`traccc`: Measuring GPU performance through code profiling

Minsi Chen

Department of Computer Science

University of Huddersfield, UK

Outline

- An overview of Nvidia profiling tools
  - Nsight Systems
  - Nsight Compute
- Profiling Acts GPU demonstrator – traccc (CUDA)
  - Performance summary
  - Clusterisation

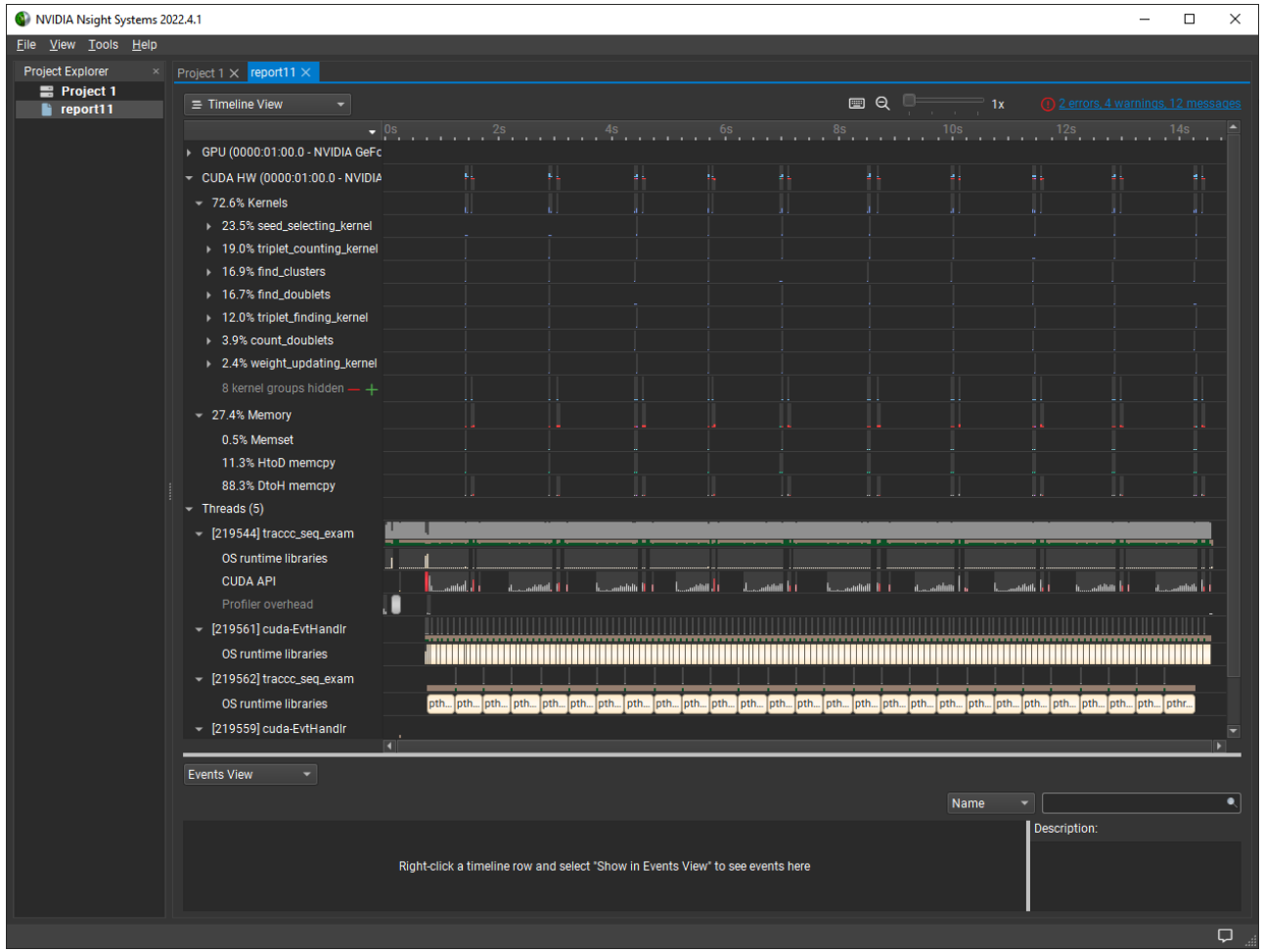
NVidia profiling tools

Nsight Systems -- https://developer.nvidia.com/nsight-systems

Nsight Compute -- https://developer.nvidia.com/nsight-compute

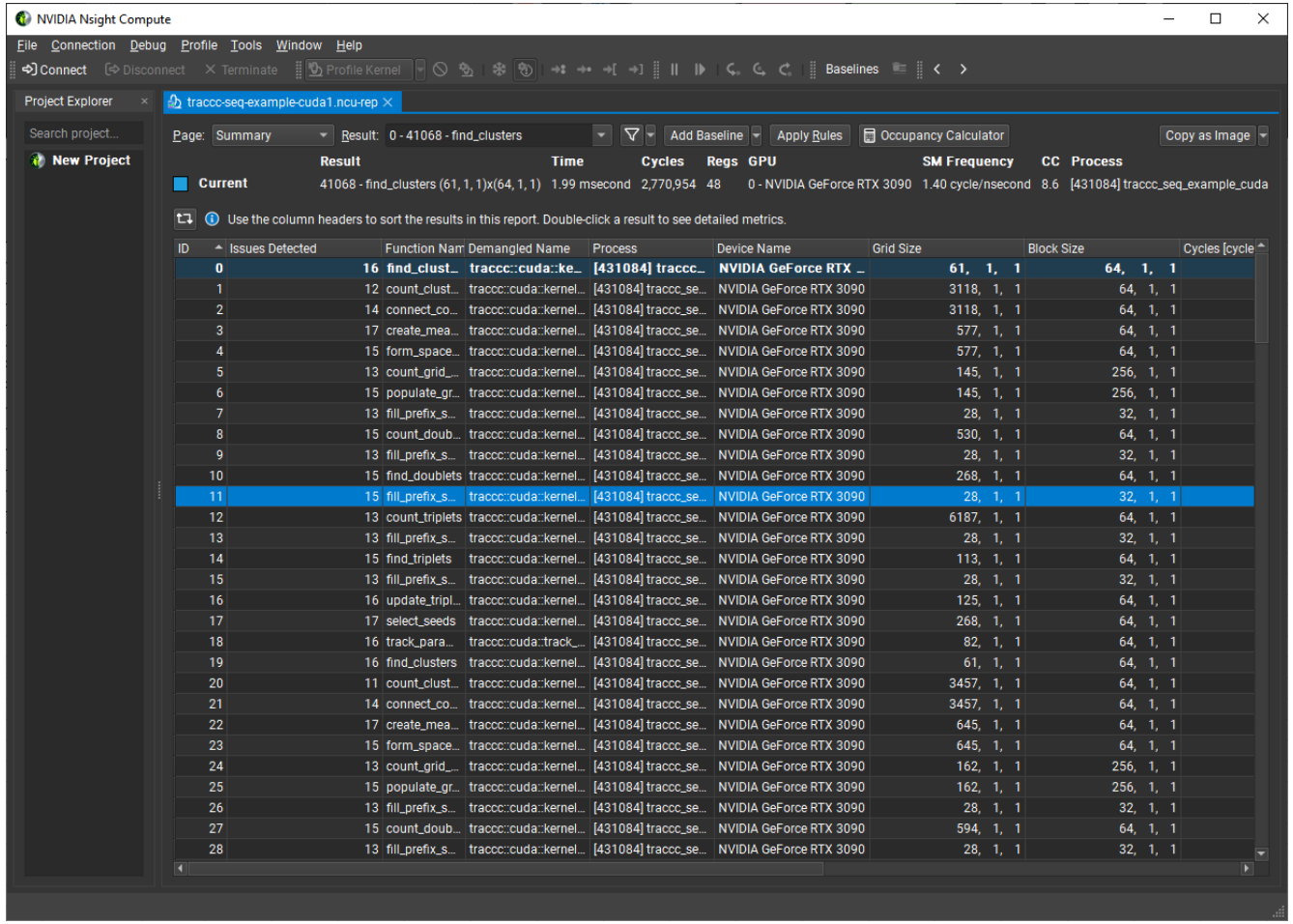
Nsight Systems

- System wide performance analysis
  - CUDA API calls
  - OS runtime libraries calls
  - Kernel and memcpy activities
- Not enough detail for fine tuning CUDA code



Nsight Compute

- Sampling based (min. 2^5 = 32 cycles/sample)
- Useful for detailed performance analysis including
  - Hardware utilisation
  - Memory workload analysis
  - Latency and timing
- Performance tuning assistance
  - Reasons for bottlenecks
  - Occupancy calculator



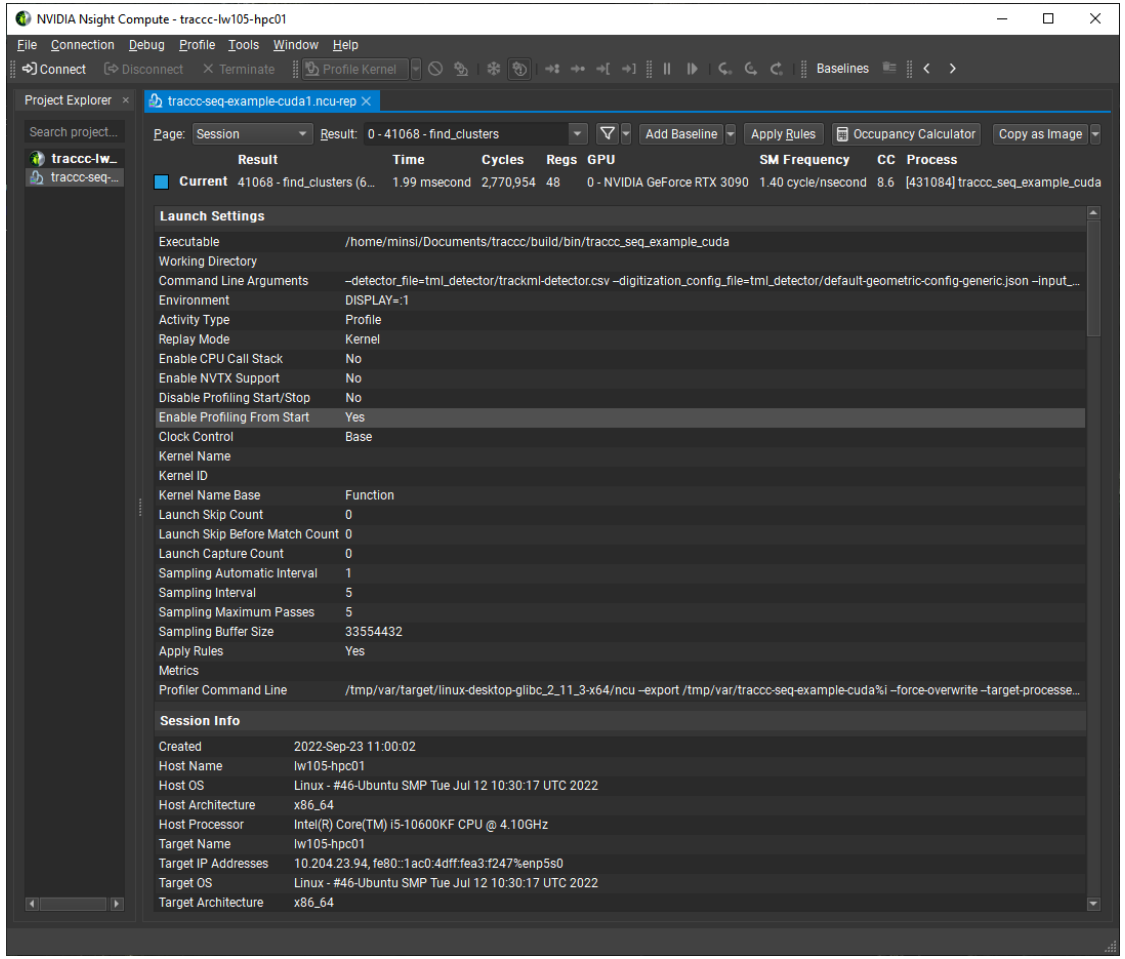
Profiling using Nsight Compute

- CLI command for a detailed profiling session (212 metrics)

