10_HandsOnNsight_nsys

June 15, 2022

#

Hands-On Session with Nsight Systems and Compute

By: Brett Neuman `bneuman@ucar.edu`, Consulting Services Group, CISL & NCAR

Date: June 16th 2022

In this notebook we explore profiling of the mini-app MiniWeather to present profiling techniques and code examples. We will cover:

1. Overview of Profiling and Performance Sampling Tools
   - Typical development workflows with profiling tools
2. NSight Systems for Overview Analysis of GPU Program Runtimes
   - How to generate nsys reports and command line parameters
   - Analysis of nsys reports and investigating the program timeline
   - Generating NSight Compute profiling commands from nsys reports

Head to the NCAR JupyterHub portal and **start a JupyterHub session on Casper login** (or batch nodes using 1 CPU, no GPUs) and open the notebook in `10_HandsOnNsight/nsys/10_HandsOnNsight_nsys.ipynb`. Be sure to clone (if needed) and update/pull the NCAR GPU_workshop directory.

```
# Use the JupyterHub GitHub GUI on the left panel or the below shell commands
git clone git@github.com:NCAR/GPU_workshop.git
git pull
```

1 Workshop Etiquette

- Please mute yourself and turn off video during the session.
- Questions may be submitted in the chat and will be answered when appropriate. You may also raise your hand, unmute, and ask questions during Q&A at the end of the presentation.
- By participating, you are agreeing to UCAR's Code of Conduct
- Recordings & other material will be archived & shared publicly.
- Feel free to follow up with the GPU workshop team via Slack or submit support requests to support.ucar.edu
  - Office Hours: Asynchronous support via Slack or schedule a time with an organizer

1.1 Notebook Setup

Set the PROJECT code to a currently active project, ie UCIS0004 for the GPU workshop, and QUEUE to the appropriate routing queue depending on if during a live workshop session (gpuworkshop), during weekday 8am to 5:30pm MT (gpudev), or all other times (casper). Due to limited shared GPU resources, please use GPU_TYPE=gp100 during the workshop. Otherwise, set GPU_TYPE=v100 (required for gpudev) for independent work. See Casper queue documentation for more info.

[ ]:
```
export PROJECT=UCIS0004
export QUEUE=gpudev
export GPU_TYPE=gp100

module load nvhpc/22.2 &> /dev/null
export PNETCDF_INC=/glade/u/apps/dav/opt/pnetcdf/1.12.2/openmpi/4.1.1/nvhpc/22.
 ↪2/include
export PNETCDF_LIB=/glade/u/apps/dav/opt/pnetcdf/1.12.2/openmpi/4.1.1/nvhpc/22.
 ↪2/lib
```

2 Profilers - Why bother?

So you have some code. Maybe you own it, maybe you're inheriting it, maybe you're trying to improve it, maybe you're just trying to keep it operational.

If you're looking to **understand, improve performance, or make informed decisions** on your code in a **timely** fashion, profiling is a **good place to start**.

The profiler does not make decisions for you. Profilers provide information that could lead to more efficient use of resources for your code! Be mindful that profiling can add significant runtime overhead to your application.

2.1 How to get there

1. Profile your code!
2. Make sure you have your baseline performance
   - Performance is relative here
   - Your baseline should be a realistic run of the application (real data, reasonable runtime)
3. Attempt to find potential performance gains using profiling tools, your experience, and working around your constraints
   - Common project constraints include:
     - cluster configurations
     - hardware architectures (CPU/GPU/NIC types)
     - memory
     - flow control (simple instructions vs branching instructions)
     - programming language

  - development time
- Tools can give you insight on what sections of code are using up significant runtime
  - A function with the highest runtime often has highest potential to be optimized .. **but not always**

2.2 Profiling data collection methods

1. **Sampling**
   - Collect data at a regular interval, or sampling frequency, to understand how much time is spent in a function or application
2. **Concurrency**
   - Identifying shared resource bottlenecks, communication overhead, and thread or kernel inefficiencies via call stack traces
3. **Memory**
   - Gathers information on data movement, allocation, and resource availability

2.3 The focus of our session

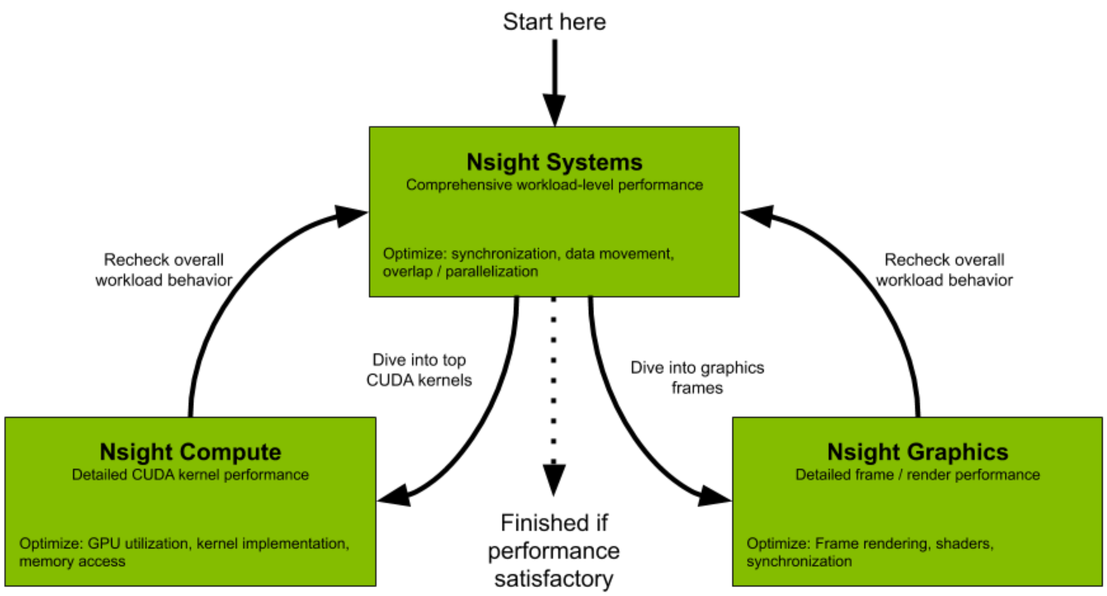
In this session we will focus on profiling code on clusters with NVidia GPUs in the role of a researcher. Our interest is in performant threads, kernels, GPU utilization, and memory efficiency.

3 NSight Systems and Compute

The Nsight Systems and Compute are used to profile, debug, and optimize applications that utilize Nvidia GPUs. You can follow along if you have Nsight Systems installed on your local machine.

Download: https://developer.nvidia.com/gameworksdownload#?dn=nsight-systems-2022-2

Casper runs Nsys version `2021.2.4.12`

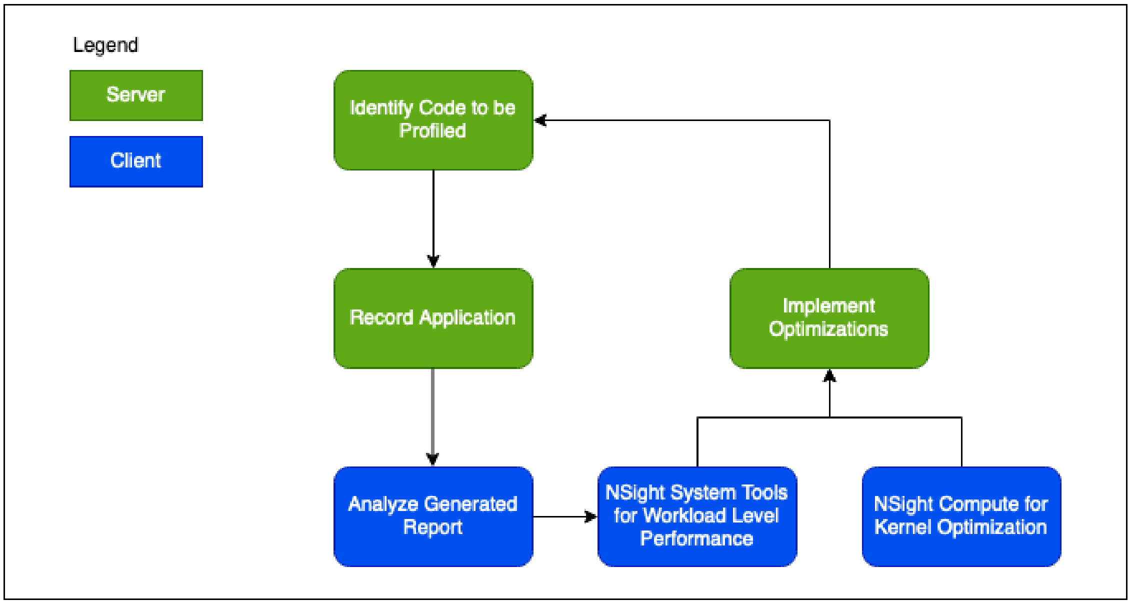


3.1 NSight Systems `nsys`

Workload level analysis: * Visualize algorithms, instruction flow, data flow, and scaling out to multiple nodes * Identify areas to optimize within the code * Maximize computational and memory utilization on the GPU

3.1.1 The NSight Systems Profiling Model

The Nsight profiling model is based on the **Client Server** model. The **Client** is your the machine you will use to view reports generated by your code profiling. The **Server** is the node you run GPU code on and generate the the profiling report from. NVidia refers to this as the **Two Phase** approach to profiling. A good workflow for profiling your code using the Client Server model would look like:



3.2 GPU kernel generation

Previously, we ran ACC directives on our miniWeather application. Compilers handle the conversion into GPU code behind the scenes but it is important to note that ACC directives are converted into a NVidia CUDA kernel. These kernels can be analyzed for performance using Nsight Systems and Compute.

