Optimize Performance with Intel® VTune™ Profiler

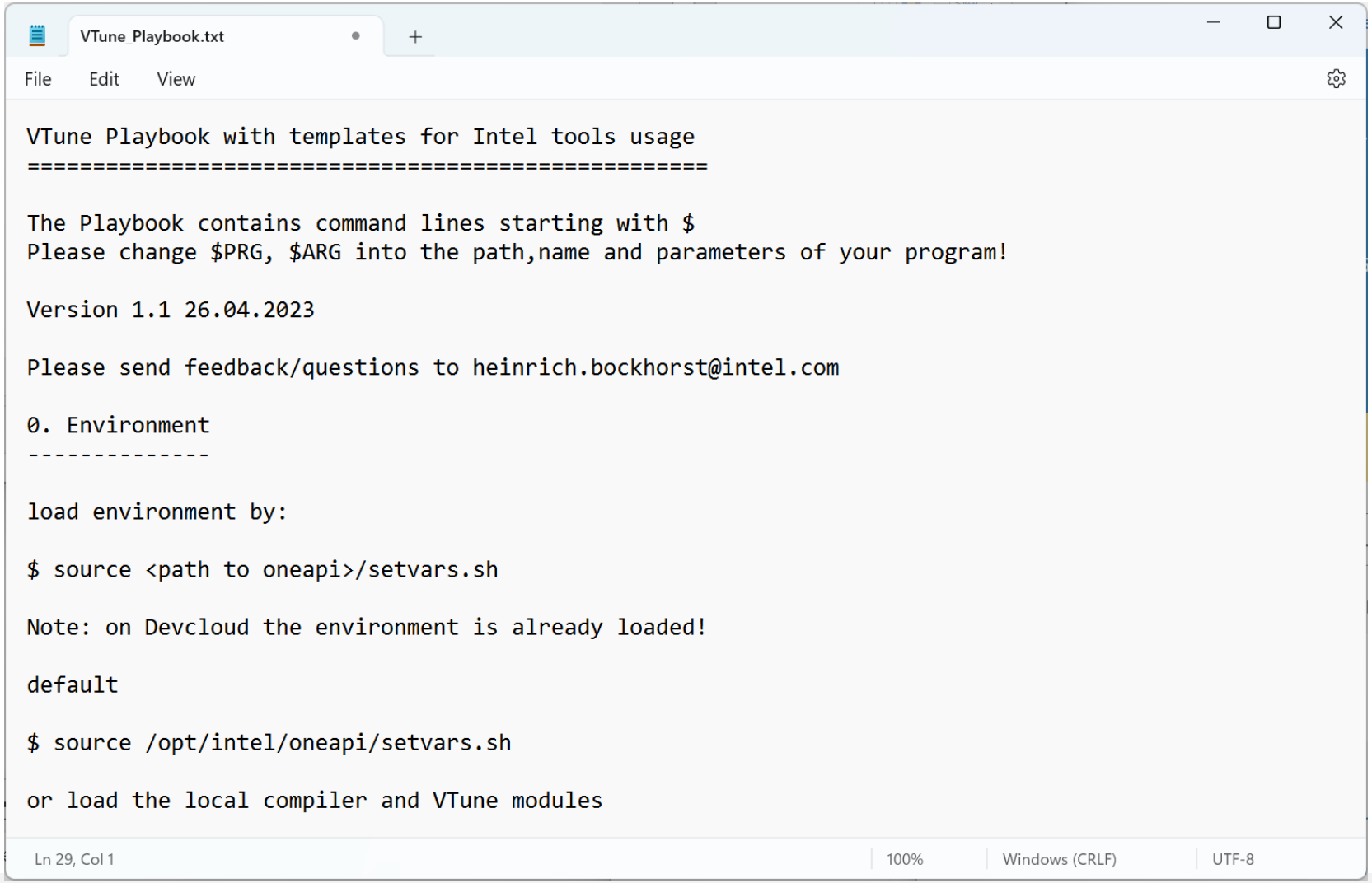
Heinrich Bockhorst

HLRS, September 15th

Agenda

- VTune overview + VTune hotspots
- How to use VTune e.g., in a Cluster environment
- Demo
- VTune HPC analysis
- VTune GPU analysis
- Demo with GROMACS

Playbook for easy access to command lines



Demo slides/life

- Presentations contain sample snapshots
- Demo content might not match exactly these snapshots

Analysis based on APS results

- APS is first step in analysis.
- Check for hints provided in APS results
- APS provides only data on whole app
- VTune will provide function/loops/source code/assembly level analysis

