Intel® VTune™ Amplifier XE

Dr. Michael Klemm
Software and Services Group
Developer Relations Division

SOFTWARE AND SERVICES

Legal Disclaimer & Optimization Notice

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:

ISBN-13 (pbk): 978-1-4302-6496-5
ISBN-13 (electronic): 978-1-4302-6497-2

Order now at http://www.clusterbook.info

SOFTWARE AND SERVICES

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


IN R 805

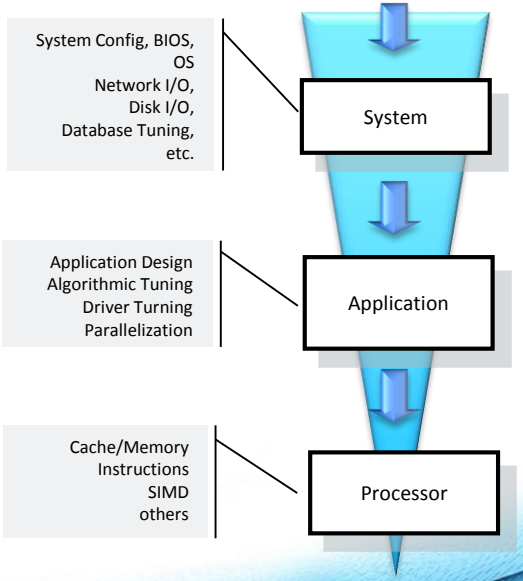
The Software Optimization Process

- 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?

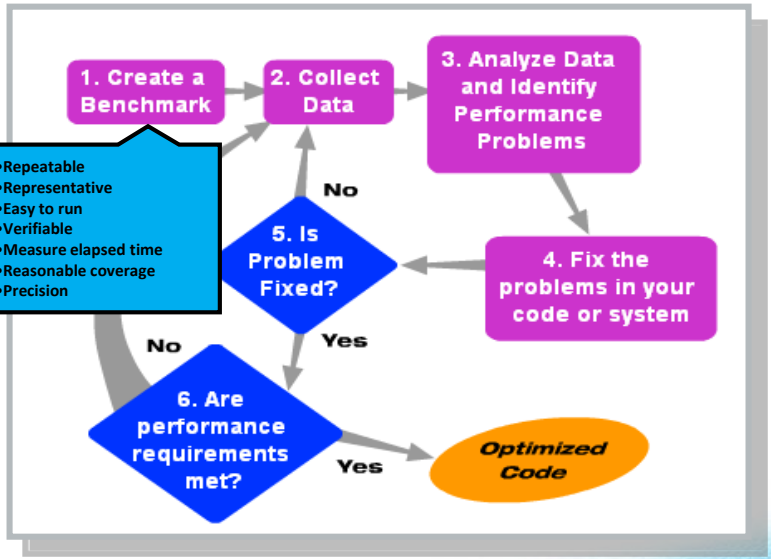
These methods aid developers to reach their performance requirements.

Performance Analysis Methodology

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



Performance Analysis Methodology



When to Stop

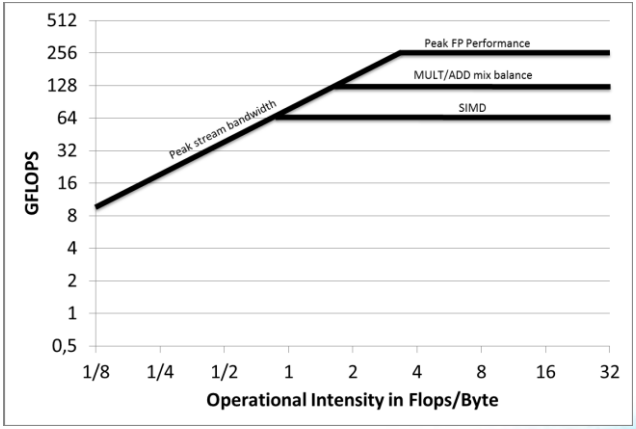
- Is architecture at maximum efficiency?
  - What this means: calculating theoretical maximum.
  - Know about best values for CPI or IPC.
  - Know the maximum FLOPS for the data type used.

- Is the performance requirement fulfilled?
  - What are the performance requirements?
  - Incrementally complete optimizations until done.
  - Key question: Are you “happy” with it?

CPI: Cycles per Instructions
IPC: Instructions per Cycle
FLOPS: Floating-Point Oper. Per Sec.

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

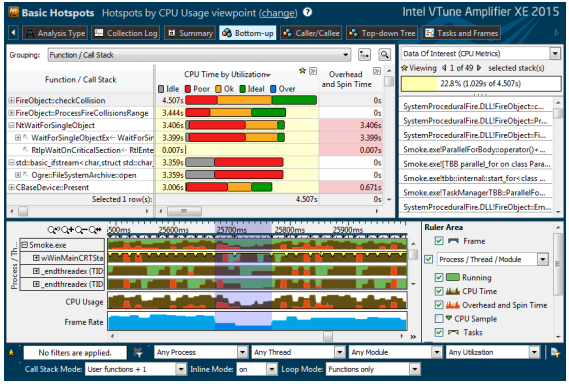
Intelligently sacrifice quality for performance

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	- Easy to install on Windows - Linux requires root (or use default perf driver without stacks)

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

A rich set of analysis types

Software Collector	Hardware Collector
Basic Hotspots Which functions use the most time?	**Advanced Hotspots** Which functions use the most time? Where to inline? – Statistical call counts
Concurrency Tune parallelism. Colors show number of cores used.	**General Exploration** Where is the biggest opportunity? Cache misses? Branch mispredictions?
Locks and Waits Tune the #1 cause of slow threaded performance: – waiting with idle cores.	**Advanced Analysis** Dig deep to tune access contention, **memory analysis**, 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:

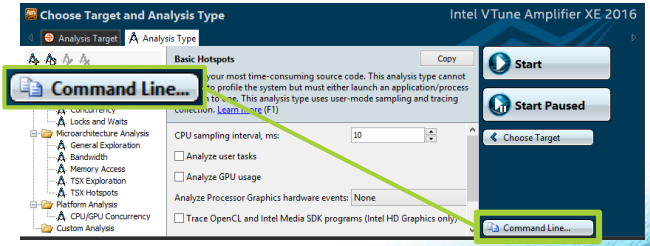
- Perf can collect stacks
- Use pre-installed perf driver
  - Intel EBS driver provides additional features not available in perf:
    - Uncore events
    - Multiple precise events
    - New events for the latest processors, even on an older OS

Easier access to the on-chip PMU for advanced performance profiling

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[ollect] <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

```
> amplxe-cl -help -collect # List analysis types available
> amplxe-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[eport] <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`**

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

- NOTE: using a GUI to view results is preferrable

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
```

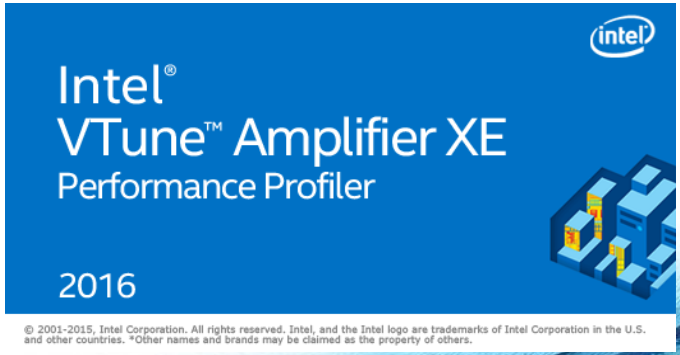
- View available columns in the result and then compile a hotspots report from specific columns

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

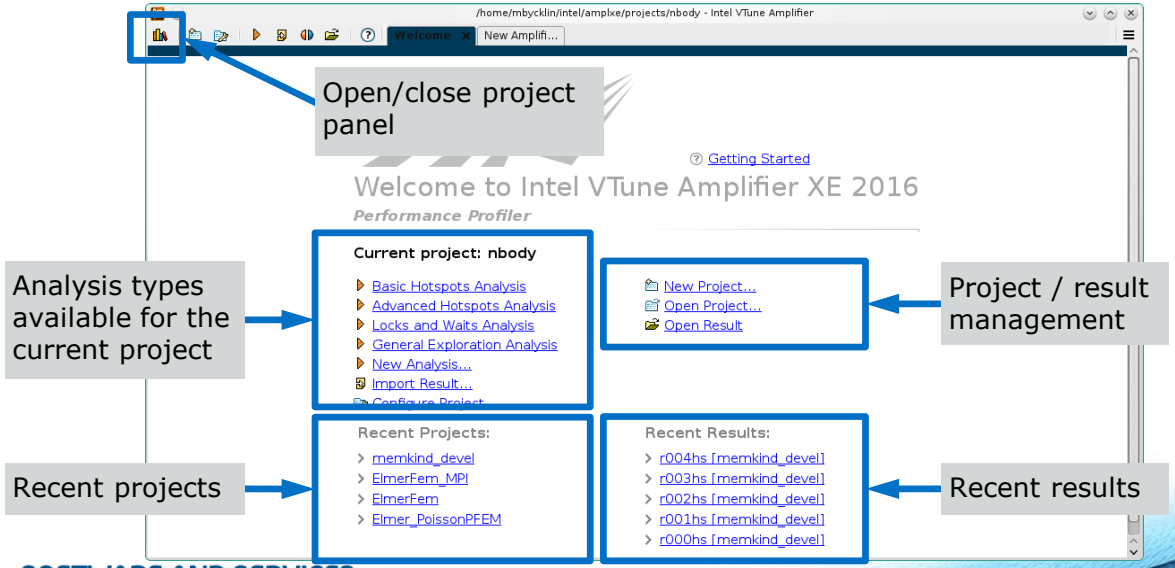
Intel® VTune™ Amplifier XE – Graphical User Interface

VTune Graphical User Interface (GUI)

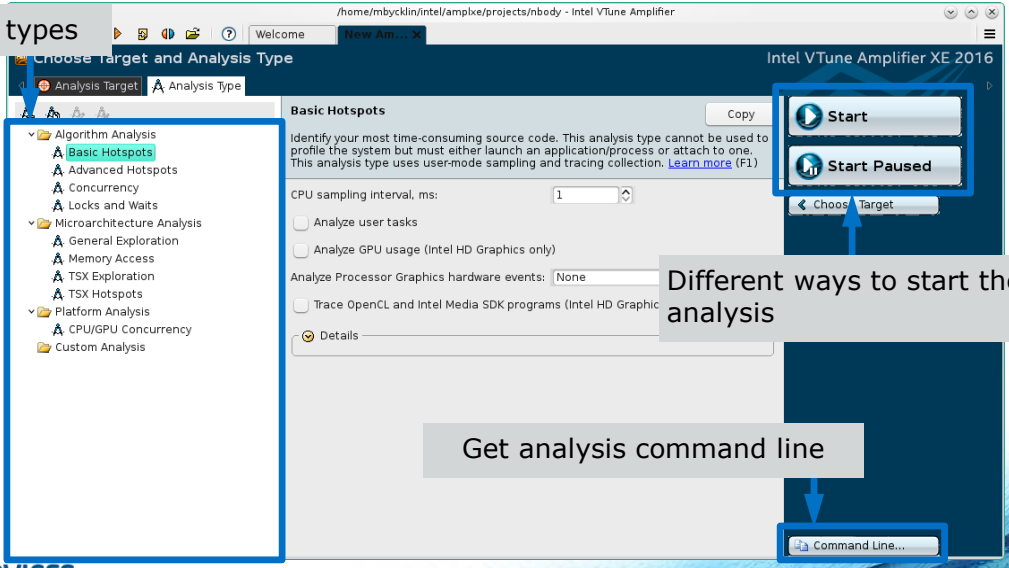
- Graphical tool **`amplxe-gui`**
  - Windows:
    `C:\Program Files (x86)\Intel\VTune Amplifier XE \bin[32|64]\amplxe-gui.exe`
  - Linux:
    `/opt/intel/vtune_amplifier_xe/bin[32|64]/amplxe-gui`
- Pure GUI workflow
  1) Set up a project
  2) Choose analysis
  3) View analysis results