4 miniWeather App openACC Profiling example

4.1 Baseline: Profile Generation and Analysis

We're going to profile the miniweather application using the most basic version of `!$acc loop parallel` without any additional flags to help the compiler generate efficient parallel loops. This might be a first step to converting a CPU based function into an OpenACC. Remember, your baseline should be a stable working version of your code with a realistic dataset and runtime. Here we're looking at one example of this implementation on the `semi_discrete_step` subroutine.

```
!Perform a single semi-discretized step in time with the form:
!state_out = state_init + dt * rhs(state_forcing)
!Meaning the step starts from state_init, computes the rhs using state_forcing, and stores the result in state_out
subroutine semi_discrete_step( state_init , state_forcing , state_out , dt , dir , flux , tend )
  implicit none
  real(rp), intent(in   ) :: state_init   (1-hs:nx+hs,1-hs:nz+hs,NUM_VARS)
  real(rp), intent(inout) :: state_forcing(1-hs:nx+hs,1-hs:nz+hs,NUM_VARS)
  real(rp), intent(  out) :: state_out    (1-hs:nx+hs,1-hs:nz+hs,NUM_VARS)
  real(rp), intent(  out) :: flux         (nx+1,nz+1,NUM_VARS)
  real(rp), intent(  out) :: tend         (nx,nz,NUM_VARS)
  real(rp), intent(in   ) :: dt
  integer , intent(in   ) :: dir
  integer :: i,k,ll
  real(rp) :: x, z, wpert, dist, x0, z0, xrad, zrad, amp

  if     (dir == DIR_X) then
    !Set the halo values for this MPI task's fluid state in the x-direction
    call set_halo_values_x(state_forcing)
    !Compute the time tendencies for the fluid state in the x-direction
    call compute_tendencies_x(state_forcing,flux,tend,dt)
  elseif (dir == DIR_Z) then
    !Set the halo values for this MPI task's fluid state in the z-direction
    call set_halo_values_z(state_forcing)
    !Compute the time tendencies for the fluid state in the z-direction
    call compute_tendencies_z(state_forcing,flux,tend,dt)
  endif

  !Apply the tendencies to the fluid state
  !$acc parallel loop
  do ll = 1 , NUM_VARS
    do k = 1 , nz
      do i = 1 , nx
        if (data_spec_int == DATA_SPEC_GRAVITY_WAVES) then
          x = (i_beg-1 + i-0.5_rp) * dx
          z = (k_beg-1 + k-0.5_rp) * dz
```

4.2 Setting up a baseline

The Nsight Systems profile launch within this script:

`nsys profile -o miniweather_baseline fortran/build/openacc -t openacc,mpi`

4.2.1 Notable flags for `nsys profile`:

- -t (–trace) parameters: cublas, cuda, cudnn, nvtx, opengl, openacc, openmp, osrt, mpi, vulkan, none
  - `-t openmp,openacc`
- -b (–backtrace) parameters: fp, lbr, dwarf, none
  - `-b fp`
- –cuda-memory-usage parameters: true, false
  - `--cuda-memory-usage=true`
- –mpi-impl parameters: openmpi, mpich
  - `--mpi-impl=openmpi`
- -o
  - `-o myreport`
  - Names the generated profiling report
- –stats
  - `--stats=true`
  - Generate data file to analyze within the CLI
  - Takes time to generate
- -h: help with explanations for all `nsys` commands plus sub commands
  - `nsys -h`
  - `nsys profile -h`

Some of these options can add significant profiler overhead to your application.

Additional options for CLI profiling can be found on the NVidia NSight CLI documentation:

https://docs.nvidia.com/nsight-systems/2020.3/profiling/index.html#cli-installing

4.3 Launching the profiler on Casper

[ ]:
```
# Comment to prevent repeat runs while testing
# qsub pbs/pbs_miniweather_baseline.sh
```

You will see a `.qdrep` file after this job has finished.

4.4 Quick analysis via CLI

[2]:
```
nsys stats reports/miniweather_baseline.qdrep | grep -v "SKIPPED"
```

```
Using reports/miniweather_baseline.sqlite for SQL queries.
Running [/glade/u/apps/dav/opt/cuda/11.4.0/nsight-systems-2021.2.4/target-
linux-x64/reports/cudaapisum.py reports/miniweather_baseline.sqlite]…

 Time(%)  Total Time (ns)  Num Calls    Average      Minimum     Maximum
StdDev            Name
 -------  ---------------  ---------  ------------  ----------  ----------
---------  --------------------
    98.9  311,045,408,877    829,502     374,978.5         480   4,110,831
672,951.6  cuStreamSynchronize
     0.6    1,952,049,331    414,751       4,706.6       2,995   1,276,614
3,407.5  cuLaunchKernel
     0.1      360,215,231     92,188       3,907.4       2,308     392,225
2,940.6  cuMemcpyHtoDAsync_v2
     0.1      353,236,646     92,374       3,824.0       2,242   1,474,974
6,696.5  cuMemcpyDtoHAsync_v2
     0.1      328,508,188     46,186       7,112.7       1,109   1,278,036
59,376.3  cuCtxSynchronize
     0.1      321,497,397    139,066       2,311.8       1,171     207,400
952.4  cuEventRecord
     0.0       57,625,752          2  28,812,876.0  28,797,402  28,828,350
21,883.5  cuMemHostAlloc
     0.0       54,836,720     92,856         590.6         423     250,978
1,077.5  cuEventSynchronize
     0.0        3,205,420         31     103,400.6       1,358   1,425,718
337,926.8  cuMemAlloc_v2
     0.0        1,062,003          2     531,001.5       6,018   1,055,985
742,438.8  cuMemAllocHost_v2
     0.0          378,549          1     378,549.0     378,549     378,549
0.0  cuModuleLoadDataEx
```

```
     0.0          45,674          4      11,418.5       2,677      25,152
10,628.6  cuMemsetD32Async
     0.0          34,418         26       1,323.8         285      12,870
2,521.0  cuEventCreate
     0.0          13,140          1      13,140.0      13,140      13,140
0.0  cuStreamCreate
     0.0           2,444          1       2,444.0       2,444       2,444
0.0  cuInit

Running [/glade/u/apps/dav/opt/cuda/11.4.0/nsight-systems-2021.2.4/target-
linux-x64/reports/gpukernsum.py reports/miniweather_baseline.sqlite]…

 Time(%)  Total Time (ns)  Instances    Average     Minimum    Maximum    StdDev
Name
 -------  ---------------  ---------  -----------  ---------  ---------
--------  ----------------------------
    34.8  107,763,085,030     46,083  2,338,456.4  2,319,212  2,593,737
7,323.0  compute_tendencies_z_369_gpu
    22.5   69,648,080,748     46,083  1,511,361.7  1,493,267  1,812,785
16,927.7  compute_tendencies_x_278_gpu
    22.3   69,197,111,596     46,083  1,501,575.7  1,383,189  1,749,617
22,199.8  compute_tendencies_z_334_gpu
    10.9   33,900,055,922     92,166    367,815.2    357,149    503,131
5,227.3  semi_discrete_step_231_gpu
     9.1   28,296,053,744     46,083    614,023.7    602,971    748,570
3,340.8  compute_tendencies_x_308_gpu
     0.2      641,650,166     46,083     13,923.8     12,288     18,624
372.0  set_halo_values_z_452_gpu
     0.1      288,514,354     46,083      6,260.8      5,408     14,560
261.2  set_halo_values_x_395_gpu
     0.1      281,057,530     46,083      6,098.9      5,855     14,624
204.3  set_halo_values_x_418_gpu
     0.0          172,575          2     86,287.5     78,335     94,240
11,246.5  reductions_871_gpu
     0.0           19,680          2      9,840.0      9,280     10,400
792.0  reductions_871_gpu__red

Running [/glade/u/apps/dav/opt/cuda/11.4.0/nsight-systems-2021.2.4/target-
linux-x64/reports/gpumemtimesum.py reports/miniweather_baseline.sqlite]…

 Time(%)  Total Time (ns)  Operations  Average  Minimum   Maximum    StdDev
Operation
 -------  ---------------  ----------  -------  -------  ---------  --------
------------------
    54.0      471,292,258      92,188  5,112.3      864  1,370,900   6,557.4
[CUDA memcpy HtoD]
    46.0      402,122,357      92,374  4,353.2      896  1,272,853  42,145.0
[CUDA memcpy DtoH]
```

```
    0.0           3,295           4    823.8      768         864      40.2
[CUDA memset]

Running [/glade/u/apps/dav/opt/cuda/11.4.0/nsight-systems-2021.2.4/target-
linux-x64/reports/gpumemsizesum.py reports/miniweather_baseline.sqlite]…

     Total      Operations  Average  Minimum   Maximum    StdDev      Operation
 -------------  ----------  -------  -------  ----------  -------
------------------
 4,640,115.031      92,374   50.232    0.008  16,383.906  543.091  [CUDA memcpy
DtoH]
 2,982,487.461      92,188   32.352    0.125  16,384.000   76.168  [CUDA memcpy
HtoD]
         0.031           4    0.008    0.008       0.008    0.000  [CUDA memset]

Running [/glade/u/apps/dav/opt/cuda/11.4.0/nsight-systems-2021.2.4/target-
linux-x64/reports/osrtsum.py reports/miniweather_baseline.sqlite]…

 Time(%)  Total Time (ns)  Num Calls      Average        Minimum        Maximum
StdDev                Name
 -------  ---------------  ---------  ---------------  -----------
---------------  ----------------  ------------------------
    33.3  987,640,768,804        107  9,230,287,558.9        1,161
328,766,957,311  50,003,634,104.2  epoll_wait
    33.3  986,559,928,564     61,088     16,149,815.5        1,030
329,169,307,528   1,883,234,444.9  poll
    22.2  659,270,247,276        167  3,947,726,031.6   42,623,923
329,639,298,880  25,355,113,168.4  select
    11.1  328,094,330,739        656    500,143,796.9  500,041,057
500,223,002          18,299.5  pthread_cond_timedwait
     0.1    1,724,974,920        708      2,436,405.3        3,076
9,646,299       2,564,917.6  pwrite
     0.0      466,505,517     93,319          4,999.0        1,002
28,009,889         168,578.5  ioctl
     0.0      288,069,694      1,340        214,977.4        1,961
69,167,686       2,062,232.5  open
     0.0      162,721,610      4,757         34,206.8        1,000
136,774,462       1,983,215.8  read
     0.0       83,935,099         29      2,894,313.8       20,654
36,350,795       8,714,027.3  pthread_cond_wait
     0.0       34,064,040          1     34,064,040.0   34,064,040
34,064,040               0.0  truncate
     0.0       29,334,013      5,797          5,060.2        1,260
34,663           1,278.6  openat
     0.0       18,916,304          6      3,152,717.3    1,933,235
7,221,737       2,008,494.4  posix_fallocate
     0.0       14,754,930        362         40,759.5        1,178
94,498          26,597.0  write
```

0.0	13,730,760	800	17,163.5	1,002	511,828	18,029.7	fgets
0.0	13,530,359	1,824	7,418.0	1,034	45,428	4,168.8	fcntl
0.0	10,610,390	100	106,103.9	67,392	1,143,113	119,644.2	pread
0.0	7,407,214	79	93,762.2	4,074	2,069,019	289,674.7	mmap64
0.0	2,023,120	7	289,017.1	87,286	704,188	273,939.8	munmap
0.0	1,743,626	11	158,511.5	135,661	183,676	16,536.6	pthread_create
0.0	1,641,874	179	9,172.5	1,727	119,641	13,449.5	fopen
0.0	1,625,239	9	180,582.1	17,973	1,009,903	315,363.9	sem_timedwait
0.0	1,365,161	229	5,961.4	2,627	109,471	11,292.9	mmap
0.0	1,303,347	30	43,444.9	23,810	100,760	17,623.9	pthread_mutex_lock
0.0	1,286,497	6	214,416.2	61,355	450,401	132,760.0	pthread_join
0.0	1,139,803	7	162,829.0	161,679	165,544	1,313.1	usleep
0.0	998,091	169	5,905.9	2,644	45,790	4,638.6	fclose
0.0	982,047	1	982,047.0	982,047	982,047	0.0	fork
0.0	622,275	373	1,668.3	1,052	9,440	1,018.0	socket
0.0	604,814	99	6,109.2	3,066	22,168	2,747.4	open64
0.0	561,750	4	140,437.5	1,537	555,991	277,036.1	recv
0.0	428,804	23	18,643.7	1,194	40,633	11,147.7	writev
0.0	299,150	1	299,150.0	299,150	299,150	0.0	ftruncate
0.0	252,765	55	4,595.7	1,000	16,299	4,538.4	recvmsg
0.0	191,132	57	3,353.2	1,004	10,467	2,386.2	mprotect
0.0	173,182	87	1,990.6	1,005	10,798	1,223.1	epoll_ctl
0.0	142,270	36	3,951.9	1,346	17,523	4,673.2	sendmsg
0.0	142,268	29	4,905.8	2,285	21,020	4,158.8	pthread_cond_broadcast