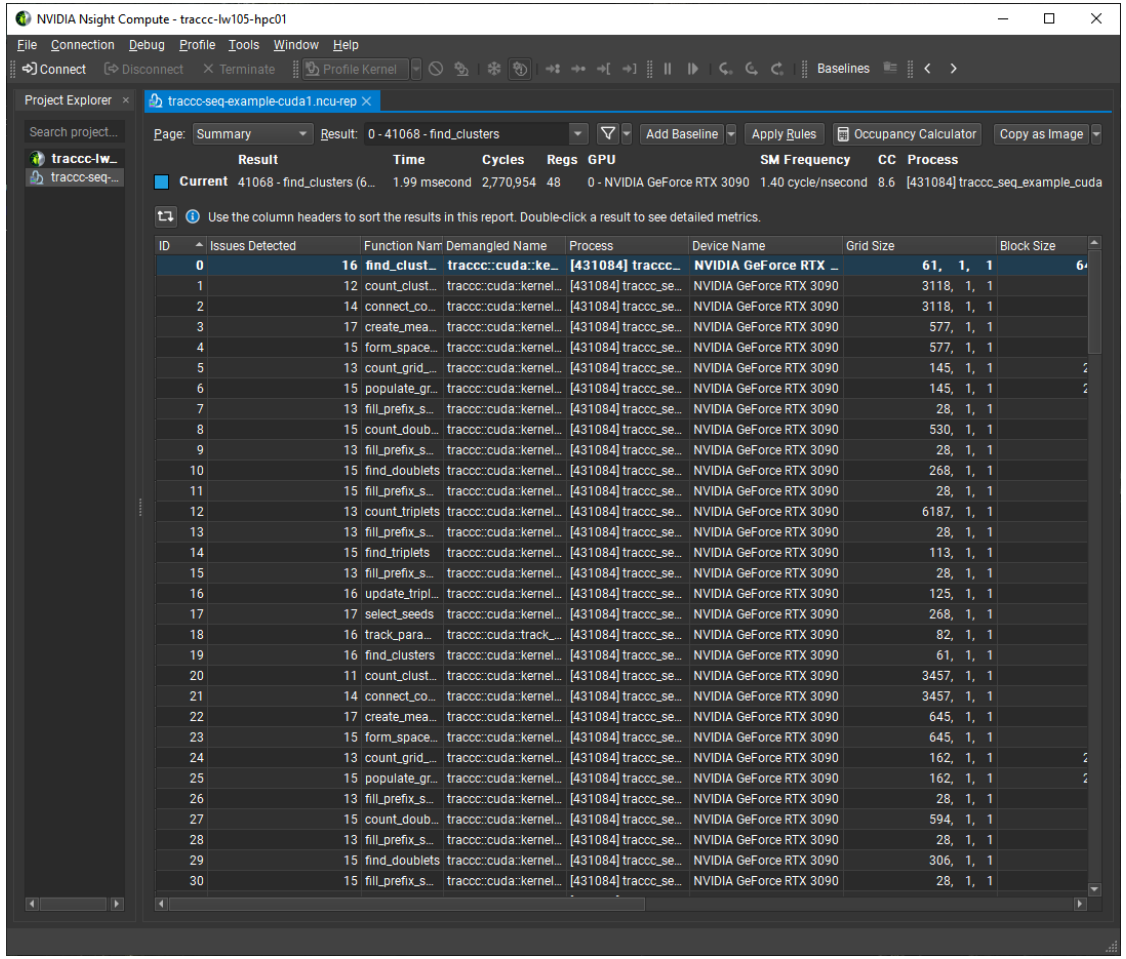
```
ncu /path/to/exec arg1 arg2 … --export /path/to/outputfile --force-overwrite --target-processes application-only --replay-mode kernel --kernel-name-base function --launch-skip-before-match 0 --filter-mode global --section ComputeWorkloadAnalysis --section InstructionStats --section LaunchStats --section MemoryWorkloadAnalysis --section Occupancy --section SchedulerStats --section SourceCounters --section SpeedOfLight --section SpeedOfLight_RooflineChart --section WarpStateStats --sampling-interval auto --sampling-max-passes 5 --sampling-buffer-size 33554432 --profile-from-start 1 --cache-control all --clock-control base --rule CPIStall --rule FPInstructions --rule HighPipeUtilization --rule IssueSlotUtilization --rule LaunchConfiguration --rule Occupancy --rule PCSamplingData --rule SOLBottleneck --rule SOLFPRoofline --rule SlowPipeLimiter --rule ThreadDivergence --rule UncoalescedGlobalAccess --rule UncoalescedSharedAccess --import-source no --check-exit-code yes
```

- Metrics are grouped into sections **(--section**)
  - *SpeedOfLight* – for high level compute and memory throughput
  - *Occupancy* – for detailed analysis on warps and compute bandwidth
- **--export** option generates one report file per profiling session (.ncu-rep)
  - Recommend viewing the report file using the GUI application

Nsight Compute report pages

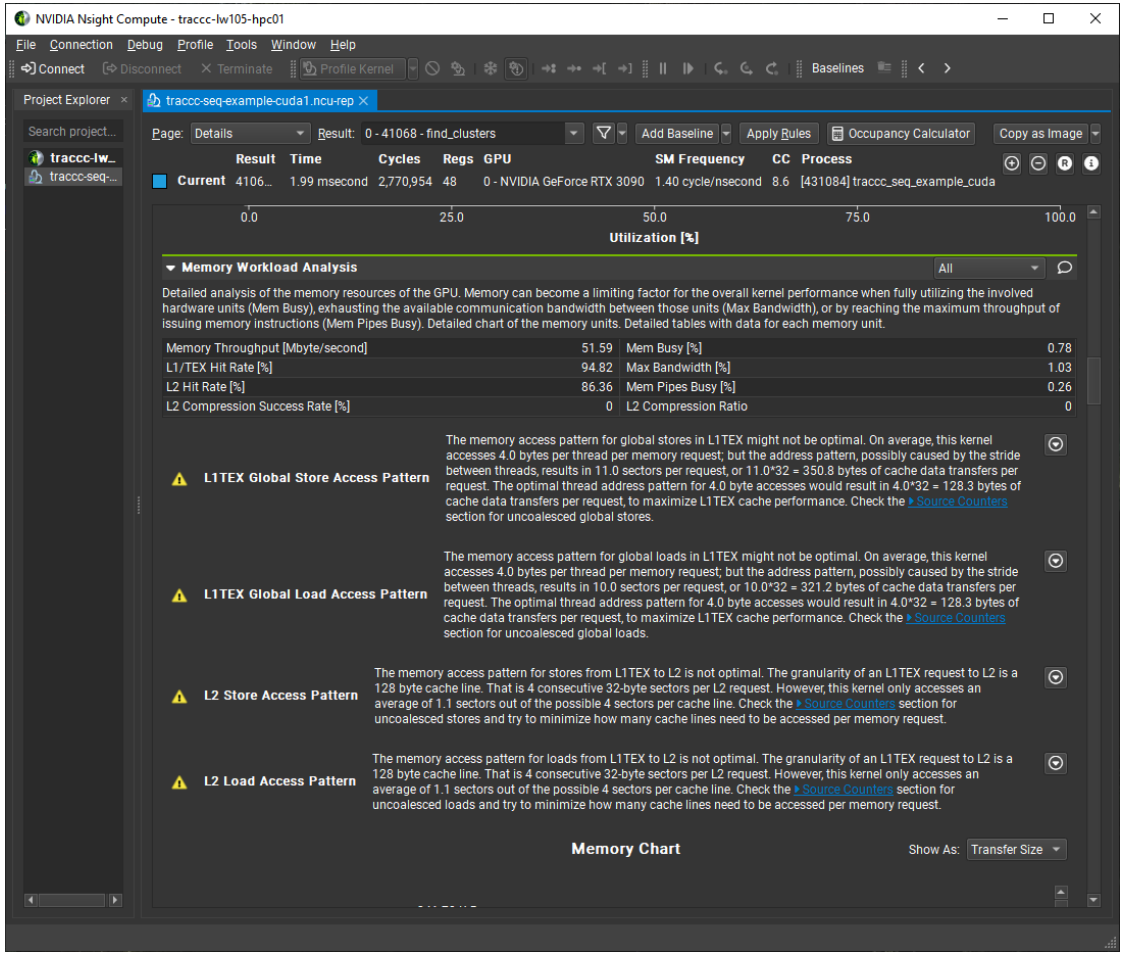


Session page: hardware, GPU capability, etc

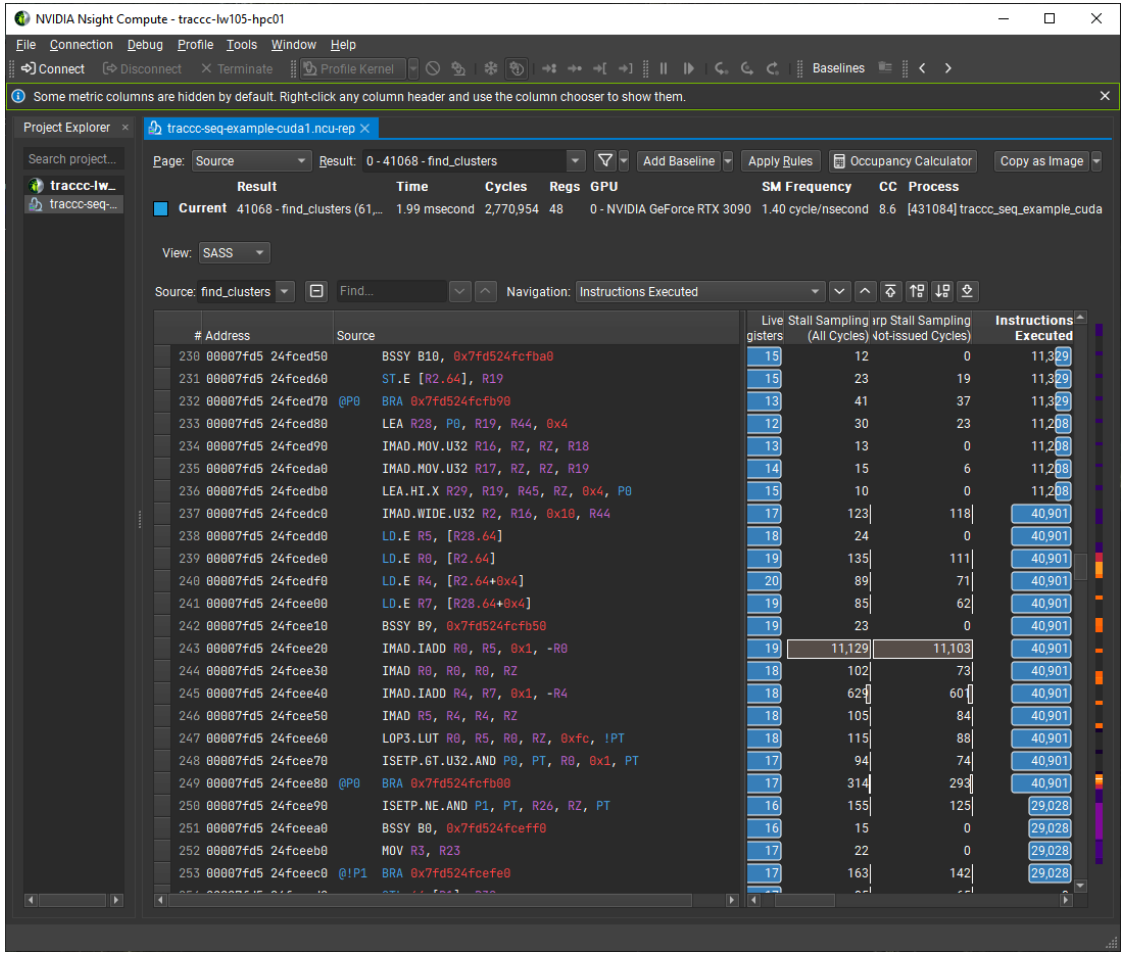


Summary page: high level performance metrics

Nsight Compute report pages



Details page: workload analysis along with tuning suggestions



Source page: disassembled instructions

Nsight Compute – limitations

- Only support three generations of Nvidia chips, i.e. Volta, Turing and Ampere, to show supported chips

```
ncu --list-chips
```

- Require administrator privilege, solutions
  - 1. For sudo/wheel group users, sudo is ok
  - 2. For normal users, system admins can add a conf file containing the following to /etc/modprobe.d

```
option nvidia “Nvreg_RestrictProfilingToAdminUsers=0”
```

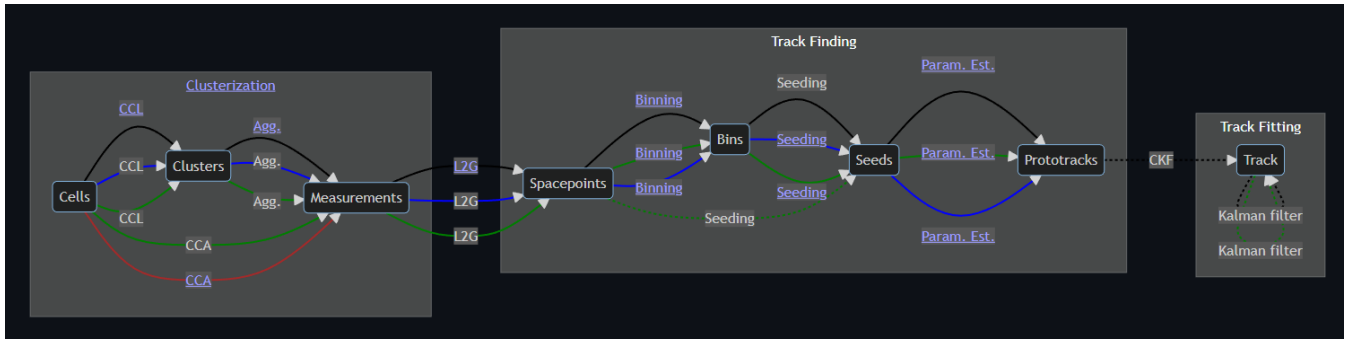
- Profiling does impact program execution time
  - Identical profiling options should be used for code optimisation

Profiling traccc

Thanks to Yusuf Manjra for collecting the profiling data

Building and profiling **traccc**

- Build tools
  - OS: Ubuntu 22.04
  - CUDA 11.7 and g++11
- Build options
  - **vecmem** – built from source with CUDA on
  - **detray** – built from source with CUDA on
  - **traccc** – build examples on
- Standalone executable profiled
  - **traccc_seq_example_cuda**
- Data used
  - TrackML – ttbar_mu200, 10 events



Main traccc repository https://github.com/acts-project/traccc

Wall time - CPU vs GPU

traccc_seq_example_cuda on 10 ttbar_mu200 event files

CPU: Intel i5-10600 @ 4.10Ghz, GPU: Nvidia RTX 3090 24GBs, CUDA 11.7, Nvidia driver: 515.43.04, CC. 8.6

GPU				CPU		
Blocksize	**cl+sp (s)**	**seeding (s)**	**trk param est (s)**	**cl+sp (s)**	**seeding (s)**	**trk param est (s)**
32	0.103683	0.0426339	0.0750294	0.198575	1.37156	0.0323424
64	0.105967	0.0424915	0.0750092	0.1967582	1.37409	0.0324095
128	0.105238	0.0424042	0.074376	0.1961375	1.37117	0.0320192
256	0.11083	0.042504	0.0746592	0.1970764	1.37425	0.0324181
512	0.112619	0.0424735	0.0741184	0.1968369	1.36962	0.031985

- GPU is faster on clusterisation, space point formation and seeding
- Wall time may not give precise GPU kernel execution time

