NVIDIA Deep Learning Profiler

User Guide
# Table of Contents

Chapter 1. Deep Learning Profiler...................................................................................... 1  
   1.1. Overview.................................................................................................................. 1  
   1.2. What’s New in 0.12.0............................................................................................... 1  
   1.3. Features.................................................................................................................. 2  

Chapter 2. Quick Start..........................................................................................................3  
   2.1. Prerequisites.............................................................................................................. 3  
   2.2. Using the NGC Docker Container............................................................................ 3  
   2.3. Running the Deep Learning Profiler....................................................................... 4  
   2.4. Analyzing Results................................................................................................... 4  

Chapter 3. Multiple Deep Learning Framework Support....................................................5  
   3.1. Simple Mode............................................................................................................. 5  
   3.2. TensorFlow 1.X......................................................................................................... 6  
   3.3. TensorFlow 2.x......................................................................................................... 6  
   3.4. PyTorch................................................................................................................... 6  

Chapter 4. Profiling.............................................................................................................. 7  
   4.1. Profiling from the NGC TensorFlow Docker Container........................................... 7  
      4.1.1. Pulling NGC Framework Containers................................................................. 7  
      4.1.2. Launching the NGC Container......................................................................... 8  
   4.2. Running DLProf....................................................................................................... 8  
   4.3. Profiling with Nsight Systems.................................................................................. 9  
   4.4. Profiling PyTorch with PyProf................................................................................ 9  
   4.5. Profiling with Delay and Duration.........................................................................10  
   4.6. Running DLProf without Profiling.........................................................................10  
   4.7. TensorFlow Graphdef Files....................................................................................10  
      4.7.1. Automatically Generating a Graphdef File.....................................................10  
      4.7.2. Supplying a Graphdef....................................................................................11  
      4.7.3. Creating a GraphDef File..............................................................................11  

Chapter 5. Tensor Core usage........................................................................................... 12  
   5.1. Mixed Precision Training...................................................................................... 12  
   5.2. Determining Tensor Core Eligibility..................................................................... 12  

Chapter 6. TensorBoard Plugin........................................................................................ 13  
   6.1. Generating TensorBoard Event Files..................................................................... 13  
   6.2. Starting TensorBoard............................................................................................ 13  

Chapter 7. Iteration Detection.......................................................................................... 14
7.1. Specifying the Key Node.........................................................................................................14
7.2. Limiting Aggregation to an Iteration Range........................................................................15

Chapter 8. Correlating Time with NVTX Markers.....................................................................16
8.1. NVTX Markers in TensorFlow.............................................................................................16
8.2. Mapping GPU Time...............................................................................................................16
8.3. Aggregating Time..................................................................................................................17

Chapter 9. Custom NVTX Ranges.............................................................................................18

Chapter 10. Report Generation.................................................................................................19
10.1. Specifying Reports and Formats.......................................................................................19
10.1.1. Selecting Reports...........................................................................................................19
10.1.2. Selecting Domains........................................................................................................19
10.1.3. Selecting File Formats..................................................................................................20
10.1.4. Report Names................................................................................................................20
10.1.5. Output Path..................................................................................................................21
10.2. Summary Report................................................................................................................21
10.3. Detailed Report...................................................................................................................23
10.4. Iteration Report..................................................................................................................24
10.5. Kernel Report.....................................................................................................................25
10.6. Tensor Core Report..........................................................................................................26
10.7. Node Op Report..................................................................................................................27
10.10. Expert Systems..................................................................................................................30
10.11. XLA Support.....................................................................................................................30

Chapter 11. User Goals.............................................................................................................31
11.1. How do I profile a deep learning network?.........................................................................31
11.2. How can I improve my network if I’m not using Tensor Cores?..........................................31
11.3. How do I find a good Key Op?............................................................................................32
11.4. How do I choose a good iteration range to analyze?...........................................................32
11.5. How can I find out profile data for a TensorFlow-TensorRT inference graph?....................33

Chapter 12. Tutorials.................................................................................................................34
12.1. Resnet50.............................................................................................................................34
12.1.1. Preparing the Example..................................................................................................34
12.1.2. Profiling Resnet50..........................................................................................................34
12.1.3. Viewing Results in TensorBoard....................................................................................35
12.1.4. Profiling with Simple Mode...........................................................................................35
12.2. MobileNet........................................................................................................................35
Chapter 1. Deep Learning Profiler

1.1. Overview

Deep Learning Profiler is a tool for profiling deep learning models to help data scientists understand and improve performance of their models visually via Tensorboard or by analyzing text reports. We will refer to Deep Learning Profiler simply as DLProf for the remainder of this guide.

1.2. What’s New in 0.12.0

- PyTorch support for DLProf build released in the PyTorch NGC container.
  - Depends on PyProf and Nsight Systems which are provided in the container.
  - Generates all reports and a GPU Event file that can be viewed with the NVIDIA DLProf Tensorboard Plugin is included for Tensorboard visualization.
- New command line switch options.
  - Align CLI switch options and style with Nsight Systems CLI.
  - Improved help messages from dlprof --help.
  - Known issue: boolean options now require specifying true or false, e.g. --force=true.
- Support for multiple Frameworks and NGC containers.
  - Each supported NGC container has a customized DLProf that is optimized for the Deep Learning framework in the container.
  - Supported frameworks:
    - Tensorflow 1.x
    - Tensorflow 2.x - simple mode only
    - PyTorch
- Recommendations for shape and data type are improved in Tensorboard and on command line.
1.3. **Features**

This release includes these commands and features:

- **Tensor Core Usage and Eligibility Detection**: DLProf can determine if an operation has the potential to use Tensor Cores and whether or not Tensor Core enabled kernels are being executed for those operations.

- **Multiple Deep Learning Framework Support**: Individual frameworks supported by DLProf can be profiled by selecting the appropriate profile mode.

- **Custom TensorBoard Plugin**: DLProf can automatically generate TensorBoard event files. These event files are used with [NVIDIA’s GPU Tensorboard plugin](https://developer.nvidia.com/) to visualize and analyze the profile results in TensorBoard.

- **Iteration Detection**: Iterations can be detected from specifying a *key node*. Reports can be aggregated based on iterations, allowing users to further drill down performance bottlenecks.

- **Time Correlation with NVTX Markers**: DLProf uses NVTX markers inserted into the profile data to correlate CPU and GPU time with model operations.

- **Report Generation**: A number of reports can be generated that aggregate data based on operation, iteration, layer, or kernel. Both JSON and CSV formats are supported for most reports.

- **Expert Systems**: A feature that analyzes the profiling data, identifying common improvement areas and performance bottlenecks, and provides suggestions on how to address the issues to improve the overall performance.

- **XLA Support**: DLProf fully supports analyzing XLA compiled TensorFlow models. Reports and TensorBoard event files will show both the XLA generated operations and the mapped, pre-compiled operations.

- **Support Custom NVTX Markers and Domains**: DLProf will support custom NVTX markers and domains specified with the [NVTX Plugin](https://developer.nvidia.com/).

- **Profile with Delay and Duration**: DLProf can delay the start of profile and stop the profile after a set duration.
Chapter 2. Quick Start

DLProf is still in beta and full functionality is available in the NGC TensorFlow container. DLProf command line options and file formats are subject to change in future releases.

2.1. Prerequisites

These steps are required to use pre-built NGC containers:

- Ensure you have access and are logged into NGC. For step-by-step instructions, see the NGC Getting Started Guide.
- Install Docker and nvidia-docker. For DGX users, see Preparing to use NVIDIA Containers. For users other than DGX, see nvidia-docker installation documentation.

2.2. Using the NGC Docker Container

Make sure you log into NGC as described in Prerequisites before attempting the steps in this section. Use docker pull to get the TensorFlow container from NGC:

```bash
$ docker pull [<container name>]
```

Where `[<container name>]` is the location for the NGC framework container. Supported containers include:

- Tensorflow 1.x: `nvcr.io/nvidia/tensorflow:<xx.yy>-tf1-py3`
- Tensorflow 2: `nvcr.io/nvidia/tensorflow:<xx.yy>-py3`
- PyTorch: `nvcr.io/nvidia/pytorch:<xx.yy>-py3`

Replace `<xx.yy>` is the version of the TensorFlow container that you want to pull.

