



# VTune for oneAPI

EuroCC Intel tools, February 16<sup>th</sup> 2022

Dr. Heinrich Bockhorst – Intel

# Agenda

- VTune™ overview
- Command lines – Playbook
- Start profiling with Application Performance Snapshot (APS)
- HPC analysis - short overview
- GPU analysis – example GROMACS
- Summary

# Optimize Performance

Intel® VTune™ Profiler

## Get the Right Data to Find Bottlenecks

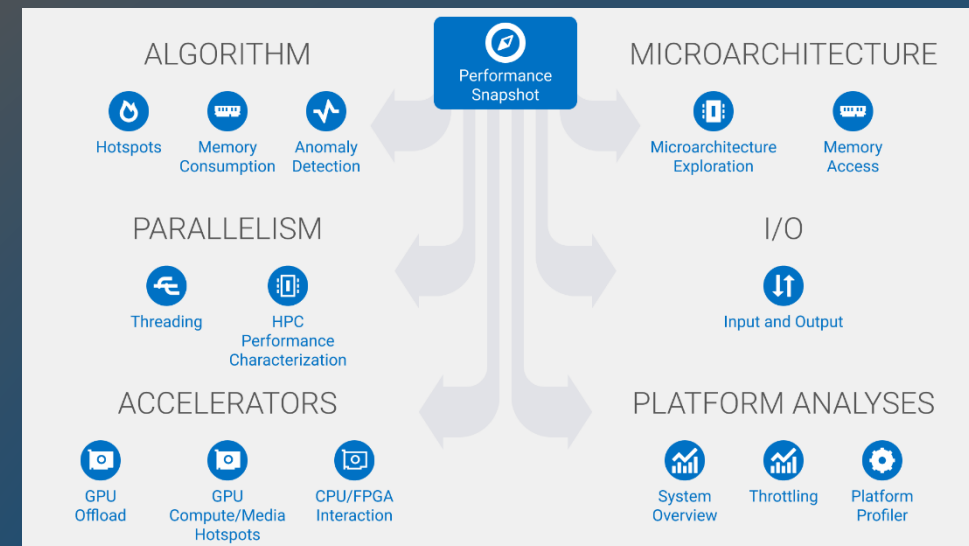
- A suite of profiling for CPU, GPU, FPGA, threading, memory, cache, storage, offload, power...
- DPC++, C, C++, Fortran, Python\*, Go\*, Java\*, or a mix
- Linux, Windows, FreeBSD, Android, Yocto and more

## Analyze Data Faster

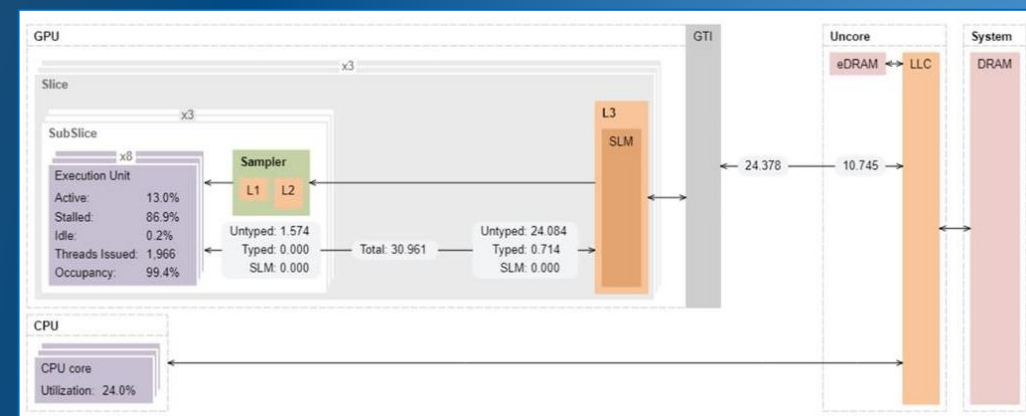
- See data on your source, in architecture diagrams, as a histogram, on a timeline...
- Filter and organize data to find answers

## Work Your Way

- User interface or command line
- Profile locally and remotely
- Install as an application
- Install as a server accessible with a web browser



Source		Assembly		GPU Instructions Executed by Instruction T...	
...	▲	Source		Control Flow	Send & Wait
158	dx = ptr[j].pos[0] - ptr[i].pos[0]			Int32 & SP Float	Int64 & DP Float
159	dy = ptr[j].pos[1] - ptr[i].pos[1]				Other
160	dz = ptr[j].pos[2] - ptr[i].pos[2]				



# Rich Set of Profiling Capabilities for Multiple Markets

Intel® VTune™ Profiler



## Single Thread

Optimize single-threaded performance.



## Multithreaded

Effectively use all available cores.



## System

See a system-level view of application performance.



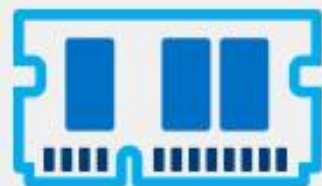
## Media & OpenCL™ Applications

Deliver high-performance image and video processing pipelines.



## HPC & Cloud

Access specialized, in-depth analyses for HPC and cloud computing.



## Memory & Storage Management

Diagnose memory, storage, and data plane bottlenecks.



## Analyze & Filter Data

Mine data for answers.



## Environment

Fits your environment and workflow.

