

Performance Tools



NERSC New User Training
September 28, 2022

Neil Mehta
Application Performance Group

Performance tools available on Perlmutter

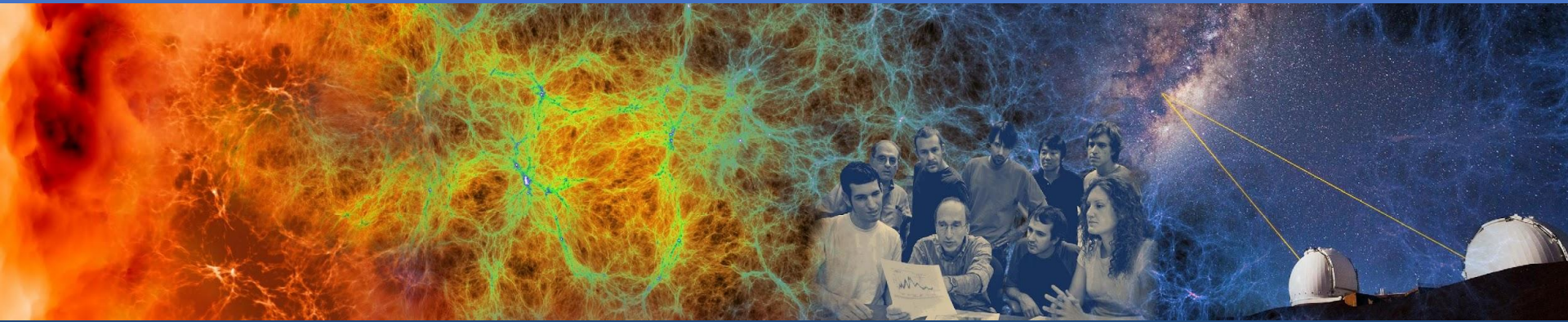
- Profiling tools are used to measure and analyze code performance
- Metrics available help identify memory and compute bottlenecks
- List of profiling tools available on Perlmutter

CrayPat Perftools, -lite, Reveal	CPU, GPU, MPI	cc, CC, ftn, omp
NVIDIA Nsight Systems, Compute	GPU	nvc++, nvcc, omp, oacc, python
ARM MAP, Performance Reports	CPU and GPU	Diverse
Darshan I/O profiler	Data I/O	Diverse
Timemory	Profiling kit	Diverse

A lot more! Full list at <https://docs.nersc.gov/tools/performance/>



CrayPAT



CrayPAT (Perftools)

- **CrayPat specifically for use on Cray machines**
- **Mostly text based**
- **Modules**
 - **perftools-base**
 - **required to load perftools or perftools-lite**
 - **perftools (full-suite)**
 - **perftools-lite (easy to use)**
 - **used for quick analysis**

CrayPAT (Perftools-lite) steps

- Profiling steps for a Jacobi solver written in Fortran with MPI and OpenMP
- Code must be run in `$SCRATCH` (or set an `env` to run it from elsewhere)
- Object (`*.o`) files must be created in a separate step and remain present
- To visualize, use App2, launch terminal with X forwarding or use NX (<https://docs.nersc.gov/connect/nx/>)
- Run steps are as follows:

```
$ module unload darshan xalt
$ module load perftools-lite
$ module load PrgEnv-cray
$ ftn -c -fopenmp jacobi_mpiomp.F90
$ ftn -fopenmp -o jacobi_mpiomp jacobi_mpiomp.o
$ salloc --nodes 4 --qos interactive --time 01:00:00 --C cpu --account=mxxxx
$ srun -n 32 --cpu-bind=cores ./jacobi_mpiomp
```

CrayPAT (Perftools-lite) results

```
#####  
#                                                                 #  
#          CrayPat-lite Performance Statistics                    #  
#                                                                 #  
#####
```

CrayPat/X: Version 22.06.0 Revision 4b5ab6256 05/21/22 02:03:49

Experiment: lite lite-samples

Number of PEs (MPI ranks): 32

Numbers of PEs per Node: 8 PEs on each of 4 Nodes

Numbers of Threads per PE: 2

Number of Cores per Socket: 64

Execution start time: Wed Sep 21 11:00:59 2022

System name and speed: nid005666 2.334 GHz (nominal)

AMD Milan CPU Family: 25 Model: 1 Stepping: 1

Core Performance Boost: 64 PEs have CPB capability

Avg Process Time: 7.13 secs

High Memory: 6,850.6 MiBytes 214.1 MiBytes per PE

I/O Read Rate: 14.231151 MiBytes/sec

I/O Write Rate: 11.174613 MiBytes/sec

CrayPAT (Perftools-lite) results

Notes for table 1:

This table shows functions that have significant exclusive sample hits, averaged across ranks.

For further explanation, use: `pat_report -v -O samp_profile ...`

Table 1: Profile by Function

Samp%	Samp	Imb.	Imb.	Group
		Samp	Samp%	Function=[MAX10]
				PE=HIDE
				Thread=HIDE
100.0%	687.2	--	--	Total

70.3%	483.0	--	--	USER

53.3%	366.3	16.7	4.5%	jacobi_mpiomp_.LOOP@li.61
16.8%	115.7	34.3	23.6%	compute_diff_.LOOP@li.261
=====				
22.3%	153.4	--	--	ETC

20.9%	143.3	32.7	19.2%	__cray_memcpy_ROME
=====				
6.3%	43.4	--	--	MPI

4.6%	31.3	13.7	31.3%	MPI_ALLREDUCE
1.7%	11.5	16.5	60.8%	MPI_SENDRECV
=====				

CrayPAT (Perftools-lite) results

Notes for table 2:

This table shows functions, and line numbers within functions, that have significant exclusive sample hits, averaged across ranks. For further explanation, use: `pat_report -v -O samp_profile+src ...`

Table 2: Profile by Group, Function, and Line

Samp%	Samp	Imb.	Imb.	Group
		Samp	Samp%	Function=[MAX10]
				Source
				Line
				PE=HIDE
				Thread=HIDE
100.0%	687.2	--	--	Total

70.3%	483.0	--	--	USER

53.3%	366.3	--	--	jacobi_mpiomp_.LOOP@li.61
3				n/namehta4/Tutorial/jacobi_mpiomp.F90

33.5%	230.3	25.7	10.4%	line.63
4	19.4%	133.0	13.0	9.2%

16.8%	115.7	--	--	compute_diff_.LOOP@li.261
3				n/namehta4/Tutorial/jacobi_mpiomp.F90

5.8%	39.9	22.1	36.8%	line.263
4	11.0%	75.4	23.6	24.6%

22.3%	153.4	--	--	ETC

20.9%	143.3	32.7	19.2%	__cray_memcpy_R0ME

6.3%	43.4	--	--	MPI

4.6%	31.3	13.7	31.3%	MPI_ALLREDUCE
1	1.7%	11.5	16.5	60.8%

CrayPAT (Perftools-lite) results

Notes for table 3:

This table shows functions that have the most significant exclusive time, taking for each thread the average time across ranks. The imbalance percentage is relative to the team observed to participate in execution.

Use -s th=ALL to see individual thread values.

For further explanation, use: `pat_report -v -O profile_th_pe ...`

Table 3: Profile by Function Group and Function

Samp%	Samp	Imb.	Imb.	Team	Group
		Samp	Samp%	Size	Function=[MAX10]
					Thread=HIDE
					PE=HIDE
100.0%	687.2	--	--	--	Total
70.3%	483.0	--	--	--	USER
53.3%	366.3	1.3	0.7%	2	jacobi_mpiomp_LOOP@li.61
16.8%	115.7	1.8	3.0%	2	compute_diff_LOOP@li.261
22.3%	153.4	--	--	--	ETC
20.9%	143.3	4.5	6.3%	2	__cray_memcpy_ROME
6.3%	43.4	--	--	--	MPI
4.6%	31.3	--	--	1	MPI_ALLREDUCE
1.7%	11.5	--	--	1	MPI_SENDRECV

CrayPAT (Perftools-lite) results

Notes for table 3:
Notes for table 4:

This table shows energy and power usage for the nodes with the maximum, mean, and minimum usage, as well as the sum of usage over all nodes.

Energy and power for accelerators is also shown, if applicable.

