Video tutorials for the NSIGHT tools

- https://www.youtube.com/watch?v=nYSdsJE2zMs
- https://www.youtube.com/watch?v=DLQwIdhrL1A (start at 16:20)

In depth resources about Nsight compute and Nsight systems can be found here and here. The most applicable uses of these profilers are explained below.

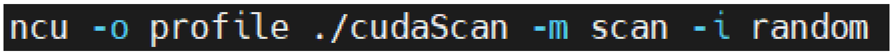
Using Nsight Compute

To profile a program:

Unset

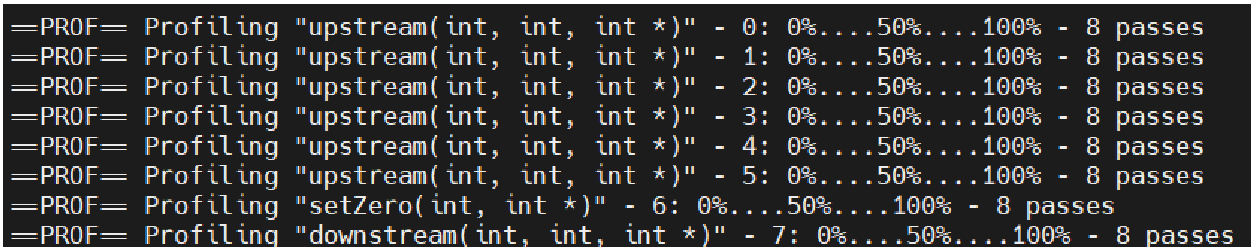
```
ncu -o <reportName> <executable + flags>
```

For example, to run this profiler on *cudaScan*, we could do something like this:



```
ncu -o profile ./cudaScan -m scan -i random
```

Where “profile” will be the name of the report file. You should also see the profiler spit out its progress into the terminal, which will look something like this:



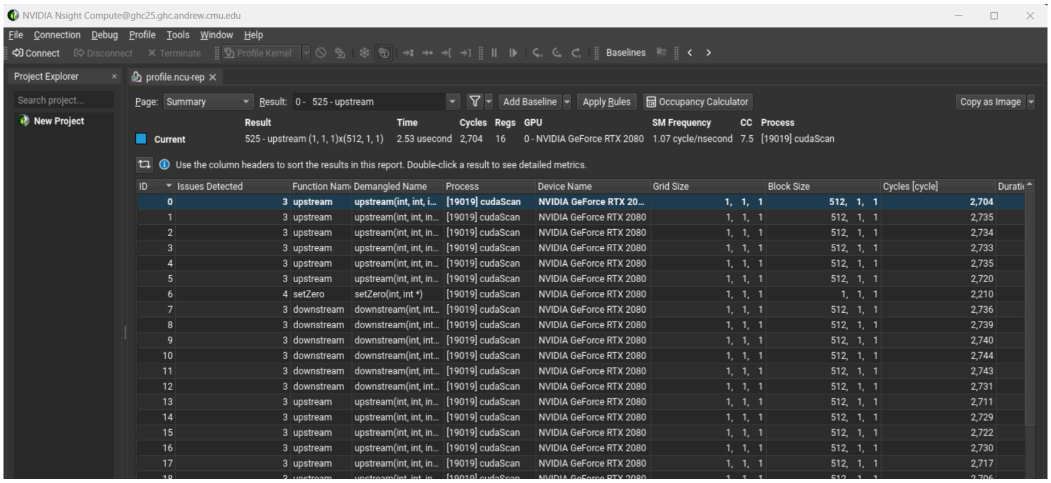
```
==PROF== Profiling "upstream(int, int, int *)" - 0: 0%....50%....100% - 8 passes
==PROF== Profiling "upstream(int, int, int *)" - 1: 0%....50%....100% - 8 passes
==PROF== Profiling "upstream(int, int, int *)" - 2: 0%....50%....100% - 8 passes
==PROF== Profiling "upstream(int, int, int *)" - 3: 0%....50%....100% - 8 passes
==PROF== Profiling "upstream(int, int, int *)" - 4: 0%....50%....100% - 8 passes
==PROF== Profiling "upstream(int, int, int *)" - 5: 0%....50%....100% - 8 passes
==PROF== Profiling "setZero(int, int *)" - 6: 0%....50%....100% - 8 passes
==PROF== Profiling "downstream(int, int, int *)" - 7: 0%....50%....100% - 8 passes
```

