Profiling Deep Learning with Nsight Systems

Daniel Horowitz
Director of Engineering
Platform Developer Tools
Agenda

● Monitors vs Profilers
● Profiling on a single node
  ○ Overview
  ○ Basic features
  ○ GPU investigations
  ○ CPU investigations & features
  ○ Common bubble recovery tactics
● Profiling multi-node
Monitors vs Profilers

- Differ in the target users & intent = tradeoffs in design
  - Pick the right tool for the job!!!
  - Don’t expect one to be fully usable as the other
  - Gray area, hybrid amalgamations, but usually still tradeoffs

- Monitors
  - Goal: Coarsely observe quality, utilization, progress
  - Users: Machine and cluster admins, users of taskman, top, netstats, etc
  - Reports at a low rate (~10 - 1 hz)
    - Sensible to observer reacting, low overhead, smooth out numbers
    - Does not correlate back to code
  - Track issues back to jobs
    - Maybe minutes or seconds but often not root cause
  - Ex: DCGM

- Profilers
  - Goal: Aid program optimization
  - Users: Engineers looking to relating back to areas of code
  - Many different breeds of profiler!!!
    - Levels of observibility
    - Overhead trade-offs
  - Often not designed for 100% up-time and continual reporting like a monitor
  - Ex: Nsight Systems and Nsight Compute
Profiler-Driven Optimization Workflow

Profile

Analyze

Optimize

Iterate until desired performance is achieved
Nsight Systems -
Analyze application
algorithm system-wide
CPU, GPU, CUDA, Graphics

Nsight Compute -
Debug/ Optimize
CUDA kernels

Nsight Graphics -
Debug/ Optimize
Graphics shaders & frames
Timeline looks like a spectrogram when zoomed out
Ex: DL model training, 2 iterations, GPU & CPU
Zooming in transforms many row graphs into ranges
Tuning an Orchestra of Tasks
Why start with Nsight Systems?

It’s designed to allow you to...

See the big picture
See how asynchronous CPUs, GPUs, NICs, and software are interacting
Who is stuck on whom
Measure the higher-level costs to pick the best opportunities
Avoid work based on intuition & false-positive indicators
  Don’t assume GPU bound and skip ahead to other tools
  Some synchronization oriented issues can look like GPU bound
Statistics alone often aren’t enough info to understand and resolve the issue

WARNING: The presenter & slides will often refer to “Nsight Systems” as Nsys or NSYS
Overview

• System-wide application algorithm tuning
• Locate optimization opportunities
  • Visualize millions of events on a very fast GUI timeline
  • See gaps of unused CPU and GPU time
• Balance your workload across multiple GPUs, CPUs, NICs
  • GPU streams, kernels, memory transfers, etc
  • CPU algorithms, utilization, and thread state
• UX: CLI, GUI timeline, statistics, data export
  • Ex: Collect nsys-rep w/CLI on cluster, SCP to PC to view, mine either place
• Multi-platform: Linux & Windows, x86-64, ARM server, Tegra
  • Mac (host-only)
Timeline Features

GPU

- GPU HW metrics sampling
- CUDA GPU-side kernel and mem-op ranges correlated to CPU API calls
- Libraries: cuBLAS, cuDNN, cuDF, TensorRT

CPU, OS, & application

- Thread state, migrations, and call-stacks
- OS runtime long call trace (>1us, pthread, glibc → mmap, file & IO, …)
  - Call-stack backtraces (>80us)
- ftrace or WDDM & ETW (page faults, signal, interrupts, …)

Code Annotations APIs

- NVTX

Networking

- UCX, MPI (OpenMPI & MPICH), OpenSHMEM, NVSHMEM, NCCL trace
- NVIDIA NIC/HCA metrics sampling (Infiniband & Ethernet)
Wait!!! I just want stats about my DNN 😞

You have barely mentioned Deep Learning!!!

Each DL framework has its own tools, so we’ll discuss how to go deeper
Many DL frameworks have their DNN layer execution instrumented with NVTX
Sometimes just the NVIDIA container if the framework doesn’t accept it upstream
Nsys has the relationships: DNN layer → CUDA Launch → GPU CUDA Kernel Execution
Nsys can export to SQLite
Nsys has python scripts & documentation on how to analyze that database
If that’s all you want, that’s the easy path
But we’re about to go deeper and show you how it’s executing!!!
So you can visually investigate, craft better stats, and create your own expert systems
CUDA

API launch to GPU kernel correlation
CPU-GPU correlation & location assistance
Code annotations APIs

NVTX = NVIDIA Tools eXtensions

Example: Visual Molecular Dynamics (VMD) algorithms visualized with NVTX on CPU
NVTX ranges project from CPU onto the GPU CUDA streams

Example: Visual Molecular Dynamics (VMD) algorithms visualized with NVTX on GPU
PyTorch

DNN layer annotations are disabled by default

Add the following: “with torch.autograd.profiler.emit_nvtx():”

TensorRT is also annotated already if that is the backend you are using

You can also add NVTX to your python script manually:

https://github.com/pytorch/pytorch/blob/master/torch/cuda/nvtx.py
DNN layers are annotated **by default** with NVTX in NVIDIA TF containers!


