Intel® VTune™ Amplifier XE

Dr. Michael Klemm
Software and Services Group
Developer Relations Division
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.

<table>
<thead>
<tr>
<th>Optimization Notice</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

Notice revision #20110804
Contents

• Introduction to Intel® VTune™ Amplifier XE
  • Features and analysis types
  • Command Line Interface (CLI)
  • Graphical User Interface (GUI)
• VTune™ Typical HPC workflow with MPI
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

<table>
<thead>
<tr>
<th>Software Collector</th>
<th>Hardware Collector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses OS interrupts</td>
<td>Uses the on chip Performance Monitoring Unit (PMU)</td>
</tr>
<tr>
<td>Collects from a single process tree</td>
<td>Collect system wide or from a single process tree.</td>
</tr>
<tr>
<td>~10ms default resolution</td>
<td>~1ms default resolution (finer granularity - finds small functions)</td>
</tr>
<tr>
<td>Either an Intel® or a compatible processor</td>
<td>Requires a genuine Intel® processor for collection</td>
</tr>
<tr>
<td>Call stacks show calling sequence</td>
<td>Optionally collect call stacks</td>
</tr>
<tr>
<td>Works in virtual environments</td>
<td>Works in a VM only when supported by the VM (e.g., vSphere*, KVM)</td>
</tr>
<tr>
<td>No driver required</td>
<td>Requires a driver</td>
</tr>
<tr>
<td></td>
<td>Easy to install on Windows</td>
</tr>
<tr>
<td></td>
<td>Linux requires root (or use default perf driver without stacks)</td>
</tr>
</tbody>
</table>

No special recompiles - C, C++, C#, Fortran, Java, Assembly
A rich set of analysis types

<table>
<thead>
<tr>
<th>Software Collector</th>
<th>Hardware Collector</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic Hotspots</strong></td>
<td><strong>Advanced Hotspots</strong></td>
</tr>
<tr>
<td>Which functions use the most time?</td>
<td>Which functions use the most time?</td>
</tr>
<tr>
<td></td>
<td>Where to inline? – Statistical call counts</td>
</tr>
<tr>
<td><strong>Concurrency</strong></td>
<td><strong>General Exploration</strong></td>
</tr>
<tr>
<td>Tune parallelism. Colors show number of cores used.</td>
<td>Where is the biggest opportunity?</td>
</tr>
<tr>
<td></td>
<td>Cache misses? Branch mispredictions?</td>
</tr>
<tr>
<td><strong>Locks and Waits</strong></td>
<td><strong>Advanced Analysis</strong></td>
</tr>
<tr>
<td>Tune the #1 cause of slow threaded performance:</td>
<td>Dig deep to tune access contention, memory analysis, etc.</td>
</tr>
<tr>
<td>– waiting with idle cores.</td>
<td></td>
</tr>
<tr>
<td>Any IA86 processor, any VM, no driver</td>
<td>Higher res., lower overhead, system wide</td>
</tr>
</tbody>
</table>