For further explanation, use: `pat_report -v -O program_energy ...`

Table 4: Program energy and power usage (from Cray PM)

Node Energy (J)	Node Power (W)	Process Time	Node Id PE=HIDE Thread=HIDE
10,873	1,524.726	7.131116	Total

2,730	382.878	7.130208	nid.2
2,729	382.716	7.130610	nid.1
2,707	379.554	7.132054	nid.0
2,707	379.579	7.131592	nid.3
=====			

CrayPAT (Perftools-lite) results

Notes for table 6:

This table shows the average time and number of bytes written to each output file, taking the average over the number of ranks that wrote to the file. It also shows the number of write operations, and average rates.

For further explanation, use: `pat_report -v -O write_stats ...`

Table 6: File Output Stats by Filename

Avg Write Time per Writer Rank	Avg Write MiBytes per Writer Rank	Write Rate MiBytes/sec	Number of Writer Ranks	Avg Writes per Writer Rank	Bytes/ Call	File Name=!x/^(proc sys)/ PE=HIDE
0.000091	0.002563	28.099071	1	101.0	26.60	stdout
0.000062	0.000439	7.044082	6	7.5	61.33	_UnknownFile_

Program invocation: `/pscratch/sd/n/namehta4/Tutorial/./jacobi_mpiomp`

CrayPAT (Perftools-lite) results

Notes for table 1:

This table shows functions that have significant exclusive time, averaged across ranks.

For further explanation, use: `pat_report -v -O profile ...`

Table 1: Profile by Function Group and Function

Time%	Time	Imb.	Imb.	Calls	Group	
		Time	Time%		Function=[MAX10]	^/(proc sys)/
100.0%	0.651397	--	--	26,518.0	Total	

98.0%	0.638061	--	--	22,056.0	CUDA	

92.3%	0.601155	--	--	2,000.0	cudaLaunchKernel	-
4.5%	0.029539	--	--	2,001.0	cudaMemcpy	-
=====						
1.8%	0.011568	--	--	2.0	USER	-

1.5%	0.009741	--	--	1.0	InitializeDataChunk	=
=====						

CrayPAT (Perftools) steps

- **Similar to steps used for Perftools-lite**
- **Code must be run in \$SCRATCH**
- **Object (*.o) files must be created in a separate step and maintained**
- **To visualize results using App2, launch terminal with X forwarding**
- **Run steps are as follows:**

```
$ pat_build -u -g mpi <name of exec>
```

```
$ module unload darshan xalt
$ module load perftools
$ export PAT_RT_EXPDIR_NAME=./data_dir
$ ftn -c -fopenmp jacobi_mpiomp.F90
$ ftn -fopenmp -o jacobi_mpiomp jacobi_mpiomp.o
$ salloc --nodes 32 --qos interactive --time 01:00:00 --C cpu --account=mxxxxx
$ pat_build -O apa jacobi_mpiomp
$ srun -n 32 ./jacobi_mpiomp+pat
```

<https://www.nersc.gov/assets/Uploads/Perftools-Reveal-Levesque-Sept2022.pdf>

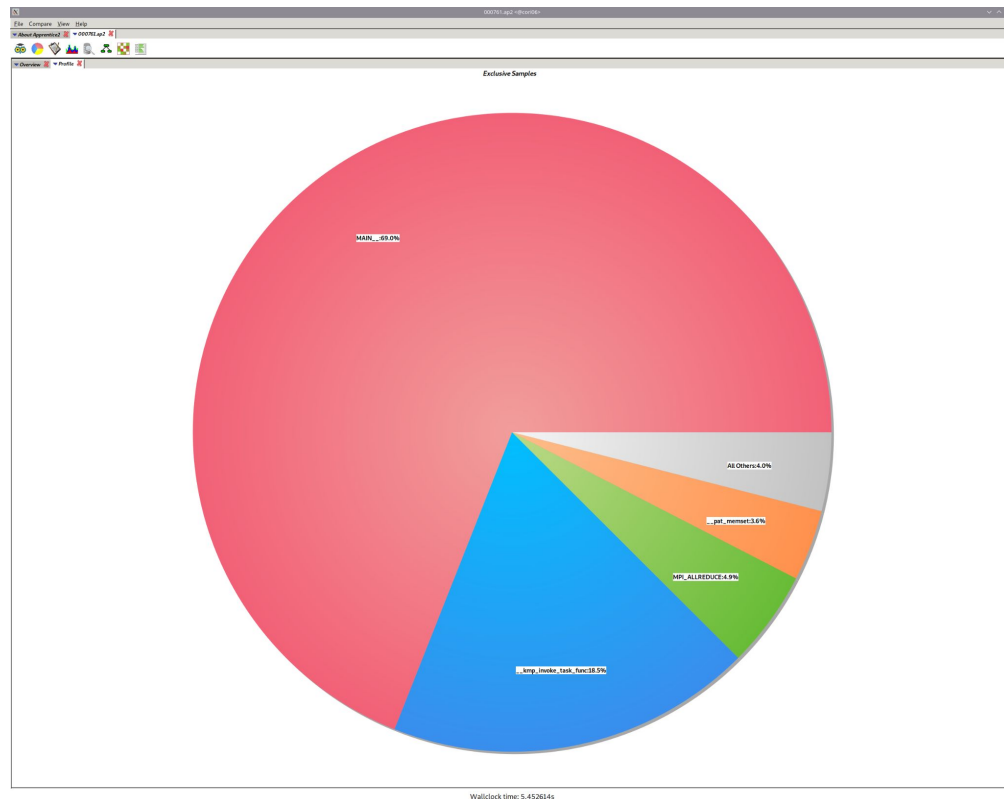
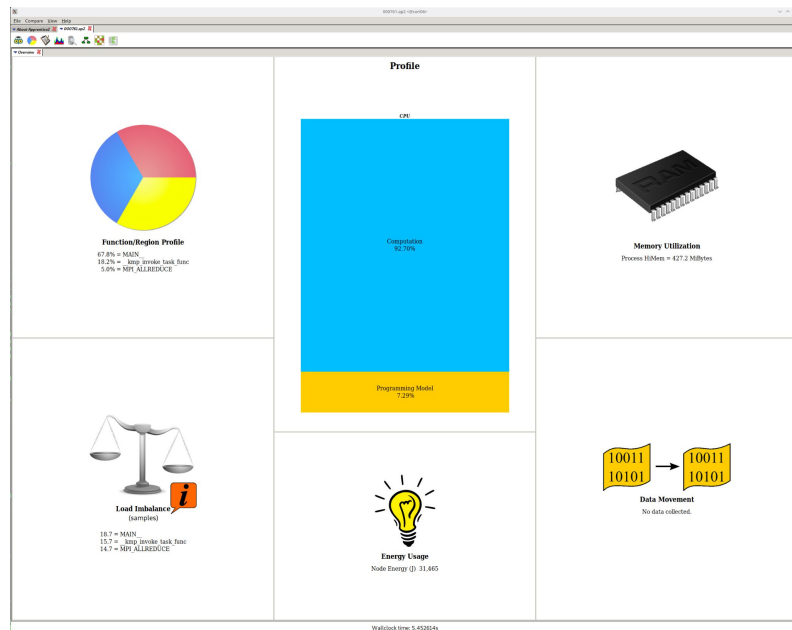
CrayPAT (Perftools) steps

- Running the +pat compiled binary will generate .xf files in the data_dir folder
- These .xf2 files have to be converted to ap2 format to visualize using App2
- Run steps are as follows:

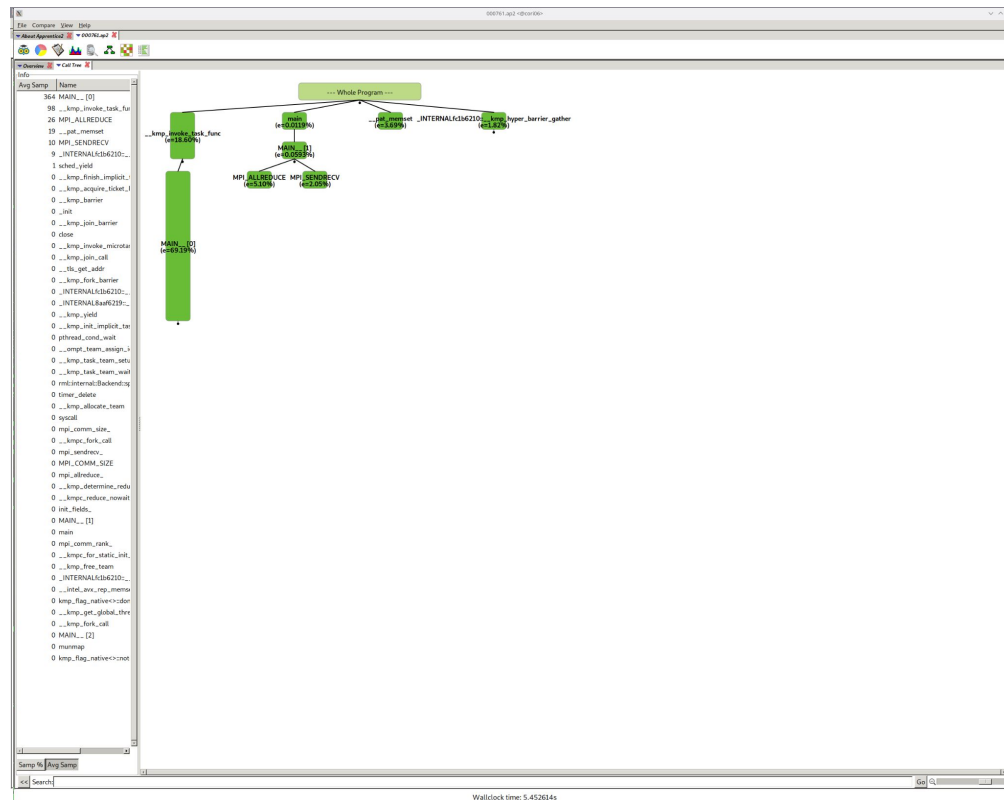
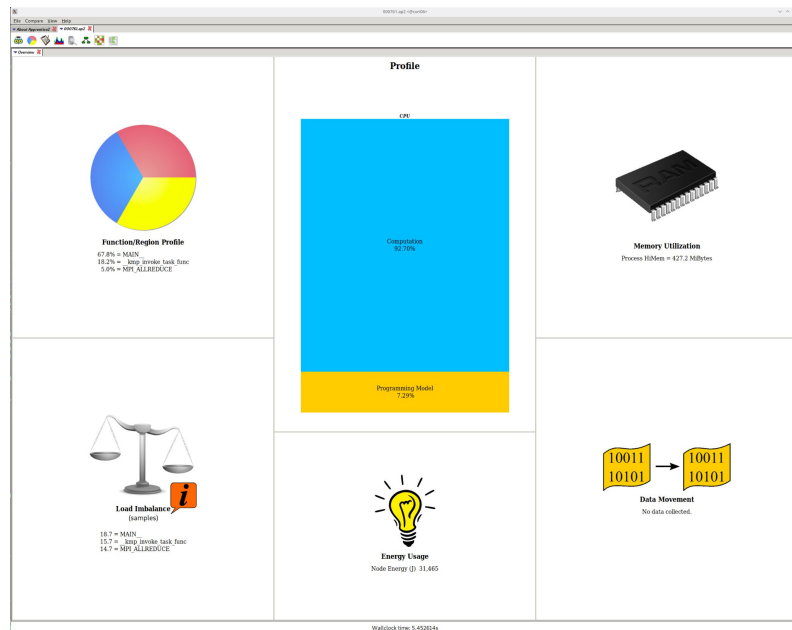
```
$ cd ./data_dir  
$ pat_report -f ap2 *.xf2  
$ app2 result.ap2
```



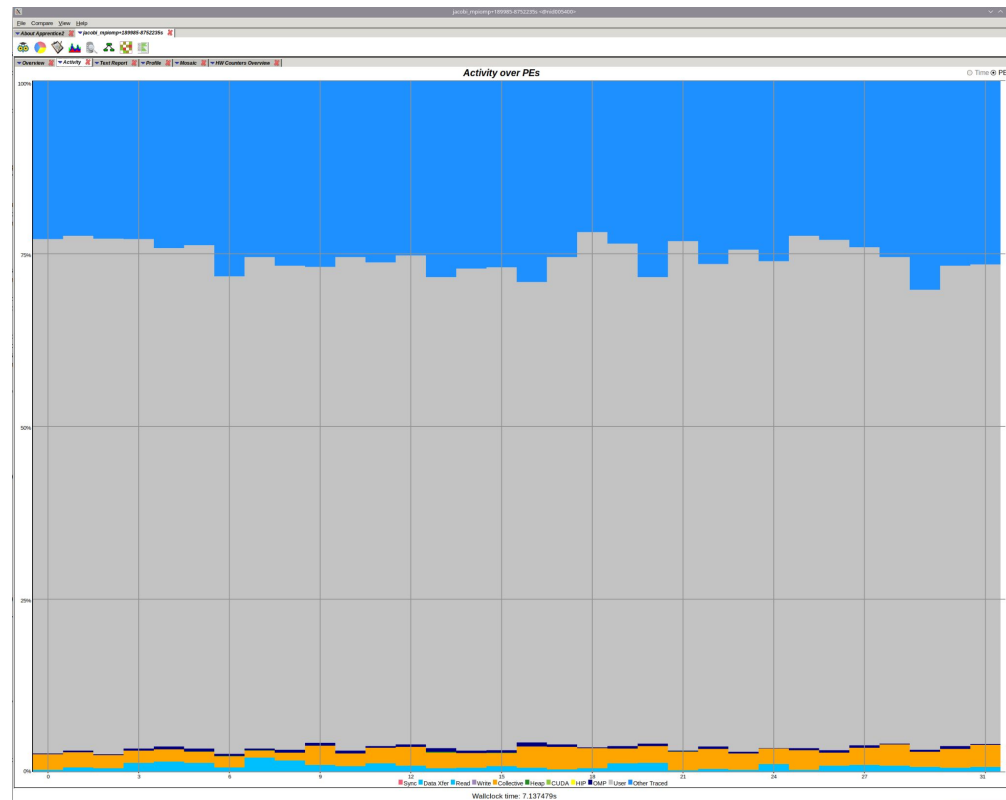
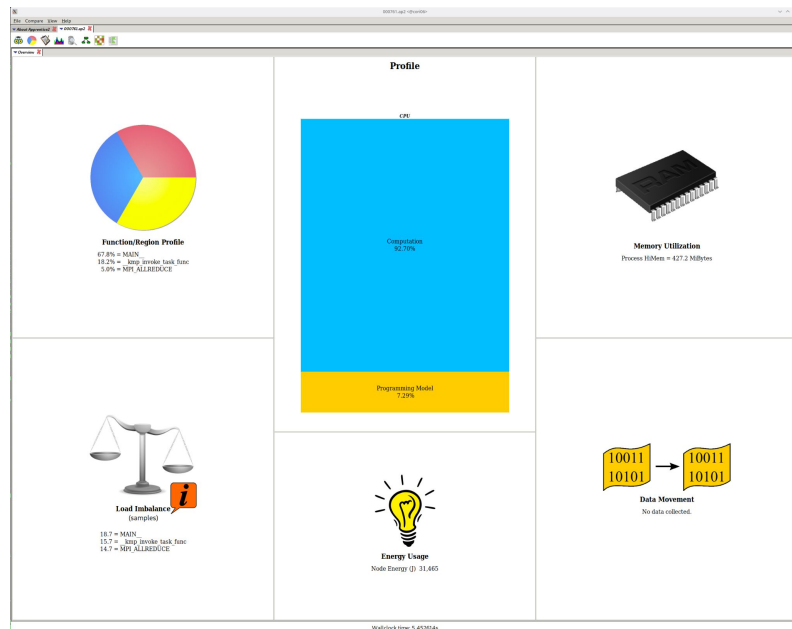
Visualizing CrayPAT (Perftools) results in App2