VTune GUI: Welcome view



VTune GUI: choose analysis view



VTune GUI: analysis view

Adjust data grouping

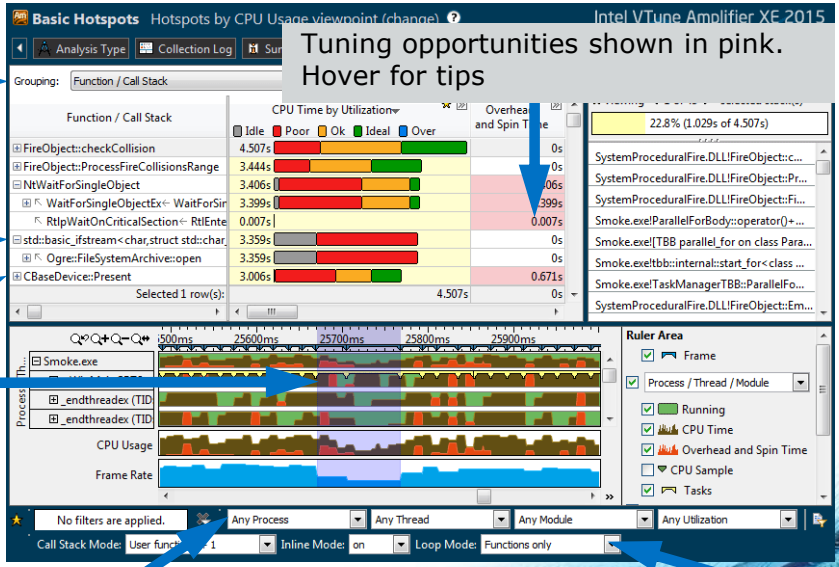
Function - Call Stack
Module - Function - Call Stack
Source File - Function - Call Stack
Thread - Function - Call Stack
... (Partial list shown)

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

Filter by process & other controls

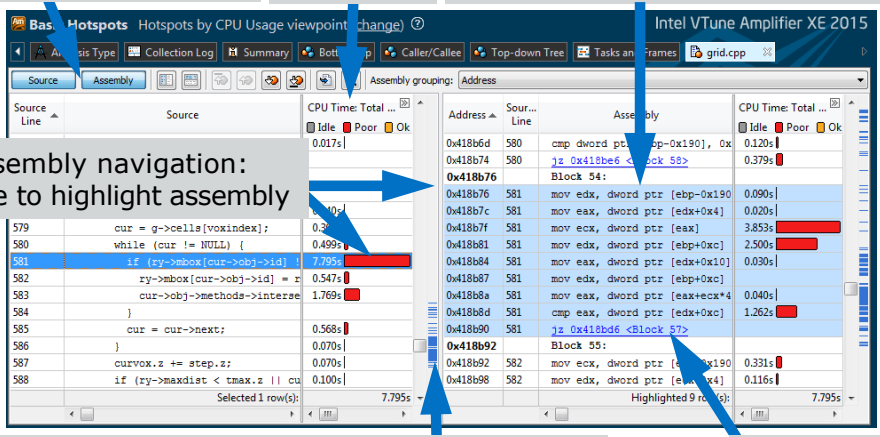
Function / loop mode

VTune GUI: view source code profile

View source / assembly or both

CPU Time

Right click for instruction reference manual

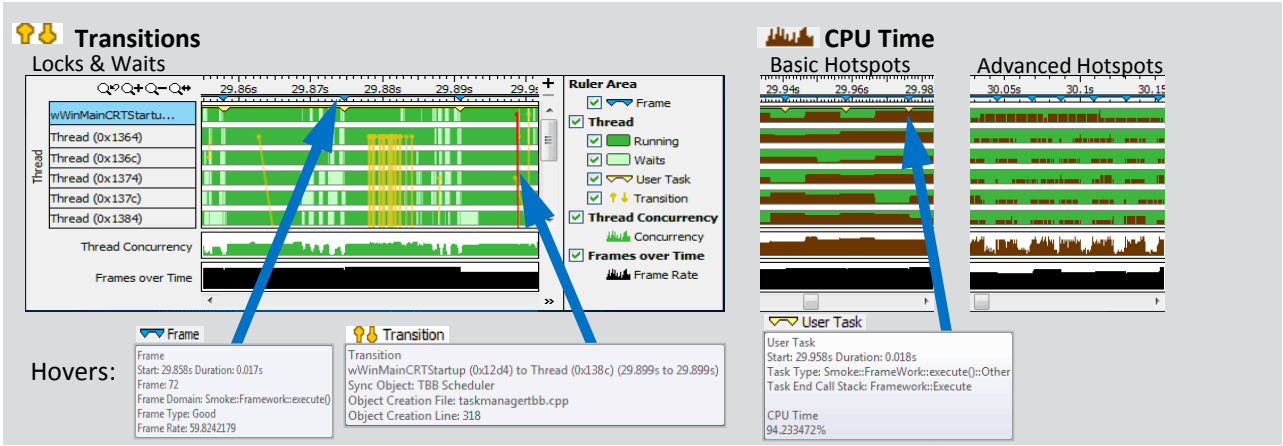


Quick assembly navigation:
Select source to highlight assembly

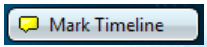
Scroll Bar "Heat Map" is an overview of hot spots