Performance summary

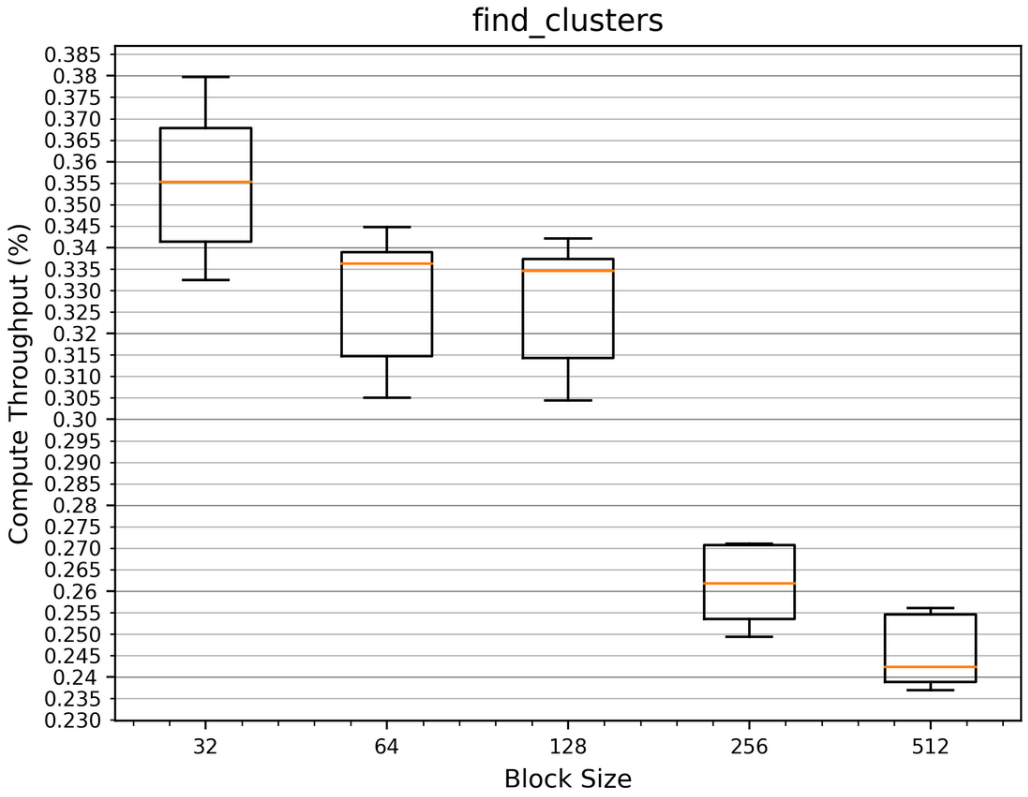
CPU: Intel i5-10600 @ 4.10Ghz, GPU: Nvidia RTX 3090 24GBs, CUDA 11.7, Nvidia driver: 515.43.04, CC. 8.6

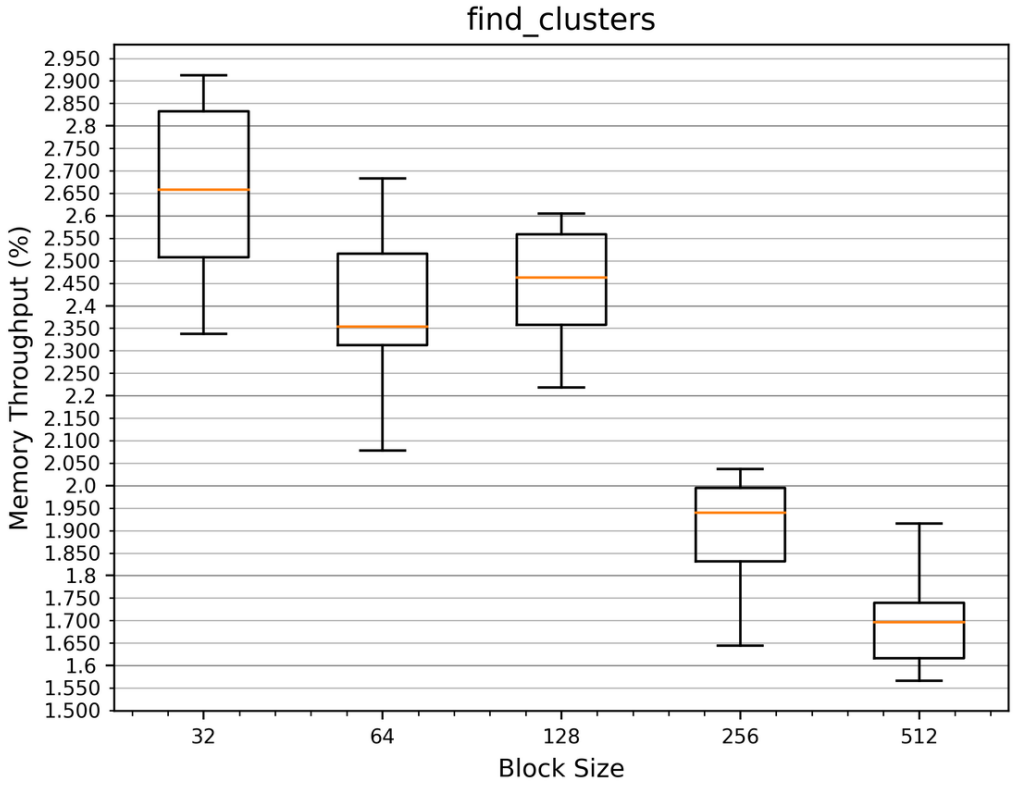
	64 THD/BLK					512 THD/BLK				
Function Name	**Cycles [cycle]**	**Duration [ms]**	**Compute Throughput [%]**	**Memory Throughput [%]**	**# Registers [register/ thread]**	**Cycles [cycle]**	**Duration [ms]**	**Compute Throughput [%]**	**Memory Throughput [%]**	**# Registers [register/ thread]**
find_clusters	3,895,811	2.79	0.31	2.35	48	5,127,261	3.72	0.24	1.72	48
count_cluster_cells	26,657	0.02	8.29	41.49	16	23,836	0.02	9.27	46.46	16
connect_components	690,531	0.5	0.78	8.54	32	669,326	0.48	0.81	20.62	32
create_measurements	752,744	0.54	2.76	19.11	70	675,834	0.51	3.08	15.95	70
form_spacepoints	35,335	0.03	8.39	39.41	38	40,254	0.03	7.36	37.96	38
count_grid_capacities	19,986	0.02	11.52	40.69	32	20,221	0.02	11.38	40.33	32
populate_grid	30,586	0.02	8.48	39.53	34	30,812	0.02	8.42	39.25	34
count_doublets	353,799	0.26	25	12.07	40	359,410	0.26	24.63	11.85	40
find_doublets	1,488,936	1.07	11.82	27.7	104	1,491,983	1.07	11.8	27.66	104
count_triplets	223,972	0.16	42.39	20.53	45	221,584	0.16	42.7	20.69	45
find_triplets	124,189	0.09	6.29	6.84	82	126,959	0.09	6.13	6.72	82
update_triplet_weights	10,552	0.01	5.34	25.08	32	10,431	0.01	5.39	25.32	32
select_seeds	142,334	0.1	14.14	14.14	74	142,188	0.1	14.24	14.24	74
estimate_track_params	35,710	0.03	3.89	49.83	47	35,298	0.03	3.94	50.33	47
Execution time		5.64					6.52			

- The track reconstruction chain excluding track fitting is executed in 14 CUDA kernels.
- Array prefix sum kernel calls were removed from the summary.

Block size adjustment – **find_clusters**

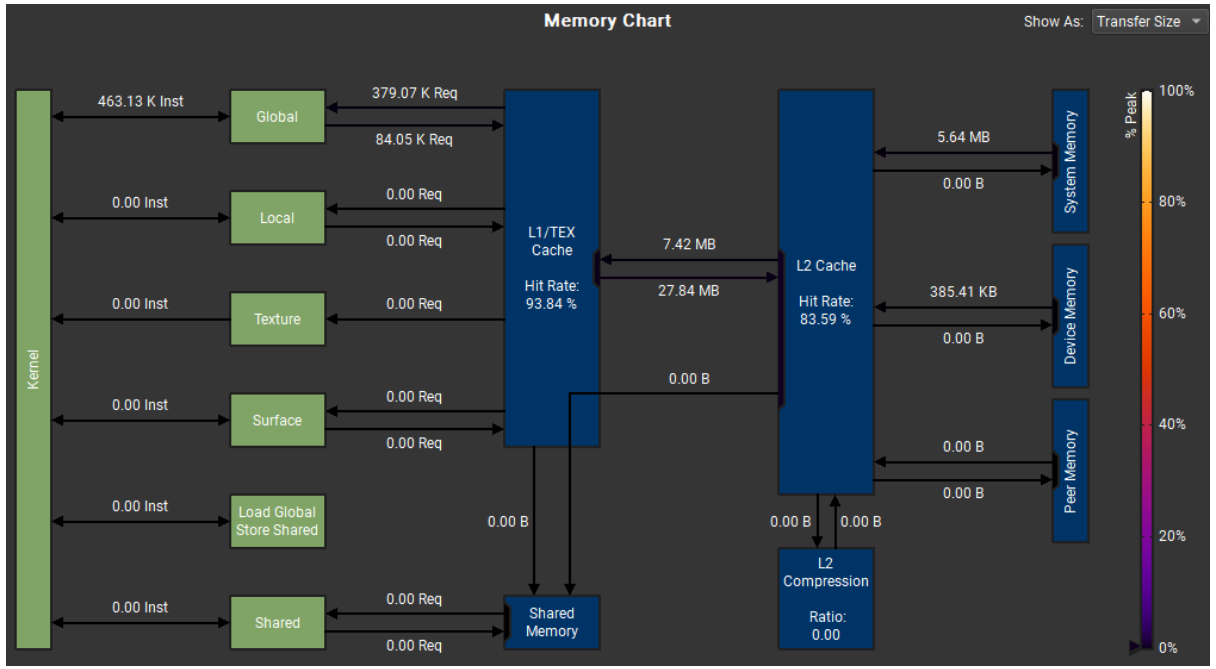
Increase thread-per-block may increase the # of active warps.



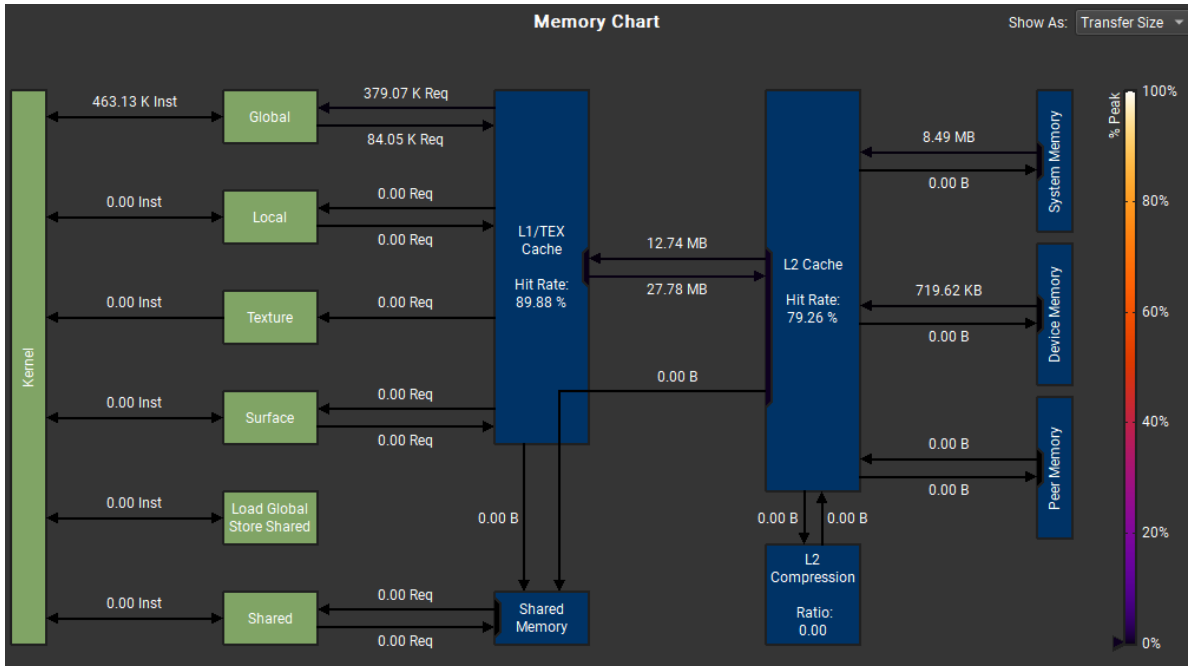


- 64thd/blk: 3,895,811 cycles (2.79ms)
- 512 thd/blk: 5,127,261 cycles (3.72ms)