This should create a report file of the same name that you have chosen, and in your current directory you should see the file with the **.ncu-rep* file type. To view this file, we will use NVIDIA’s NCU UI:

Unset

```
ncu-ui <reportName.ncu-rep>
```

Upon entering this command, a popup like this should appear:


NVIDIA Nsight Compute@ghc25.ghc.andrew.cmu.edu
File Connection Debug Profile Tools Window Help
Connect Disconnect Terminate Profile Kernel Baselines
Project Explorer
Search project...
New Project
profile.ncu-rep
Filter
Add Baseline
Apply Rules
Occupancy Calculator
Copy as Image

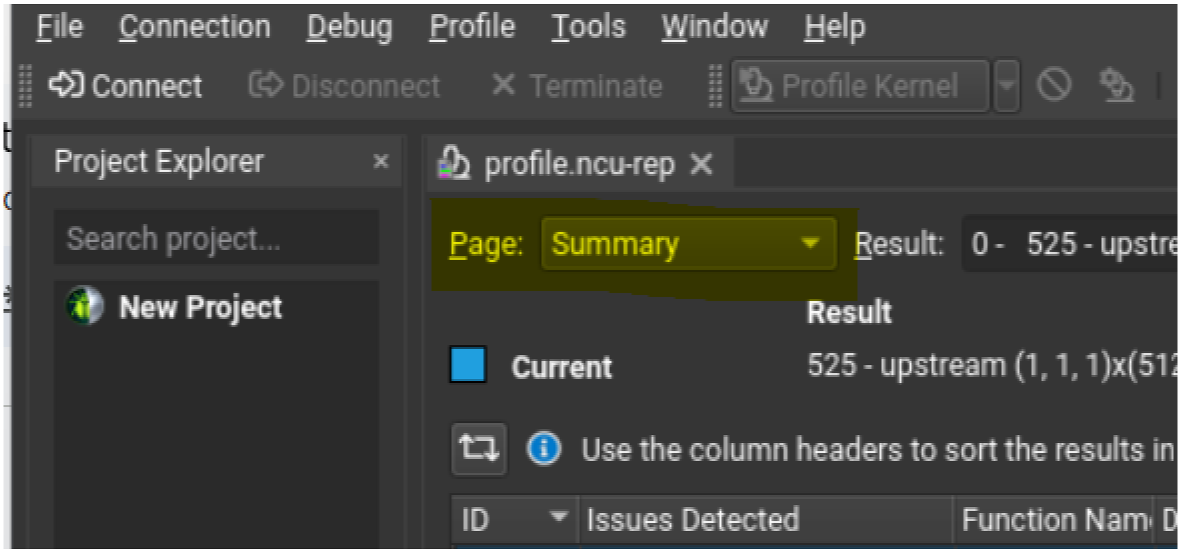
Page: Summary Result: 0 - 525 - upstream

	Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	525 - upstream (1, 1, 1)x(512, 1, 1)	2.53 usecond	2,704	16	0 - NVIDIA GeForce RTX 2080	1.07 cycle/nsecond	7.5	[19019] cudaScan

Use the column headers to sort the results in this report. Double-click a result to see detailed metrics.

