10_HandsOnNsight_nsys

June 21, 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 rchelp.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.

1.2 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.

1.3 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

1.4 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

1.5 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.

2 NSight Systems and Compute

The Nsight Systems and Compute tools are used to profile, debug, and optimize applications that utilize Nvidia GPUs. You can follow along by installing a free Nsight Systems client on your local machine.

Running nsys -v, Casper provides Nsight Systems version 2021.2.4.12 with cuda/11.4.0 module. The nvhpc/22.5 module provides version 2022.2.11.



2.1 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

2.1.1 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 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:



2.2 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 NVIDIA CUDA kernels.

These kernels can be analyzed for performance using Nsight Systems and Compute.

3 miniWeather App OpenACC Profiling Example

3.1 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.

```
erform 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 )
   Uproutine sem__discrete_step( state_init , state_rorcing , state_out , dt
implicit none
real(rp), intent(in ) :: state_init (1-hs:nx+hs,1-hs:nz+hs,NUM_VARS)
real(rp), intent(inout) :: state_out (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
intencer intent(in ) :: dt
                                             ) :: dt
) :: dir
    integer , intent(in ) :: dir
integer :: i,k,ll
real(rp) :: x, z, wpert, dist, x0, z0, xrad, zrad, amp
      f (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)
   !Apply the tendencies to the fluid state
   !$<mark>acc</mark> parallel loop
       0 11 = 1 , NUM_VARS
do k = 1 , nz
            k = 1 , nz
do i = 1 , nx
               if (data_spec_int == DATA_SPEC_GRAVITY_WAVES) then
                           (i_beg-1 + i-0.5_rp) * dx
(k beg-1 + k-0.5 rp) * dz
```

3.2 Setting up a baseline

The Nsight Systems profile launch within this script:

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

3.2.1 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
- -0
 - -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
 - Run below cells to see help text

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

Additional options for CLI profiling can be found in the NVIDIA NSight CLI documentation.

