ILLINOIS

ECE 498 ICC: IoT and Cognitive Computing
Professors D. Chen, W. Hwu, J. Xiong
Spring 2020

Lecture 23

Using Nsight Compute and Nsight Systems

Carl Pearson

pearson@Illinois.edu

Objective

- Nsight Toolsuite
  - Nsight Systems System-Level Profiling
  - Nsight Compute: Kernel-Level Profiling
- Not discussed
  - Other performance tools: gprof, perf, vtune, ...

Outline

- Introduction to Profiling
- Development Model and Profiling Strategy
- Preparing Application for Profiling
- Measuring time with CUDA Events
- Reminder / Introduction to Matrix Multiplication
- Nvidia Nsight Compute
- Nvidia Nsight Systems

Resources

- Everything used in this lecture is at github.com/cwpearson/nvidia-performance-tools

- Use it any way you want (with attribution).
  - Docker images for amd64 and ppc64le with CUDA and recent versions of Nsight
  - Matrix multiplication examples (in `sgemm/`)
  - rai_build.yml for sgemm (in `sgemm/`) if you have access to rai
  - Build and profile the examples on any system

- Profile data files and source code can be downloaded here:
- https://uofi.box.com/s/kh7plqoxqyy2yp54abokhqdtopga0oxh

System- and Kernel-Level Profiling

- Nsight Compute: Kernel-Level Profiling
  - How fast does the GPU execute my kernel?

- Nsight Systems: System-level Profiling
  - How effectively is my system delivering work to the GPU?
  - What is my system doing while the GPU is working?
  - How fast is data moving to/from the GPU?
  - How much time does the CPU take to control the GPU?
  - When do asynchronous operations occur?

Common GPU Development Model

Client

control: ssh

Data: scp, network file system, …

Target Platform
(GPUs of interest)

Record-and-analyze

Client

Target Platform
(GPUs of interest)

2

Transfer recorded files

3

Analyze with GUI

```
nsight-sys ...
nv-nsight-cu ...
```

1

Record with CLI

```
nsys profile ...
nv-nsight-cu-cli ...
```

Preparing for Profiling: Correctness

- Subtle errors that do not cause your kernel to terminate under normal conditions can cause errors with profiling
  - esp. writing outside of allocated memory
- Run your code with cuda-memcheck if profiling crashes or misbehaves
  - Automatically instruments for bad memory behavior
  - Causes something like 100x slowdown, so try small datasets first
  - Fix any errors that come up, then profile again

```
cuda-memcheck ./my-cuda-binary
```

Preparing for Profiling: Host Code Annotations

Nvidia Tools Extensions

`#include <nvToolsExt.h>` and link with `-lnvToolsExt`

Will show up as a named span in the Nsight System GUI timeline

Useful for marking parts of the code for later reference.

```
nvtxRangePush(“sleeping”);
sleep(100);
nvtxRangePop();
```

Preparing for Profiling: Compiling

- Compile device code *with optimizations*
  - non-optimized or debug code often has many more memory references
  - nvcc by default applies many optimizations to device code
  - remove any -G flag (this flag generates debug info for device code)
- Compile device code with line number annotations
  - add `-lineinfo` flag to all `nvcc` calls
  - puts some info in the binary about what source file locations generated what machine code

```
$ nvcc -G main.cu
```

```
$ nvcc -lineinfo main.cu
```

Preparing for Profiling: Compiling

--generate-line-info / -lineinfo

Generate line-number information for device code.

Annotates the binary with information to correlate ptx back to CUDA source code

Compiled PTX

```
.loc 1 18 12 // file 1 line 18 col 12
cvta.to.global.u64 %rd1, %rd6;
mov.u32 %r27, %ctaid.x;
mov.u32 %r1, %ntid.x;
mov.u32 %r28, %tid.x;
mad.lo.s32 %r2, %r27, %r1, %r28;
```

CUDA Source Code

```
18: int gidx = blockDim.x *
19:   blockIdx.x + threadIdx.x;
```

Preparing for Profiling: Compiling

Don’t use any of these for Nsight profiling!

--profile / -pg
    Instrument generated code/executable for use by gprof (Linux only).
--debug / -g
    Generate debug information for host code.
--device-debug / -G
    Generate debug information for device code. Turns off all optimizations.
    Don't use for profiling; use -lineinfo instead.

Preparing for Profiling: System

Nsight System uses various system hooks to accomplish profiling.

Some errors would reduce the amount or accuracy of gathered info, some will make system profiling impossible. Consult the documentation for how to correct.

An example of a GOOD output: (check with `nsys status -e`)

```
$ nsys status -e
Sampling Environment Check
Linux Kernel Paranoid Level = 2: OK
Linux Distribution = Ubuntu
Linux Kernel Version = 4.16.15-41615: OK
Linux perf_event_open syscall available: OK
Sampling trigger event available: OK
Intel(c) Last Branch Record support: Available
Sampling Environment: OK
```

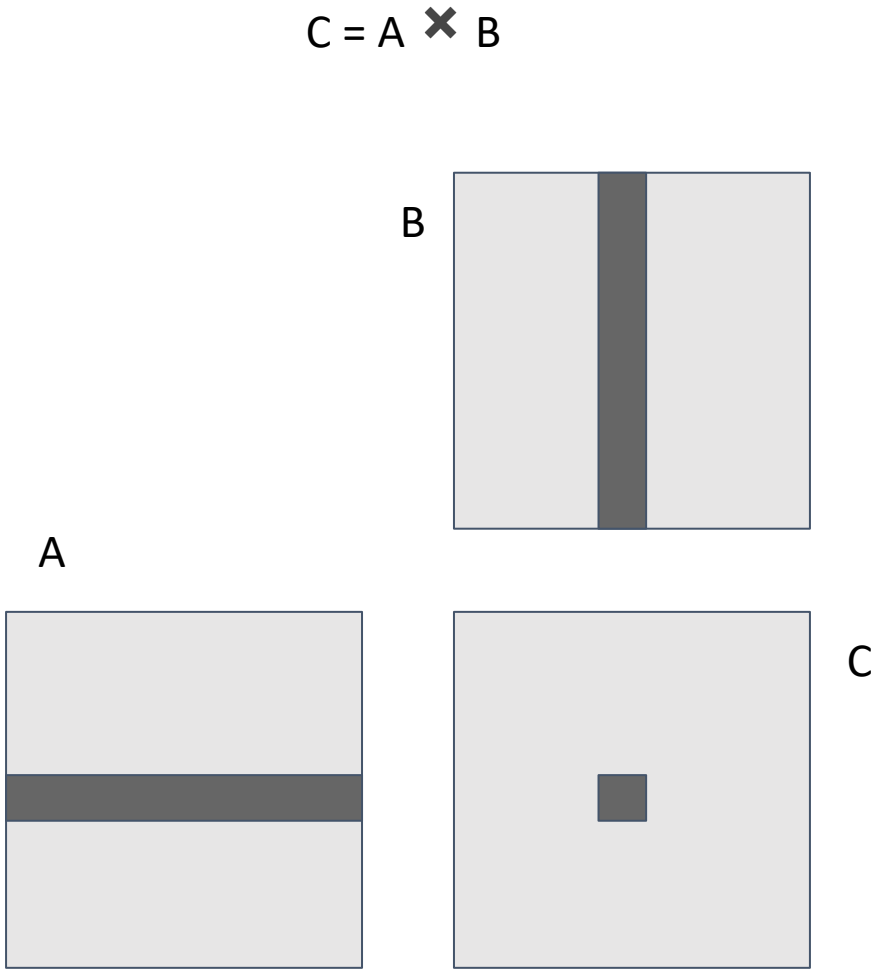
Caveats

Profiling affects the performance of your kernel!

It will help you improve the speed, but do not report the time *during* profiling as the performance of your code. Always run and time without profiling.

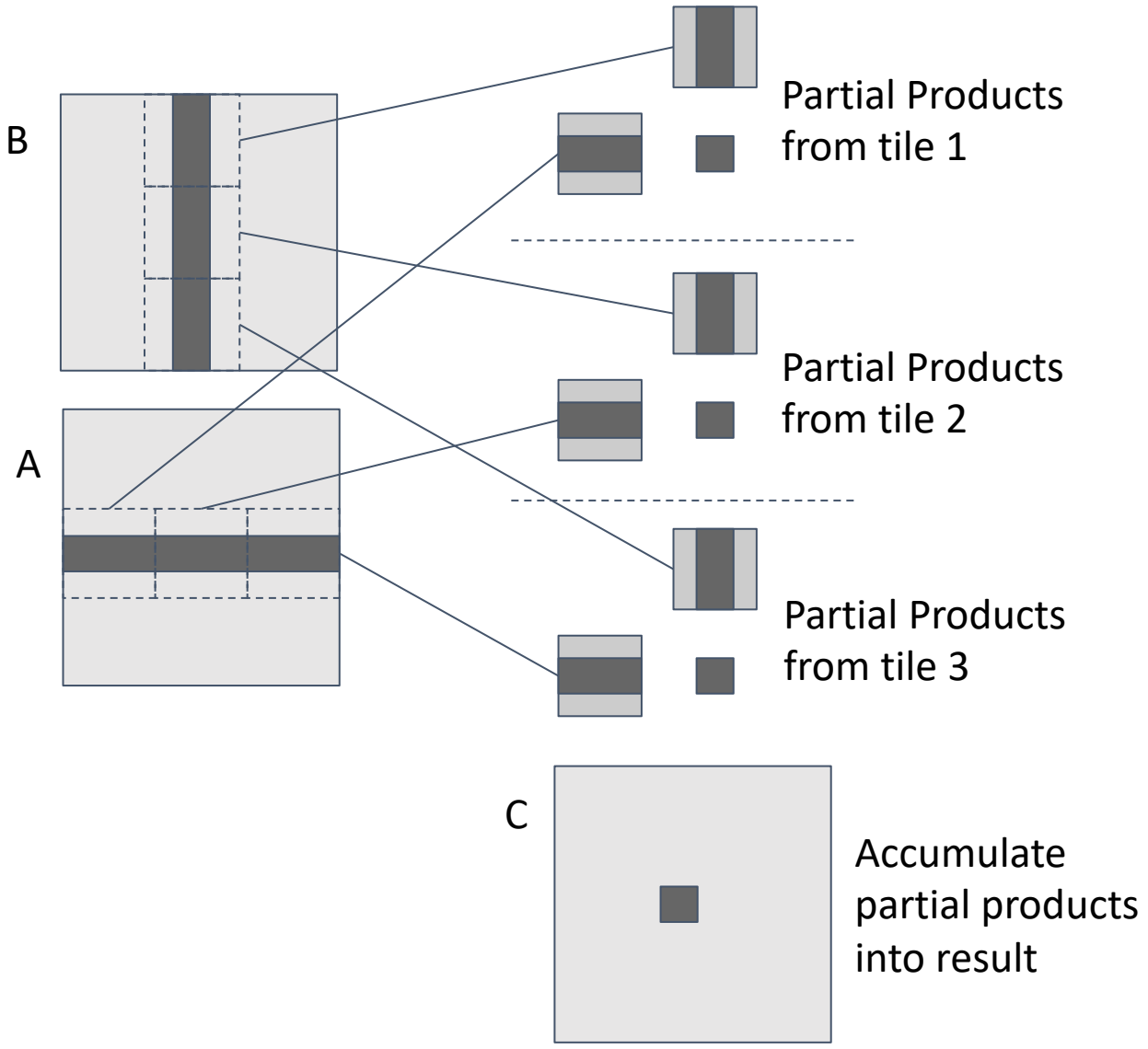
Matrix Multiplication Review

Dense Matrix Multiplication



- Each thread produces a single product value $C_{i,j}$ by dot(A_i, B_j)
- A and C are column-major, B is row-major
  - access to B is coalesced
- Each entry of the A/B matrices loaded from global memory multiple times

Shared-Memory Tiling



- Each thread produces a single product value $C_{i,j}$ by dot(A_i, B_j)
- Each thread block collaboratively loads tiles of A and B to accumulate partial products
  - Much reuse comes from fast shared memory instead of slow global memory

SGEMM Comparison

	A Reuse	B Reuse	Product Data per Block	Shared Memory/block
Basic	1	1	1024	0
Tiled	32 (TILE_SIZE)	32 (TILE_SIZE)	1024 ($TILE\_SIZE^2$)	32*32*2*4B = 8KB

Measuring Time with CUDA Events

Timing Operations

```
#include <chrono>
typedef std::chrono::high_resolution_clock Clock;
typedef std::chrono::duration<float> Duration;
cudaDeviceSynchronize();
auto wallStart = Clock::now();
kernel<<<...>>>(...);
cudaDeviceSynchronize();
Duration wallElapsed = Clock::now() - wallStart;
std::cout << wallElapsed.count() << " s";
```

Ensure the GPU is not doing other work.

Get wall time **before** and **after** the **things you want to measure.**

Be sure to synchronize device so asynchronous operations have finished.