[https://github.com/NVIDIA/tensorflow](https://github.com/NVIDIA/tensorflow)

TF_DISABLE_NVTX_RANGES=1 if you want to disable for production

Adding more detail to the timeline for setup, eager mode, tf.data.Dataset.from_generator, etc

[https://github.com/NVIDIA/NVTX/tree/dev/python](https://github.com/NVIDIA/NVTX/tree/dev/python)
nsys profile --trace=cuda,nvtx,osrt,cudnn,cublas --backtrace=dwarf
--capture-range=cudaProfilerApi --gpu-metrics-devices=all --output=oft-profile-dwarf4
sh scripts/expt-singleGPU.sh --profile 50 --profile_start 5000 --profile_epoch 1

Nsight Systems CLI command
Select APIs to trace
Enable GPU memory use tracking (but there is extra overhead)
Collect thread call-stack sample backtraces via DWARF info - deeper but more expensive to collect
Trigger collection on cudaProfilerStart API in application, or consider timer-based options
GPU metrics sampling at default 10khz
Name the report file
Application command - plus arguments for when to start profiling


srun nsys profile ... required on multi-node or multi-container
nsys profile mpirun ... optional on single node to produce a single report

Nsight Systems CLI command line example
PyTorch Transformer with NVTX ranges projected onto the GPU
PyTorch Transformer with NVTX ranges projected onto the GPU
Example: NVTX on GPU in PyTorch transformer model in eager mode (ie non-hybrid)
TensorFlow Resnet50 DNN nodes as NVTX ranges projected onto the GPU
NVSHMEM & NCCL
NVIDIA DALI trace
Data Loading Library
Core Investigation Strategy

- **What’s HOT?**
  - Will it be easier to shrink what I have?
  - This is where MOST people concentrate. ...intuitive but not always better

- **What’s COLD?**
  - Will it be easier to take advantage of the something unused?
  - Free money? Yes please!

- Is it doing what I intended & budgets? (hint: often not as well as you thought)

- Cold spots are often clear, measurable opportunities!!!
  - How can I remove or fill them?
  - Where do I have incorrect/unnecessary/unexpected dependencies/synchronization?

- **Hot spots**
  - Might be parallelizable?
  - Might not be shrinkable without compromising accuracy, memory, etc
GPU Bubble Detective

- Find the crime (cold or cool spot)
- Use correlation to track back to the CPU
  - Select surrounding GPU CUDA ops
- Investigate what was in the gap
  - Thread call-stack backtrace samples
  - OS Runtime long function backtraces
  - API & library traces
  - User-coded annotations
CUDA trace based GPU idle and low utilization level of detail
Investigate by select kernels around bubble to find related CPU range
CUDA GPU utilization graph is based on percentage time coverage

Zooming in reveals gaps where there were valleys

CUDA streams eventually convert from graphs to ranges
GPU Metrics Sampling - how to interpret it
Interpreting GPU Metrics Sampling

- More info, no trace overhead, collected device-wide OOP
  - But no kernel names

- GR Active ⇒ It’s doing some work
  - % GPU has any SM active (or NVENC, NVDEC, graphics)

- SM Active ⇒ How well is it using the width of the GPU
  - If low, increase batch sizes or look at kernel grid dimension

- SM Instructions Issued ⇒ Is it performing a lot of instructions
  - or might I be waiting on memory if low
  - Not enough warps to cover memory latency; larger kernel block dimensions can help

- SM Instructions tensor active ⇒ Using very faster special hardware
  - performance gains but slightly counter SM instructions can drop (vary by architecture)
  - can be limited by SM shared memory & waiting for loads

- Warp occupancy ⇒ Ratio of SM code types
  - Don’t optimize for this! Ultra-optimized kernels don’t always maximize warps!!! Ex: cuBLAS

- Memory and bus activity

- NOTE 1: Requires disabling DCGM and any DL framework built-in profiler
GPU Metrics Sampling
Ex: TensorFlow2 ResNet50
GPU Metrics Sampling - Mask-RCNN
GPU Metrics Sampling - NCCL using NVLink
Optimization Tactics (1)

- Ensure using tensor cores & correct format to avoid conversions/transposes
- Increase batch/grid sizes to more efficiently use the GPUs’ width
- Conventional parallelism
  - Increase framework’s worker threads (CLI args)
  - Python async, await, future, tasks, Dask, …
  - C/C++ OpenMP, OpenACC, pthreads, boost::asio, std::async, …
- Parallel pipelining - each stage can run in parallel
  - Are there a data dependencies between stages? No? Parallelize!
  - Ex: prefetch next batch/iteration of data while current batch/iteration is executing
  - Ex: within loader: net transfer, map, parse, load & transform, upload to GPU
- Reorder - could i do that sooner?
  - Prefetch - load data sooner & in parallel so it’s there before needed
- Fusing tiny kernels, copies, or memsets, or use cudaGraphs
- Overlap training (or inference)
  - CUDA MPS to share contexts & avoid switch overhead
  - Possibly difficult to fix both into memory.
Optimization Tactics (2)

