysis Type Summary Bottom-up Caller/Callee Top-down Tree Platform

Grouping: OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack

OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack

Imbalance	Lock Con...	Crea...	Sch...	Red...	Ato...	Other	Elapsed Time	Num. of Ope... thre...	Inst... Count	Ope.. Loop Chu..	Ope.. Loop Sch... Type	Avg Ope.. Loop Itera...	Max Ope.. Loop Itera...	Min Ope.. Loop Itera...
3.944s	0s	0.000s	0.002s	0.000s	0s	0.010s	11.095s	24	76		Static	75.000	75.000	75.000
3.725s	0s	0s	0.000s	0s	0s	0.008s	10.445s	24	3125					

**Advanced Hotspots Hotspots viewpoint (change)** Intel VTune Amplifier XE 2015

Source Assembly Assembly grouping: Address

Source	CPU Time	Instructions Retired
S14   \$omp parallel default(shared) private(j,k,cg,t,suml,alpha,beta)		
S15   \$omp\$ shared(d,rho,rho,sum)		
S16		
S17   c-----		
S18   c Initialize the CG algorithm:		
S19   c-----		
S20    \$omp do		
S21   do j=1,naa+1	0.002s	0
S22   q(j) = 0.0d0	0.034s	8,100,000
S23   z(j) = 0.0d0	0.022s	10,800,000
S24   r(j) = x(j)	0.014s	2,700,000
S25   p(j) = r(j)	0.041s	5,400,000
S26   enddo		
S27    \$omp end do		
S28		
S29		

# Intel® VTune™ Amplifier XE – Typical HPC workflow



# Profiling HPC applications

- VTune can profile hybrid MPI+OpenMP applications on a cluster
  - For profiling MPI, use Intel<sup>®</sup> Trace Analyzer and Collector or Intel<sup>®</sup> MPI Performance Snapshot
- Recommended workflow:
  - Run **collect** with CLI on *a cluster*
  - Run **report** with GUI on *a local workstation* or a cluster login node
    - Collection results can be transferred if needed

# VTune with MPI applications (1/3)

- Single node application launch:

```
<vtune_command> [--] <mpi_command> <application>
```

```
> amplxe-cl -collect advanced-hotspots -r result_dir -- mpirun -  
np 48 ./mpi_app
```

- Encapsulates all the ranks to result directory
  - Example: ranks 0-47 in `result_dir`
- Works whenever VTune is able to track the processes created
  - Limited to profiling over a single node

# VTune with MPI applications (2/3)

- Multiple node application launch:

```
<mpi_command> <vtune_command> [--] <application>
```

```
> aprun -n 48 -ppn 16 amplxe-cl -collect hotspots -r result_dir  
./mpi_app
```

- Results encapsulated to per-node directories suffixed with hostname
  - Example: ranks 0-15 in `result_dir.hostname1`, ranks 16-31 in `result_dir.hostname2`, ranks 32-47 in `result_dir.hostname3`

# VTune with MPI applications (3/3)

- Selective rank profiling by modifying the MPI process launch:

```
> mpirun -n 1 ./mpi_app : -n 1 amplxe-cl -collect hotspots -r  
result_dir ./mpi_app : -n 14 ./mpi_app
```

- Intel MPI supports `-gtool "<command>:<rank-set>[=mode]"` option:

```
> mpirun -n 16 -gtool "amplxe-cl -collect hotspots -r result_dir  
:1" ./mpi_app
```