# 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

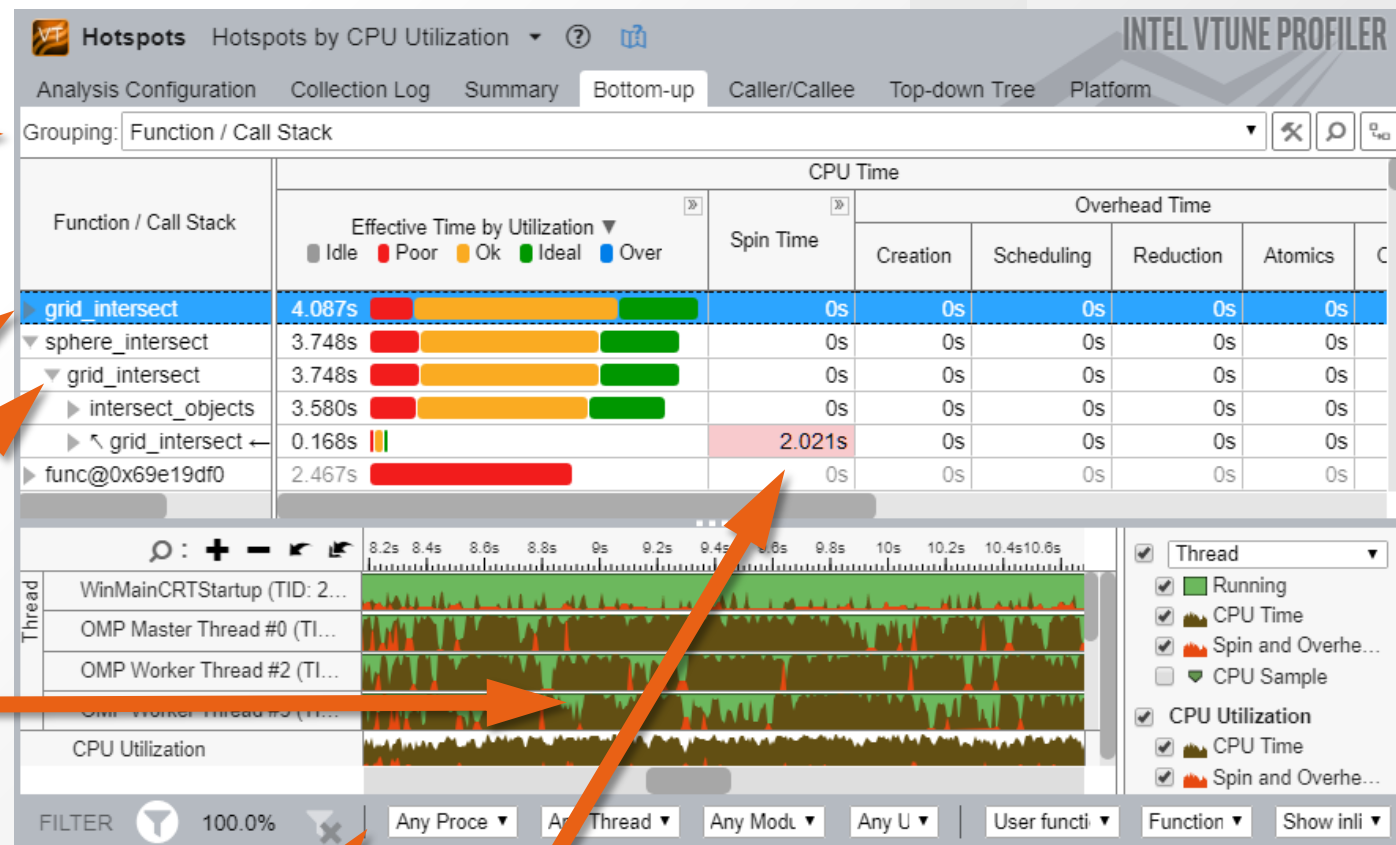
## Click [▶] for Call Stack

## 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



# VTune Playbook – for this session

- Easy access to command lines
- Example for DevCloud – please try

```
~/INCOMING/22.02.15-EuroCC
Playbook for using VTune tool on devcloud or other clusters

Note: command lines start with "$" prompt.

1. Log into DevCloud
-----
$ ssh devcloud
Alternative: open a jupyter notebook and start the terminal

2. Clone Samples GitHub
-----
$ git clone https://github.com/oneapi-src/oneAPI-samples.git

3. Start interactive session on a node with DGL Xe GPU (iris_xe_max)
-----
(People with NDA accounts may use ATS-P gpu)

it is better to compile on compute node because login node has very limited memory etc.
$ qsub -I -l nodes=1:iris_xe_max:ppn=2

4. Check properties
-----
$ sycl-ls --verbose

prints out all backends (GPU device + low level driver level_zero or opencl)
Level Zero shows:
Platform [#4]:
  Version : 1.2
  Name    : Intel(R) Level-Zero
  Vendor  : Intel(R) Corporation
  Devices : 1
    Device [#0]:
```

# How to start an Analysis

- VTune offers different analysis types with additional “knobs”
- Application Performance Snapshot (VTune or standalone)
- APS does first analysis and guides to the right VTune analysis or a different tool available in the oneAPI toolkits.
- APS analyses HW counters, OpenMP and MPI
- APS can be used for MPI and/or OpenMP only to avoid too much data being collected