[]: nsys -h

```
[]: nsys profile -h
```

3.3 Launching the Profiler on Casper

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

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

3.4 Quick Analysis via CLI

The below command providese summary output about an **nsys** profile report and will look familiar if you have used **nvprof** to profile codes previously.

```
[1]: 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/targetlinux-x64/reports/cudaapisum.py reports/miniweather_baseline.sqlite]...

Time(%)	Total Time (ns)	Num Calls	Average	Minimum	Maximum
StdDev	Name				
	211 0/5 /00 077	 900 E00	274 070 E	100	1 110 021
90.9	311,045,408,877	829,502	314,918.5	480	4,110,831
672,951.0	b custreamsynchron	llze			
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 c	uEventRecord				
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	е			
0.0	3,205,420	31	103,400.6	1,358	1,425,718

337,9	926.8	cuMemAlloc_v2				
	0.0	1,062,003	2	531,001.5	6,018	1,055,985
742,4	438.8	cuMemAllocHost_v2				
	0.0	378,549	1	378,549.0	378,549	378,549
0.0	cuMod	uleLoadDataEx				
	0.0	45,674	4	11,418.5	2,677	25,152
10,6	28.6	cuMemsetD32Async				
	0.0	34,418	26	1,323.8	285	12,870
2,52	1.0 c	uEventCreate				
	0.0	13,140	1	13,140.0	13,140	13,140
0.0	cuStr	eamCreate				
	0.0	2,444	1	2,444.0	2,444	2,444
0.0	cuIni	t				

Running [/glade/u/apps/dav/opt/cuda/11.4.0/nsight-systems-2021.2.4/targetlinux-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 tendencie	s 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_tendenci	es_x_278_gp	u			
22.3	69,197,111,596	46,083	1,501,575.7	1,383,189	1,749,617	
22,199.8	compute_tendenci	es_z_334_gp	u			
10.9	33,900,055,922	92,166	367,815.2	357,149	503,131	
5,227.3	<pre>semi_discrete_ste</pre>	p_231_gpu				
9.1	28,296,053,744	46,083	614,023.7	602,971	748,570	
3,340.8	compute_tendencie	s_x_308_gpu				
0.2	641,650,166	46,083	13,923.8	12,288	18,624	
372.0 se	et_halo_values_z_4	52_gpu				
0.1	288,514,354	46,083	6,260.8	5,408	14,560	
261.2 se	et_halo_values_x_3	95_gpu				
0.1	281,057,530	46,083	6,098.9	5,855	14,624	
204.3 se	et_halo_values_x_4	18_gpu				
0.0	172,575	2	86,287.5	78,335	94,240	
11,246.5	reductions_871_g	pu				
0.0	19,680	2	9,840.0	9,280	10,400	
792.0 re	eductions_871_gpu_	_red				

Running [/glade/u/apps/dav/opt/cuda/11.4.0/nsight-systems-2021.2.4/targetlinux-x64/reports/gpumentimesum.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/targetlinux-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
HtoDJ						
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/targetlinux-x64/reports/osrtsum.py reports/miniweather_baseline.sqlite]...

Time(%)	Total	Time	(ns)	Num	Calls	Average	Minimum	Maximum
StdDev			Name	Э		-		
33.3	987,64	40,768	3,804		107	9,230,287,558.9	1,161	
328,766,95	7,311	50,0	003,634	1,104	.2 ep	oll_wait		
33.3	986,5	59,928	3,564	6	51,088	16,149,815.5	1,030	
329,169,30	7,528	1,8	383,234	1,444	.9 po	11		
22.2	659,2	70,247	7,276		167	3,947,726,031.6	42,623,923	
329,639,29	8,880	25,3	355,113	3,168	8.4 se	lect		
11.1	328,09	94,330),739		656	500,143,796.9	500,041,057	
500,223,00	2		18,299	9.5	pthrea	d_cond_timedwait		
0.1	1,7	24,974	1,920		708	2,436,405.3	3,076	
9,646,299		2,564	1,917.0	5 pw	rite			
0.0	40	66,505	5,517	9	3,319	4,999.0	1,002	
28,009,889)	16	58,578	.5 i	octl			
0.0	28	88,069	9,694		1,340	214,977.4	1,961	
69,167,686	i	2,06	52,232	.5 o	pen			
0.0	10	62,721	1,610		4,757	34,206.8	1,000	
136,774,46	2	1,9	983,219	5.8	read			
0.0	5	83,935	5,099		29	2,894,313.8	20,654	
36,350,795	,	8,71	14,027	.З р	thread	_cond_wait		
0.0	:	34,064	1,040		1	34,064,040.0	34,064,040	
34,064,040)		0	.0 t	runcat	e		
0.0		29,334	1,013		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_fal	locate	
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_created_	ate	
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_timed	wait	
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_mut	ex_lock	
0.0	1,286,497	6	214,416.2	61,355
450,401	132,760.0	pthread_joi:	n	
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	$pthread_cond_b$	roadcast	
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	00.070.5	0 055
0.0	57,357	2	28,678.5	8,355
49,002	28,741.8	connect		F 400
0.0	56,751	0	9,458.5	5,408
14,082	3,481.1	getdelim	F 201 C	1 104
0.0	4/,/14	9 fmoto	5,301.0	1,194
12,703	3,094.4	16	0 135 6	1 00/
3 660	700 0 1	ind	2,133.0	1,094
0,000	32 437	3	10 812 3	1 544
20 221	9 339 3	send	10,012.0	1,011
0.0	24,649	5	4,929,8	4,294
5.844	653.9	socketpair	1,02010	1,201
0.0	17,315	6	2,885.8	2,188
3,803	628.6 p	oipe		
0.0	16,192	2	8,096.0	7,430
8,762	941.9 s	shmdt		
0.0	11,878	5	2,375.6	1,129
6,104	2,104.0 s	sigaction		
0.0	9,314	3	3,104.7	2,156
4,708	1,396.3 p	pthread_rwlock_	trywrlock	
0.0	8,542	2	4,271.0	3,595
4,947	956.0 s	shmat		
0.0	8,515	2	4,257.5	3,651
4,864	857.7 a	accept		
0.0	6,379	2	3,189.5	2,643
3,736	772.9 p	process_vm_writ	ev	
0.0	5,661	1	5,661.0	5,661
5,661	0.0 F	pipe2		
0.0	4,542	4	1,135.5	1,016
1,339	148.9 s	shmctl .	0 075 0	0 07-
0.0	3,075	1	3,075.0	3,075
3,075	0.0 1	othread_mutex_t	rylock	, . <u> </u>
