Nvidia Nsight Systems nsys

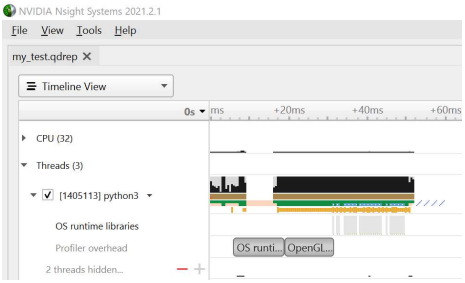
July 26, 2021 11:52 AM

Docs	https://docs.nvidia.com/nsight-systems/UserGuide/index.html

--export=sqlite	From <https://stackoverflow.com/questions/66626185/how-do-i-get-my-kernel-execution-times-from-the-sqlite3-output-of-nsight-systems>
Example	nsys profile --force-overwrite=true --duration=20 --stats=true -o my_test python3 jaxviz/examples/cartpole_mlp/99_full/main.py
CLI command	nsys profile -w true — show target process' output -t cuda,nvtx,osrt,cudnn,cublas — APIs to be traced -s cpu — capture CPU events -o {n_env}_env — output file name -f true — force overwrite previous profile -x true — stop collecting when all processes have exited --capture-range=cudaProfilerApi — start profiling at CUDA profiler start to limit range --stop-on-range-end=true — stop on range end --export sqlite — export data into .sqlite file run_model torch -e {n_env} {n_env} -s 10 -i 1 -r 1 -p — process to profile (-p flag for profiling)

Install and Test

1. If you're using a docker container you might need to rebuild the container with "docker run … --cap-add=SYS_ADMIN" to get perf privileges. You might be able to skip this.
2. Install CLI component on target machine to collect profile:
   - apt install cuda-nsight-systems-11-4 (or whatever 11-x is appropriate for your Nvidia driver)
     - Should give the command "nsys"
3. Install GUI component on desktop to view profile:
   - Nsight Systems 2021.5 (Windows Host)
   - ~~Nsight Systems 2021.2 (Windows Host)~~
     ~~From <https://developer.nvidia.com/gameworksdownload#?dn=nsight-systems-2021-2-1-58>~~
     - Should give the program "Nsight Systems"
4. Test that it works by profiling hello world in python:
   - nsys profile -o test python3 -c "print('hello world')"
     - This should make the file "test.qdrep"
5. Copy test.qdrep to your desktop machine. You could use SCP:
   - scp remote_hostname:my_test.qdrep ./local_desktop_folder
6. On your desktop open test.qdrep with Nsight Systems. It should look something like this:



Bonus: video about how to use Nsight Systems
Profiling GPU Applications with Nsight Systems

More complicated test

```
import time
import torch
import nvtx
@nvtx.annotate("style A", color="purple")
def foo():
    for i in range(5):
        print(f"loop {i}")
        with nvtx.annotate(f"style B {i}", color="green"):
            time.sleep(0.1)
torch.cuda.cudart().cudaProfilerStart()
```

```
torch.cuda.nvtx.range_push("style C")
foo()
torch.cuda.nvtx.range_pop()
torch.cuda.cudart().cudaProfilerStop()
```