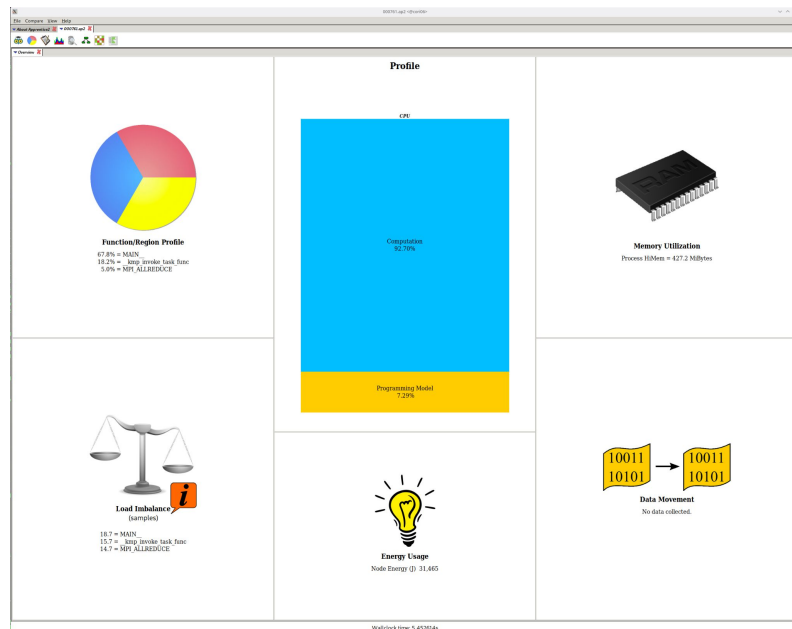
Visualizing CrayPAT (Perftools) results in App2



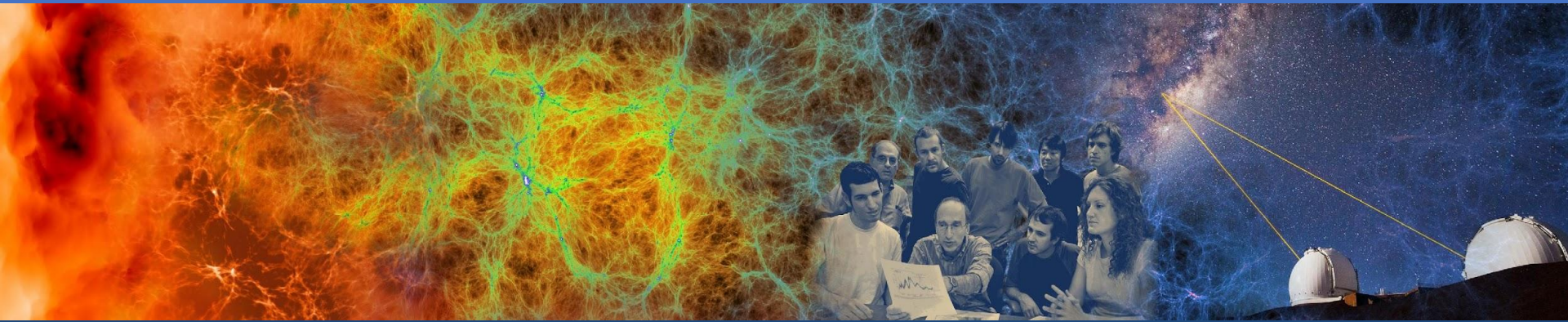
Visualizing CrayPAT (Perftools) results in App2



Visualizing CrayPAT (Perftools) results in App2



Nsight Systems



NVIDIA Nsight Systems

- **Nsight systems is a low overhead profiler**
- **Provides general broad description of GPU based applications**
- **Modules**
 - **cuda**
 - **required to load Nsight systems**
- **Application must be compiled by linking cuda libraries**
- **Supports:**
 - **cuda (nvcc, nvc++)**
 - **Kokkos (cuda, OpenMP)**
 - **OpenMP**
 - **OpenACC**

NVIDIA Nsight Systems steps

- **Profiling steps for an OpenMP Offload based application compiled using clang++**
- **Code run in \$SCRATCH**
- **To visualize results, it is recommended to transfer profile files to local machine and use local install of Nsight Systems (available for free)**
- **Run steps are as follows:**

```
$ module unload darshan
$ module load PrgEnv-<required>
$ ... compile your code ...
$ salloc --nodes 1 --qos interactive --time 01:00:00 --C gpu --account=mxxxx
$ srun -n 1 nsys profile --stats=true ./test_snap.exe -ns 100
```

NVIDIA Nsight Systems steps

- Generates text of profiling results

CUDA API Statistics:

Time(%)	Total Time (ns)	Num Calls	Average	Minimum	Maximum	Name
83.9	6563435467	805	8153336.0	2413	44859498	cuStreamSynchronize
14.1	1100682085	33	33354002.6	2335	656900656	cuMemcpyHtoDAsync_v2
1.3	99718751	1	99718751.0	99718751	99718751	cuDevicePrimaryCtxRelease_v2
0.5	40948782	200	204743.9	26326	1158675	cuMemcpyDtoHAsync_v2
0.1	11082944	700	15832.8	6302	64274	cuLaunchKernel
0.1	6082361	17	357785.9	2668	2189898	cuMemAlloc_v2
0.0	858283	1	858283.0	858283	858283	cuModuleLoadDataEx
0.0	668654	32	20895.4	1345	353717	cuStreamCreate
0.0	654039	1	654039.0	654039	654039	cuModuleUnload
0.0	192187	32	6005.8	3933	21337	cuStreamDestroy_v2
0.0	159833	7	22833.3	14950	58247	cuMemcpyDtoH_v2
0.0	2250	1	2250.0	2250	2250	cuDevicePrimaryCtxSetFlags_v2

NVIDIA Nsight Systems steps

- Kernel names may be mangled (eq:

`__omp_offloading_70e68f56_5d00401c__ZN3SNA10compute_yiEPd_1471)`

CUDA Kernel Statistics:

Time(%)	Total Time (ns)	Instances	Average	Minimum	Maximum	Name
-----	-----	-----	-----	-----	-----	
61.4	3946535611	100	39465356.1	39233639	44223639	__ZN3SNA10compute_yiEPd_1471
17.5	1125861668	100	11258616.7	11178964	11771631	__ZN3SNA14compute_duidrjEv_1652
16.3	1048090801	100	10480908.0	10082240	11828974	__ZN3SNA10compute_uiEv_1425
3.8	243394690	100	2433946.9	2403145	2521384	__ZN3SNA14compute_deidrjEv_1577
0.8	53957727	100	539577.3	535003	635610	__ZN3SNA14zero_uarraytotEv_1687
0.1	8023377	100	80233.8	79007	93024	__ZN3SNA10compute_yiEPd_1460
0.0	2334567	100	23345.7	22720	29376	__ZN3SNA17addself_uarraytotEd_1704