# APS usage

- Help menu

```
$ aps -help
```

- Run with APS

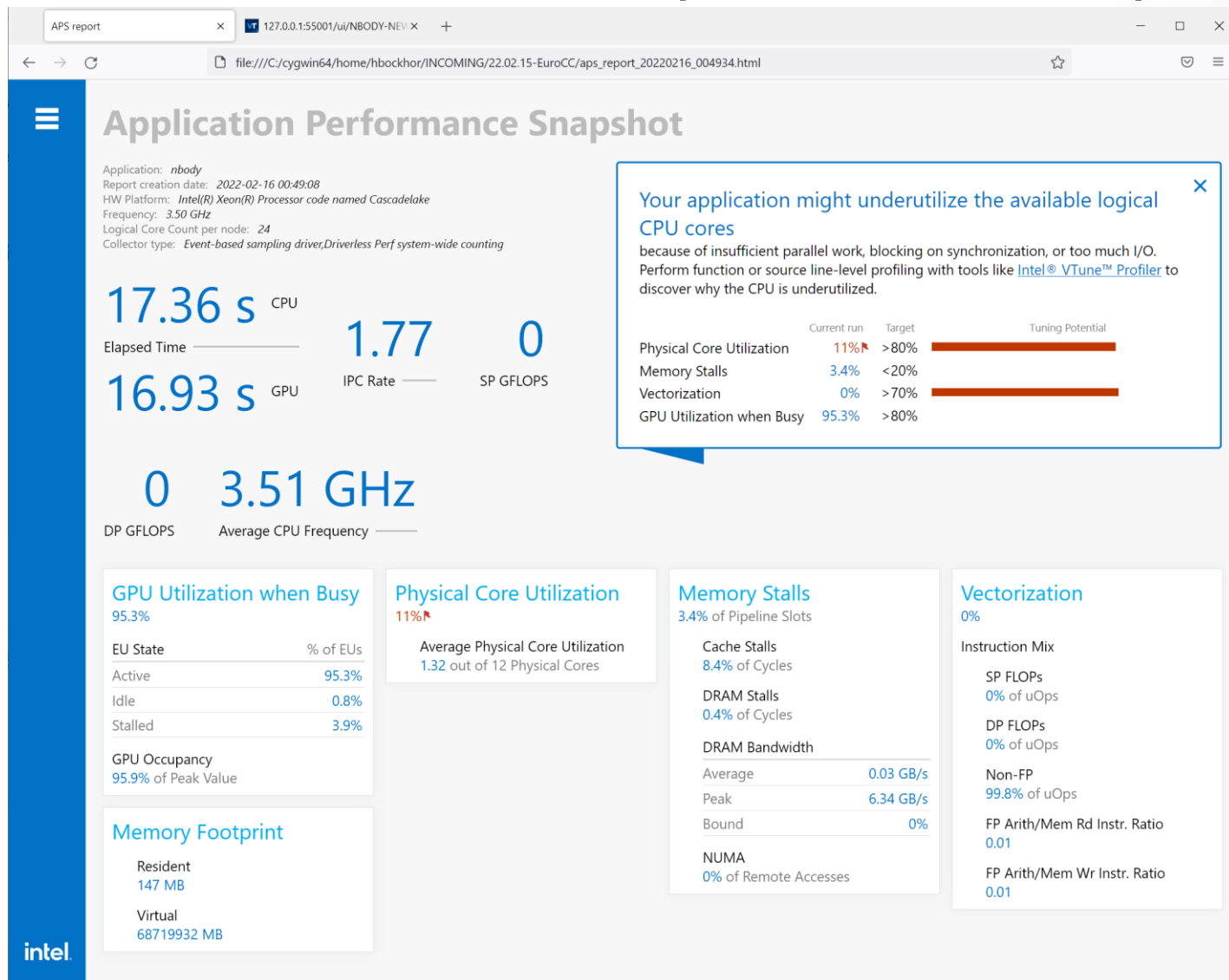
```
$ aps <application> <app paramter>
```

- Run with MPI (Intel MPI or compatible)

```
$ mpirun -n <N> aps -- <application> < app parameter>
```



# APS HTML output (nbody)



# APS MPI + OpenMP analysis

- HW drivers can be switched off
- Max. time consuming MPI functions shown
- Imbalance time shown for OpenMP + MPI (Waiting in barriers)
- MPI session shows more

# VTune HPC Analysis

- Detailed OpenMP analysis
- Memory/Bandwidth analysis
- Some MPI analysis – better use ITAC

