Intel® Developer Workshop
Intel VTune™

Dr. Heinrich Bockhorst

April 2026

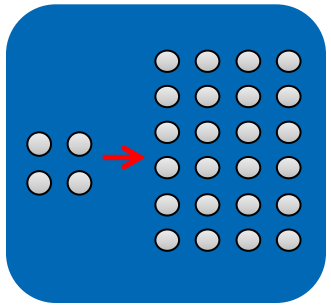
Agenda

- Application Performance Snapshot (APS) – Start of any analysis
- APS MPI features
- APS Example GEM
- VTune usage in Cluster Environment
- VTune hotspots including flame graphs
- VTune HPC Performance including OMP analysis

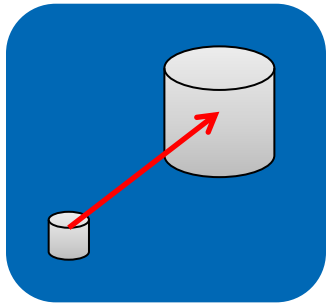
How to start an HPC analysis?

Application Performance Snapshot

Scalable Profiling for MPI

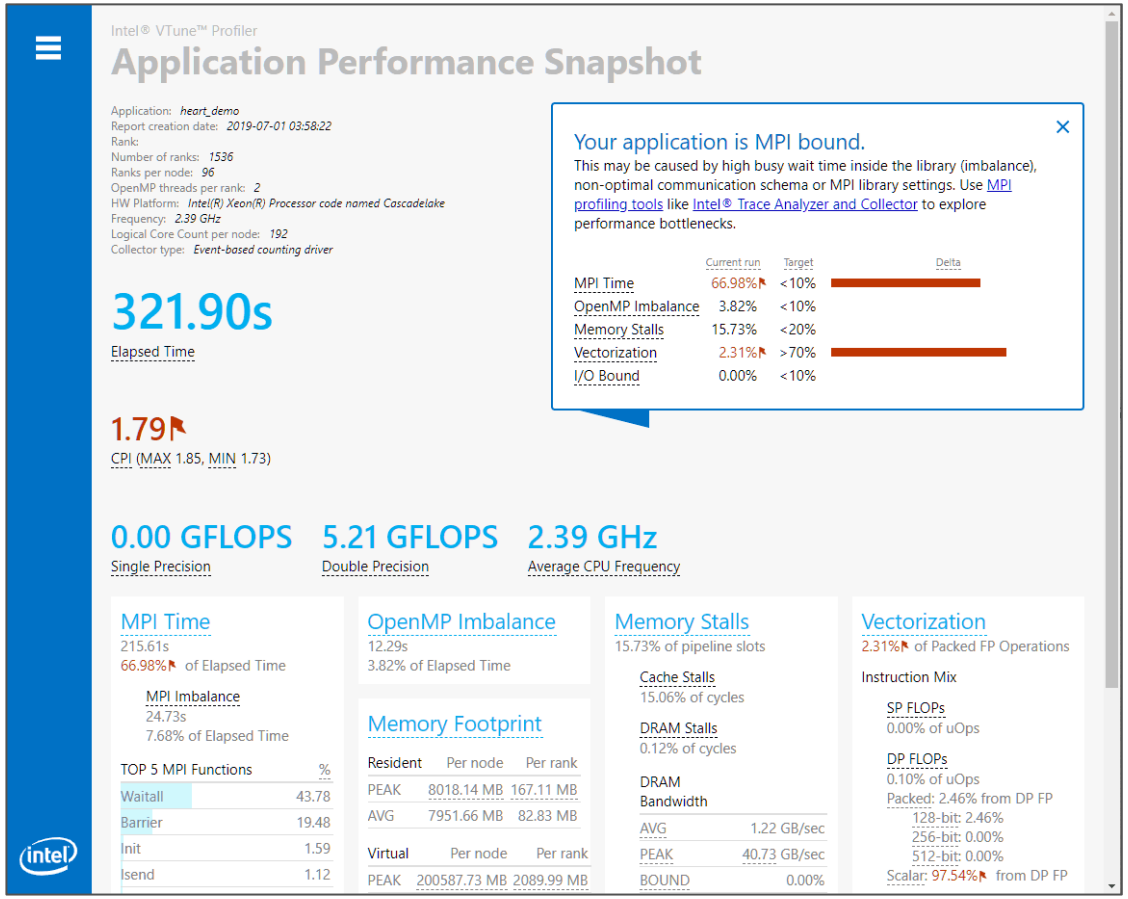


- **Lightweight** : Low overhead profiling.



- **Scalable:** Performance variation at scale can be detected sooner.

- **Identify Key Metrics:** Shows MPI*/OpenMP* imbalances.