NVIDIA Nsight Systems steps

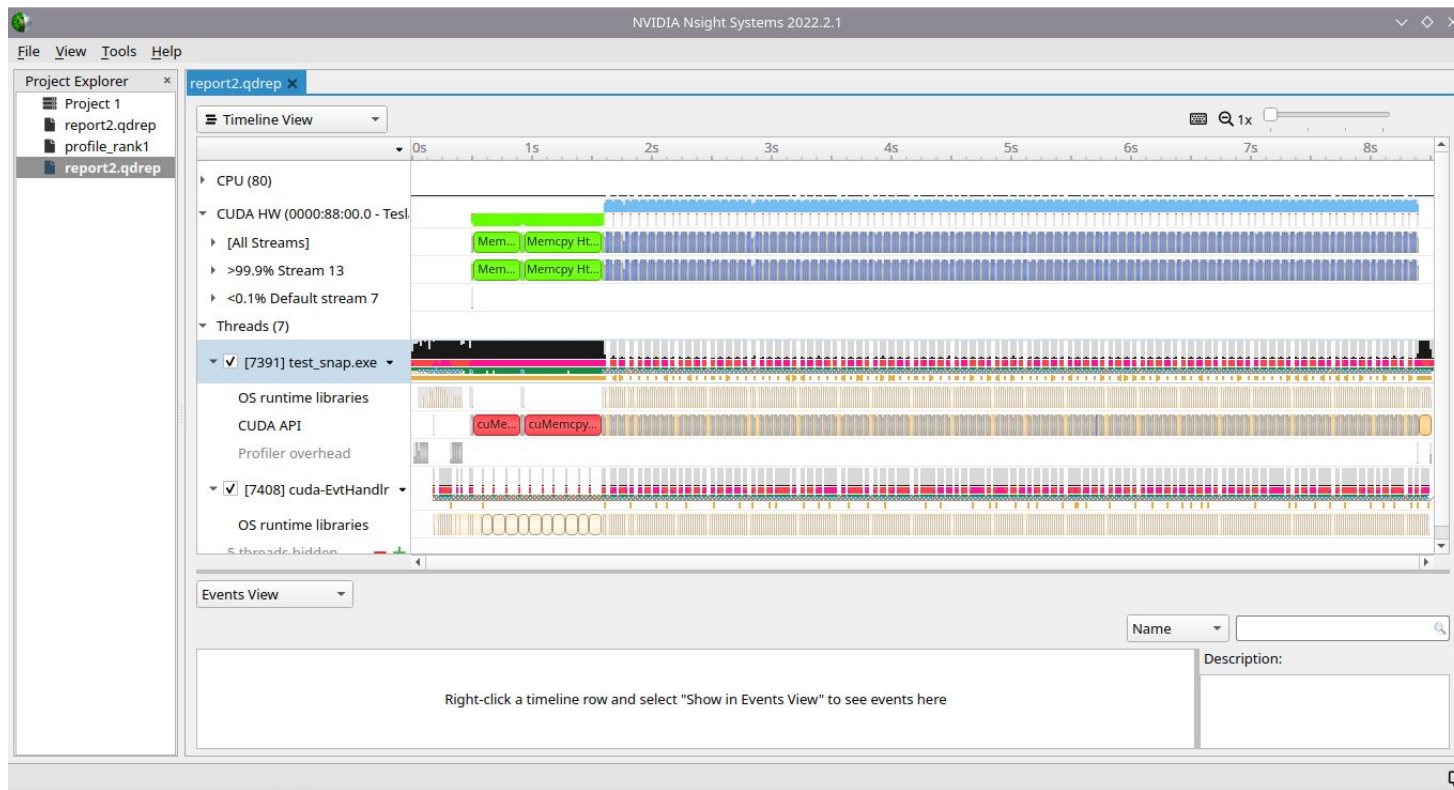
CUDA Memory Operation Statistics (by time):

Time(%)	Total Time (ns)	Operations	Average	Minimum	Maximum	Operation
-----	-----	-----	-----	-----	-----	-----
99.1	1098309937	33	33282119.3	1408	656553887	[CUDA memcpy HtoD]
0.9	10013795	207	48375.8	1696	99007	[CUDA memcpy DtoH]

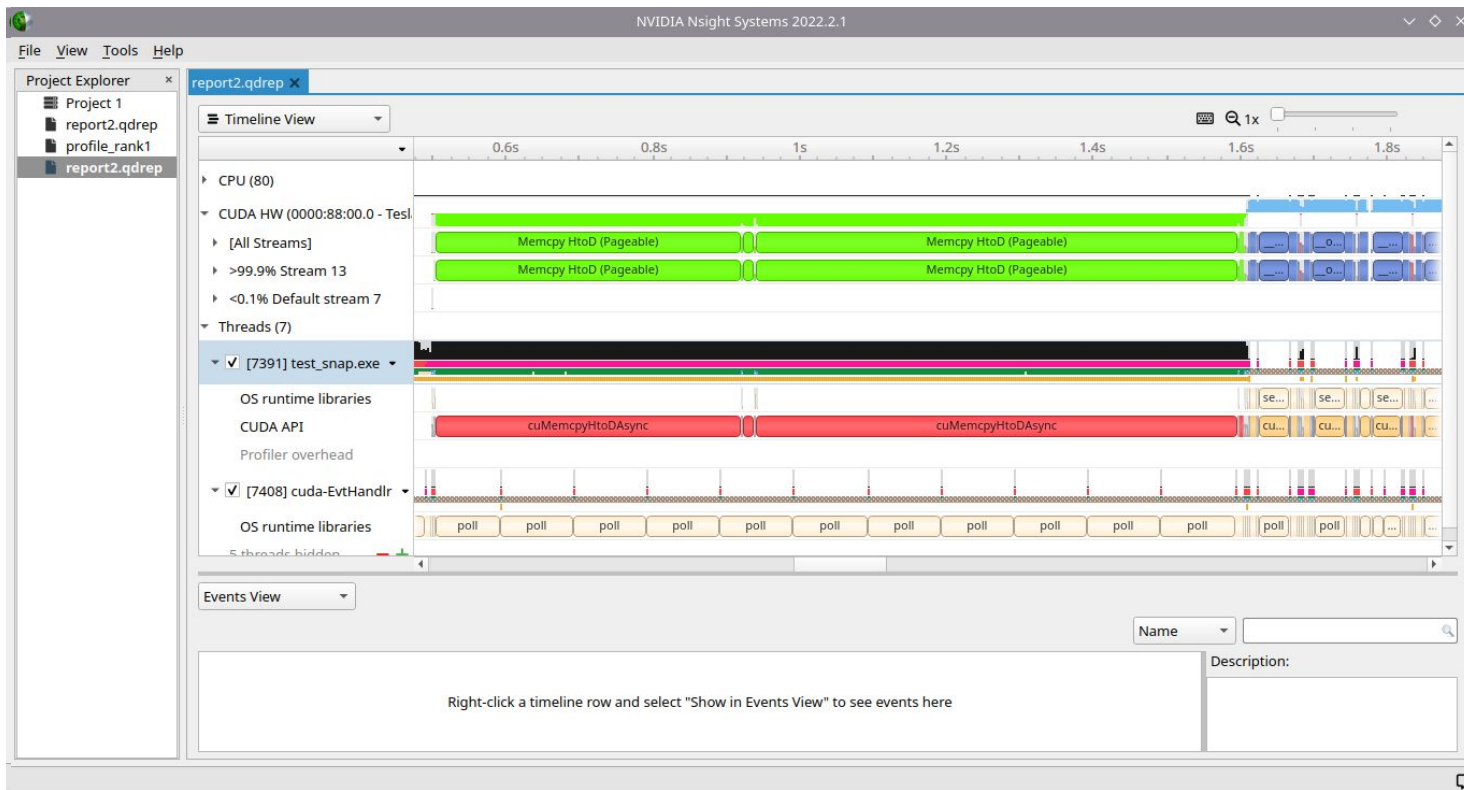
CUDA Memory Operation Statistics (by size in KiB):

Total	Operations	Average	Minimum	Maximum	Operation
-----	-----	-----	-----	-----	-----
2660577.207	33	80623.552	0.008	1589250.000	[CUDA memcpy HtoD]
121875.788	207	588.772	0.001	1218.750	[CUDA memcpy DtoH]