0.0	1,470	1	1,470.0	1,470

3.5 Timeline Analysis via Nsight Systems GUI

3.5.1 Transfer or View the 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+with
 - X session works but can be slow

3.5.2 Nsight Systems GUI

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

•••	NVIDIA Nsight Systems 2022.2.1							
<u>File View T</u> ools <u>H</u> elp								
Project Explorer ×	Project 1 X							
➡ Project 1 ■ vecAdd_profile.qdrep	Select target for profiling 🔹 🌶							
miniweather_openacc_smallqdrep	Last used target: bneuman@casper.ucar.edu (SSH). <u>Select</u>	Start						
 miniweather _openacc_large_lcpu.qdrep miniweather _openacc_large_24cpu.qdrep 	Select a target to see available options.							
miniweather_openacc_large_24cpu_noasy								
miniweather_openacc_noNCoutput.qdrep								
								

3.5.3 Projects

	NVIDIA Nsight Systems 2022.2.1		
<u>File View Tools H</u> elp		<u>File</u> View <u>T</u> ools <u>H</u> elp	
Project Explorer ×		New Project	жn
Project 1		<u>Open</u>	жо
<pre>wecAdd_profile.qdrep miniweather_openacc_small.qdrep miniweather_openacc_large_tcpu.qdrep miniweather_openacc_large_tcpu.qdrep</pre>	View and open available reports	Import	961
	accopiated with your project	Export miniweather _openacc_24cpu_nooutput.qdrep	æε
miniweather_openacc_large_24cpu.qdrep miniweather_openacc_large_24cpu_noasv	associated with your project	Close miniweather_openacc_noNCoutput.qdrep	жw
i miniweather_openacc_noNCoutput.qdrep		⊗ E <u>x</u> it	#Q
			Ç.

3.5.4 Navigation



3.5.5 Event Descriptions

• • •		NVIDIA Nsight Systems 2022.2.1
<u>File View Tools Help</u>		
Project Explorer ×	Project 1 × vecAdd_profile.qdrep ×	miniweather_openacc_smallqdrep × miniweather_openacc_large_1cpu.qdrep × miniweather_openacc_large_24cpu.qdrep × miniwea
	≡ Timeline View 👻	📾 🔍 tx 🛛 🔜 🔥 8 warnings, 20 messages
		0s 4s 8s 12s 16s 20s 24s 28s 32s 36s 40s ≜
	- CPU (72)	
CPU Utilization	CPU 32	
	23 CPUs hidden+	
	✓ Processes (2)	
	✓ ● [174874] openacc	
Runtime traces	✓ Threads (11)	
not in profiling	- [174874] openacc -	
set in proming	OS runtime libraries	It material terretation of the second contraction of the second terretation of the second s
command	CUDA API	. ARKODERATIO DEBURALEKERKERERERERERERERERERERERERERERERERER
	Profiler overhead	
	10 threads hidden — +	
	~ CUDA HW (0000:18:00.0 ·	
	▶ [All Streams]	
GPU Kernels	 >99.9% Stream 14 > 06.0% Kernele 	ABBULENDE UT UMBERBEITERUNGEN INGER ENGENEUNEN UMBEREITEN AN ANDER EINE UND UMBERBEITEN AUF AN AN AN AN AN AN A Nataun im eine eine eine eine eine eine eine
	 90.9% Kernels 3.1% Memory 	
	 ♦ <0.1% Stream 13 	
	•	
	Events View 👻	
		Name 👻 🔍
	# Anme	Start Duration (* Description:
	83034 Memcpy HtoD	19.3067s 4.769 μs Begins: 19.31s
	83035 Memcpy DtoH	19.3079s 2.912 μs Ends: 19.3112s (+1.272 ms) DtoH memcpy 16,777,120 bytes
	83036 Memcpy DtoH	19.3079s 2.976 µs Source memory kind: Device
	83037 Memcpy HtoD	19.308s 4.896 µs Destination memory kind. Printed Throughput: 13.191 GiB/s
	83038 Memcpy HtoD	19.308s 5.056 µs Correlation ID: 379013
	83039 Memcpy DtoH	19.30838 2.880 µs

3.6 Baseline Timeline View

miniweather_baseline.qdrep

Project 1 x miniweather_baseli	ne.qdrep	× miniweather	_async.qdrep 3	< miniweathe	r_async_coll	apsed.qdrep 3	< miniweather	r_nooutput.q	drep 🗙 miniw	eather_noasy	ync_collapsed.q	drep 🗙
≡ Timeline View 👻									Q 1x		A 8 warr	
•	0s	30s	60s	90s	120s	150s	180s	210s	240s	270s	300s	330s
 CPU (72) 												
 Processes (2) 												
- 💿 [19334] openacc												
✓ Threads (11)												
✓ ✓ [19334] openacc •	_			-		-						
OS runtime libraries	k											
CUDA API												nalant.
Profiler overhead												
10 threads hidden – +												_
CUDA HW (0000:18:00.0 ·	-											
▶ [All Streams]	, ninii								internetien			
▶ 99.7% Stream 13		minikaanninaa										nalahik
▶ 0.3% Stream 14												
▶ ○ [19341] orted												

3.7 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.

■ Timeline View Image: Construction of the second of	Project 1 x miniweather_baseline.	qdrep × miniweather _async.qdre	ep 🗙 miniweather_async_co	llapsed.qdrep 🗙 miniweather	_async_noNCoutput.qdrep 🗙 🔣
705 300ms +310ms +320ms +340n 70s 348.6ms ns +370ms +380ms * Threads (11) * * [19334] openacc *	≡ Timeline View 👻			📼 Q 1x 🔲	^a <u>A warnings,20 messages</u>
 Threads (11) I [19334] openacc - OS runtime libraries CUDA API Profiler overhead 10 threads hidden + + CUDA HW (0000:18:00.0 - F (All Streams] I I I I I I I I I I I I I I I I I I I	70s + 30	0ms +310ms +320ms	+330ms +340n 70s	346.6ms ns +360ms	+370ms +380ms 📤
 V [19334] openacc - OS runtime libraries CUDA API Profiler overhead 10 threads hidden	✓ Threads (11)				
OS runtime libraries Image: Sector Secto	- ✓ [19334] openacc -				
CUDA API Image: Cup a API Profiler overhead Image: Cup a API 10 threads hidden = + Image: Cup a API CUDA HW (0000:18:00.0 - Image: Cup a API Image: Cup a HW (0000:18:00.0 - Image: Cup a API Image: Cup a HW (0000:18:00.0 - Image: Cup a API Image: Cup a HW (0000:18:00.0 - Image: Cup a API Image: Cup a HW (0000:18:00.0 - Image: Cup a API Image: Cup a HW (0000:18:00.0 - Image: Cup a API Image: Cup a HW (0000:18:00.0 - Image: Cup a API Image: Cup a API Image: Cup a API Image: C	OS runtime libraries				