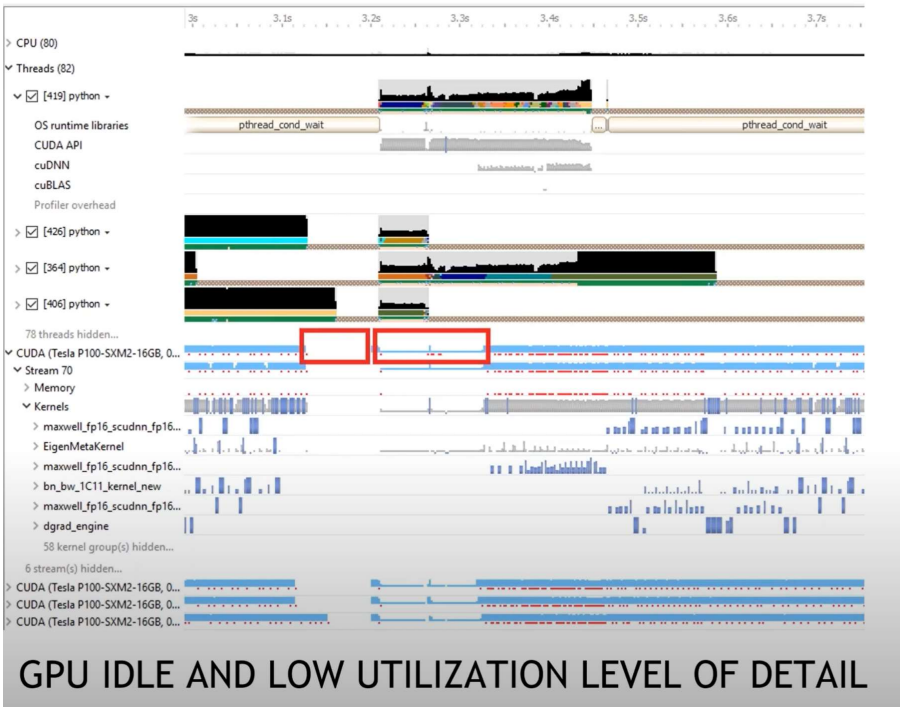
Which gives:

Another example profiling just one portion of code

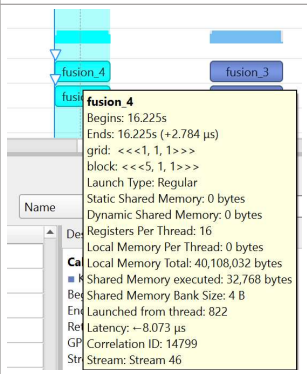
```
import time
import torch
import nvtx

def foo():
    for i in range(5):
        print(f"iteration {i}")
        if i == 3:
            torch.cuda.cudart().cudaProfilerStart()
            with nvtx.annotate(f"iteration {i}", color="green"):
                time.sleep(0.1)
            torch.cuda.cudart().cudaProfilerStop()
        else:
            time.sleep(0.1)
foo()
```

Example NSYS command found in RLScope Simulator experiments

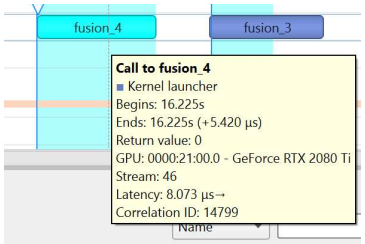
Code	cmd.extend(["nsys", "profile" , "--show-output=true" # show target process' output , "--trace=cuda,nvtx,osrt,cudnn,cublas" # APIs to be traced , "--sample=cpu" # capture CPU events , f"--output={nsys_output}" # output file name , "--force-overwrite=true" # force overwrite previous profile , "--stop-on-exit=true" # stop collecting when all processes have exited , "--capture-range=nvtx" # mark profiling range with nvtx , "--nvtx-capture=profile_range" # capture the nvtx range named "profile_range" , "--stop-on-range-end=true" # stop when the nvtx range ends # NOTE: always generate sqlite file for simplicity of CLI. , "--export=sqlite" # export data into .sqlite file]) # Don't limit captured ranges to nvtx registered strings child_env["NSYS_NVTX_PROFILER_REGISTER_ONLY"] = "0" https://github.com/UofT-EcoSystem/rlscope_simulator_experiments/blob/544eaca19ba338a8e6181bc48eb34347b037a022/simulator_experiments/plot/run_model.py#L1148



Get SQL data from NSight Systems

Collect trace	nsys profile --force-overwrite=true --duration=20 **--export=sqlite** -o my_test python3 jaxviz/examples/cartpole_mlp/99_full/main.py
Kernel runtime	 $ sqlite3 -csv my_test.sqlite 'SELECT end-start AS duration FROM CUPTI_ACTIVITY_KIND_KERNEL WHERE correlationId = "**14799**";' **2784**

Launch runtime



$ sqlite3 -csv my_test.sqlite 'SELECT end-start AS duration FROM CUPTI_ACTIVITY_KIND_RUNTIME WHERE correlationId = "**14799**";'
5420

Documentation

Located under folder where Nsight Systems is installed "documentation>nsys-exporter"