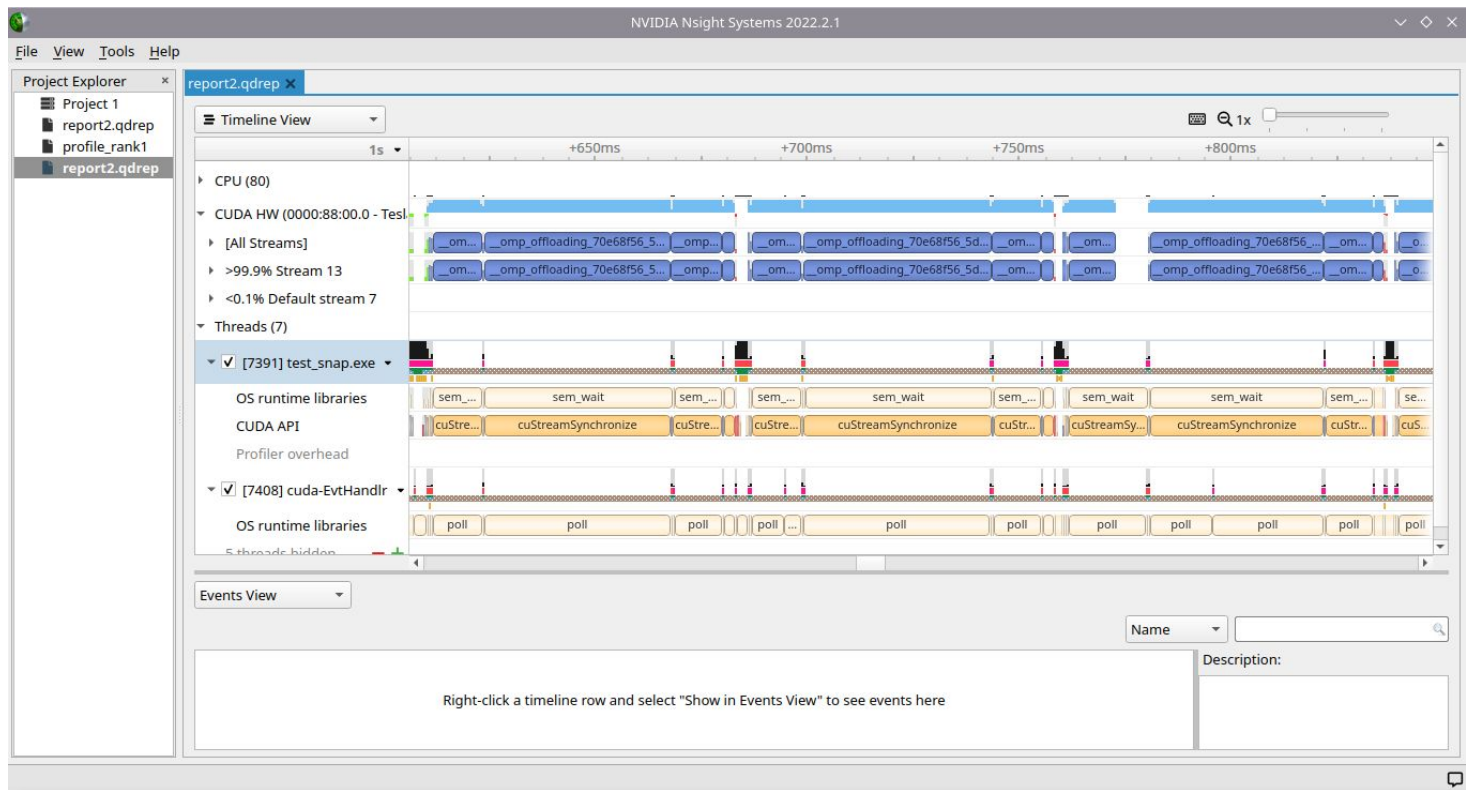
Visualizing results in NVIDIA Nsight Systems



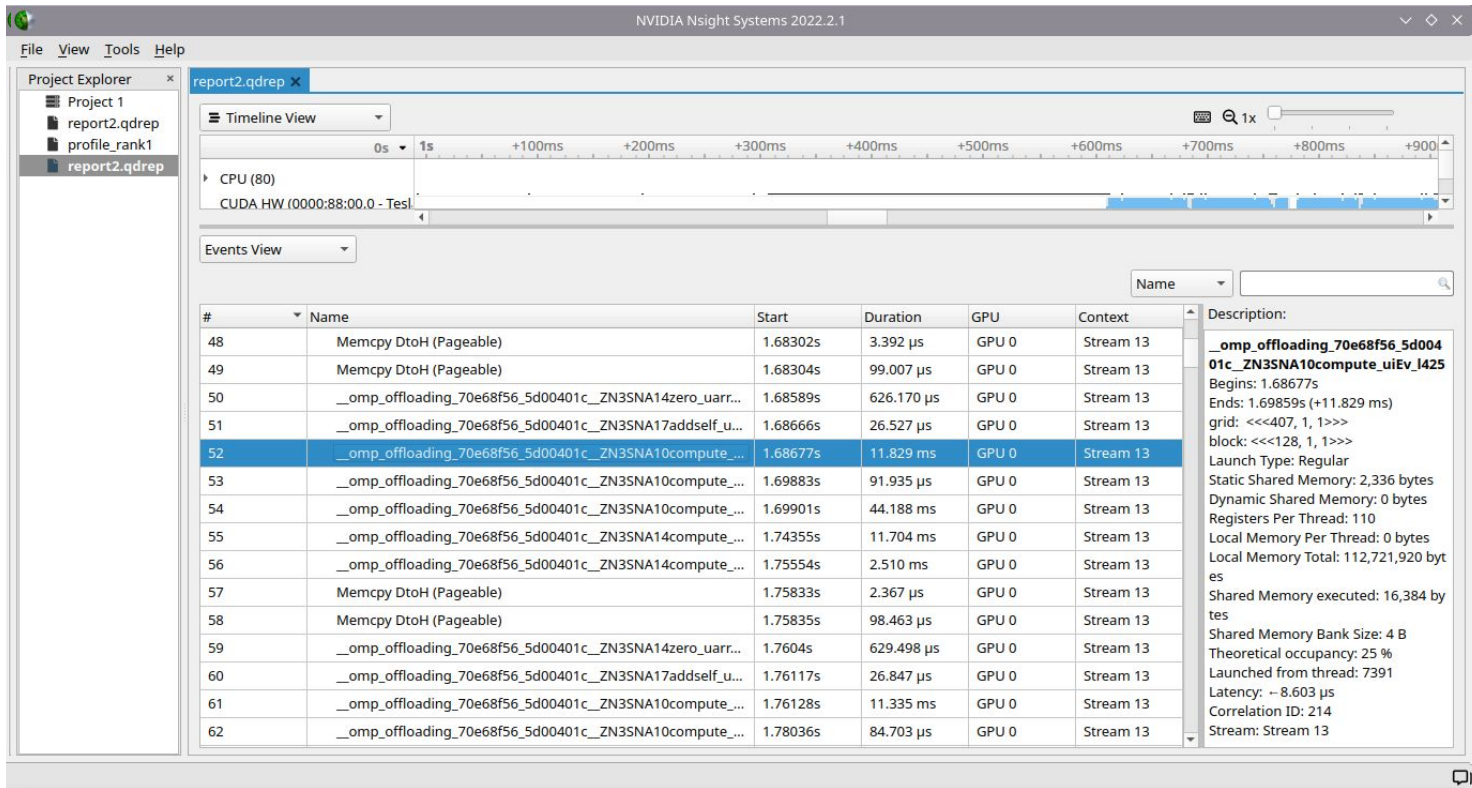
Visualizing results in NVIDIA Nsight Systems



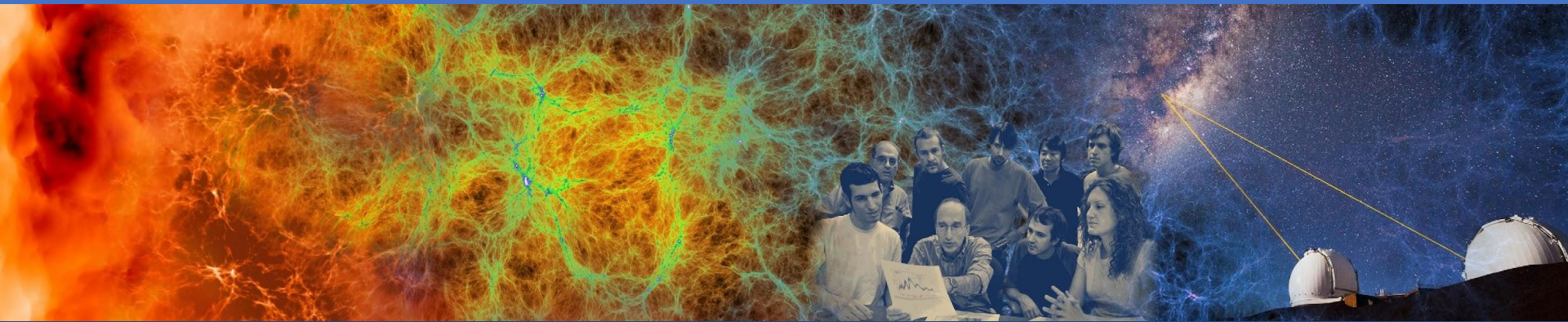
Visualizing results in NVIDIA Nsight Systems



Visualizing results in NVIDIA Nsight Systems



Nsight Compute



NVIDIA Nsight Compute steps

- Case 1: **baseline**
- Exploit the ability to **collapse** nested **for** loops
- Case 2: **collapse**

```
{  
#pragma omp target teams distribute parallel for  
    for(int natom = 0; natom < num_atoms; ++natom)  
        for(int nbor = 0; nbor < num_nbor; ++nbor)  
            for(int j = 0; j < idxu_max; ++j)  
            {  
                compute();  
            }  
}
```

