How to profile with DLProf
CONTENT
How to use DLProf

● Content
  ○ DLProf Overview
  ○ How to install DLProf
  ○ Profile with DLProf
    ■ TensorFlow 1.x
    ■ TensorFlow 2.x
    ■ PyTorch
  ○ Demo
    ■ Profile PyTorch model with DLProf
DLPROF OVERVIEW

“Are my GPUs being utilized?” “Am I using Tensor Cores?” “How can I improve performance?”

FW Support: TF, PyT, and TRT
Lib Support: DALI, NCCL

Visualize Analysis and Recommendations
DEEP LEARNING PROFILER
Deep Learning Profiler (DLProf) is a tool for profiling deep learning models

DLProf (CLI & Viewer)
- Helps data scientists understand and improve performance of their DL models by analyzing text reports or visualizing the profiling data

DLProf CLI
- Uses Nsight Systems profiler under the hood
- Aggregates and correlates CPU and GPU profiling data from a training run to DL model
- Provides accurate Tensor Core usage detection for operations and kernels
- Identifies performance issues and provides recommendations via Expert Systems

DLProf Viewer
- Uses the results from DLProf CLI and provides visualization of the data
- Currently exists as a TensorBoard plugin
GETTING STARTED

Running DLProf with TensorFlow, TRT, and PyTorch

1. TensorFlow and TRT require no additional code modification
2. Profile using DLProf CLI - prepend with `dlprof`
3. Visualize results with DLProf Viewer

1. Add few lines of code to your training script to enable `nvidia_dlprof_pytorch_nvtx` module
2. Profile using DLProf CLI - prepend with `dlprof`
3. Visualize with DLProf Viewer
SETUP DLPROF
Install DLProf with pip

- Installing Using a Python Wheel
  $ pip install nvidia-pyindex
  $ pip install nvidia-dlprof

- TensorFlow 1.x
  $ pip install nvidia-dlprof[tensorflow]

- TensorFlow 2.x
  Pip install is not supported

- PyTorch
  $ pip install nvidia-dlprof[pytorch]
SETUP DLPROF
DLProf comes pre-installed in NGC container

- **NGC**
  GPU-optimized Software Hub for AI
  DLProf comes pre-installed in NGC containers

- **TensorFlow 1.x**
  $ docker pull nvcr.io/nvidia/tensorflow:21.xx-tf1-py3

- **TensorFlow 2.x**
  $ docker pull nvcr.io/nvidia/tensorflow:21.xx-tf2-py3

- **PyTorch**
  $ docker pull nvcr.io/nvidia/pytorch:21.xx-py3
BEFORE YOU PROFILE
Do’s and Don’ts

● **Do:**
  ○ make sure your code runs without an issue
  ○ make a habit of using profiler when you make changes to your code
    ■ Observe if changes you made improve the training performance
  ○ get familiar with the optional arguments DLProf provides
    ■ Iteration range, delay, duration and etc,,

● **Don’t:**
  ○ profile for extended periods of time. It will take very long to profile
    ■ DL training is repetitive and you only need a couple minutes to profile to learn
  ○ try to open DLProf database with your TensorBoard
    ■ You need NVIDIA TB GPU plugin to visualize DLProf event files
Deep learning models are time and compute intensive

Profiling helps to find where to update code to *accelerate* training time on GPUs, *without any loss of accuracy*
1. TensorFlow and TRT require no additional code modification
2. Profile using DLProf CLI - prepend with `dlprof`
3. Visualize results with DLProf Viewer

1. Add few lines of code to your training script to enable `nvidia_dlprof_pytorch_nvtx` module
2. Profile using DLProf CLI - prepend with `dlprof`
3. Visualize with DLProf Viewer
PROFILE W/ DLPROF

TensorFlow2

- Launch container

$ docker run --gpus all -it --rm --network=host
   -v local_dir:container_dir
nvcr.io/nvidia/tensorflow:xx.xx-tfx-py3

(ex) $ docker run --gpus all -it --rm
   --network=host -v $PWD:/workspace/tf2
nvcr.io/nvidia/tensorflow:21.04-tf2-py3
Run DLProf once to find *key_node*

*key_node* is a framework-specific node in your DL model that is to define an iteration. *key_node* would show up once an iteration. Without *key_node* DLProf outputs 0 iterations.