Assuming the training data for the model is available in `full/path/to/training/data`, you can launch the container with the following command:

```bash
$ docker run --rm --gpus=1 --shm-size=1g --ulimit memlock=-1 \
--ulimit stack=67108864 -it -p6006:6006 -v full/path/to/training/data:/data \
[<container name>]
```
2.3. Running the Deep Learning Profiler

Using this command is the fastest way to profile your model training:

$ dlprof python <train script>

Where `<train script>` is the full command line you would normally use to train your model. DLProf will automatically create the correct Nsight System command line needed to profile your training session and create the necessary event files needed to view the results in TensorBoard. The following collateral will be created:

- `nsys_profile.qdrep`: The QDREP file is generated by Nsight Systems and can be opened in the Nsight Systems GUI to view the timeline of the profile.
- `nsys_profile.sqlite`: A SQLite database of the profile data that is used by DLprof.
- `event_files`: A folder containing the automatically generated TensorBoard event files.

2.4. Analyzing Results

To analyze the results in TensorBoard, run the following command inside the same TensorFlow container:

$ tensorboard --logdir ./event_files

The TensorBoard server will launch within the container. To view TensorBoard, enter `http://<IP Address>:6006` in a browser.

See the NVIDIA DLProf Plugin for TensorBoard User Guide for more information.
DLProf is designed to be agnostic to the underlying Deep Learning framework when analyzing and presenting profile results. However, profiling is very specific to the individual framework. It is not always possible to automatically detect which framework a training or inferencing script is using. In DLProf, the correct framework can be selected by setting the execution mode with the `--mode` command line option. The available modes of operation are listed below.

A special build of DLProf is created for each supported framework container on the NVIDIA GPU Cloud [NGC]. This build will set the appropriate default mode to the underlying framework in the container, so it is not necessary to set the `--mode` flag within the container. Additional options may be limited depending on which features are supported in each framework. See Command Line Options for a full list of available options per framework.

### 3.1. Simple Mode

DLProf provides a **Simple Mode** that is completely framework agnostic and works with any framework, both supported and unsupported. Simple mode is activated with the DLProf command line option `--mode=simple`.

Unlike framework specific modes, simple mode does not require NVTX annotation markers to be embedded at the framework level. These markers are already present in the NGC TensorFlow1 but not in TensorFlow2 or PyTorch. Simple mode will only provide the **Total Wall Clock Time**, **Total GPU Time**, and **Total Tensor Core Kernel Time**. Additionally, simple mode can generate a Kernel Report showing the time aggregation for each executed kernel.

Without the aid of NVTX markers, all additional features and reports are not available in simple mode. This includes creating the GPU Event Files needed for the TensorBoard viewer and Expert Systems output. However, simple mode can provide an overview of GPU utilization and Tensor Core Kernel utilization.

Simple mode still requires Nsight Systems and you can use `--delay` and `--duration` options to limit the profile window for the training script.
3.2. TensorFlow 1.X

DLProf officially supports TensorFlow 1.x. To profile a TensorFlow 1.x model, use the command line option --mode=tensorflow1. This mode is set by default in the DLProf released in the NGC TensorFlow 1.x container and does not need to be explicitly called.

All reports and features are fully supported in TensorFlow 1.x. See the Command Line Options section for full list of available options.

3.3. TensorFlow 2.x

DLProf does not yet officially support TensorFlow 2.x. However, a simple mode only version of DLProf is provided in the NGC TensorFlow 2.x container. The only mode option available is --mode=simple and it is set by default in the NGC TensorFlow 2.x container.

3.4. PyTorch

DLProf officially supports PyTorch. To profile a PyTorch model, use the command line option --mode=pytorch. This mode is set by default in the DLProf released in the NGC PyTorch container and does not need to be explicitly called.

DLProf uses both PyProf and Nsight Systems to profile PyTorch models and are available in the NGC PyTorch container. All reports and features are fully supported in PyTorch. See the Command Line Options section for full list of available options.
Chapter 4. Profiling

The NVIDIA Deep Learning Profiler (DLProf) is still in beta. Note that due to the beta status, backwards compatibility is not guaranteed. Command line arguments, file formats, and event file protobufs may change between releases. For the best experience, make sure to use the compatible versions of the GPU Driver, CUDA, TensorFlow, TensorBoard, and Nsight Systems specified in the release notes.

DLProf is a wrapper tool around Nsight Systems that correlates profile timing data and kernel information to a Machine Learning model. The correlated data is presented to a Data Scientist in a format that can be easily digested and understood by the Data Scientist. The results highlight GPU utilization of model and DL/ML operations. The tools provide different reports to aid in identifying bottlenecks and Tensor Core usage.

4.1. Profiling from the NGC TensorFlow Docker Container

DLProf is provided in the TensorFlow 1.x container on the NVIDIA GPU Cloud (NGC). The version of TensorFlow inside the container has been modified by NVIDIA to automatically insert NVTX range markers around the TensorFlow executor. The NVTX markers are required for DLProf in order to correlate GPU time with the TensorFlow model.

4.1.1. Pulling NGC Framework Containers

Before you can pull a container from the NGC container registry, you must have Docker and nvidia-docker installed. For DGX users, this is explained in Preparing to use NVIDIA Containers Getting Started Guide. For users other than DGX, follow the nvidia-docker installation documentation to install the most recent version of CUDA, Docker, and nvidia-docker.

After performing the above setup, you can use the following commands to pull the desired framework container.

TensorFlow 1.x

docker pull nvcr.io/nvidia/tensorflow:20.06-tf1-py3

TensorFlow 2.x

docker pull nvcr.io/nvidia/tensorflow:20.06-tf2-py3
PyTorch

docker pull nvcr.io/nvidia/pytorch:20.06-py3

Replace the current container version with the version of the container and profiler release that you want to pull.

4.1.2. Launching the NGC Container

Assuming the training data for the model is available in /full/path/to/training/data, you can launch the container with the following command:

```bash
$ docker run --rm --gpus=1 --shm-size=1g --ulimit memlock=-1 \
--ulimit stack=67108864 -it -p6006:6006 -v/full/path/to/training/data:/data \\
[<container name>]
```

Replace `[<container name>]` with the name of the container used in the previous docker pull instruction.

The `--gpus` option is required to use nvidia-docker and specifies the number of GPUs to provide to the container. At this time, DLProf only supports a single gpu, so the option should remain `--gpus=1`.

The `--gpus device=<number>` allows users to specify which GPU device to use during profiling. This is helpful when a host has more than one GPU device.

**Note:** DLProf uses only one GPU device when profiling. This option specifies which GPU device to use.

The nvidia-docker `-v` option maps `/full/path/to/training/data` on the host into the container at `/data`. You may also map additional host directories into the container with separate `-v` options.

The `-p` flag exposes the container port for the TensorBoard server (port 6006).

The `--shm-size` and `--ulimit` flags are recommended to improve the server’s performance. For `--shm-size` the minimum recommended size is 1g but smaller or larger sizes may be used depending on the number and size of models being served.

4.2. Running DLProf

One of the main goals for DLProf is to automate and simplify the profiling experience. In its simplest form, a user would just need to prepend the training script with `dlprof`.

```bash
dlprof [training_script.py]
```

DLProf automatically creates the correct Nsight System command line needed to profile your training session and create the necessary event files needed to view the results in TensorBoard. The following collateral will be created:

- `nsys_profile.qdrep`: The QDREP file is generated by Nsight Systems and can be opened in the Nsight Systems GUI to view the timeline of the profile.
- `nsys_profile.sqlite`: A SQLite database of the profile data that is used by DLProf.
Profiling

- event_files: A folder containing the automatically generated TensorBoard event files.

All **DLProf specific options** must be passed before the training script in the following format:

dlprof <args> [training_script.py]

### 4.3. Profiling with Nsight Systems

Nsight Systems passively logs CUDA API calls. The result is the ability to profile the entire model network, both GPU and CPU, in near real time. DLProf then extracts the timing and NVTX range information for every executed kernel. Getting timing information for the operations that ran during model training can be an important debugging tool to determine where optimization is needed.

DLProf determines the Tensor Core utilization from the name of the kernel. This method can accurately identify cuDNN kernels that use Tensor Cores, but will not identify custom kernels or kernels outside of cuDNN that use Tensor Cores.