Memory workload– find_clusters



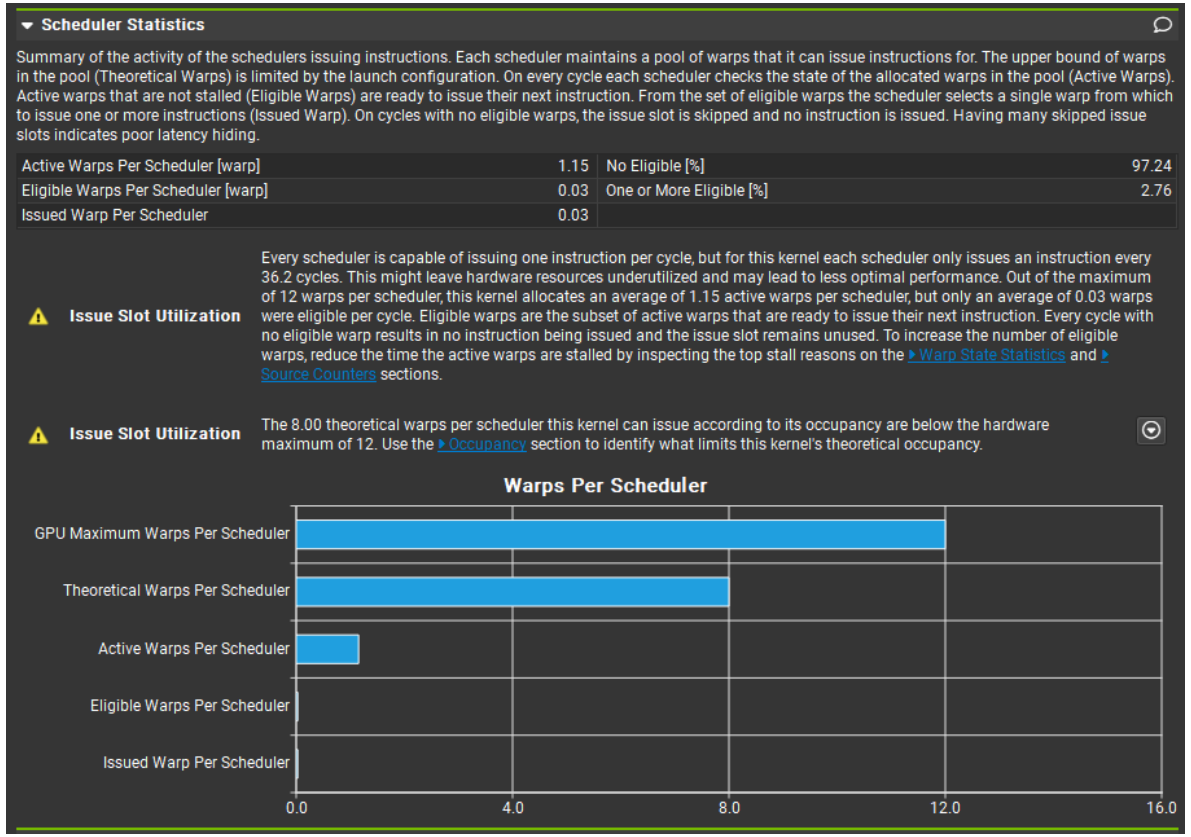
64 thd/blk



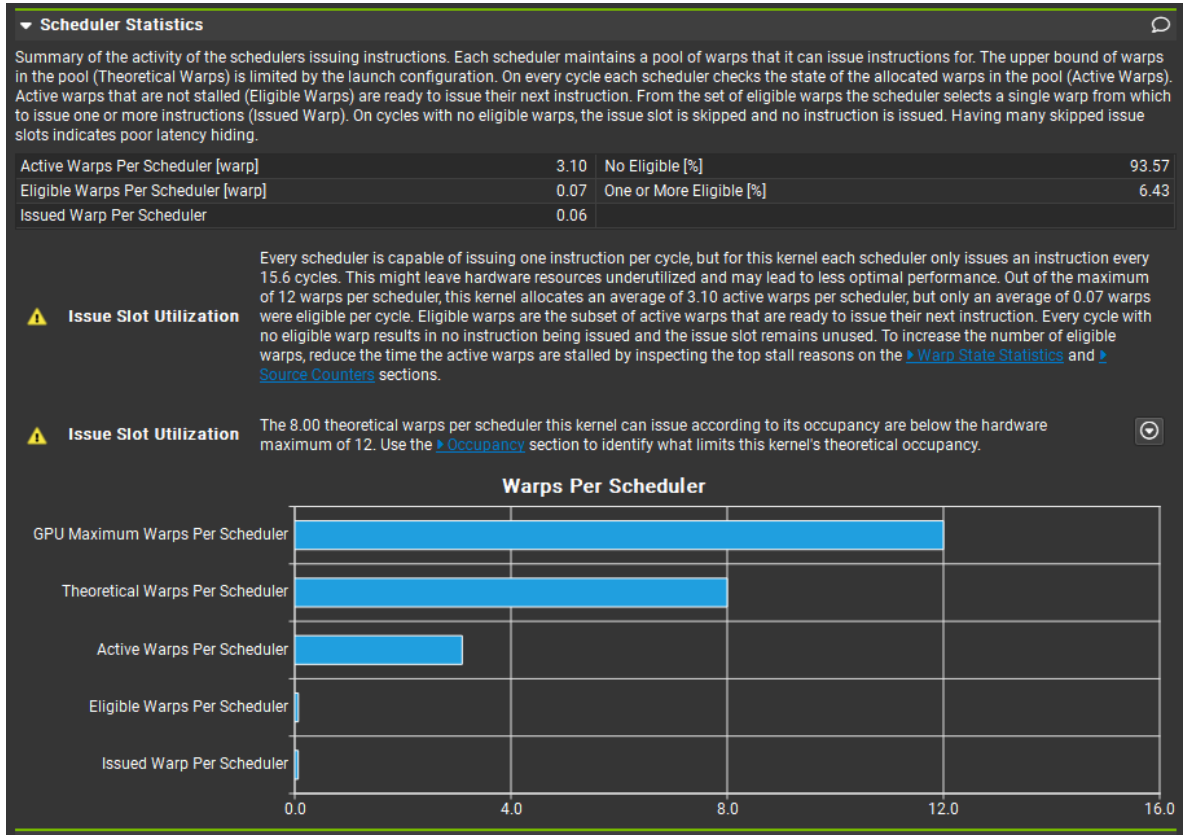
512 thd/blk

- L1 hit rate has been reduced
- 50% more data accessed in 512 thd/blk

Warp scheduler statistics – **find_clusters**



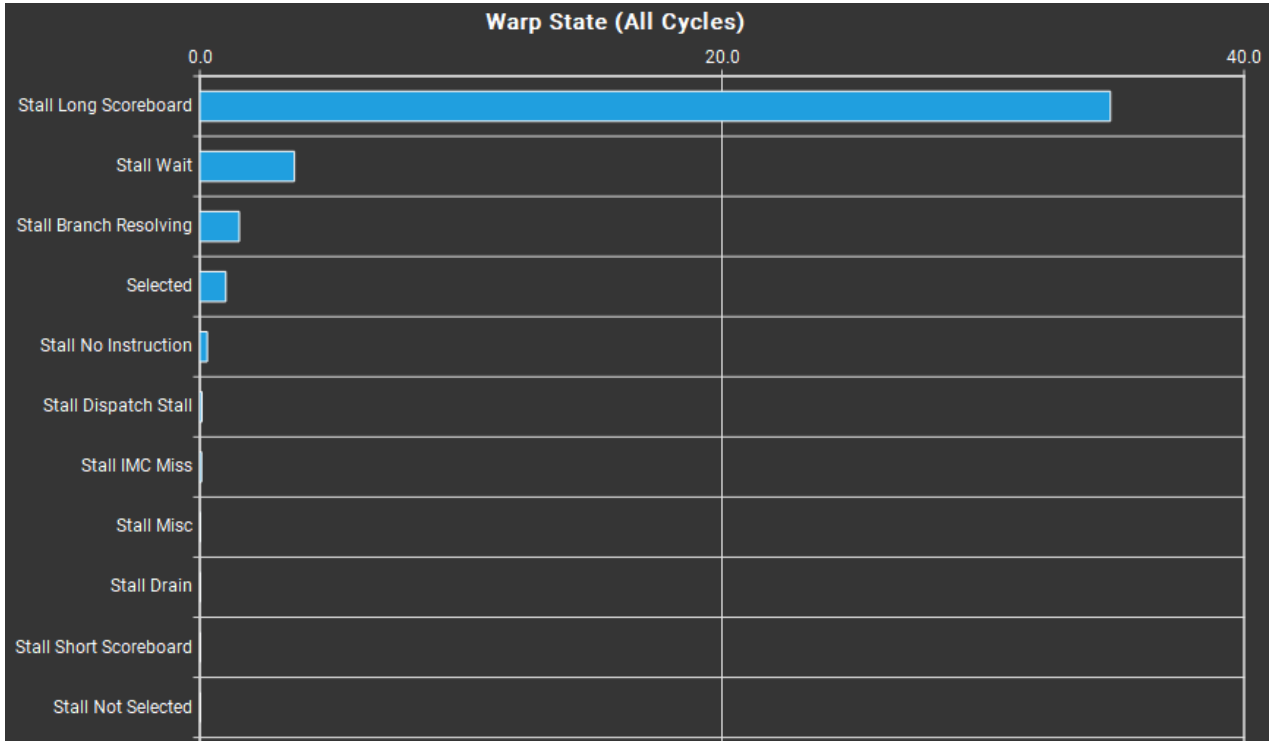
64 thd/blk



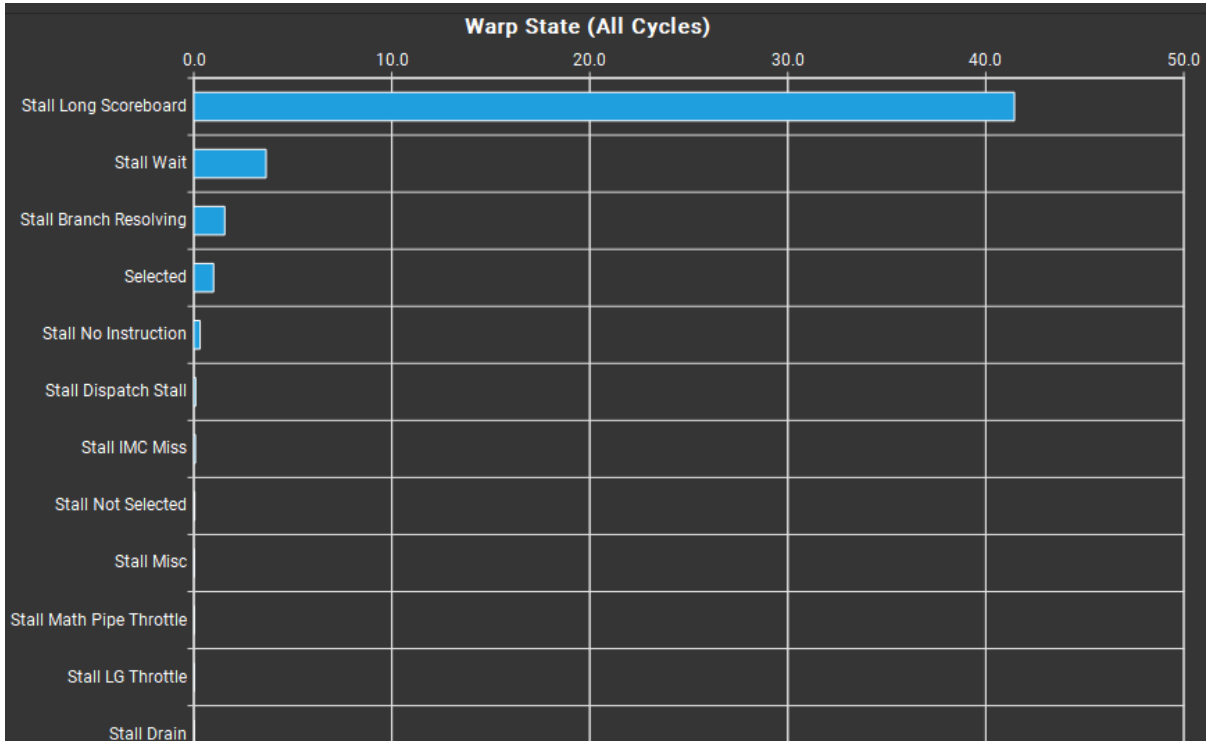
512 thd/blk

- Active Warps = Stalled + Eligible
- Stalled warps: waiting for all threads to be on the same instruction.