- cudaDeviceSynchronize() may be relatively costly for fast operations
- Must synchronize with CPU
- CPU may be needed for other things during GPU execution

Terminology

- Stream (`cudaStream_t`)
  - A queue of sequential CUDA events. Each is executed after the prior one finishes
  - A program can use any number of CUDA streams
  - Associated with a device
- Default Stream (`cudaStream_t stream = 0`)
  - A special stream that is used when no stream is provided
- Event (`cudaEvent_t`)
  - Records the state of a stream
- See CUDA programing guide for stream synchronization edge cases
- *Generally*, to overlap operations:
  - different streams
  - do not use pageable memory
  - use *async CUDA runtime functions

Timing Async Operations with CUDA Events

```
cudaEvent_t start, stop;
cudaStream_t stream;
cudaStreamCreate(&stream);
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start, stream);
cudaMemcpyAsync(..., stream);
kernel<<<dimGrid, dimBlock, 0, stream>>>();
cudaEventRecord(stop, stream);
cudaMemcpyAsync(..., stream);
cudaEventSynchronize(stop);
float millis;
cudaEventElapsedTime(&millis, start, stop);
```

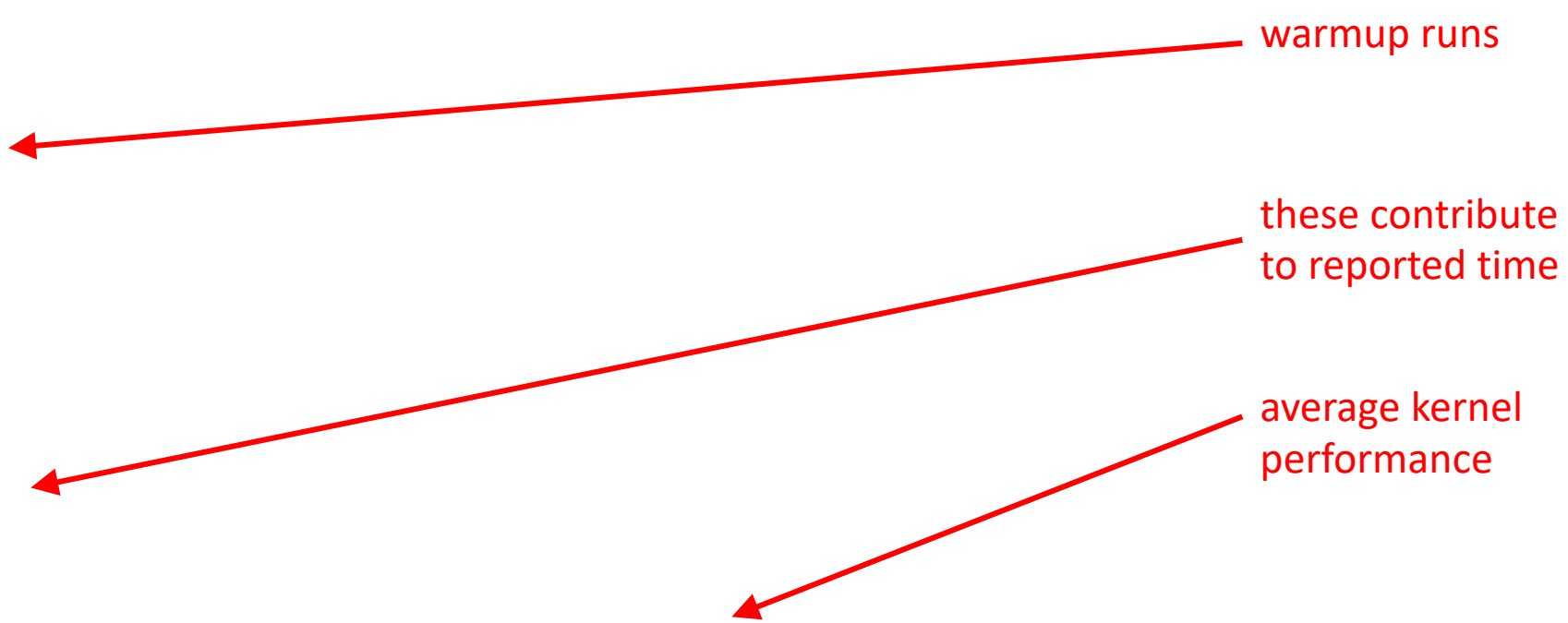
Place and events in the stream **before** and **after** the **things you want to measure.** Executed when the stream reaches the event, not when cudaEventRecord is called.

Wait for the final event to be reached. Could use cudaStreamSynchronize or cudaDeviceSynchronize too.

Get the time between the start and stop event.

Warmup & Multiple Measurements

```
* Running bash -c "./1-1-pinned-basic | tee 1-1-
pinned-basic.txt"
generate data
0: 3.75206
1: 3.73158
2: 3.73213
3: 3.73149
4: 3.73379
5: 3.73146 *
6: 3.73043 *
7: 3.72659 *
8: 3.73094 *
9: 3.72835 *
kernel 1787.02GFLOPS (6664784846 flop, 0.00372956s)
```



Kernel Profiling with Nsight Compute

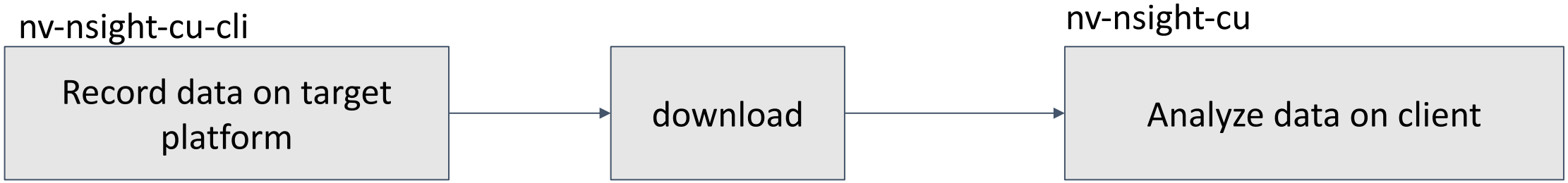
Problem Setup

- Dense matrix C = A * B
- C and A are column-major, B is row-major
- All matrices 1500 x 1500 floats
  - 9 MB each, 27 MB total (larger than L2 cache)

Kernel	Performance	Speedup
Basic	1787 GFLOPS	-
Tiled	2585 GFLOPS	1.45

Nvidia Nsight Compute

- Record and analyze detailed kernel performance metrics
- Two interfaces:
  - GUI (nv-nsight-cu)
  - CLI (nv-nsight-cu-cli)
- Directly consuming 1000 metrics is challenging, we use the GUI to help



Kernel Profiling

- Device has many performance counters to record detailed information
  - Made available as “metrics”.
  - Titan V on rai supports ~1100 metrics, some shown below
  - `$ nv-nsight-cu-cli --devices 0 --query-metrics`

lts__t_sectors_srcunit_l1_op_atom_dot_cas	# of LTS sectors from unit L1 for atomic CAS
l1tex__data_pipe_lsu_wavefronts_mem_shared_cmd_write	# of shared write wavefronts processed by Data-Stage
lts__t_sectors_srcunit_l1_aperture_sysmem_op_read	# of LTS sectors from unit L1 accessing system memory
(sysmem) for readslts__t_requests_op_red_lookup_hit	# of LTS requests for reductions that hit
lts__t_sectors_equiv_l1tagmiss_pipe_tex_mem_texture_op_ld	# of sectors requested for TLD instructions
l1tex__t_bytes_pipe_tex_lookup_miss	# of bytes requested that missed for TEX pipe
l1tex__texin_requests_mem_texture	# of texture requests (quads) sent to
TEXINl1tex__t_bytes_pipe_lsu_mem_local_op_ld_lookup_miss	# of bytes requested that missed for local
loadsl1tex__t_bytes_pipe_tex_mem_surface_op_red_lookup_miss	# of bytes requested that missed for surface reductions
…	

Record Kernel Trace

```
$ nv-nsight-cu-cli                \
    --kernel-id ::mygemm:6        \
    --section ".*"                \
    -o 1-1-pinned-basic           \
    1-1-pinned-basic
```

Profile the 6th time the "mygemm" kernel runs

Record metrics for all report sections

Create "1-1-pinned-basic.nsight-cuprof-report"

Name of the CUDA executable to profile

Nsight Compute Sections

A group of related measurements
The default list can be generated by
$ nv-nsight-cu-cli --list-sections

Without the --sections options, this is what would be recorded

We provide a regex that matches all sections

```
---------------------------- ------------------------------- ------- ------------------------------------------------
Identifier                   Display Name                    Enabled Filename
---------------------------- ------------------------------- ------- ------------------------------------------------
ComputeWorkloadAnalysis      Compute Workload Analysis       no      .../../../sections/ComputeWorkloadAnalysis.section
InstructionStats             Instruction Statistics          no      ...64/../../sections/InstructionStatistics.section
LaunchStats                  Launch Statistics               yes     ...1_3-x64/../../sections/LaunchStatistics.section
MemoryWorkloadAnalysis       Memory Workload Analysis        no      ...4/../../sections/MemoryWorkloadAnalysis.section
MemoryWorkloadAnalysis_Chart  Memory Workload Analysis Chart  no      ..../sections/MemoryWorkloadAnalysis_Chart.section
MemoryWorkloadAnalysis_Tables Memory Workload Analysis Tables no     .../sections/MemoryWorkloadAnalysis_Tables.section
Occupancy                    Occupancy                       yes     ...ibc_2_11_3-x64/../../sections/Occupancy.section
SchedulerStats               Scheduler Statistics            no      ...-x64/../../sections/SchedulerStatistics.section
SourceCounters               Source Counters                 no      ..._11_3-x64/../../sections/SourceCounters.section
SpeedOfLight                 GPU Speed Of Light              yes     ..._2_11_3-x64/../../sections/SpeedOfLight.section
WarpStateStats               Warp State Statistics           no      ...-x64/../../sections/WarpStateStatistics.section
```

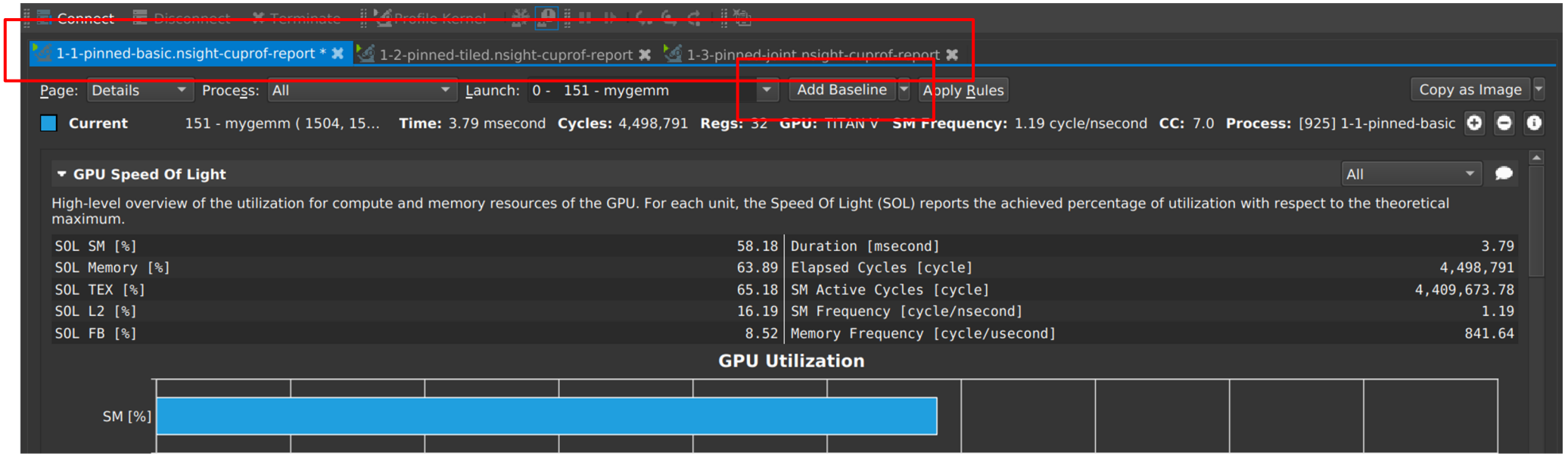
Open in Nsight Compute

Start Nsight Compute

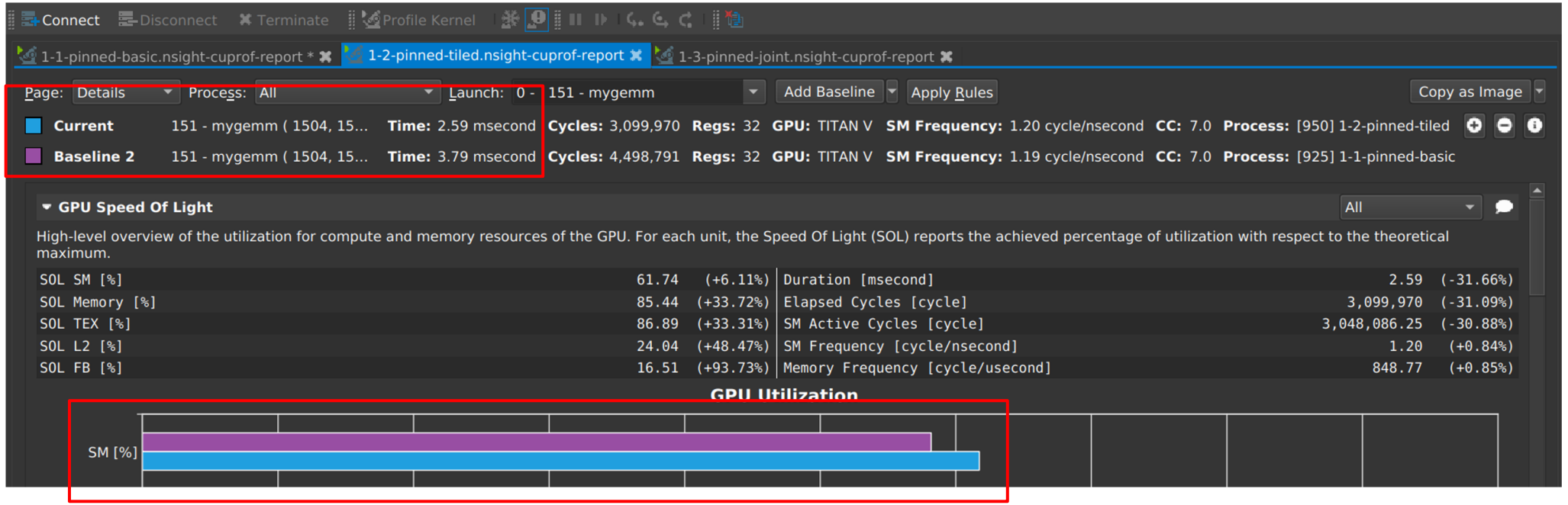
File > Open File ... > `1-1-pinned-basic.nsight-cuprof-report`

