# TABLE OF CONTENTS

Chapter 1. Deep Learning Profiler ................................................................. 1
  1.1. Overview .................................................................................................. 1
  1.2. What’s New in 0.10.0 ............................................................................. 1
  1.3. Features ................................................................................................. 1

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

Chapter 3. Profiling ....................................................................................... 5
  3.1. Profiling from the NGC TensorFlow Docker Container ............................... 5
  3.2. Running DLProf ................................................................................... 6
  3.3. Profiling with Nsight Systems ................................................................. 6
  3.4. Profiling with Delay and Duration .......................................................... 7
  3.5. Running DLProf without Profiling .......................................................... 7
  3.6. Automatically Generating a Graphdef File ............................................... 8
  3.7. Supplying a Graphdef .......................................................................... 8
  3.8. Creating a GraphDef File ....................................................................... 8

Chapter 4. Tensor Core usage......................................................................... 10
  4.1. Mixed Precision Training ....................................................................... 10
  4.2. Determining Tensor Core Eligibility ...................................................... 10

Chapter 5. TensorBoard Plugin ..................................................................... 11
  5.1. Generating TensorBoard Event Files ..................................................... 11
  5.2. Starting TensorBoard ........................................................................... 11

Chapter 6. Iteration Detection ..................................................................... 12
  6.1. Specifying the Key Node ....................................................................... 12
  6.2. Limiting Aggregation to an Iteration Range ......................................... 13

Chapter 7. Correlating Time with NVTX Markers ........................................... 14
  7.1. NVTX Markers in TensorFlow ................................................................ 14
  7.2. Mapping GPU Time ............................................................................ 14
  7.3. Custom NVTX Ranges ......................................................................... 15
  7.4. Aggregating Time ................................................................................ 15

Chapter 8. Report Generation ...................................................................... 16
  8.1. Specifying Reports and Formats ........................................................... 16
    8.1.1. Selecting Reports ........................................................................... 16
    8.1.2. Selecting Domains .......................................................................... 16
    8.1.3. Selecting File Formats .................................................................... 17
    8.1.4. Report Names ................................................................................ 17
    8.1.5. Output Path .................................................................................. 18
  8.2. Summary Report .................................................................................. 18
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 0.10.0

- Expert Systems feature that analyzes performance results, looks for common performance issues, and suggests recommended fixes that may improve performance
- Support for additional domains from custom NVTX markers
  - Reports are generated for the domain specified using markers
  - Data is aggregated only from NVTX markers in the same domain
- Passing a Graphdef is now optional. User can specify a Graphdef with --graphdef or set it to auto for a TensorBoard graph event file to be created.
- System information is gathered in the background and exposed to the summary report, database, and TensorBoard event files.
- Consistent command line arguments.

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.
- **Custom TensorBoard Plugin**: DLprof can automatically generate TensorBoard event files. These event files are used with NVIDIA's GPU Tensorboard plugin 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 show the XLA generated operations.

- **Support Custom NVTX Markers and Domains**: DLProf will support custom NVTX markers and domains specified with the NVTX Plugin.

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

- **Support for profiling Tensorflow-TensorRT inference**: DLProf can profile optimized TF-TRT graph and show timing data for TRT-compatible subgraph.
Chapter 2.
QUICK START

DLProf is still in beta and is only 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:

```
$ docker pull nvcr.io/nvidia/tensorflow:<xx.yy>-tf1-py3
```

Where `<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:

```
$ docker run --rm --gpus=1 --shm-size=1g --ulimit memlock=-1 \
--ulimit stack=67108864 -it -p6006:6006 -v /full/path/to/training/data:/data \
nvcr.io/nvidia/tensorflow:<xx.yy>-tf1-py3
```

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.
- `graph_dump`: A folder containing Tensorflow Graphdef files generated automatically 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 DLProf Plugin for TensorBoard User Guide for more information.
Chapter 3. PROFILING

The NVIDIA Deep Learning Profiler (DLProf) is still in beta. It is currently only available in the NGC TensorFlow container. Note that due to the beta status, backwards compatibility is not guaranteed. Command line arguments, file formats, and event file protobuffs may change between releases. For the best experience, make sure to compatible versions 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 tools provide different reports to aid in identifying bottlenecks and Tensor Core usage.