ID	Issues Detected	Function Nam	Demangled Name	Process	Device Name	Grid Size	Block Size	Cycles [cycle]	Duratic
0	3	upstream	upstream(int, int, i...	[19019] cudaScan	NVIDIA GeForce RTX 20...	1, 1, 1	512, 1, 1	2,704	
1	3	upstream	upstream(int, int, in...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,735	
2	3	upstream	upstream(int, int, in...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,734	
3	3	upstream	upstream(int, int, in...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,733	
4	3	upstream	upstream(int, int, in...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,735	
5	3	upstream	upstream(int, int, in...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,720	
6	4	setZero	setZero(int, int *)	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	1, 1, 1	2,210	
7	3	downstream	downstream(int, int...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,736	
8	3	downstream	downstream(int, int...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,739	
9	3	downstream	downstream(int, int...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,740	
10	3	downstream	downstream(int, int...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,744	
11	3	downstream	downstream(int, int...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,743	
12	3	downstream	downstream(int, int...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,731	
13	3	upstream	upstream(int, int, in...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,711	
14	3	upstream	upstream(int, int, in...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,729	
15	3	upstream	upstream(int, int, in...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,722	
16	3	upstream	upstream(int, int, in...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,730	
17	3	upstream	upstream(int, int, in...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,717	
18	3	upstream	upstream(int, int, in...	[19019] cudaScan	NVIDIA GeForce RTX 2080	1, 1, 1	512, 1, 1	2,706	

Looking at the top left corner you should see a drop down menu like this:


File Connection Debug Profile Tools Window Help
Connect Disconnect Terminate Profile Kernel
Project Explorer
Search project...
New Project
profile.ncu-rep

Page: Summary Result: 0 - 525 - upstre

Result
Current 525 - upstream (1, 1, 1)x(51

Use the column headers to sort the results in

ID Issues Detected Function Nam D

Here you can choose different parts of the report that the profiler has collected. The *Details* page has the most understandable information, and definitions of terms used can be explained by hovering over the term. The *Source* page can be interesting to look at if lower-level source code is interesting to you. Feel free to explore other pages as well.

<u>Using Nsight Systems</u>

Note: Conduct these profilers on terminating code, ie. use the "-b" (bench) option for the render assignment so that the program will terminate. Also note for the *./render* executable make sure to include the "-r cuda" option such that the code is run through CUDA.

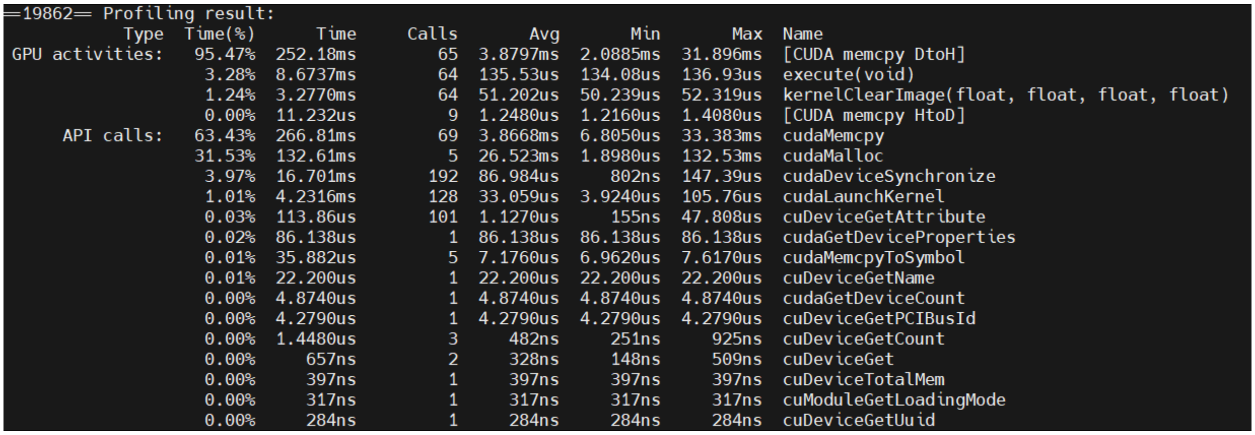
To view how long specific API calls and GPU activities take, we can use another part of NVIDIA's profiling tools called Nsight systems. To get a view of this data you can use:

Unset

```
nvprof <executable>
```

The format of the executable is identical to the example used above.