How to do an APS analysis

Setup Environment

$ module load intel-vtune # currently 2025.7

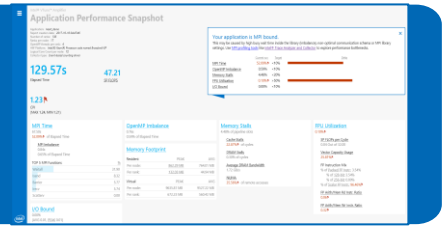
Run Application

$ aps <application and args>

MPI: $ mpirun <mpi options> aps <application and args>

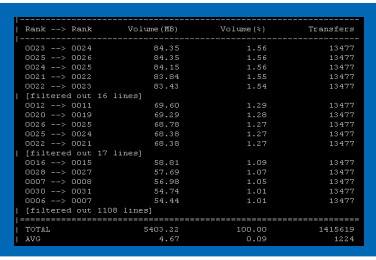
Generate Report on Result Folder

$ aps –report <result folder>



Generate CL reports with detailed MPI statistics on Result Folder

$ aps-report –<option> <result folder>



5

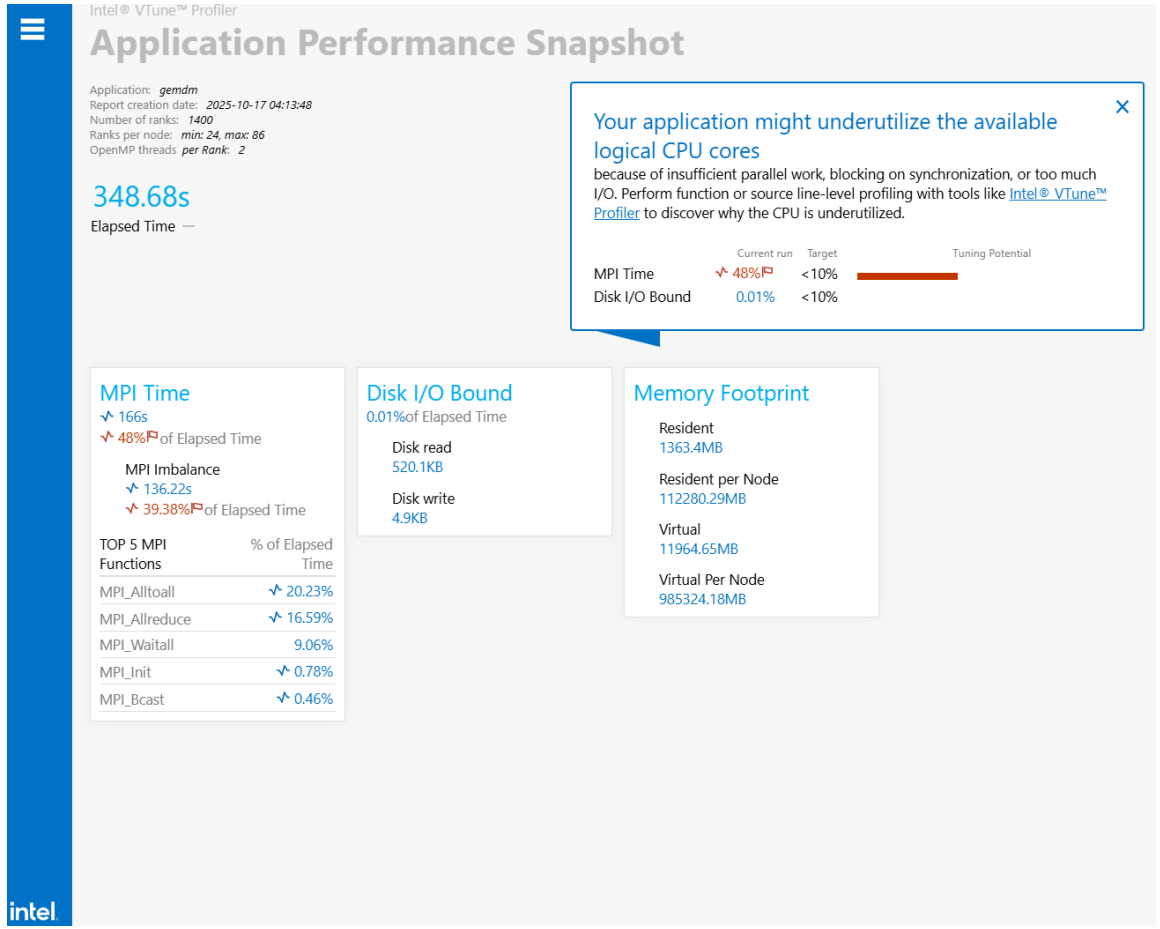
Example output GEM

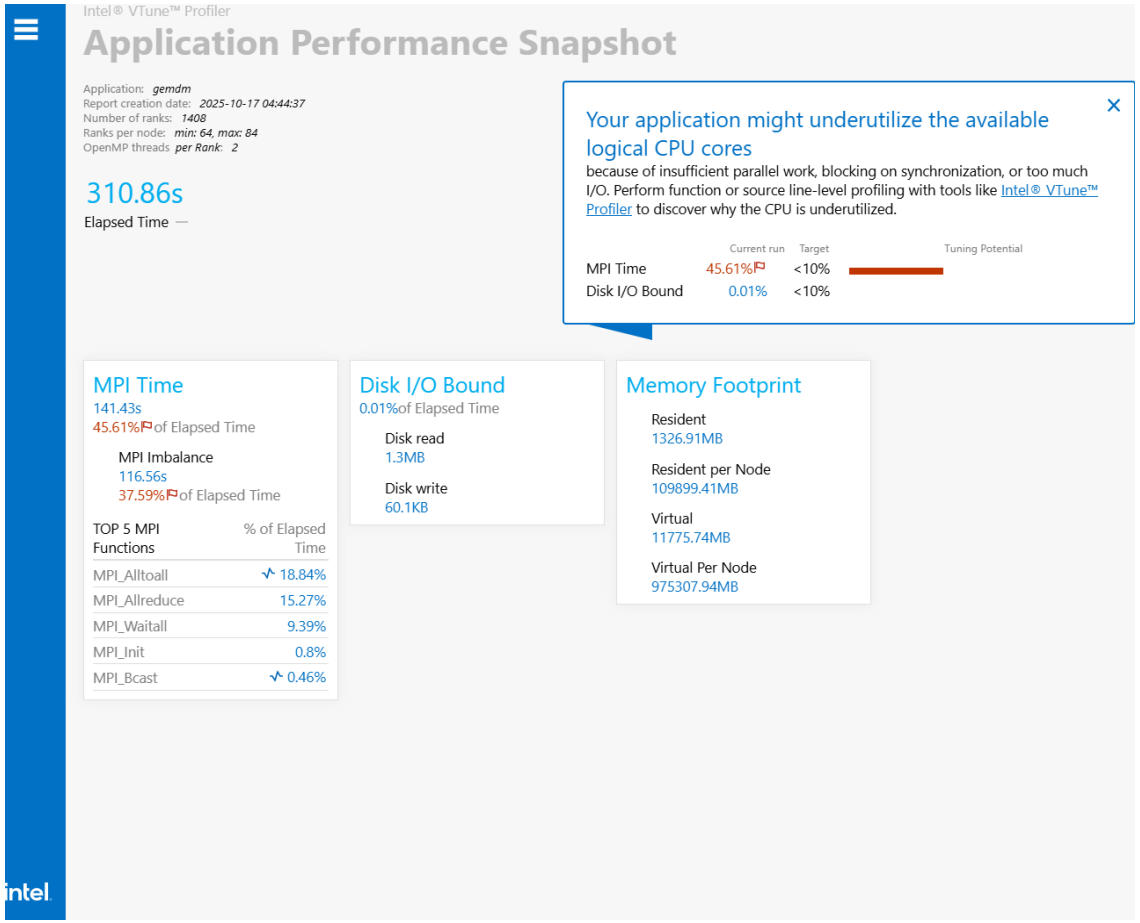
- Global Environmental Multiscale Model. From ECCC (Canadian Climate…)

  https://collaboration.cmc.ec.gc.ca/science/rpn/gef_html_public/index.html

- Benchmarking on 86 core GNR Server with 17 nodes and ~1400 ranks
- Different ways to distribute ranks on nodes: ppn=86 vs. ppn=84

GEM APS output with 2 ways of rank mapping





APS MPI analysis features

- APS provides additional options for MPI analysis
- Advantage : easy to use and very high scaling
- Works best when only focus on MPI or MPI+OMP:

```
--collection-mode=mpi[,omp]
```

- Setting of environment variables may be necessary for more information

```
$ export APS_STAT_LEVEL=N      # N = 1-5
$ export APS_IMBALANCE_TYPE=K # K = 1-3
```

- User's Guide provides details:
  https://www.intel.com/content/www/us/en/docs/vtune-profiler/user-guide-application-snapshot-linux/2025-0/overview.html

APS Command Line Reports – Advanced MPI statistics (1/3)

- MPI Time per rank
  - aps-report –t <result>

MPI Time per Rank

Rank	LifeTime(sec)	MPI Time(sec)	MPI Time(%)	Imbalance(sec)	Imbalance(%)
0007	72.52	14.31	19.74	4.84	6.67
0004	72.53	11.57	15.96	3.26	4.50
0005	72.52	11.40	15.72	3.20	4.42
0006	72.51	11.11	15.32	3.17	4.37
0000	72.49	11.08	15.29	4.33	5.97
0001	72.52	10.95	15.10	3.01	4.15
0002	72.49	10.79	14.88	2.57	3.55
0003	72.50	10.64	14.68	2.50	3.45
TOTAL	580.07	91.86	15.84	26.88	4.63
AVG	72.51	11.48	15.84	3.36	4.63