0.0	141,888	13	10,914.5	1,926	73,755	19,303.3	shmget
0.0	132,897	17	7,817.5	1,880	20,340	6,390.6	fread
0.0	107,304	45	2,384.5	1,010	12,512	2,311.5	fwrite
0.0	93,083	8	11,635.4	1,010	39,867	12,875.3	listen
0.0	66,044	5	13,208.8	5,001	32,386	11,680.4	shutdown
0.0	57,357	2	28,678.5	8,355	49,002	28,741.8	connect
0.0	56,751	6	9,458.5	5,408	14,082	3,481.1	getdelim
0.0	47,714	9	5,301.6	1,194	12,703	3,894.4	fgetc
0.0	34,170	16	2,135.6	1,094	3,660	700.0	bind
0.0	32,437	3	10,812.3	1,544	20,221	9,339.3	send
0.0	24,649	5	4,929.8	4,294	5,844	653.9	socketpair
0.0	17,315	6	2,885.8	2,188	3,803	628.6	pipe
0.0	16,192	2	8,096.0	7,430	8,762	941.9	shmdt
0.0	11,878	5	2,375.6	1,129	6,104	2,104.0	sigaction
0.0	9,314	3	3,104.7	2,156	4,708	1,396.3	pthread_rwlock_trywrlock
0.0	8,542	2	4,271.0	3,595	4,947	956.0	shmat
0.0	8,515	2	4,257.5	3,651	4,864	857.7	accept
0.0	6,379	2	3,189.5	2,643	3,736	772.9	process_vm_writev
0.0	5,661	1	5,661.0	5,661	5,661	0.0	pipe2
0.0	4,542	4	1,135.5	1,016	1,339	148.9	shmctl
0.0	3,075	1	3,075.0	3,075	3,075	0.0	pthread_mutex_trylock
0.0	1,470	1	1,470.0	1,470	1,470	0.0	recvfrom

This output will look familiar if you have used `nvprof` to profile codes previously.

4.5 Timeline Analysis via Nsight Systems GUI

4.5.1 Transfer or view your report

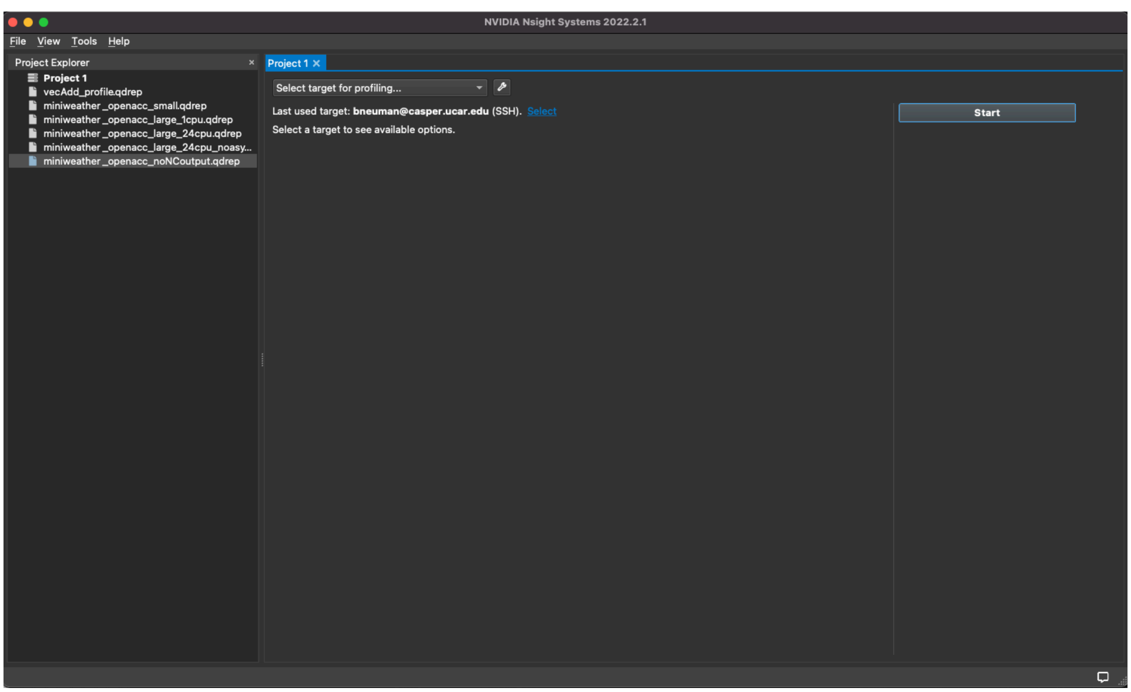
Reports for analysis are located in the `reports` folder. For our baseline we will use the generated report:

`miniweather_baseline.qdrep`

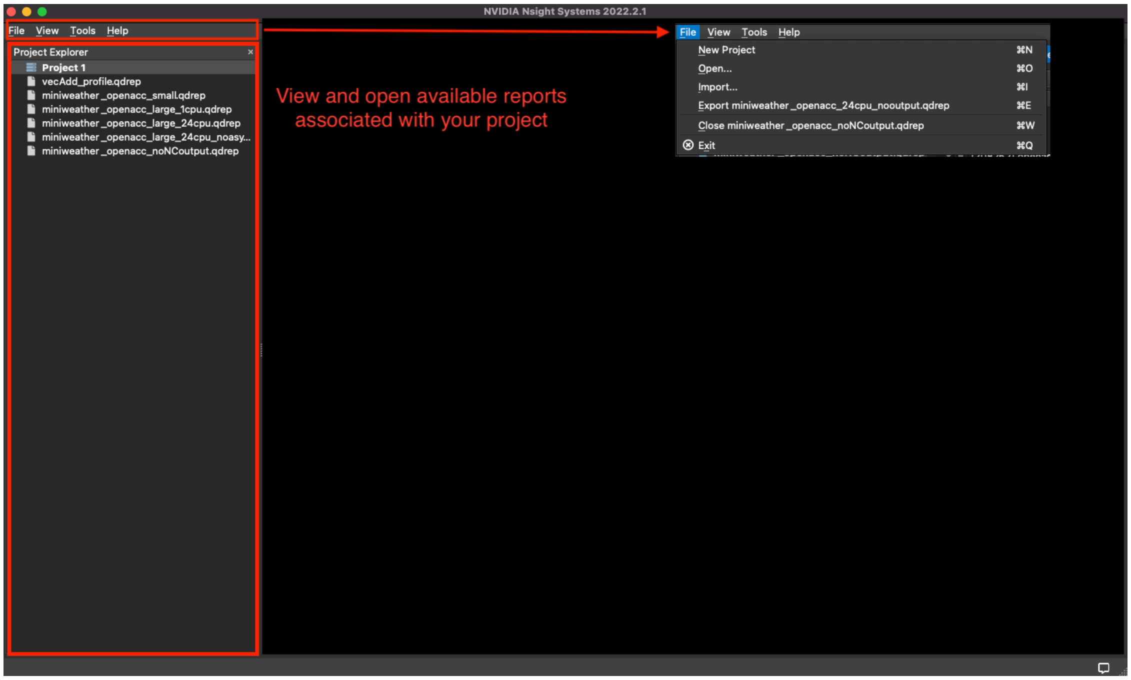
1. Transfer the `.qdrep` file to your local machine and load in into your local installation of the NSight Systems application
   - Download the file by right clicking and selecting `Download` on the JupyterHub browser on the left.
2. Launch a X or VNC session on a GP100 GPU node on Casper. Launch `nsight-nsys`.
   - KB Article to set up VNC: https://kb.ucar.edu/display/RC/Using+remote+desktops+on+Casper+wit
   - X session works but can be slow

4.5.2 Nsight Systems GUI

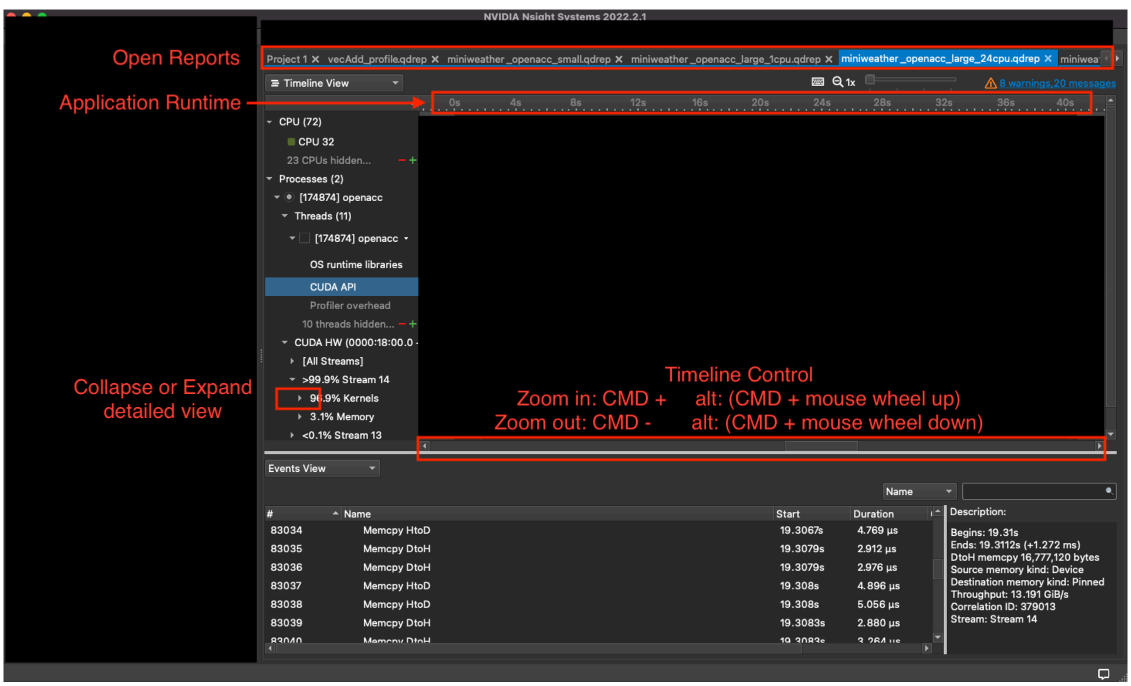
Open the file in the NSight Systems application. Below is the default view upon opening the application.