Optimize Performance

Intel® VTune™ Profiler

Get the Right Data to Find Bottlenecks

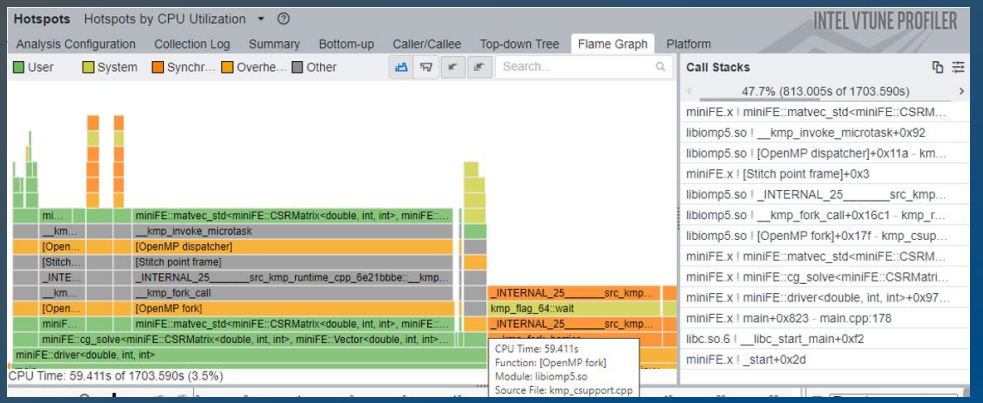
- A suite of profiling for CPU, GPU, FPGA, threading, memory, cache, storage, offload, power…
- Application or system-wide analysis
- DPC++, C, C++, Fortran, Python*, Go*, Java*, or a mix
- Linux, Windows, FreeBSD, Android, Yocto and more
- Containers and VMs

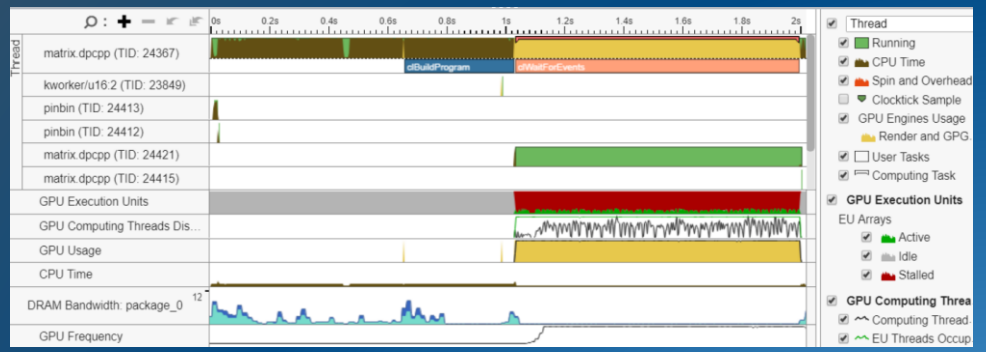
Analyze Data Faster

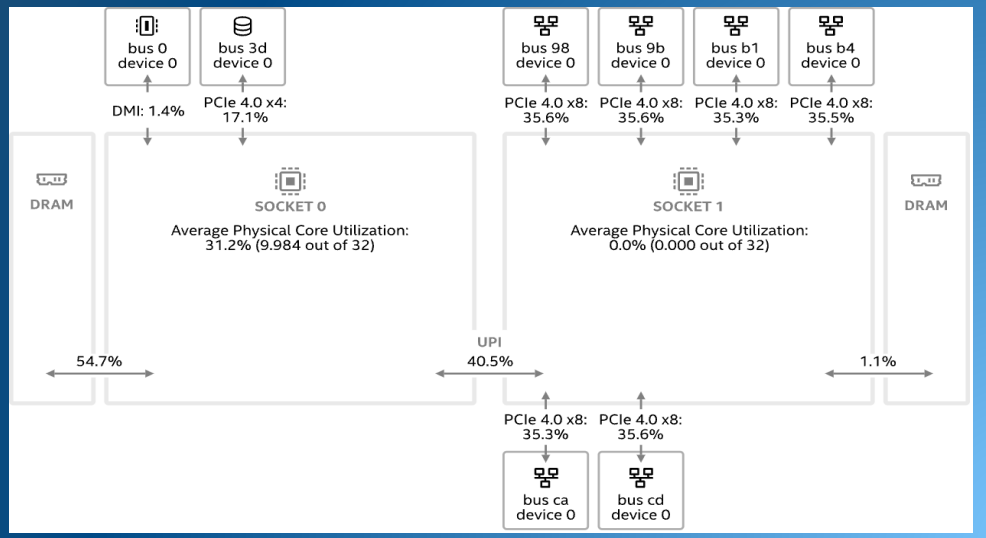
- Collect data HW/SW sampling and tracing w/o re-compilation
- See results on your source, in architecture diagrams, as a histogram, on a timeline…
- Filter and organize data to find answers

Work Your Way

- User interface or command line
- Profile locally and remotely
- GUI (desktop or web) or command line







Find Answers Fast

Intel® VTune™ Profiler

Adjust Data Grouping

Function / Call Stack

Source Function / Function / Call Stack

Sync Object / Function / Call Stack

Sync Object / Thread / Function / Call Stack

... (Partial list shown)