Warp state statistics – **find_clusters**



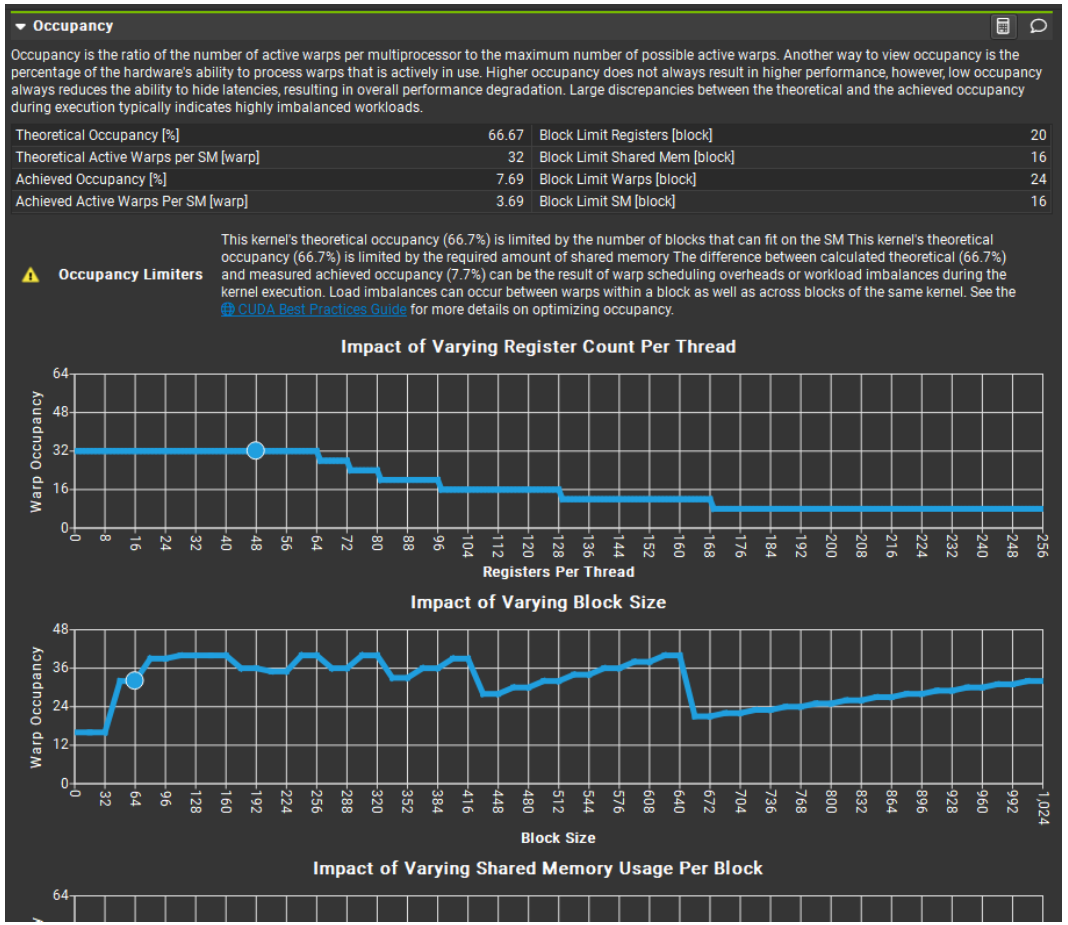
64 thd/blk



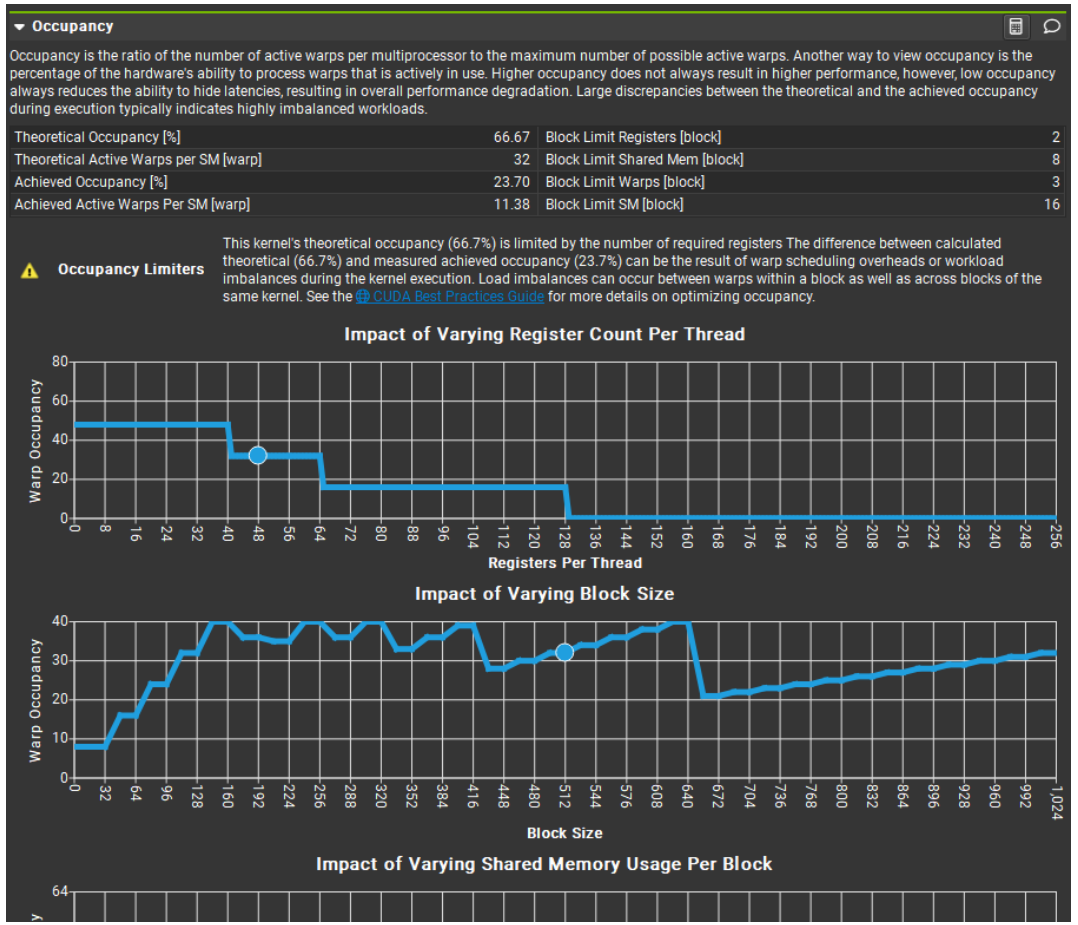
512 thd/blk

- Long scoreboard is related to L1 cache resolution for global memory access

Occupancy analysis – **find_clusters**



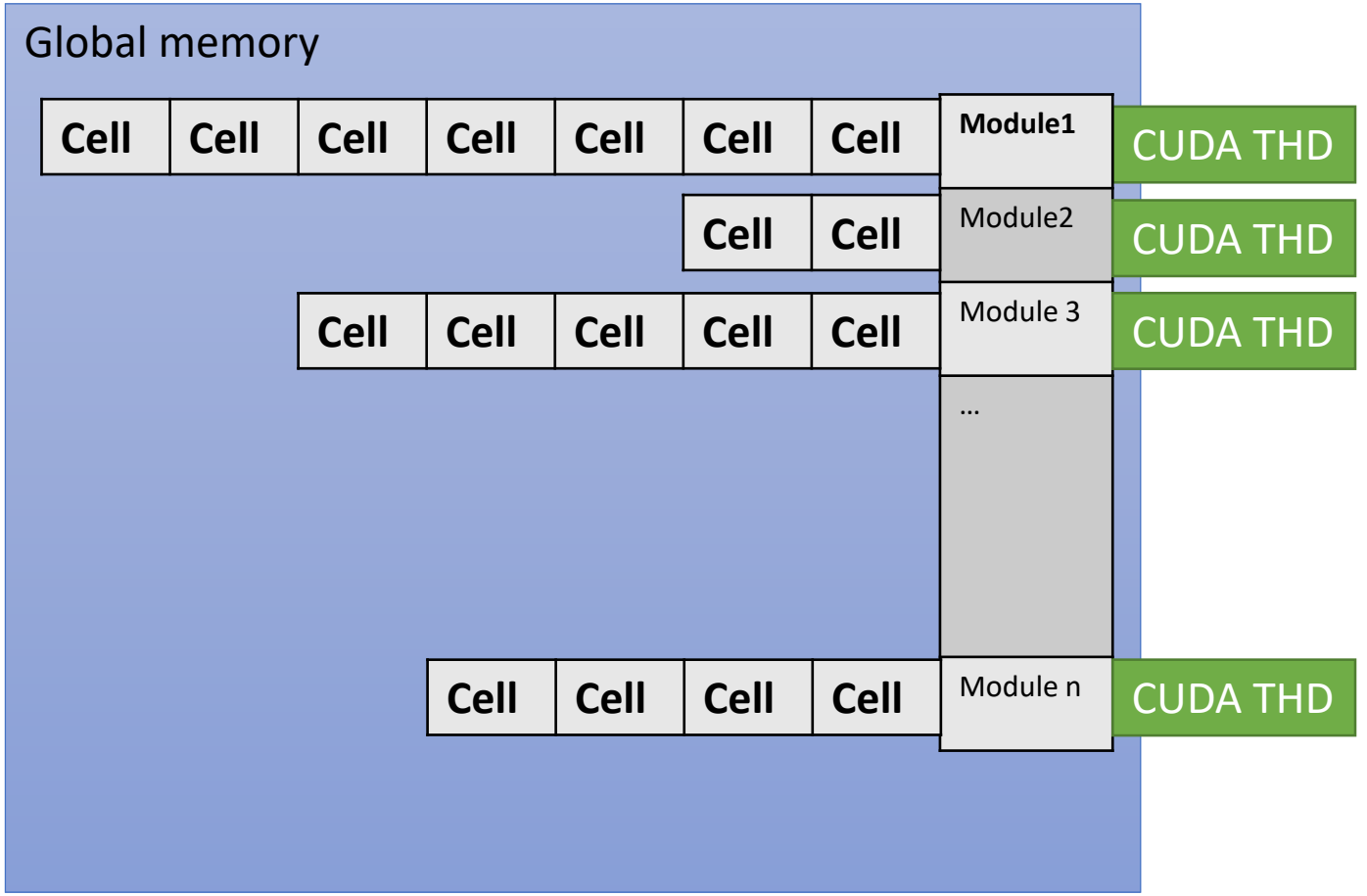
64 thd/blk



512 thd/blk

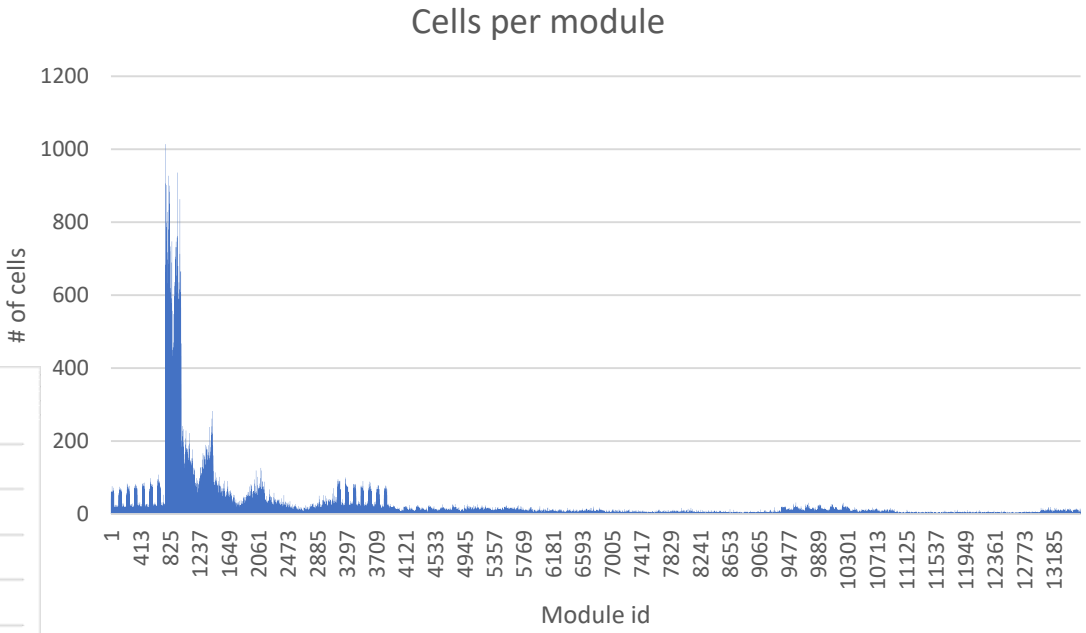
Existing parallelisation – **clusterisation**

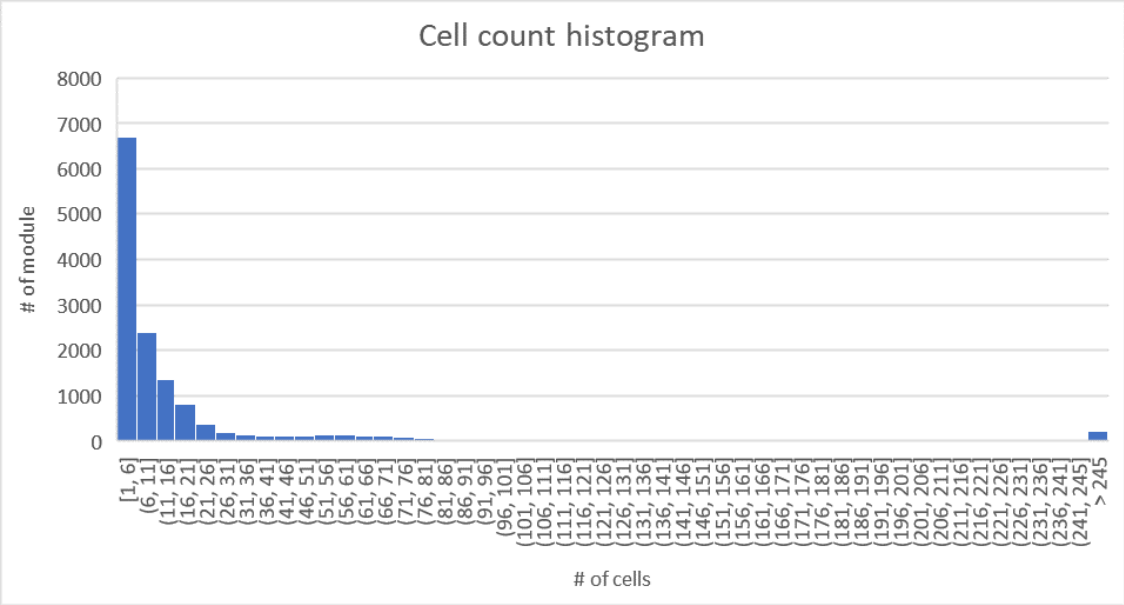
- One thread per module
- All data reside in global memory
- Jagged arrays are widely used
  - Some overhead
  - Reasonably optimised in vecmem



Clusterisation thread workload

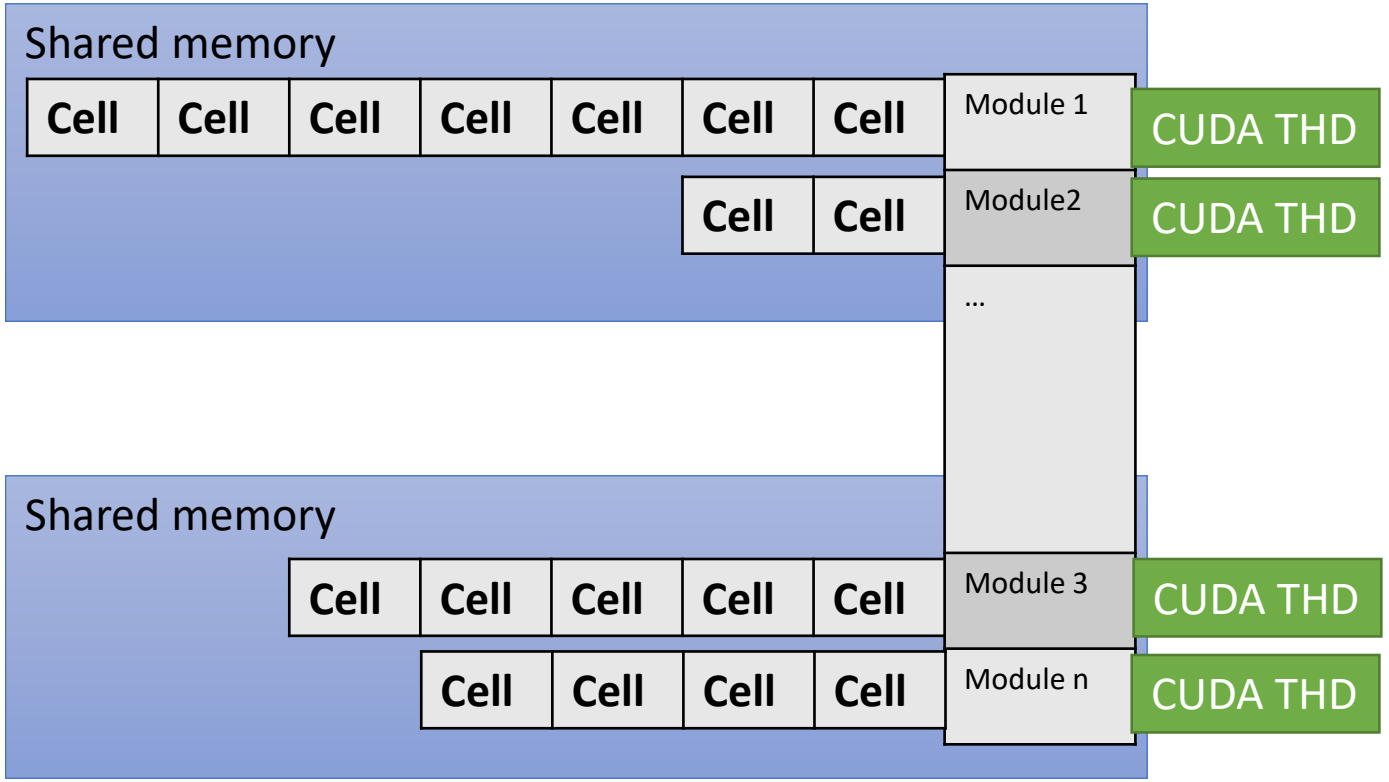
- TrackML ttbar_mu200 event data
  - ~330k cells
  - ~13565 modules
  - ~65 unique geometries
  - Most modules have very few cells
- sparse_ccl is an O(N^2) algorithm
- Parallelisation by module is not ideal