- Pass buffers by pointer - avoid copies
- Multi-buffering - don’t make everyone wait on the same piece of memory
  - Often referred to as double or triple buffering; consider swap patterns
- Avoid returning data to CPU
- Avoid CPU pageable memory (prefer pinned / page-locked)
- Avoid unnecessary synchronizes
  - Avoid cuda*Synchronize functions, use cudaStreamWaitEvent instead
  - Avoid synchronous memory operation
  - Avoid CUDA default stream (if multi-stream)
- Pre-allocate memory, or use recycling tactics
- Minimize CUDA managed memory page faults on CPU & GPU (use prefetch)
- …
AWS Blog: Deepset achieves a 3.9x speedup and 12.8x cost reduction for training NLP models by working with AWS and NVIDIA
Deepset achieves a 3.9x speedup and 12.8x cost reduction for training NLP models by working with AWS and NVIDIA


Cliff-notes

- Heavy use of Nsight Systems
- Switched from torch.nn.DataParallel (DP) to DistributedDataParallel (DDP)
- Enabled larger batch sizes by switching to Automatic Mixed Precision (AMP)
- Introduced a StreamingDataSilo & DALI to prefetch data

...
Fusion opportunities

CPU launch cost + small GPU work size $\approx$ GPU sparse idle

This can apply to DNN nodes/layers
cudaMemcpyAsync behaving synchronous
Device to host pageable memory
Mitigate with pinned memory
Example GPU idle caused by stream synchronization
CPU & OS

For cases not caused by CUDA API usage or clarified by NVTX
Permissions

Some features are tied to OS permissions

- CPU thread state, core occupancy, user-space call-stack periodic sampling
  - Paranoid level too high
  - Container SECCOMP blocking perf_event_open
  - OS kernel samples require even lower paranoid levels and/or sudo
- ftrace
  - CAP_SYS_ADMIN | CAP_SYS_PERFMON

See online Nsight Systems documentation, and UI warnings
https://docs.nvidia.com/nsight-systems/InstallationGuide/index.html#linux-requirements
OS Runtime API Trace

Example: Mask-RCNN

Map/unmap hiccups
Mitigate by pipelining
- Map 1 batch ahead
- Unmap last batch
- Swap pointers here instead
OS runtime trace (OSRT)
Includes backtraces of long running functions
FTrace
Example demonstrates interrupts
Function table shows statistics from periodic call-stack backtraces
CLI statistics and export
DL Rank Data Processing Imbalances

- **Hurry up and wait**
  - If anyone takes longer to reach the all-reduce then everyone is stuck!!!

- **Load time**
  - Did this batch take longer than the norm to load?
  - If parallel, did any rank have to wait?

- **Processing time**
  - Did this rank take longer than other

- **Some remedies**
  - Fix the data
  - 1 JPEG among PNGs or 1 MP3 among WAVs?
  - Wrong resolution, sample rate, precision, …
  - Reorganize your batch order