DLProf enables you to customize the **Nsight Systems command line**. By default, DLProf calls Nsight Systems with the following command line arguments:

```
nsys profile -t cuda,nvtx -s none --show-output=true <training_script.py>
```

You can customize the NSight System arguments using this DLProf option:

```
--nsys_opts [option list]--
```

For example,

```
dlprof --nsys_opts -t cpu,cuda,nvtx --show-output=false -- <training_script.py>
```

creates and executes the following Nsight Systems command:

```
nsys profile -t cpu,cuda,nvtx --show-output=false <training_script.py>
```

DLProf can also change the base filename for Nsight Systems output files from `nsys_profile` with

```
--nsys_base_output_filename <basename>
```

This can be useful when profiling multiple configurations and you require keeping the profile data from each run.

### 4.4. Profiling PyTorch with PyProf

When profiling PyTorch models, DLProf uses **PyProf** to insert the correct NVTX markers. PyProf must first be enabled in the PyTorch Python script before it can work correctly.

To enable the PyProf, you must add the following lines to your PyTorch network:

```
import torch.cuda.profiler as profiler
import pyprof
pyprof.init()
```

The PyTorch model can then be **profiled normally with DLProf**.
4.5. Profiling with Delay and Duration

DLProf can delay the start of the profile with this command line option:

```
--delay <seconds>
```

This adds the correct command line to Nsight Systems that will delay the start of the profile by the specified number of seconds. Note that the first iteration starts with the first key node found after the delay, and will not include any iterations before the delayed time.

DLProf can stop the profile and the execution of the model after a specified number of seconds with the following command line option:

```
--duration <seconds>
```

Both delay and duration can be used together to limit the profiling to a specified number of seconds in the middle of a model run.

4.6. Running DLProf without Profiling

It is possible to run DLProf without calling Nsight Systems to profile the model again. This is useful to create a new report, specify a different key node, or aggregate data over different iteration ranges. In each of these cases, it is better to reuse profile data that has already been collected.

An SQLite database created by an initial Nsight Systems profile. The format for the DLProf command line becomes:

```
dlprof --nsys_database=[nsys_profile.sqlite] [<args>]
```

where `[nsys_profile.sqlite]` is the SQLITE file generated by Nsight Systems. All other DLProf options are valid and optional.

4.7. TensorFlow Graphdef Files

It is possible to specify and/or create GraphDef files when profiling TensorFlow 1.x models. When specified, DLProf will create a custom Graph Event File that can be loaded into NVIDIA customized TensorBoard available in all supported framework NGC Containers. The following sections describe how to use the TensorFlow 1.x GraphDef feature.

4.7.1. Automatically Generating a Graphdef File

The model is the basis for correlating profile results and determining CPU/GPU time as well as eligibility of using Tensor Cores.

**Note:** Only the TensorFlow GraphDef model is officially supported and tested by DLProf in this release.
When using the option below, DLProf automatically attempts to generate a graphdef file from Tensorflow:

```
--graphdef=auto
```

This creates the `graphdef_dump` directory in the working directory and will generate a GraphDef for each TensorFlow session. DLProf will combine all GraphDefs together for viewing in TensorBoard.

**Note:** This option should be used when profiling.

### 4.7.2. Supplying a Graphdef

Optionally, a pre-generated GraphDef file, or directory of files, can be specified using:

```
--graphdef=/path/to/file.pb
--graphdef=/path/to/file.pbtxt
--graphdef=/path/to/directory
```

**Note:** An auto-generated `graph_dump` directory (using `--graphdef=auto`) can also be reused on a later profiling (using `--graphdef=/path/to/graph_dump`).

In this case, the TF environment variables from the auto generated step are not used.

### 4.7.3. Creating a GraphDef File

If the TensorFlow script doesn't contain an option to create a graphdef, the following code can be inserted into your TensorFlow python script after the TensorFlow session has been created:

```python
graph_def = session.graph.as_graph_def()
with open('graphdef.pb', 'wb') as f:
    f.write(graph_def.SerializeToString())
with open('graphdef.pbtxt', 'w') as f:
    f.write(str(graph_def))
```

Now run the training script for an iteration to create the graphdef.pb file.
NVIDIA’s Tensor Cores is a revolutionary technology that accelerates AI performance by enabling efficient mixed-precision implementation. It accelerates large matrix multiplication and accumulation operations in a single operation.

5.1. Mixed Precision Training

Mixed precision methods combine the use of different numerical formats in one computational workload. Mixed precision training offers significant computational speedup by performing operations in half-precision format, while storing minimal information in single-precision to retain as much information as possible in critical parts of the network. Since the introduction of Tensor Cores in the Volta, Turing, and Ampere architecture, significant training speedups are experienced by switching to mixed precision -- up to 3x overall speedup on the most arithmetically intense model architectures.

5.2. Determining Tensor Core Eligibility

DLProf provides feedback on Tensor Core utilization in the TensorFlow model. Tensor Cores are mixed precision floating point operations available for Volta GPUs (Titan V) and beyond. The cuDNN and cuBLAS libraries contain several Tensor Core enabled GPU kernels for most Convolution and GEMM operations.

DLProf determines the Tensor Core eligibility of a TensorFlow graph node based on the operation. Tensor Core usage is determined from executed GPU kernels found in the Nsight Systems profile results.
Chapter 6.  TensorBoard Plugin

The NVIDIA TensorBoard GPU plugin for DLProf makes it easy to find and visualize the performance of your models by showing Top 10 operations that took the most time, eligibility of Tensor Core operations and Tensor Core usage, interactive iteration reports. For information on how to use the TensorBoard Plugin, please the NVIDIA DLProf Plugin for TensorBoard User Guide.

6.1. Generating TensorBoard Event Files

By default, DLProf generates two TensorBoard event files, tfevents, <xxx>.<yyy> and tfdlprof.<xxx>.<yyy>. The files are added to the event_files/ directory in the current working directory. If the directory does not exist, one will be created. The event files are time stamped, so that TensorBoard always opens the newest file.

To specify a different event files directory, use the argument:

```
--out_tb_dir=<path/to/new/event_files>
```

To prevent DLProf from creating the events, use the argument:

```
--suppress_tb_files
```

6.2. Starting TensorBoard

TensorBoard and the GPU Plugin are installed in the TensorFlow 1.x container on the NVIDIA GPU Cloud [NGC]. The container must be run with the -p6006:6006 option to open port 6006 for the TensorBoard server.

TensorBoard is launched directly from the container:

```
tensorboard --logdir <event_files>
```

Where <event_files> is the path to the event files directory. Once running, TensorBoard can be viewed in a browser with the URL:

```
http://<machine IP Address>:6006
```
Chapter 7. Iteration Detection

An iteration interval is one pass through both forward and backward propagation, for a single batch. DLProf attempts to automatically determine iteration intervals using the NVTX start times of a key node. A key node is an op node that is executed only once, each iteration, preferably the very first operation of each iteration. Typically this would be GlobalStep, or something similar.

Once the iteration intervals are found, every model operation and kernel call instance are sorted into the intervals. Metrics can be aggregated per interval for specific reports and is an extremely useful aid in locating bottlenecks.

Iteration intervals always start from time 0 and end with the final stopping timestamp in the profile. For N instances of Key Node, the intervals would be:

\[ [0, \text{Node[1].start-1}], \ [\text{Node[1].start}, \text{Node[2].start-1}], \ldots, \ [\text{Node[N].start}, \text{last}] \]

Resulting in N+1 intervals.

**Note:** If no iterations are found, then the entire profiled model is treated as a single iteration. This will be reflected in the Iteration Report and the Summary Report will show 0 iterations found.

7.1. Specifying the Key Node

By default, DLProf will look for a framework-specific node as the key node. However, not all models will use this default name. If DLProf outputs 0 iterations, then the current key node was not found in the model. When the default key node is not found, you need to identify and select a new key node with the following command argument:

```
--key_node=<key_node>
```

where <key_node> is the name of the new key node as listed in the Node Op report or Detailed report. For the PyTorch framework, you can pass in `--key_node=""` to try to have it auto-detect the backward function as the key node.
7.2. Limiting Aggregation to an Iteration Range

DLProf can specify an interval range to use when aggregating the profile data for all of the reports. This is useful to ignore profile data captured during the warm up and tear down phases. To limit the aggregation range, use the following command line arguments:

```
--iter_start <start_iter> --iter_stop <stop_iter>
```

The aggregation range is inclusive. All timing data aggregates from iteration `<start_iter>` to `<stop_iter>`, including both `<start_iter>` and `<stop_iter>`.
Chapter 8. Correlating Time with NVTX Markers

The core purpose of DLProf is to correlate NVTX (NVIDIA Tools Extension) annotated results from Nsight Systems profiles with a high-level model description. From here, any number of reports can be created to deliver the profile results in a format familiar to the Data Scientist.