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
  - -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

```shell
> 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[report] \ <report\ \ type>\ \ [-report\ -option]\n  \• \ The \ type \ of \ report \ defined \ with \ report\ \ type\n  \• \ Report \ type \ defines \ the \ set \ of \ available \ report-\n    \ option \ modifiers\n  
• 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

- Analysis types available for the current project
- Open/close project panel
- Recent projects
- Project / result management
- Recent results
VTune GUI: choose analysis view

Available analysis types

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)

Filter by process & other controls

Tuning opportunities shown in pink. Hover for tips

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

SOFTWARE AND SERVICES
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?

<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU Time Difference</th>
<th>Module</th>
<th>CPU Timer:007s</th>
<th>CPU Timer:006s</th>
</tr>
</thead>
<tbody>
<tr>
<td>FireObject::checkCollision</td>
<td>4.850s</td>
<td>SystemProceduralFire.DLL</td>
<td>6.281s</td>
<td>1.431s</td>
</tr>
<tr>
<td>FireObject::ProcessFireCollisionsRange</td>
<td>4.644s</td>
<td>SystemProceduralFire.DLL</td>
<td>5.643s</td>
<td>0.999s</td>
</tr>
<tr>
<td>dllStopPlugin</td>
<td>3.765s</td>
<td>RenderSystem_Direct3D9.DLL</td>
<td>9.184s</td>
<td>5.419s</td>
</tr>
</tbody>
</table>

SOFTWARE AND SERVICES
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.

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

- 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)

Actual Parallel Region Elapsed Time

Fork

Join

Effective time (sampling)
Imbalance (tracing)
Lock spinning (sampling)
Scheduling (sampling)
Work creation (sampling)
Atomics (sampling)
Reduction (sampling)

Potential Gain as a sum of inefficiencies normalized by number of threads

Effective time / Number of Threads

Estimated Ideal Time =

SOFTWARE AND SERVICES
VTune GUI: OpenMP region view (2/2)

### Imbalance

<table>
<thead>
<tr>
<th>OpenMP Region</th>
<th>Function</th>
<th>Call Stack</th>
<th>Imbalance (%)</th>
<th>Lock Con.</th>
<th>Creation</th>
<th>Schd.</th>
<th>Reduction</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>congrad_sompparallel24@/home/vx/ora/apps/NP3.3.1/NP3.3-OMP-05/CG/cg:553.g605</td>
<td></td>
<td></td>
<td>34.6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>MAIN_sompparallel24@/home/vx/ora/apps/NP3.3.1/NP3.3-OMP-05/CG/cg:553.g231</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Serial - outside any region</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>MAIN_sompparallel24@/home/vx/ora/apps/NP3.3.1/NP3.3-OMP-05/CG/cg:553.g345</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>MAIN_sompparallel24@/home/vx/ora/apps/NP3.3.1/NP3.3-OMP-05/CG/cg:553.g345</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>MAIN_sompparallel24@/home/vx/ora/apps/NP3.3.1/NP3.3-OMP-05/CG/cg:553.g345</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

### Dynamic scheduling overhead

<table>
<thead>
<tr>
<th>OpenMP Region</th>
<th>Function</th>
<th>Call Stack</th>
<th>Imbalance (%)</th>
<th>Lock Con.</th>
<th>Creation</th>
<th>Schd.</th>
<th>Reduction</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>congrad_sompparallel24@/home/vx/ora/apps/NP3.3.1/NP3.3-OMP-05/CG/cg:553.g605</td>
<td></td>
<td></td>
<td>25.9</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>MAIN_sompparallel24@/home/vx/ora/apps/NP3.3.1/NP3.3-OMP-05/CG/cg:553.g231</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Serial - outside any region</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>MAIN_sompparallel24@/home/vx/ora/apps/NP3.3.1/NP3.3-OMP-05/CG/cg:553.g345</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>MAIN_sompparallel24@/home/vx/ora/apps/NP3.3.1/NP3.3-OMP-05/CG/cg:553.g345</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>MAIN_sompparallel24@/home/vx/ora/apps/NP3.3.1/NP3.3-OMP-05/CG/cg:553.g345</td>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**SOFTWARE AND SERVICES**
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
VTune GUI: OpenMP Barrier-to-Barrier view (2/2)

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

**Imbalance on a loop barrier**

| OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack | OpenMP Potential Gain | OpenMP Potential Gain (Gain % of Collection Thread) | Elapsed Time | Number of OpenMP Threads | Ins... | OpenMP Loop Schedule Type | OpenMP Loop Chunk | Avg OpenMP Loop Iteration Count |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| con.grad_tmpparallel.2468/hr directory | 3.945% | 0.020% | 0.020% | 0.003s | 11.99s | 24 | 76 | Static | 3125 | 75,000 |
| con.grad_tmpparallel.barrier.parallel@/home/vtune/work/apps/NPB/NPB3.1/NPB3.1-OMP/Cfg/514-695 | 3.725% | 0.016% | 0.016% | 0.002s | 11.88s | 24 | 76 | Static | 3125 | 75,000 |

**Dynamic scheduling overhead on a parallel loop**

- Dynamic scheduling overhead on a parallel loop

SOFTWARE AND SERVICES
VTune GUI: OpenMP serial hotspots

- **Serial hotspots under Master Thread**
- **Time Filter to exclude initialization phase**
VTune GUI: OpenMP scalable timeline

Region frames on the ruler

More green color – more efficient multithreaded execution

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:
  \[
  \langle \text{vtune\_command} \rangle \ [\--] \ \langle \text{mpi\_command} \rangle \ \langle \text{application} \rangle
  \]

  \[
  > \text{amplxe-cl -collect advanced-hotspots -r result\_dir -- mpirun -np 48 ./mpi\_app}
  \]

- 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:**
  
  ![HPC Performance Characterization](image)

  - 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.
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 with legacy instruction set
  • memory bound limited loops not benefiting from vectorization etc.

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU time</th>
<th>FPU Utilization</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="empParallelFor@S811">Loop at line 84 in LagrangeNode</a></td>
<td>20.148s</td>
<td>44%</td>
<td>Vectorized (Body)</td>
</tr>
<tr>
<td><a href="empParallelFor@S811">Loop at line 84 in main</a></td>
<td>9.763s</td>
<td>22.5%</td>
<td></td>
</tr>
<tr>
<td>CalcElemShapeFunctionDerivatives</td>
<td>9.022s</td>
<td>25.3%</td>
<td></td>
</tr>
<tr>
<td><a href="empParallelFor@S811">Loop at line 84 in CalcLagrangeElements</a></td>
<td>6.856s</td>
<td>22.2%</td>
<td>Vectorized (Body)</td>
</tr>
<tr>
<td><a href="empParallelFor@S811">Loop at line 84 in LagrangeNode</a></td>
<td>5.704s</td>
<td>10.6%</td>
<td>Vectorized (Body)</td>
</tr>
<tr>
<td>Others</td>
<td>66.079s</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

GFLOPS:
- Scalar GFLOPS: 17.193
- Packed GFLOPS: 3.126

FPU Utilization: 14.1%

• CPU Utilization: 18.8%
• Memory Bound: 46.2%
• FPU Utilization: 4.3%
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