Profiler overhead 10 threads hidden + • CUDA HW (0000:18:00.0 · - • [All Streams] - • 99.7% Stream 13 - • >99.9% Kernels - • 34.8% compute_tenc CO CO CO • 22.5% compute_tend CO CO CO • 22.3% compute_tend CO CO • 10.9% semi_discrete_ I • 9.1% compute_tend I • 10.9% semi_discrete_ I • 9.1% compute_tend I • 10.9% semi_discrete_ I • 9.1% compute_tend I <th>CUDA API</th> <th></th> <th></th> <th></th> <th></th>	CUDA API				
10 threads hidden + - • CUDA HW (0000:18:00.0 - - • [All Streams] - • 99.7% Stream 13 - • >99.9% Kernels - • 34.8% compute_tenc - • 22.5% compute_tend - • 10.9% semi_discrete_ - • 9.7% compute_tend - • 9.7% compute_tend - • 10.9% semi_discrete_ - • 9.1% compute_tend - • 9.1% compute_tende - • 9.1% compute_tende - • 9.1% compute_tende - • 9.1% compute_tende -	Profiler overhead				
CUDA HW (0000:18:00.0 -	10 threads hidden — +				
> [All Streams] [All All All All All All All All All All	- CUDA HW (0000:18:00.0 -				
> 99.7% Stream 13	▶ [All Streams]	هه اه اه اه اه اه اه		فالعا والعصاوها وها	81,81,81,8 31,8 39,8 39,8
> >99.9% Kernels <td< th=""><th>✓ 99.7% Stream 13</th><th>.ei.ei.ei.ecie@ie@</th><th>e</th><th>6 ,8 ,8 ,8@ :6 8@ </th><th>81.81,81,8 61,8 62,8 63,8</th></td<>	✓ 99.7% Stream 13	.ei.ei.ei.ecie@ie@	e	6 ,8 ,8 ,8@ : 6 8@	81.81,81,8 61,8 62,8 63,8
> 34.8% compute_tenct CC CC CC CC CC CC CC CC CC CC CC CC > 22.5% compute_tend CC CC CC CC CC CC CC CC CC CC CC CC > 22.3% compute_tend CC CC CC > 10.9% semi_discrete_ IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	→ >99.9% Kernels	.ei .ei .ei .e .e .e .e .e	81, 81, 81, 8 co., 8 co., 8 co.,		8], 8], 8], 8], 8], 8], 8], 8], 8], 8],
> 22.5% compute_tend Image: Compute_tend	▶ 34.8% compute_tenc	co co co	CO CO CO	co co co	CO CO CO
> 22.3% compute_tend ■	22.5% compute_tend	8 8 8	8 8 8		388 8
→ 10.9% semi_discrete_ I <th>22.3% compute_tend</th> <th></th> <th></th> <th>• • •</th> <th>C 🗉 🖸</th>	22.3% compute_tend			• • •	C 🗉 🖸
▶ 9.1% compute_tende	▶ 10.9% semi_discrete_				
	▶ 9.1% compute_tende				

3.7.1 Stats View

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



3.8 Asynchronous Loops Profile

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

Stats System View 🔻										
CUDA API Summary	Time		Instances	Avg	Med	Min	Max	StdDev	Category	Operation
CUDA API Trace	49.7%	311.045 s	829502	374.978 µs	6.000 μs	480 ns	4.111 ms	672.951 μs	CUDA_API	cuStreamSynchronize
CUDA GPU Trace	17.2%	5 107.763 s	46083	2.338 ms	2.338 ms	2.319 ms	2.594 ms	7.323 μs	CUDA_KERNEL	compute_tendencies_z_3
CUDA Summary (API/Kernels/MemOp DX11 PIX Range Summary	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
DX12 GPU Cmd List PIX Ranges Sumn	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
DX12 PIX Range Summary GPU MemOps Summary (by Size)	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_
GPU MemOps Summary (by Time)	4.5%	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
GPU Summary (Kernels/MemOps) Kernel Launch & Exec Time Summary	0.3%	5 1.952 s	414751	4.706 μs	4.489 µs	2.995 µs	1.277 ms	3.407 μs	CUDA_API	cuLaunchKernel
Kernel Launch & Exec Time Trace	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_
NVTX Push/Pop Range Summary NVTX Push/Pop Range Trace	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]
NVTX Range Kernel Summary	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]
NVTX Range Projection NVTX Range Summary	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:
CLI command:: nsvs stats -r apigpusum /Users/	0.1%	328.508 ms	46186	7.112 μs	4.461 μs	1.109 μs	1.278 ms	59.376 μs	CUDA_API	cuCtxSynchronize
bneuman/Downloads/ miniweather_baseline.sqlite	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.



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

3.8.1 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	Time 🔻	Total Time	Instances	Avg	Med	Min	Max	StdDev	Category	Operation
CUDA API Trace CUDA GPU Kernel Summary	49.2%	304.581 s	46186	6.595 ms	2.463 ms	2.141 µs	17.019 ms	5.904 ms	CUDA_API	cuCtxSynchronize
CUDA GPU Trace	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
CUDA Summary (API/Kernels/MemOp DX11 PIX Range Summary	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
DX12 GPU Cmd List PIX Ranges Sumn	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
DX12 PIX Range Summary GPU MemOps Summary (by Size)	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_c
GPU MemOps Summary (by Time)	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
GPU Summary (Kernels/MemOps)	0.3%	1.560 s	92856	16.804 µs	584 ns	450 ns	15.159 ms	492.650 µs	CUDA_API	cuEventSynchronize
Kernel Launch & Exec Time Trace	0.2%	1.539 s	414751	3.710 µs	3.309 µs	2.782 µs	1.104 ms	4.423 μs	CUDA_API	cuLaunchKernel
NVTX Push/Pop Range Summary	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_c