8.1. NVTX Markers in TensorFlow

TensorFlow in the NGC TensorFlow container has been modified to automatically insert NVTX Start/Stop range markers into the execution of the model. The NVTX markers are wrapped around the execution nodes of the model and named exactly the same as the node. Nsight Systems will associate all GPU kernels to the NVTX range that was active when the kernel was scheduled.

**Note:** The modification to TensorFlow to automatically insert NVTX ranges has not been upstreamed to TensorFlow and is only available in the version of TensorFlow provided in the NGC Tensorflow container.

Since the NVTX name has a 1:1 mapping to a node in the TensorFlow graph, DLProf can correlate kernels to a particular node. DLProf will also associate any metrics gathered for a kernel from Nsight Systems, such as Tensor Core usage, start time, and stop time.

8.2. Mapping GPU Time

The NVTX range is the time stamp for the start and end of a TensorFlow operation on a CPU thread. This range then becomes synonymous with CPU time for that instance of the TensorFlow operations. To determine the GPU time, Nsight Systems correlates all of the CUDA API calls to specific NVTX range in which they were called.

CUDA API calls on the CPU thread schedule a corresponding CUDA kernel onto the GPU. A CUDA kernel is a small, parallel function executed on the GPU and makes GPGPU computing possible. Nsight Systems tracks which CUDA API call started each kernel and can correlate the actual execution of the kernel back to the CPU API call and NVTX range.
Nsight Systems has a notion of *Mapped GPU Time* for each NVTX range. The mapped GPU time starts with the starting time stamp on the GPU for the first kernel from the NVTX range, and stops with the stopping time stamp for the last kernel executed on the GPU from that same NVTX time range.

### 8.3. Aggregating Time

There are two ways that time is combined when computing statistics:

- **Flattening** is done by taking multiple time intervals and performing a union, where any intervals that share any time are joined. This eliminates any overlaps from being double counted. This is done when gathering global statistics such as GPU IDLE time, or when gathering parent node statistics from multiple children like the group_node report.

- **Accumulating** is done by taking multiple time intervals and summing their times together, while keeping a count of how many time intervals have been added. This is used when aggregating multiple instances of a single object, such as the GPU times for all instances of a single kernel or the CPU time for all instances of a single op node. The end result is the calculation of the total/average/min/max statistics that exist in most reports.
Chapter 9. Custom NVTX Ranges

In addition to the NVTX markers automatically added by the framework, the user can specify custom markers by annotating the model with custom NVTX ranges. This allows statistics and reports to be gathered for parts of the model that the user is most interested in.

In order to use custom NVTX markers, you will need to install nvtx-plugins:

```
pip install nvtx-plugins
```

To run an example model with custom NVTX ranges through DLProf, follow these instructions:

```
git clone https://github.com/NVIDIA/nvtx-plugins.git

cd nvtx-plugins

dlprof --reports=summary,detail /usr/bin/python \
examples/tf_session_example.py
```

That example is annotated with NVTX markers that put the forward pass in a new domain called "Forward", and the backward pass in a new domain called "Gradient". The result is that a summary and detail report will be created for the Forward domain and the Gradient domain in addition to the default domain reports that encompass the entire model.

More information on custom NVTX ranges can be found here: [https://nvtx-plugins.readthedocs.io/en/latest/](https://nvtx-plugins.readthedocs.io/en/latest/)
Chapter 10. Report Generation

DLProf can create several textual reports in both JSON and CSV formats. This section details the available reports that can be created.

10.1. Specifying Reports and Formats

This section discusses how to select which reports will be created and in what file formats.

10.1.1. Selecting Reports

A user may choose to generate reports by passing the report types to the --report option.

```
--reports=<type1>[,type2][,...]
```

The following types are allowed:

- `summary`: creates a Summary Report
- `detail`: creates a Detailed Report
- `iteration`: creates an Iteration Report
- `kernel`: creates a Kernel Report
- `tensor`: creates a Tensor Core Report
- `node_op`: creates a Node Op Report.
- `group_node`: creates a Group Node Report.

Some usage examples include:

```
--reports=kernel,iteration,summary
--reports iteration tensor node_op --
--reports summary
```

10.1.2. Selecting Domains

If the model has been annotated with custom NVTX ranges, then more than one domain will exist in the profile run. By default, DLProf will output the requested reports separately for
10.1.3. Selecting File Formats

By default, DLProf will create a CSV file for each report specified by --report. DLPROF can also output reports in a JSON file format. If multiple formats are selected, then a report will be created in each format, if possible. To specify the output format for the reports, use the --file_formats option:

```
--file_formats=<opt1>[,opt2][,...]
```

The following format options are allowed:

- **csv**: a comma-separated file format that can be easily imported into a spreadsheet
- **json**: a [JSON file format](https://en.wikipedia.org/wiki/JSON), useful for importing data into third-party applications

Some usage examples include:

```
--file_formats json
--file_formats=csv,json
--file_formats json csv --
```

10.1.4. Report Names

The file names for the reports are in the following format:

```
[base_name]_[report_type]_[domain_name].[csv|json]
```

Where [base_name] is the base report name, [report_type] is the same string passed to --reports to select the report, [domain_name] is the name of the domain (or blank for the default domain), and the final extension is either csv or json, depending on the file format. By default, the base name is `dlprof`, but can be changed using:

```
--profile_name <base_name>
```

For example, the following options:

```
--reports=summary,iteration --file_formats=csv,json --domains dom1,dom2
```

will create the following files:

- `dlprof_summary_dom1.csv`
- `dlprof_summary_dom1.json`
- `dlprof_iteration_dom1.csv`
- `dlprof_iteration_dom1.json`
- `dlprof_summary_dom2.csv`
- `dlprof_summary_dom2.json`
- `dlprof_iteration_dom2.csv`
- `dlprof_iteration_dom2.json`
10.1.5. Output Path

By default, all reports will be written in the current working directory. However, you may choose a different output directory for reports with:
```
--output_path <path/to/output>
```
where `<path/to/output>` is the new results folder. If the folder does not exist, DLProf will attempt to create it.

10.2. Summary Report

The Summary Report provides high level metrics on the performance results of all the operations and kernels in the entire model. This report contains several individual tables.

System config

This table provides configuration of the system used to profile the model.

<table>
<thead>
<tr>
<th>Row Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Count</td>
<td>The number of GPUs found while profiling.</td>
</tr>
<tr>
<td>GPU Name(s)</td>
<td>The name for each GPU found while profiling.</td>
</tr>
<tr>
<td>GPU Driver Version</td>
<td>The version of the GPU Driver found while profiling.</td>
</tr>
<tr>
<td>CUDA Version</td>
<td>The version of CUDA found while profiling.</td>
</tr>
<tr>
<td>DLProf Version</td>
<td>The version of DLProf used to profile.</td>
</tr>
<tr>
<td>DLProf Build</td>
<td>The build ID for DLProf used to profile.</td>
</tr>
</tbody>
</table>

Summary Report

<table>
<thead>
<tr>
<th>Row Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall Clock Time (ns)</td>
<td>Total wall clock time for the found iteration range.</td>
</tr>
<tr>
<td>Tensor Core Kernel Utilization %</td>
<td>$100 \times \frac{\text{Time of Tensor Core Kernels}}{\text{Total time of all kernels in Tensor Core eligible nodes}}$. Higher is better.</td>
</tr>
<tr>
<td>GPU Idle %</td>
<td>Percent of the time that the GPU is idle. Note, this includes the time that the GPU is waiting on the data pipeline. Lower is better.</td>
</tr>
</tbody>
</table>

Iterations

This table provides information on the number of iterations found and aggregated.