The result of this command should look like this:



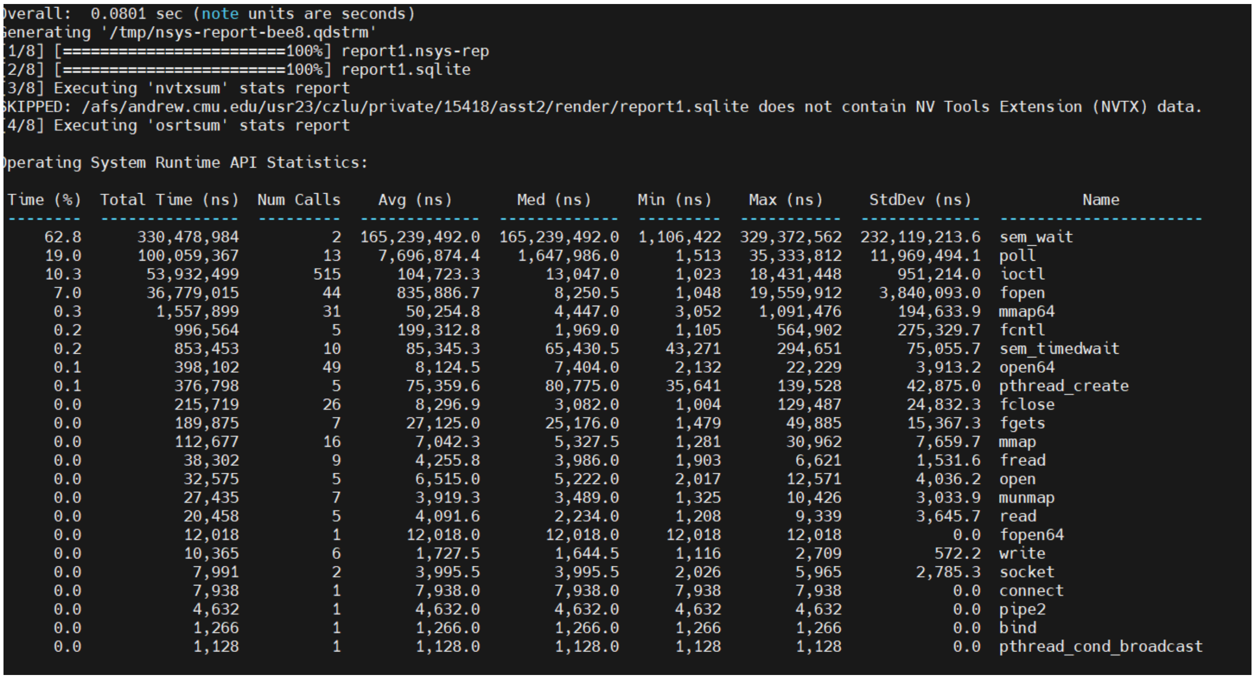
```
==19862== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   95.47%  252.18ms        65  3.8797ms  2.0885ms  31.896ms  [CUDA memcpy DtoH]
                    3.28%  8.6737ms        64  135.53us  134.08us  136.93us  execute(void)
                    1.24%  3.2770ms        64  51.202us  50.239us  52.319us  kernelClearImage(float, float, float, float)
                    0.00%  11.232us         9  1.2480us  1.2160us  1.4080us  [CUDA memcpy HtoD]
      API calls:   63.43%  266.81ms        69  3.8668ms  6.8050us  33.383ms  cudaMemcpy
                   31.53%  132.61ms         5  26.523ms  1.8980us  132.53ms  cudaMalloc
                    3.97%  16.701ms       192  86.984us     802ns  147.39us  cudaDeviceSynchronize
                    1.01%  4.2316ms       128  33.059us  3.9240us  105.76us  cudaLaunchKernel
                    0.03%  113.86us       101  1.1270us     155ns  47.808us  cuDeviceGetAttribute
                    0.02%  86.138us         1  86.138us  86.138us  86.138us  cudaGetDeviceProperties
                    0.01%  35.882us         5  7.1760us  6.9620us  7.6170us  cudaMemcpyToSymbol
                    0.01%  22.200us         1  22.200us  22.200us  22.200us  cuDeviceGetName
                    0.00%  4.8740us         1  4.8740us  4.8740us  4.8740us  cudaGetDeviceCount
                    0.00%  4.2790us         1  4.2790us  4.2790us  4.2790us  cuDeviceGetPCIBusId
                    0.00%  1.4480us         3     482ns     251ns     925ns  cuDeviceGetCount
                    0.00%     657ns         2     328ns     148ns     509ns  cuDeviceGet
                    0.00%     397ns         1     397ns     397ns     397ns  cuDeviceTotalMem
                    0.00%     317ns         1     317ns     317ns     317ns  cuModuleGetLoadingMode
                    0.00%     284ns         1     284ns     284ns     284ns  cuDeviceGetUuid
```