A P	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]
CLI command::	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]
nsys stats -r apigpusum /Users/ bneuman/Development/NSight_Reports/	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_
miniweather_async.sqlite	•									

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 parallezation of loops?

Project 1 × miniweather	r_baseline.qdrep	o 🗙 miniweath	er_async.qdrep	🛛 🗙 miniweathe	er_async_colla	osed.qdrep
= Timeline View						
	48s -	+230ms	48s 235.86ms	+240ms	+245ms	+250r
 Processes (2) 						
▼ ● [40940] openacc						
▼ Threads (11)						
∽ 🗸 [40940] opena	acc -					
OS runtime libra	aries "					
CUDA API		L cuCtxSy CuC	txSy	cuCtxSyn	chronize	
Profiler overhea	ıd					
10 threads hidden	· _ +					
io theads hader	····					
- CUDA HW (0000:18	B:00.0 -					-
 CUDA HW (0000:18 ▶ [All Streams] 	B:00.0 -	com		om compute com	compute	compute
 ✓ CUDA HW (0000:18 ▶ [All Streams] ✓ >99.9% Stream 1 	B:00.0 -	com com		om compute com.	compute com	compute
 ✓ CUDA HW (0000:18 → [All Streams] ✓ >99.9% Stream 1 ✓ 99.7% Kernels 	8:00.0 - 4			om compute (com om compute (com om compute (com	compute com compute com	compute
 CUDA HW (0000:18 [All Streams] >99.9% Stream 1 99.7% Kernels 34.8% computer 	8:00.0 - 4 ite_tenc	, com, com , com, com		om compute Com. om compute Com. om compute Com.	compute com compute com compute com compute	compute
 CUDA HW (0000:18 [All Streams] >99.9% Stream 1 99.7% Kernels 34.8% comput 100.0% comp 	4 4 ite_tenc ipute_te	com com		om) compute (com. om) compute (com. om) compute (com. compute)	compute com compute com compute com compute	compute
 CUDA HW (0000:18 [All Streams] >99.9% Stream 1 99.7% Kernels 34.8% computition 100.0% computition 22.4% computition 	4 ite_tenc ipute_te te_tend	, com , com , com , com , com		om compute (com. om compute (com. om compute (com. compute	compute Com compute Com compute Com compute compute	compute
 ✓ CUDA HW (0000:18 ▶ [All Streams] ✓ >99.9% Stream 1 ✓ 99.7% Kernels ✓ 34.8% compute 100.0% compute ▶ 22.4% compute ▶ 22.3% compute 	4 4 te_tenc upute_te te_tend te_tend	, com, com		om compute (com. om compute (com. compute (com. compute)	compute com compute com compute compute	compute compute compute compute compute

3.9 Collapsed Loops Profile

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



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

3.9.1 Collapsed Loops Analysis

miniweather_async_collapsed.qdrep

Project 1 × miniweather _baseline.qdre	p 🗙 miniwea	ther_async.qd	rep 🗙 <mark>miniwe</mark> a	ather_async_c	ollapsed.qdre	o 🗙 miniweat	her_async_no:	NCoutput.qdre	ep 🗙 miniweather_	_noasync_collapsed.qdrep 🗙
≡ Timeline View →							8	🛛 Q 1x 🔲 🗏		<u> 8 warnings,20 message</u>
						6s 20	0s 24s			36s 40s
- CPU (72)										
CPU 32		11			ii ii					
CPU 31							1	6		
22 CPUs hidden		+								
Processes (2)										
 ✓ ● [174874] openacc 										
Threads (11)										
		1.100								
▼ ✓ [174874] openacc ▼										
OS runtime libraries		եւ հուս			k					
CUDA API			NAMERICAN AND	LICHARICHARI AGALARI	Natural naturation	UMANIJAN IN A	CIGACICICALICICALICICA	ING A A A A A A A A A A A A A A A A A A A	UMUMUMUMUMUM	
Profiler overhead										
10 threads hidden		+								
 CUDA HW (0000:18:00 0 - Quadro 	GP100)								HUNHUNHUNHUNH	In the second second
► [All Streams]				I NUMARINA AMARINA.	Includent indentitieren	IN THE REPORT OF THE	CILLARICE AL INCLARING	INTELLE ALLER INTELLE	undun nahalunundun diakatunun	LU BAADAAN BAADAA
> >99 9% Stream 14				I NUMAN NUMAN NA KADADAN.	Natural di Katalana	E ISAN MARAN IN IN A JA	E RELACIONAL DE RELACIÓN	I III IIII A IIII A IIII IIIII	LIGHT INTERNIS	
▶ 96.9% Kernels				INTERNAL INTERNAL	National Internation	NIGER IN GRADE AND	CIGLICIALI CIALICIA	INCLUSION DATA	unan munan munan munan munan muna	
> 3 1% Memory				500000000000000000000000000000000000000						
<0.1% Stream 13										
v Co.the Stream 15										
										<u>`</u>
Stats System View										
CUDA API Summary	Time 👻	Total Time	Instances	Avg	Med	Min	Max	StdDev	Category	Operation
CUDA GPU Kernel Summary	43.2%	22.799 s	46186	493.637 µs	241.860 µs	1.906 µs	4.155 ms	355.174 μs	CUDA_API	cuCtxSynchronize
CUDA GPU Trace	17.0%	8.992 s	92166	97.567 μs	97.312 μs	96.415 μs	541.693 µs	2.303 µs	CUDA_KERNEL	semi_discrete_step_231_c
DX11 PIX Range Summary	10.2%	5.361 s	46083	116.333 µs	116.127 μs	115.679 μs	140.063 µs	1.514 μs	CUDA_KERNEL	compute_tendencies_z_3
DX12 GPU Cmd List PIX Ranges Sumn	8.9%	4.691 s	46083	101.786 µs	101.631 μs	101.119 µs	121.727 μs	1.309 μs	CUDA_KERNEL	compute_tendencies_x_3
GPU MemOps Summary (by Size)	6.7%	3.551 s	46083	77.052 μs	78.336 µs	71.360 µs	89.695 µs	2.635 µs	CUDA_KERNEL	compute_tendencies_z_3
GPU MemOps Summary (by Time)	6.5%	3.409 s	46083	73.974 µs	74.848 µs	67.360 µs	452.093 µs	3.447 µs	CUDA_KERNEL	compute_tendencies_x_2
GPU Summary (Kernels/MemOps) Kernel Launch & Exec Time Summary	2.9%	1.542 s	414751	3.717 μs	3.398 µs	2.827 μs	869.459 μs	3.651 μs	CUDA_API	cuLaunchKernel