- Can open multiple files, will be open in multiple tabs
  - Can also use different runs as “baselines” for comparison in the same tab
  - Click “Add Baseline”

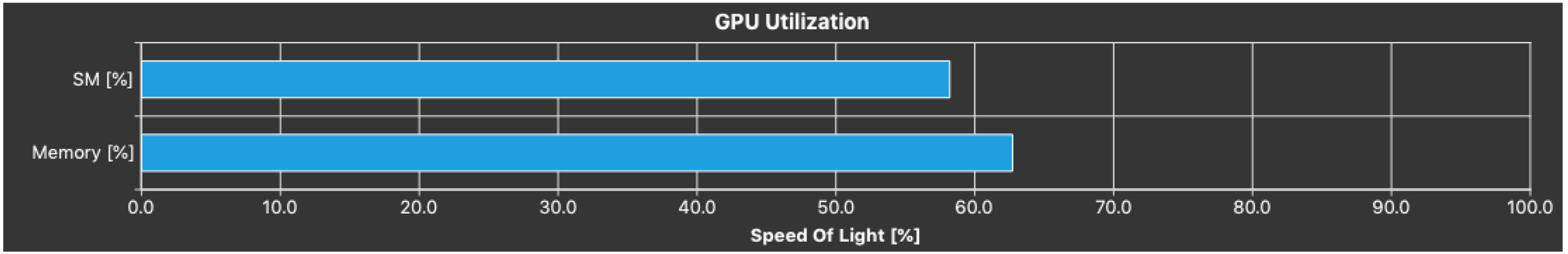
Tabs and baseline button



Next tab now has comparison



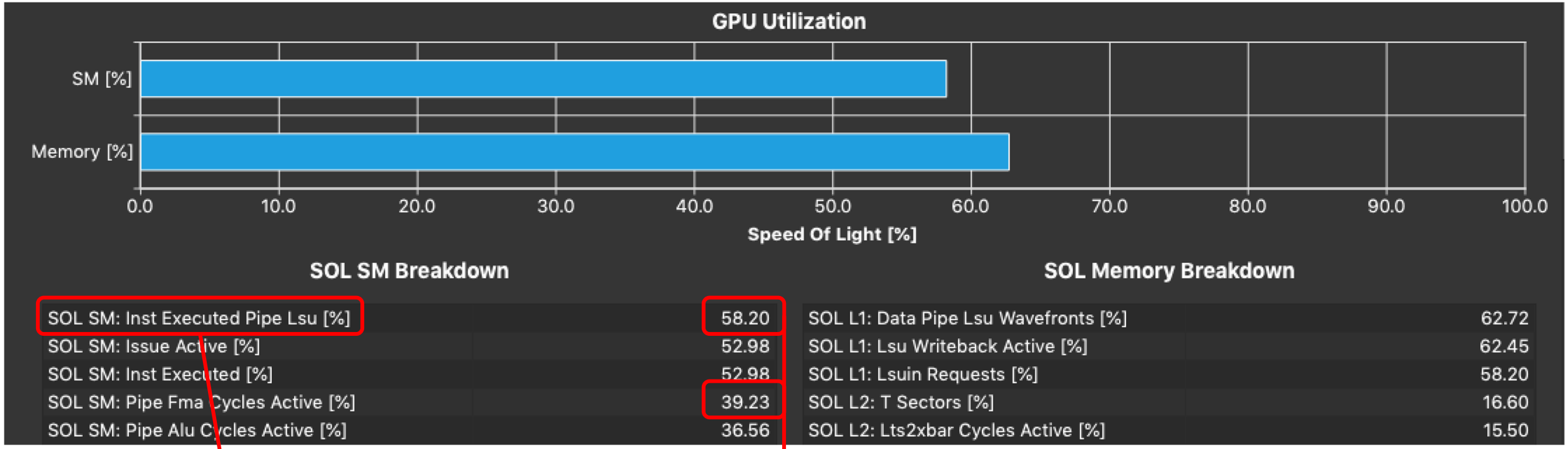
GPU “Speed of Light” (SoL): High-Level Bottleneck



1-1-pinned-basic.nsight-cuprof-report

Achieved % of utilization with respect to maximum

	SM < 60	**SM > 60**
Memory < 60	Latency	Compute
Memory > 60	Bandwidth	Compute & Bandwidth



Mouse over to see metric name

SM is mostly busy issuing memory operations.
58.2% instructions executed by Load-Store Unit.
only 35-40% of cycles ALU / FMA is busy

Section: GPU Speed of Light

- Achieved percentage of utilization w.r.t theoretical maximum

	Basic	**Tiled**
(GFLOPS)	1787	2585
SoL SM	58.18	61.74
SoL Memory	63.89	85.44

Workload Memory Analysis: Memory Chart

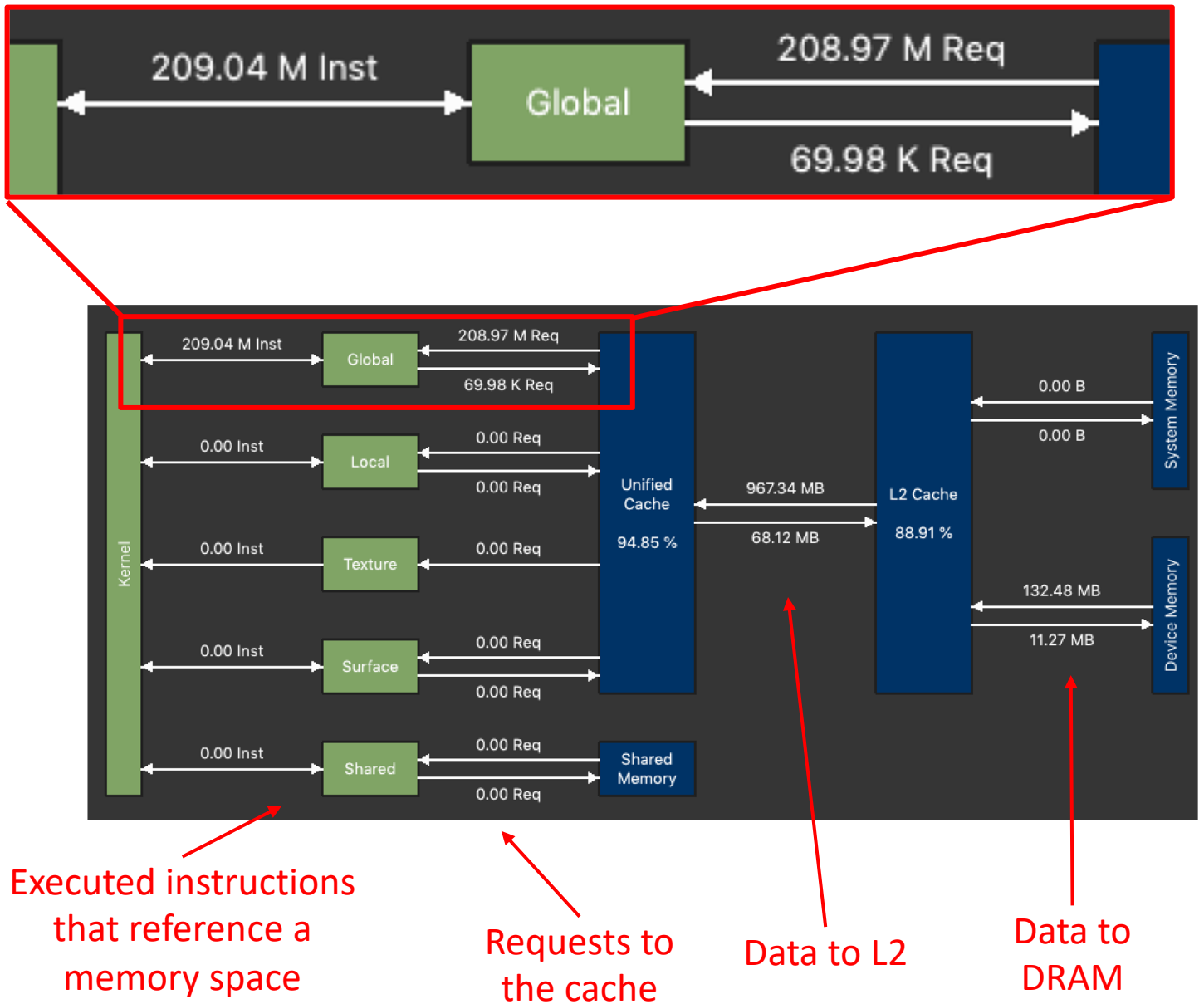
Global Memory: shared by all threads

Local Memory: private per-thread

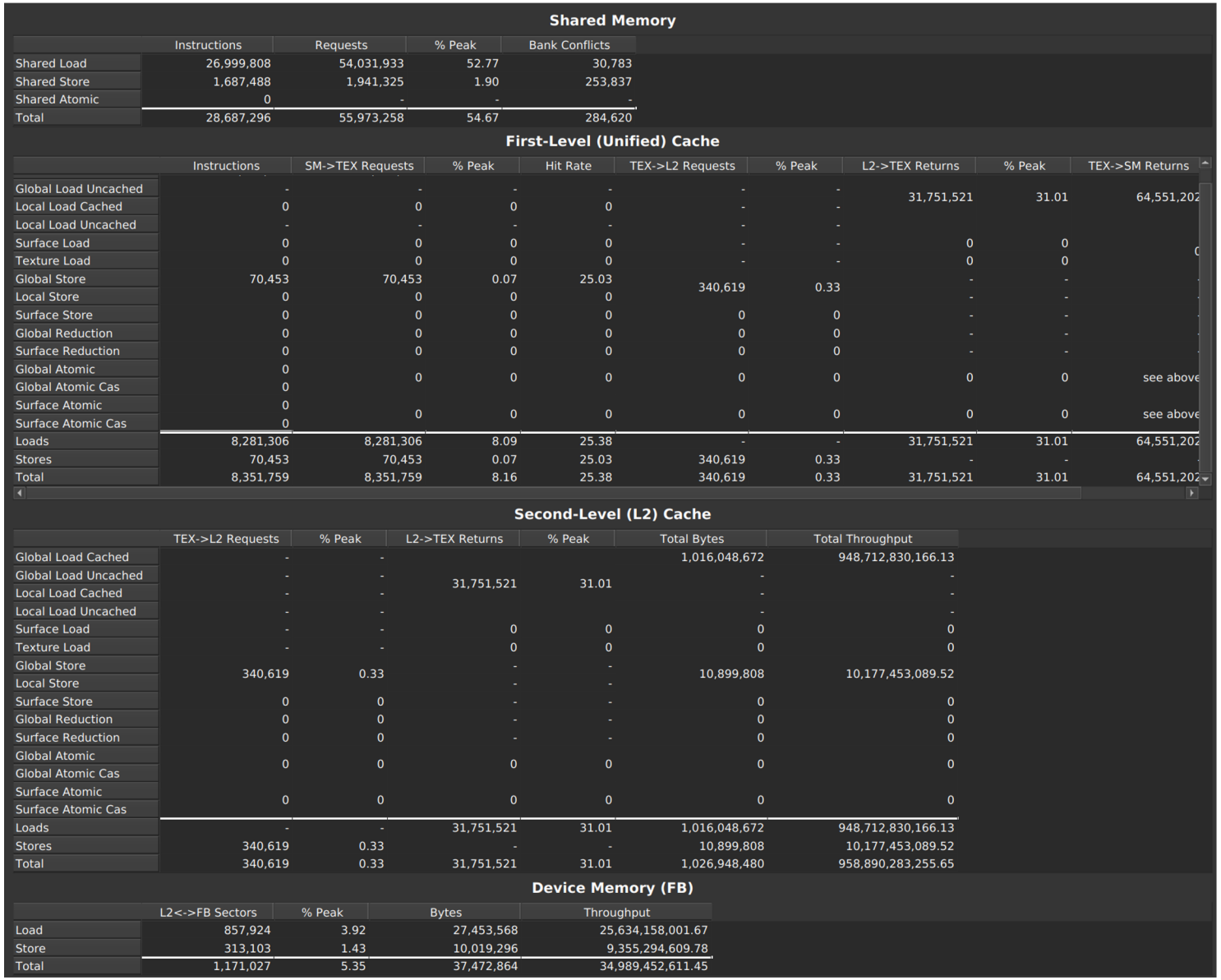
Shared Memory: shared by threads in a block