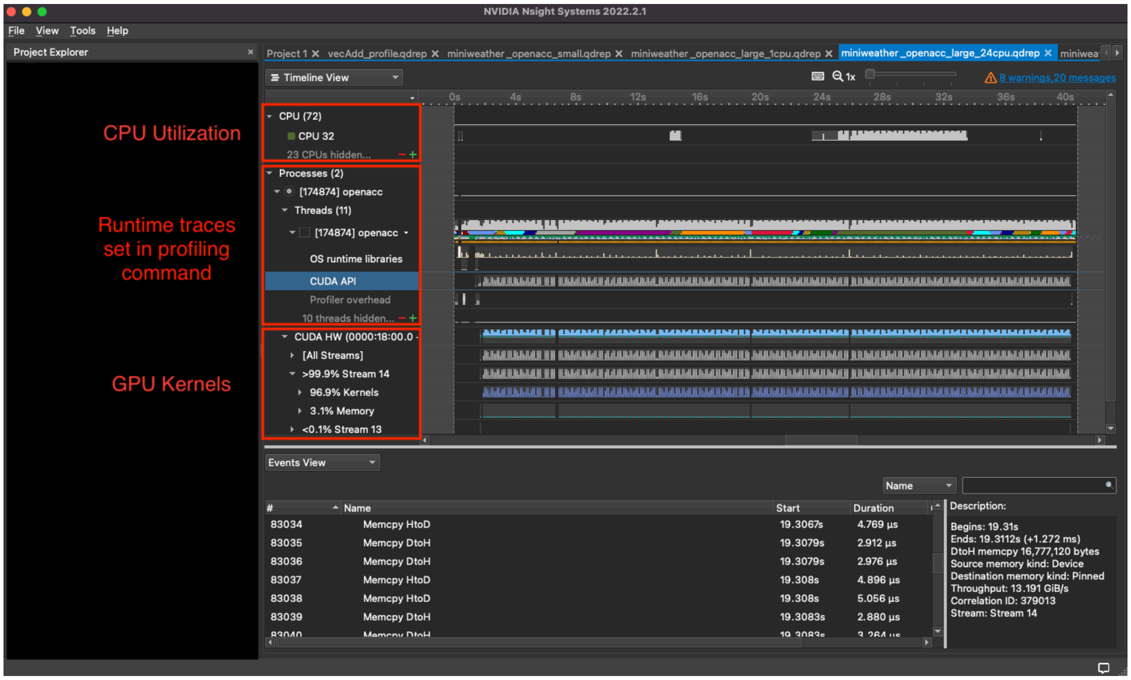
4.5.3 Projects



4.5.4 Navigation

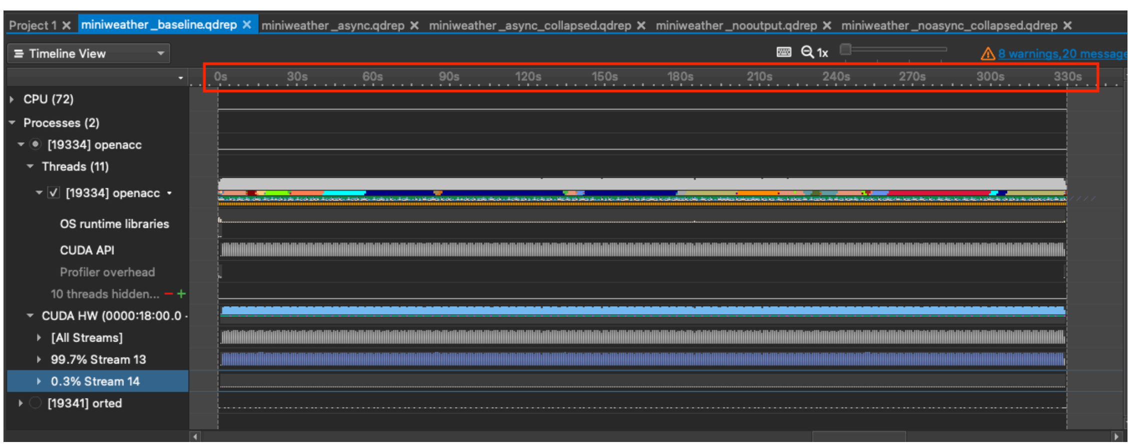


4.5.5 Event Descriptions



4.6 Baseline Timeline View

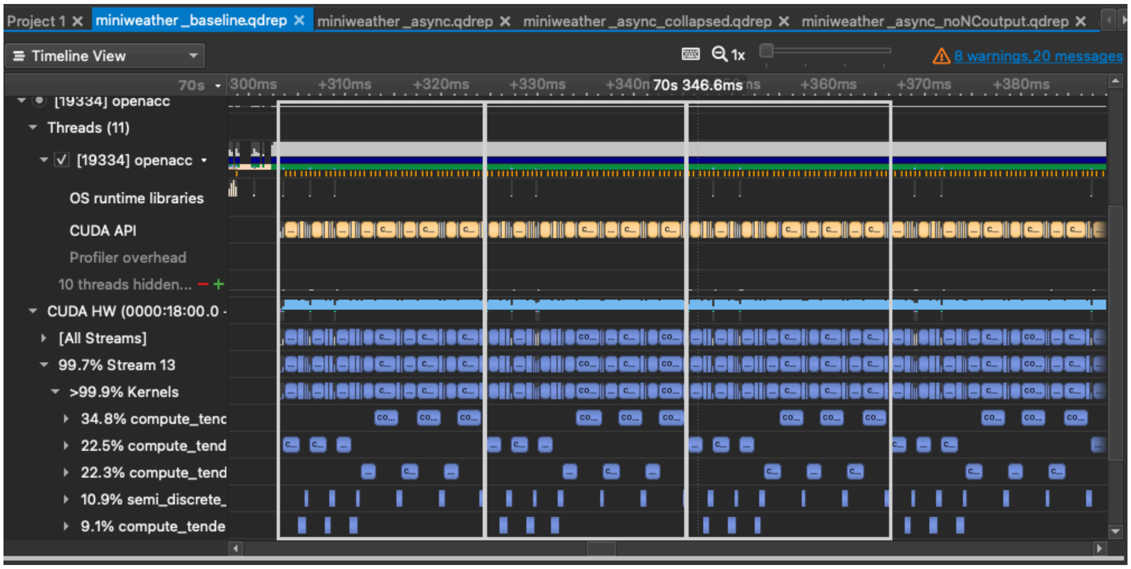
`miniweather_baseline.qdrep`



4.7 Patterns, Gaps, Walltime and Kernels

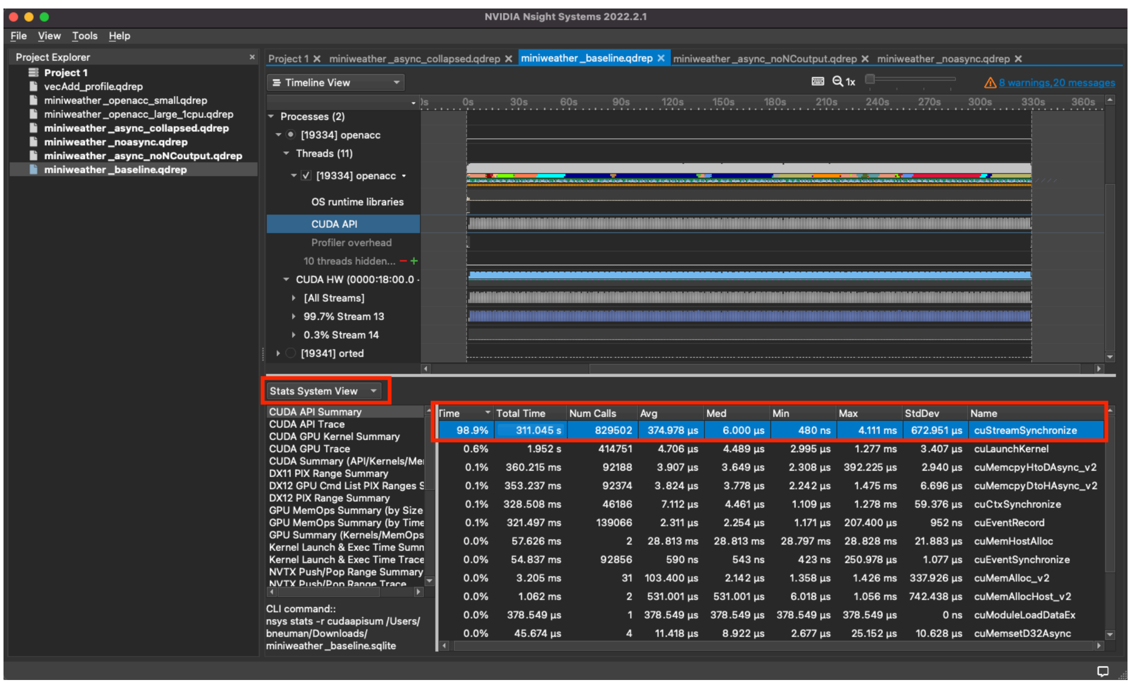
We can find instruction patterns of interest, sections where the GPU is idle, and also view details on which kernel is running at a given time using the Timeline view. Below is an example of a repeated pattern found in the baseline report. It will be useful to note that the time to complete this repeated pattern is about 20ms.

Note that we zoomed into the timeline significantly.



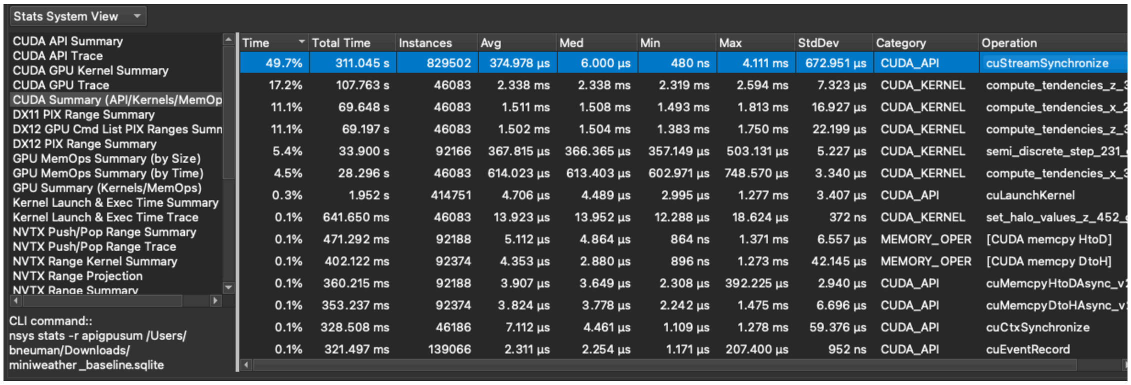
4.7.1 Stats View

Quickly find CUDA API and GPU Kernel instruction runtimes. This is a good place to get ideas on how to make improvements.