<table>
<thead>
<tr>
<th>Row Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Iterations</td>
<td>The total number of iterations found based on the number of unique key node operation instances.</td>
</tr>
<tr>
<td>Aggregated Iterations</td>
<td>The total number of iterations used to aggregate the profile data over and used to generate report data.</td>
</tr>
</tbody>
</table>
### Start Iteration
The starting iteration in the aggregated iteration range.

### Stop Iteration
The stopping iteration in the aggregated iteration range.

### Average Iteration Time [ns]
The average time for all aggregated iterations in nanoseconds.

### Iteration Time Standard Deviation [ns]
The standard deviation for all aggregated iterations in nanoseconds.

## All Ops
This table provides a high level breakdown of all operations found in the model.

<table>
<thead>
<tr>
<th>Row Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Op</td>
<td>Total metrics for all operations found.</td>
</tr>
<tr>
<td>Ops Using TC</td>
<td>Total metric for operations that use Tensor Core kernels.</td>
</tr>
<tr>
<td>Op eligible for TC but not using</td>
<td>Total metrics for operations that are eligible to use Tensor Core kernels but are currently not using any.</td>
</tr>
<tr>
<td>All other ops</td>
<td>Total metrics for all other operations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Time [ns]</td>
<td>Total time of all kernels for the operation spent executing on the GPU.</td>
</tr>
<tr>
<td>CPU Time [ns]</td>
<td>Total time for the operation spent executing on the CPU.</td>
</tr>
<tr>
<td># Ops</td>
<td>The number of unique operations found in this category.</td>
</tr>
</tbody>
</table>

## All Kernels
This table provides a high level breakdown of all kernels found in the model.

<table>
<thead>
<tr>
<th>Row Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Kernels</td>
<td>Total metrics for all kernels found.</td>
</tr>
<tr>
<td>Kernels Using TC</td>
<td>Total metric for kernels that use Tensor Core enabled operations.</td>
</tr>
<tr>
<td>Memory</td>
<td>Total metrics for operations that are eligible to use Tensor Core kernels but are currently not using any.</td>
</tr>
<tr>
<td>All other kernels</td>
<td>Total metrics for all other operations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Time [ns]</td>
<td>Total time of all kernels spent executing on the GPU.</td>
</tr>
</tbody>
</table>
10.3. Detailed Report

The Detailed Report contains correlated information for every group node, leaf node, and kernel executed in the profile. The report contains the GPU and CPU time metrics, kernel counts, and whether Tensor Core are used in the node. By sorting this report, a user can identify the top N GPU nodes or top N CPU nodes, identify quickly which nodes are using Tensor Cores and which can use Tensor Cores.

Each row in the table represents a unique node or operation in the model as determined by an NVTX range. The report contains the following columns:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the node / NVTX range.</td>
</tr>
<tr>
<td>Op Type</td>
<td>The operation type.</td>
</tr>
<tr>
<td>Origin</td>
<td>The source of the node.</td>
</tr>
<tr>
<td>No. Calls</td>
<td>Number of instances that the operation was called / executed.</td>
</tr>
<tr>
<td>TC Eligibility</td>
<td>Indicates if the node can use Tensor Cores based on operation name.</td>
</tr>
<tr>
<td>Using TC</td>
<td>Indicates if a Tensor Core enabled kernel was used by the node.</td>
</tr>
<tr>
<td>Total CPU Time (ns)</td>
<td>The total CPU time of all instances of the node.</td>
</tr>
<tr>
<td>Avg. CPU Time (ns)</td>
<td>The average CPU time of all instances of the node.</td>
</tr>
<tr>
<td>Min CPU Time (ns)</td>
<td>The minimum CPU time found amongst all instances of the node.</td>
</tr>
<tr>
<td>Max CPU Time (ns)</td>
<td>The maximum CPU time found amongst all instances of the node.</td>
</tr>
<tr>
<td>Total GPU Time (ns)</td>
<td>The total GPU time of all instances of the node.</td>
</tr>
<tr>
<td>Avg. GPU Time (ns)</td>
<td>The average GPU time of all instances of the node.</td>
</tr>
<tr>
<td>Min GPU Time (ns)</td>
<td>The minimum GPU time found amongst all instances of the node.</td>
</tr>
</tbody>
</table>
### Column name

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max GPU Time (ns)</td>
</tr>
<tr>
<td>The maximum GPU time found amongst all instances of the node.</td>
</tr>
<tr>
<td>Total CPU Overhead Time (ns)</td>
</tr>
<tr>
<td>The total CPU overhead of all instances of the node.</td>
</tr>
<tr>
<td>Avg. CPU Overhead Time (ns)</td>
</tr>
<tr>
<td>The average CPU overhead of all instances of the node.</td>
</tr>
<tr>
<td>Min CPU Overhead Time (ns)</td>
</tr>
<tr>
<td>The minimum CPU overhead found amongst all instances of the node.</td>
</tr>
<tr>
<td>Max CPU Overhead Time (ns)</td>
</tr>
<tr>
<td>The maximum CPU overhead found amongst all instances of the node.</td>
</tr>
<tr>
<td>Total GPU Idle Time (ns)</td>
</tr>
<tr>
<td>The total GPU idle time of all instances of the node.</td>
</tr>
<tr>
<td>Avg. GPU Idle Time (ns)</td>
</tr>
<tr>
<td>The average GPU idle time of all instances of the node.</td>
</tr>
<tr>
<td>Min GPU Idle Time (ns)</td>
</tr>
<tr>
<td>The minimum GPU idle time found amongst all instances of the node.</td>
</tr>
<tr>
<td>Max GPU Idle Time (ns)</td>
</tr>
<tr>
<td>The maximum GPU idle time found amongst all instances of the node.</td>
</tr>
<tr>
<td>Data Type</td>
</tr>
<tr>
<td>The data type of the operation. This column won’t exist if the user specifies detailed_mode=false.</td>
</tr>
<tr>
<td>Input Shapes</td>
</tr>
<tr>
<td>A list of shapes for all inputs into the operation. This column won’t exist if the user specifies detailed_mode=false.</td>
</tr>
</tbody>
</table>

CPU overhead is the time spent within the NVTX range that is not attributed to the CUDA API call. GPU idle time is the time between GPU kernel operations for a node when the GPU is not executing a kernel.

### 10.4. Iteration Report

The Iteration Report lists each kernel executed for every node and on every iteration. The kernel start time has been included as well, so the table can be sorted chronologically by kernels. Each row in the iteration report represents an instance of a kernel call. The report contains the following columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Name</td>
<td>The name of the node / NVTX range that call the kernel.</td>
</tr>
<tr>
<td><strong>Column Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Node Op</td>
<td>The TensorFlow operation name.</td>
</tr>
<tr>
<td>Kernel Name</td>
<td>The name of the GPU kernel.</td>
</tr>
<tr>
<td>Iteration</td>
<td>The iteration interval number that the kernel was launched.</td>
</tr>
<tr>
<td>Uses TC</td>
<td>True if the kernel uses Tensor Cores.</td>
</tr>
<tr>
<td>API Call Start (ns)</td>
<td>The time stamp for when the kernel was called by the CPU.</td>
</tr>
<tr>
<td>API Call Time (ns)</td>
<td>The time spent on the CPU making the CUDA API call.</td>
</tr>
<tr>
<td>GPU Time (ns)</td>
<td>The time spent on the GPU executing the kernel.</td>
</tr>
</tbody>
</table>

See [Iteration Detection](#) for more information on how to specify iteration intervals.