Texture/Surface: Cached for 2D spatial locality

Constant (?): Cached in the constant cache



Memory Workload Analysis: Charts



Shared Memory

	Instructions	Requests	% Peak	Bank Conflicts
Shared Load	26,999,808	54,031,933	52.77	30,783
Shared Store	1,687,488	1,941,325	1.90	253,837
Shared Atomic	0	-	-	-
Total	28,687,296	55,973,258	54.67	284,620

First-Level (Unified) Cache

	Instructions	SM->TEX Requests	% Peak	Hit Rate	TEX->L2 Requests	% Peak	L2->TEX Returns	% Peak	TEX->SM Returns
Global Load Uncached	-	-	-	-	-	-	31,751,521	31.01	64,551,202
Local Load Cached	0	0	0	0	-	-			
Local Load Uncached	-	-	-	-	-	-			
Surface Load	0	0	0	0	-	-	0	0	0
Texture Load	0	0	0	0	-	-	0	0	
Global Store	70,453	70,453	0.07	25.03	340,619	0.33	-	-	-
Local Store	0	0	0	0			-	-	-
Surface Store	0	0	0	0	0	0	-	-	-
Global Reduction	0	0	0	0	0	0	-	-	-
Surface Reduction	0	0	0	0	0	0	-	-	-
Global Atomic	0	0	0	0	0	0	0	0	see above
Global Atomic Cas	0								
Surface Atomic	0								
Surface Atomic Cas	0	0	0	0	0	0	0	0	see above
Loads	8,281,306	8,281,306	8.09	25.38	-	-	31,751,521	31.01	64,551,202
Stores	70,453	70,453	0.07	25.03	340,619	0.33	-	-	
Total	8,351,759	8,351,759	8.16	25.38	340,619	0.33	31,751,521	31.01	64,551,202

Second-Level (L2) Cache

	TEX->L2 Requests	% Peak	L2->TEX Returns	% Peak	Total Bytes	Total Throughput
Global Load Cached	-	-			1,016,048,672	948,712,830,166.13
Global Load Uncached	-	-	31,751,521	31.01	-	-
Local Load Cached	-	-			-	-
Local Load Uncached	-	-			-	-
Surface Load	-	-	0	0	0	0
Texture Load	-	-	0	0	0	0
Global Store	340,619	0.33	-	-	10,899,808	10,177,453,089.52
Local Store			-	-		
Surface Store	0	0	-	-	0	0
Global Reduction	0	0	-	-	0	0
Surface Reduction	0	0	-	-	0	0
Global Atomic	0	0	0	0	0	0
Global Atomic Cas						
Surface Atomic	0	0	0	0	0	0
Surface Atomic Cas						
Loads	-	-	31,751,521	31.01	1,016,048,672	948,712,830,166.13
Stores	340,619	0.33	-	-	10,899,808	10,177,453,089.52
Total	340,619	0.33	31,751,521	31.01	1,026,948,480	958,890,283,255.65

Device Memory (FB)

	L2<->FB Sectors	% Peak	Bytes	Throughput
Load	857,924	3.92	27,453,568	25,634,158,001.67
Store	313,103	1.43	10,019,296	9,355,294,609.78
Total	1,171,027	5.35	37,472,864	34,989,452,611.45

- Detailed information summarized in the Memory Chart
- Uses TEX to mean the first-level cache.

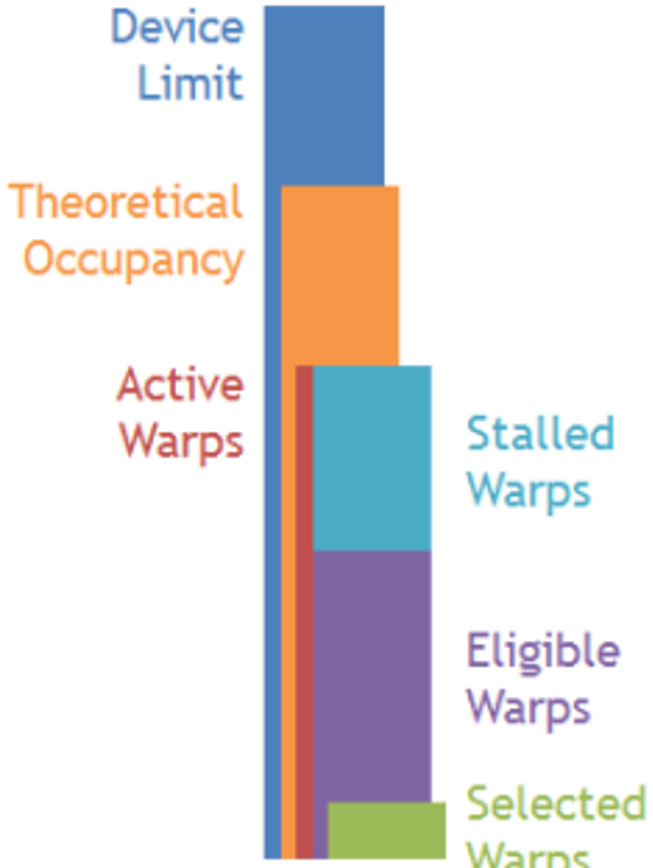
Memory Workload Analysis

	Basic	Tiled
GFLOPS	1787	2585
Speed of Light: Memory	63.89	85.44
Global Load Cached (% peak)	59.24	2.70
Global Load Cached (SM->TEX REQ)	209M	6.6M
Shared Load (REQ)	0	160M
L1 Hit Rate	94.84	74.75
Global Load (B)	197M	266M
Global Store (B)	11.7M	11.9M

Replaced global loads with shared loads

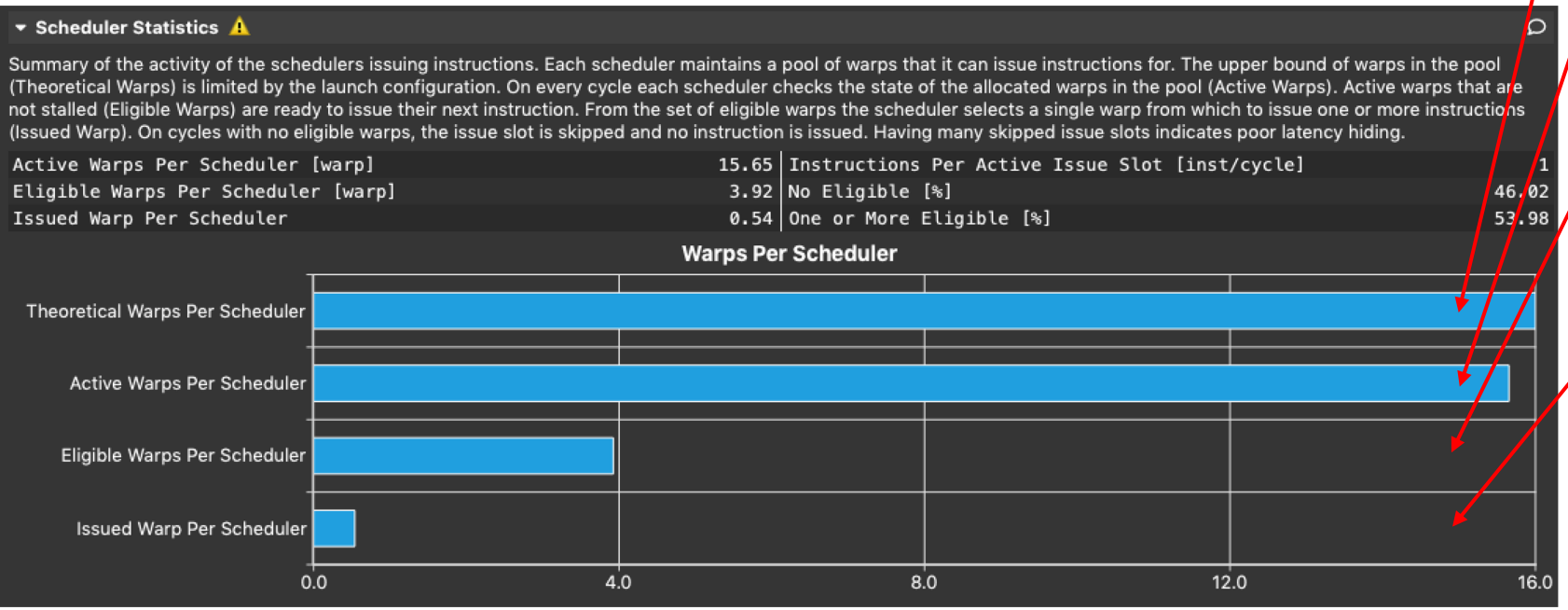
Cache reduces global memory traffic better than shared memory?

Scheduler Statistics



	Maximum Fermi	Maximum Kepler	Scope	Limiter
Device Limit	48 Warps	64 Warps		
Theoretical Occupancy	≤48 Warps	≤64 Warps	Launch	Launch Configuration
Active Warps			Warp Launch & Complete	Scheduler Load Balancing
Stalled Warps			Cycle	
Eligible Warps			Cycle	Issue Stalls
Selected Warps	≤2 Warps	≤4 Warps	Cycle	Eligible Warps & # Warp Schedulers

Scheduler Statistics



Pool of warps that the scheduler can pick from. Limited by device.

Number of warps actually given to SM: not enough work, or work imbalance

Number of warps ready to execute: waiting for barrier, watching for instruction fetch, waiting for data…

Number of issued warps: usually maximum of 1 or 2 depending on hardware.

Just because average value is good, doesn't mean warp scheduling chances are missed

39

Scheduler Statistics Comparison

	Basic	Tiled
GFLOPS	1787	2585
Theoretical Warps / Scheduler	16	16
Active Warps / Scheduler	15.65	15.70
Eligible Warps / Scheduler	3.92	2.58
Issued Warps / Scheduler	0.54	0.33
No Eligible (%)	46	67
One or More Eligible (%)	54	33

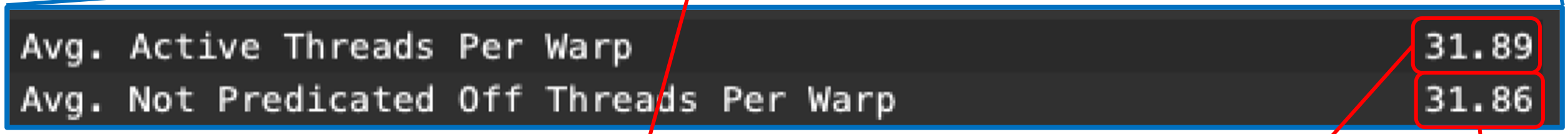
Average 2-3 cycles between a warp being ready

Warp State Statistics

Warp State Statistics

Analysis of the states in which all warps spent cycles during the kernel execution. The warp states describe a warp's readiness or inability to issue its next instruction. The warp cycles per instruction define the latency between two consecutive instructions. The higher the value, the more warp parallelism is required to hide this latency. For each warp state, the chart shows the average number of cycles spent in that state per issued instruction. Stalls are not always impacting the overall performance nor are they completely avoidable. Only focus on stall reasons if the schedulers fail to issue every cycle.