9

APS Command Line Reports – Advanced MPI statistics (2/3)

- Message Size Summary by all ranks
  - aps-report –m <result>

Message Sizes summary for all ranks

Message size(B)	Volume(MB)	Volume(%)	Transfers	Time(sec)	Time(%)
8	1.49	0.09	195206	27.79	37.93
176	0.41	0.02	2420	27.67	37.78
4	0.00	0.00	1150	15.55	21.22
100264	115.89	6.94	1212	0.27	0.37
98400	113.74	6.81	1212	0.19	0.26
66256	38.29	2.29	606	0.17	0.23
[filtered out 57 lines]					
TOTAL	1670.60	100.00	265160	73.25	100.00

10

APS Command Line Reports – Advanced MPI statistics (3/3)

- Data Transfers for Rank-to-Rank Communication
  - aps-report –x <result>

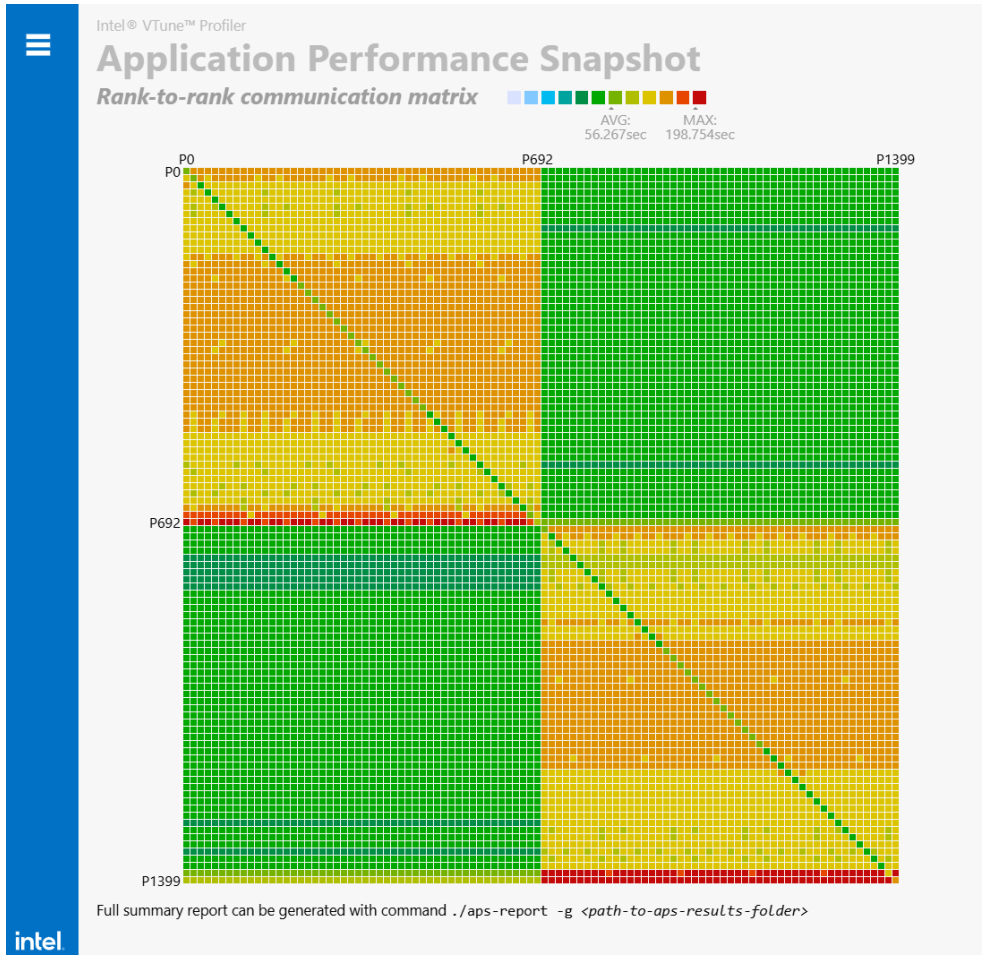
And many others – check

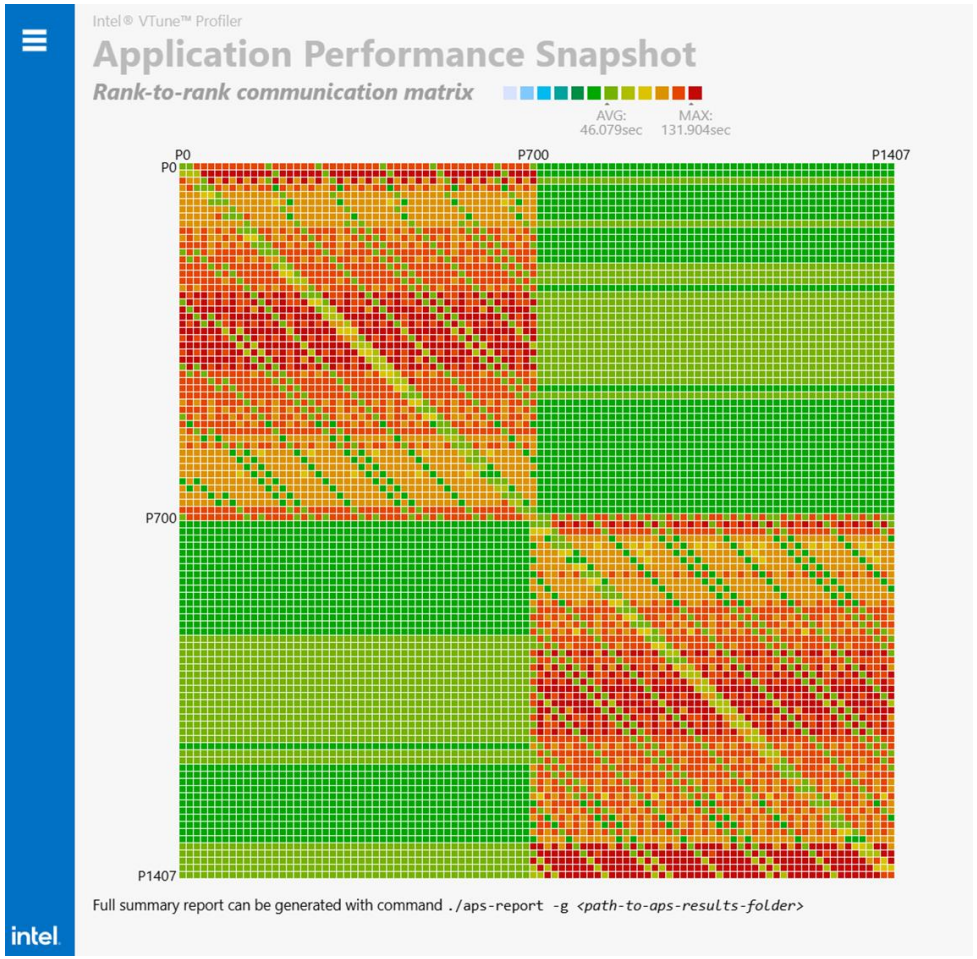
- aps-report -help

Rank --> Rank	Volume(MB)	Volume(%)	Transfers
0023 --> 0024	84.35	1.56	13477
0025 --> 0026	84.35	1.56	13477
0024 --> 0025	84.15	1.56	13477
0021 --> 0022	83.84	1.55	13477
0022 --> 0023	83.43	1.54	13477
[filtered out 16 lines]			
0012 --> 0011	69.60	1.29	13477
0020 --> 0019	69.29	1.28	13477
0026 --> 0025	68.78	1.27	13477
0025 --> 0024	68.38	1.27	13477
0022 --> 0021	68.38	1.27	13477
[filtered out 17 lines]			
0016 --> 0015	58.81	1.09	13477
0028 --> 0027	57.69	1.07	13477
0007 --> 0008	56.98	1.05	13477
0030 --> 0031	54.74	1.01	13477
0006 --> 0007	54.44	1.01	13477
[filtered out 1108 lines]			
TOTAL	5403.22	100.00	1415619
AVG	4.67	0.09	1224

11

Rank to Rank communication can be visualized





APS Tips and Tricks