Click jump to scroll assembly

VTune GUI: view thread timeline



- Optional: Use API to mark frames and user tasks 

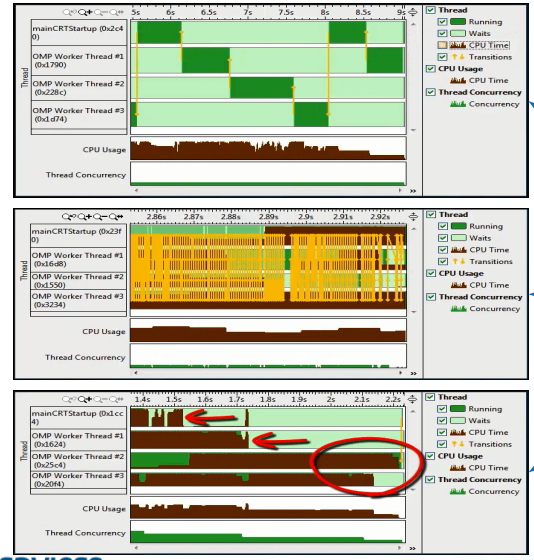
- Optional: Add a mark during collection 

VTune GUI: find concurrency issues

Coarse grain locks

High lock contention

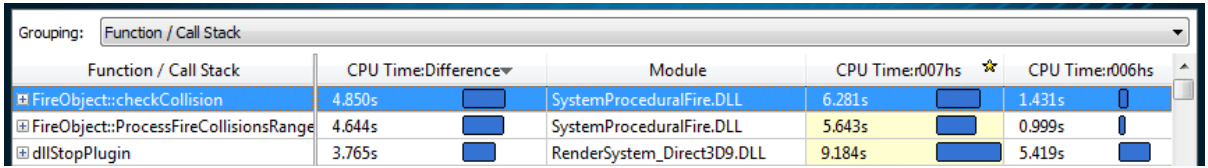
Load imbalance



Low concurrency

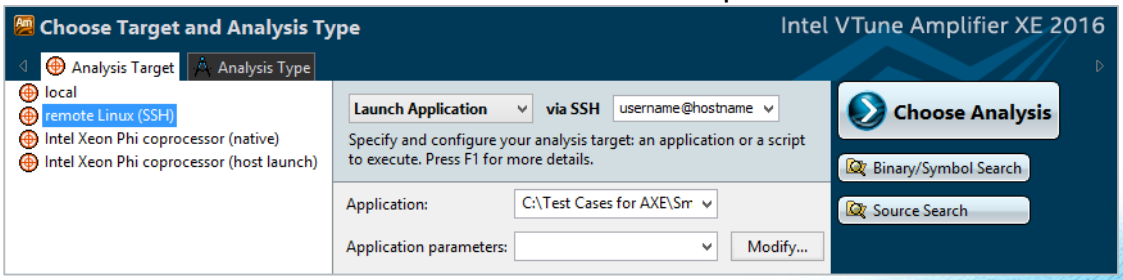
VTune GUI: results comparison

- 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?