Correlate CUDA kernel launches with CUDA API kernel launches

```
ALTER TABLE CUPTI_ACTIVITY_KIND_RUNTIME ADD COLUMN name TEXT;
ALTER TABLE CUPTI_ACTIVITY_KIND_RUNTIME ADD COLUMN kernelName TEXT;

UPDATE CUPTI_ACTIVITY_KIND_RUNTIME SET kernelName =
    (SELECT value FROM StringIds
    JOIN CUPTI_ACTIVITY_KIND_KERNEL AS cuda_gpu
        ON cuda_gpu.shortName = StringIds.id
        AND CUPTI_ACTIVITY_KIND_RUNTIME.correlationId = cuda_gpu.correlationId);

UPDATE CUPTI_ACTIVITY_KIND_RUNTIME SET name =
    (SELECT value FROM StringIds WHERE nameId = StringIds.id);

-- Select 10 Longest CUDA API ranges that resulted in kernel execution.
SELECT name, kernelName, start, end FROM CUPTI_ACTIVITY_KIND_RUNTIME
    WHERE kernelName IS NOT NULL ORDER BY end - start LIMIT 10;
```

Results:

```
name                  kernelName              start      end
--------------------  ----------------------  ---------  ----------
cudaLaunchKernel_v7000  RadixSortScanBinsKernel  658863435  658868490
cudaLaunchKernel_v7000  RadixSortScanBinsKernel  609755015  609760075
cudaLaunchKernel_v7000  RadixSortScanBinsKernel  632683286  632688349
cudaLaunchKernel_v7000  RadixSortScanBinsKernel  606495356  606500439
cudaLaunchKernel_v7000  RadixSortScanBinsKernel  603114486  603119586
cudaLaunchKernel_v7000  RadixSortScanBinsKernel  802729785  802734906
cudaLaunchKernel_v7000  RadixSortScanBinsKernel  593381170  593386294
cudaLaunchKernel_v7000  RadixSortScanBinsKernel  658759955  658765090
cudaLaunchKernel_v7000  RadixSortScanBinsKernel  681549917  681555059
cudaLaunchKernel_v7000  RadixSortScanBinsKernel  717812527  717817671
```

Limiting profiling to a NVTX range

CLI command

With cuda profiler:

nsys profile \	
-w true \	show target process' output
-t cuda,nvtx,osrt,cudnn,cublas \	APIs to be traced
-s cpu \	capture CPU events
-o profile_report \	output file name
-f true \	force overwrite previous profile output
-x true \	stop profiling on exit
--capture-range=cudaProfilerApi \	start profiling at CUDA profiler start to limit range
--stop-on-range-end=true \	stop on range end
--export sqlite \	export data into .sqlite file
run_model torch -e {pow_env} {pow_env} -s {n_step} -i 2 -r 1	process to profile (-p flag for profiling)

With nvtx range

NSYS_NVTX_PROFILER_REGISTER_ONLY=0	don't limit profiling to only nvtx-registered strings
nsys profile \	
-w true \	show target process' output
-t cuda,nvtx,osrt,cudnn,cublas \	APIs to be traced
-s cpu \	capture CPU events
-o profile_report \	output file name
-f true \	force overwrite previous profile output
-x true \	stop profiling on exit
--capture-range=nvtx \	mark profiling range with nvtx to limit range
--nvtx-capture=profiling_range \	Start capture on the range "profiling_range"
--stop-on-range-end=true \	stop on range end
--export sqlite \	export data into .sqlite file

	run_model jax -e {pow_env} {pow_env} -s {n_step} -i 2 -r 1	process to profile

Profiling and range stack in PyTorch

Cudart profiler:

```
torch.cuda.cudart().cudaProfilerStart()
torch.cuda.nvtx.range_push("profiling")
runner.run_repetitions()
torch.cuda.nvtx.range_pop()
torch.cuda.cudart().cudaProfilerStop()
```

(use torch.cuda.nvtx.range_push/range_pop instead of nvtx.start_range/end_range)

Capturing a range (Nsight Systems CLI)

-c	--capture-range	none, cudaProfilerApi, nvtx	none	When -c cudaProfilerApi (or nvtx) is used, profiling will start only when cudaProfilerStart API is invoked or the specified NVTX range (specified using -p/--nvtx-capture) is started in the application.
	--stop-on-range-end	true,false	true	Stop profiling when the capture range ends. Applicable only when used along with --capture-range option.
-p	--nvtx-capture	range@domain,range,range@		Specify NVTX capture range. See below for details. This option is applicable only when used along with --capture-range=nvtx.