- **A perfect job for scripts & statistics**
<table>
<thead>
<tr>
<th>Time(%)</th>
<th>Time (ns)</th>
<th>Calls</th>
<th>Avg (ns)</th>
<th>Min (ns)</th>
<th>Max (ns)</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.6</td>
<td>1719858838790</td>
<td>66612</td>
<td>258198.5</td>
<td>28064</td>
<td>132759</td>
<td>nhwc_batch_norm fwd</td>
</tr>
<tr>
<td>11.3</td>
<td>141283434514</td>
<td>20875</td>
<td>708763.9</td>
<td>168352</td>
<td>1780335</td>
<td>nhwc_batch_norm bd_add relu</td>
</tr>
<tr>
<td>18.4</td>
<td>131686394681</td>
<td>48158</td>
<td>326227.4</td>
<td>177824</td>
<td>1342740</td>
<td>volta_fp16_s84cudnn_fp16_128x128_1dg8_dgrad_f2f_exp_interior_nhwc tt_v1</td>
</tr>
<tr>
<td>18.1</td>
<td>12608112728</td>
<td>42745</td>
<td>2508133.1</td>
<td>177824</td>
<td>764812</td>
<td>volta_fp16_s84cudnn_fp16_128x128_1dg8_relu_f2f_exp_interior_nhwc tt_v1</td>
</tr>
<tr>
<td>9.3</td>
<td>11744851567</td>
<td>41418</td>
<td>283568.0</td>
<td>167880</td>
<td>755651</td>
<td>volta_fp16_s84cudnn_fp16_128x128_1dg8_ugrad_idx_exp_interior_nhwc nt</td>
</tr>
<tr>
<td>7.1</td>
<td>8935938081</td>
<td>23848</td>
<td>374678.8</td>
<td>229856</td>
<td>895710</td>
<td>volta_s84cudnn_fp16_04x4_sliced1x4_1dg8_ugrad_idx_exp_interior_nhwc nt</td>
</tr>
<tr>
<td>6.2</td>
<td>782977120</td>
<td>41400</td>
<td>189978.7</td>
<td>20800</td>
<td>135672</td>
<td>nhwc_batch_norm bd relu</td>
</tr>
<tr>
<td>4.1</td>
<td>5145597181</td>
<td>1320</td>
<td>372529.8</td>
<td>325513</td>
<td>524159</td>
<td>volta_fp16_s84cudnn_fp16_128x128_1dg8_relu_f2f_exp small_nhwc tt_v1</td>
</tr>
<tr>
<td>3.8</td>
<td>4796808092</td>
<td>12551</td>
<td>382319.8</td>
<td>311649</td>
<td>775264</td>
<td>volta_fp16_s84cudnn_fp16_128x128_1dg8_dgrad_f2f_exp small_nhwc tt_v1</td>
</tr>
<tr>
<td>2.9</td>
<td>3591130531</td>
<td>482226</td>
<td>7463.5</td>
<td>1760</td>
<td>868905</td>
<td>dclQuantInvJogKernel</td>
</tr>
<tr>
<td>2.4</td>
<td>2660957684</td>
<td>5818</td>
<td>593667.5</td>
<td>124767</td>
<td>143188</td>
<td>nhwc_batch_norm bd</td>
</tr>
<tr>
<td>2.2</td>
<td>2139970797</td>
<td>162480</td>
<td>150992.0</td>
<td>116</td>
<td>173012</td>
<td>yclc_tn_formatKernel rol</td>
</tr>
<tr>
<td>1.6</td>
<td>2033353889</td>
<td>4768</td>
<td>533910.1</td>
<td>381407</td>
<td>784892</td>
<td>dgrad l1 x stride 2x2</td>
</tr>
<tr>
<td>1.5</td>
<td>1843354548</td>
<td>1256</td>
<td>1467631.8</td>
<td>1451800</td>
<td>1691492</td>
<td>pooling by max nhwc kernel</td>
</tr>
<tr>
<td>1.2</td>
<td>1565495722</td>
<td>3768</td>
<td>414404.1</td>
<td>397112</td>
<td>488321</td>
<td>volta_fp16_s84cudnn_fp16_256x64_1dg8_relu_f2f_exp small_nhwc tt_v1</td>
</tr>
<tr>
<td>1.2</td>
<td>1547470365</td>
<td>3768</td>
<td>408957.1</td>
<td>341520</td>
<td>766834</td>
<td>volta_s84cudnn_fp16_256x64_1dg8_dgrad_f2f_exp small_nhwc tt_v1</td>
</tr>
<tr>
<td>1.1</td>
<td>1373574502</td>
<td>5214</td>
<td>546805.1</td>
<td>403785</td>
<td>665834</td>
<td>dgrad id</td>
</tr>
<tr>
<td>1.0</td>
<td>1267695893</td>
<td>5214</td>
<td>504248.0</td>
<td>489289</td>
<td>135451</td>
<td>volta_fp16_s84cudnn_fp16_256x64_1dg8_relu_f2f_exp small_nhwc tt_v1</td>
</tr>
<tr>
<td>1.0</td>
<td>1235618995</td>
<td>5072</td>
<td>26691.0</td>
<td>1112</td>