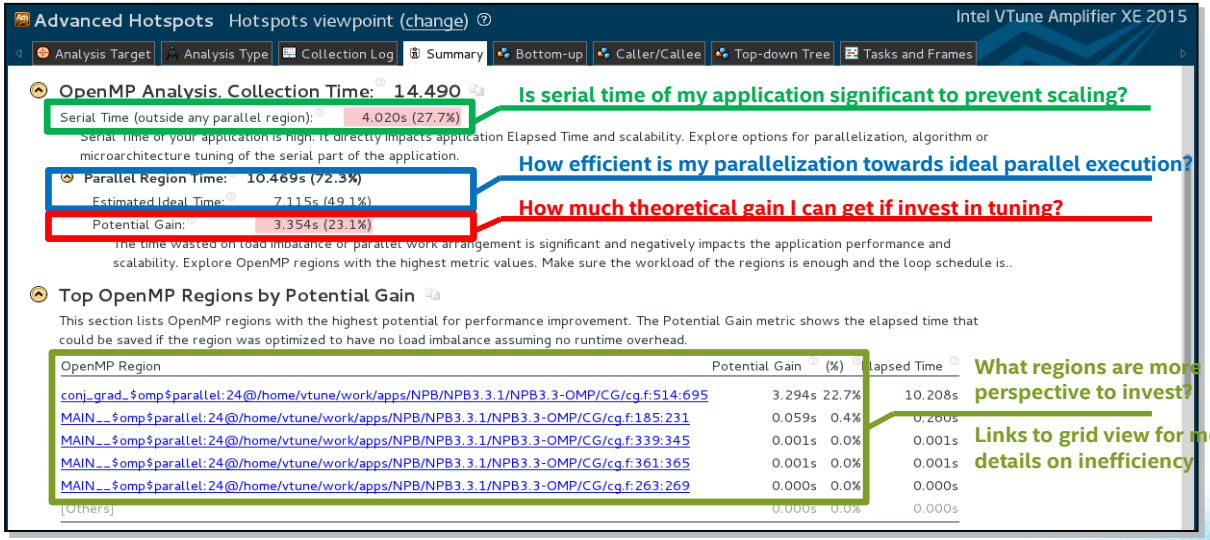
VTune GUI: remote data collection

- 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



Conveniently use your local UI to analyze remote systems

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



- An overview of the "Summary" pane

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

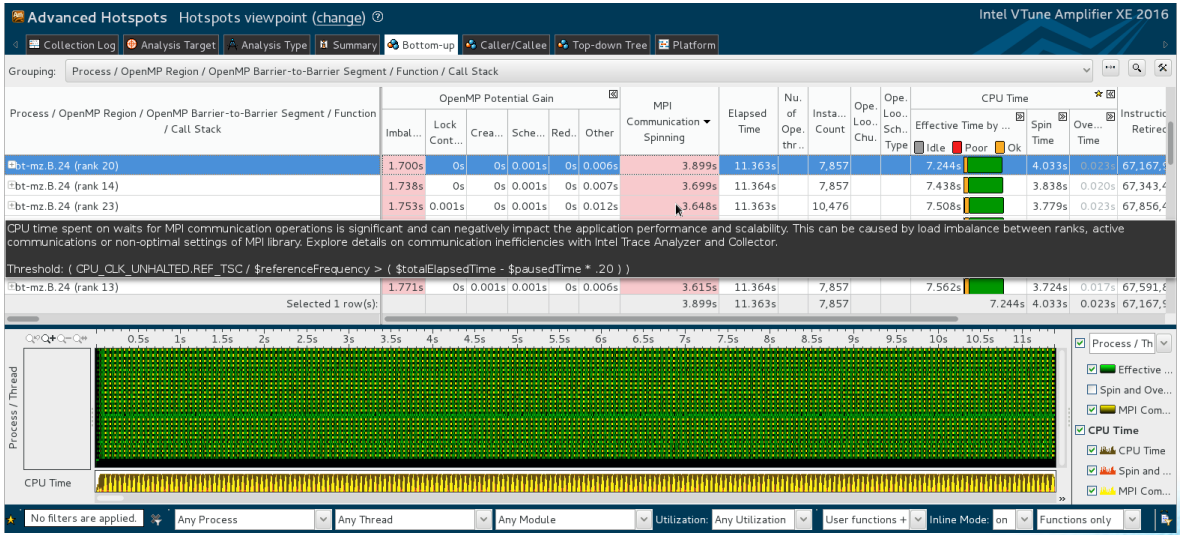
Top OpenMP Processes by MPI Communication Spin Time

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	(%)	OpenMP Potential Gain	(%)	Serial Time	(%)
bt-mz.B.4 (rank 2)	37954	4.428s	16.8%	11.234s	42.7%	5.410s	20.6%
bt-mz.B.4 (rank 0)	37948	5.236s	19.9%	9.953s	37.8%	6.542s	24.9%
bt-mz.B.4 (rank 3)	37957	5.274s	20.0%	10.329s	39.3%	6.384s	24.3%
bt-mz.B.4 (rank 1)	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)



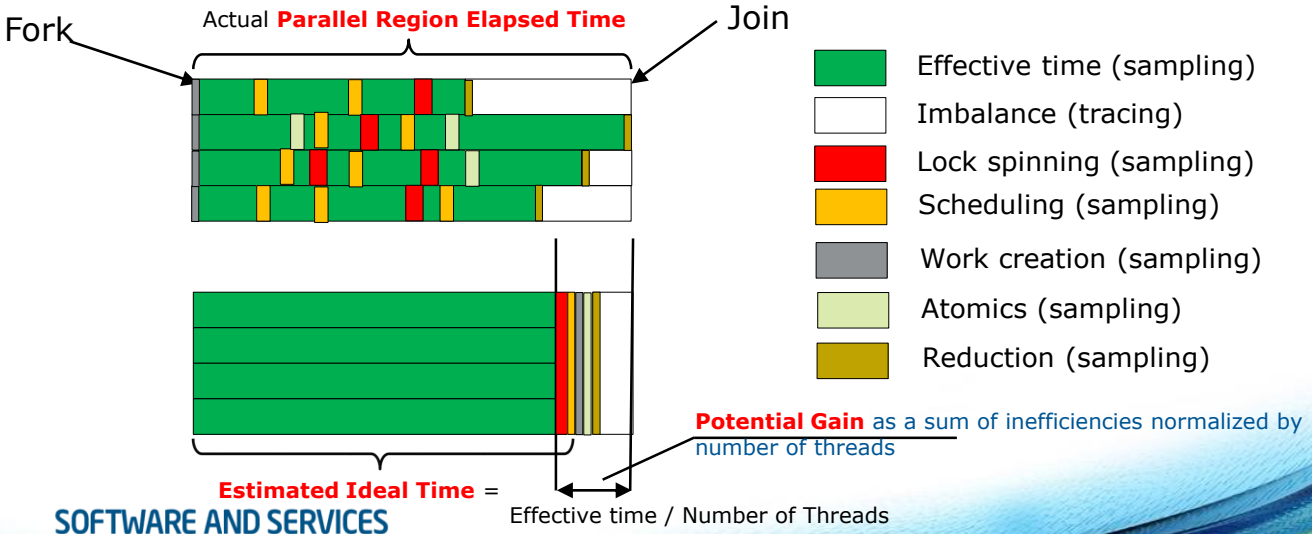
- "MPI communication-aware" grid and Process/Thread scalable timeline view

VTune GUI: OpenMP analysis

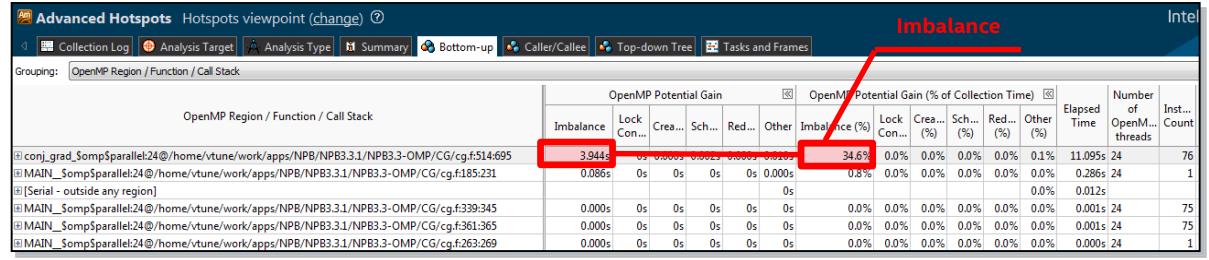
- **Tracing** of OpenMP constructs to provide region/work sharing context and imbalance on barriers
  - Advanced hotspots w/o stacks is recommended to make sampling representative for small regions
- VTune is provided with information by Intel OpenMP RTL
  - Fork-Join points of parallel regions with 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