Warp Cycles Per Issued Instruction	29.00	Avg. Active Threads Per Warp	31.89
Warp Cycles Per Issue Active	29.00	Avg. Not Predicated Off Threads Per Warp	31.86
Warp Cycles Per Executed Instruction [cycle]	29.00	–	–



latency between two consecutive instructions
More latency: more warp parallelism needed to hide

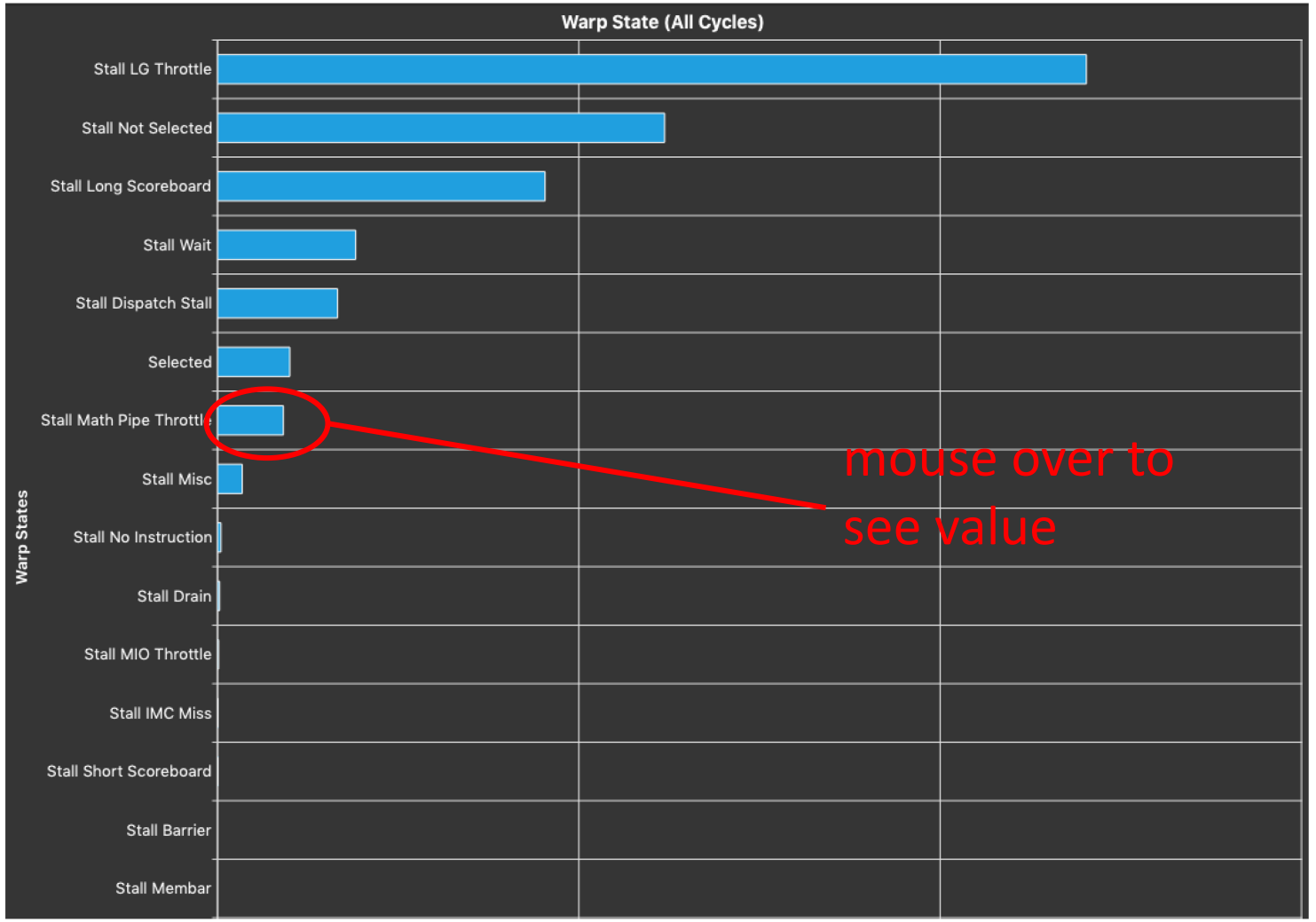
Block size not multiple of warp size
or
threads do not exit kernel at same time

Average control divergence

Warp State Statistics

What portion of the cycles per instruction the warps were stalled in each state.

Stalls cannot always be avoided and only really matter if instructions can't be issued every cycle



Warp State Statistics

	Description	Basic	Tiled
GFLOPS		1787	2585
Warp Cycles per Issued Instruction	latency between two consecutive instructions	29.00	47.34
Stall Long Scoreboard	Waiting for local, global, texture, or surface load	4.53	6.52
Stall Barrier	Waiting at barrier	0	4.73
Stall MIO Throttle	Waiting for MIO queue, caused by loads (incl. shared), and special math	0.01	22.61
Stall LG Throttle	Waiting for load-store-unit queue for local and global memory instructions	11.83	1.79
Stall Not selected	Eligible but not selected because another eligible warp was	6.89	6.82

Replaced global loads with shared

Section: Occupancy

Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

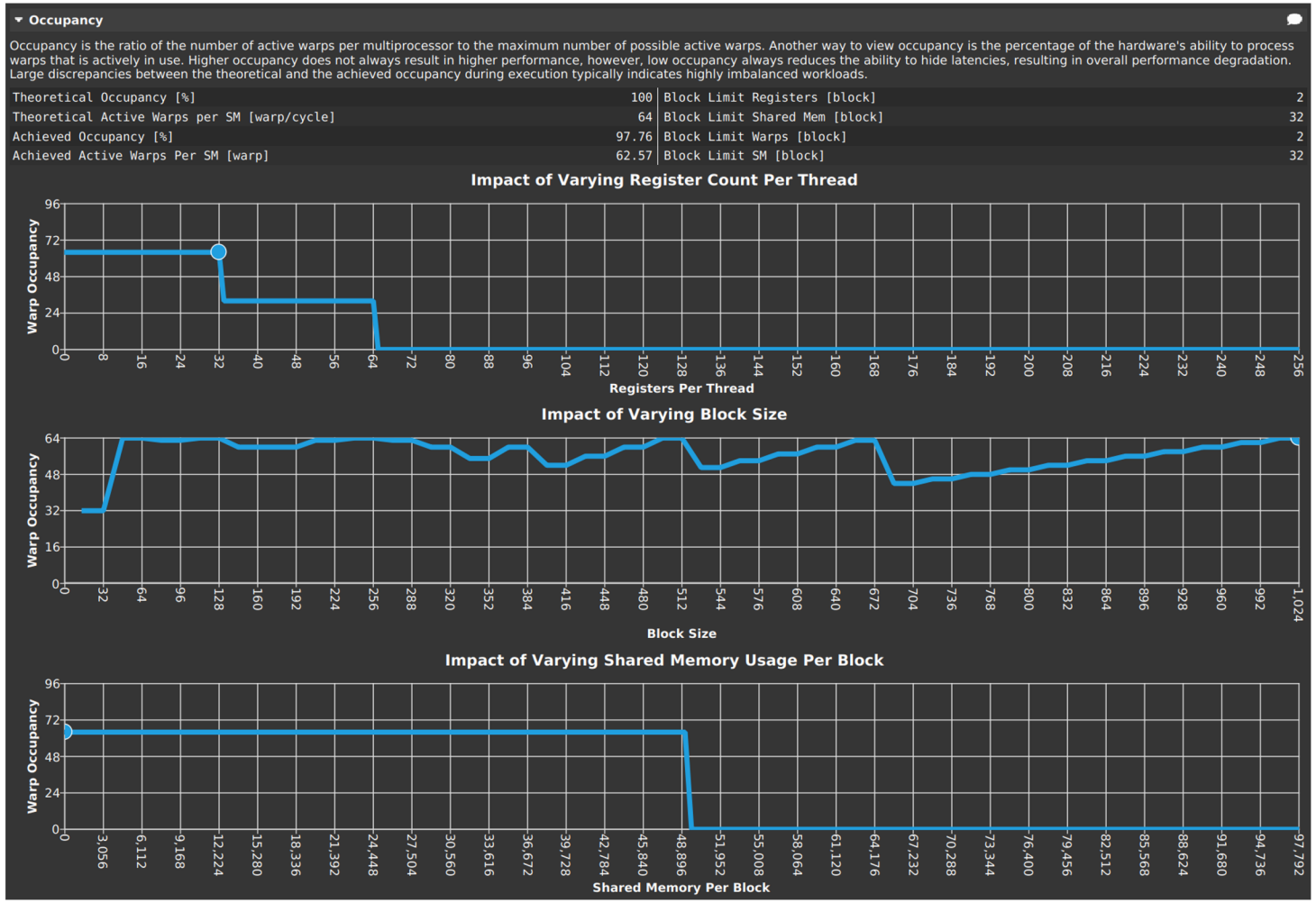
Theoretical Occupancy [%]	100	Block Limit Registers [block]	2
Theoretical Active Warps per SM [warp/cycle]	64	Block Limit Shared Mem [block]	32
Achieved Occupancy [%]	97.77	Block Limit Warps [block]	2
Achieved Active Warps Per SM [warp]	62.57	Block Limit SM [block]	32

Achieved occupancy: true amount of average active warps.
Lower if workload is imbalanced across blocks, too few blocks, or last wave is empty

Each SM can handle fixed number of warps, blocks, registers, and shared memory.

How each requirement restricts the theoretical occupancy

Section: Occupancy



Theoretical occupancy limited by device hardware and launch configuration

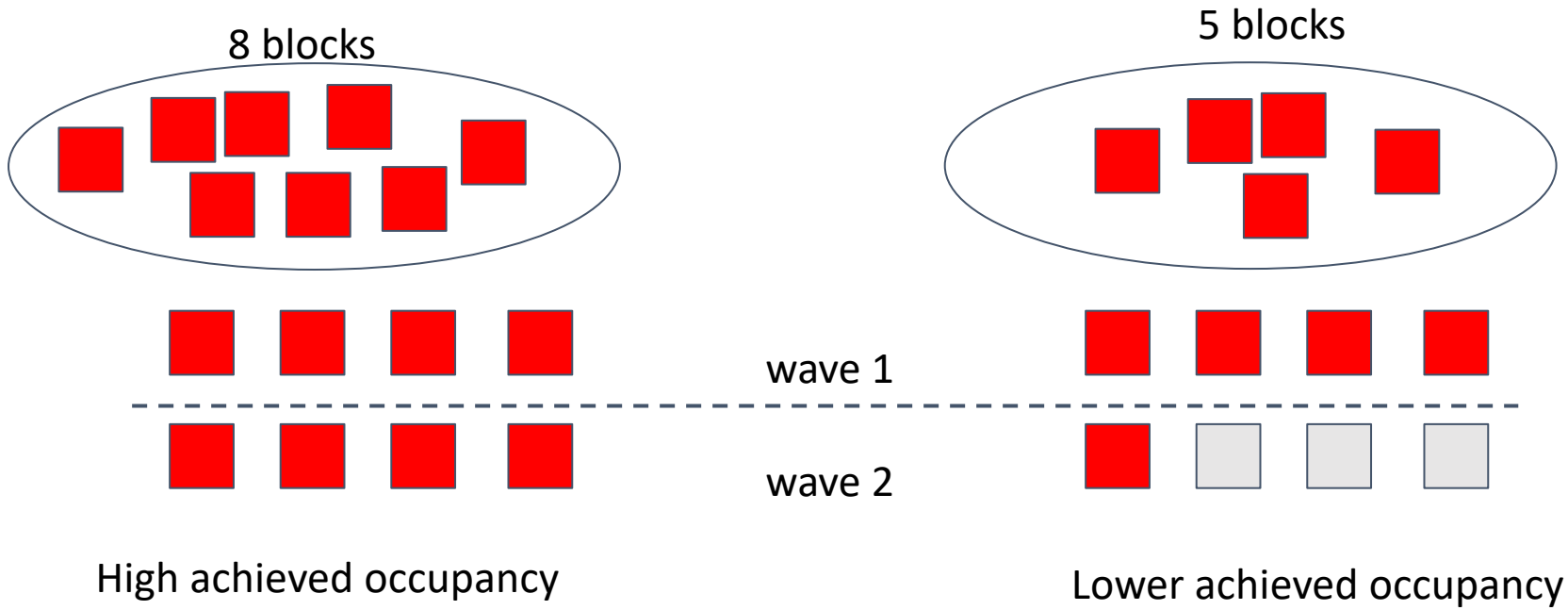
Achieved occupancy: true number of active warps as average

Lower if workload within or across blocks is imbalanced, if there are too few blocks, or the last wave is not large enough to fill GPU

Charts show how resources affect theoretical occupancy

Waves

Assume: 1 block per SM, GPU with 4 SMs



Launch Statistics and Occupancy

	Basic	Tiled
GFLOPS	1787	2585
Theoretical Occupancy	100	100
Th. Active Warps per SM	64	64
Achieved Occupancy	97.76	98.34
Waves per SM	13.81	13.81
Registers Per Thread	32	32

May not include registers for program counter!
Consult Nvidia's architecture whitepapers.
Titan V uses 2 additional registers for PC.