# HPC Performance Analysis

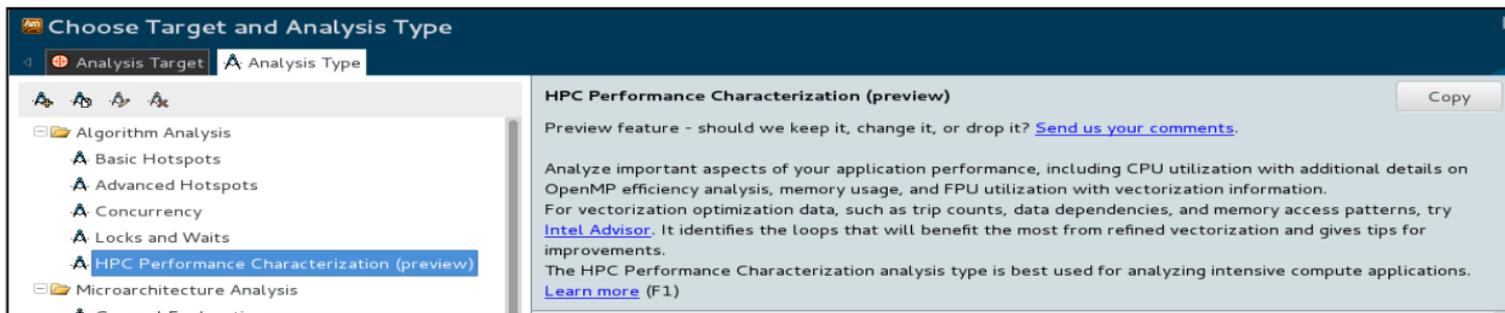


# HPC Performance Analysis

- CLI:

```
> mpirun -n 16 -gtool "amplxe-cl -collect hpc-performance -r  
result_dir :1" ./mpi_app
```

- GUI:



**Choose Target and Analysis Type**

Analysis Target | Analysis Type

- Algorithm Analysis
  - Basic Hotspots
  - Advanced Hotspots
  - Concurrency
  - Locks and Waits
  - HPC Performance Characterization (preview)**
- Microarchitecture Analysis
  - General Evaluation

**HPC Performance Characterization (preview)** Copy

Preview feature - should we keep it, change it, or drop it? [Send us your comments.](#)

Analyze important aspects of your application performance, including CPU utilization with additional details on OpenMP efficiency analysis, memory usage, and FPU utilization with vectorization information. For vectorization optimization data, such as trip counts, data dependencies, and memory access patterns, try [Intel Advisor](#). It identifies the loops that will benefit the most from refined vectorization and gives tips for improvements.

The HPC Performance Characterization analysis type is best used for analyzing intensive compute applications. [Learn more](#) (F1)

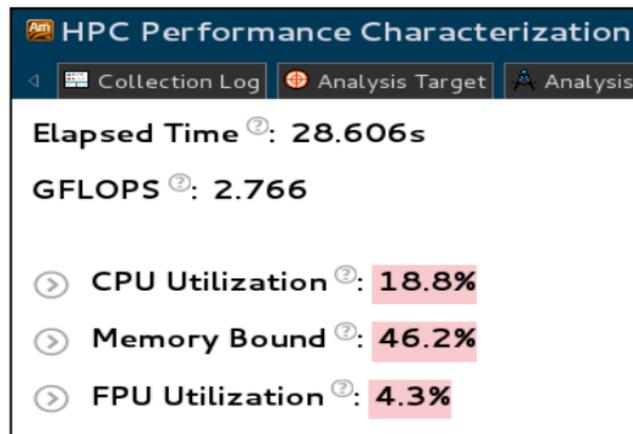
# Analysis Structure and Metrics

## Two characterization metrics

- Elapsed Time
- GFLOPs\*

## Three performance aspects

- CPU Utilization
- Memory Bound
- FPU Utilization\*



\*Metrics are available on HW that supports floating point PMU events (IVB/IVT, BDW, SKL..)