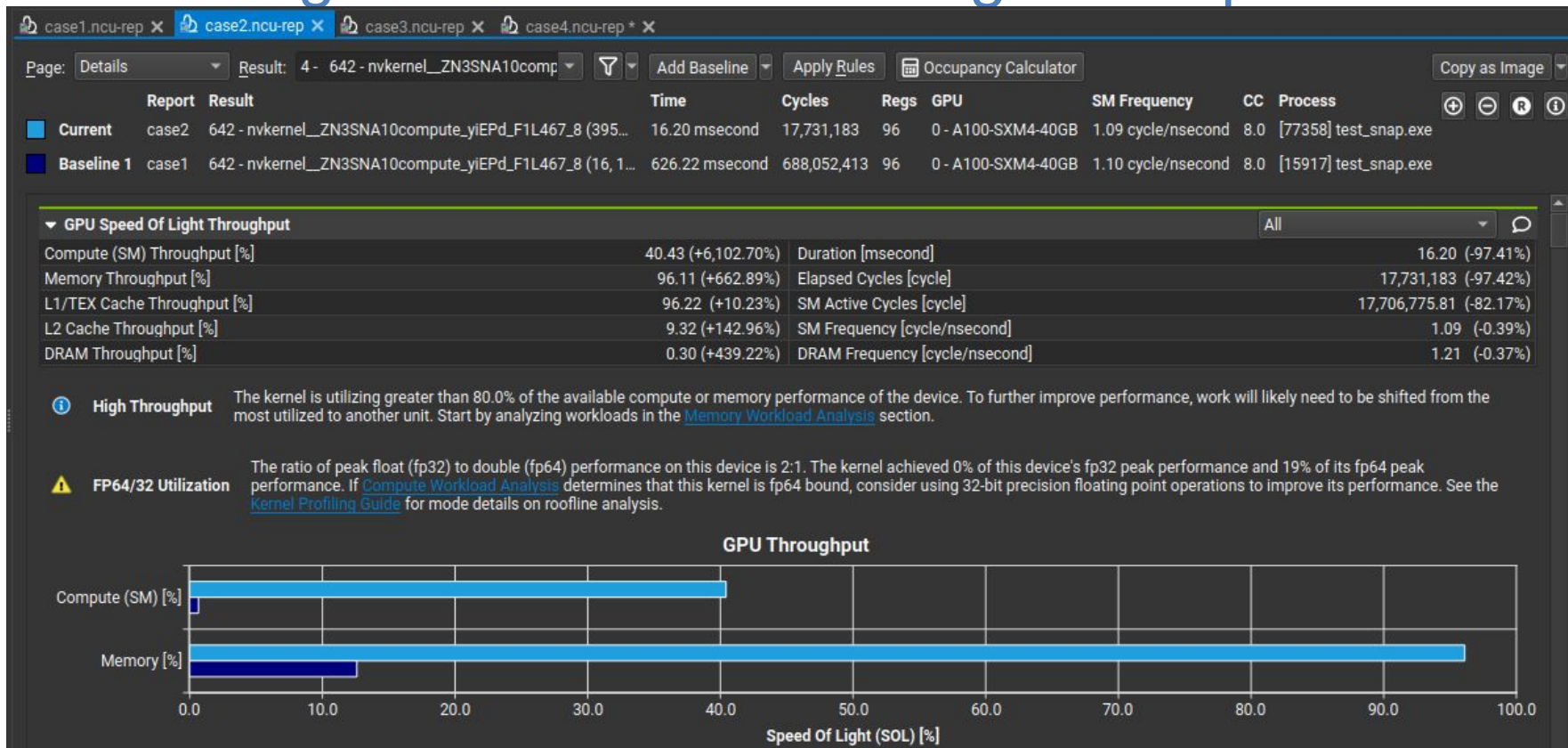
```
{  
#pragma omp target teams distribute parallel for collapse(2)  
    for(int natom = 0; natom < num_atoms; ++natom)  
        for(int nbor = 0; nbor < num_nbor; ++nbor)  
            for(int j = 0; j < idxu_max; ++j)  
            {  
                compute();  
            }  
}
```

NVIDIA Nsight Compute

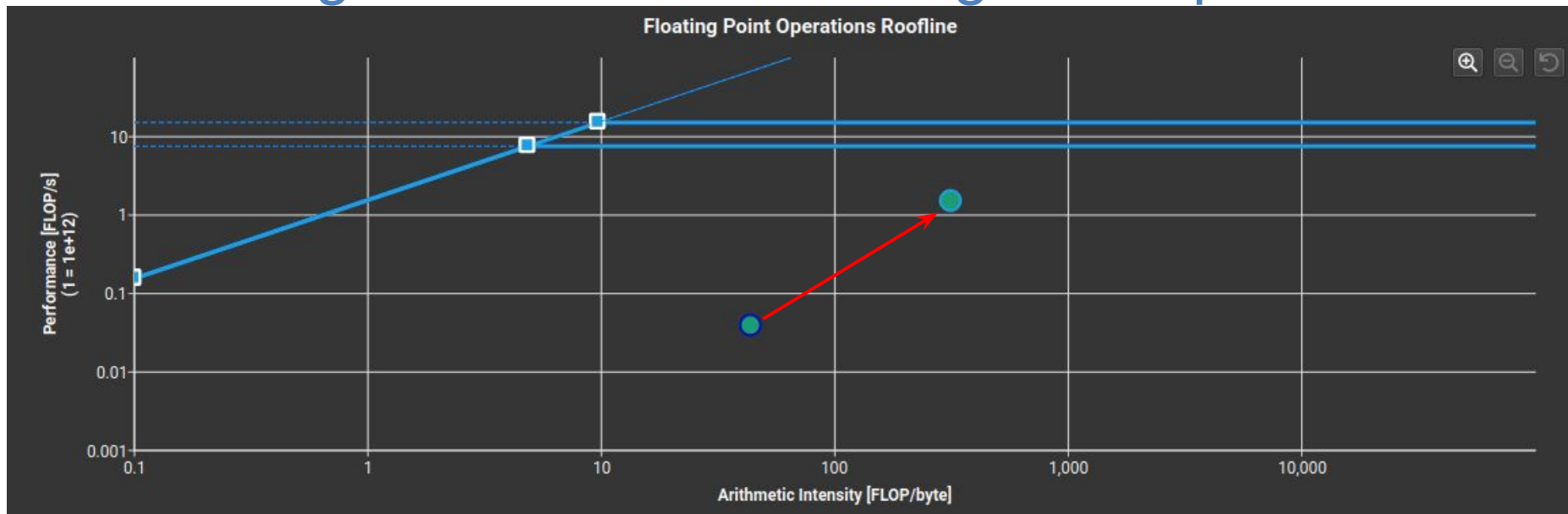
- Nsight compute allows a deeper dive for profiling GPU based applications
- Modules
 - cuda
 - required to load Nsight compute
- To visualize results, it is recommended to transfer profile files to local machine and use local install of Nsight Compute (available for free) or use NX (<https://docs.nersc.gov/connect/nx/>)

```
$ module unload darshan
$ module load PrgEnv-<required>
$ ... compile your code ...
$ salloc --nodes 1 --qos interactive --time 01:00:00 --C gpu --account=mxxxx
$ dcgmi profile --pause
$ srun -n 1 ncu -o profile_snap --set full ./test_snap.exe -ns 100
```