$ dlprof --mode=tensorflow2 --reports=detail python <TF2 script>

(ex) $ dlprof --mode=tensorflow2 --reports=detail --delay=30 --duration=30 python <TF2 script>
Run DLProf once to find \textit{key_node}

\textit{key_node} is a framework-specific node in your DL model that is to define an iteration. \textit{key_node} would show up once an iteration. Without \textit{key_node} DLProf outputs 0 iterations

$\text{dlprof } \text{--mode=tensorflow2 } \text{--reports=detail}\ python\ <\text{TF2 script}>$

(ex) $\text{dlprof } \text{--mode=tensorflow2 } \text{--reports=detail } \text{--delay=30 } \text{--duration=30}\ python\ <\text{TF2 script}>$
PROFILE W/ DLPROF

TensorFlow

- Determine **key_node**
  There are two ways you can choose to determine **key_node (Op Name)**

  1. Use TensorBoard
     $ tensorboard --logdir=event_files
     You know your loss calculation happens only once an iteration
     **key_node**

  2. Use **dlprof_detailed.csv** report
Determine *key_node*
There are two ways you can choose to determine *key_node* (Op Name)

1. Use TensorBoard
   
   $ tensorboard --logdir=event_files
   
   You know your loss calculation happens only once an iteration *key_node*

2. Use *dlprof_detailed.csv* report

   **key_node**
Profile with selected *key_node*

$ dlprof --mode=tensorflow2
--key_node=<key_node Op Name>
--reports=detail --delay=10 --duration=30
python <TF2 script>

(ex) $ dlprof --mode=tensorflow2
--key_node=sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEntropyWithLogits --iter_start=100
--iter_stop=200 python <TF2 script>
PROFILE W/ DLPROF
TensorFlow2

- Launch TensorBoard
  (ex) $ tensorboard --port 8000
  --logdir=event_files

  Open a browser at:
  (ex) http://dgx1v-loki-23.nvidia.com/:8000/
GETTING STARTED
Running DLProf with TensorFlow, TRT, and PyTorch

1. TensorFlow and TRT require no additional code modification
2. Profile using DLProf CLI - prepend with `dlprof`
3. Visualize results with DLProf Viewer

1. Add few lines of code to your training script to enable `nvidia_dlprof_pytorch_nvtx` module
2. Profile using DLProf CLI - prepend with `dlprof`
3. Visualize with DLProf Viewer
PROFILE W/ DLPROF

PyTorch

- Launch container

```
$ docker run --gpus all -it --rm --network=host -v local_dir:container_dir nvcr.io/nvidia/pytorch:xx.xx-py3
```

(ex) $ docker run --gpus "device=2" -it --rm --network=host -v $PWD:/workspace/pyt nvcr.io/nvidia/pytorch:21.04-py3

```
jamess@dgx1v-loki-01:/mnt/nvdl/usr/jamess/profiler/dlprof/pyt$ docker run --gpus 1 -it --rm --network=host -v $PWD:/workspace/pyt nvcr.io/nvidia/pytorch:21.04-py3
```

NVIDIA Release 21.04 (build 22382700)
PyTorch Version 1.9.0a0+2ecb2c7

Container image Copyright (c) 2021, NVIDIA CORPORATION. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License. By pulling and using the container, you accept the terms and conditions of this license:

NOTE: Legacy NVIDIA Driver detected. Compatibility mode ENABLED.
PROFILE W/ DLPROF
PyTorch

- Code changes

NVIDIA Tools Extension (NVTX) annotates events, code ranges and resources in the code

```sh
$ import nvidia_dlprof_pytorch_nvtx as nvtx
$ nvtx.init(enable_function_stack=True)
$ with torch.autograd.profiler.emit_nvtx():
    <training loop>
```
PROFILE W/ DLPROF

PyTorch

- Run DLProf with python cmd

$ dlprof --mode=pytorch python <PyT script>

(ex) $ dlprof --mode pytorch
--reports=summary --iter_start=200
--iter_stop=400 python <PyT script>

```
root@dpxiv-lok1-01:/workspace/yt# dlprof --mode=pytorch --reports=summary --iter_start 200 --iter_stop=400 --force=true python profile_mnist.py
```
PROFILE W/ DLPROF

PyTorch

- Launch TensorBoard

$ tensorboard --port 8000 --logdir=event_files

Open a browser at:

(ex) http://dgx1v-loki-23.nvidia.com/:8000/
DLPROF VIEWER
Dashboard: Top Row

- **GPU Utilization Chart:**
  - Shows the percentage of the wall clock time that the GPU is active. For multi-gpu, it is an average utilization across all GPUs

- **Op GPU Time Chart:**
  - Splits all operations into 3 categories: Operations that used tensor cores, operations that were eligible to use tensor cores but didn't, and operations that were ineligible to use tensor cores

- **Kernel GPU Time Chart:**
  - Breaks down all kernel time into 3 categories: Kernels that used tensor cores, memory kernels, and all other kernels

- **Tensor Core Kernel Efficiency Chart:**
  - Gives a single number that measures what percentage of GPU time inside of TC-eligible ops are using tensor cores.
**Performance Summary:**
- A straightforward panel that shows all of the key metrics from the run in one place.

**Iteration Summary:**
- A bar chart that shows the amount of time each iteration took during the run. The colored bars are the ones that were used to generate all of the statistics, while the gray bars are iterations that were outside the selected range. Each colored bar shows the breakdown of iteration time into GPU using TC, GPU not using TC, and all other non-GPU time.

**Top 10 GPU Ops:**
- Shows the top 10 operations in the run sorted by the amount of GPU time they took. This is a great starting point for trying to find potential for improvements.
**System Config:**
- Shows the system configuration for the run.