# Performance Aspect: CPU Utilization

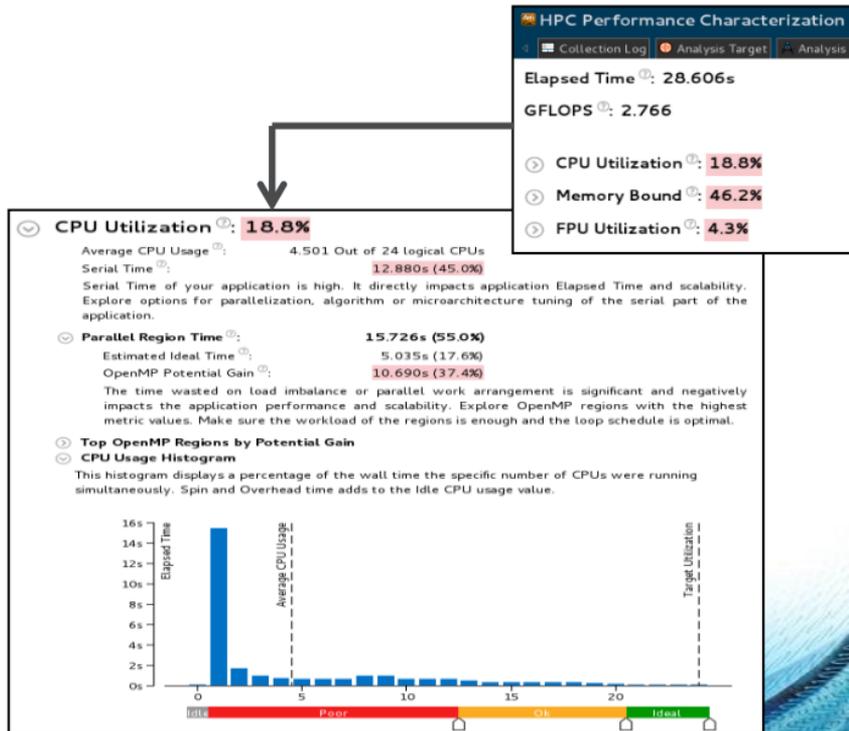
## CPU Utilization

- % of effective CPU usage under profiling (threshold 90%)
- Under assumption that the app should use all available logical cores on a node
- Subtracting spin/overhead time spent in MPI and threading runtimes

## Metrics in CPU utilization section

- Average CPU usage
- Additional MPI and OpenMP scalability metrics impacting effective CPU utilization
- CPU utilization histogram

## SOFTWARE AND SERVICES



# Performance Aspect: Memory Access

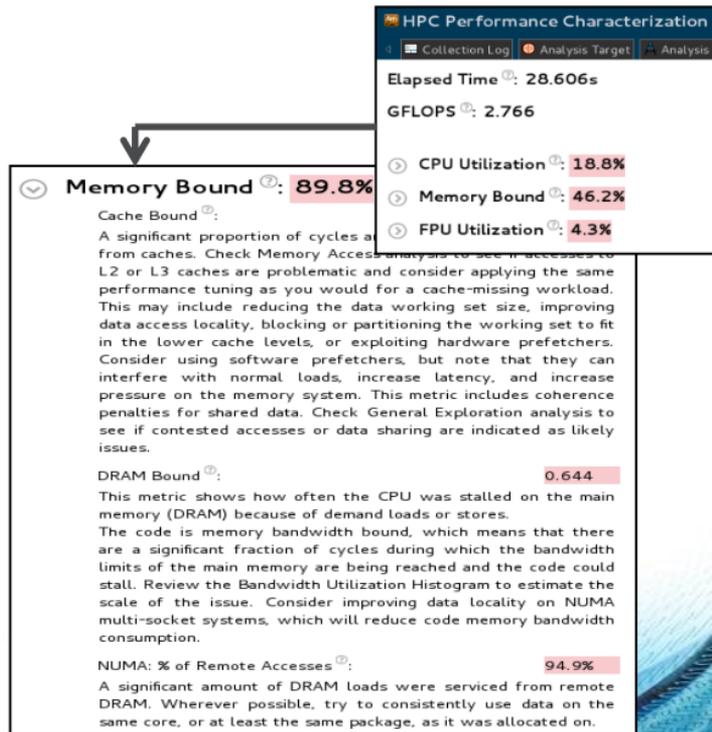
## Memory Bound

- % of potential execution pipeline slots lost due to memory accesses to different levels of the hierarchy (threshold 80%)

## Metrics in Memory Bound section

- Cache bound, DRAM bound
- Issue description specifies if the code is bandwidth or latency bound with proper advice of how to fix
- NUMA: % of remote accesses
- Important to explore if the code is bandwidth bound
- Bandwidth utilization histogram