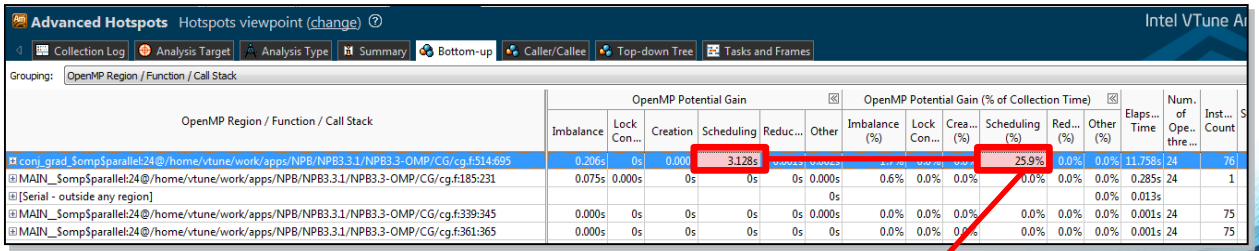
VTune GUI: OpenMP region view (1/2)

Definition of Region Potential Gain (elapsed time metric)



VTune GUI: OpenMP region view (2/2)

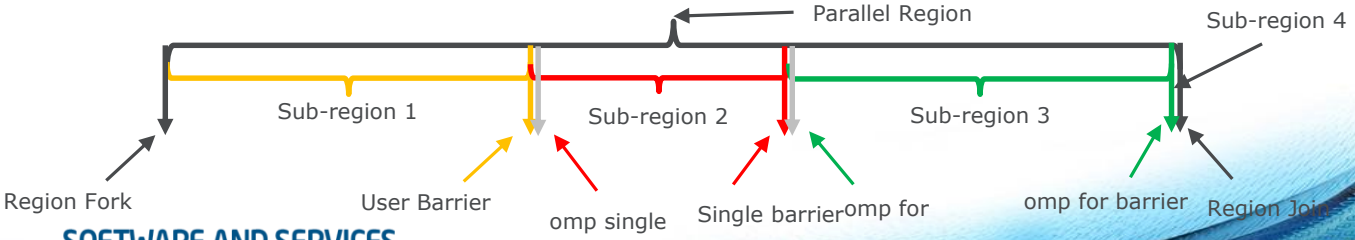




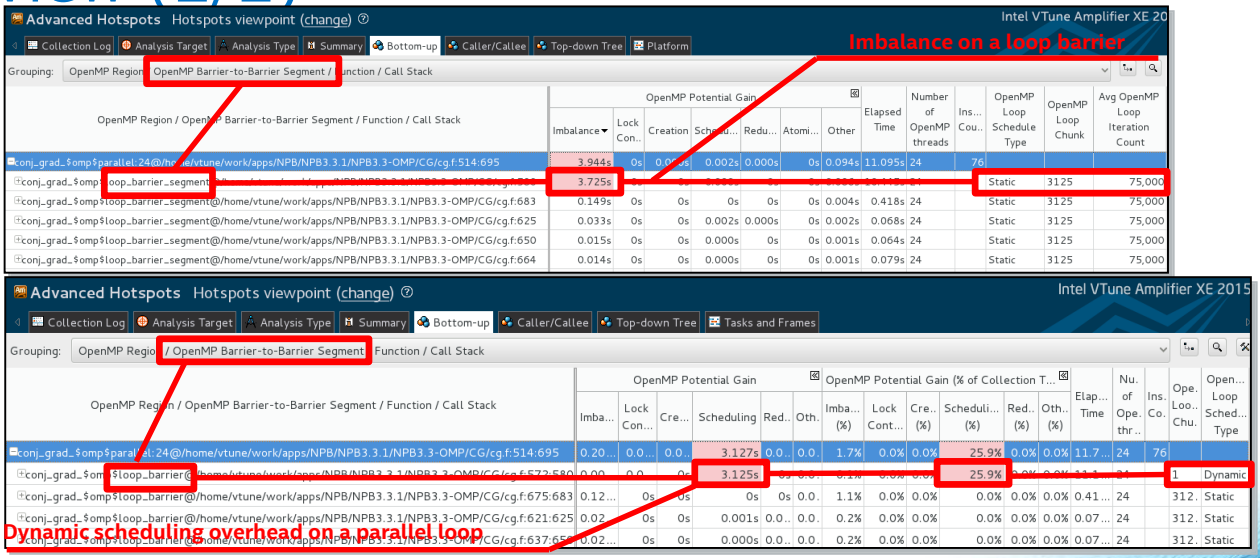
VTune GUI: OpenMP Barrier-to-Barrier view (1/2)