## 10.5. Kernel Report

The Kernel Report lists all the kernels launched in the network. Unlike the Iteration Report, this report contains an entry in the report for each unique kernel and provides timing metrics for instances of that kernel. The report contains the following columns:

<table>
<thead>
<tr>
<th><strong>Column</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel Name</td>
<td>The name of the GPU kernel.</td>
</tr>
<tr>
<td>Node Name</td>
<td>The name of the node / NVTX range that call the kernel.</td>
</tr>
<tr>
<td>Uses TC</td>
<td>True if the kernel uses Tensor Cores.</td>
</tr>
<tr>
<td>Total GPU Time (ns)</td>
<td>The total GPU time for all instances of the node.</td>
</tr>
<tr>
<td>Avg. GPU Time (ns)</td>
<td>The average GPU time for all instances of the node.</td>
</tr>
<tr>
<td>Min GPU Time (ns)</td>
<td>The minimum GPU time found amongst all instances of the node.</td>
</tr>
<tr>
<td>Max GPU Time (ns)</td>
<td>The maximum GPU time found amongst all instances of the node.</td>
</tr>
<tr>
<td>Total API Time (ns)</td>
<td>The total CPU time spent on CUDA API call for all instances of the node.</td>
</tr>
<tr>
<td>Avg. API Time (ns)</td>
<td>The average CPU time spent on CUDA API for all instances of the node.</td>
</tr>
</tbody>
</table>
### 10.6. Tensor Core Report

The Tensor Core Report lists all unique Tensor Core kernels that were executed. The report contains the following columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Name</td>
<td>The name of the node / NVTX range that call the kernel.</td>
</tr>
<tr>
<td>Node Op</td>
<td>The TensorFlow operation name.</td>
</tr>
<tr>
<td>Uses TC</td>
<td>True if the node uses Tensor Cores.</td>
</tr>
<tr>
<td>Total GPU Time</td>
<td>The total GPU time for all instances of the node.</td>
</tr>
<tr>
<td>TC GPU Time</td>
<td>The GPU time executing Tensor Cores for all instances of the node.</td>
</tr>
<tr>
<td>Non-TC GPU Time</td>
<td>The GPU time not executing Tensor Cores for all instances of the node.</td>
</tr>
<tr>
<td>TC Utilization (%)</td>
<td>100 * (TC GPU Time) / (Total GPU Time)</td>
</tr>
<tr>
<td>Total Kernel Count</td>
<td>The total number of unique kernels executed by the node.</td>
</tr>
<tr>
<td>TC Kernel Count</td>
<td>The total number of unique kernels that use Tensor Cores for this node.</td>
</tr>
<tr>
<td>TC Kernel Names</td>
<td>A list of all the names of kernels using Tensor Cores for this node.</td>
</tr>
<tr>
<td>Non-TC Kernel Count</td>
<td>The total number of unique kernels that do not use Tensor Cores for this node.</td>
</tr>
<tr>
<td>Non-TC Kernel Names</td>
<td>A list of all the names of kernels are not using Tensor Cores for this node.</td>
</tr>
</tbody>
</table>
10.7. Node Op Report

The Node Op Report lists leaf nodes in the network. For each instance of the node, the CPU and GPU times are flattened and rolled up. Statistical values are calculated across the individual instances to find the total sum, average, minimum, and maximum values for each measured metric. The report generates a table with the following columns:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Op</td>
<td>The TensorFlow operation name.</td>
</tr>
<tr>
<td>No. Nodes</td>
<td>The total number of nodes operation is included in.</td>
</tr>
<tr>
<td>No. Calls</td>
<td>Number of instances that the operation was called / executed.</td>
</tr>
<tr>
<td>TC Eligibility</td>
<td>Indicates if the node can use Tensor Cores based on operation name.</td>
</tr>
<tr>
<td>Using TC</td>
<td>Indicates if a Tensor Core enabled kernel was used by the node.</td>
</tr>
<tr>
<td>Total CPU Time (ns)</td>
<td>The total CPU time of all instances of the node.</td>
</tr>
<tr>
<td>Avg. CPU Time (ns)</td>
<td>The average CPU time of all instances of the node.</td>
</tr>
<tr>
<td>Min CPU Time (ns)</td>
<td>The minimum CPU time found amongst all instances of the node.</td>
</tr>
<tr>
<td>Max CPU Time (ns)</td>
<td>The maximum CPU time found amongst all instances of the node.</td>
</tr>
<tr>
<td>Total GPU Time (ns)</td>
<td>The total GPU time of all instances of the node.</td>
</tr>
<tr>
<td>Avg. GPU Time (ns)</td>
<td>The average GPU time of all instances of the node.</td>
</tr>
<tr>
<td>Min GPU Time (ns)</td>
<td>The minimum GPU time found amongst all instances of the node.</td>
</tr>
<tr>
<td>Max GPU Time (ns)</td>
<td>The maximum GPU time found amongst all instances of the node.</td>
</tr>
<tr>
<td>Total CPU Overhead Time (ns)</td>
<td>The total CPU overhead of all instances of the node.</td>
</tr>
<tr>
<td>Avg. CPU Overhead Time (ns)</td>
<td>The average CPU overhead of all instances of the node.</td>
</tr>
</tbody>
</table>
### 10.8. Group Node Report

The Group Node Report lists all non-leaf nodes in the network. For each non-leaf node, it flattens and rolls up all statistics from its sub-tree. All metrics are calculated on a per-iteration basis. The report contains the following columns:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name (hierarchy) of the sub-tree.</td>
</tr>
<tr>
<td>No. Calls Aggregated</td>
<td>Total number of leaf node instances in this sub-tree.</td>
</tr>
<tr>
<td>No. TC Eligibility Node Ops</td>
<td>Total number of leaf nodes in this sub-tree that are eligible to use Tensor Cores.</td>
</tr>
<tr>
<td>No. Node Ops Using TC</td>
<td>Total number of leaf nodes in this sub-tree that use Tensor Cores.</td>
</tr>
<tr>
<td>Total CPU Time [ns]</td>
<td>The total CPU time of all instances of the sub-tree.</td>
</tr>
<tr>
<td>Avg. CPU Time [ns]</td>
<td>The average CPU time for all instances of the sub-tree on a per-iteration basis.</td>
</tr>
<tr>
<td>Min CPU Time [ns]</td>
<td>The minimum CPU time for all instances of the sub-tree on a per-iteration basis.</td>
</tr>
<tr>
<td>Max CPU Time [ns]</td>
<td>The maximum CPU time for all instances of the sub-tree on a per-iteration basis.</td>
</tr>
<tr>
<td>Total GPU Time [ns]</td>
<td>The total GPU time for all instances of the sub-tree.</td>
</tr>
</tbody>
</table>
10.9. **Expert Systems Report**

The expert systems report will list all of the problems detected by Expert Systems and give actionable feedback for how to resolve the potential problems. The report contains the following columns:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>The potential problem that was discovered.</td>
</tr>
<tr>
<td>Recommendation</td>
<td>The recommended action to take to try to resolve the problem.</td>
</tr>
</tbody>
</table>
10.10. Expert Systems

Expert Systems is a feature (currently in beta) that analyzes the model and the profile data to detect potential problems or inefficiencies. Any problems detected will come with a recommendation of action for you to attempt to resolve the issue. The results can be found by enabling the Expert Systems Report.

Expert Systems contains a number of problem detectors. Each detector will look for a specific problem. More detectors are planned in the future. Here is the current list of detectors and what they look for:

<table>
<thead>
<tr>
<th>Name</th>
<th>Problem Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad Iteration Range Detector</td>
<td>Detects the case when the Iteration Range contains a lot of variations between iterations.</td>
</tr>
<tr>
<td>No Iteration Detector</td>
<td>Detects the case where no iterations are found because the Key Node is unspecified or invalid.</td>
</tr>
<tr>
<td>Bad Tensor Shape Detector</td>
<td>Detects the case where tensor cores are not used because of bad tensor shape or data type in the model.</td>
</tr>
<tr>
<td>No Fusion Detector</td>
<td>Detects the case where fusion is disabled.</td>
</tr>
</tbody>
</table>

10.11. XLA Support

DLProf supports profiling with the XLA compiler engaged. XLA is a deep learning graph compiler for Tensorflow. For information on how to enable XLA when training DNNs see: https://www.tensorflow.org/xla/#enable_xla_for_tensorflow_models.

Run DLProf with XLA to improve GPU Tensor Core utilization, visualize clustering nodes in TensorBoard graphs plugin to understand optimization. For more information see the DLProf Plugin for TensorBoard User Guide.
Chapter 11. User Goals

When profiling any computer program, the objective is to inspect its code to determine if performance can be maximized. In DLProf, profiling determines if GPUs are being fully utilized to take advantage of the hardware optimization and whether utilization can be improved without loss of accuracy. Typically, profiling is done at the time of training a model, so that adjustments can be made based on the results. DLProf can be used to understand how a deep learning model is performing with respect to the Tensor Core hardware. Objectives may be summarized as follows:

1. Determine how the deep learning model performed in terms of GPU utilization time and percent usage as a summary.
2. Understand visually, with inspection of the TensorBoard graph, the prominent nodes where optimization with mixed precision is possible.
3. Drill down into the prominent node to understand individual operations and their correlation with Tensor Core compatibility.
4. Get deeper insights into Kernel level information, which kernels are using Tensor Cores, for how long and what percent of the entire run.