# 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
```

# Tune OpenMP for Efficiency and Scalability

1)▶

2)▶

3)▶

4)▶

⌵

**OpenMP Analysis. Collection Time<sup>?</sup>: 11.400**

⌵

Serial Time (outside any parallel region)<sup>?</sup>: 0.017s (0.1%)

⌵

Parallel Region Time<sup>?</sup>: 11.384s (99.9%)

Estimated Ideal Time<sup>?</sup>: 7.351s (64.5%)

OpenMP Potential Gain<sup>?</sup>: 4.033s (35.4%) 🚩

⌵

**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 <sup>?</sup>	(%) <sup>?</sup>	OpenMP Region Time <sup>?</sup>
<a href="#">conj_grad_\$omp\$parallel:24@/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:514:695</a>	3.946s 🚩	34.6% 🚩	11.095s
<a href="#">MAIN_\$omp\$parallel:24@/NPB/NPB3.3.1/NPB3.3-OMP/CG/cg.f:185:231</a>	0.086s	0.8%	0.286s

The summary view shown above gives fast answers to four important OpenMP tuning questions:

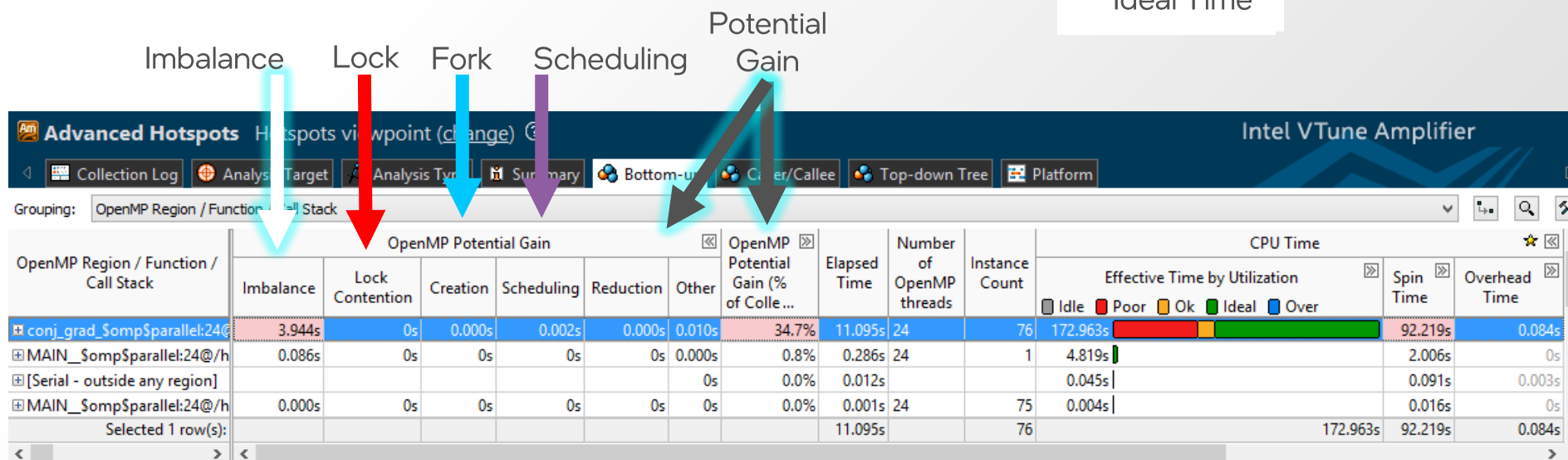
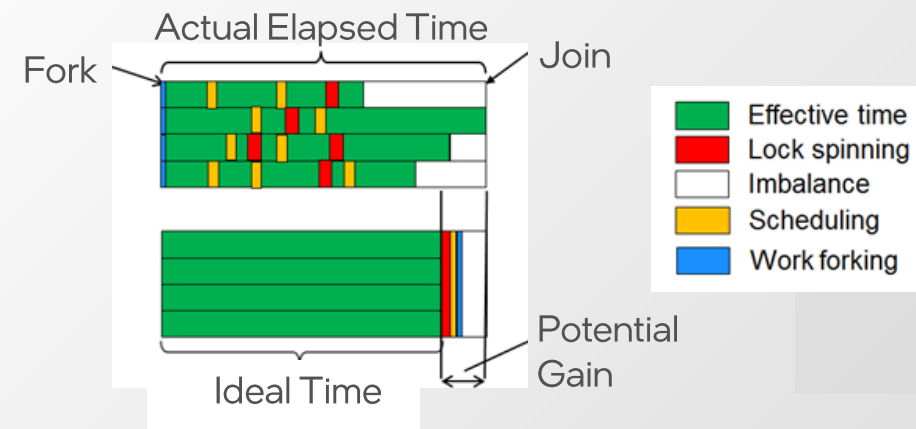
- 1) Is the serial time of my application significant enough to prevent scaling?
- 2) How much performance can be gained by tuning OpenMP?
- 3) Which OpenMP regions / loops / barriers will benefit most from tuning?
- 4) What are the inefficiencies with each region? (click the link to see details)

# 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?





# Tune OpenMP for Efficiency and Scalability

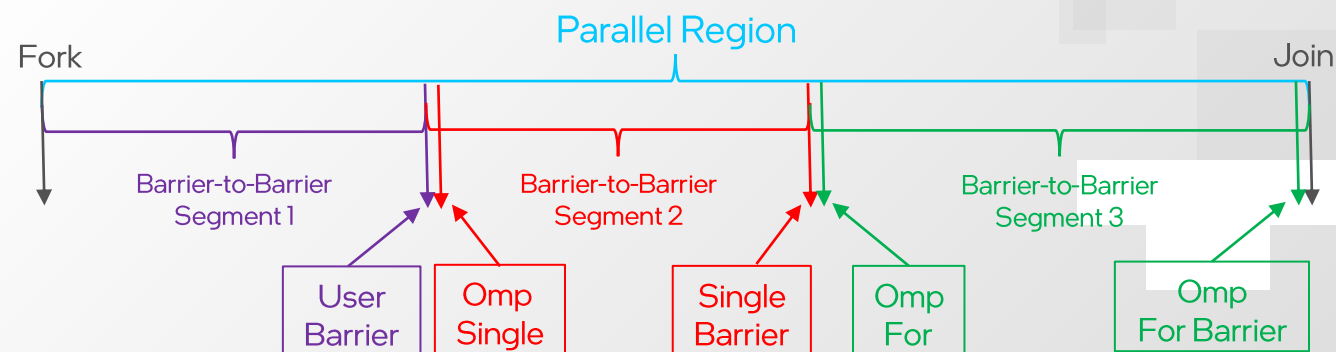
See inside each parallel region – Understand the cause of inefficiency

- Detailed Barrier to Barrier Analysis
  - Tune each segment separately
  - Easier to see tuning opportunities