## SOFTWARE AND SERVICES



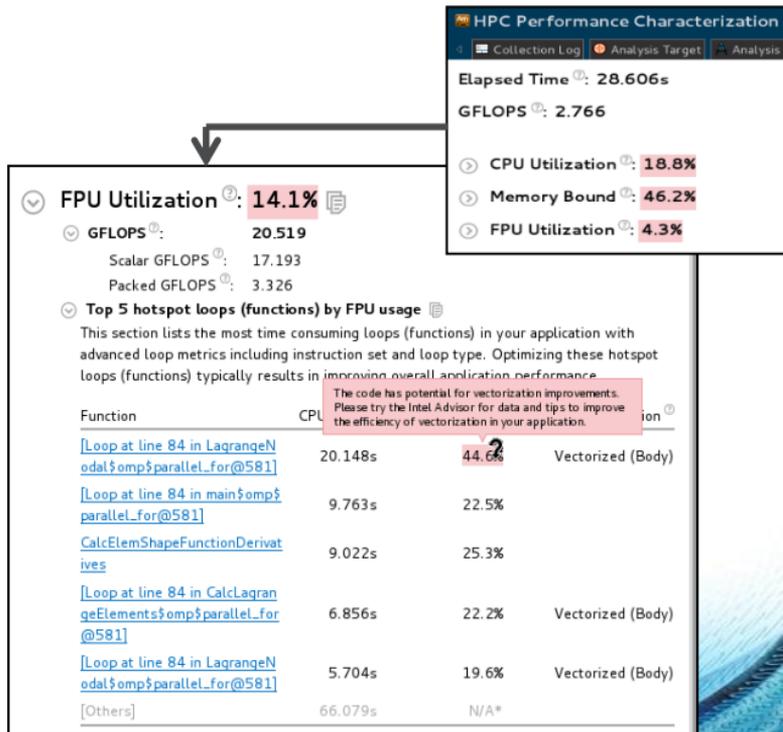
# Performance Aspects: FPU Utilization

## FPU utilization

- % of FPU load (100% - FPU is fully loaded, threshold 50%)

## Metrics in FPU utilization section

- GLOPs broken down by scalar and packed
- Top 5 loops/functions by FPU usage
- Dynamically generated issue descriptions on low FPU usage help to define the reason and next steps:
- Non-vectorized/vectorized with legacy instruction set
- memory bound limited loops not benefiting from vectorization etc.



# Bottom-Up Grid View

## Metrics by OpenMP regions or functions/loops

- Regulated by choosing proper grouping
- Wall time/global metrics like elapsed time, GLOPs, serial time, OpenMP potential gain are available for Process/Region groupings

Grouping: OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack

OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack	Elaps... Time	GFLOPS	Serial Time	Ope. Pot.. Gain	Memory Bound					NUMA: % of Remote Accesses	FPU Utili...	CPU Time	Nu. of Ope. thr..	Ins... Cou.	Ope. Loo... Chu.	Ope. Loo... Sch... Type	Avg Ope. Loo... Ite...	Max Ope. Loo... Ite...	Min Ope. Loo... Ite...
					L1 Bound	L2 Bound	L3 Bound	DRA.. Bound	Other										
conj_grad_omp\$parallel: 24@/home/vtune/wor	13.30...	4.253	0s	5.200s	0.069	0.000	0.266	0.231	0.000	55.5%	1.6%	296.673s	24	76					
conj_grad_omp\$loop_barrier_segment@/hom	2.918s	0.165	0s	2.672s	0.208	0.000	0.003	0.012	0.003	0.0%	0.1%	60.395s	24		25	Dyn...	75...	75...	75...
conj_grad_omp\$loop_barrier_segment@/hom	1.028s	0.933	0s	0.912s	0.107	0.016	0.039	0.000	0.010	0.0%	0.4%	21.792s	24		312.	Sta...	75...	75...	75...
conj_grad_omp\$loop_barrier_segment@/hom	7.923s	6.502	0s	0.636s	0.015	0.000	0.407	0.356	0.000	55.5%	2.3%	184.694s	24		20	Gui...	75...	75...	75...
conj_grad_omp\$loop_barrier_segment@/hom	0.722s	0.665	0s	0.568s	0.115	0.023	0.000	0.035	0.033	0.0%	0.3%	13.923s	24		312.	Sta...	75...	75...	75...
conj_grad_omp\$barrier_segment@/home/vtu	0.236s	0.000	0.000s	0.215s	0.088	0.015	0.000	0.000	0.009	0.0%	0.0%	5.263s	24						