<td>1235618</td>
<td>MappKernel</td>
</tr>
<tr>
<td>1.0</td>
<td>1008970218</td>
<td>5769</td>
<td>139906.3</td>
<td>59206</td>
<td>735965</td>
<td>mapped_generator kernel</td>
</tr>
<tr>
<td>0.9</td>
<td>90565506</td>
<td>65290</td>
<td>135992.0</td>
<td>64800</td>
<td>123808</td>
<td>convertTensorKernel</td>
</tr>
<tr>
<td>0.9</td>
<td>893426559</td>
<td>65290</td>
<td>352604.3</td>
<td>192223</td>
<td>315069</td>
<td>BatchedSeparableResampleKernel</td>
</tr>
<tr>
<td>0.7</td>
<td>868771683</td>
<td>1257</td>
<td>691146.4</td>
<td>670495</td>
<td>821372</td>
<td>first_layer ugrad kernel</td>
</tr>
<tr>
<td>0.7</td>
<td>859482390</td>
<td>1257</td>
<td>698905.9</td>
<td>59985</td>
<td>739230</td>
<td>first_layer fud kernel</td>
</tr>
<tr>
<td>0.7</td>
<td>854624830</td>
<td>1257</td>
<td>698905.9</td>
<td>669880</td>
<td>726048</td>
<td>volta_fp16_s84cudnn_fp16_256x64_1dg8_relu_f2f_exp_interior_nhwc tt_v1</td>
</tr>
<tr>
<td>0.6</td>
<td>479257374</td>
<td>6430</td>
<td>74756.3</td>
<td>2560</td>
<td>185593</td>
<td>dCAcDecodeKernel</td>
</tr>
<tr>
<td>0.4</td>
<td>456891858</td>
<td>1268</td>
<td>354809.1</td>
<td>223551</td>
<td>198503</td>
<td>BatchedCropMirrorNormalizePermuteKernel</td>
</tr>
<tr>
<td>0.3</td>
<td>362673350</td>
<td>1268</td>
<td>356609.1</td>
<td>304000</td>
<td>362652</td>
<td>pooling_fud_kernel max nhwc</td>
</tr>
<tr>
<td>0.3</td>
<td>368487793</td>
<td>1257</td>
<td>208084.6</td>
<td>164002</td>
<td>305205</td>
<td>volta_fp16_s84cudnn_fp16_256x64_1dg8_dgrad_f2f_exp_interior_nhwc tt_v1</td>
</tr>
<tr>
<td>0.3</td>
<td>354605440</td>
<td>6430</td>
<td>55548.6</td>
<td>2176</td>
<td>140696</td>
<td>transposekernel</td>
</tr>
<tr>
<td>0.3</td>
<td>338893122</td>
<td>92</td>
<td>3683811.0</td>
<td>892335</td>
<td>26852414</td>
<td>convolve common engine_float_NHWC</td>
</tr>
<tr>
<td>0.2</td>
<td>525898329</td>
<td>6528</td>
<td>4806.2</td>
<td>923</td>
<td>48576</td>
<td>scalePackOffsetsKernel</td>
</tr>
<tr>
<td>0.2</td>
<td>277514711</td>
<td>12310</td>
<td>2254.2</td>
<td>1216</td>
<td>34304</td>
<td>computeOffsetsKernel</td>
</tr>
<tr>
<td>0.2</td>
<td>10942584</td>
<td>1261</td>
<td>154617.4</td>
<td>158848</td>
<td>24547</td>
<td>volta_fp16_s84cudnn_fp16_256x64_1dg8_relu_f2f_exp_interior_nhwc tt_v1</td>
</tr>
<tr>
<td>0.1</td>
<td>176075242</td>
<td>46</td>
<td>381809.7</td>
<td>57814</td>
<td>794291</td>
<td>ugradalgo_engine_nhwc</td>
</tr>
<tr>
<td>0.1</td>
<td>160608174</td>
<td>6528</td>
<td>2408.2</td>
<td>1508</td>
<td>30084</td>
<td>computealgo00000174kernel</td>
</tr>
<tr>
<td>0.1</td>
<td>116473612</td>
<td>1250</td>
<td>92734.9</td>
<td>91528</td>
<td>242912</td>
<td>pooling_by_avg_nhwc kernel</td>
</tr>
<tr>
<td>0.1</td>
<td>115309220</td>
<td>2159</td>
<td>53441.5</td>
<td>3000</td>
<td>123410</td>
<td>dostuffkernel</td>
</tr>
<tr>
<td>0.1</td>
<td>106643254</td>
<td>5733</td>
<td>1878.5</td>
<td>908</td>
<td>34916</td>
<td>computeOffsetsKernel</td>
</tr>
<tr>
<td>0.1</td>
<td>770893176</td>
<td>23</td>
<td>3359022.9</td>
<td>1906667</td>
<td>789856</td>
<td>wgrad _algo 4720 nhwc</td>
</tr>
<tr>
<td>0.1</td>
<td>64827363</td>
<td>1256</td>
<td>51614.1</td>
<td>49663</td>
<td>67488</td>
<td>pooling_f4_4dkernel</td>
</tr>
</tbody>
</table>