```
#pragma omp parallel
{
    ... '// Segment 1
#pragma omp barrier

#pragma omp single
{
    ... // Segment 2
}

#pragma omp for
{
    ... // Segment 3
}
```



Grouping: OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack							
OpenMP Region / OpenMP Barrier-to-Barrier Segment / Function / Call Stack	OpenMP Potential Gain						OpenMP Potential Gain (% of Collection ...)
	Imbalance	Lock Contention	Creation	Scheduling	Reduction	Other	
3.3-OMP/CG/cg.f:514:695	3.944s	0s	0.000s	0.002s	0.000s	0.010s	34.7%
NPB3.3.1/NPB3.3-OMP/CG/cg.f:580	3.725s	0s	0s	0.000s	0s	0.008s	32.8%
NPB3.3.1/NPB3.3-OMP/CG/cg.f:683	0.149s	0s	0s	0s	0s	0.000s	1.3%
NPB3.3.1/NPB3.3-OMP/CG/cg.f:664	0.014s	0s	0s	0.000s	0s	0.000s	0.1%

# VTune HPC Analysis

- Additional analysis for Memory and Bandwidth available
- NUMA analysis – check if threads are pinned
- Some MPI analysis is also possible – check cookbooks
- More CPU than GPU related

# 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>

Intel® VTune™ Profiler

# GROMACS on Devcloud

# VTune GPU analysis

- Host API and GPU kernels
- Detailed source view on Kernels
- Bandwidth data with Hierarchical Memory Diagram
- Estimate for timing per source line (basic block timing)
- Estimate for memory latency per source line

# SYCL version of GROMACS

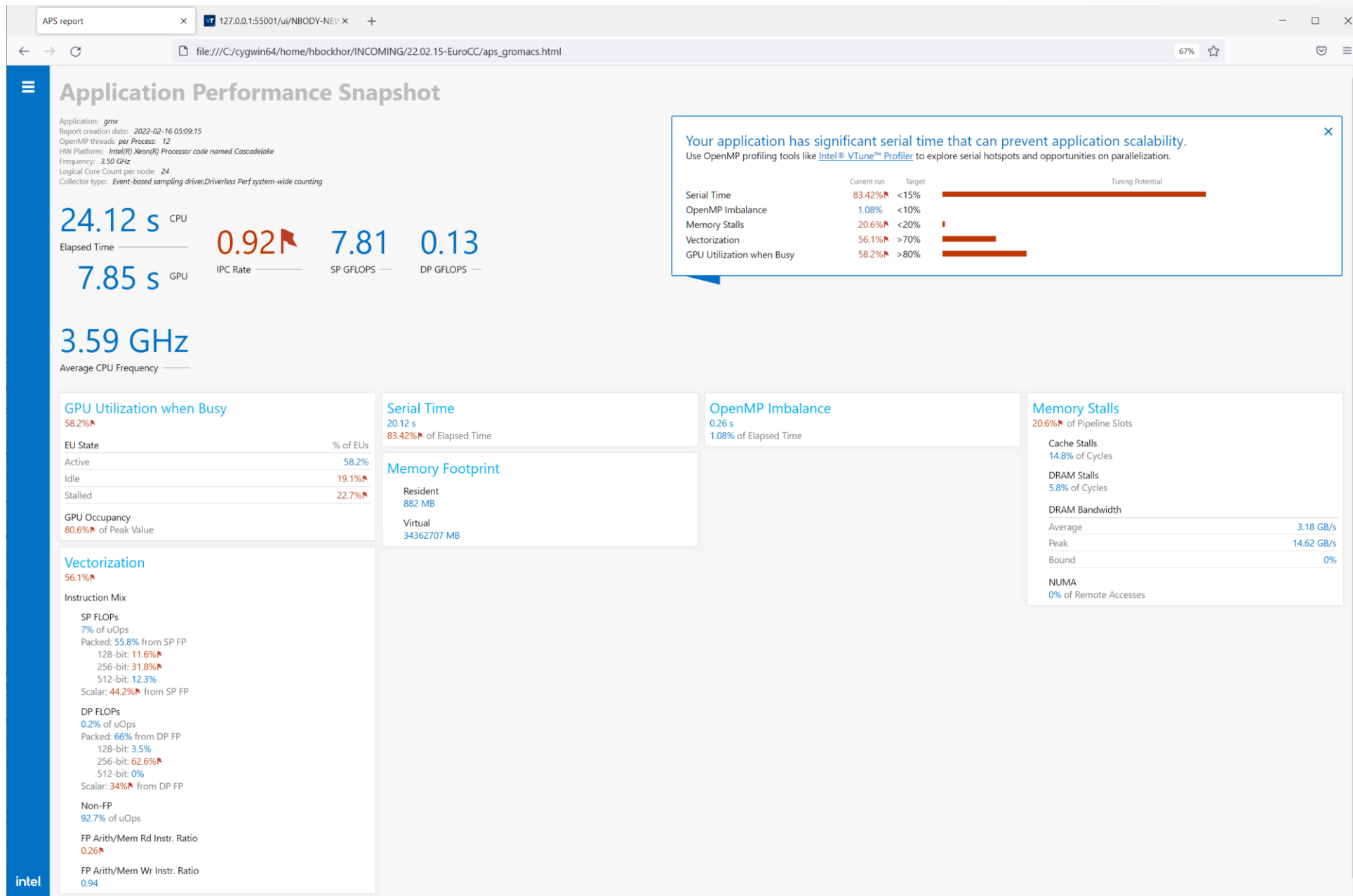
- Clone master branch:  
\$ git clone <https://gitlab.com/gromacs/gromacs.git>
- Workload: <https://www.mpinat.mpg.de/632209/benchMEM.zip>