Instruction Hotspots

Show various metrics correlated with source code lines and PTX instructions

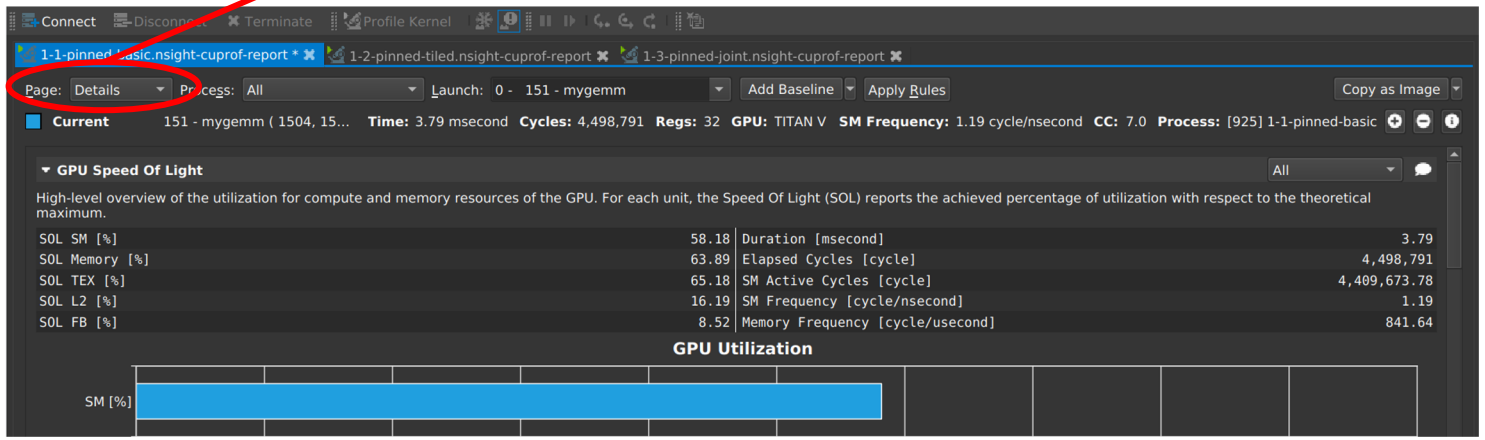
Some source code lines create many many PTX instructions: sometimes, split up a source line into many lines to get more details

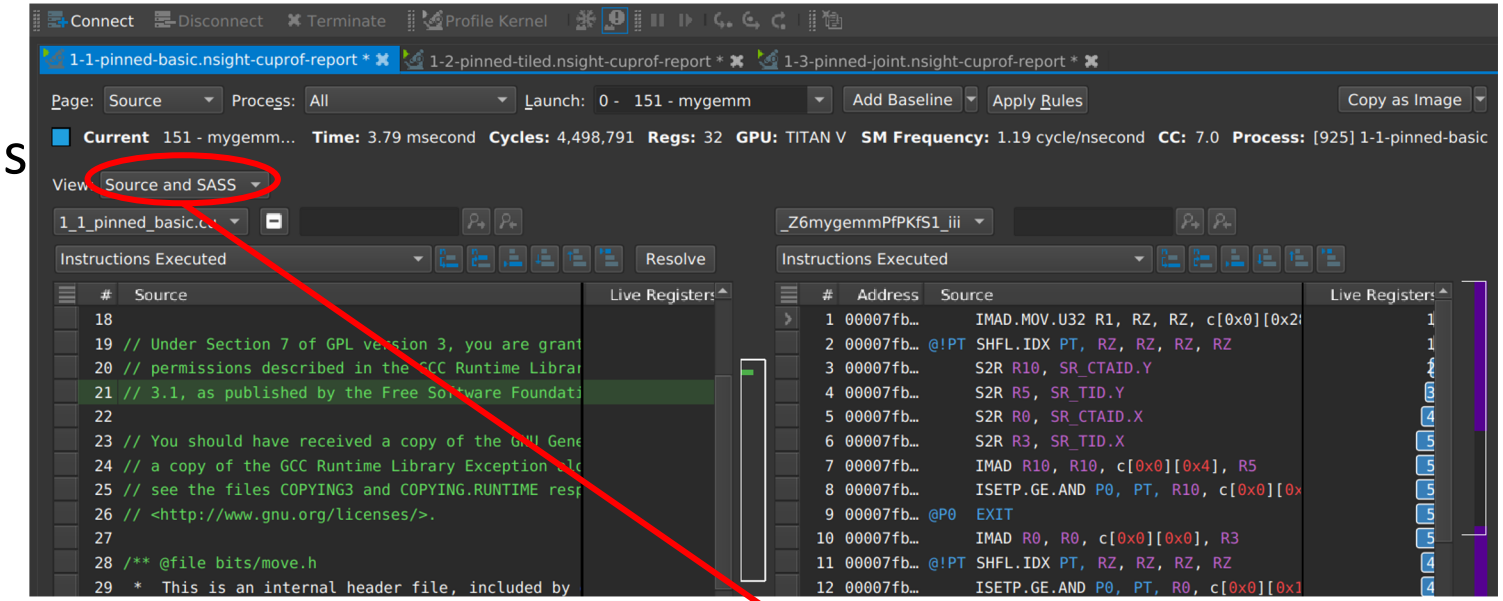
```
dst[i] = src[i] + reg;
```

vs

```
temp v = src[i];
v += reg;
dst[i] = v;
```

Switch page to “Source”





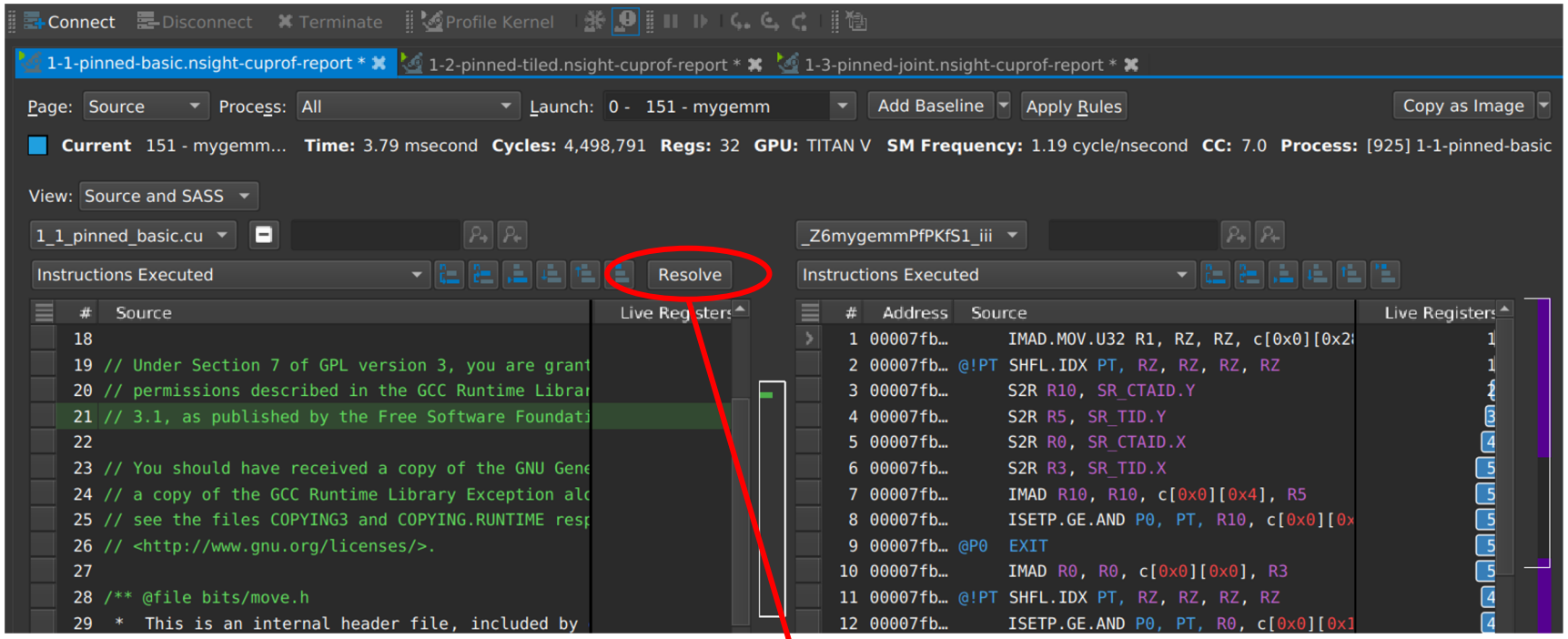
“Source and PTX” (usually) or “Source and SASS”

PTX: higher-level assembly, same between GPU models

SASS: specific code for a GPU model

Instruction Hotspots

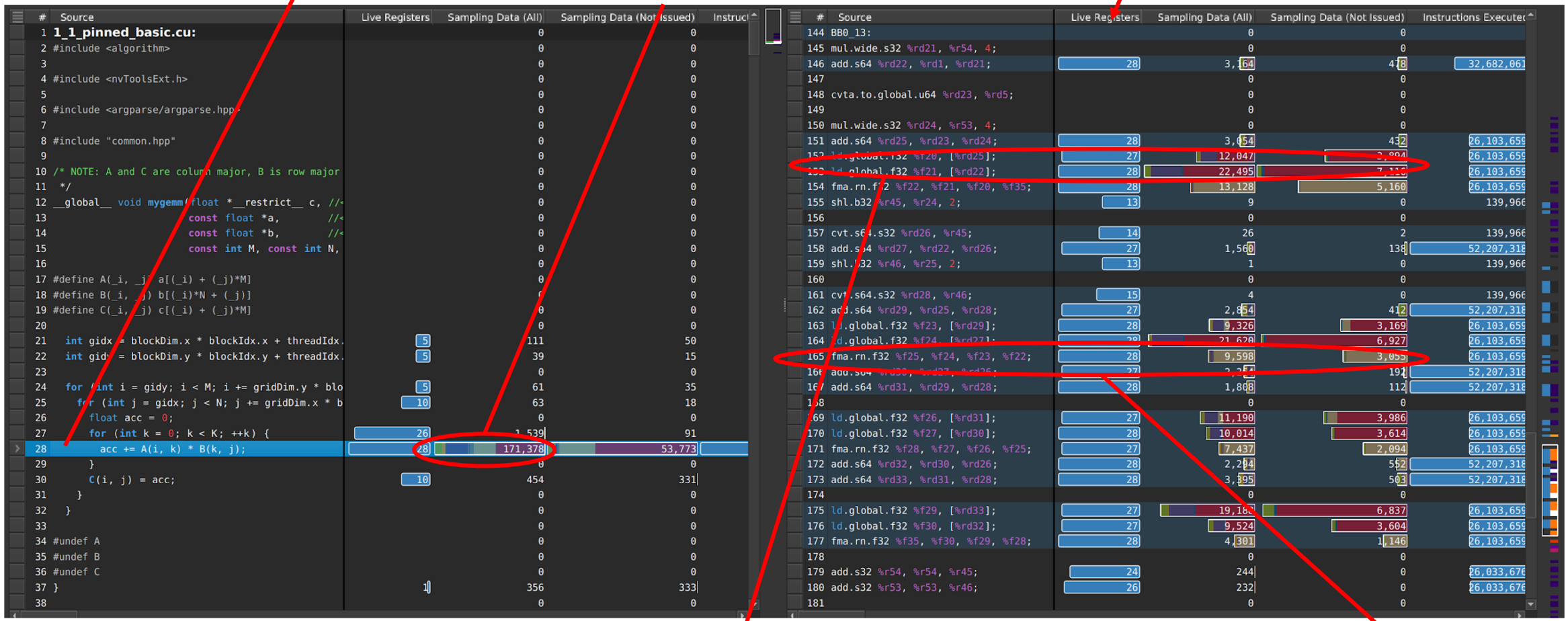
- If profiling on a different system, source file may not automatically load since paths may not match.



Hit “resolve” and load the corresponding source file from local file system

Instruction Sampling

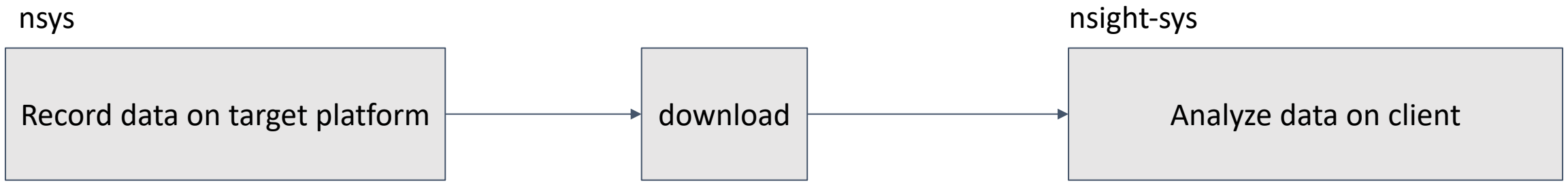
- Every so often, the position of the program counter is recorded
- Slower instructions are more likely to be recorded
- There will be many samples in slow parts of the code, and few in fast parts of the code


Click a line to highlight lines from other side
Program counter spends most of its time on instructions from this line. Mouse over for breakdown.
Corresponding PTX/SASS lines over here.
Multiple PTX lines per source line
Sometimes, stalls can show in a following instruction that depends on a previous one

System Profiling with Nsight Systems

Nvidia Nsight Systems

- Deliver work to the GPU effectively
  - Understand performance of surrounding system
- Two interfaces:
  - GUI (nsight-sys)
  - CLI (nsys)
- Like Nsight Compute, use a two-part record-then-analyze flow



Record Application Traces

```
$ nsys profile                  \
    -o 2-5-pinned-joint         \
    2-5-pinned-joint
```

Create “2-5-pinned-basic.qdrep”