4.8 Asynchronous Loops Profile

I'm using the information about 50% of our runtime in `cuStreamSynchronize` to make changes to the existing `!$acc loop parallel` sections.



Stats System View

CUDA API Summary
CUDA API Trace
CUDA GPU Kernel Summary
CUDA GPU Trace
CUDA Summary (API/Kernels/MemOp
DX11 PIX Range Summary
DX12 GPU Cmd List PIX Ranges Sumn
DX12 PIX Range Summary
GPU MemOps Summary (by Size)
GPU MemOps Summary (by Time)
GPU Summary (Kernels/MemOps)
Kernel Launch & Exec Time Summary
Kernel Launch & Exec Time Trace
NVTX Push/Pop Range Summary
NVTX Push/Pop Range Trace
NVTX Range Kernel Summary
NVTX Range Projection
NVTX Range Summary

CLI command::
nsys stats -r apigpusum /Users/bneuman/Downloads/miniweather_baseline.sqlite

Time	Total Time	Instances	Avg	Med	Min	Max	StdDev	Category	Operation
49.7%	311.045 s	829502	374.978 µs	6.000 µs	480 ns	4.111 ms	672.951 µs	CUDA_API	cuStreamSynchronize
17.2%	107.763 s	46083	2.338 ms	2.338 ms	2.319 ms	2.594 ms	7.323 µs	CUDA_KERNEL	compute_tendencies_z_3
11.1%	69.648 s	46083	1.511 ms	1.508 ms	1.493 ms	1.813 ms	16.927 µs	CUDA_KERNEL	compute_tendencies_x_2
11.1%	69.197 s	46083	1.502 ms	1.504 ms	1.383 ms	1.750 ms	22.199 µs	CUDA_KERNEL	compute_tendencies_z_3
5.4%	33.900 s	92166	367.815 µs	366.365 µs	357.149 µs	503.131 µs	5.227 µs	CUDA_KERNEL	semi_discrete_step_231_
4.5%	28.296 s	46083	614.023 µs	613.403 µs	602.971 µs	748.570 µs	3.340 µs	CUDA_KERNEL	compute_tendencies_x_3
0.3%	1.952 s	414751	4.706 µs	4.489 µs	2.995 µs	1.277 ms	3.407 µs	CUDA_API	cuLaunchKernel
0.1%	641.650 ms	46083	13.923 µs	13.952 µs	12.288 µs	18.624 µs	372 ns	CUDA_KERNEL	set_halo_values_z_452_
0.1%	471.292 ms	92188	5.112 µs	4.864 µs	864 ns	1.371 ms	6.557 µs	MEMORY_OPER	[CUDA memcpy HtoD]
0.1%	402.122 ms	92374	4.353 µs	2.880 µs	896 ns	1.273 ms	42.145 µs	MEMORY_OPER	[CUDA memcpy DtoH]
0.1%	360.215 ms	92188	3.907 µs	3.649 µs	2.308 µs	392.225 µs	2.940 µs	CUDA_API	cuMemcpyHtoDAsync_v
0.1%	353.237 ms	92374	3.824 µs	3.778 µs	2.242 µs	1.475 ms	6.696 µs	CUDA_API	cuMemcpyDtoHAsync_v
0.1%	328.508 ms	46186	7.112 µs	4.461 µs	1.109 µs	1.278 ms	59.376 µs	CUDA_API	cuCtxSynchronize
0.1%	321.497 ms	139066	2.311 µs	2.254 µs	1.171 µs	207.400 µs	952 ns	CUDA_API	cuEventRecord

Modify the ACC loops to perform asynchronously. OpenACC will no longer wait for the flagged loop to finish before launching another and should pipeline the loop iterations. We need to include `!$acc wait` flags for sections to allow individual loop sections to finish before operating on a different loop.

```
!Apply the tendencies to the fluid state
!$acc parallel loop async
do ll = 1 , NUM_VARS
  do k = 1 , nz
    do i = 1 , nx
      if (data_spec_int == DATA_SPEC_GRAVITY_WAVES) then
        x = (i_beg-1 + i-0.5_rp) * dx
        z = (k_beg-1 + k-0.5_rp) * dz
```

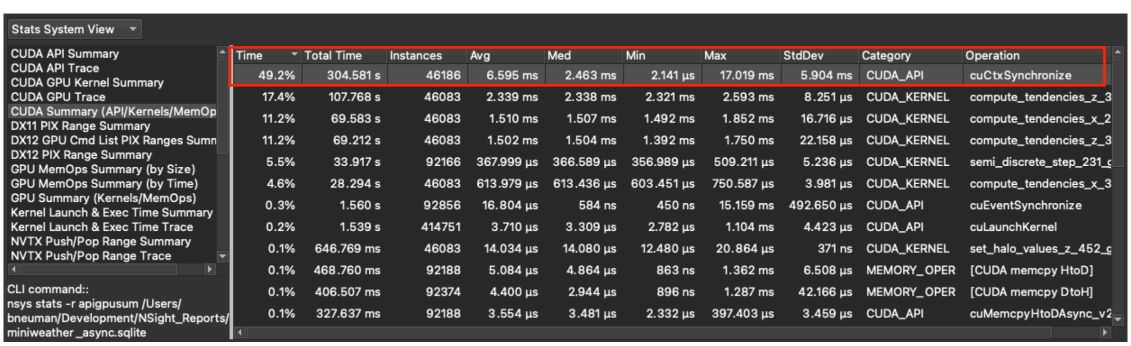
Recompile and profile the code again to see the changes you've made. Launch the script with the new `nsys profile` command on Casper.

```
nsys profile -o miniweather_async fortran/build/openacc -t openacc,mpi
```

4.8.1 Asynchronous Analysis

`miniweather_async.qdrep`

Not a significant change. The command `CuStreamSynchronize` changed to `CuCtxSynchronize` but still takes almost 50% of the runtime.



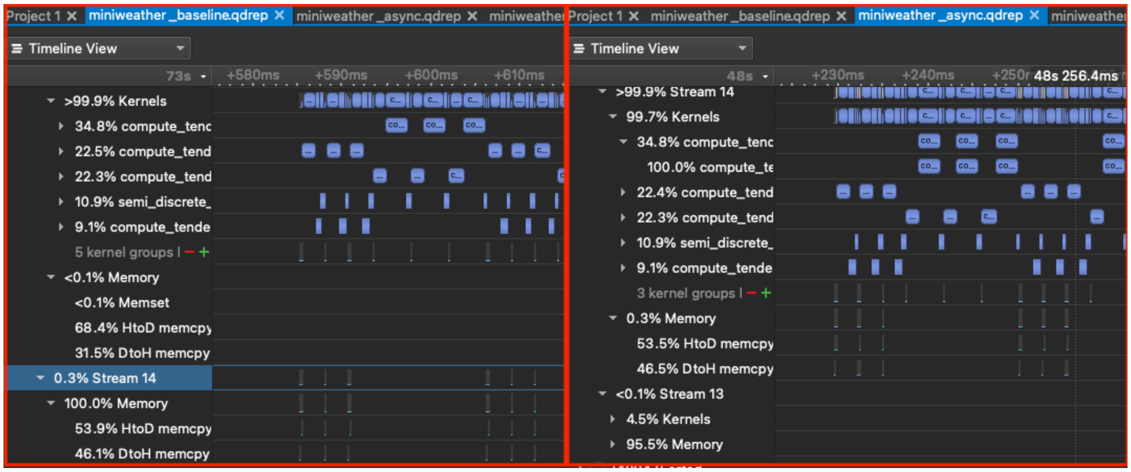
Stats System View

CUDA API Summary
CUDA API Trace
CUDA GPU Kernel Summary
CUDA GPU Trace
CUDA Summary (API/Kernels/MemOp
DX11 PIX Range Summary
DX12 GPU Cmd List PIX Ranges Sumn
DX12 PIX Range Summary
GPU MemOps Summary (by Size)
GPU MemOps Summary (by Time)
GPU Summary (Kernels/MemOps)
Kernel Launch & Exec Time Summary
Kernel Launch & Exec Time Trace
NVTX Push/Pop Range Summary
NVTX Push/Pop Range Trace

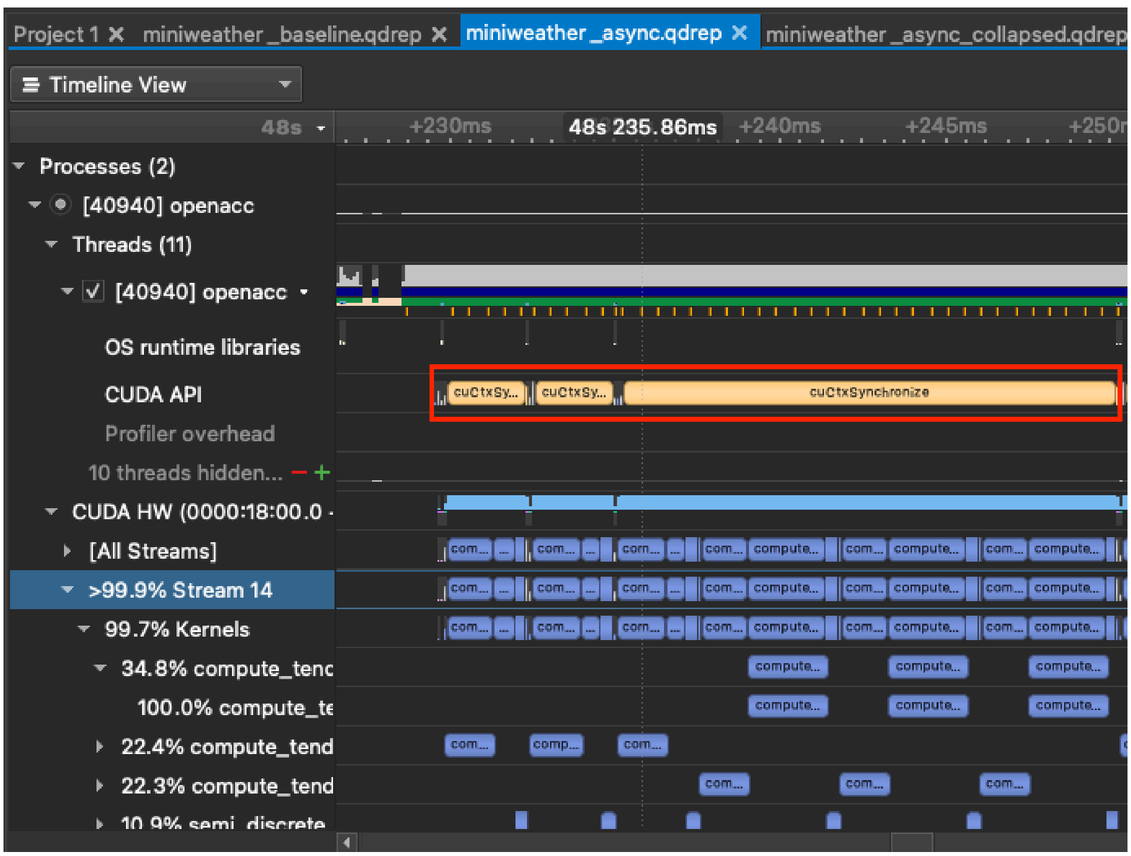
CLI command::
nsys stats -r apigpusum /Users/bneuman/Development/NSight_Reports/miniweather_async.sqlite