Double Click Function to View Source

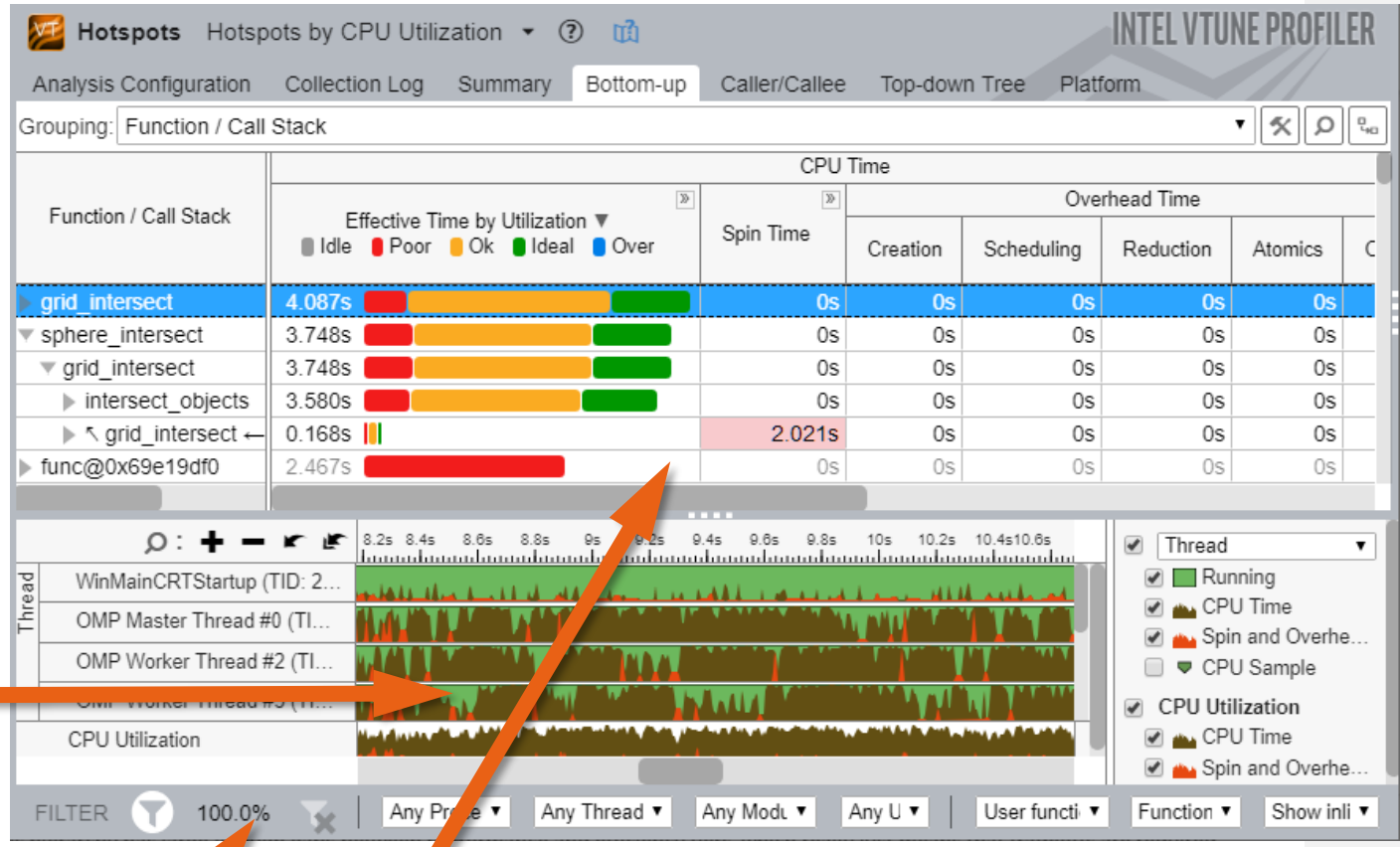
Click [▶] for Call Stack

Filter by Timeline Selection (or by Grid Selection)