Stats/Export - CUDA kernel summary
<table>
<thead>
<tr>
<th>Time(%)</th>
<th>Time (ns)</th>
<th>Instances</th>
<th>Avg (ns)</th>
<th>Min (ns)</th>
<th>Max (ns)</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>162789746</td>
<td>70</td>
<td>2325567.8</td>
<td>248462</td>
<td>10421312</td>
<td>TensorRT::ExecutionContext::enqueue</td>
</tr>
<tr>
<td>0.4</td>
<td>16335803</td>
<td>24</td>
<td>680658.5</td>
<td>94707</td>
<td>2036983</td>
<td>TensorRT::predictions</td>
</tr>
<tr>
<td>0.3</td>
<td>10447691</td>
<td>70</td>
<td>149252.7</td>
<td>16094</td>
<td>3604518</td>
<td>TensorRT::conv1 + activation_1/Relu</td>
</tr>
<tr>
<td>0.2</td>
<td>10169055</td>
<td>70</td>
<td>145272.2</td>
<td>25936</td>
<td>1169143</td>
<td>TensorRT::conv1 + activation_1/Relu input reformatter 0</td>
</tr>
<tr>
<td>0.2</td>
<td>8698016</td>
<td>24</td>
<td>362414.7</td>
<td>9581</td>
<td>2970733</td>
<td>TensorRT::block_3b_conv_2</td>
</tr>
<tr>
<td>0.2</td>
<td>8625565</td>
<td>24</td>
<td>359398.5</td>
<td>28257</td>
<td>1242590</td>
<td>TensorRT::average_pooling2d_1</td>
</tr>
<tr>
<td>0.2</td>
<td>7733550</td>
<td>46</td>
<td>168128.7</td>
<td>30474</td>
<td>3698224</td>
<td>TensorRT::conv2d_cov/Sigmoid</td>
</tr>
<tr>
<td>0.1</td>
<td>6672310</td>
<td>70</td>
<td>95318.7</td>
<td>11271</td>
<td>1151050</td>
<td>TensorRT::block_1a_conv_1 + activation_2/Relu</td>
</tr>
<tr>
<td>0.1</td>
<td>5312256</td>
<td>70</td>
<td>75889.4</td>
<td>8316</td>
<td>2133255</td>
<td>TensorRT::block_4a_conv_2</td>
</tr>
<tr>
<td>0.1</td>
<td>5197452</td>
<td>70</td>
<td>216560.5</td>
<td>9572</td>
<td>1543570</td>
<td>TensorRT::block_3b_conv_1 + activation_12/Relu</td>
</tr>
<tr>
<td>0.1</td>
<td>4922521</td>
<td>70</td>
<td>70321.7</td>
<td>11732</td>
<td>1281643</td>
<td>TensorRT::block_1a_conv_shortcut + add_1 + activation_3/Relu</td>
</tr>
<tr>
<td>0.1</td>
<td>4151987</td>
<td>24</td>
<td>172999.5</td>
<td>10592</td>
<td>1625630</td>
<td>TensorRT::block_1b_conv_2</td>
</tr>
<tr>
<td>0.1</td>
<td>4121931</td>
<td>70</td>
<td>58884.7</td>
<td>10616</td>
<td>1307464</td>
<td>TensorRT::block_2a_conv_2</td>
</tr>
<tr>
<td>0.1</td>
<td>4108970</td>
<td>70</td>
<td>58699.6</td>
<td>9311</td>
<td>834374</td>
<td>TensorRT::block_1a_conv_2</td>
</tr>
<tr>
<td>0.1</td>
<td>4027181</td>
<td>24</td>
<td>167799.2</td>
<td>10564</td>
<td>1728250</td>
<td>TensorRT::block_1b_conv_shortcut + add_2 + activation_5/Relu</td>
</tr>
<tr>
<td>0.1</td>
<td>4018713</td>
<td>24</td>
<td>167446.4</td>
<td>9636</td>
<td>2431179</td>
<td>TensorRT::block_2b_conv_1 + activation_8/Relu</td>
</tr>
<tr>
<td>0.1</td>
<td>3799363</td>
<td>70</td>
<td>54276.6</td>
<td>8595</td>
<td>481048</td>
<td>TensorRT::block_3a_conv_2</td>
</tr>
<tr>
<td>0.1</td>
<td>3762886</td>
<td>24</td>
<td>156786.9</td>
<td>13969</td>
<td>1088096</td>
<td>TensorRT::block_4b_conv_shortcut + add_8 + activation_17/Relu</td>
</tr>
<tr>
<td>0.1</td>
<td>3376988</td>
<td>24</td>
<td>148708.2</td>
<td>12773</td>
<td>1641372</td>
<td>TensorRT::block_4b_conv_2</td>
</tr>
<tr>
<td>0.1</td>
<td>3361499</td>
<td>24</td>
<td>140062.5</td>
<td>11195</td>
<td>1248129</td>
<td>TensorRT::block_2b_conv_2</td>
</tr>
<tr>
<td>0.1</td>
<td>3227736</td>
<td>24</td>
<td>134489.0</td>
<td>15949</td>
<td>736287</td>
<td>TensorRT::predictions/Softmax</td>
</tr>
<tr>
<td>0.1</td>
<td>3023238</td>
<td>70</td>
<td>125968.2</td>
<td>11972</td>
<td>827474</td>
<td>TensorRT::block_2a_conv_shortcut + add_3 + activation_7/Relu</td>
</tr>
<tr>
<td>0.1</td>
<td>2996494</td>
<td>24</td>
<td>124937.2</td>
<td>12327</td>
<td>1217984</td>
<td>TensorRT::block_2a_conv_1 + activation_6/Relu</td>
</tr>
<tr>
<td>0.1</td>
<td>2853998</td>
<td>24</td>
<td>62843.4</td>
<td>10971</td>
<td>1983123</td>
<td>TensorRT::block_3a_conv_shortcut + add_3 + activation_7/Relu</td>
</tr>
<tr>
<td>0.1</td>
<td>2773491</td>
<td>24</td>
<td>115562.1</td>
<td>12019</td>
<td>809395</td>
<td>TensorRT::block_4a_conv_shortcut + add_7 + activation_15/Relu</td>
</tr>
<tr>
<td>0.1</td>
<td>2744711</td>
<td>24</td>
<td>114363.0</td>
<td>10474</td>
<td>892787</td>
<td>TensorRT::block_2b_conv_shortcut + add_4 + activation_9/Relu</td>
</tr>
<tr>
<td>0.1</td>
<td>2147200</td>
<td>24</td>
<td>89466.7</td>
<td>12637</td>
<td>645486</td>
<td>TensorRT::block_3a_conv_1 + activation_10/Relu</td>
</tr>
<tr>
<td>0.0</td>
<td>1914194</td>
<td>24</td>
<td>79758.1</td>
<td>10599</td>
<td>842947</td>
<td>TensorRT::block_4b_conv_1 + activation_16/Relu</td>
</tr>
<tr>
<td>0.0</td>
<td>1732011</td>
<td>24</td>
<td>72167.1</td>
<td>9659</td>
<td>488562</td>
<td>TensorRT::block_3a_conv_shortcut + add_5 + activation_11/Relu</td>
</tr>
<tr>
<td>0.0</td>
<td>1675232</td>
<td>24</td>
<td>69901.3</td>
<td>10073</td>
<td>623120</td>
<td>TensorRT::block_4a_conv_1 + activation_14/Relu</td>
</tr>
<tr>
<td>0.0</td>
<td>1525283</td>
<td>24</td>
<td>63553.5</td>
<td>10386</td>
<td>540733</td>
<td>TensorRT::block_3b_conv_shortcut + add_6 + activation_13/Relu</td>
</tr>
<tr>
<td>0.0</td>
<td>1358781</td>
<td>46</td>
<td>29538.7</td>
<td>18645</td>
<td>241099</td>
<td>TensorRT::block_3a_conv_1 + activation_6/Relu</td>
</tr>
<tr>
<td>0.0</td>
<td>1301126</td>
<td>46</td>
<td>28285.3</td>
<td>11929</td>
<td>244583</td>
<td>TensorRT::conv2d_cov</td>
</tr>
</tbody>
</table>