To get a similar but slightly more in-depth view of the information, you can use:

Unset

```
nsys profile --stats=true <executable>
```

You should see something like this:



```
verall:  0.0801 sec (note units are seconds)
enerating '/tmp/nsys-report-bee8.qdstrm'
[1/8] [========================100%] report1.nsys-rep
[2/8] [========================100%] report1.sqlite
[3/8] Executing 'nvtxsum' stats report
KIPPED: /afs/andrew.cmu.edu/usr23/czlu/private/15418/asst2/render/report1.sqlite does not contain NV Tools Extension (NVTX) data.
[4/8] Executing 'osrtsum' stats report

perating System Runtime API Statistics:

 Time (%)  Total Time (ns)  Num Calls    Avg (ns)       Med (ns)      Min (ns)    Max (ns)     StdDev (ns)            Name
 --------  ---------------  ---------  -------------  -------------  ---------  -----------  -------------  ----------------------
     62.8      330,478,984          2  165,239,492.0  165,239,492.0  1,106,422  329,372,562  232,119,213.6  sem_wait
     19.0      100,059,367         13    7,696,874.4    1,647,986.0      1,513   35,333,812   11,969,494.1  poll
     10.3       53,932,499        515      104,723.3       13,047.0      1,023   18,431,448      951,214.0  ioctl
      7.0       36,779,015         44      835,886.7        8,250.5      1,048   19,559,912    3,840,093.0  fopen
      0.3        1,557,899         31       50,254.8        4,447.0      3,052    1,091,476      194,633.9  mmap64
      0.2          996,564          5      199,312.8        1,969.0      1,105      564,902      275,329.7  fcntl
      0.2          853,453         10       85,345.3       65,430.5     43,271      294,651       75,055.7  sem_timedwait
      0.1          398,102         49        8,124.5        7,404.0      2,132       22,229        3,913.2  open64
      0.1          376,798          5       75,359.6       80,775.0     35,641      139,528       42,875.0  pthread_create
      0.0          215,719         26        8,296.9        3,082.0      1,004      129,487       24,832.3  fclose
      0.0          189,875          7       27,125.0       25,176.0      1,479       49,885       15,367.3  fgets
      0.0          112,677         16        7,042.3        5,327.5      1,281       30,962        7,659.7  mmap
      0.0           38,302          9        4,255.8        3,986.0      1,903        6,621        1,531.6  fread
      0.0           32,575          5        6,515.0        5,222.0      2,017       12,571        4,036.2  open
      0.0           27,435          7        3,919.3        3,489.0      1,325       10,426        3,033.9  munmap
      0.0           20,458          5        4,091.6        2,234.0      1,208        9,339        3,645.7  read
      0.0           12,018          1       12,018.0       12,018.0     12,018       12,018            0.0  fopen64
      0.0           10,365          6        1,727.5        1,644.5      1,116        2,709          572.2  write
      0.0            7,991          2        3,995.5        3,995.5      2,026        5,965        2,785.3  socket
      0.0            7,938          1        7,938.0        7,938.0      7,938        7,938            0.0  connect
      0.0            4,632          1        4,632.0        4,632.0      4,632        4,632            0.0  pipe2
      0.0            1,266          1        1,266.0        1,266.0      1,266        1,266            0.0  bind
      0.0            1,128          1        1,128.0        1,128.0      1,128        1,128            0.0  pthread_cond_broadcast
```

This will give you another breakdown of all the API calls and other stats that occurred during your program's runtime.

This command does output two files, but the information in said files is not practical for our purposes.