Kornal Launch & Evan Time Trace	0.9%	452.095 ms	92188	4.904 μs	4.832 μs	896 ns	1.360 ms	6.330 µs	MEMORY_OPER	[CUDA memcpy HtoD]
CLI command::	0.8%	408.949 ms	92374	4.427 μs	2.944 μs	896 ns	1.839 ms	42.562 μs	MEMORY_OPER	[CUDA memcpy DtoH]
nsys stats -r apigpusum /Users/	0.6%	316.891 ms	92188	3.437 µs	3.342 μs	2.355 μs	394.217 μs	2.696 µs	CUDA_API	cuMemcpyHtoDAsync_v2
miniweather_openacc_24cpu.sqlite	< <u>^ ^ / / / / / / / / / / / / / / / / / </u>	044 050	00074	0 070	0 407	0.004	000 00F	4 0 4 0		

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

Project 1 🗙 miniweather_t	baseline.qdrep 🗙	miniweather_as	ync.qdrep 🗙	miniweather_as	ync_collapsed.qdr	ep 🗙 miniwe	ather_async	:_noNCou
= Timeline View	-							
		ns 48s 2	35.51ms					ms
CUDA API		CtxSyn cuCtxSyn		cuCtxSynch	ronize	CuCtxS	yn cuCtxSyn	ul -
Profiler overhead			Call to out					
10 threads hidden			CUDA dr	iver API calls				
- CUDA HW (0000:18:0			Begins: 48	.2353s	2)			
▶ [All Streams]		m	Return valu	1035 (+14.976 m ie: 0	pute com com	pute	com	com
✓ >99.9% Stream 14		m	Correlation	ID: 118325	pute] [com]com	pute	com	com
y 99.7% Kernels	CO	m	Call stack:		pute com com	pute	com	com
	_tenc		libcuda.so.5 [Max depth]!	IO.47.03[3 Frames] [Max depth]	pute_t com	pute_t		
100.0% compu	ıte_t∈		1997 - 19	compute_t	compute_t com	pute_t		
▶ 22.4% compute_	tend 🔤	np	comp			comp	comp	comp
▶ 22.3% compute_	_tend		comp	comp	comp			
▶ 10.9% semi_disc	crete_							
▶ 9.1% compute_t	ende	_	=			(C
3 kernel groups l	-+							
 0.3% Memory 								
53.5% HtoD me	тсру							1
46.5% DtoH mer	тсру							

Project 1 × miniweather_base	line.qdrej	p 🗙 miniwea	ther_async.qdr	ep 🗙 miniweathe	r_async_colla	psed.qdrep ×	miniweather_asyn
5s •	+380m		+380.5ms	+3[5s	381.139ms	+381.5ms	+382ms
CUDA API	. . I	CuCtxS	mchrCuCt	Synchro		cuCtxSynchron	ize
Profiler overhead 10 threads hidden → + CUDA HW (0000:18:00.0 + [All Streams] >99.9% Stream 14 96.9% Kernels > 33.9% semi_discrete > 20.2% compute tend	ц. Ц.	, , co., com., (, , co., com., (, co., com., (som c com som c com som c com		Call to cu CUDA cu Ends: 5 Ends: 5.3 Return va correlation Call stack corr Max depth comp.	Ctx Synchronia driver API calls .38112s 821s (+978.898 lue: 0 nn ID: 76255 510.47.03(0x2b1d][Max depth] comp.	ze somp_sem_ 3 μs) somp_sem_ jomp_sem_ fb77b618 sem_ comp_
▶ 17.7% compute_tend	•	com	com	com)		
▶ 13.4% compute_tend	l				œ	co	CO
12.8% compute_tend		<u>co</u>	<u>co</u>	CO			
3 kernel groups I — +	L		- 1				l
3.1% Memory	<u> </u>	4	u u	i I			
52.4% HtoD memcpy		al de la companya de					
47.6% DtoH memcpy							
 ▶ <0.1% Stream 13 	4						

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

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

Project 1 🗙 miniweather _basel	ine.qdrep	× miniweather	·_as	sync.qdrep 🗙	mini	weather_as	ync_collapsed	l.qdrep × min	iweather_asyn
Ξ Timeline View 👻									
	+380ms			.5ms 5	s 38(0.929ms		81.5ms	+382ms
CUDA API	. .	cuCtxSynchr.		cuCtxSynchro				cuCtxSynchronize	
Profiler overhead						1		and a second second	
10 threads hidden – +						Call to se	mi_discrete_s auncher	step_231_gpu	
- CUDA HW (0000:18:00.0 -	ų, k	-1-	۰.		۰,	Begins: 5.	38093s		
▶ [All Streams]	i.	co com sem	Li i	c com sem		Return val	8093s (+3.519 ue: 0	9 μs)	co comp sem
▼ >99.9% Stream 14	i.	co com sem	Li i	c com sem_	1	GPU: 000	0:18:00.0 - Qu	uadro GP100	co comp sem
 96.9% Kernels 		co com sem	.	c com sem_	.	Latency: 1	i 62.410 μs→		co comp sem
33.9% semi_discrete		semi_		semi_		Correlatio	n ID: 76237		semi
20.2% compute_tend							comp	comp	comp
17.7% compute_tende		com		com		com			
13.4% compute_tend							co	CO	co
12.8% compute_tend		CO		co		co			
3 kernel groups I — +	L.	j –	l.				1		1 1
 3.1% Memory 		4			11				
52.4% HtoD memcpy		al de la companya de							
47.6% DtoH memcpy									
▶ <0.1% Stream 13									

3.9.2 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.

Project 1 × miniweather_baseline.qdrep × r	miniweather_async.qdrep	x miniweather_async_c	ollapsed.qdrep × miniweather	r_async_noNCoutput.qdrep 🗙	miniweather _noasync_collapsed.qdrep 🗙
≡ Timeline View 👻				🔤 Q 1x 🗌	▲ 8 warnings,20 message
	- 5.4s	5.6s 5.8s	6s 6.2s 6.4s	6. 6.666s 6.8s	7s 7.2s 7.4s 7.6s
- CPU (72)					
CPU 32					
CPU 31					
22 CPUs hidden					
✓ Processes (2)					
▼ ● [174874] openacc					
✓ Threads (11)					
✓ ✓ [174874] openacc		<u>~</u>	(4)		
OS runtime libraries					
CUDA API				open	H
Profiler overhead				Ends: 6.68177s (+3	34.648 ms)
10 threads hidden				Call stack at 6 64	712-0
✓ CUDA HW (0000:18:00.0 - Quadro GP10	10)			libpthread-2.17.so!	libc_open
▶ [All Streams]				libmca_common_omp	Imca_snaredtp_sm_tile_open
✓ >99.9% Stream 14				libmpi.so.40.30.11mc	a_io_base_file_select
▶ 96.9% Kernels				libmpi.so.40.30.11PM	IPI_File_open
▶ 3.1% Memory				openaccincmpio_begin	n_indep_data
▶ <0.1% Stream 13				openacciminiweather	_output
	4			openaccimain libc-2.17.sol libc st	art main
Stats System View 👻				openacc!_start	

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.