Zoom In And Filter On Selection

Filter In by Selection

Remove All Filters



Filter by Process & Other Controls

Tuning Opportunities Shown in Pink. Hover for Tips

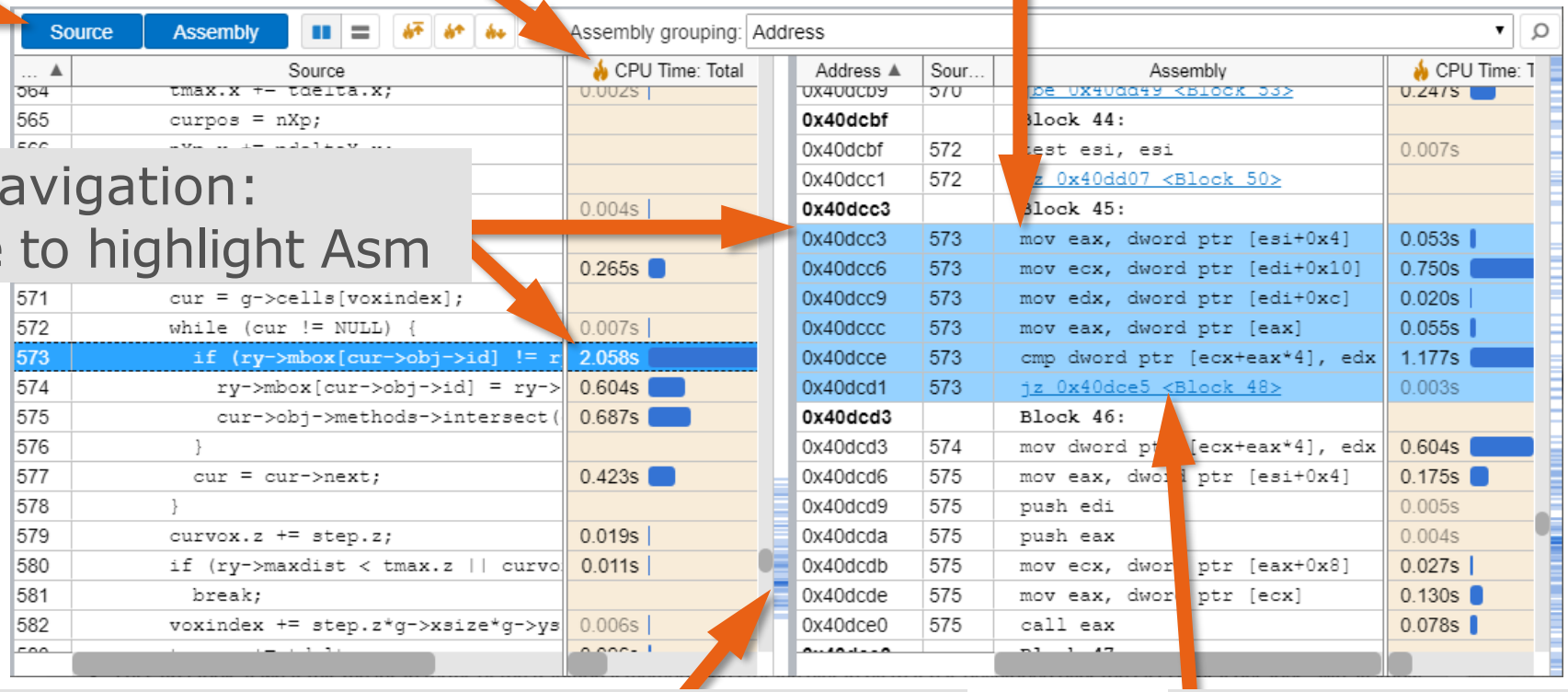
See Profile Data On Source / Asm

Double Click from Grid or Timeline

View Source / Asm or both

CPU Time

Right click for instruction reference manual



Quick Asm navigation:
Select source to highlight Asm

Scroll Bar “Heat Map” is an overview of hot spots

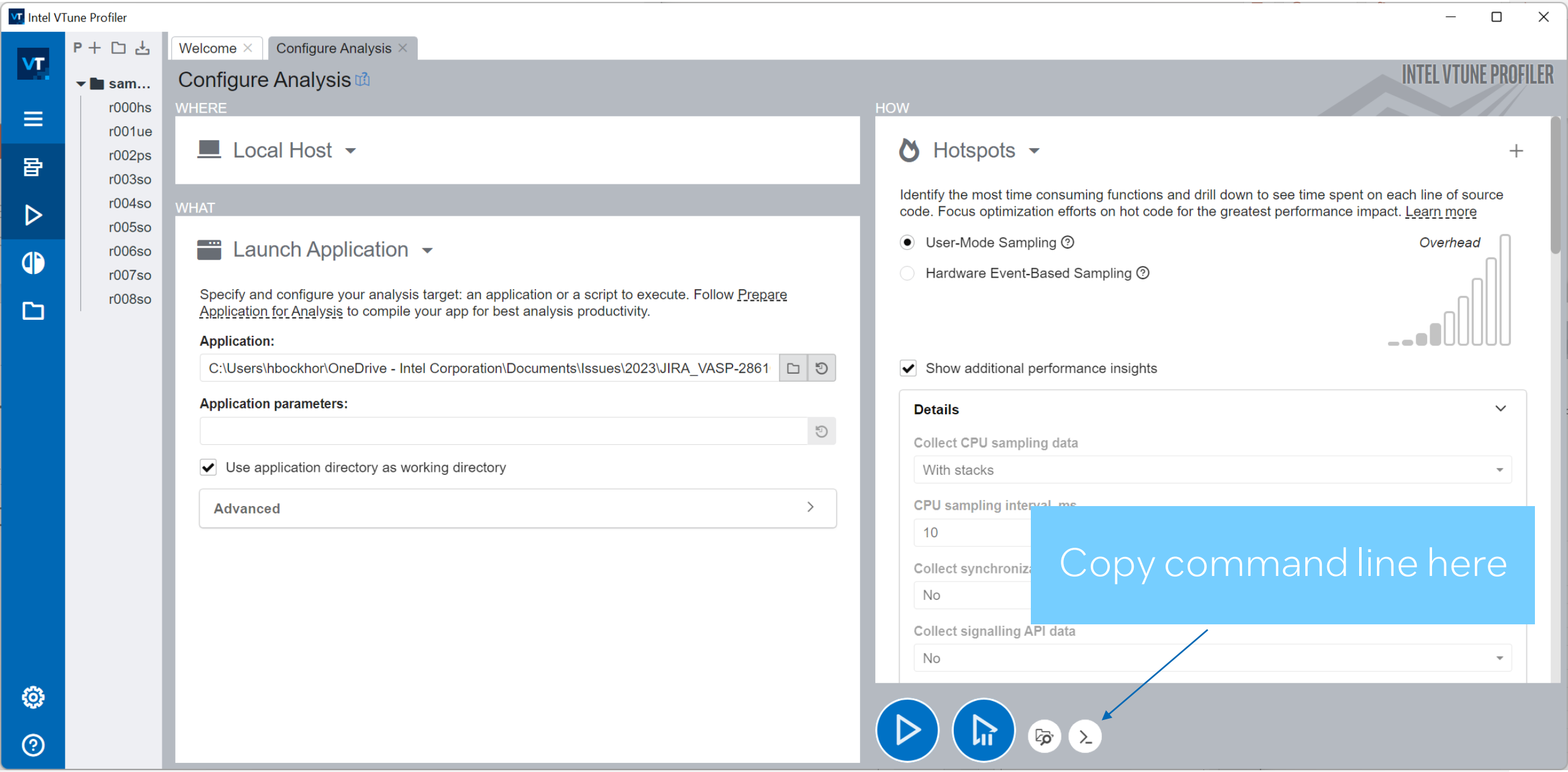
Click jump to scroll Asm

Bottom-up tab – most popular tab

- Grouping: different ordering of results –check out different choices
- Source view: double click on function or loop will open another window – source must be compiled with “-g”. Source code must be available
- Zoom and filter in timeline section. Grid will adapt your choice
- Filter process/user code/libraries/loops/system functions (e.g. libc)
- Values in pink: e.g. high overhead like barrier waiting or low cpu utilization.

How to start an analysis GUI vs CMD

- Start GUI by `$ vtune-gui` on command line or by double clicking on Windows
- Click on “Configure Analysis …”
- 3 Sections: WHERE,WHAT, HOW
- WHERE: local vs remote. We do local here
- WHAT : define your application with parameters and environment
- HOW: Analysis type like “Hotspots” or “APS” with additional parameters


Intel VTune Profiler
Welcome
Configure Analysis
Configure Analysis
INTEL VTUNE PROFILER
WHERE
Local Host
WHAT
Launch Application
Specify and configure your analysis target: an application or a script to execute. Follow Prepare Application for Analysis to compile your app for best analysis productivity.
Application:
C:\Users\hbockhor\OneDrive - Intel Corporation\Documents\Issues\2023\JIRA_VASP-2861
Application parameters:
Use application directory as working directory
Advanced
HOW
Hotspots
Identify the most time consuming functions and drill down to see time spent on each line of source code. Focus optimization efforts on hot code for the greatest performance impact. Learn more
User-Mode Sampling
Hardware Event-Based Sampling
Overhead
Show additional performance insights
Details
Collect CPU sampling data
With stacks
CPU sampling interval, ms
10
Collect synchroniz
No
Collect signalling API data
No
Copy command line here