- Check help menu

```
$ aps –help
```

- Increase level of analysis by setting APS_STAT_LEVEL
- provide date or Job ID in output directory name – avoid overwriting

VTune usage in Cluster Environment
Hotspot analysis

How to use VTune in Cluster (MPI) environment

- I_MPI_GTOOL environment variable offers a convenient way for inserting a tool into a MPI command line. No command line change necessary.

  $ `export I_MPI_GTOOL="vtune -c <analysis> -r <result_name> :<ranks>"`

  <analysis> : analysis type e.g. hotspots
  <result_name> : name of output directory
  <ranks> : ranks that perform VTune analysis e.g. 0

- Running the MPI executable will now generate a result for specified ranks
- Hint: result_name should contain some "counter" like SLURM_JOB_ID.
  If the name is not changed in another experiment VTune will not overwrite!

Hotspots Analysis

Intel VTune Profiler

Elapsed Time: 121.361s

CPU Time: 104.202s

Total Thread Count: 2

Paused Time: 0s

Top Hotspots

This section lists the most active functions in your application.

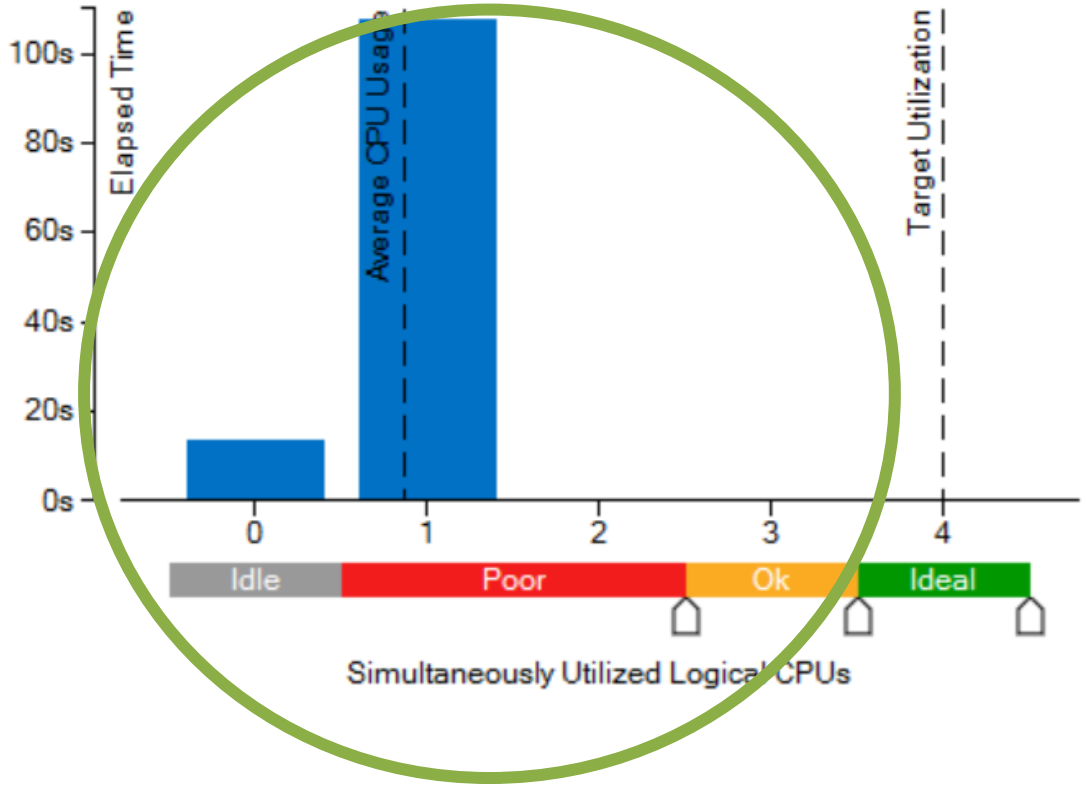
Function	Module	CPU Time
multiply0	matrix.exe	104.068s
free	ucrtbase.dll	0.117s
init_arr	matrix.exe	0.016s
WaitForMultipleObjects	KernelBase.dll	0.001s

Quickly identify hotspots

Visualize CPU usage

CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs CPU usage value.



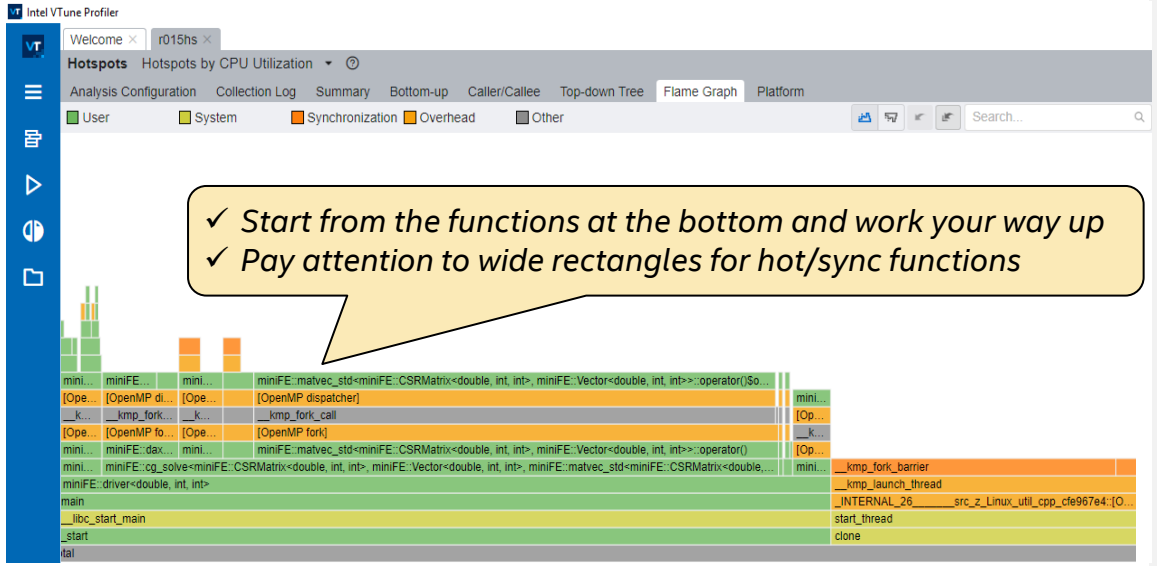
Easily Identify Hot Code Paths
Flame Graphs* for Hotspots Analysis