3.1. 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.

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 pull the TensorFlow 1.x container using the following command:

docker pull nvcr.io/nvidia/tensorflow:20.03-tf1-py3
Replace the current profiler version with the version of the profiler release 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 using this command:

```
$ docker run --rm --gpus=1 --shm-size=1g --ulimit memlock=-1 \
--ulimit stack=67108864 -it -p6006:6006 -v/full/path/to/training/data:/data \
nvcr.io/nvidia/tensorflow:20.03-tf1-py3
```

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` option maps /full/path/to/training/data on the host into the container at /data. You can 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.

### 3.2. 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`.

```
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.
- **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 [options] <training_script.py>
```

### 3.3. 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.

### 3.4. 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.

### 3.5. 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 range. 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 [options] --nsys_database <nsys_profile.sqlite>
```

where `<nsys_profile.sqlite>` is the SQLite file generated by Nsight Systems. All other DLProf options are valid and optional.

### 3.6. 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.

Only the TensorFlow GraphDef model is officially supported and tested by DLProf in this release.

When using the option below, DLProf will automatically attempt to generate a graphdef file from Tensorflow:

```
--graphdef=auto
```

This creates the graphdef_dump/ folder in the working directory and generates a GraphDef for each TensorFlow session. DLProf automatically combines all GraphDefs together for viewing in TensorBoard.

This option should be used when profiling with XLA enabled.

### 3.7. 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
```

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.

### 3.8. 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:

```
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 multiply and accumulate operations in a single operation.

4.1. 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 and Turing architecture, significant training speedups are experienced by switching to mixed precision -- up to 3x overall speedup on the most arithmetically intense model architectures.

4.2. 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.
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 refer to the NVIDIA GPU Plugin for TensorBoard User Guide.

5.1. Generating TensorBoard Event Files

By default, DLProf generates two TensorBoard event files, tfevents, `<xxx>`. `<yyy>` and tfgpusummary. `<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`

5.2. 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 6.
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:

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

Resulting in N+1 intervals.

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.

6.1. Specifying the Key Node

By default, DLProf will look for **global_step** as the key node. However, not all models will use this 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.
6.2. 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 7.
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.

7.1. 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.

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 the profilers as well, such as Tensor Core usage, start time, and stop time.

7.2. 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.

### 7.3. 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.

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

```bash
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/)

### 7.4. 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 8.
REPORT GENERATION

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

8.1. Specifying Reports and Formats

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

8.1.1. 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.
- **expert_systems**: creates an Expert Systems Report.

Some usage examples include:

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

8.1.2. 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 each domain, including the default domain. If one or more domains are specified via the --domains option, then reports will only be generated for the requested domains:

```
--domains=<domain1>,[domain2][,...]
```

### 8.1.3. 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://www.nvidia.com), useful for importing data into third-party applications

Some usage examples include:

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

### 8.1.4. 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 with:

```
--report_base_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`
8.1.5. 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 and path does not exist, then DLProf will attempt to create it.

8.2. Summary Report

The Summary Report provides high level metrics on the performance results of all the operations and kernels in the entire model.

The report contains the following rows:

<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>Number of found iterations</td>
<td>Number of iterations found in the model</td>
</tr>
<tr>
<td>Average Iteration Time (ns)</td>
<td>Average time of each iteration</td>
</tr>
<tr>
<td>Iteration Time Standard Deviation</td>
<td>Standard deviation of the iteration time</td>
</tr>
<tr>
<td>Tensor Core Utilization</td>
<td>100 * (Time of Tensor Core Kernels) / (Total time of all kernels in Tensor Core eligible nodes)</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.</td>
</tr>
<tr>
<td>All nodes</td>
<td>CPU time, GPU time, and count of all nodes in the run</td>
</tr>
<tr>
<td>Nodes using TC</td>
<td>CPU time, GPU time, and count of nodes that use Tensor Cores</td>
</tr>
<tr>
<td>Nodes eligible for TC but not using</td>
<td>CPU time, GPU time, and count of nodes that are eligible to use Tensor Cores but don’t end up using them</td>
</tr>
<tr>
<td>All other nodes</td>
<td>CPU time, GPU time, and count of nodes that are not eligible to use Tensor Cores</td>
</tr>
<tr>
<td>All Kernels</td>
<td>CPU time, GPU time, and count of all kernels in the run</td>
</tr>
<tr>
<td>Kernels using TC</td>
<td>CPU time, GPU time, and count of all kernels that use Tensor Cores</td>
</tr>
</tbody>
</table>
8.3. 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 node, 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>Node Op</td>
<td>The TensorFlow operation name</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>
</tbody>
</table>
### Column name | Description
---|---
Avg. GPU Time (ns) | The average GPU time of all instances of the node
Min GPU Time (ns) | The minimum GPU time found amongst all instances of the node
Max GPU Time (ns) | The maximum GPU time found amongst all instances of the node
Total CPU Overhead Time (ns) | The total CPU overhead of all instances of the node
Avg. CPU Overhead Time (ns) | The average CPU overhead of all instances of the node
Min CPU Overhead Time (ns) | The minimum CPU overhead found amongst all instances of the node
Max CPU Overhead Time (ns) | The maximum CPU overhead found amongst all instances of the node
Total GPU Idle Time (ns) | The total GPU idle time of all instances of the node
Avg. GPU Idle Time (ns) | The average GPU idle time of all instances of the node
Min GPU Idle Time (ns) | The minimum GPU idle time found amongst all instances of the node
Max GPU Idle Time (ns) | The maximum GPU idle time found amongst all instances of the node
Data Type | The data type of the operation. This column won't exist if the user specifies detailed_mode=false
Input Shapes | A list of shapes for all inputs into the operation. This column won't exist if the user specifies detailed_mode=false

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.

### 8.4. 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:
8.5. 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>Column</th>
<th>Description</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>
</tbody>
</table>

See Iteration Detection for more information on how to specify iteration intervals.
### Column Description

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. API Time (ns)</td>
<td>The average CPU time spent on CUDA API for all instances of the node</td>
</tr>
<tr>
<td>Min API Time (ns)</td>
<td>The minimum CPU time spent on CUDA API found amongst all instances of the node</td>
</tr>
<tr>
<td>Max API Time (ns)</td>
<td>The maximum CPU time spent on CUDA API found amongst all instances of the node</td>
</tr>
</tbody>
</table>

#### 8.6. 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>Node Origin</td>
<td>Origin of the node, for example from the graph, XLA, or AMP</td>
</tr>
<tr>
<td>GPU Time (ns)</td>
<td>The total GPU time for all instances of the node</td>
</tr>
<tr>
<td>Uses TC</td>
<td>True if the kernel uses Tensor Cores</td>
</tr>
<tr>
<td>Total kernel Count</td>
<td>The total number of unique kernels executed by the node</td>
</tr>
<tr>
<td>Kernel Count Using Tensor Cores</td>
<td>The total number of unique kernels that use Tensor Cores for this node</td>
</tr>
<tr>
<td>Kernel Names Using Tensor Cores</td>