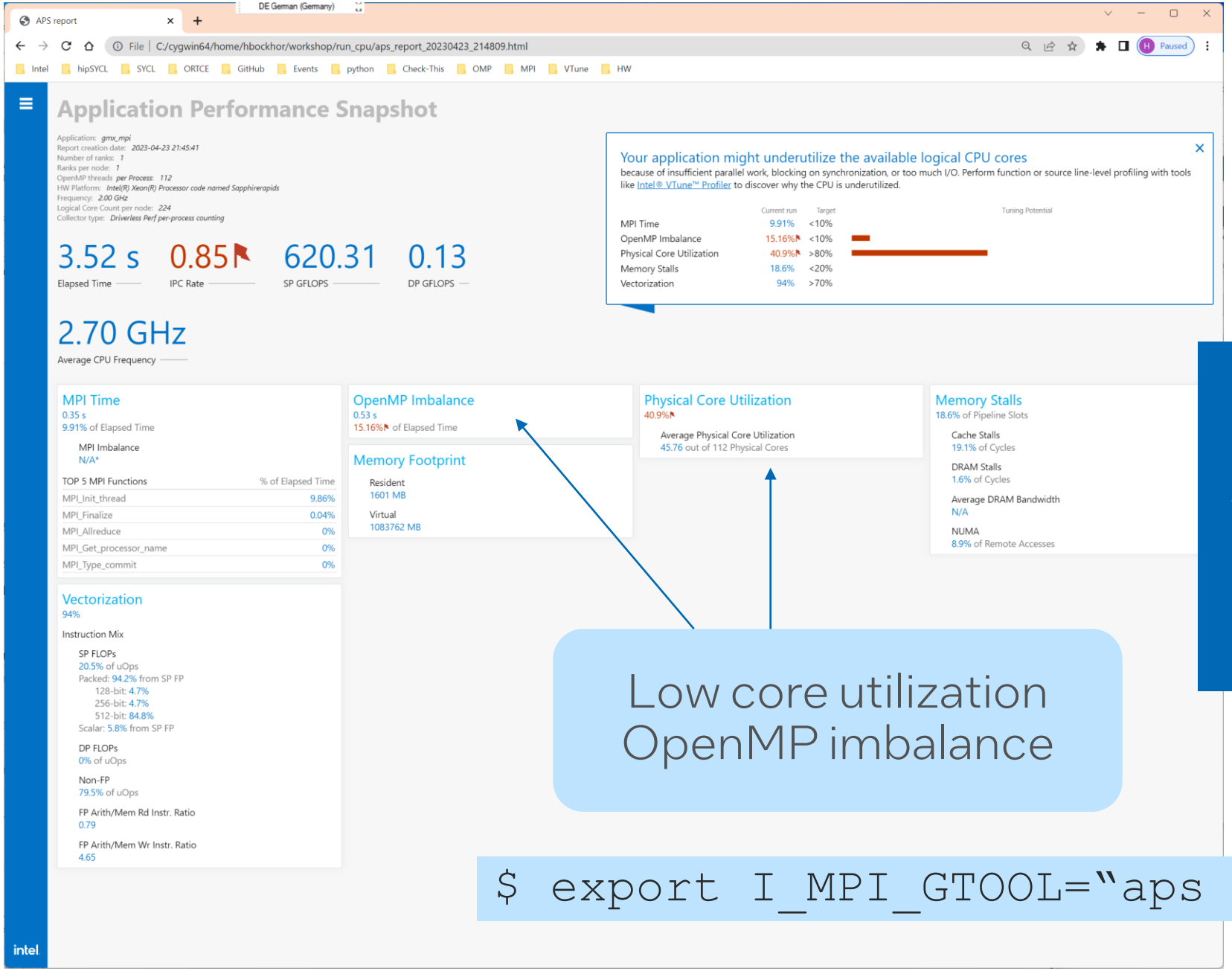
VTune and MPI

- To run VTune in an MPI job you may use the “-gtool” flag
- More convenient is the I_MPI_GTOOL environment variable. Example for HPC analysis:

  ```
  $ export I_MPI_GTOOL= "vtune -c hpc-performance -r HPC:0"
  ```

  run your program, as usual, under MPI. The setting will collect data on rank #0. Use a list of ranks or `:all` for multi rank analysis.

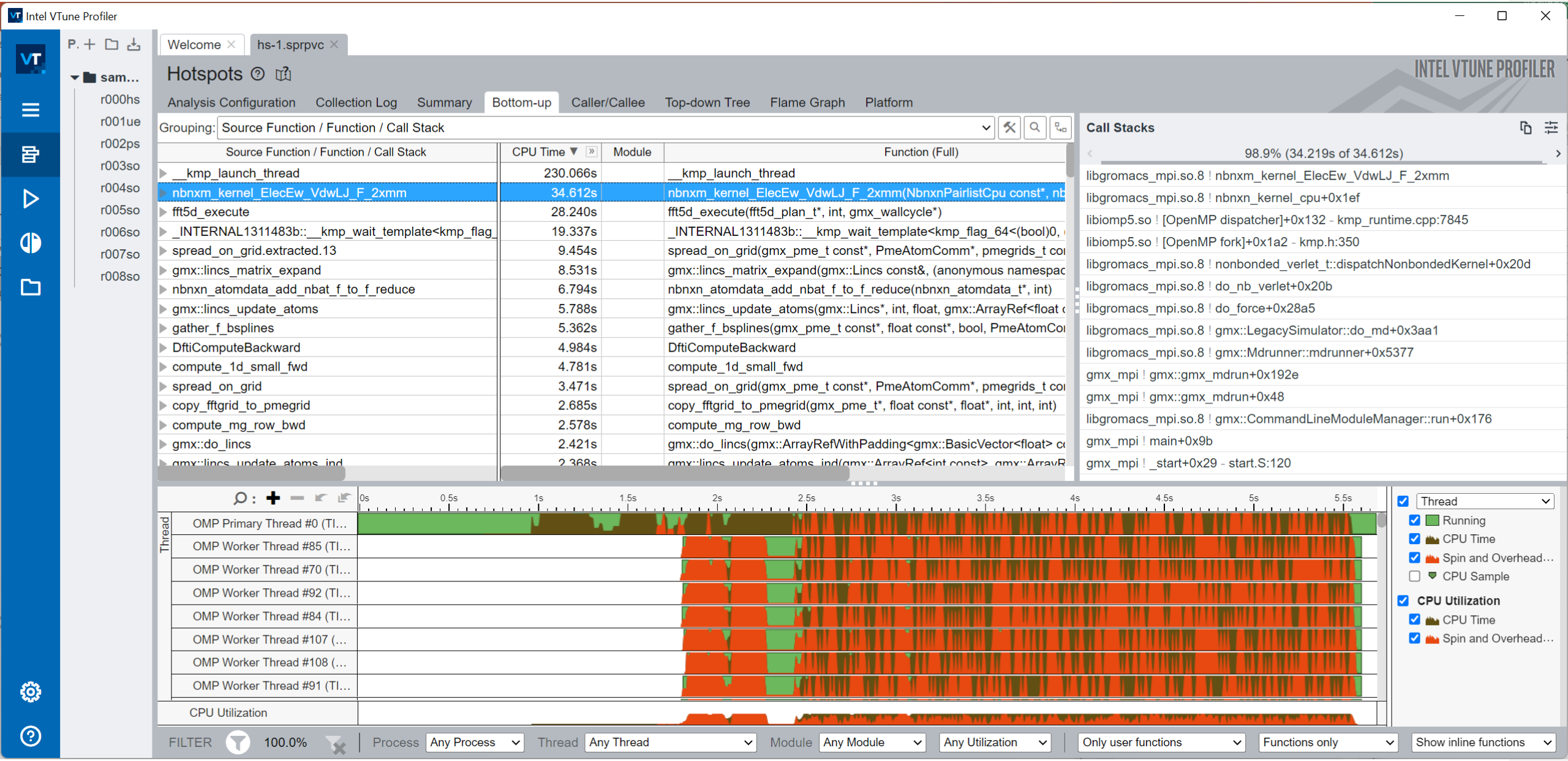
- More information:
  https://www.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top/command-reference/mpiexec-hydra/gtool-options.html