Stats/Export - NVTX code annotations

Note this includes TensorRT domains
### Event Table & Statistics Table

**Event Table**

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Description</th>
<th>Duration</th>
<th>TID</th>
<th>Start</th>
</tr>
</thead>
<tbody>
<tr>
<td>686</td>
<td>ID3D12GraphicsCommandList:BeginEvent</td>
<td>5.91 ms</td>
<td>3400</td>
<td>2.27097s</td>
</tr>
<tr>
<td>687</td>
<td>CPU Marker Start (5) Particle Update</td>
<td>200 ns</td>
<td>3400</td>
<td>2.27257s</td>
</tr>
<tr>
<td>689</td>
<td>ID3D12GraphicsCommandList:EndEvent</td>
<td>200 ns</td>
<td>3400</td>
<td>2.27789s</td>
</tr>
<tr>
<td>691</td>
<td>CPU Marker Start (5) RenderLightShadows</td>
<td>200 ns</td>
<td>3400</td>
<td>2.27786s</td>
</tr>
<tr>
<td>694</td>
<td>ID3D12GraphicsCommandList:EndEvent</td>
<td>100 ns</td>
<td>3400</td>
<td>2.27807s</td>
</tr>
<tr>
<td>695</td>
<td>CPU Marker Start (5) PostPass</td>
<td>200 ns</td>
<td>3400</td>
<td>2.27814s</td>
</tr>
<tr>
<td>696</td>
<td>ID3D12GraphicsCommandList:BeginEvent</td>
<td>200 ns</td>
<td>3400</td>
<td>2.27814s</td>
</tr>
<tr>
<td>697</td>
<td>CPU Marker Start (5) Opaque</td>
<td>1 ns</td>
<td>3400</td>
<td>2.27812s</td>
</tr>
<tr>
<td>698</td>
<td>ID3D12GraphicsCommandList:BeginEvent</td>
<td>1 ns</td>
<td>3400</td>
<td>2.27812s</td>
</tr>
<tr>
<td>699</td>
<td>CPU Marker Start (5) PostPass</td>
<td>200 ns</td>
<td>3400</td>
<td>2.27836s</td>
</tr>
<tr>
<td>700</td>
<td>ID3D12GraphicsCommandList:EndEvent</td>
<td>200 ns</td>
<td>3400</td>
<td>2.27836s</td>
</tr>
<tr>
<td>701</td>
<td>CPU Marker Start (5) Cutout</td>
<td>100 ns</td>
<td>3400</td>
<td>2.27842s</td>
</tr>
<tr>
<td>704</td>
<td>ID3D12GraphicsCommandList:EndEvent</td>
<td>100 ns</td>
<td>3400</td>
<td>2.27842s</td>
</tr>
<tr>
<td>705</td>
<td>CPU Marker End (5)</td>
<td>100 ns</td>
<td>3400</td>
<td>2.27843s</td>
</tr>
<tr>
<td>706</td>
<td>ID3D12GraphicsCommandList:EndEvent</td>
<td>100 ns</td>
<td>3400</td>
<td>2.27843s</td>
</tr>
<tr>
<td>707</td>
<td>CPU Marker Start (5) PostPass</td>
<td>100 ns</td>
<td>3400</td>
<td>2.27843s</td>
</tr>
<tr>
<td></td>
<td>CPU Marker Start (5) Opaque</td>
<td>1 ns</td>
<td>3400</td>
<td>2.27843s</td>
</tr>
</tbody>
</table>