Time	Total Time	Instances	Avg	Med	Min	Max	StdDev	Category	Operation
49.2%	304.581 s	46186	6.595 ms	2.463 ms	2.141 µs	17.019 ms	5.904 ms	CUDA_API	cuCtxSynchronize
17.4%	107.768 s	46083	2.339 ms	2.338 ms	2.321 ms	2.593 ms	8.251 µs	CUDA_KERNEL	compute_tendencies_z_3
11.2%	69.583 s	46083	1.510 ms	1.507 ms	1.492 ms	1.852 ms	16.716 µs	CUDA_KERNEL	compute_tendencies_x_2
11.2%	69.212 s	46083	1.502 ms	1.504 ms	1.392 ms	1.750 ms	22.158 µs	CUDA_KERNEL	compute_tendencies_z_3
5.5%	33.917 s	92166	367.999 µs	366.589 µs	356.989 µs	509.211 µs	5.236 µs	CUDA_KERNEL	semi_discrete_step_231_g
4.6%	28.294 s	46083	613.979 µs	613.436 µs	603.451 µs	750.587 µs	3.981 µs	CUDA_KERNEL	compute_tendencies_x_3
0.3%	1.560 s	92856	16.804 µs	584 ns	450 ns	15.159 ms	492.650 µs	CUDA_API	cuEventSynchronize
0.2%	1.539 s	414751	3.710 µs	3.309 µs	2.782 µs	1.104 ms	4.423 µs	CUDA_API	cuLaunchKernel
0.1%	646.769 ms	46083	14.034 µs	14.080 µs	12.480 µs	20.864 µs	371 ns	CUDA_KERNEL	set_halo_values_z_452_g
0.1%	468.760 ms	92188	5.084 µs	4.864 µs	863 ns	1.362 ms	6.508 µs	MEMORY_OPER	[CUDA memcpy HtoD]
0.1%	406.507 ms	92374	4.400 µs	2.944 µs	896 ns	1.287 ms	42.166 µs	MEMORY_OPER	[CUDA memcpy DtoH]
0.1%	327.637 ms	92188	3.554 µs	3.481 µs	2.332 µs	397.403 µs	3.459 µs	CUDA_API	cuMemcpyHtoDAsync_v2

We can see that the memory operations are launching from within the same stream now, suggesting that there is pipelining.



We're still spending a lot of time in `CuStreamSynchronize`. Can we try to improve our parallelzation of loops?



4.9 Collapsed Loops Profile

Modify the ACC loops to perform asynchronously and also collapse loops based on how deep the loop structure is.

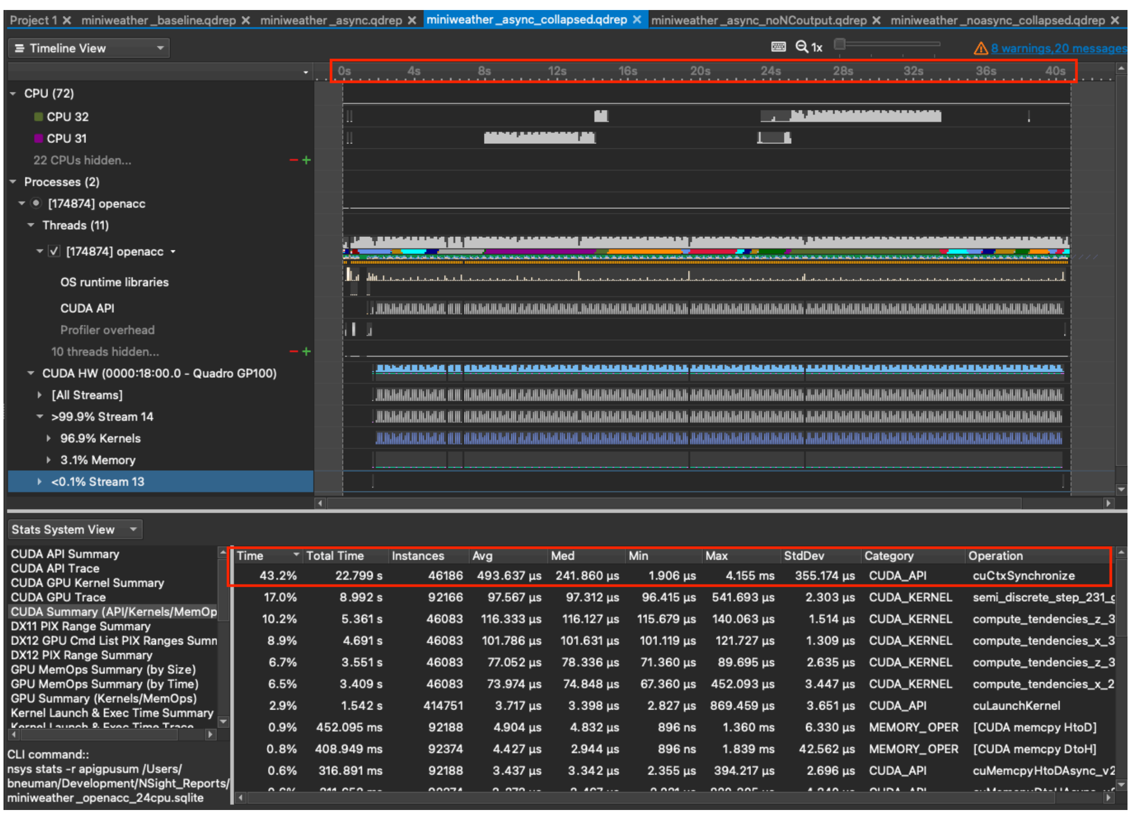
```
!Apply the tendencies to the fluid state
!$acc parallel loop collapse(3) async
do ll = 1 , NUM_VARS
  do k = 1 , nz
    do i = 1 , nx
      if (data_spec_int == DATA_SPEC_GRAVITY_WAVES) then
        x = (i_beg-1 + i-0.5_rp) * dx
        z = (k_beg-1 + k-0.5_rp) * dz
```

Recompile and profile the code again to see the changes you've made. Launch the script with the new `nsys profile` command on Casper.

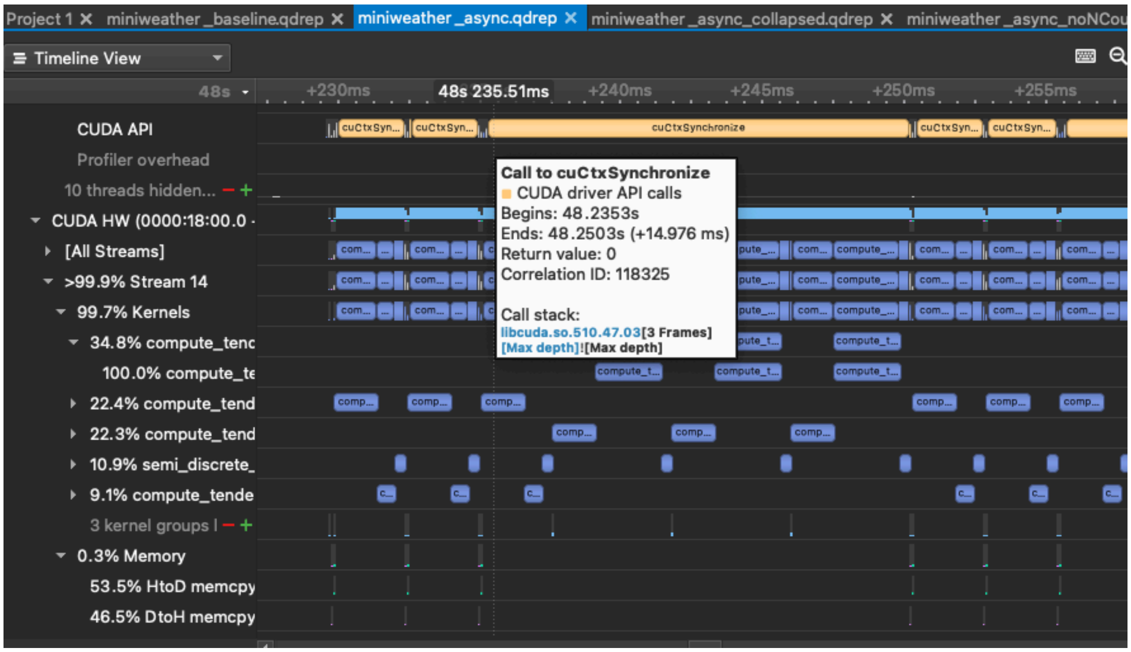
```
nsys profile -o miniweather_async_collapsed fortran/build/openacc -t openacc,mpi
```