Project 1 🗙 miniweather_	_baseline.qdrep 🗙	miniweather_a	sync.qdrep 🗙 m	iniweather_a	sync_collapsed.qdr	rep × miniweather	_async_noNCc	utput.qdrep ×	miniweather_noa	sync_collapsed.qdr
≡ Timeline View								λ,1x ■	<u> </u>	
CUDA API										
Profiler overhead		L L								
10 threads hidden										
- CUDA HW (0000:18:	00.0 ·	-								
▶ [All Streams]										
→ >99.9% Stream 14										
▶ 34.2% semi_dis	crete	dalliterabelleresterilleres								
20.4% compute	e_tenc									
▶ 17.8% compute	_tend									
13.0% compute	_tend									
▶ 12.5% compute	_tend									
3 kernel groups	1-+									
▶ 2.7% Memory										
 ► <0.1% Stream 13 										
										·

3.9.3 Expert View

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



3.10 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/

NVTX FORTRAN Example:

4 Launching Nsight Compute with Nsight Systems

Project 1 ¥ miniweather base	line adren X miniwest	her async adren	* miniwest	har async colla	nsed adren ¥	miniweather	async noN
			A minweat	ner_async_cona		manneather	_async_non
➡ Timeline View							
4s -	⊦1.85ms +1.9ms	+1.95ms +2	ms +2.0)5ms +2 4s 2 .	1219ms 5ms	+2.2ms	+2.25ms
CUDA API	_	cuCtxSy	nchronize				
Profiler overhead							
10 threads hidden — 1							
- CUDA HW (0000:18:00.0							
▶ [All Streams]	se_ compute_tendencies_z	. compute_tendencies_	z_369_gpu	semi_discrete_step_23	1_gpu		compute_tendencie
▼ >99.9% Stream 14	se_ compute_tendencies_z	. compute_tendencies_	z_369_gpu	semi_discrete_step_20	gpu		compute tendencie
▼ 97.3% Kernels	se_ compute_tendencies_z	. compute_tendencies_	z_369_gpu	semi_discrete_step_23	Semi_discret Begins: 4.002	205s	_gpu
34.2% semi_discrete	50_			semi_discrete_step_23	Ends: 4.00214	4s (+98.656	μs)
20.4% compute_ten	c	compute_tendencies_	z_369_gpu		grid: <<<163 block: <<<128	84, 1, 1>>> 3, 1, 1>>>	
17.8% compute_tend	ł				Launch Type:	Regular	
13.0% compute_tend	compute_tendencies_z				Dynamic Shared	Memory: 0 c red Memory:	oytes 0 bytes
12.5% compute_tend	di 🛛				Registers Per	Thread: 38	
3 kernel groups I – †					Local Memory	/ Per Thread: / Total: 91,75	0 bytes 0,400 bytes
2.7% Memory					Shared Memo	ry executed:	0 bytes
▶ <0.1% Stream 13					Theoretical of	ccupancy: 75	:4В %
	•				Launched from	n thread: 200	8252
Stats System View 👻					Correlation ID	o.2o3 μs : 71717	
	1 Time	Total Time	Num Ca		Stream: Strea	im 14	

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:

Project 1 × miniweather_base	eline.qdrep 🗙 miniweath	er_async.qdrep 🗙 miniwea	ther_async_collapsed.qc	drep 🗙 miniweat	ther_async_	noNCoutput.qdrep 🗙 👖
≡ Timeline View 👻						🔤 Q 1x 🛛
CUDA API		cuCtxSynchronize				11.
Profiler overhead						
10 threads hidden — -						
- CUDA HW (0000:18:00.0					•	· · ·
▶ [All Streams]	secompute_tendencies_z	compute_tendencies_z_369_gpu	semi_discrete_step_231_gpu		compute_ter	dencies_x compute_tendencie
	secompute_tendencies_z	compute_tendencies_z_369_gpu	semi_discrete_step_231_gpu		compute_ter	dencies_x compute_tendencie
▼ 97.3% Kernels	secompute_tendencies_z	compute_tendencies_z_369_gpu	semi_discrete_step_231_gpu		compute_ter	dencies_x compute_tendencie
▶ 34.2% semi_discret			semi_discrete_step_231_gpu			
▶ 20.4% compute_te	Remove Filter	dencies_z_369_gpu				
I7.8% compute_ter	Undo Zoom (59)					compute_tendencie
▶ 13.0% compute_tei	Reset Zoom					
12.5% compute_ter	7 Pin row				compute_ter	ndencies_x
3 kernel groups I –	Show in Events View				1. Contract (1997)	
				11 01		
▶ 2.7% Memory						
 ▶ 2.7% Memory ▶ <0.1% Stream 13 						
 ▶ 2.7% Memory ▶ <0.1% Stream 13 	•					
 ▶ 2.7% Memory ▶ <0.1% Stream 13 	4				_	
	4				_	Name
	4		Start	Duration	GPU	Name Context
	∢		Start 1.36247s	Duration 115.712 μs	GPU GPU 0	Name Context Stream 14
	∢ ete_step_231_gpu rete_step_231_gpu		Start 1.36247s 1.36292s	Duration 115.712 μs 113.439 μs	GPU GPU 0 GPU 0	Name Context Stream 14 Stream 14
	<pre>ete_step_231_gpu ete_step_231_gpu rete_step_231_gpu rete_step_231_gpu</pre>		Start 1.36247s 1.36292s 1.36331s	Duration 115.712 μs 113.439 μs 113.471 μs	GPU GPU 0 GPU 0 GPU 0	Name Context Stream 14 Stream 14 Stream 14
	ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu rete_step_231_gpu		Start 1.36247s 1.36292s 1.36331s 1.36366s	Duration 115.712 μs 113.439 μs 113.471 μs 113.119 μs	GPU GPU 0 GPU 0 GPU 0 GPU 0	Name Context Stream 14 Stream 14 Stream 14 Stream 14
	ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu		Start 1.36247s 1.36292s 1.36331s 1.36366s 1.36399s	Duration 115.712 μs 113.439 μs 113.471 μs 113.119 μs 113.247 μs	GPU GPU 0 GPU 0 GPU 0 GPU 0 GPU 0	Name Context Stream 14 Stream 14 Stream 14 Stream 14 Stream 14
 ▶ 2.7% Memory ▶ <0.1% Stream 13 Events View # ▲ Name 1 semi_discr 2 semi_discr 3 semi_discr 4 semi_discr 5 semi_discr 6 semi_discr 	ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu		Start 1.36247s 1.36292s 1.36331s 1.36366s 1.36399s 1.36433s	Duration 115.712 μs 113.439 μs 113.471 μs 113.119 μs 113.247 μs 113.600 μs	GPU GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0	Name Context Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14