Name of the CUDA executable to profile

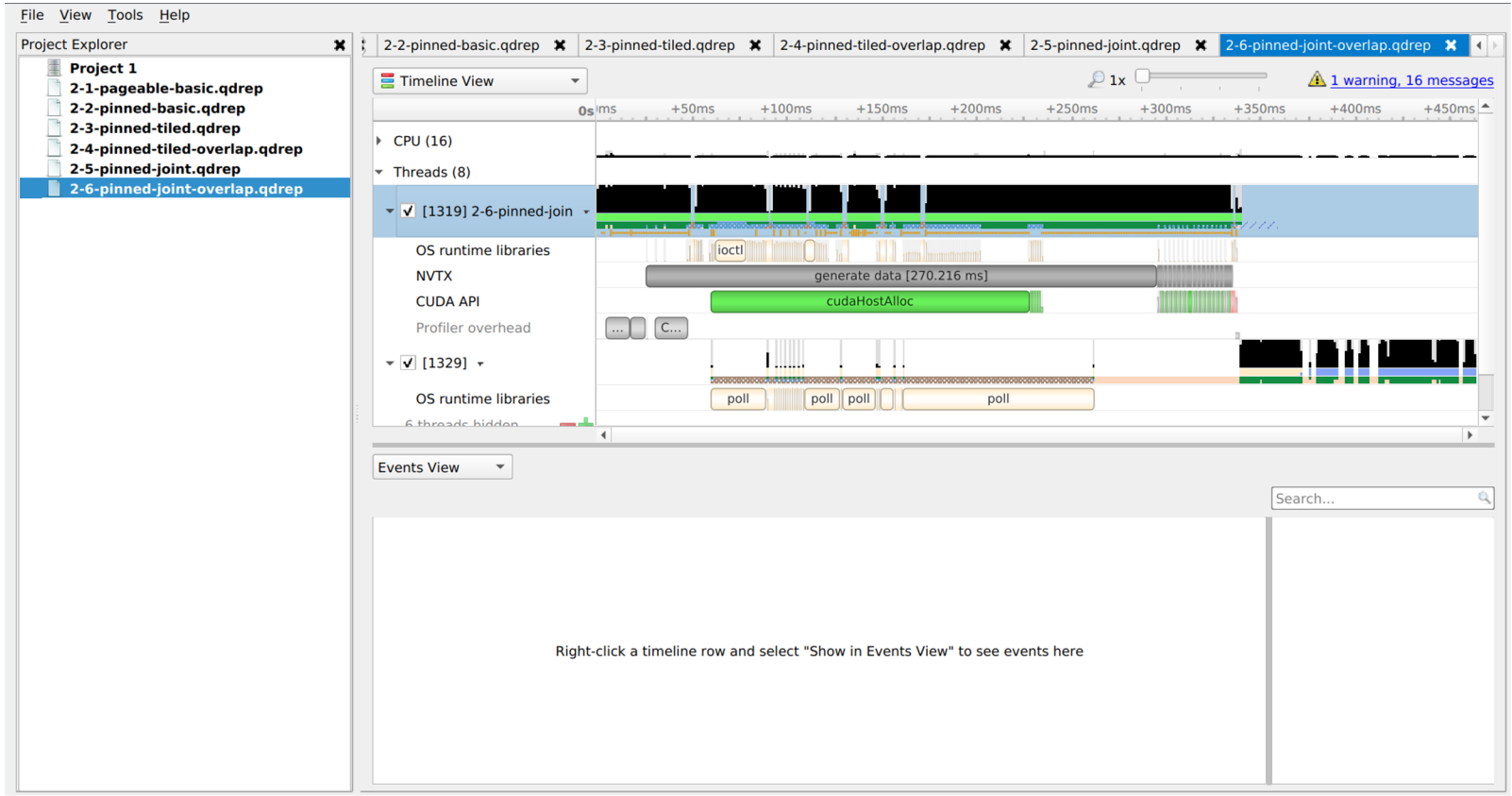
Nsight Systems GUI

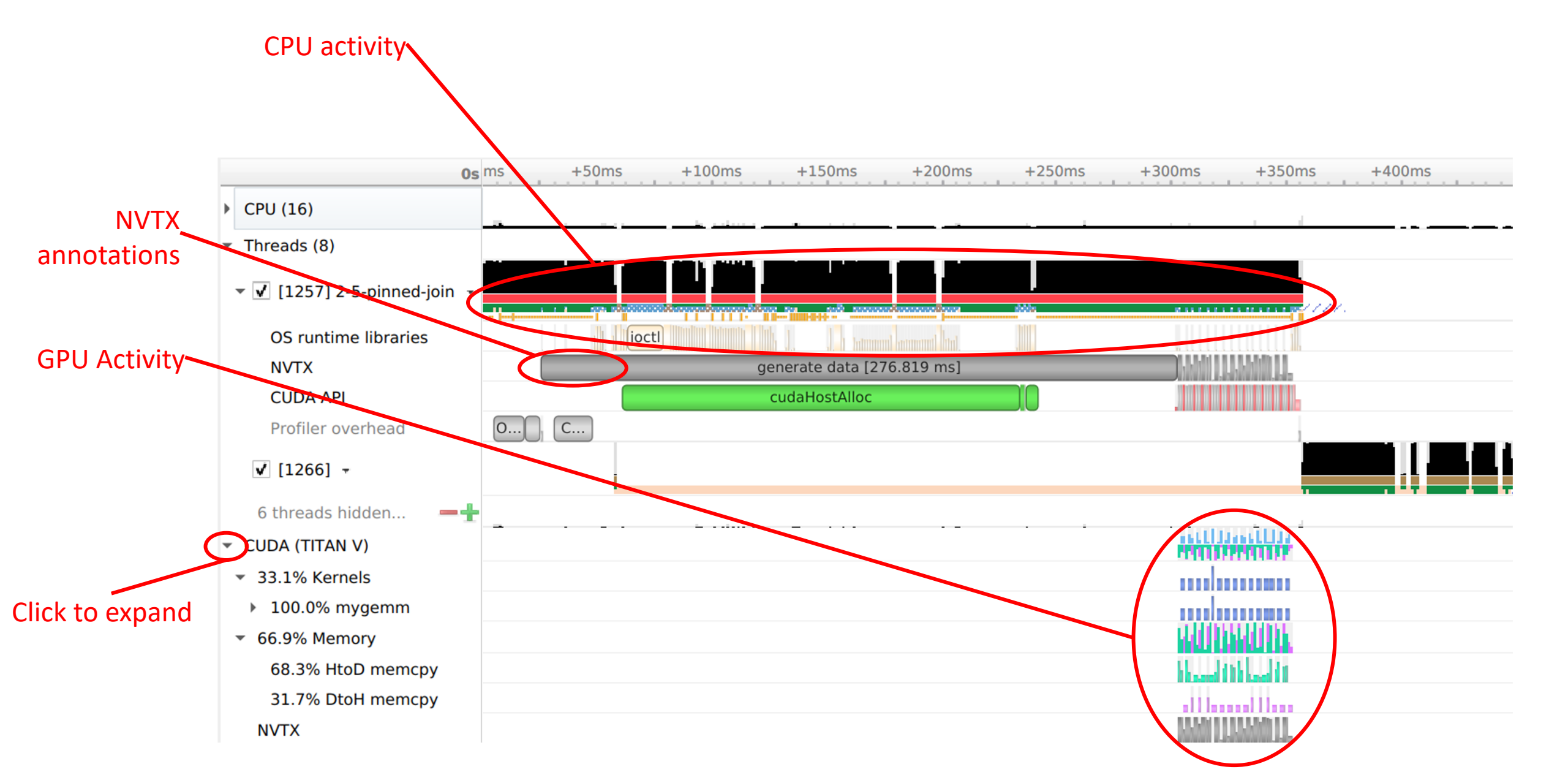
File > Open > `file.qdrqp`

Multiple files will be open, shown on the left pane.

Main view is a timeline of OS calls, CUDA calls, NVTX events, CUDA API calls, and GPU activity.

Open all the `.qdrep` files from the rai build directory you downloaded.

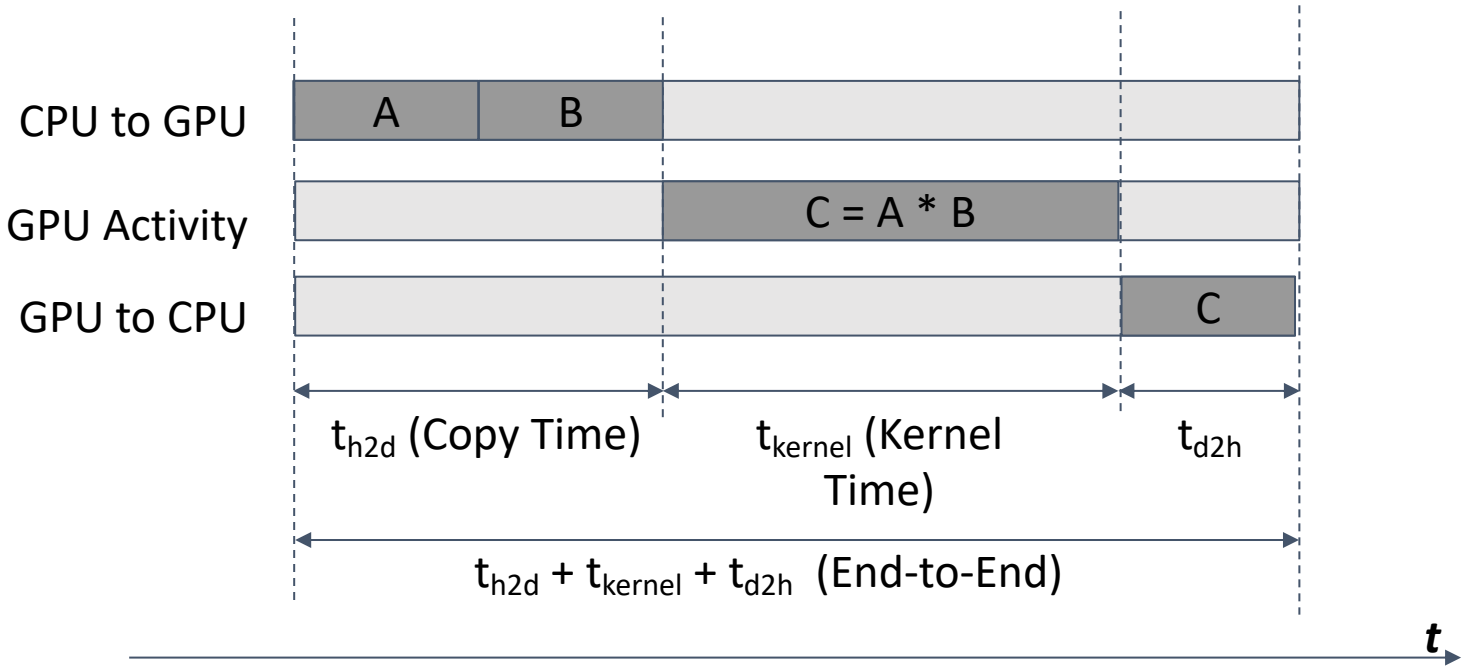



CPU activity
NVTX annotations
GPU Activity
Click to expand
0s
ms
+50ms
+100ms
+150ms
+200ms
+250ms
+300ms
+350ms
+400ms
CPU (16)
Threads (8)
[1257] 2-5-pinned-join
OS runtime libraries
ioctl
NVTX
generate data [276.819 ms]
CUDA API
cudaHostAlloc
Profiler overhead
O...
C...
[1266]
6 threads hidden...
CUDA (TITAN V)
33.1% Kernels
100.0% mygemm
66.9% Memory
68.3% HtoD memcpy
31.7% DtoH memcpy
NVTX