Parallelisation strategy discussion

- Reorganise modules to balance out block workload
  - Partition large modules
  - Merge smaller modules
- Store cells in shared memory
- Max. 163 KBs shared memory per block on A100, based on the current EDM
  - ~10000 cells (16B/cell)
  - ~6700 measurements (24B/meas)
  - ~ 4500 spacepoints (36B/sp)



Summary

- GPU works on a *Single Instruction Multiple Threads*(SIMT) principle
  - Prioritise on waiting for more eligible threads
  - Latency is necessary but should be minimised
- Current profiling results commonly point to deficiencies in memory access
  - Data layout should follow access pattern
  - Use shared memory where practical
- Possible to tune GPU performance without coding
  - by adjusting block size, register per thread, shared memory size
- Parallelisation strategy matters
  - What should each thread do? (e.g. partition based on input data, or detector geometries)
  - Consider compute and memory activity

Ongoing work

- Instrumentation based profiling
  - Use NVTX to look at isolated code region (https://github.com/NVIDIA/NVTX)
- Profile against high data workload
  - Use ATLAS specific geometries and data
- Analyse detector geometries and event data layout and access pattern on device memory
- Integrate and evaluate a CUDA implementation of FastSV CCA from traccc into the track reconstruction chain

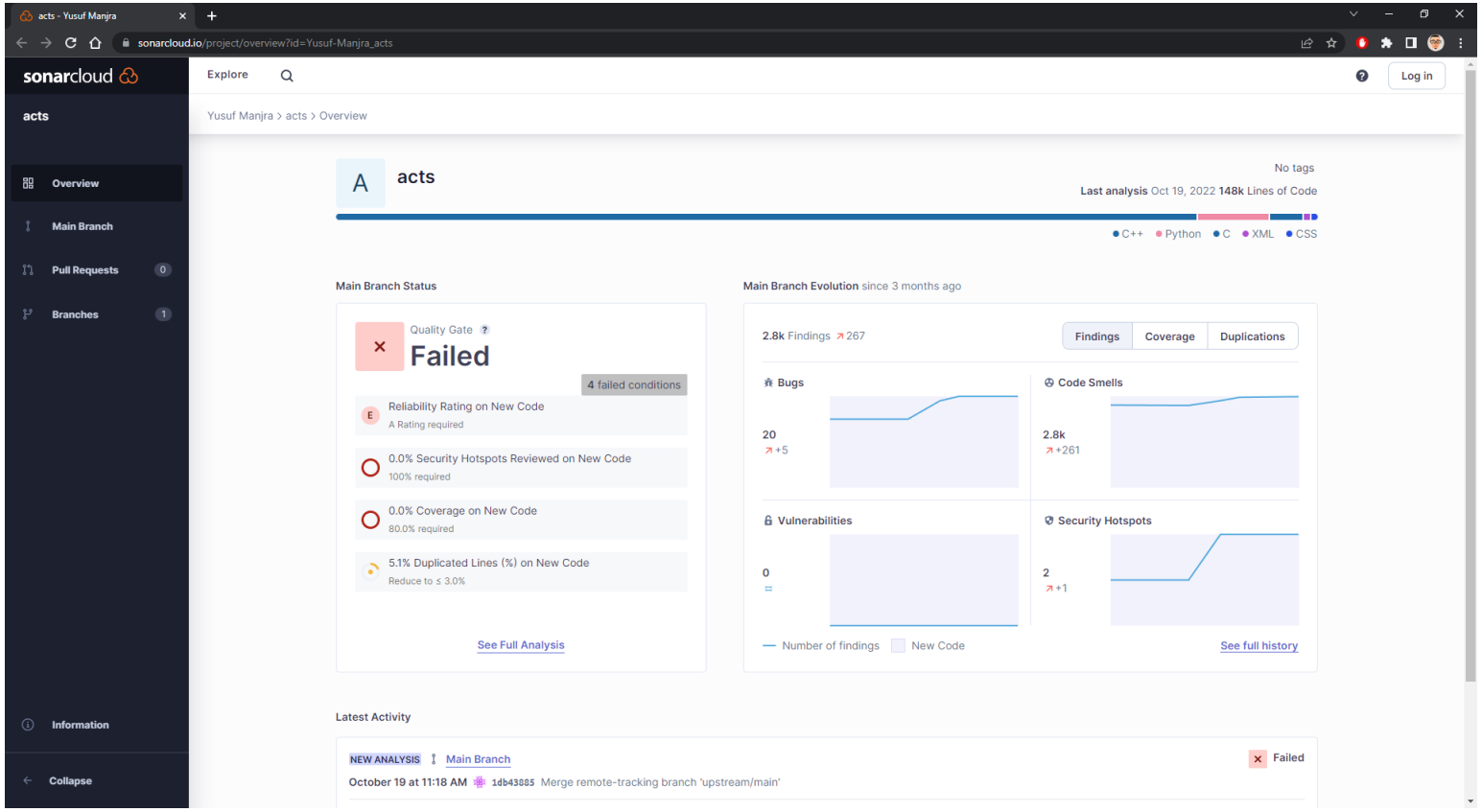
Questions?

Thank you for listening

Backup

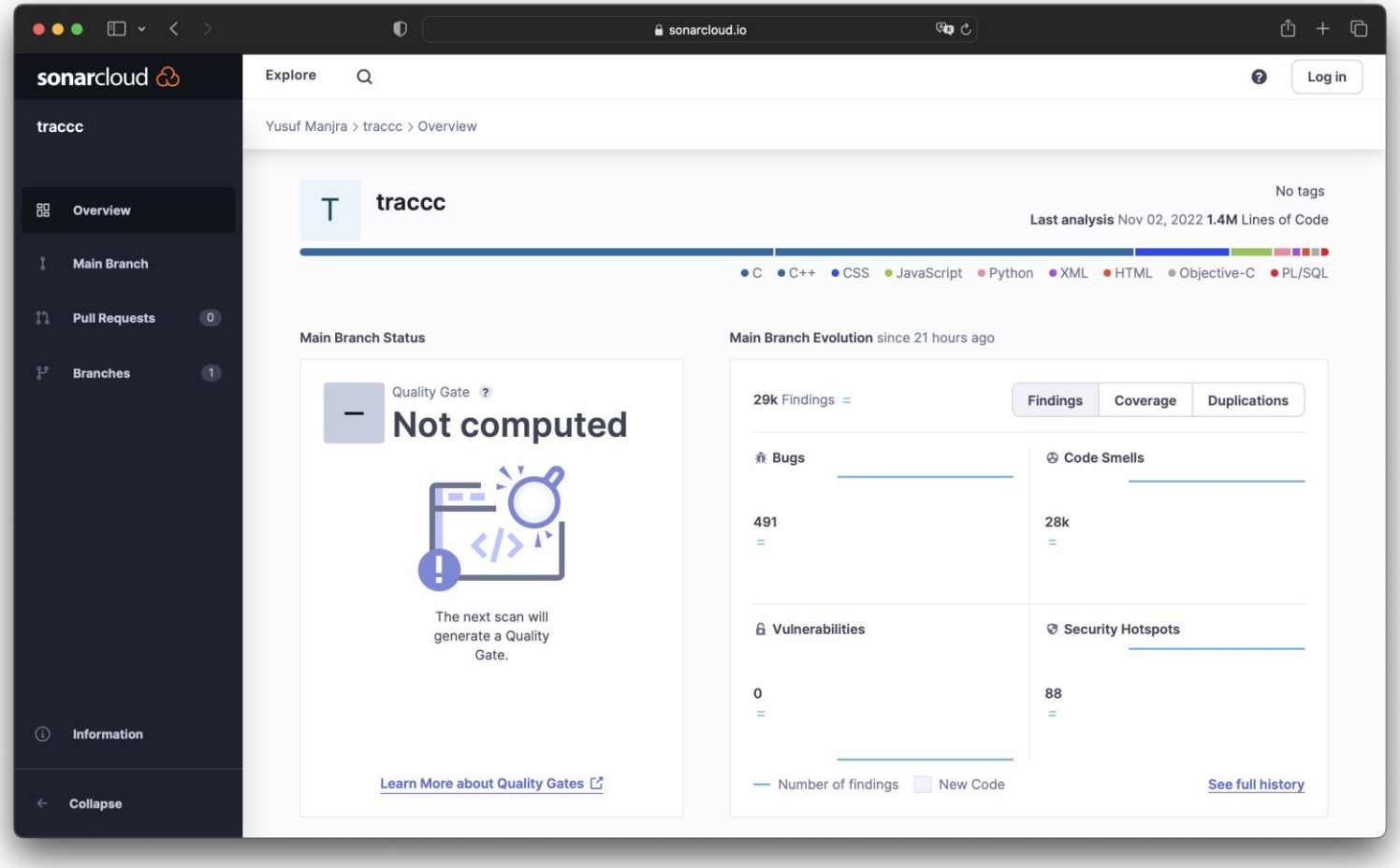
Static code analysis – Acts on sonarcloud

- We maintain a static code analysis portal for Acts using sonarcloud.
- Identifies a range of software sustainability metrics on architectural hotspots and code smells including
  - Complexity
  - Cognitive complexity
  - Duplication etc



Acts on sonarcloud: https://sonarcloud.io/project/overview?id=Yusuf-Manjra_acts

Static code analysis – traccc on sonarcloud



traccc on sonarcloud: https://sonarcloud.io/project/overview?id=Yusuf-Manjra_traccc

Parallelisation strategy matters

Ai, X., Mania, G., Gray, H.M. *et al* 2021. *Comput Softw Big Sci* **5,** 20

- Ai *et. al.* reported their Kalman Filter implantation on GPU
- Two parallelisation strategies were discussed
  - inter-tracks – one thread per track
  - intra-tracks – matrix operations on one track over multiple threads
- Performance gained by using two levels of parallelisation

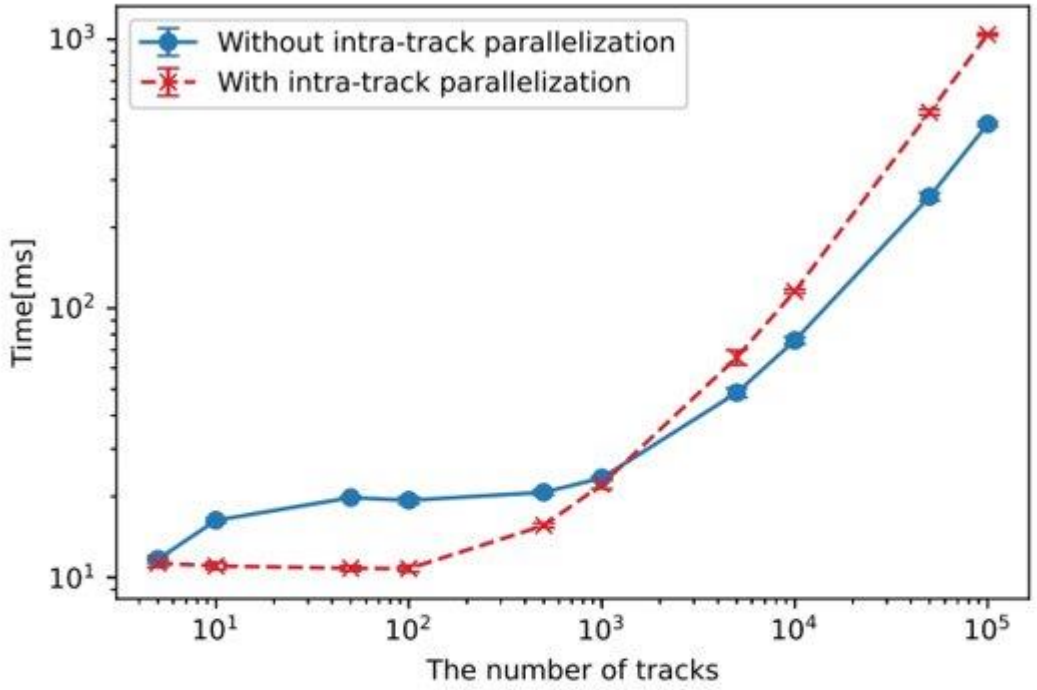
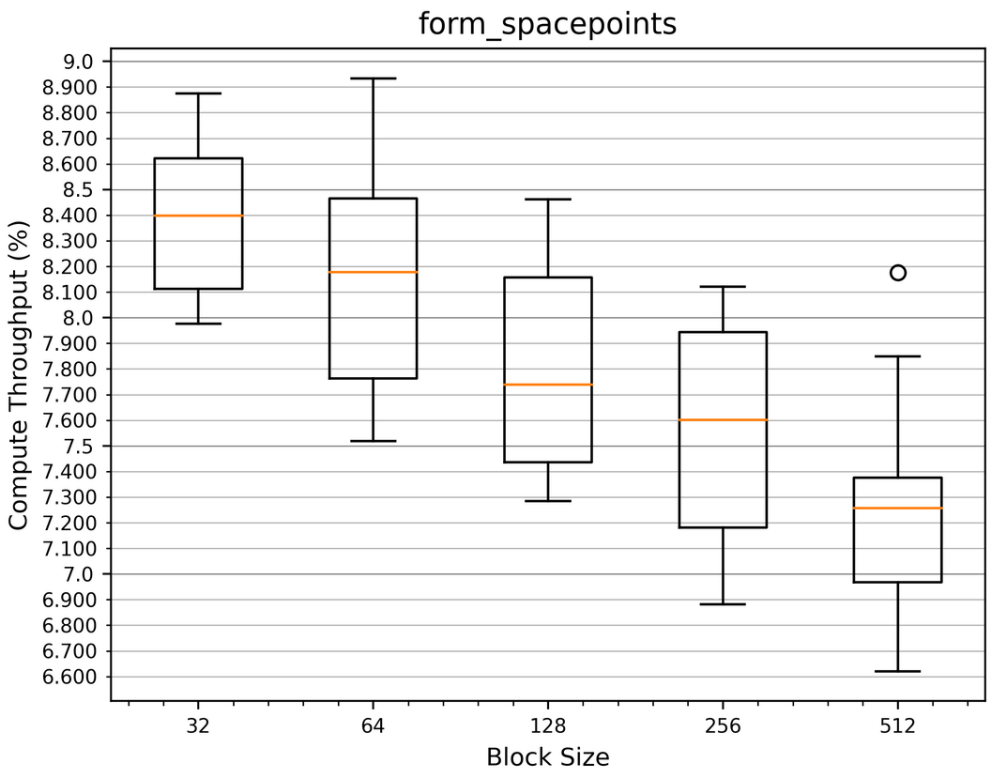
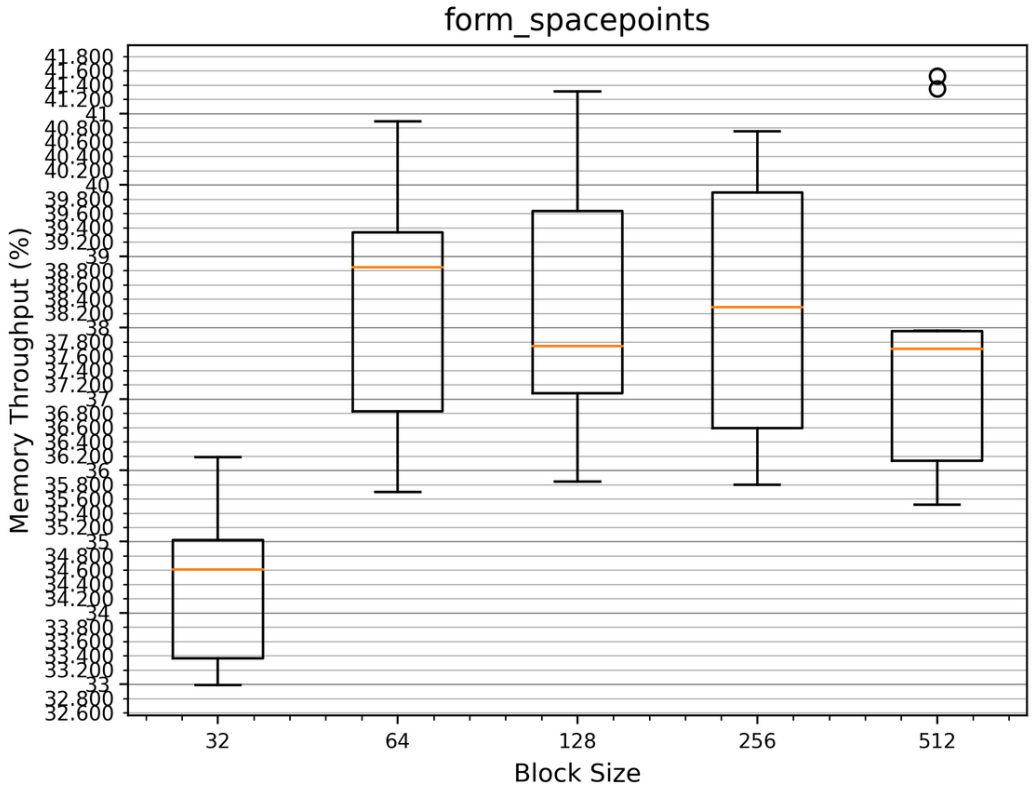