Visualize Total Function CPU time spent

- Explore stacks and stack frames
  - Aggregation
  - Side-by-side visualization
  - Function bar as fraction of CPU Time
  - Different colors per function type
- Identify the time spent in each function and its callees

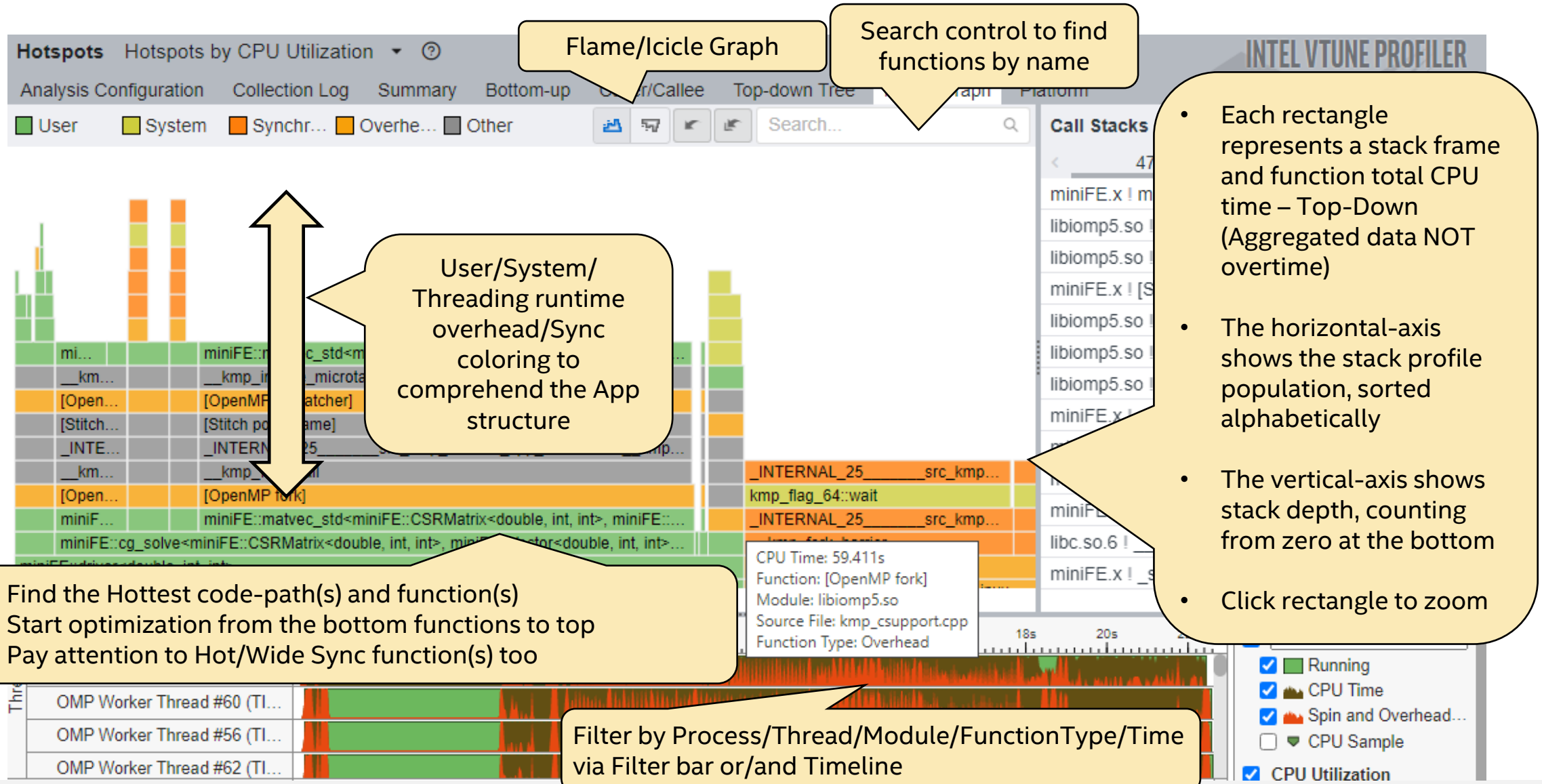
Rich Experience with Intel® VTune™ Profiler UI

- Select your visualization of choice
  - Flame or Icicle Graph
- Filter data by process, thread, time region, and more
- Jump to the function source code via stack pane



*Adapted based on [Brendan Gregg's Flame Graphs]

Flame Graph


Flame/Icicle Graph
Search control to find functions by name
INTEL VTUNE PROFILER
Hotspots
Hotspots by CPU Utilization
Analysis Configuration
Collection Log
Summary
Bottom-up
Top-down Tree
User
System
Synchr...
Overhe...
Other
Search...
Call Stacks
User/System/ Threading runtime overhead/Sync coloring to comprehend the App structure
_INTERNAL_25_____src_kmp...
kmp_flag_64::wait
[OpenMP fork]
miniFE::matvec_std<miniFE::CSRMatrix<double, int, int>, miniFE::...
CPU Time: 59.411s
Function: [OpenMP fork]
Module: libiomp5.so
Source File: kmp_csupport.cpp
Function Type: Overhead
OMP Worker Thread #60 (TI...
OMP Worker Thread #56 (TI...
OMP Worker Thread #62 (TI...
Running
CPU Time
Spin and Overhead...
CPU Sample
CPU Utilization
Each rectangle represents a stack frame and function total CPU time – Top-Down (Aggregated data NOT overtime)
The horizontal-axis shows the stack profile population, sorted alphabetically
The vertical-axis shows stack depth, counting from zero at the bottom
Click rectangle to zoom
Find the Hottest code-path(s) and function(s)
Start optimization from the bottom functions to top
Pay attention to Hot/Wide Sync function(s) too
Filter by Process/Thread/Module/FunctionType/Time via Filter bar or/and Timeline

VTune HPC Performance

Three Keys to HPC Performance:

Threading, Memory Access, Vectorization – Intel VTune™ Profiler

Threading: CPU Utilization

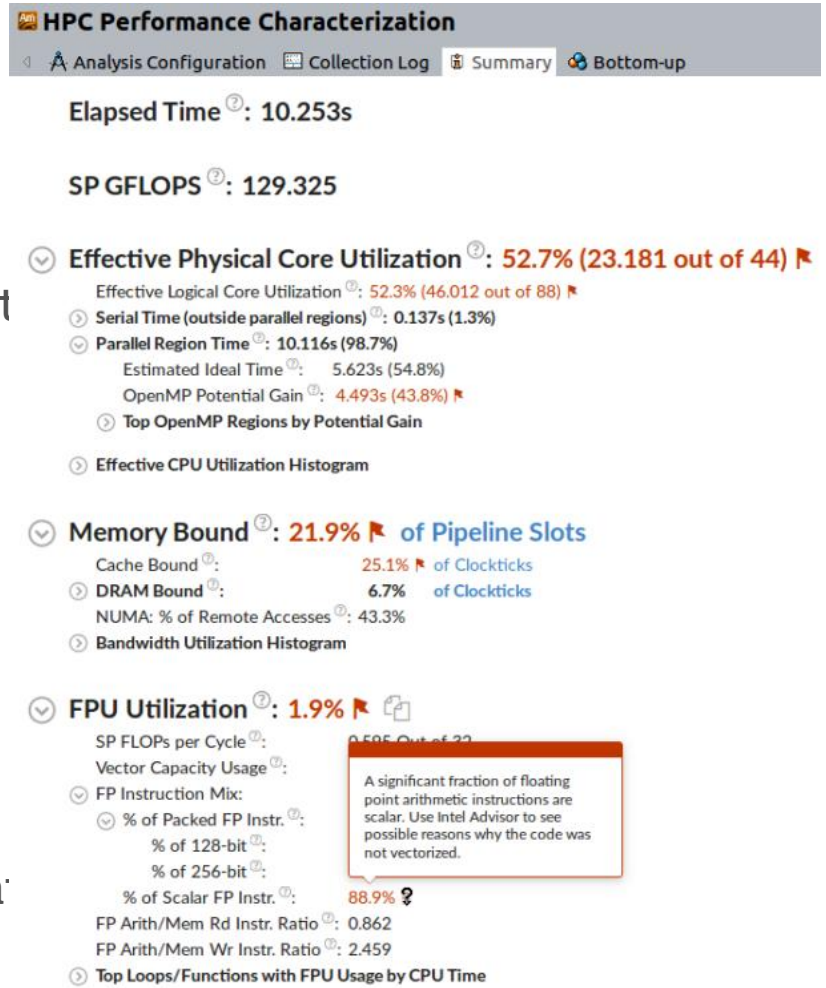
- Serial vs. Parallel time
- Top OpenMP regions by potential gain
- Tip: Use hotspot OpenMP region analysis for more det

Memory Access Efficiency

- Stalls by memory hierarchy
- Bandwidth utilization
- Tip: Use Memory Access analysis

Vectorization: FPU Utilization

- FLOPS † estimates from sampling
- Tip: Use Intel Advisor for precise metrics and vectoriza



OpenMP and HPC command lines

- HPC Analysis:

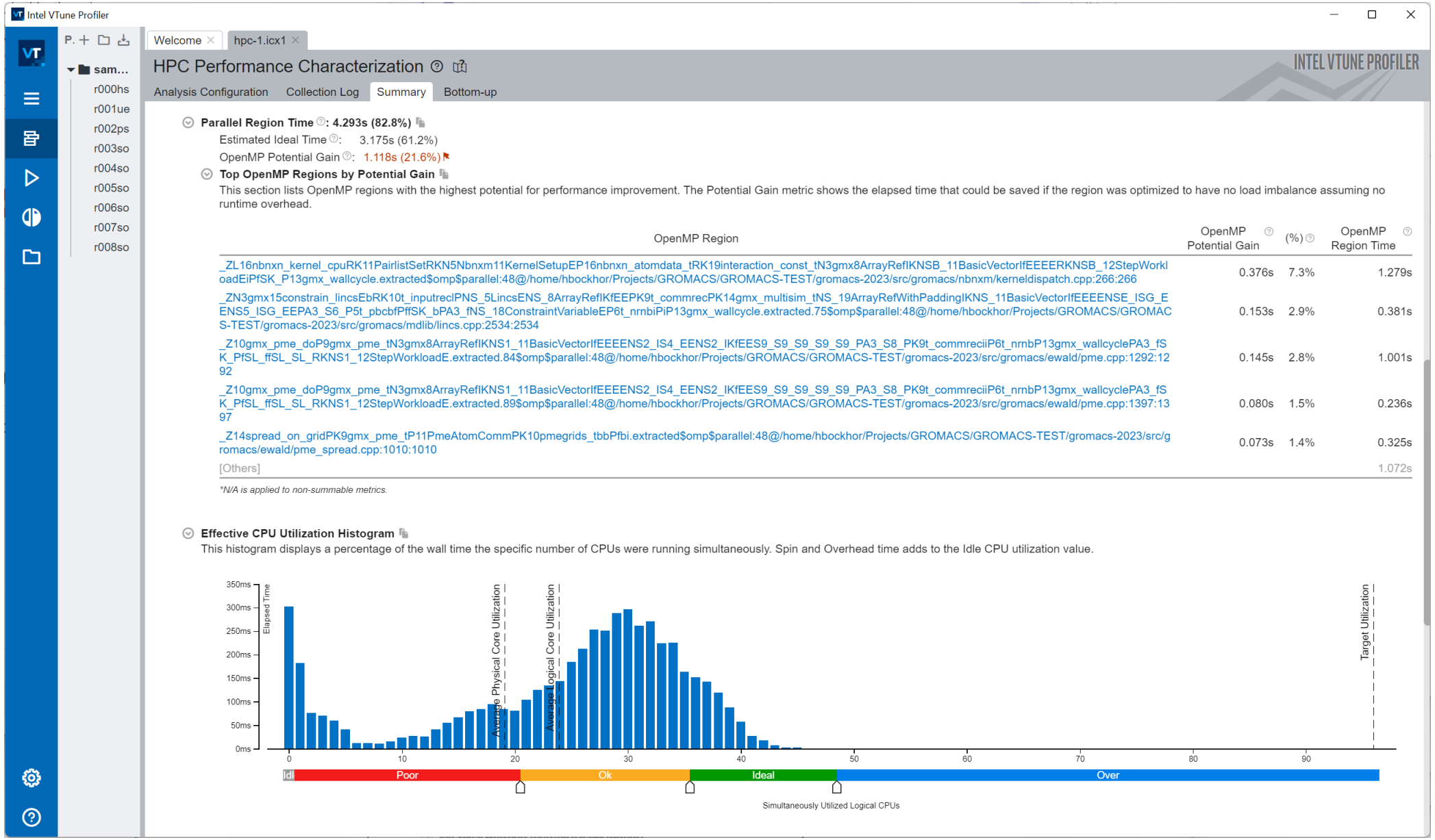
```
$ vtune -collect hpc-performance <your app> <app parameter>
```

- Threading Analysis:

```
$ vtune -collect threading <your app> <app parameter>
```