11.1. How do I profile a deep learning network?

Start with downloading the NGC TensorFlow container. Generate a graphdef file of the DNN that you want to profile.

Issue the `dlprof` command to profile training run. Nvidia recommends running the model for 50 to 100 iterations or batches. If the model is recursive or variable length, such as an RNN, the recommended number of iterations is between 15 and 25.

11.2. How can I improve my network if I’m not using Tensor Cores?

Navigate to the Top 10 Op Nodes and sort by GPU. Find the longest running Op Node in the last that is eligible for Tensor Cores, but is not using Tensor Cores. In the python code, find
11.3. How do I find a good Key Op?

Using a good node for the key op is essential to maximize the analysis at a per iteration step level. By default, DLProf will use a key node that is found by default in most framework models. However, this is not always the case and there may be models that the node operation does not exist. In this event, DLProf will alert the user to find a new key node.

First, run the profile as normal, making sure to save the generated SQL database and the Detailed Report. If possible, try to limit the number of steps executed, such as running only 20 batches.

```
dlprof --reports=detail python my_script.pl
```

In the detail report, sort the data by "No. Calls". Choose a node operation that has "No. Calls" value that is close to the same number of steps executed in the script. You will also want to choose the operation that occurs as close to the start of the iteration step as possible. This may require analyzing the graph or script.

When a node has been selected, you can re-run DLProf on the previous profile to specify the new Key Node without re-profiling the model.

```
dlprof --key_node=[<node_name>] --nsys_database=[<nsys.sqlite>]
```

DLProf should now show more than one iteration found and the Iteration Report will provide a GPU usage breakdown per iteration.

11.4. How do I choose a good iteration range to analyze?

By default, DLProf aggregates the data for all iterations found in the profile. However, this may not provide an accurate representation of the stabilized run-time performance. Most frameworks will typically have special warm-up and tear-down iteration, which can dramatically throw off the overall results. For a standard non-RNN network, iterations should have little time variation.

To find an ideal set of iteration, run the profile as normal, but generate an iteration report and save the generated SQL database.

```
dlprof --reports=iteration python my_script.pl
```

It should be trivial to view the iteration report in a spreadsheet application. A simple bar chart on the iteration times can expose an iteration range that has consistent time. Note the starting and stopping iteration number in this range. DLProf can then be re-ran using this new range and the existing SQL database to quickly re-aggregate the results over that range.

```
dlprof --iter_start=[<start_iter>] --iter_stop=[<stop_iter>] --nsys_database=[<nsys.sqlite>]
```
All reports will then be generated from data gathered only between and including these two iterations.

11.5. How can I find out profile data for a TensorFlow-TensorRT inference graph?

DLProf supports profiling TF-TRT graphs which means it can profile models that have been optimized with TensorRT and aggregate GPU time over the optimized TF nodes.

TF-TRT inference workflow allows TensorRT optimizations to be performed when running inference for a Tensorflow model.

NVIDIA containers of TensorFlow are built with enabling TensorRT, TF-TRT is part of the TensorFlow binary in the container and can be used out of the box. To run a dlprof inference workflow with TF-TRT:

- Freeze trained model checkpoints
- Deserialize frozen graph
- Convert with TF-TRT with a few lines of code
- Run dlprof with TF-TRT to visualize the TRTEngineOp operation and the GPU time spent

Here is an example:

After you deserialize frozen graph, create a TensorRT graph using the conversion script.

```python
converter = trt.TrtGraphConverter(input_graph_def=frozen_graph,
                                  nodes_blacklist=['logits', 'classes'])
trt_graph = converter.convert()
```

Generate synthetic input

```python
features = np.random.normal(loc=112, scale=70, size=(8, 3, 224, 224)).astype(np.float32)
features = np.clip(features, 0.0, 255.0)
```

Import TensorRT graph into a new graph and run for 200 iterations

```python
new_graph = tf.import_graph_def(trt_graph, input_map={'input': features},
                                return_elements=['logits', 'classes'])
for i in range(0, 200):
    sess.run(new_graph)
```

Run DLProf specifying the key node as TRTEngineOp_0

```bash
dlprof --key_node= import/resnet_v1_50/TRTEngineOp_0 --iter_start 51 --iter_stop 200
python inference.py
```

On TensorBoard you can see the time spent in different operations.

You can try to run examples from nvidia-examples	ensorrt	ftrt in the container.
Chapter 12. Tutorials

The following tutorial examples are run within the NGC TensorFlow container. See Profiling from the NGC TensorFlow Container for instructions on how to setup and run the container.

12.1. Resnet50

This is an example of running DLProf to profile Resnet50 model (resnet50_v1.5) located in the /workspace/tensorflow-examples/models directory of the NGC TensorFlow container.

12.1.1. Preparing the Example

1. Copy training data locally to /path/to/training/data Training data can be downloaded from ImageNet.
2. Run the NGC TensorFlow container, mapping the training data and result data directories.

```bash
docker run --privileged --gpus=1 --rm --shm-size=1g --ulimit memlock=-1
--ulimit stack=67108864 -it -p6006:6006 -v<path/to/training/data>:/data
-v<path/to/results>:/results nvcr.io/nvidia/tensorflow:20.06-tf1-py3
```

12.1.2. Profiling Resnet50

To profile with DLProf, use the command shown below. This command will profile over the training data and generate detailed reports in addition to TensorBoard event files.

```bash
$ cd /workspace/nvidia-examples/resnet50v1.5
$ mkdir results
$ dlprof --graphdef=auto --reports=summary,detail,iteration
   --iter_start 20 --iter_stop 80
   /usr/bin/python main.py
   --mode=train --iter_unit=batch --num_iter=100
   --batch_size=128 --warmup_steps=1 --use_cosine_lr
   --label_smoothing 0.1 --lr_init=0.256 --lr_warmup_epochs=8
   --momentum=0.875 --weight_decay=3.0517578125e-05 --use_tf_amp
   --data_dir=/data/train-val-tfrecord-480 --results_dir=./results
```

This command profiles 100 batches of the NVIDIA Resnet50 example using Automatic Mixed Precision [AMP]. There will be three output report files in /workspace/nvidia-examples/resnet50v1.5.

- dlprof_summary.csv - The summary report
- dlprof_detailed.csv - The detailed node report
12.1.3. Viewing Results in TensorBoard

TensorBoard event files will also be added to /workspace/nvidia-examples/resnet50v1.5/event_files and can be launched in TensorBoard with

$ tensorboard --logdir /workspace/nvidia-examples/resnet50v1.5/event_files

To view TensorBoard, enter http://<IP Address>:6006 in a browser.

12.1.4. Profiling with Simple Mode

Simple mode allows gathering profile data for any application no matter the underlying framework. You can use --delay and --duration options to limit the profile window for the training script. Following is an example of how to use simple mode and the example output produced.

Example:

dlprof --mode=simple /usr/bin/python main.py --mode=train --iter_unit=batch --num_iter=100
--batch_size=128 --use_tf_amp --data_dir=/data/train-val-tfrecord-480 --results_dir=./results

Output:
Wall Time (ns): 96073471820
Total GPU Time (ns): 67737443513
TC GPU Time (ns): 5084521010

12.2. MobileNet

Here’s an example of running DLProf to profile MobileNetV2 from TensorFlow.