Visualizing results in NVIDIA Nsight Compute



Visualizing results in NVIDIA Nsight Compute



- Improvement in AI and Performance due to atom and neighbor loop fusing

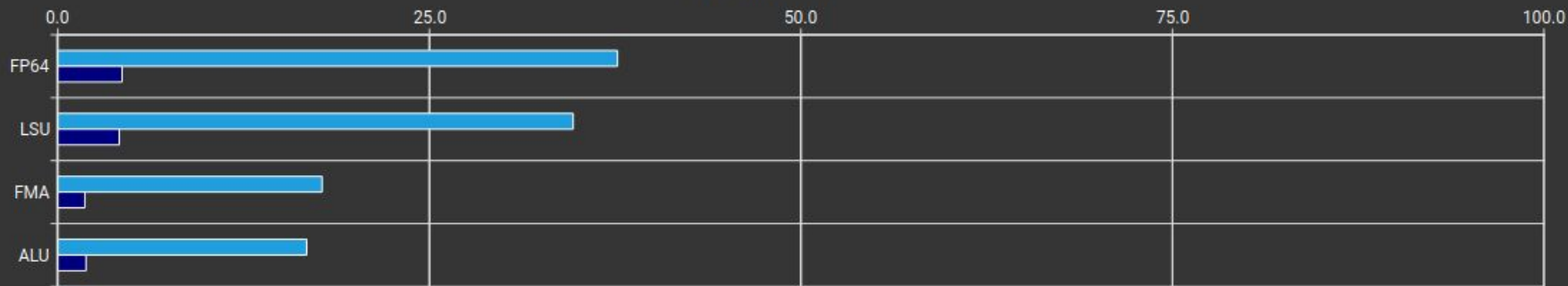
Visualizing results in NVIDIA Nsight Compute

▼ Compute Workload Analysis

Executed Ipc Elapsed [inst/cycle]	1.62 (+6,197.40%)	SM Busy [%]	40.48 (+810.22%)
Executed Ipc Active [inst/cycle]	1.62 (+809.91%)	Issue Slots Busy [%]	40.48 (+810.22%)
Issued Ipc Active [inst/cycle]	1.62 (+810.22%)		

 **Balanced** No pipeline is over-utilized.

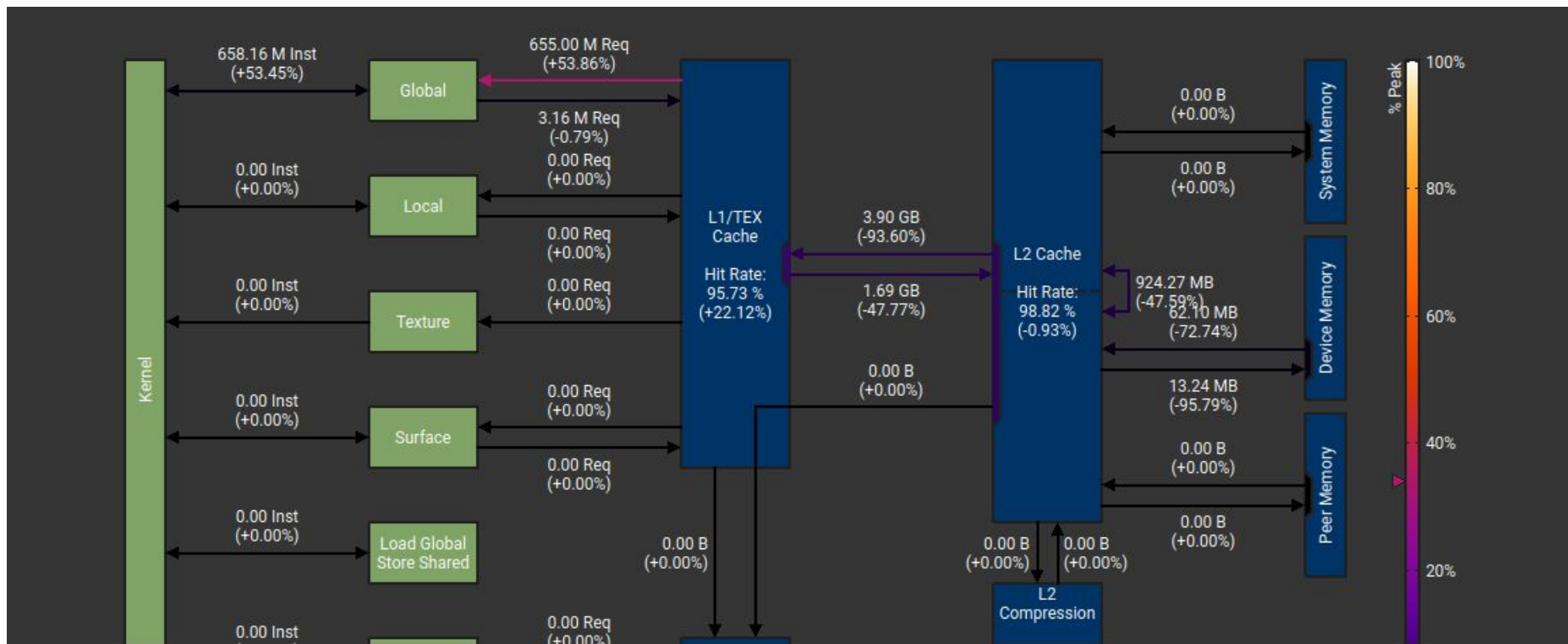
Pipe Utilization



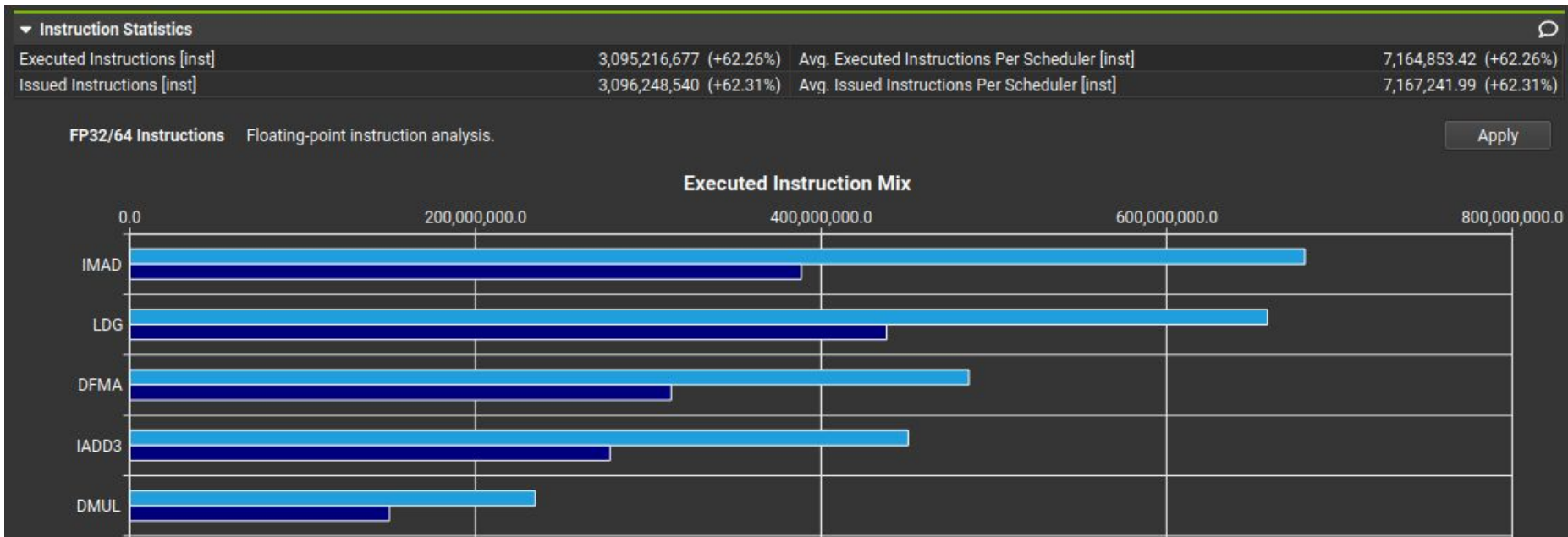
▼ Memory Workload Analysis

Memory Throughput [Gbyte/second]	4.65 (+437.23%)	Mem Busy [%]	96.11 (+662.89%)
L1/TEX Hit Rate [%]	95.73 (+22.12%)	Max Bandwidth [%]	67.65 (+1,374.25%)
L2 Hit Rate [%]	98.82 (-0.93%)	Mem Pipes Busy [%]	36.19 (+5,451.48%)
L2 Compression Success Rate [%]	0 (+0.00%)	L2 Compression Ratio	0 (+0.00%)

Visualizing results in NVIDIA Nsight Compute



Visualizing results in NVIDIA Nsight Compute



Thank You and
Welcome to
NERSC!