(PyTorch forum: using Nsys to profile GPU workloads) https://dev-discuss.pytorch.org/t/using-nsight-systems-to-profile-gpu-workload/59
(Profiling PyTorch with PyProf (& Nsys)) https://docs.nvidia.com/deeplearning/frameworks/pyprof-user-guide/profile.html#profile-with-nsight-systems

[MLCommons] DLRM Workload speedup

From <https://mail.google.com/mail/u/0/#sent/FMfcgzGljvSxbDRCRtBrIJkZhTrJqFMN>

Hi Rakshith,

Great work!

DLRM workloads often have complex parallelization strategies. I'm actually reviewing a conference paper about a "4D" parallelization strategy for DLRM. Hopefully our implementation will be much simpler, but still fast enough to quickly benchmark optimizers.

Here are my suggestions for profiling.

Before trying Nsight tools you might take another look at these PyTorch profiling articles:

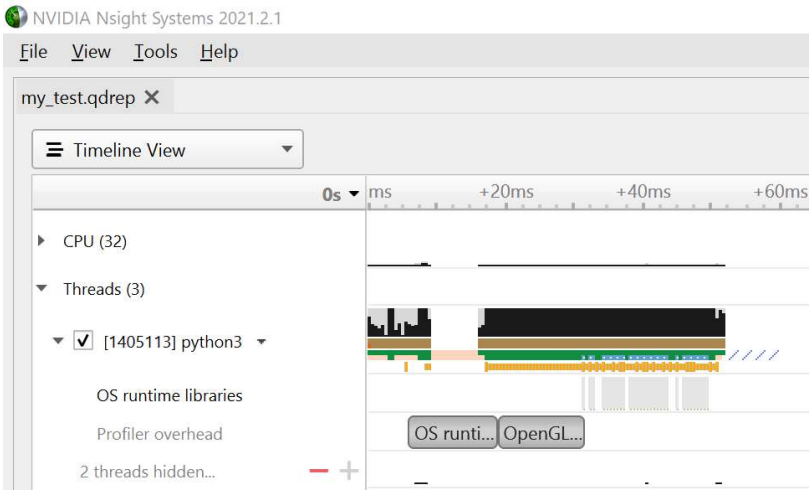
- Using profiler to analyze long-running jobs to avoid slow and very large trace files. https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html
- The “Step Time Breakdown” shows distribution of time spent over different categories of execution. https://pytorch.org/tutorials/intermediate/tensorboard_profiler_tutorial.html

Nsight profiling tools are more powerful but not as easy to use as PyTorch tools.

Here is how to use Nsight tools. Use NVTX annotations to attach human-friendly names to important sections of your DLRM implementation. Then those human-friendly names that show up in Nsight Systems timeline trace, so that we can see what sections of code are taking the majority of the time. The trace is very detailed.

Nsight Systems Installation and Usage

1. If you're using a docker container you might need to rebuild the container with "docker run … --cap-add=SYS_ADMIN" to get perf privileges. You might be able to skip this.
2. Install CLI component on target machine to collect profile:
   - apt install cuda-nsight-systems-11-1 (or whatever 11-x is appropriate for your Nvidia driver)
     - Should give the command "nsys"
3. Install GUI component on desktop to view profile:
   - Nsight Systems 2021.2 (Windows Host)
     From <https://developer.nvidia.com/gameworksdownload#?dn=nsight-systems-2021-2-1-58>
     - Should give the program "Nsight Systems"
4. Test that it works by profiling hello world in python:
   - nsys profile -o test python3 -c "print('hello world')"
     - This should make the file "test.qdrep"
5. Copy test.qdrep to your desktop machine. You could use SCP:
   scp remote_hostname:my_test.qdrep ./local_desktop_folder
6. On your desktop open test.qdrep with Nsight Systems. It should look something like this:



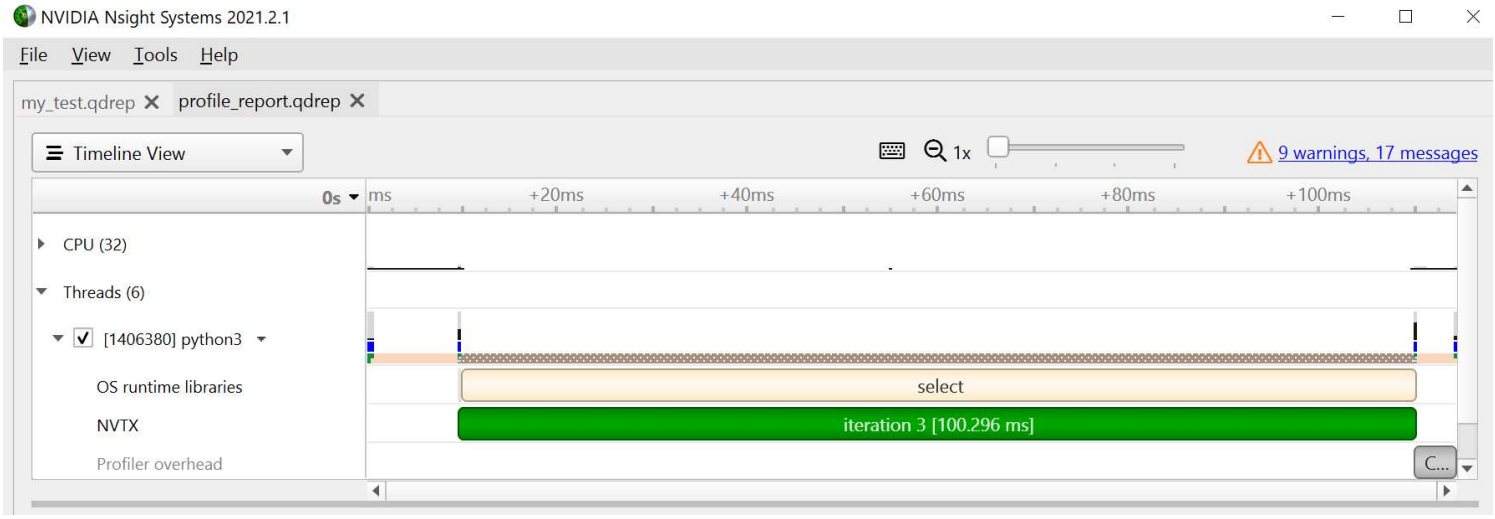
If that works then you are all set to profile a real workload.
Here's a more sophisticated example showing how we can profile just a small section of code:

```
import time
import torch
import nvtx

def foo():
    for i in range(5):
        print(f"iteration {i}")
        if i == 3:
            torch.cuda.cudart().cudaProfilerStart()
            with nvtx.annotate(f"iteration {i}", color="green"):
                time.sleep(0.1)
            torch.cuda.cudart().cudaProfilerStop()
        else:
            time.sleep(0.1)
foo()
```

For this we want to use nsys with more powerful options:

```
nsys profile \
   -w true \
   -t cuda,nvtx,osrt,cudnn,cublas \
   -s cpu \
   -o profile_report \
   -f true \
   -x true \
   --capture-range=cudaProfilerApi \
   --stop-on-range-end=true \
   python3 test.py
```

Which should result in the following trace:



The range of code that will be profiled by Nsight Systems lies in between "cudaProfilerStart()" and "cudaProfilerStop()". Keep this range small and profiling will go faster.

You should have at least one human-friendly NVTX annotation name so that you can find your code in the profiler timeline. Using code like this,
"with nvtx.annotate(f"iteration {i}", color="green"):" you can name different parts of your code. Here the name "iteration" shows up in green in the profiler output.

Here is some documentation on NVTX.
Here is some documentation on Nsight Systems and a video explainer.

I'm happy to take a look at any of your profiler output. Let me know if you have any questions. I'm always looking for better ways to find bottlenecks.

Best wishes,
Dan

From <https://mail.google.com/mail/u/0/#sent/FMfcgzGljvSxbDRCRtBrlJkZhTrJqFMN>