- Help Menu:

```
$ vtune -help
$ vtune -help collect
$ vtune -help collect threading
```


Intel VTune Profiler
Welcome
hpc-1.icx1
HPC Performance Characterization
INTEL VTUNE PROFILER
Analysis Configuration
Collection Log
Summary
Bottom-up
Parallel Region Time: 4.293s (82.8%)
Estimated Ideal Time: 3.175s (61.2%)
OpenMP Potential Gain: 1.118s (21.6%)
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
OpenMP Potential Gain
(%)
OpenMP Region Time
_ZL16nbnxn_kernel_cpuRK11PairlistSetRKN5Nbnxm11KernelSetupEP16nbnxn_atomdata_tRK19interaction_const_tN3gmx8ArrayRefIKNSB_11BasicVectorIfEEEERKNSB_12StepWorkloadEiPfSK_P13gmx_wallcycle.extractedompparallel:48@/home/hbockhor/Projects/GROMACS/GROMACS-TEST/gromacs-2023/src/gromacs/nbnxm/kerneldispatch.cpp:266
0.376s 7.3% 1.279s
_ZN3gmx15constrain_lincsEbRK10t_inputreclPNS_5LincsENS_8ArrayRefIKfEEPK9t_commrecPK14gmx_multisim_tNS_19ArrayRefWithPaddingIKNS_11BasicVectorIfEEEENSE_ISG_EENS5_ISG_EEPA3_S6_P5t_pbcbfPffSK_bPA3_fNS_18ConstraintVariableEP6t_nrnbiPiP13gmx_wallcycle.extracted.75ompparallel:48@/home/hbockhor/Projects/GROMACS/GROMACS-TEST/gromacs-2023/src/gromacs/mdlib/lincs.cpp:2534:2534
0.153s 2.9% 0.381s
_Z10gmx_pme_doP9gmx_pme_tN3gmx8ArrayRefIKNS1_11BasicVectorIfEEEENS2_IS4_EENS2_IKfEES9_S9_S9_S9_S9_PA3_S8_PK9t_commreciiP6t_nrnbP13gmx_wallcyclePA3_fSK_PfSL_ffSL_SL_RKNS1_12StepWorkloadE.extracted.84ompparallel:48@/home/hbockhor/Projects/GROMACS/GROMACS-TEST/gromacs-2023/src/gromacs/ewald/pme.cpp:1292:1292
0.145s 2.8% 1.001s
_Z10gmx_pme_doP9gmx_pme_tN3gmx8ArrayRefIKNS1_11BasicVectorIfEEEENS2_IS4_EENS2_IKfEES9_S9_S9_S9_S9_PA3_S8_PK9t_commreciiP6t_nrnbP13gmx_wallcyclePA3_fSK_PfSL_ffSL_SL_RKNS1_12StepWorkloadE.extracted.89ompparallel:48@/home/hbockhor/Projects/GROMACS/GROMACS-TEST/gromacs-2023/src/gromacs/ewald/pme.cpp:1397:1397
0.080s 1.5% 0.236s
_Z14spread_on_gridPK9gmx_pme_tP11PmeAtomCommPK10pmegrids_tbbPfbi.extractedompparallel:48@/home/hbockhor/Projects/GROMACS/GROMACS-TEST/gromacs-2023/src/gromacs/ewald/pme_spread.cpp:1010:1010
0.073s 1.4% 0.325s
[Others]
1.072s
*N/A is applied to non-summable metrics.
Effective CPU Utilization Histogram
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.
Elapsed Time
350ms
300ms
250ms
200ms
150ms
100ms
50ms
0ms
Average Physical Core Utilization
Average Logical Core Utilization
Target Utilization
0
10
20
30
40
50
60
70
80
90
Idl
Poor
Ok
Ideal
Over
Simultaneously Utilized Logical CPUs

Intel VTune Profiler

Welcome | hpc-1.icx1

HPC Performance Characterization

INTEL VTUNE PROFILER

Analysis Configuration | Collection Log | Summary | Bottom-up

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

OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack	Elapsed Time ▼	SP GFLOPS	OpenMP Potential Gain					
			Imbalance	Lock Contention	Creation	Scheduling	Reduction	Atomics
_ZL16nbnxn_kernel_cpuRK11PairlistS	1.279s	1350.762	0.376s	0s	0.000s	0s	0s	0s
_Z10gmx_pme_doP9gmx_pme_tN3gr	1.001s	83.340	0.145s	0s	0s	0s	0s	0s
[Serial - outside parallel regions]	0.894s	0.147						
_ZN3gmx15constrain_lincsEbRK10t_i	0.381s	26.404	0.153s	0s	0s	0s	0s	0s
_Z14spread_on_gridPK9gmx_pme_tP	0.325s	3.366	0.072s	0s	0.000s	0s	0s	0s
_Z14spread_on_gridPK9gmx_pme_tP	0.300s	15.980	0.063s	0s	0s	0s	0s	0s
_Z10gmx_pme_doP9gmx_pme_tN3gr	0.236s	194.914	0.080s	0s	0.000s	0s	0s	0s
_ZL16calcBondedForcesRK22Interact	0.122s	55.170	0.032s	0s	0s	0s	0s	0s
_ZL37nbnxn_atomdata_add_nbat_f_t	0.111s	0.000	0.027s	0s	0s	0s	0s	0s
_Z14spread_on_gridPK9gmx_pme_tP	0.089s	0.000	0.026s	0s	0.000s	0.000s	0s	0s
_ZN3gmx11Constraints4Impl5applyEb	0.078s	140.524	0.025s	0s	0s	0.000s	0s	0s
_Z31nbnxn_atomdata_copy_x_to_nba	0.050s	0.000	0.014s	0s	0s	0.000s	0s	0s
_ZN3gmx6Update4Impl13update_coo	0.042s	26.419	0.012s	0s	0.000s	0.000s	0s	0s
_ZN3gmx6Update4Impl13finish_updat	0.041s	0.000	0.010s	0s	0s	0.000s	0s	0s
_ZN3gmx12_GLOBAL__N_124reduce	0.040s	0.000	0.010s	0s	0s	0.000s	0s	0s
_ZN11PairlistSet18constructPairlistsEl	0.040s	48.416	0.008s	0s	0s	0s	0s	0s
_Z23unwrap_periodic_pmegridP9gmx	0.030s	0.000	0.010s	0s	0s	0.000s	0s	0s
_ZN11PairlistSet19dispatchPruneKern	0.028s	698.915	0.009s	0s	0s	0s	0s	0s
_ZL10clearRVecsN3gmx8ArrayRefIN5	0.015s	0.000	0.005s	0s	0.000s	0.000s	0s	0s
_Z25dd_make_local_constraintsP12g	0.014s	0.000	0.012s	0s	0s	0s	0s	0s

Elapsed Time: 5.187s
- SP GFLOPS: 368.912
- DP GFLOPS: 0.013
- x87 GFLOPS: 0.000
- CPI Rate: 1.318
- Average CPU Frequency: 3.1 GHz
- Total Thread Count: 48