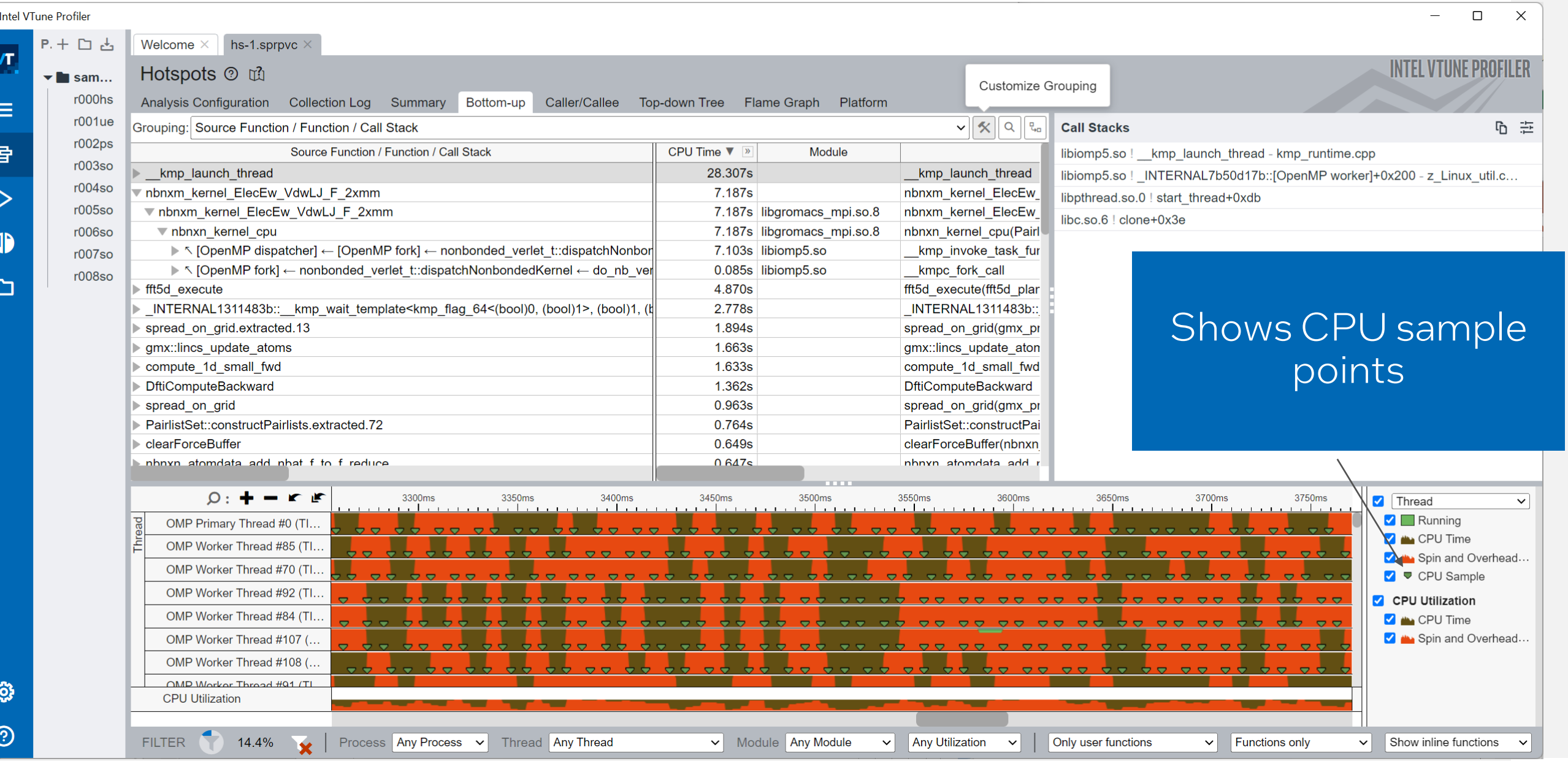


Low core utilization
OpenMP imbalance

Note! This uses switched off pinning just to show VTune features!

```
$ export I_MPI_GTOOL="aps -r result :all"
```