Fig. 11 The fitting time as a function of the number of tracks with linear grid of size 100,000×1 with (dashed red) or without (solid blue) intra-track parallelization on Cori-V100

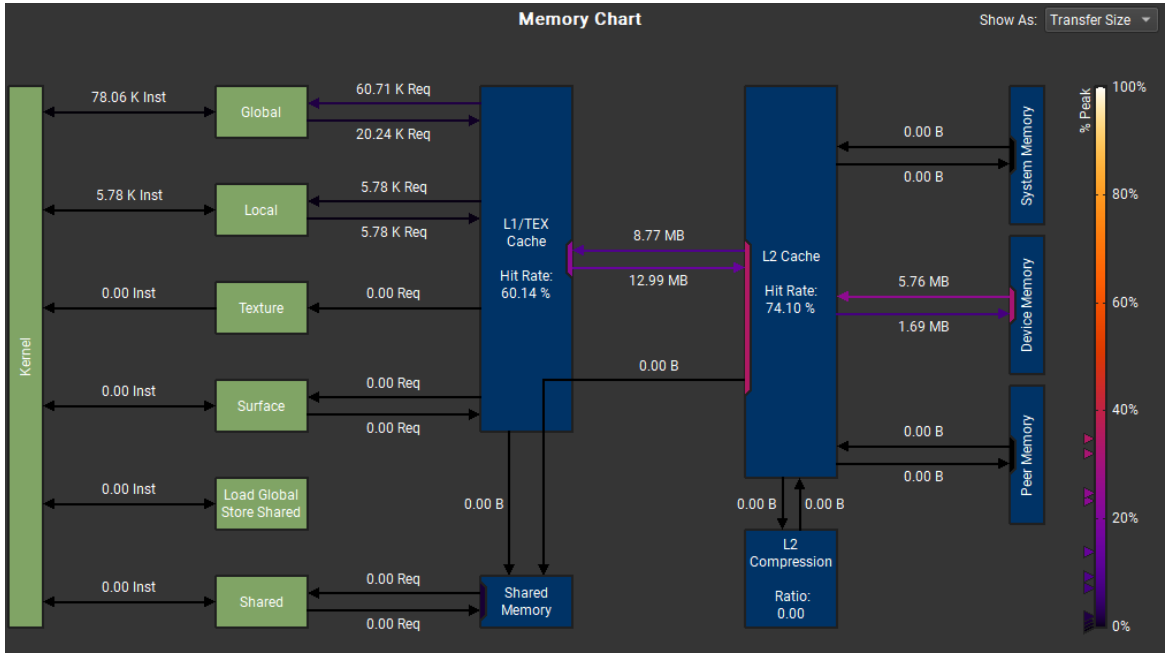
Block size adjustment – **form_spacepoints**



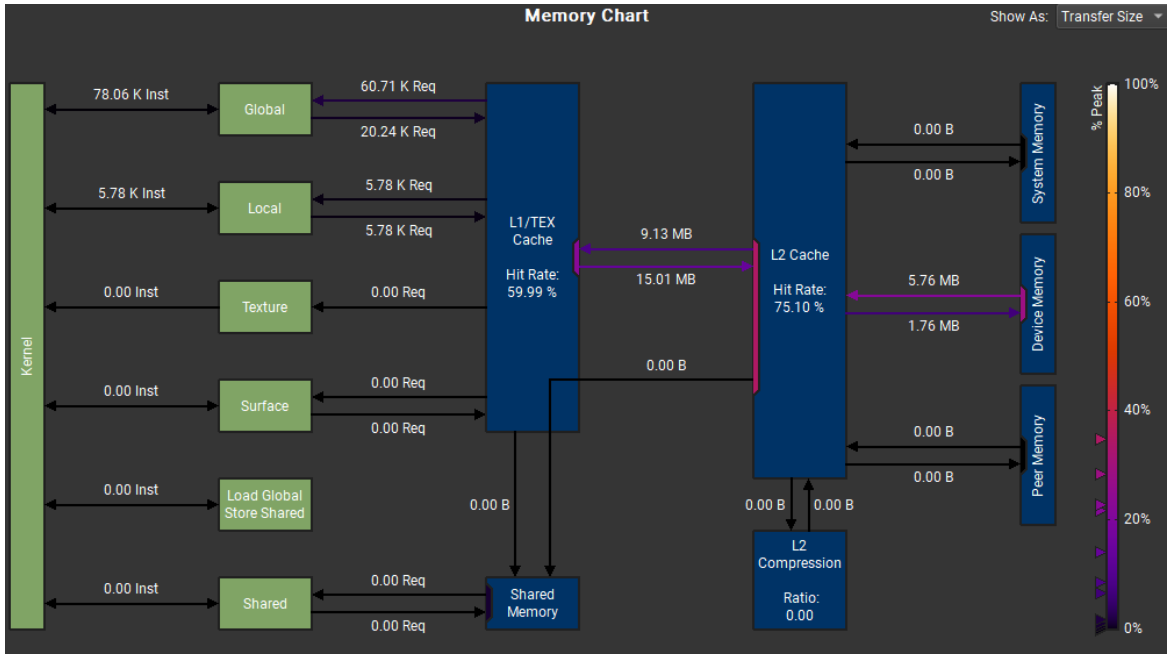


- 64thd/blk: 35,335 cycles (29.63us), 512 thd/blk: 40,254 cycles (31.97us)