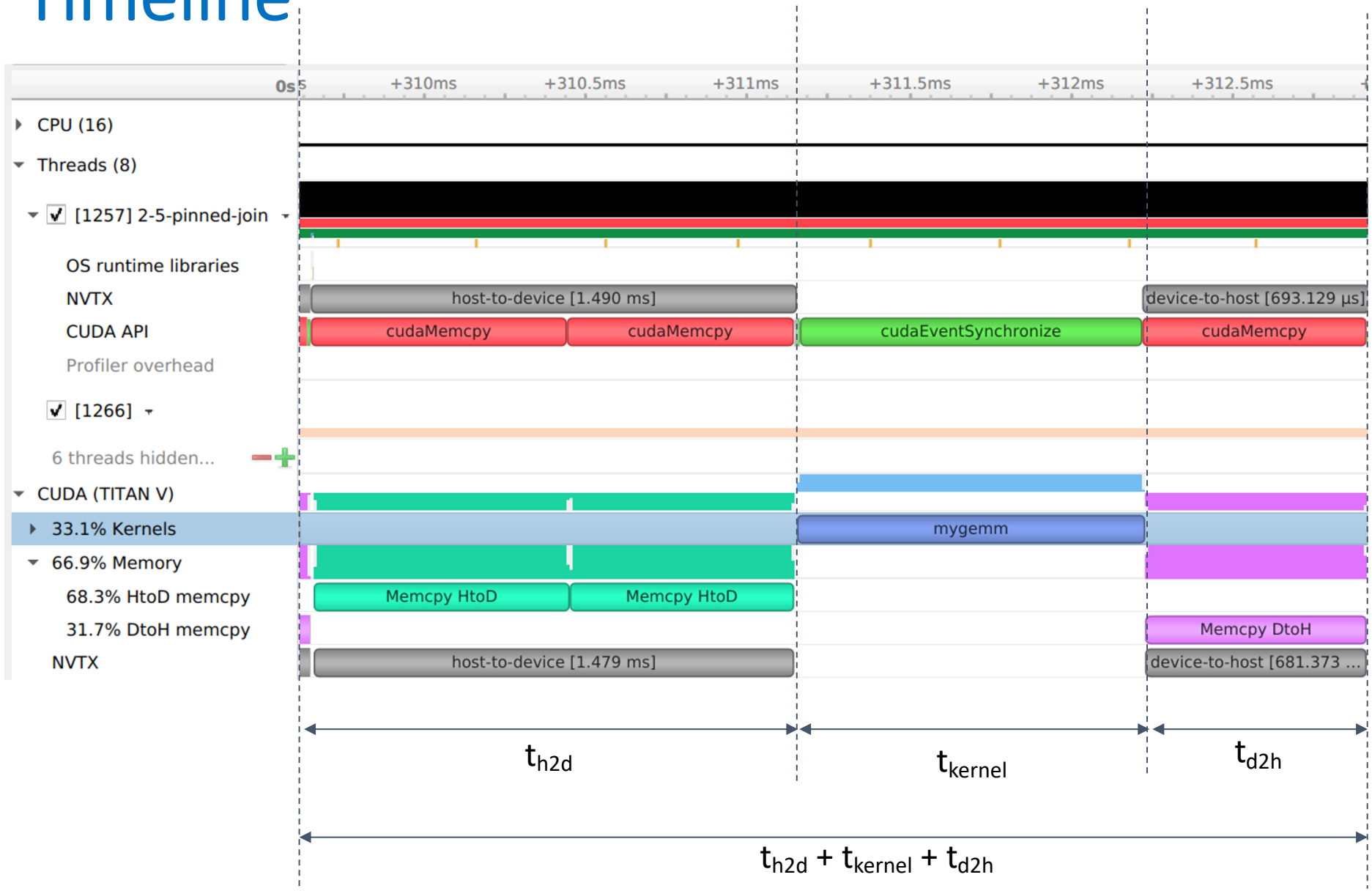
Click and drag to zoom in

Mouse over spans for more info

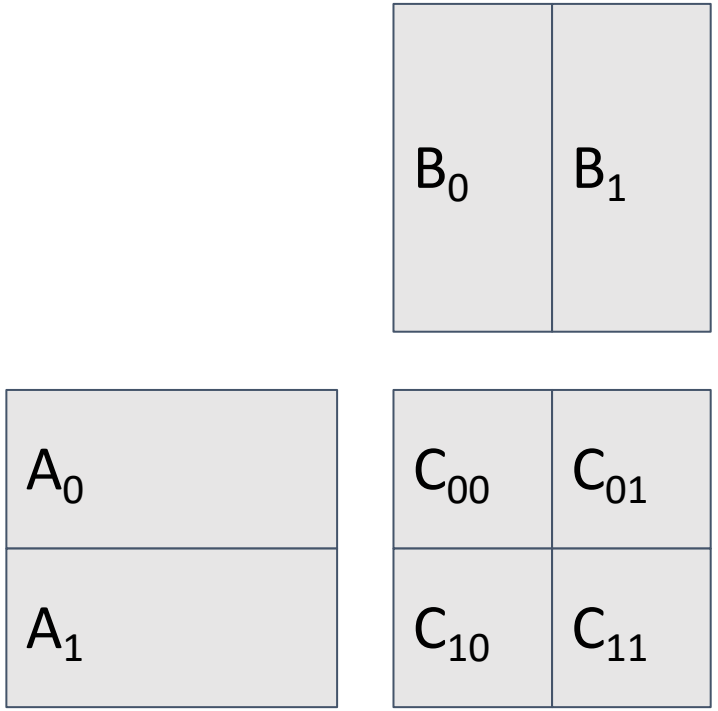
Kernel Time vs End-to-End Time



Read Timeline

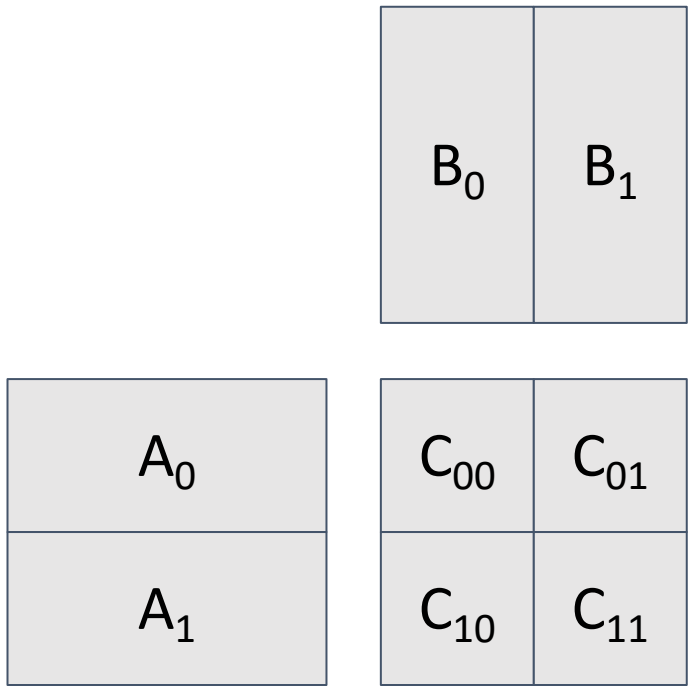


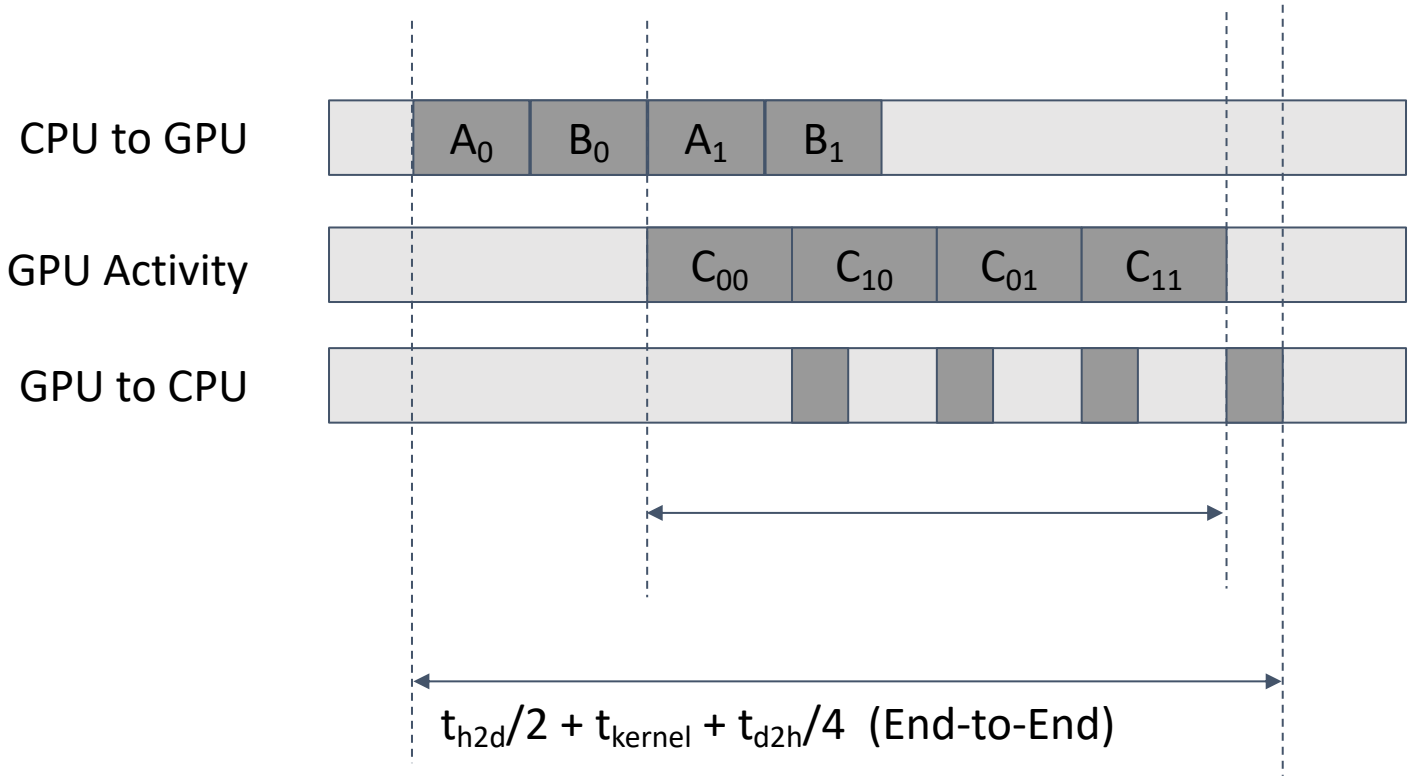
Overlap to reduce End-to-end Time



- C = A * B as four multiplications.
  - $C_{00} = A_0 * B_0$: needs only A_0, B_0
  - $C_{01} = A_0 * B_1$: after C_{00}, needs only B_1
  - $C_{10} = A_1 * B_0$: after C_{01}, needs only A_1
  - $C_{11} = A_1 * B_1$: immediately after C_{10}
- Copy slices of A and B onto GPU, and immediately start the multiplication.
- Also can copy results back as soon as they're ready

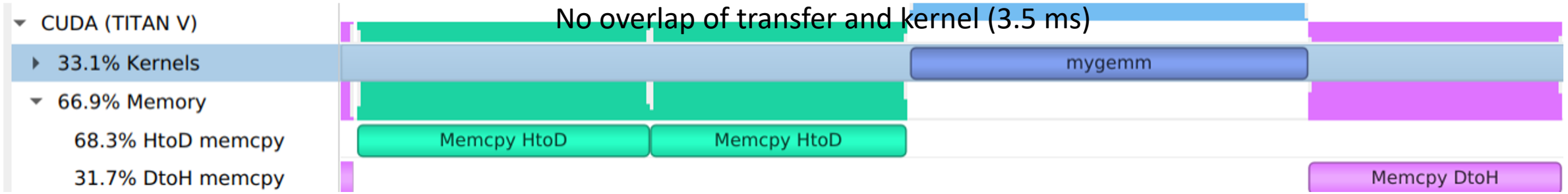
Overlap to Reduce End-to-End Time



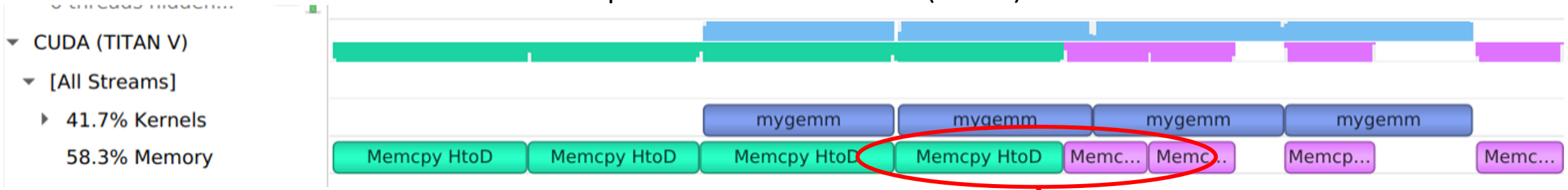


Real Timelines: Overlapping

No overlap of transfer and kernel (3.5 ms)



Overlap of transfer and kernel! (2.5ms)



D2H and H2D transfers in the same stream,
so they are not overlapped with each other

Further Reading

- Nsight Systems Documentation
  - https://docs.nvidia.com/nsight-systems/
- Nsight Compute Documentation
  - https://docs.nvidia.com/nsight-compute/
- Nvidia Developer Blog
  - Nsight Systems Exposes GPU Optimization (May 30 2018): https://devblogs.nvidia.com/nsight-systems-exposes-gpu-optimization/
  - Using Nsight Compute to Inspect your Kernels (Sep 16 2019): https://devblogs.nvidia.com/using-nsight-compute-to-inspect-your-kernels/
  - Using Nvidia Nsight Systems in Containers and the Cloud (Jan 29 2020) : https://devblogs.nvidia.com/nvidia-nsight-systems-containers-cloud/
- Workload Memory Analysis
  - CUDA Memory Model: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#memory-hierarchy
  - Device Memory Access Performance Guidelines: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#device-memory-accesses
- Stall Reasons
  - Nsight Graphics Docs: Stall Reasons: https://docs.nvidia.com/drive/drive_os_5.1.12.0L/nsight-graphics/activities/#shaderprofiler_stallreasons
  - Issue Efficiency Nsight Visual Studio Edition: https://docs.nvidia.com/gameworks/content/developertools/desktop/analysis/report/cudaexperiments/kernellevel/issueefficiency.htm
- Occupancy:
  - Nsight Visual Studio Edition: https://docs.nvidia.com/gameworks/content/developertools/desktop/analysis/report/cudaexperiments/kernellevel/achievedoccupancy.htm

Not Discussed

- Measuring across multiple streams with CUDA events
- Profiling through the Nsight Compute GUI
  - local/remote
- Profiling through the Nsight Systems GUI
  - local/remote
- In-kernel timing with clock()/clock64()
- Custom profiling hooks with CUDA Performance Tools Interface (CUPTI)

Thank You