```
SRC=$HOME/gromacs  
PRE=$HOME/GROMACS/
```

```
cmake $SRC \\\n  -DCMAKE_INSTALL_PREFIX=$PRE \\\n  -DGMX_OPENMP=ON \\\n  -DGMX_OPENMP_MAX_THREADS=128 \\\n  -DGMX_GPU=SYCL \\\n  -DGMX_FFT_LIBRARY=mk1 \\\n  -DCMAKE_EXE_LINKER_FLAGS= \\\n  -DCMAKE_C_COMPILER=icx \\\n  -DCMAKE_CXX_COMPILER=icpx
```

- Run:  
\${EXE\_DIR}/gmxdmrun -ntmpi \$RANKS -ntomp \$OMP\_NUM\_THREADS -nb \$MODE -pme cpu -s benchMEM.tpr -nsteps \$steps





# Command lines used

- Basic gpu analysis:

```
vtune -collect gpu-hotspots -r <your-result-dir> -- <executable><args>
```

- Full instrumentation:

```
vtune -c gpu-hotspots -knob characterization-mode=instruction-count \  
-r <your-result-dir> -- <executable><args>
```

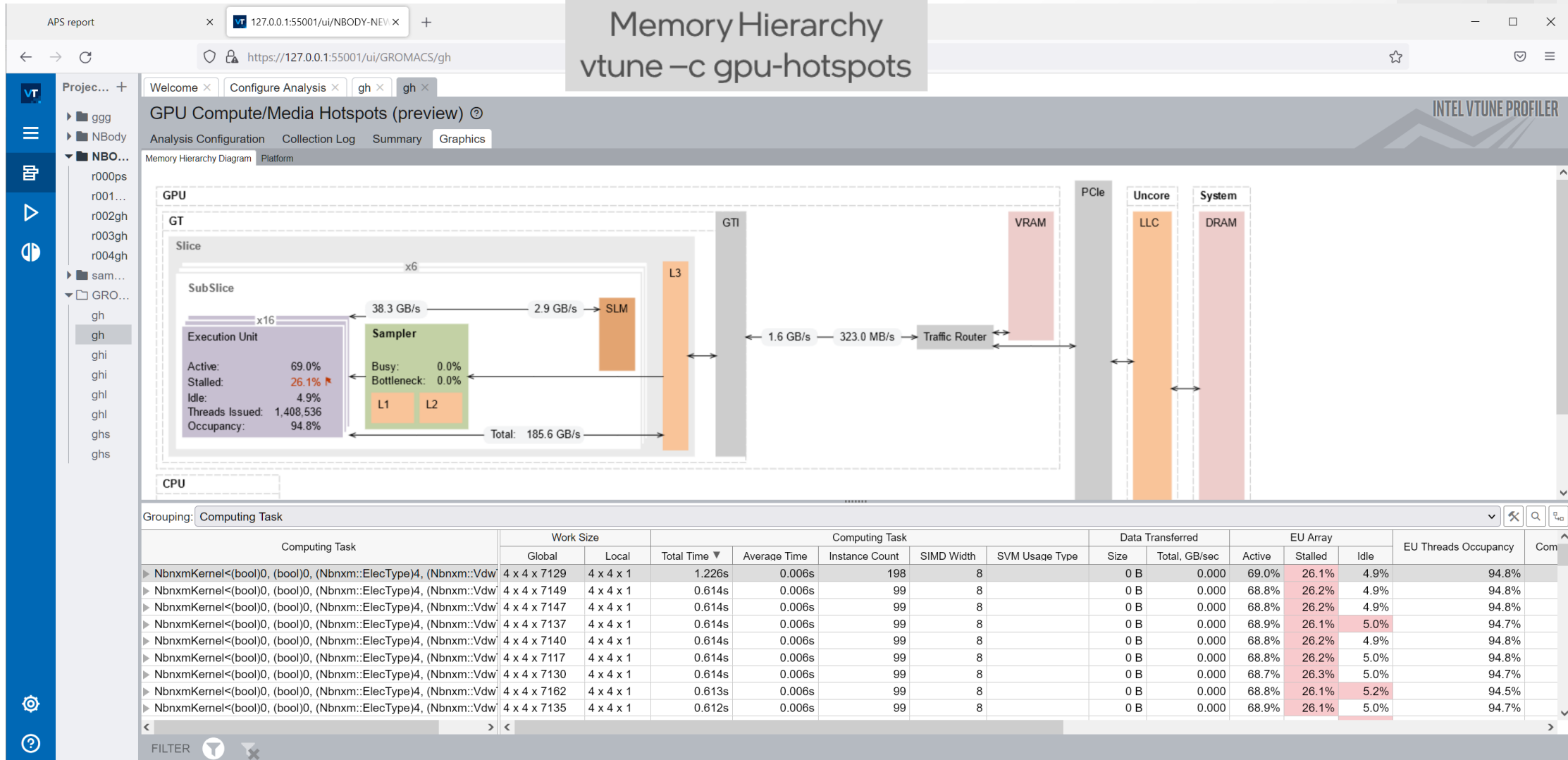
- Source Instrumentation with timing of basic blocks:

```
vtune -c gpu-hotspots -knob profiling-mode=source-analysis\  
-r <your-result-dir> -- <executable><args>
```

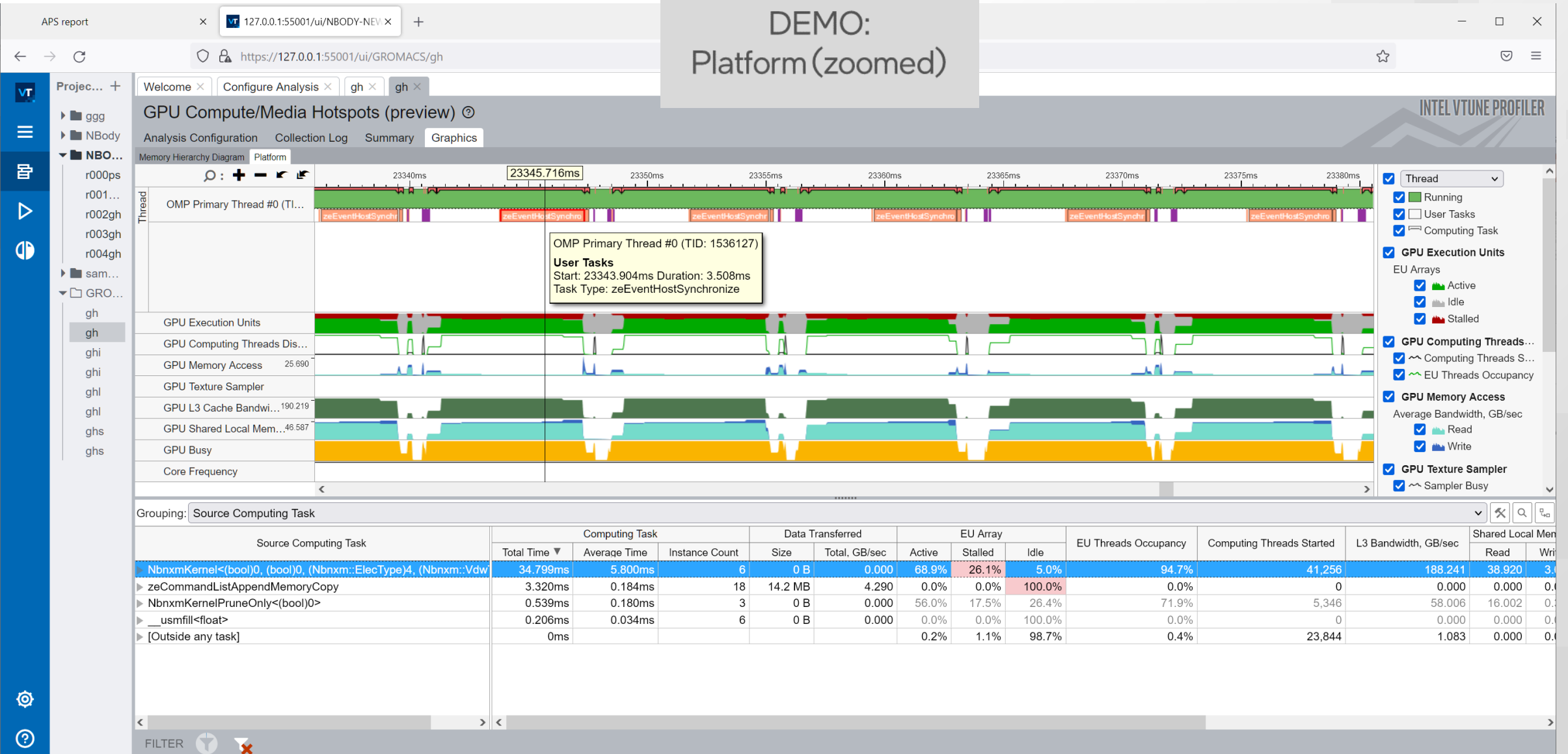
- Source Instrumentation with only memory inst. timed:

```
vtune -c gpu-hotspots -knob profiling-mode=source-analysis \  
-knob source-analysis=mem-latency -r ...
```

# DEMO: Memory Hierarchy vtune -c gpu-hotspots



DEMO:  
Platform (zoomed)



APS report

127.0.0.1:55001/ui/NBODY-NEW

← → ↺

https://127.0.0.1:55001/ui/GROMACS/ghi

☆

🔒

☰

Projec... +

ggg

NBody

NBO...

r000ps

r001...

r002gh

r003gh

r004gh

sam...

GRO...

gh

gh

ghi

ghl

ghs

ghs

Welcome ×

Configure Analysis ×

gh ×

gh ×

ghi ×

GPU Compute/Media Hotspots (preview) ⓘ

Analysis Configuration

Collection Log

Summary

Graphics

nbnxm\_sycl\_kernel.cpp ×

Source

Assembly

⏸

⏮

⏭

⏴

⏵

Source Line ▲	Source	
847	// cutoff & exclusion check	
848		
849	const bool notExcluded = doExclusionForces ? (nonSelfInteraction   (ci != cj))	
850	: (wexcl & maskJI);	
851		
852	// SYCL-TODO: Check optimal way of branching here.	
853	if ((r2 < rCoulombSq) && notExcluded)	15,247,802,964
854	{	
855	const float qi = xqi[3];	
856	int atomTypeI; // Only needed if (!props.vdwComb)	
857	float sigma, epsilon;	
858	Float2 c6c12;	
859		
860	if constexpr (!props.vdwComb)	
861	{	
862	/* LJ 6*C6 and 12*C12 */	
863	atomTypeI = sm_atomTypeI[i * c_clSize + tidxi];	3,158,901,217
864	c6c12 = a_nbfp[numTypes * atomTypeI + atomTypeJ];	89,057,962,872
865	}	
866	else	
867	{	
868	const Float2 ljCombI = sm_ljCombI[i * c_clSize + tidxi];	
869	if constexpr (props.vdwCombGeom)	
870	{	
871	c6c12 = Float2(ljCombI[0] * ljCombJ[0], ljCombI[1] * ljCombJ[1]);	
872	}	
873	else	
874	{	
875	static_assert(props.vdwCombLB);	
876	// LJ 20*(1/6)*sigma and 12*epsilon	

🔥 GPU Instructions Executed by Instruction Type

Control Fl...

Se...

Synchronizat...

Int16 & HP Fl...

Int32 & SP F

DEMO:  
 characterization-mode  
 =instruction-count

DEMO:  
profiling-mode=source-  
analysis

APS report 127.0.0.1:55001/ui/NBODY-NEV x +

https://127.0.0.1:55001/ui/GROMACS/ghs

Project... +

- ggg
- NBody
- NBO...
- r000ps
- r001...
- r002gh
- r003gh
- r004gh
- sam...
- GRO...
- gh
- gh
- ghi
- ghi
- ghl
- ghl
- ghs
- ghs

Welcome x Configure Analysis x gh x gh x ghi x ghs x

GPU Compute/Media Hotspots (preview) 2

Analysis Configuration Collection Log Summary Graphics nbxnm\_sycl\_kernel.cpp x

Source Assembly

Source Line ▲	Source	Estimated GPU Cycles
852	// SYCL-TODO: Check optimal way of branching here.	
853	if ((r2 < rCoulombSq) && notExcluded)	1.1%
854	{	
855	const float qi = xqi[3];	
856	int atomTypeI; // Only needed if (!props.vdwComb)	
857	float sigma, epsilon;	
858	Float2 c6c12;	
859		
860	if constexpr (!props.vdwComb)	
861	{	
862	/* LJ 6*C6 and 12*C12 */	
863	atomTypeI = sm.atomTypeI[i * c.clSize + tidxi];	1.4%
864	c6c12 = a.nbf[numTypes * atomTypeI + atomTypeJ];	17.3%
865	}	
866	else	
867	{	
868	const Float2 ljCombI = sm.ljCombI[i * c.clSize + tidxi];	
869	if constexpr (props.vdwCombGeom)	
870	{	
871	c6c12 = Float2(ljCombI[0] * ljCombJ[0], ljCombI[1] * ljCombJ[1]);	
872	}	
873	else	
874	{	
875	static_assert(props.vdwCombLB);	
876	// LJ 2^(1/6)*sigma and 12*epsilon	
877	sigma = ljCombI[0] + ljCombJ[0];	
878	epsilon = ljCombI[1] * ljCombJ[1];	
879	if constexpr (doCalcEnergies)	
880	{	
881	c6c12 = convertSigmaEpsilonToC6C12(sigma, epsilon);	
882	}	

INTEL VTUNE PROFILER



# DEMO: source-analysis=mem- latency

APS report x VT 127.0.0.1:55001/ui/NBODY-NEV x +

https://127.0.0.1:55001/ui/GROMACS/ghl

GPU Compute/Media Hotspots (preview) ?

Analysis Configuration Collection Log Summary Graphics nbnxm\_sycl\_kernel.cpp x

Source Assembly

Source Line ▲	Source	🔥 Average Latency, Cycles	Estimated GPU Cycles
848			
849	const bool notExcluded = doExclusionForces ? (nonSelfInteraction   (ci != cj))		
850	: (wexcl & maskJI);		
851			
852	// SYCL-TODO: Check optimal way of branching here.		
853	if ((r2 < rCoulombSq) && notExcluded)		
854	{		
855	const float qi = xqi[3];		
856	int atomTypeI; // Only needed if (!props.vdwComb)		
857	float sigma, epsilon;		
858	Float2 c6c12;		
859			
860	if constexpr (!props.vdwComb)		
861	{		
862	/* LJ 6*C6 and 12*C12 */		
863	atomTypeI = sm_atomTypeI[i * c_clSize + tidxi];	58 7.1%	
864	c6c12 = a_nbf[b[numTypes * atomTypeI + atomTypeJ];	185 22.6%	
865	}		
866	else		
867	{		
868	const Float2 ljCombI = sm_ljCombI[i * c_clSize + tidxi];		
869	if constexpr (props.vdwCombGeom)		
870	{		
871	c6c12 = Float2(ljCombI[0] * ljCombJ[0], ljCombI[1] * ljCombJ[1]);		
872	}		

# Documentation

VTune cookbooks:

<https://www.intel.com/content/www/us/en/develop/documentation/vtune-cookbook/top.html>

Vtune on DevCloud:

<https://www.intel.com/content/www/us/en/develop/documentation/vtune-cookbook/top/configuration-recipes/using-vtune-server-with-vs-code-intel-devcloud.html>

GPU profiling:

<https://www.intel.com/content/www/us/en/develop/documentation/vtune-help/top/analyze-performance/accelerators-group/gpu-compute-media-hotspots-analysis.html>

# Notices & Disclaimers

Performance varies by use, configuration, and other factors. Learn more at [www.Intel.com/PerformanceIndex](https://www.intel.com/PerformanceIndex).

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.

Your costs and results may vary.

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

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.



intel<sup>®</sup>