4.9.1 Collapsed Loops Analysis

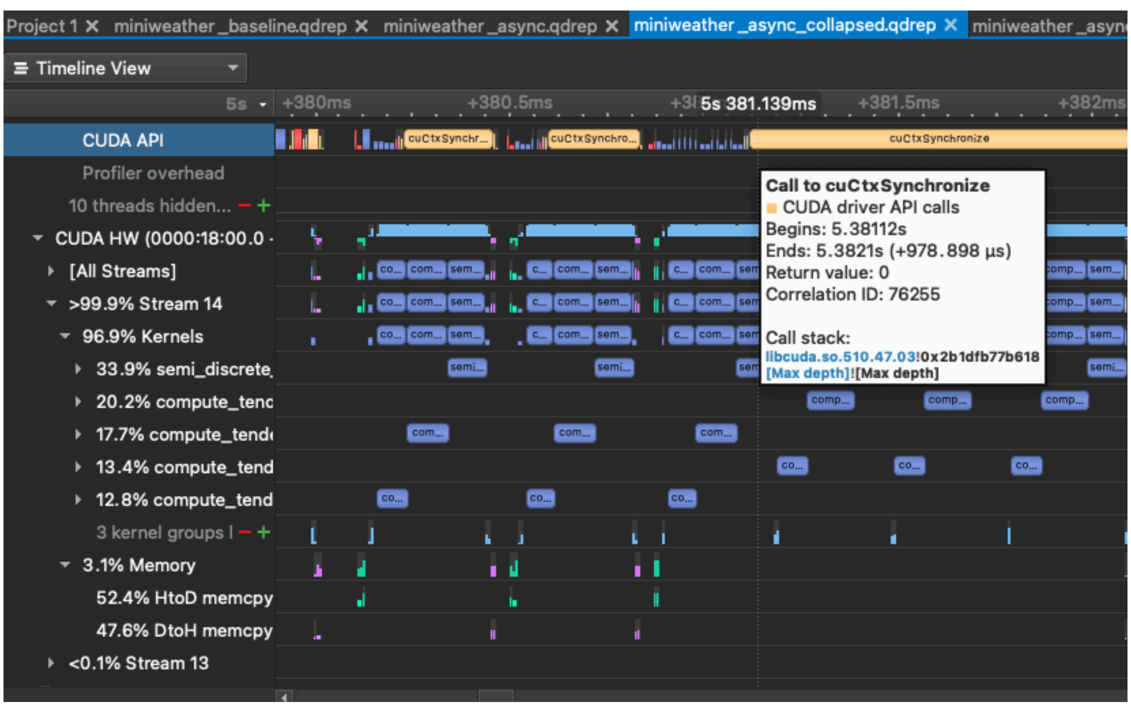
`miniweather_async_collapsed.qdrep`



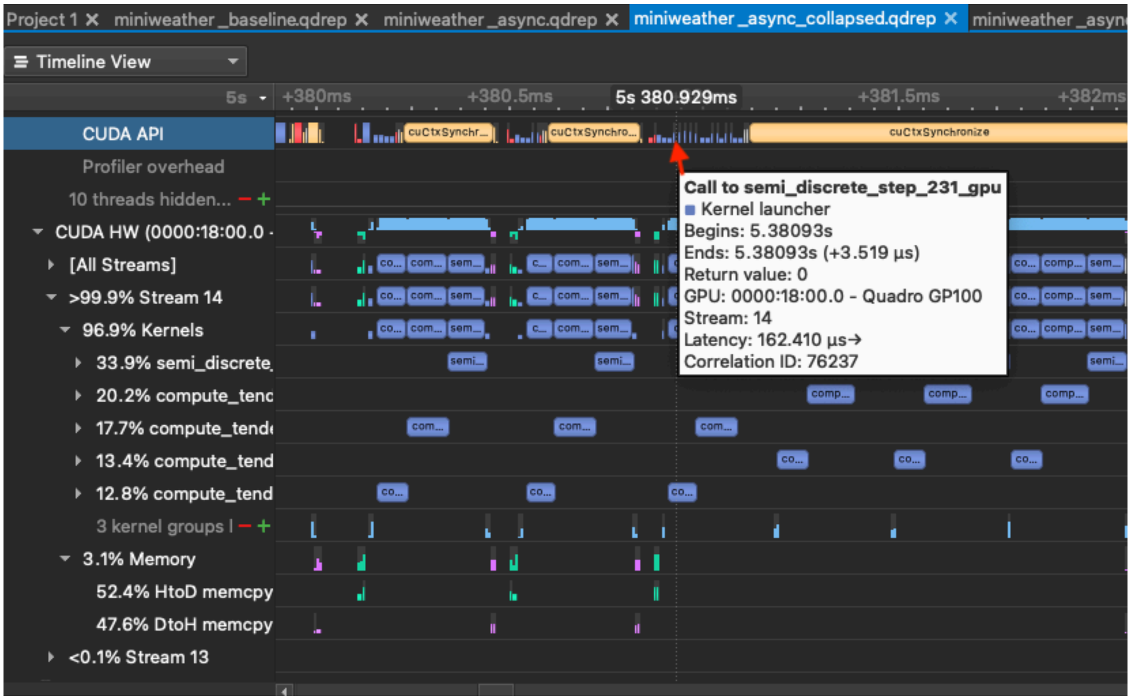
Here is the `CuCtxSynchronize` wait time for the Async profile. 15 seconds spent waiting to launch a new round of instructions.



The same `CuCtxSynchronize` with the Collapsed loops profile. Down to 1 second.

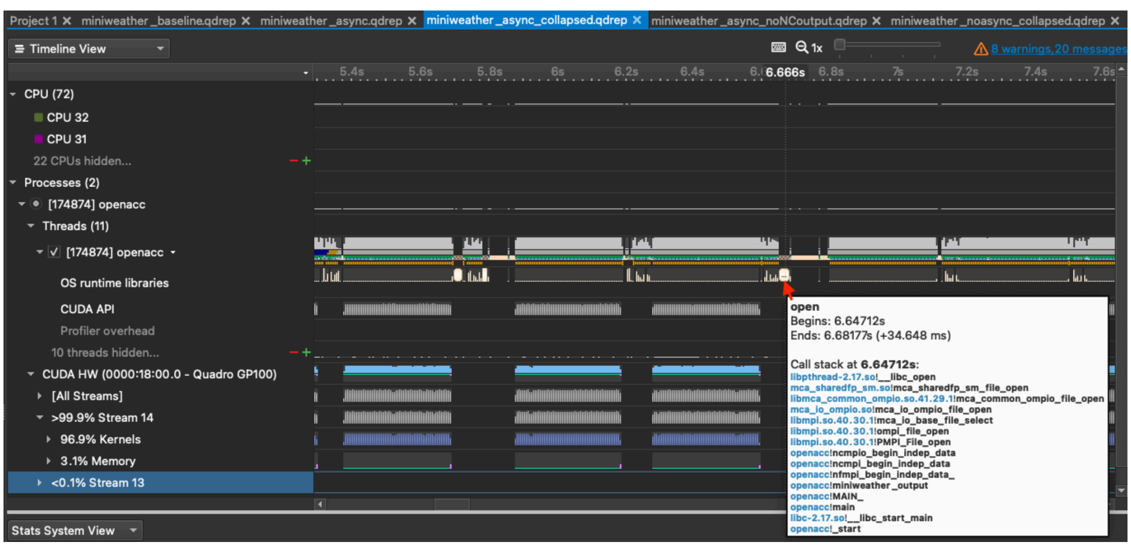


You can also spot additional calls to kernels in between synchronization, so we've improved parallelism.



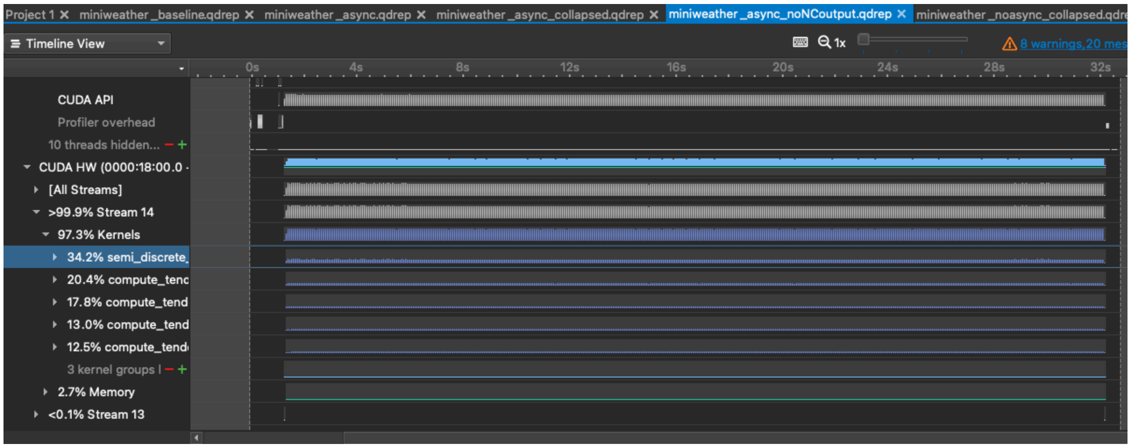
4.9.2 Output to file and I/O operations

After zooming into the timeline for the `miniweather_async_collapsed.qdrep` file you will notice that there is an operation that occurs between kernel operations frequently.



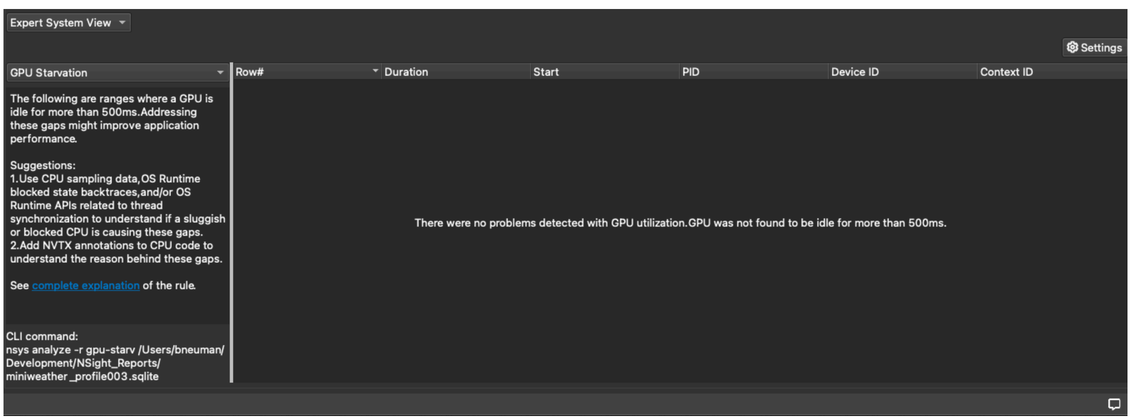
Hovering over the operation gives us the call stack where we can identify the IO operation. Here we see it coming from the `_output` subroutine. Recording the results of your simulation is important but let's see what sort of performance we can get by eliminating the call to `output`.

Compare the full timeline view of the `miniweather_async_collapsed.qdrep` and the `miniweather_nooutput.qdrep`. You'll notice the bubbles are gone and the walltime is 32s compared to 41s (1.28x). Reducing idle time on the GPU and also reducing memory transfers between host and device give us a good performance gain.



4.9.3 Expert View

Good spot to go for general recommendations based on common GPU problems and can provide hints on where to start optimizing.