<td>A list of all the names of kernels using Tensor Cores for this node</td>
</tr>
<tr>
<td>Kernel Count Not Using Tensor Cores</td>
<td>The total number of unique kernels that do not use Tensor Cores for this node</td>
</tr>
<tr>
<td>Kernel Names Not Using Tensor Cores</td>
<td>A list of all the names of kernels are not using Tensor Cores for this node</td>
</tr>
</tbody>
</table>
8.7. 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 number of unique nodes executing this operation type</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>
<tr>
<td>Column name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Min CPU Overhead Time (ns)</td>
<td>The minimum CPU overhead found amongst all instances of the node</td>
</tr>
<tr>
<td>Max CPU Overhead Time (ns)</td>
<td>The maximum CPU overhead found amongst all instances of the node</td>
</tr>
<tr>
<td>Total GPU Idle Time (ns)</td>
<td>The total GPU idle time of all instances of the node</td>
</tr>
<tr>
<td>Avg. GPU Idle Time (ns)</td>
<td>The average GPU idle time of all instances of the node</td>
</tr>
<tr>
<td>Min GPU Idle Time (ns)</td>
<td>The minimum GPU idle time found amongst all instances of the node</td>
</tr>
<tr>
<td>Max GPU Idle Time (ns)</td>
<td>The maximum GPU idle time found amongst all instances of the node</td>
</tr>
</tbody>
</table>

### 8.8. 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>
<tr>
<td>Column Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Avg. GPU Time (ns)</td>
<td>The average GPU time for all instances of the sub-tree on a per-iteration basis</td>
</tr>
<tr>
<td>Min GPU Time (ns)</td>
<td>The minimum GPU time for all instances of the sub-tree on a per-iteration basis</td>
</tr>
<tr>
<td>Max GPU Time (ns)</td>
<td>The maximum GPU time for all instances of the sub-tree on a per-iteration basis</td>
</tr>
<tr>
<td>Total CPU Overhead Time (ns)</td>
<td>The total CPU overhead time for all instances of the sub-tree</td>
</tr>
<tr>
<td>Avg. CPU Overhead Time (ns)</td>
<td>The average CPU overhead time for all instances of the sub-tree on a per-iteration basis</td>
</tr>
<tr>
<td>Min CPU Overhead Time (ns)</td>
<td>The minimum CPU overhead time for all instances of the sub-tree on a per-iteration basis</td>
</tr>
<tr>
<td>Max CPU Overhead Time (ns)</td>
<td>The maximum CPU overhead time for all instances of the sub-tree on a per-iteration basis</td>
</tr>
<tr>
<td>Total GPU Idle Time (ns)</td>
<td>The total GPU idle time for all instances of the sub-tree</td>
</tr>
<tr>
<td>Avg. GPU Idle Time (ns)</td>
<td>The average GPU idle time for all instances of the sub-tree on a per-iteration basis</td>
</tr>
<tr>
<td>Min GPU Idle Time (ns)</td>
<td>The minimum GPU idle time for all instances of the sub-tree on a per-iteration basis</td>
</tr>
<tr>
<td>Max GPU Idle Time (ns)</td>
<td>The maximum GPU idle time for all instances of the sub-tree on a per-iteration basis</td>
</tr>
</tbody>
</table>

### 8.9. Expert Systems Report

The Expert Systems report lists detected problems and gives 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>
</tbody>
</table>
8.10. 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 the user to take 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>Not NHWC Detector</td>
<td>Detects the case where the model does not use NHWC format and as a result many kernels are used to convert to that format.</td>
</tr>
<tr>
<td>No Fusion Detector</td>
<td>Detects the case where fusion is disabled.</td>
</tr>
</tbody>
</table>

8.11. XLA Support

DLProf is able to profile models that have enabled XLA. In XLA, new XLA optimized nodes are created that replace the originally created nodes. Reports generated by DLprof display the optimized nodes and correctly aggregates all profile data for those new optimized nodes.
When profiling any computer program, the objective is to inspect its code to determine if
effectiveness 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.

9.1. 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.

9.2. 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 out if operations that are running in floating point 32 mode can be switched
to floating point 16. Use Automatic Mixed Precision to automatically change operations
to use mixed precision operations wherever safe. By optimizing the model to use Tensor
Cores, you will speed up the performance of training.
Chapter 10.
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.

10.1. 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.

10.1.1. 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 --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:20.03-tf1-py3
```