- 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
    {
        ...
    }
}
```

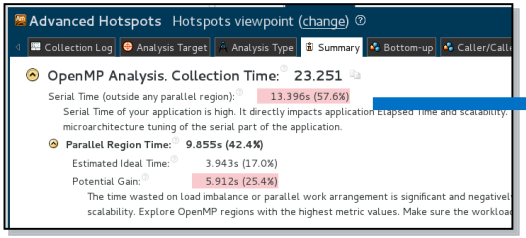


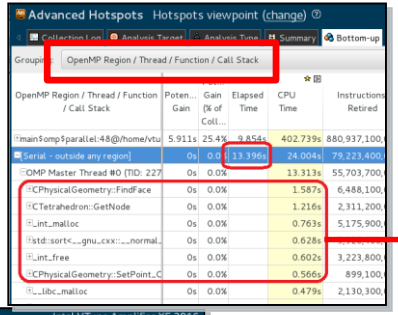
VTune GUI: OpenMP Barrier-to-Barrier view (2/2)



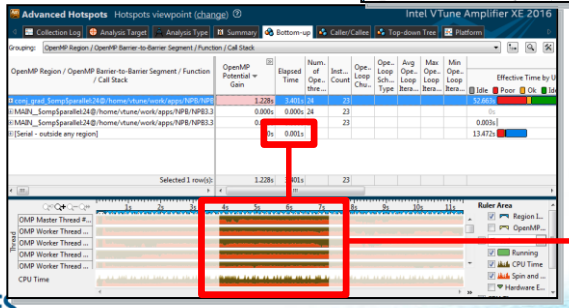
- Imbalance distribution by loop, single, reduction, user, join barriers

VTune GUI: OpenMP serial hotspots



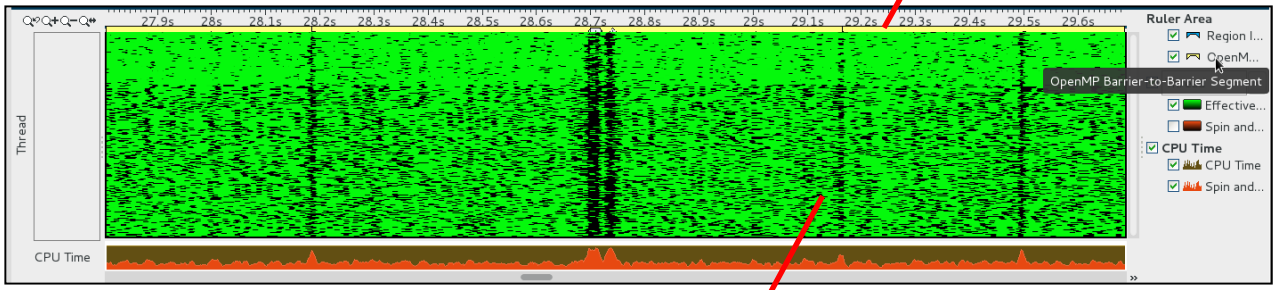


Serial hotspots under Master Thread



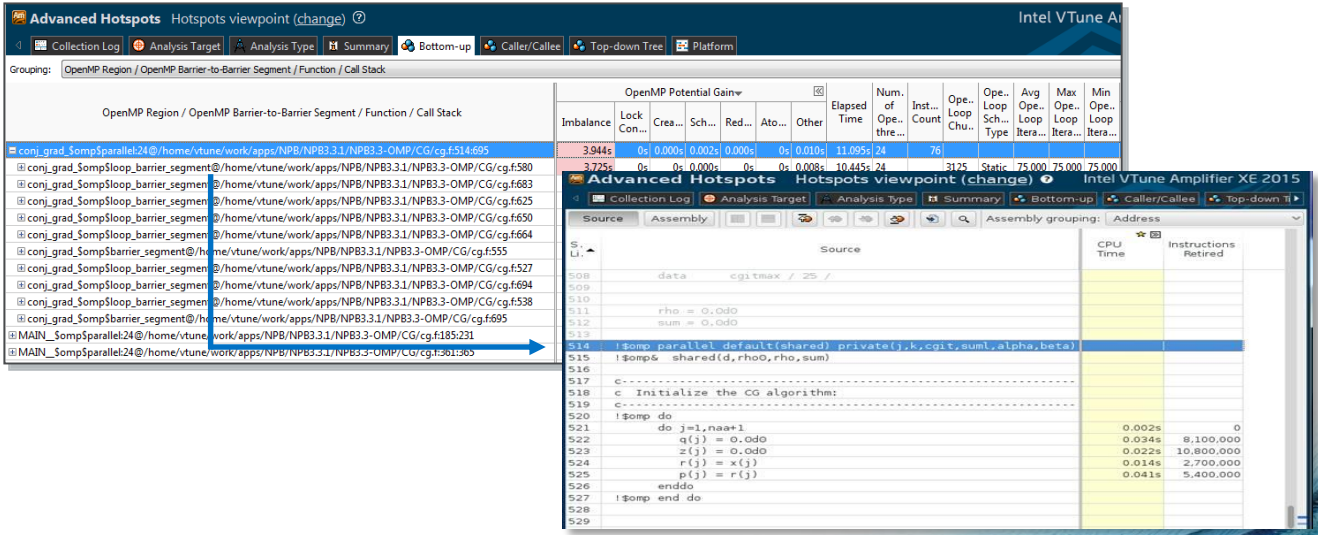
Time Filter to exclude initialization phase

VTune GUI: OpenMP scalable timeline



Intel® Xeon Phi™ profiling result with 288 threads

VTune GUI: OpenMP region source view



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® Trace Analyzer and Collector or Intel® 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
```

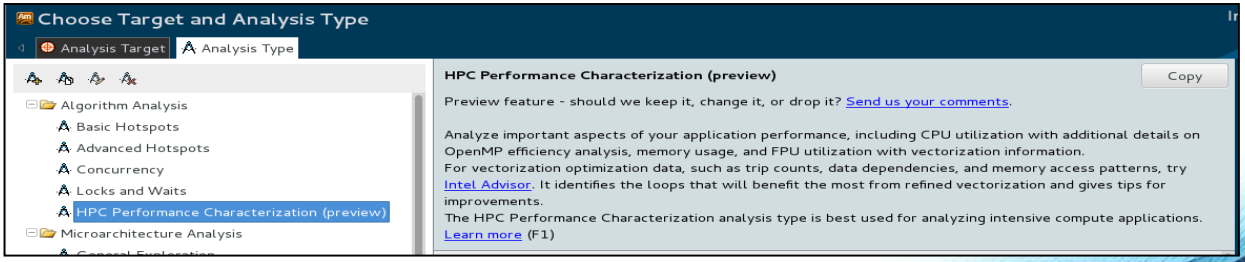
HPC Performance Analysis

HPC Performance Analysis

- CLI:

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

- GUI:



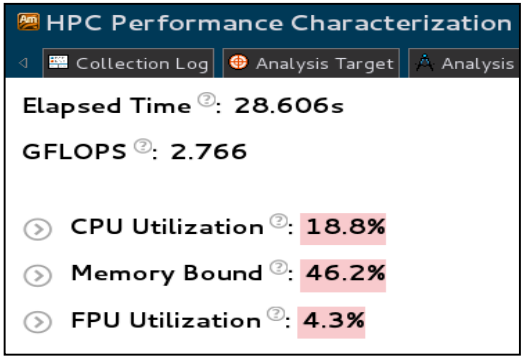
Analysis Structure and Metrics

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..)