**Statistics Table**

- **Threads (1)**: 1
- **DX12 API**: 1
- **Command List Markers**: 1
- **Profiler overhead**: 1
- **Frame duration (60 FPS)**: 1
- **CPU frame duration**: 1
- **GPU frame duration**: 1
- **DX12**: 1

**Graphics command list (Reset):**

- **Graphics command list (Reset Close):**

---

**Current View:** Events View, Process [16084] ModelViewer.exe (1 of 1 thread)
Cluster
Multi-node

All the problems on a single node server (DONE)
Now you need to worry about
  Working with your cluster job/task scheduler
  Multi-report views
But how do you pick which reports among thousands?
  Wall-clock time
  Networking
  Data analysis
Working with your cluster work scheduler

- `srun <slurm_args> nsys profile <nsys_args> app <app_args>`
- Make sure your report names are unique
  - `--output=friendlyName_%q{SLURM_NODEID}_%q{SLURM_PROCID}.nsys-rep`
- Avoid getting system-wide data from all ranks by moving nsys & app into a shell script
  - `IF [ $SLURM_LOCALID == 0 ] THEN`
    - Add `--nic-metrics=true`
    - Add `--gpu-metrics-device=all`
- LIMIT your recording time otherwise you will run out of memory opening reports in the GUI on your laptop
  - 30sec-1min is often good enough. Or even less… capture a few iteration.
- I have more reports than I know what to do with. Now what?
Loading Multiple Reports into one Timeline

<= 2022.1 via file menu

>= 2022.2 via open dialog, select multiple report

NOTE: Pick a subset of reports otherwise you may run out of RAM
Wall-Clock Time

Timelines need it, especially if they are going to fit together accurately.

Data is collected independently.

Relies on the system’s wall clock time to be accurate.

Otherwise there is timeline drift.

From worst to best:

- NTP = 1ms accuracy
- PTP software = 10 to 100 microseconds accuracy
- PTP hardware = can get down to ~10ns
Data Analysis

- Did not manually look at all files…. too many!
  - Maybe mix-n-match randomly with multi-report views?
- Use data analysis to hone in on interesting iterations → ranks → report
  - Cluster iteration times global total time
  - Per-rank iteration data load times
  - Per-rank iteration processing times
    - Total time to reach all-reduce
    - Forward pass
    - Backward pass
  - Per-rank iteration communication time, if not overlapped
  - DNN layer stats?
- Visually compare an average, min, & max report to try to understand how and why they differ
- You may need to add NVTX to your app to get some of this information
- GTC Talk: Scaling Transformer in PyTorch Across Multiple Nodes
  - We'll make this type of stuff easier in the future
Data Processing Imbalances

- Make every rank consistent in the iteration
- Load time
  - Did this batch take longer than the norm to load?
  - If parallel, did any rank have to wait?
- Processing time
  - Did this batch take longer than the norm to process with the DNN?
- Hurry up and wait
  - If anyone takes longer to reach the all-reduce, everyone is stuck!!!
- Some remedies
  - Fix the data
    - A lonely JPEG in a PNG world, or MP3 among WAVs?
    - Wrong resolution, sample rate, precision, …
  - Reorganize your batch order
Other Products
Nsight Compute
CUDA Kernel Profiler

Interactive CUDA API debugger

Advanced CUDA Kernel Profiling
  CUDA-C/PTX/SASS correlation
  Source correlated performance metrics
  Diff’ing for performance reports
  Programmable expert system

NVTX-range-defined kernel profiling

High performance GUI visualization and CLI data collection

NOTE: See earlier slides about relationship with Nsight Systems! Start there to get big picture!

Windows 10, Linux Ubuntu 16.04/18.04/20.4, RHEL 7.x
Nsight Deep Learning Designer

A new tool to add DL-based image processing feature to applications that have strict performance requirements.

Designer, inspector, profiler image & video processing models.

Download:
https://developer.nvidia.com/nsight-dl-designer

Quick intro (2.5min) https://www.youtube.com/watch?v=7AraPM8dhyc

Extended Intro (35min)
Nsight Visual Studio Code Edition

VSCode plugin for CUDA dev, compile, & debug


Quick Intro (4min)  
https://www.youtube.com/watch?v=gN3XeFwZ4ng

Microsoft Intro (14min)  
https://www.youtube.com/watch?v=l6PgYhiQr-I
ADDITIONAL CUDA DEVELOPER TOOLS

Command-line CUDA tools

CUDA-gdb

Unified CPU and CUDA Debugging

CUDA-C/PTX/SASS support

Compute Sanitizer - API and utility

memcheck: reports out of bounds/misaligned memory access errors

racecheck: identifies races on __shared__ memory

initcheck: usage of uninitialized global memory

synccheck: identify invalid usage of __syncthreads() and __syncwarp()
THANK YOU!


NOTE: Website version is newer than CUDA Toolkit

Let us know about your successes!