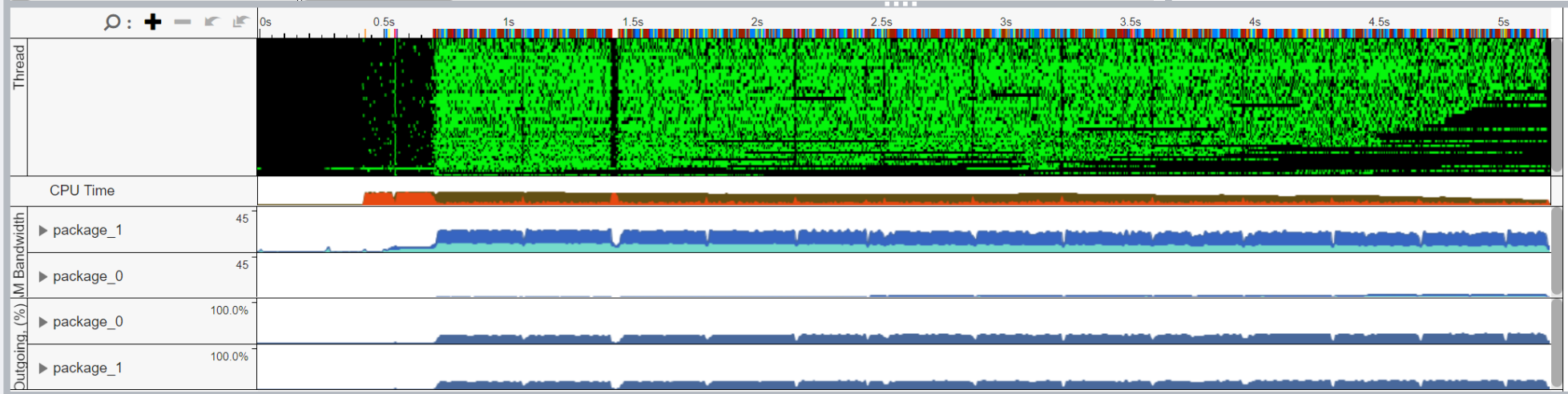
Effective Physical Core Utilization: 39.8% (19.118 out of 48)
- Effective Logical Core Utilization: 24.9% (23.928 out of 96)
- **Serial Time (outside parallel regions): 0.894s (17.2%)**
- **Parallel Region Time: 4.293s (82.8%)**
- Estimated Ideal Time: 3.175s (61.2%)
- OpenMP Potential Gain: 1.118s (21.6%)

Effective CPU Utilization Histogram

Memory Bound: 28.3% of Pipeline Slots
- Cache Bound: 23.0% of Clockticks
- **DRAM Bound: 7.4% of Clockticks**
- DRAM Bandwidth Bound: 0.0% of Elapsed Time
- NUMA: % of Remote Accesses: 19.0%

Bandwidth Utilization Histogram

Vectorization: 99.1% of Packed FP Operations
- Instruction Mix:



Scale Markers:
- Region Instance
- OpenMP Barrier-to-Barrier Segment
- Thread
  - Effective Time
  - Spin and Overhead...
- **CPU Time**
  - CPU Time
  - Spin and Overhead...
- **DRAM Bandwidth**
  - Average Bandwidth, ...
    - Read
    - Write
  - Total, GB/sec

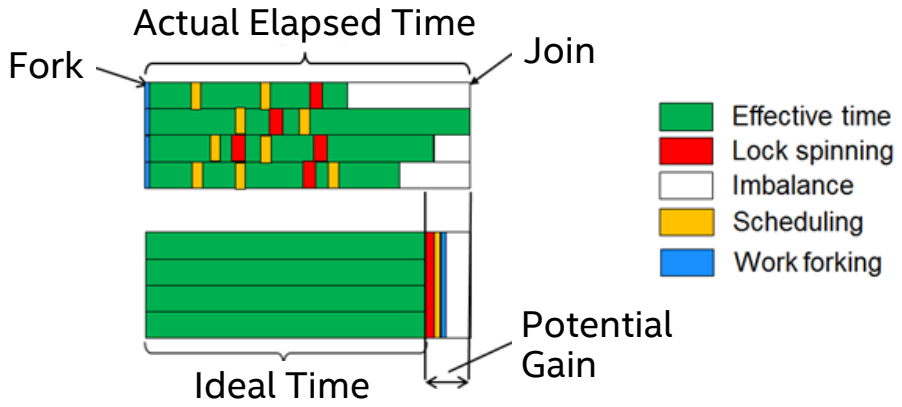
FILTER 100.0% | Process: Any Process | Module: Any Module | Call Stack Mode: User functions + 1 | Loop Mode: Loops and functions | Inline Mode: Show inline functions

Tune OpenMP for Efficiency and Scalability

See the wall clock impact of inefficiencies, identify their cause

Focus On What's Important

- What region is inefficient?
- Is the potential gain worth it?
- Why is it inefficient?
  Imbalance? Scheduling? Lock spinning?



Imbalance Lock Fork Scheduling Potential Gain

Advanced Hotspots Intel VTune Amplifier

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

Grouping: OpenMP Region / Function / Call Stack

OpenMP Region / Function / Call Stack	OpenMP Potential Gain: Imbalance	Lock Contention	Creation	Scheduling	Reduction	Other	OpenMP Potential Gain (% of Colle...)	Elapsed Time	Number of OpenMP threads	Instance Count	CPU Time: Effective Time by Utilization (Idle Poor Ok Ideal Over)	Spin Time	Overhead Time
conj_grad_ompparallel:24@	3.944s	0s	0.000s	0.002s	0.000s	0.010s	34.7%	11.095s	24	76	172.963s	92.219s	0.084s
MAIN__ompparallel:24@/h	0.086s	0s	0s	0s	0s	0.000s	0.8%	0.286s	24	1	4.819s	2.006s	0s
[Serial - outside any region]						0s	0.0%	0.012s			0.045s	0.091s	0.003s
MAIN__ompparallel:24@/h	0.000s	0s	0s	0s	0s	0s	0.0%	0.001s	24	75	0.004s	0.016s	0s
Selected 1 row(s):								11.095s		76	172.963s	92.219s	0.084s

Tune Tips and Tricks

- Analysis can be done on all MPI ranks, but overhead is growing and numbers of some HW counters are limited
  → start analysis on a single rank
- Experiment with VTune features like Grouping and Filtering. New views may reveal issues easier
- Run analysis with command line and analyze by using the VTune GUI
- VTune ascii playbook (see labs) offers sample command lines

More Resources

- Intel® VTune™ Profiler – Performance Profiler
- Product page – overview, features, FAQs
- Cookbooks, User Guide, Processor Tuning Guides – Training materials
- Support Forum - External
- Online Service Center - Secure Priority Support
- What's New?
- Newsletter

Summary

- APS offers and easy way of generating an application overview
- VTune MPI cluster usage
- VTune Hotspots
- VTune HPC Performance including OMP analysis
- Many tutorials + YouTube videos available

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more on the Performance Index site.

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

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

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

All product plans and roadmaps are subject to change without notice.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

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


intel