12.2.1. Preparing the Example

1. Copy training data locally to /path/to/training/data
   Training data can be downloaded from ImageNet http://image-net.org/download
2. Run the NGC TensorFlow docker container, and mapping the training data and result data directories.
   docker run --privileged --rm --gpus=1 --shm-size=1g --ulimit memlock=-1
   --ulimit stack=67108864 -it -p6006:6006 -v<path/to/training/data>:/data
   -v<path/to/results>:/results nvcr.io/nvidia/tensorflow:20.06-tf1-py3
3. In the docker container, install the TensorFlow benchmarks into /workspace
   mkdir /workspace/tensorflow-examples &&
   cd /workspace/tensorflow-examples &&
   git clone https://github.com/tensorflow/models &&
   git clone https://github.com/tensorflow/benchmarks.git &&
   cd benchmarks &&
   git checkout cnn_tf_v1.15_compatible &&
   export PYTHONPATH=/workspace/tensorflow-examples/models &&
   cd /workspace/tensorflow-examples/benchmarks/scripts/tf_cnn_benchmarks
12.2.2. Profiling MobileNet

The following command line is the minimum needed to profile the model and generate an event file.

dlprof \
/usr/bin/python tf_cnn_benchmarks.py \n--num_gpus=1 --batch_size=256 --model=mobilenet --device=gpu --gpu_indices=1 \n--data_name=imagenet --data_dir=/data/train-val-tfrecord-480 \n--num_batches=50 --use_fp16 --fp16_enable_auto_loss_scale

The only report output will be the TensorBoard event files which can be found in: 
/workspace/tensorflow-examples/benchmarks/scripts/tf_cnn_benchmarks/event_files

12.2.3. Viewing Results in TensorBoard

The following command line will launch TensorBoard.

tensorboard --logdir ./event_files

To view TensorBoard, enter http://<IP Address>:6006 in a browser.
Chapter 13. Troubleshooting FAQ

13.1. Error loading libnvidia-ml.so.1

If you get this error:

dlprof: error while loading shared libraries: libnvidia-ml.so.1: cannot open shared object file: No such file or directory

You may not meet the prerequisite drivers and CUDA version. Update your driver and CUDA SDK to match the minimal versions needed for this release.
Chapter 14. Reference

The following section contains additional reference material.

14.1. Usage

dlprof [options] [command line to run model]
dlprof [options] --in_graphdef=<graph.pbtxt|graphs_dir> --
in_nsys_db_filename=<nsys.sqlite>

» The command line to profile the model is not needed when specifying the graphdef and
  database. In this mode, DLProf can be freely rerun to generate different reports and
  aggregate over different iterations.
» By default, only the TensorBoard event files will be created. Additional options are needed
  to generate other reports.

You must use one of the following formats for options:

Single arguments

» --optionA=val1
» --optionA val1

Multiple arguments

» --optionA=val1,val2
» --optionA val1 val2 --
» --optionA=val1 --optionA=val2
» --optionA val1 --optionA val2

14.2. Command Line Options

The NVIDIA Deep Learning Profiler command lines can have one of two forms:
dlprof [optional switch_options] [application] [optional application_options]
or
dlprof [optional switch_options] --nsys_database=<nsys.sqlite>
All command line options are case sensitive. For optional switch options, when short options are used, the parameters should follow the switch after a space; e.g. `-m simple`. When long options are used, the switch should be followed by an equal sign and then the parameter(s); e.g. `--mode=simple`.

### 14.2.1. CLI Global Options

The following options are available for every mode and framework build, unless otherwise specified.

<table>
<thead>
<tr>
<th>Global Options Short</th>
<th>Global Options Long</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--help</td>
<td>Help message providing information about available command switches and their options.</td>
</tr>
<tr>
<td>-V</td>
<td>--version</td>
<td>Output the NVIDIA Deep Learning Profiler version information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generic Option Short</th>
<th>Generic Option Long</th>
<th>Possible Parameters</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-f</td>
<td>--force</td>
<td>true, false</td>
<td>false</td>
<td>If true, overwrite all existing result files with the same output filename (QDSTREM, QDREP, SQLITE, CSV, JSON).</td>
</tr>
<tr>
<td>-v</td>
<td>--verbosity</td>
<td>quiet, minimal, normal, detailed, diagnostic</td>
<td>normal</td>
<td>Specify the output verbosity level.</td>
</tr>
<tr>
<td>-m</td>
<td>--mode</td>
<td>simple, tensorflow1, pytorch</td>
<td>build dependent</td>
<td>Specify the target framework being profiled. Use 'simple' to generate only high level metrics agnostic to any framework. Use all other options to generate detailed metrics and reports specific to the framework.</td>
</tr>
</tbody>
</table>

The mode options and availability is dependent on the specific container build. For a simple mode only build, the `--mode` switch is not available and defaults to 'simple'. For all other framework specific builds, the default mode is the target framework; e.g. 'tensorflow1' for the DLProf built for the TensorFlow 1.x NGC container. All framework specific builds will always have the option to run in simple mode.
## 14.2.2. Nsight System Options

The following Nsight System options are available for all profile modes unless otherwise specified.

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Possible Parameters</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--nys_database</td>
<td>&lt; database filename &gt;</td>
<td>NA</td>
<td>Input SQLITE file generated by Nsight Systems. When specified, DLProf will aggregate profile data directly from the database. This option can be used to evaluate different aggregation options or generate new reports. If specified, additional application commands are ignored and the application will not be profiled.</td>
<td></td>
</tr>
<tr>
<td>-y</td>
<td>--delay</td>
<td>&lt; seconds &gt;</td>
<td>0</td>
<td>Collection start delay in seconds.</td>
</tr>
<tr>
<td>-d</td>
<td>--duration</td>
<td>&lt; seconds &gt;</td>
<td>NA</td>
<td>Collection duration in seconds, duration must be greater than zero.</td>
</tr>
<tr>
<td>--detailed_mode</td>
<td>true, false</td>
<td>true</td>
<td>Gather detailed NVTX information, including tensor.</td>
<td></td>
</tr>
</tbody>
</table>
Note: `--detailed_mode` is a feature that is only available in tensorflow1 operation mode or in the TensorFlow 1.x NGC container. This feature is enabled by default and will add more information to the NVTX markers which may increase profiling overhead.

Setting custom Nsight Systems options requires surrounding the custom options in quotes, e.g. `--nsys_options="-t cuda,nvtx -s none"`.

### 14.2.3. Data Aggregation Options

Data aggregation options are not available in simple mode or the simple mode only builds of DLProf.

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Possible Parameters</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--key_node</td>
<td>&lt; node name &gt;</td>
<td></td>
<td>global_step</td>
<td>Iteration intervals are determined from the NVTX start times of each key node instance. If DLProf is not detecting intervals correctly, try specifying a different key node.</td>
</tr>
<tr>
<td>--iter_start</td>
<td>&lt; iteration number &gt;</td>
<td></td>
<td>0</td>
<td>Set the iteration interval to start aggregating data. Profile data from iteration intervals less than the starting interval are excluded from all aggregated reports.</td>
</tr>
<tr>
<td>--iter_stop</td>
<td>&lt; iteration number &gt;</td>
<td></td>
<td>NA</td>
<td>Set the iteration interval to stop aggregating data. Profile data from iteration intervals greater than the stopping interval are excluded from all aggregated reports. The stop iteration number must be greater than or equal to the start iteration.</td>
</tr>
<tr>
<td>Short</td>
<td>Long</td>
<td>Possible Parameters</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
<td>---------------------------------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>--output_path</td>
<td>&lt; path &gt;</td>
<td>./</td>
<td>Specify the output path for all generated aggregated collateral.</td>
</tr>
<tr>
<td></td>
<td>--base_name</td>
<td>&lt; base name &gt;</td>
<td>dlprof</td>
<td>Specify a common base name that is prepended to all generated report file names.</td>
</tr>
<tr>
<td></td>
<td>--reports</td>
<td>summary, detail, kernel, iteration, tensor, op_type, group_node, expert_systems</td>
<td>NA</td>
<td>Select the aggregated report[s] to generate. Multiple reports can be selected, separated by commas only (no spaces).</td>
</tr>
<tr>
<td></td>
<td>--formats</td>
<td>csv, json</td>
<td>csv</td>
<td>Specify output file format[s]. Multiple formats can be selected,</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--domains &lt; domain list&gt;</td>
<td>Specify NVTX domains to aggregate. Multiple domains can be selected, separated by commas only (no spaces). Every requested report is created for each specified domain.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--dump_model_data true, false</td>
<td>If true, a json file is created that contains the raw, correlated model data.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--tb_dir &lt; folder name &gt; event_files</td>
<td>Specify the output directory for all generated TensorBoard event files.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-b --suppress_tb_files true, false</td>
<td>If true, TensorBoard event files will not be created.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The only available report type for simple mode is 'kernel'. The --tb_dir and --suppress_tb_files are not available in simple mode, since simple mode does not generate any TensorBoard event files.
Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.