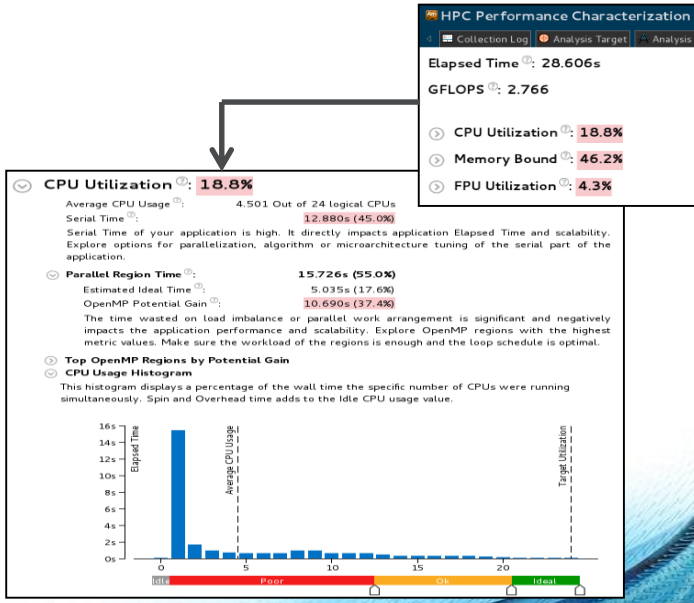
Performance Aspect: CPU Utilization

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



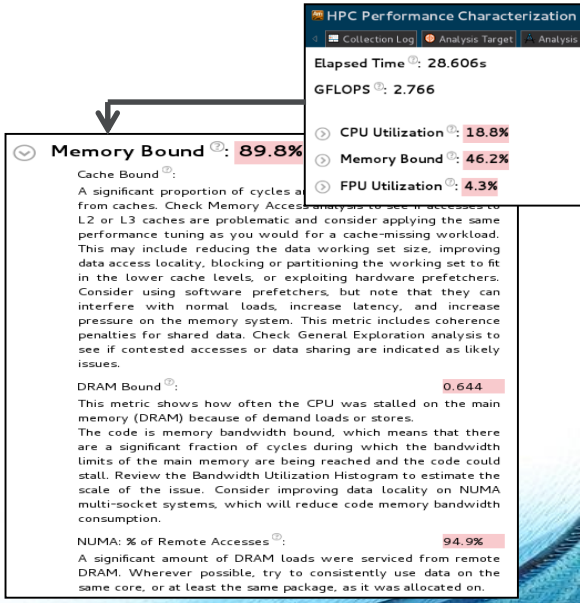
Performance Aspect: Memory Access

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



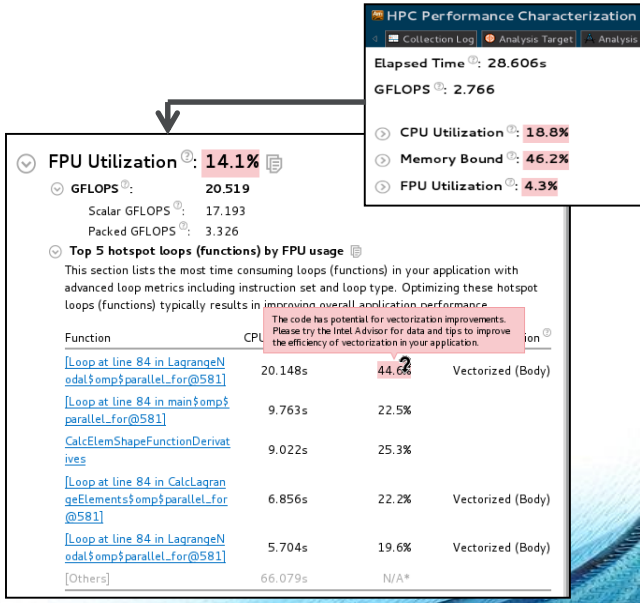
Performance Aspects: FPU Utilization

FPU utilization

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

Metrics in FPU utilization section

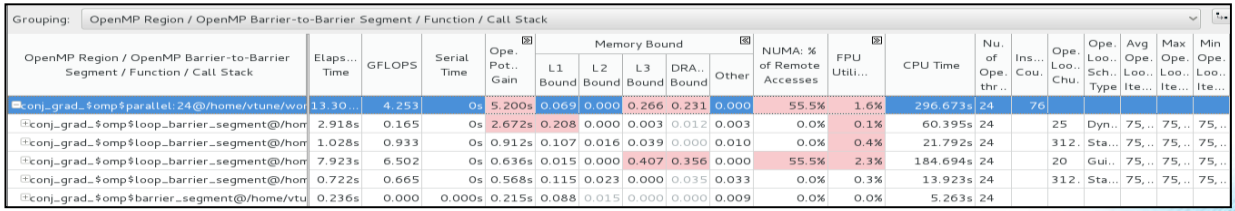
- GLOPs broken down by scalar and packed
- Top 5 loops/functions by FPU usage
- Dynamically generated issue descriptions on low FPU usage help to define the reason and next steps:
- Non-vectorizedvectorized with legacy instruction set
- memory bound limited loops not benefiting from vectorization etc.



Bottom-Up Grid View

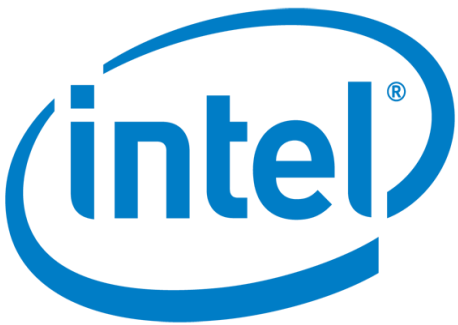
Metrics by OpenMP regions or functions/loops

- Regulated by choosing proper grouping
- Wall time/global metrics like elapsed time, GLOPs, serilal 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_ompparallel: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_omploop_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_omploop_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_omploop_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_omploop_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_ompbarrier_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						


intel®