**Expert Systems Recommendations:**
- Shows any potential problems that DLProf found and recommendations for how to fix them.

**Guidance Panel:**
- Provides some helpful links to learn more about GPU utilization and performance improvements.

### System Config

- **GPU Count:** 4
- **GPU Name(s):**
  - Tesla V100-SGIX-32GB
  - Tesla V100-SGIX-32GB
  - Tesla V100-SGIX-32GB
- **GPU Model:**
  - Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20 GHz
- **GPU Driver Version:** 450.51.00
- **Framework:** TensorFlow 2.4.0
- **CUDA Version:** 11.2
- **cuDNN Version:** 8.1.1
- **NVIDIA Version:** 457.41.06
- **DLProf Version:** 1.0.0
- **TensorBoard Version:** 1.15.0+nv

### Recommendations

<table>
<thead>
<tr>
<th>Problem</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLA is not enabled: No XLA detected</td>
<td>Try enabling XLA. See <a href="https://www.tensorflow.org/stable/enable_xla_for_tensorflow_v_models_for_information_on_how_to_enable_xla">https://www.tensorflow.org/stable/enable_xla_for_tensorflow_v_models_for_information_on_how_to_enable_xla</a>.</td>
</tr>
</tbody>
</table>

### Guidance

- Understanding GPU utilization and timing details of the operations in the first step in profiling your model.
  - To learn more about Tensor cores and Mixed Precision training, visit this site: [https://developer.nvidia.com/tensor_cores](https://developer.nvidia.com/tensor_cores).
  - Note that if there are multiple kernels being observed on single op node, these are likely performing data transposes to prepare the data for efficient use by tensor cores. Such transposes themselves would not use tensor cores.
Detailed views can be found by opening the navigation panel in the upper left corner of the viewer.

All data tables in the detailed views can be sorted, filtered, and exported.
The **Op Type Summary view** gives a summary of each operation type in the run.
The Ops and Kernels Summaries allows you to drill down into the kernels used by a given operation. When you select an op in the first table, all of the kernels across all instances of that operation will appear in the bottom table.
The Kernels By Iterations view allows you to drill down into the kernel instances of an iteration. When you select an iteration in the first table, all of the kernel instances will appear in the bottom table.
- The **Kernels By Op** view allows you to drill down into the kernel instances of a single op instance. First pick an iteration in the top table, and then an op instance in the second table. All kernel instances for that op will appear in the bottom table.
The **Iterations View** shows a zoomed in iteration bar chart, and gives the user the ability to reaggregate the profiled data using a different iteration range and/or key node values.
• The **GPUs View** shows a list of all visible GPUs and their utilizations during the run. This view will not exist if the run only used a single GPU.
DEMO
Overview

- Recordings in the followings slides demonstrate the steps to profile PyTorch model with DLProf

- Notes
  - Model used in the demo is available at NVIDIA Deep Learning Examples
  - Container used in the demo is NGC PyTorch 21.03 container
  - Profiling results are not included in the demo
DEMO

Baseline

● Resnet50 w/ Imagenet

● PyTorch 21.03 from NGC

● NUM_GPUS=8;
  BATCH_SIZE=256;
  EPOCHS=1

● --duration 240
  --iter_start 10
  --iter_stop 20
DEMO
Automatic Mixed Precision

- Resnet50 w/ Imagenet
- PyTorch 21.03 from NGC
- NUM_GPUS=8;
  BATCH_SIZE=256;
  EPOCHS=1
  AMP *) passing --amp to your PyTorch code doesn’t activate AMP automatically. Please refer to the included model code and next slide to learn how to activate AMP in yours
- --duration 240
- --iter_start 10
- --iter_stop 20
Insert ~two lines of code~ to use Automatic Mixed-Precision and get up to a 3X speedup

Support for TensorFlow, PyTorch and MXNet

Automatic mixed precision applies two techniques to maximize performance while preserving accuracy:
1) Optimizing per operation precision by casting to FP16 or FP32
2) Dynamic loss scaling to prevent gradient underflow
import torch
# Creates once at the beginning of training
scaler = torch.cuda.amp.GradScaler()

for data, label in data_iter:
    optimizer.zero_grad()
    # Casts operations to mixed precision
    with torch.cuda.amp.autocast():
        loss = model(data)
        # Scales the loss, and calls backward()
        # to create scaled gradients
        scaler.scale(loss).backward()
        # Unscales gradients and calls
        # or skips optimizer.step()
        scaler.step(optimizer)
        # Updates the scale for next iteration
        scaler.update()

References

- AMP PyTorch doc
- AMP PyTorch examples
- AMP Developer Blog
**DEMO**

**Bigger batch size**

- Resnet50 w/ Imagenet
- PyTorch 21.03 from NGC
- NUM_GPUS=8;
  BATCH_SIZE=384;
  EPOCHS=1
  AMP
- --duration 240
  --iter_start 10
  --iter_stop 20
Q&A