4.9.4 Other profiling tools

There is a lot of profiling work being done in the deep learning and scientific computing spheres. There are other tools available to analyzing training time, visualization insight, and other DL/ML focused profilers: 1) DLProf: https://docs.nvidia.com/deeplearning/frameworks/dlprof-user-guide/ 2) Tensorboard: https://www.tensorflow.org/tensorboard/get_started 3) NVidia Tools Extension (NVTX) * NVIDIA Tools Extension (NVTX) is an API that allows for additional control for profiling your applications. NVTX can be particularly useful when you have a specific section of your code that you need to gather performance information on. It can also be a useful intermediate step between the higher level Nsight Systems view and the kernel optimization of Nsight Compute. * NVTX header file used and code marked to profile specific sections of your larger codebase * Jiri Kraus (our next workshop presenter) has a very good walkthrough of using

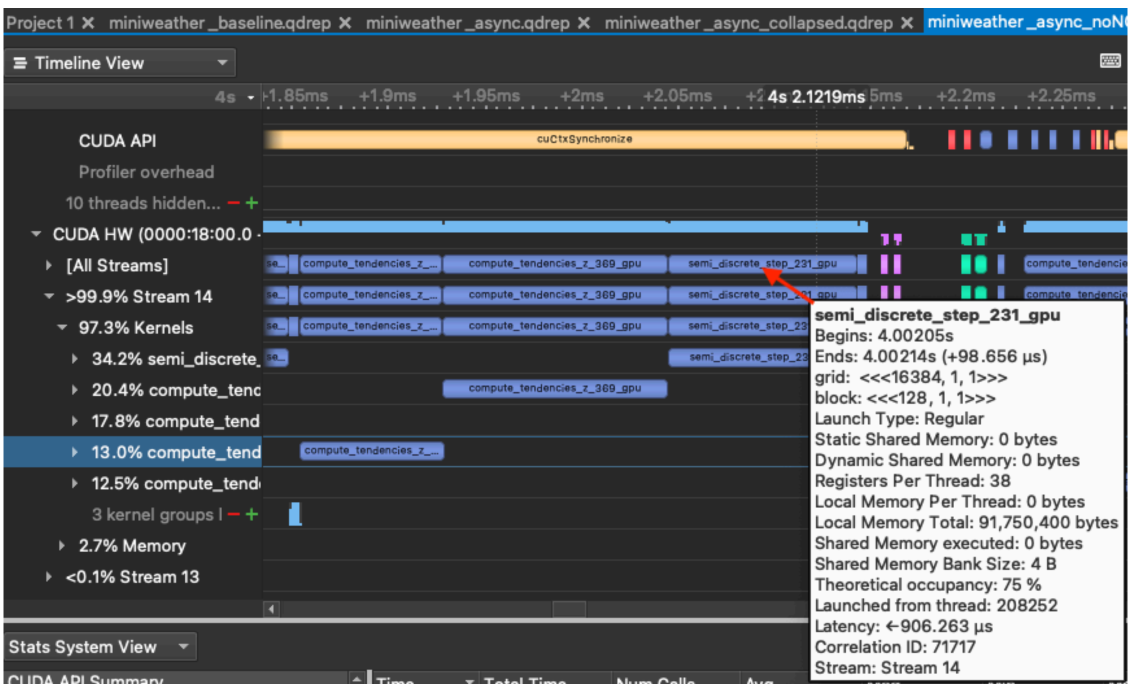
NVTX for C/C++: https://developer.nvidia.com/blog/cuda-pro-tip-generate-custom-application-profile-timelines-nvtx/

C Example:

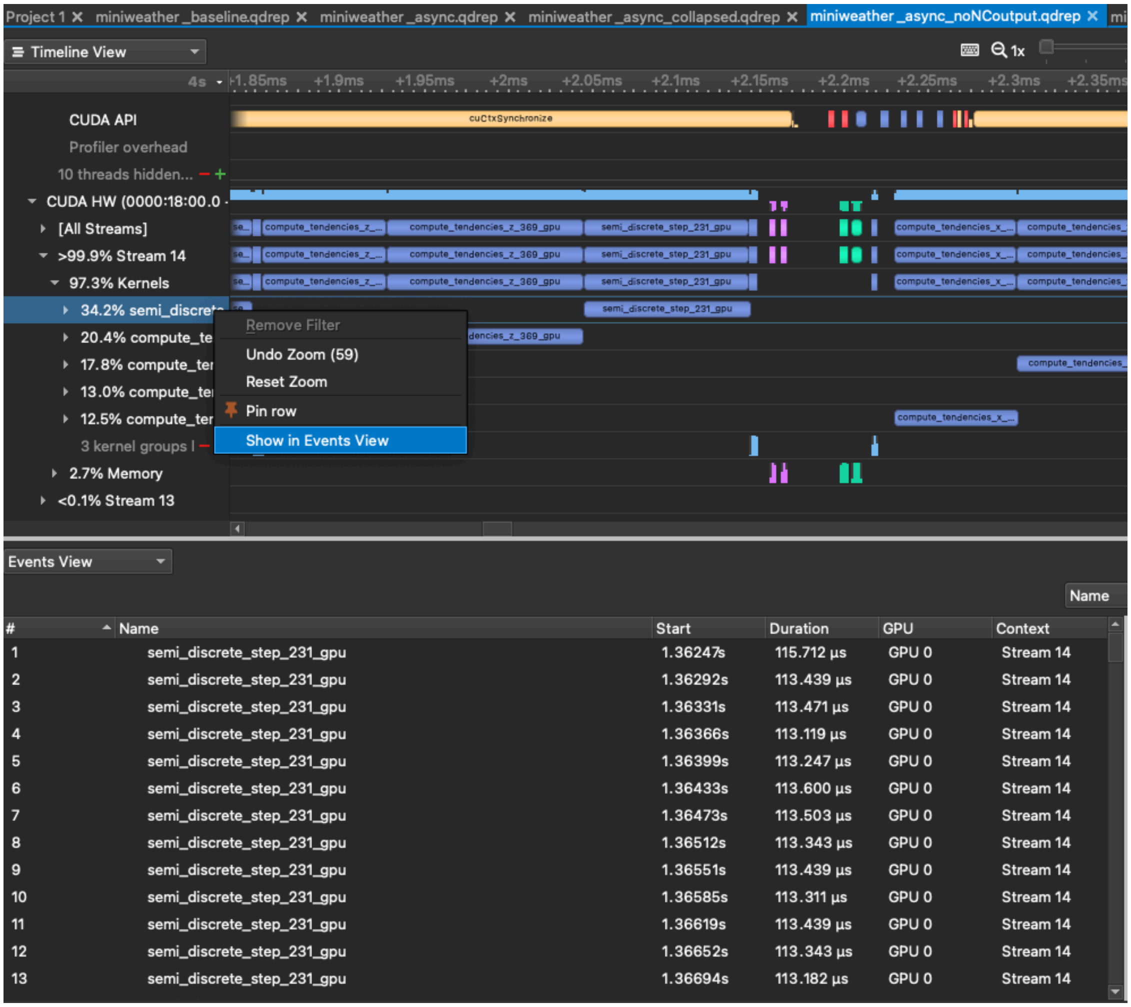
```c
#include "nvToolsExt.h"
...
void init_host_data( int n, double * x ) {
    nvtxRangePushA("init_host_data");
    //initialize x on host
    ...
    nvtxRangePop();
}
...
```

5 Launching Nsight Compute with Nsight Systems

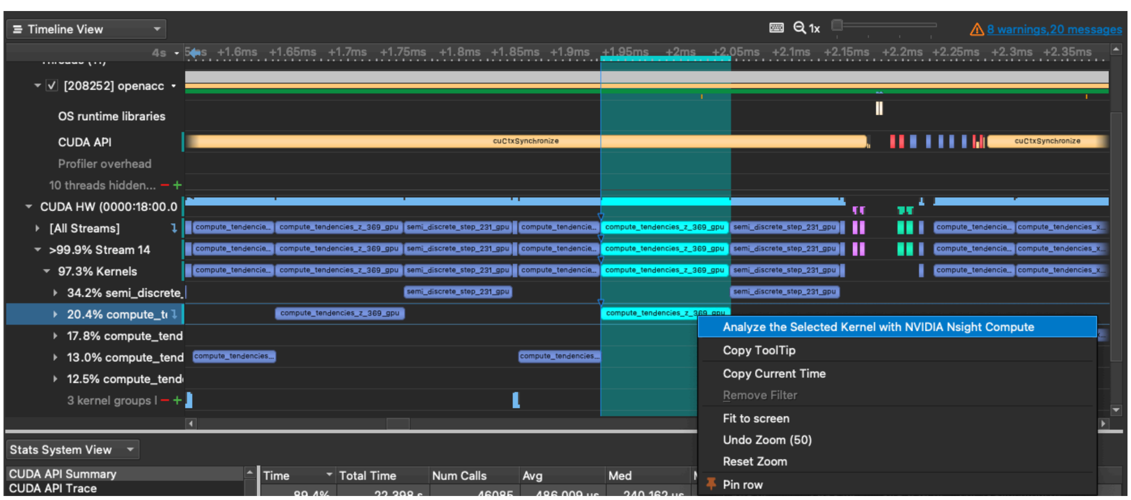
Information from hovering over a kernel launch instruction:



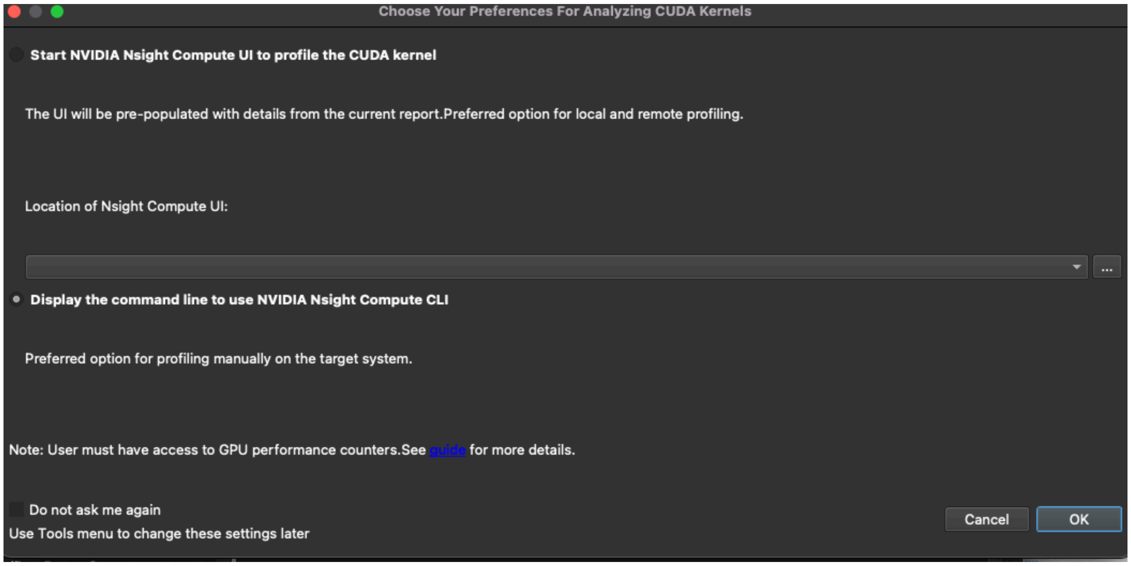
You can also right click on the kernel and see a textual timeline of all instances of that kernel in your application:



From here you can right click on the kernel launch instruction in the timeline and analyze it in Nsight Compute. Select `Analyze the Selected Kernel with NVIDIA Nsight Compute`:



Here is the window to launch Nsight Compute:



6 Resources

NVidia Nsight Systems User Guide: https://docs.nvidia.com/nsight-systems/UserGuide/index.html

Climate related optimizations for GPUs https://github.com/mrnorman/miniWeather/wiki/A-Practical-Introduction-to-GPU-Refactoring-in-Fortran-with-Directives-for-Climate

Overview of common profiling methods https://www.atatus.com/blog/what-is-code-profiling-a-detailed-explanation/#Types-of-Code-Profiling

NVTX Walkthrough: https://developer.nvidia.com/blog/cuda-pro-tip-generate-custom-application-profile-timelines-nvtx/

OpenACC Best Practices for GPU Refactoring: https://www.openacc.org/sites/default/files/inline-files/OpenACC_Programming_Guide_0_0.pdf