Memory workload– form_spacepoints



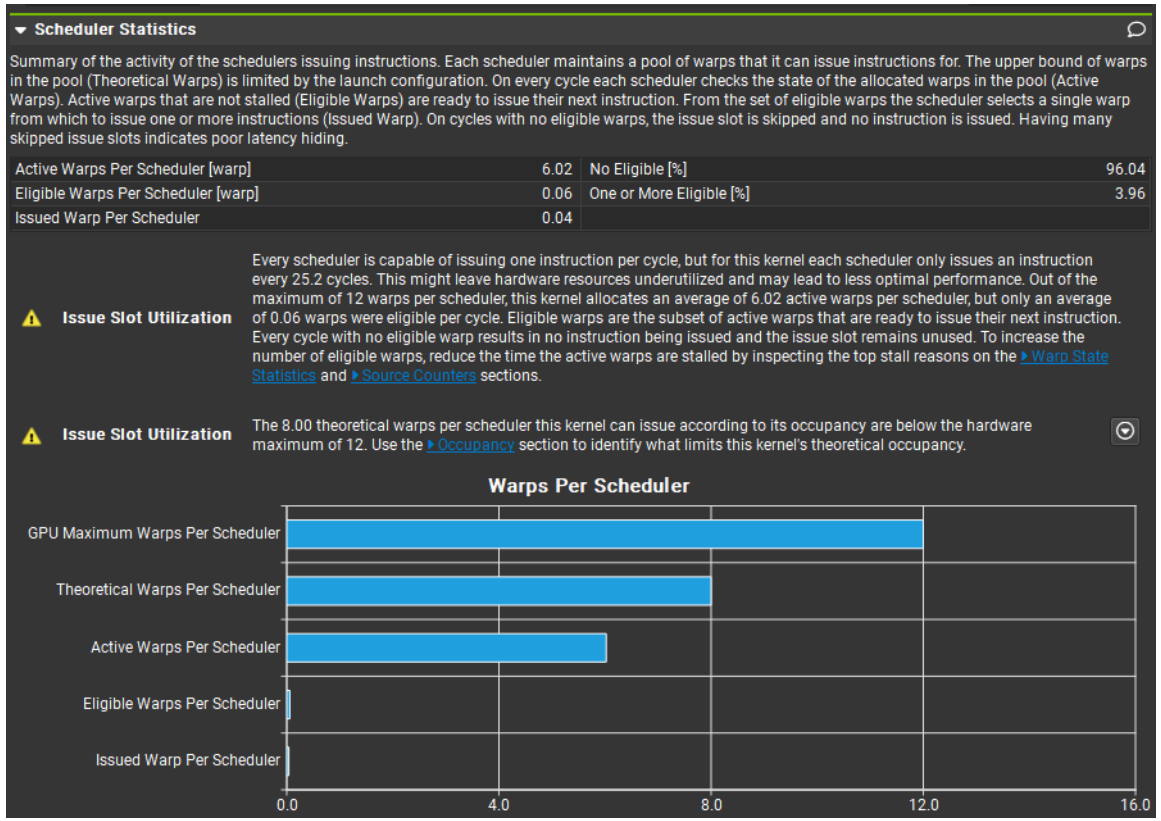
64 thd/blk



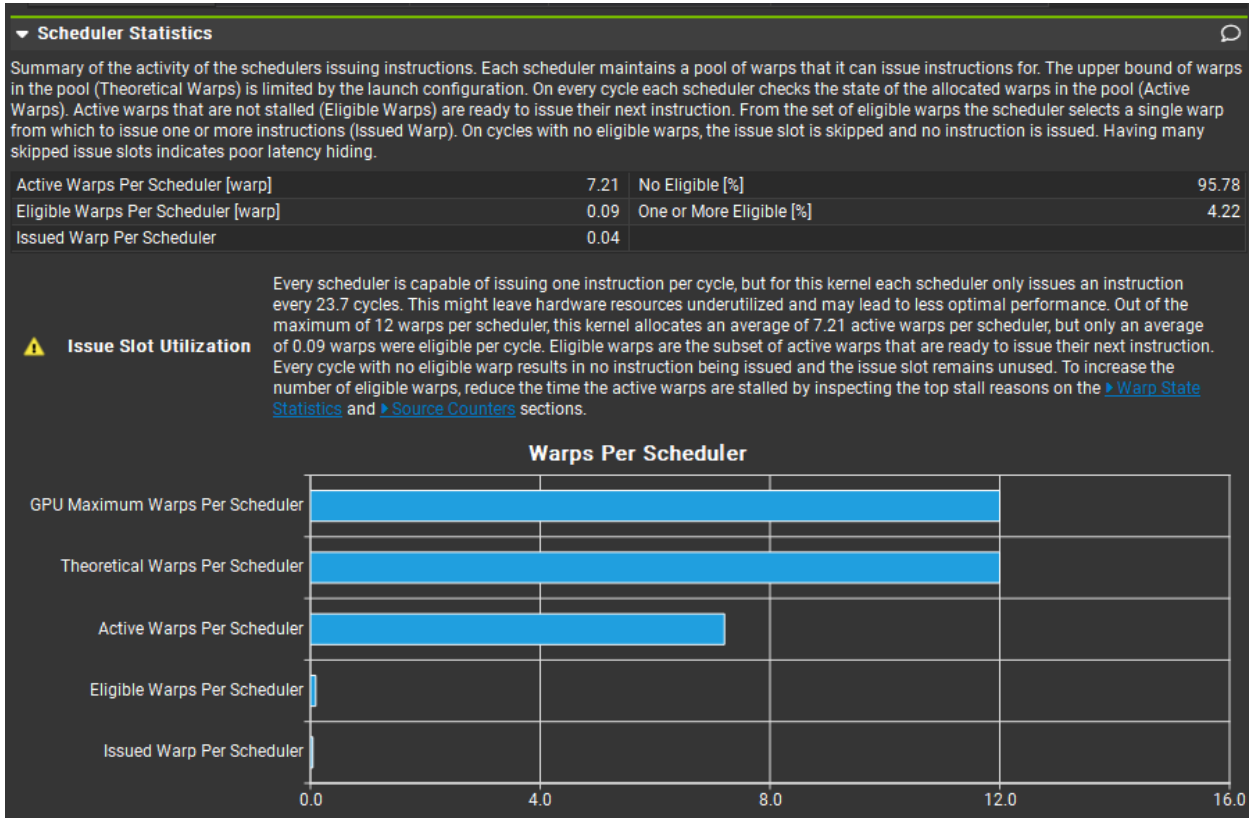
512 thd/blk

- Low L1 hit rate
- Little difference in memory workload

Warp scheduler statistics – **form_spacepoints**



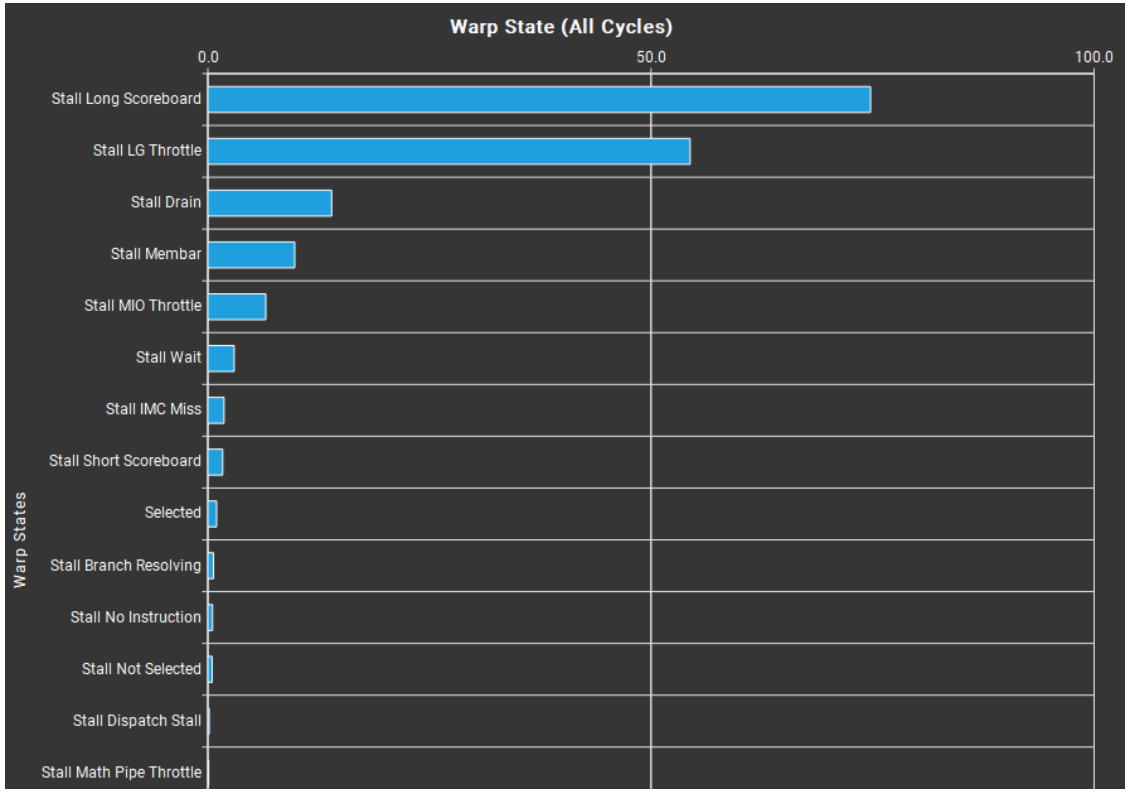
64 thd/blk



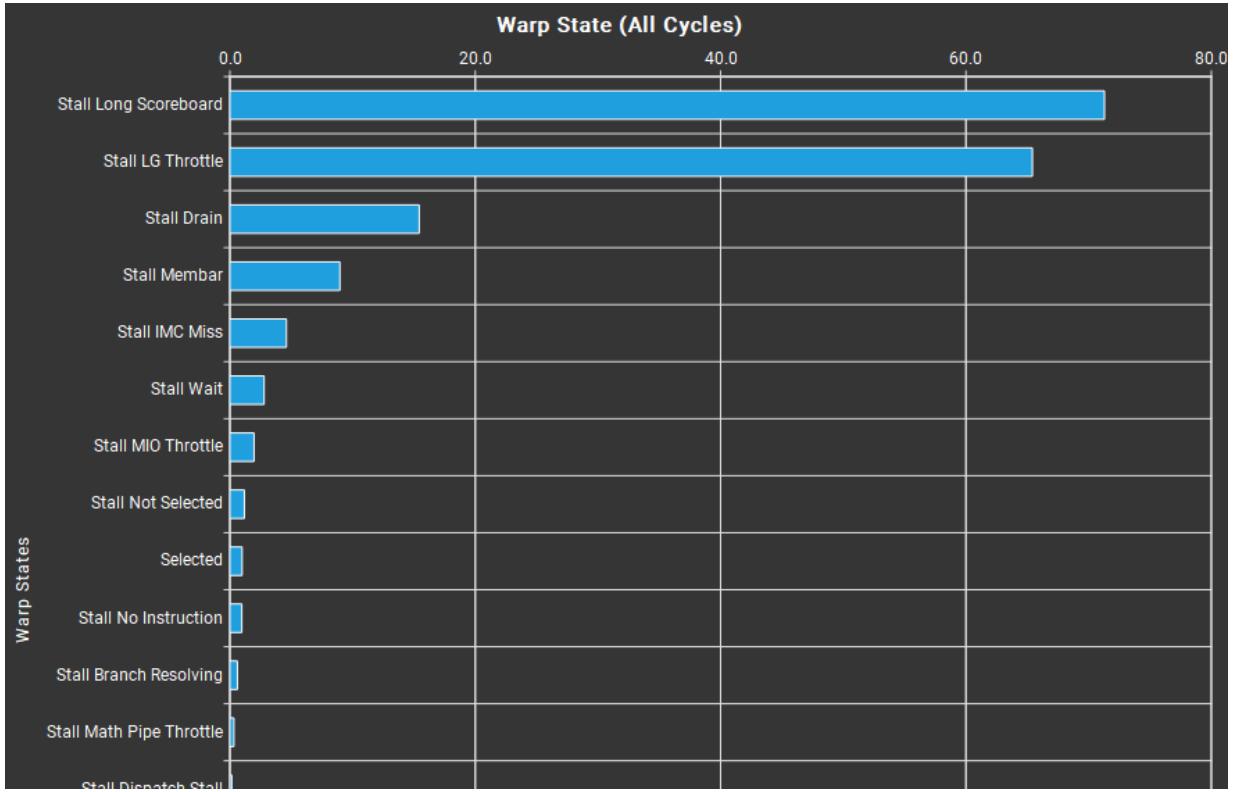
512 thd/blk

Theoretical warps/scheduler hits hardware limit @ 512 THD/BLK, but eligible warp count is still low.

Warp state statistics – form_spacepoints



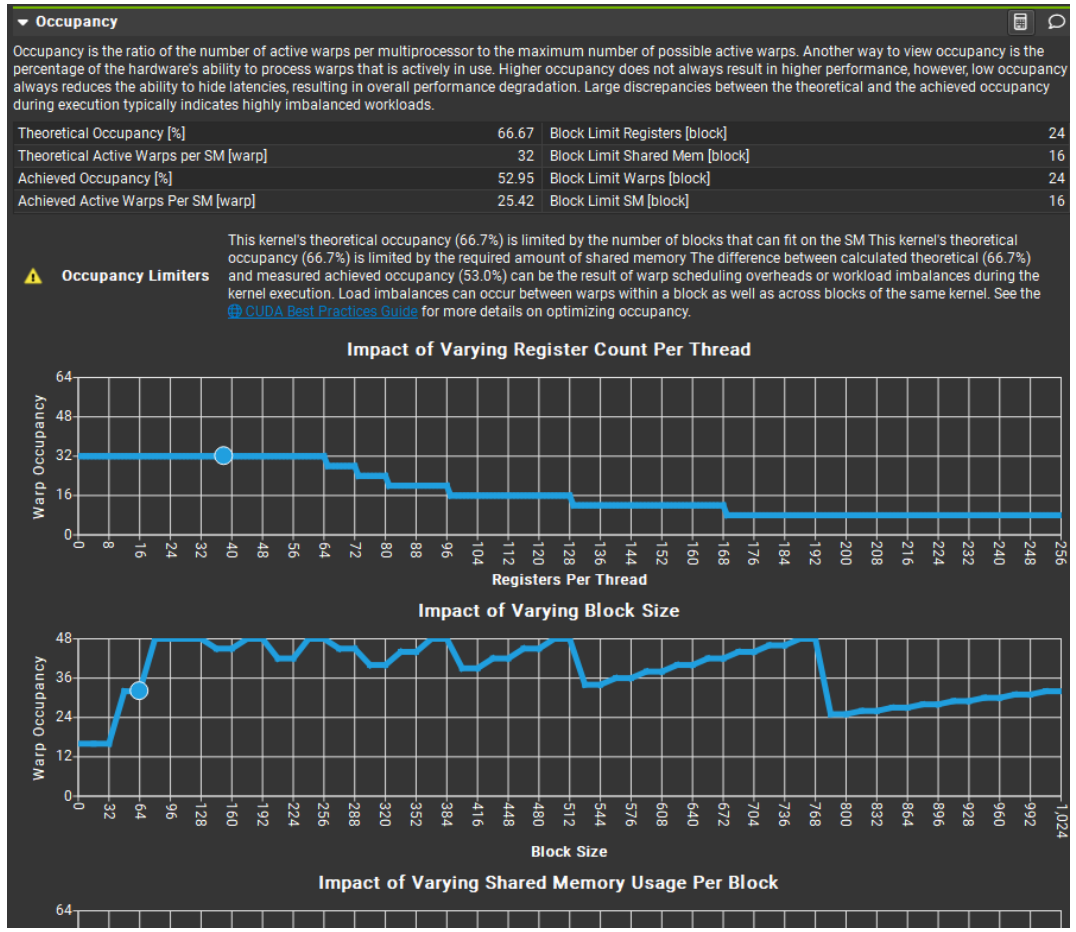
64 thd/blk



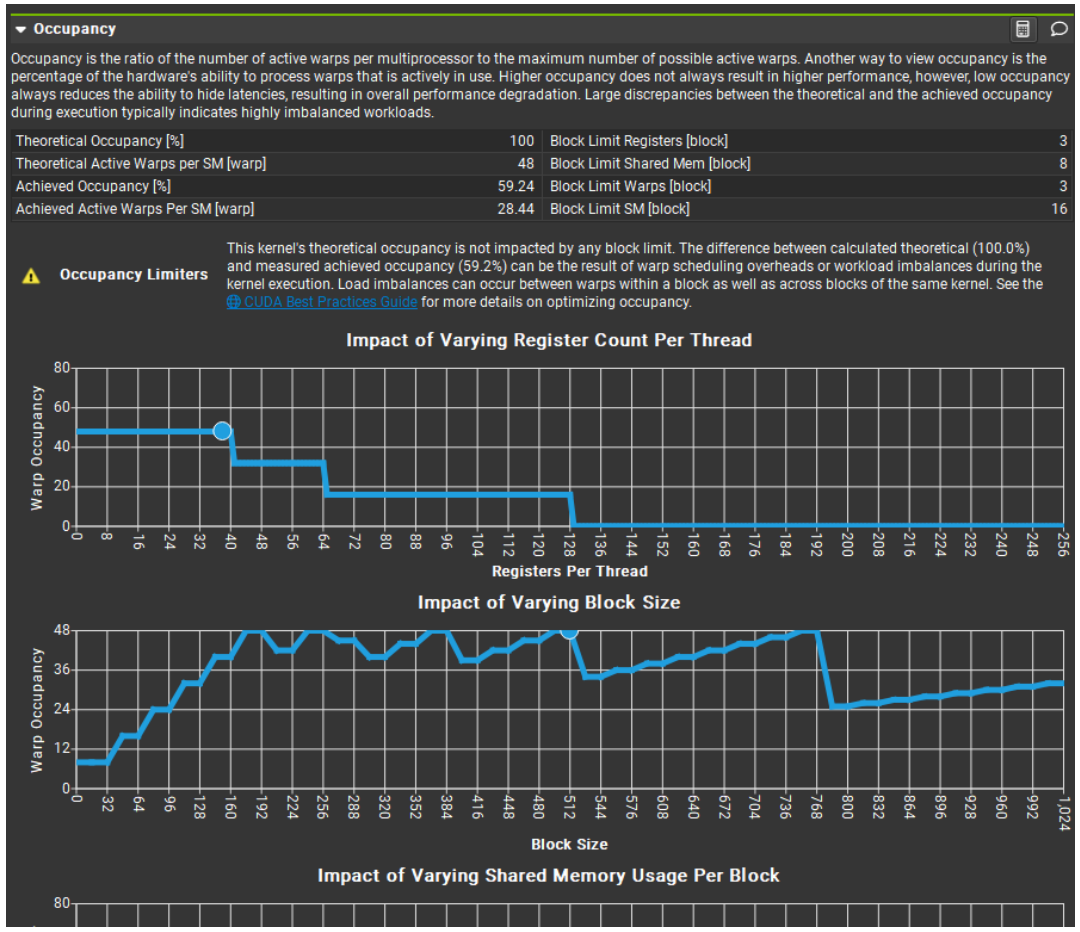
512 thd/blk

- LG Throttle caused by frequent local-global memory access
- More frequent LG activities as thread # increases

Occupancy analysis – form_spacepoints



64 thd/blk



512 thd/blk