 2.7% Memory <0.1% Stream 13 Events View # ^ Name 1 semi_discr 2 semi_discr 3 semi_discr 4 semi_discr 5 semi_discr 6 semi_discr 7 semi_discr	ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu		Start 1.36247s 1.36292s 1.36331s 1.36366s 1.36399s 1.36433s 1.36473s	Duration 115.712 μs 113.439 μs 113.471 μs 113.119 μs 113.247 μs 113.600 μs 113.503 μs	GPU GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0	Name Context Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14
 2.7% Memory <0.1% Stream 13 Events View Mame semi_discr semi_discr semi_discr semi_discr semi_discr semi_discr semi_discr semi_discr semi_discr 	ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu		Start 1.36247s 1.36292s 1.36331s 1.36366s 1.36399s 1.36433s 1.36473s 1.36512s	Duration 115.712 μs 113.439 μs 113.471 μs 113.119 μs 113.247 μs 113.600 μs 113.503 μs 113.503 μs	GPU GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0	Name Context Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14
 2.7% Memory <0.1% Stream 13 Events View Mame semi_discr 	ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu		Start 1.36247s 1.36292s 1.36331s 1.36366s 1.36399s 1.36433s 1.36473s 1.36512s 1.36551s	Duration 115.712 μs 113.439 μs 113.471 μs 113.119 μs 113.247 μs 113.600 μs 113.503 μs 113.503 μs 113.343 μs 113.3439 μs	GPU GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0	Name Context Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14
 2.7% Memory <0.1% Stream 13 Events View Mame 1 semi_discr 2 semi_discr 3 semi_discr 4 semi_discr 5 semi_discr 6 semi_discr 7 semi_discr 8 semi_discr 9 semi_discr 10 semi_discr 	ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu		Start 1.36247s 1.36292s 1.36331s 1.36366s 1.36399s 1.36433s 1.36473s 1.36512s 1.36551s 1.36551s 1.36551s	Duration 115.712 μs 113.439 μs 113.471 μs 113.247 μs 113.600 μs 113.503 μs 113.503 μs 113.343 μs 113.439 μs 113.439 μs	GPU GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0	Name Context Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14 Stream 14
 2.7% Memory <0.1% Stream 13 Events View Mame 1 semi_discr 3 semi_discr 4 semi_discr 5 semi_discr 6 semi_discr 7 semi_discr 8 semi_discr 9 semi_discr 10 semi_discr 11 semi_discr 	ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu		Start 1.36247s 1.36292s 1.36331s 1.36366s 1.36399s 1.36433s 1.36473s 1.36512s 1.36551s 1.36551s 1.36551s 1.36585s 1.36619s	Duration 115.712 μs 113.439 μs 113.471 μs 113.119 μs 113.247 μs 113.600 μs 113.503 μs 113.343 μs 113.439 μs 113.311 μs 113.439 μs	GPU GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0 GPU 0	Name Context Stream 14 Stream 14
 2.7% Memory <0.1% Stream 13 Events View Mame 1 semi_discr 3 semi_discr 4 semi_discr 5 semi_discr 6 semi_discr 7 semi_discr 8 semi_discr 9 semi_discr 10 semi_discr 11 semi_discr 12 semi_discr 	ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu ete_step_231_gpu		Start 1.36247s 1.36292s 1.36331s 1.36366s 1.36399s 1.36433s 1.36473s 1.36512s 1.36551s 1.36551s 1.36555s 1.36619s 1.36652s	Duration 115.712 μs 113.439 μs 113.471 μs 113.247 μs 113.600 μs 113.503 μs 113.343 μs 113.439 μs 113.311 μs 113.439 μs 113.439 μs 113.439 μs	GPU GPU 0 GPU 0	Name Context Stream 14 Stream 14

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:

Ξ Timeline View -		Q 1x Q 1x Q 1x Q 2x Q 2x	ges
45 •		+1.90ms +2ms +2.00ms +2.1ms +2.10ms +2.2ms +2.20ms +2.3ms +2.30ms	-
- 🗸 [208252] openacc -			
OS runtime libraries		l l	
CUDA API	cuCtxSynchronize	, III CuCtxSynchronize	
Profiler overhead			
10 threads hidden — +			
- CUDA HW (0000:18:00.0		The set	
▶ [All Streams] 1	compute_tendencie compute_tendencies_z_369_gpu semi_discrete_step_231_gpu compute_tendencie	compute_tendencies_z_369_gpu semi_discrete_step_231_gpu compute_tendencies_x	
✓ >99.9% Stream 14	compute_tendencie compute_tendencies_z_360_gpu semi_discrete_step_231_gpu compute_tendencie co	compute_tendencies_z_369_gpu semi_discrete_step_231_gpu Compute_tendencies_x	
 97.3% Kernels 	compute_tendencie compute_tendencies_z_369_gpu semi_discrete_step_231_gpu compute_tendencie co	compute_tendencies_z_369_gpu semi_discrete_step_231_gpu compute_tendencies_x	
34.2% semi_discrete_	semi_discrete_step_221_gpu	semi_discrete_step_231_gpu	
▶ 20.4% compute_tr 1	compute_tendencies_z_389_gpu	compute_tendencies_z_389_cou	
17.8% compute_tend		Analyze the Selected Kernel with NVIDIA Nsight Compute	
▶ 13.0% compute_tend	compute_tendencies compute_tendencies	Copy ToolTip	
▶ 12.5% compute_tende		Copy Current Time	
3 kernel groups I – +) E E	<u>R</u> emove Filter	
		Fit to screen	e F
State System View		Undo Zoom (50)	
		Reset Zoom	
CUDA API Summary CUDA API Trace	Time Total Time Num Calls Avg	And Fin row	

Here is the window to launch Nsight Compute:



5 Resources

- NVIDIA Nsight Systems User Guide
- Climate related optimizations for GPUs, by Matt Norman (ORNL)
- Overview of common profiling methods
- NVTX Walkthrough
- OpenACC Best Practices for GPU Refactoring

5.1 Move On to Nsight Compute Profiler Tool

Nsight Compute Profiler