Intel VTune Profiler
Welcome
hs-1.sprpvc
Hotspots
INTEL VTUNE PROFILER
Analysis Configuration
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Flame Graph
Platform
Grouping: Source Function / Function / Call Stack
Source Function / Function / Call Stack
CPU Time
Module
Function (Full)
__kmp_launch_thread 230.066s
nbnxm_kernel_ElecEw_VdwLJ_F_2xmm 34.612s
fft5d_execute 28.240s
_INTERNAL1311483b::__kmp_wait_template<kmp_flag_ 19.337s
spread_on_grid.extracted.13 9.454s
gmx::lincs_matrix_expand 8.531s
nbnxn_atomdata_add_nbat_f_to_f_reduce 6.794s
gmx::lincs_update_atoms 5.788s
gather_f_bsplines 5.362s
DftiComputeBackward 4.984s
compute_1d_small_fwd 4.781s
spread_on_grid 3.471s
copy_fftgrid_to_pmegrid 2.685s
compute_mg_row_bwd 2.578s
gmx::do_lincs 2.421s
Call Stacks
98.9% (34.219s of 34.612s)
libgromacs_mpi.so.8 ! nbnxm_kernel_ElecEw_VdwLJ_F_2xmm
libgromacs_mpi.so.8 ! nbnxn_kernel_cpu+0x1ef
libiomp5.so ! [OpenMP dispatcher]+0x132 - kmp_runtime.cpp:7845
libiomp5.so ! [OpenMP fork]+0x1a2 - kmp.h:350
libgromacs_mpi.so.8 ! nonbonded_verlet_t::dispatchNonbondedKernel+0x20d
libgromacs_mpi.so.8 ! do_nb_verlet+0x20b
libgromacs_mpi.so.8 ! do_force+0x28a5
libgromacs_mpi.so.8 ! gmx::LegacySimulator::do_md+0x3aa1
libgromacs_mpi.so.8 ! gmx::Mdrunner::mdrunner+0x5377
gmx_mpi ! gmx::gmx_mdrun+0x192e
gmx_mpi ! gmx::gmx_mdrun+0x48
libgromacs_mpi.so.8 ! gmx::CommandLineModuleManager::run+0x176
gmx_mpi ! main+0x9b
gmx_mpi ! _start+0x29 - start.S:120
0s 0.5s 1s 1.5s 2s 2.5s 3s 3.5s 4s 4.5s 5s 5.5s
Thread
OMP Primary Thread #0 (TI...
OMP Worker Thread #85 (TI...
OMP Worker Thread #70 (TI...
OMP Worker Thread #92 (TI...
OMP Worker Thread #84 (TI...
OMP Worker Thread #107 (...
OMP Worker Thread #108 (...
OMP Worker Thread #91 (TI...
CPU Utilization
Thread
Running
CPU Time
Spin and Overhead...
CPU Sample
CPU Utilization
CPU Time
Spin and Overhead...
FILTER 100.0%
Process Any Process
Thread Any Thread
Module Any Module
Any Utilization
Only user functions
Functions only
Show inline functions


Intel VTune Profiler
Welcome
hs-1.sprpvc
Hotspots
INTEL VTUNE PROFILER
Customize Grouping
Analysis Configuration
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Flame Graph
Platform
Grouping: Source Function / Function / Call Stack
Source Function / Function / Call Stack
CPU Time
Module
__kmp_launch_thread 28.307s __kmp_launch_thread
nbnxm_kernel_ElecEw_VdwLJ_F_2xmm 7.187s nbnxm_kernel_ElecEw_
nbnxm_kernel_ElecEw_VdwLJ_F_2xmm 7.187s libgromacs_mpi.so.8 nbnxm_kernel_ElecEw_
nbnxn_kernel_cpu 7.187s libgromacs_mpi.so.8 nbnxn_kernel_cpu(Pairl
[OpenMP dispatcher] ← [OpenMP fork] ← nonbonded_verlet_t::dispatchNonbor 7.103s libiomp5.so __kmp_invoke_task_fur
[OpenMP fork] ← nonbonded_verlet_t::dispatchNonbondedKernel ← do_nb_ver 0.085s libiomp5.so __kmpc_fork_call
fft5d_execute 4.870s fft5d_execute(fft5d_plar
_INTERNAL1311483b::__kmp_wait_template<kmp_flag_64<(bool)0, (bool)1>, (bool)1, (b 2.778s _INTERNAL1311483b::
spread_on_grid.extracted.13 1.894s spread_on_grid(gmx_pr
gmx::lincs_update_atoms 1.663s gmx::lincs_update_aton
compute_1d_small_fwd 1.633s compute_1d_small_fwd
DftiComputeBackward 1.362s DftiComputeBackward
spread_on_grid 0.963s spread_on_grid(gmx_pr
PairlistSet::constructPairlists.extracted.72 0.764s PairlistSet::constructPai
clearForceBuffer 0.649s clearForceBuffer(nbnxn
Call Stacks
libiomp5.so ! __kmp_launch_thread - kmp_runtime.cpp
libiomp5.so ! _INTERNAL7b50d17b::[OpenMP worker]+0x200 - z_Linux_util.c…
libpthread.so.0 ! start_thread+0xdb
libc.so.6 ! clone+0x3e
Shows CPU sample points
3300ms
3350ms
3400ms
3450ms
3500ms
3550ms
3600ms
3650ms
3700ms
3750ms
Thread
OMP Primary Thread #0 (TI…
OMP Worker Thread #85 (TI…
OMP Worker Thread #70 (TI…
OMP Worker Thread #92 (TI…
OMP Worker Thread #84 (TI…
OMP Worker Thread #107 (…
OMP Worker Thread #108 (…
CPU Utilization
Thread
Running
CPU Time
Spin and Overhead…
CPU Sample
CPU Utilization
CPU Time
Spin and Overhead…
FILTER
14.4%
Process
Any Process
Thread
Any Thread
Module
Any Module
Any Utilization
Only user functions
Functions only
Show inline functions

More Resources

Intel® VTune™ Profiler – Performance Profiler

- Product page – overview, features, FAQs…
- Training materials – Cookbooks, User Guide, Processor Tuning Guides
- Support Forum
- Online Service Center - Secure Priority Support
- What's New?

Additional Analysis Tools

- Intel® Advisor – Design code for efficient vectorization, threading, memory usage, and accelerator offload
- Intel® Inspector – memory and thread checker/ debugger
- Intel® Trace Analyzer and Collector - MPI Analyzer and Profiler

Additional Development Products

- oneAPI: A new era of heterogenous computing

Summary

- Hotspots is the most common analysis after APS
- After Summary you may try bottom up
- Check out grouping and filtering features
- Many tutorials + YouTube videos available

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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. https://software.intel.com/en-us/articles/optimization-notice

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. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.


intel