10.1.2. 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. Adding --graphdef=auto will generate graphdef file automatically and tensorboard will be able to show the graphs plugin.

```bash
$ cd /workspace/nvidia-examples/resnet50v1.5
$ mkdir results
$ dlprof --reports=summary,detail,iteration --iter_start 20 --iter_stop 80 --graphdef=auto /usr/bin/python main.py
```

```bash
--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
```
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
- `dlprof_iteration.csv` - The detailed iteration report

### 10.1.3. 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.

### 10.2. MobileNet

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

#### 10.2.1. 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](http://image-net.org/download)

2. Run the NGC TensorFlow docker container, and map the training data and a result data folder

   ```bash
   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:20.03-tf1-py3
   ```

3. In the docker container, install the TensorFlow benchmarks into `/workspace`

   ```bash
   mkdir /workspace/tensorflow-examples && \
   cd /workspace/tensorflow-examples && \
   git clone https://github.com/tensorflow/models.git && \
   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
   ```

#### 10.2.2. Profiling MobileNet

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

```
dlprof \ 
   --key_node=tower_0/v/add /usr/bin/python tf_cnn_benchmarks.py \ 
   --num_gpus=1 --batch_size=256 --model=mobilenet --device=gpu --gpu_indices=1 \ 
   --data_name=imagenet --data_dir=/data/train-val-tfrecord-480 \ 
   --num_batches=50 --use_fp16 --fp16_enable_auto_loss_scale
```
The only output will be the TensorBoard events which can be found in:
/workspace/tensorflow-examples/benchmarks/scripts/tf_cnn_benchmarks/event_files

10.2.3. 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.
11.1. 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 12.
REFERENCE

The following section contains additional reference material.

12.1. 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

12.2. Command Line Options

DLProf command line options
<table>
<thead>
<tr>
<th>Options</th>
<th>Possible Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--help, -h</td>
<td></td>
<td>Print this help message</td>
</tr>
<tr>
<td>--version, -v</td>
<td></td>
<td>Display version</td>
</tr>
<tr>
<td>--mode</td>
<td>tensorflow1, pytorch</td>
<td>Specify the framework for the network being profiled.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Follows the single option format.</td>
</tr>
<tr>
<td>--force, -f</td>
<td>true</td>
<td>Overwrite existing output files</td>
</tr>
<tr>
<td>--verbosity</td>
<td>error, warning, info, verbose</td>
<td>Level of logging output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Follows the single option format.</td>
</tr>
<tr>
<td><strong>Nsight Systems Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--nsys_database</td>
<td>&lt; filename&gt;</td>
<td>Path and filename for the Nsight Systems generated SQLite DB</td>
</tr>
<tr>
<td>--nsys_base_name</td>
<td>&lt; name &gt;</td>
<td>Base name for all Nsight Systems output files</td>
</tr>
<tr>
<td>--nsys_opts</td>
<td>&lt; options &gt;</td>
<td>Custom Nsight Systems profile command line options</td>
</tr>
<tr>
<td><strong>Profiling Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--graphdef, -g</td>
<td>auto, &lt;path to graphef file/directory&gt;</td>
<td>Path to GraphDef file/directory or 'auto' to autogenerate.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Follows the single option format.</td>
</tr>
<tr>
<td>--detailed_mode</td>
<td>true, false</td>
<td>Enable/disable detailed NVTX information (default: true).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Follows the single option format.</td>
</tr>
<tr>
<td>--key_node</td>
<td>&lt; key node name &gt;</td>
<td>Set the node that specifies the iteration intervals.</td>
</tr>
<tr>
<td>--iter_start</td>
<td>&lt; number &gt;</td>
<td>Iteration that all report aggregation will begin.</td>
</tr>
<tr>
<td>--iter_stop</td>
<td>&lt; number &gt;</td>
<td>Iteration that all report aggregation will end.</td>
</tr>
<tr>
<td>--delay</td>
<td>&lt; number &gt;</td>
<td>Delay (in seconds) before starting to profile.</td>
</tr>
<tr>
<td>--duration</td>
<td>&lt; number &gt;</td>
<td>Duration (in seconds) to profile.</td>
</tr>
<tr>
<td><strong>Output Options</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--output_path</td>
<td>&lt; path &gt;</td>
<td>Specify the output path for all profile collateral.</td>
</tr>
<tr>
<td>--file_formats</td>
<td>csv, json</td>
<td>File output format options.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Follows the single option format.</td>
</tr>
<tr>
<td>--report_base_name</td>
<td>&lt; base name &gt;</td>
<td>Base name prepended to all generated report file names.</td>
</tr>
<tr>
<td></td>
<td>Default: dlprof</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>--reports</td>
<td>Generate various reports. Follows the multiple option format.</td>
<td></td>
</tr>
<tr>
<td>--domains</td>
<td>Generate reports for the specified domains. Follows the multiple option format.</td>
<td></td>
</tr>
<tr>
<td>--dump_model_data</td>
<td>Creates a json file containing the raw, correlated model data.</td>
<td></td>
</tr>
<tr>
<td>--out_tb_dir, -b</td>
<td>Set output directory for TensorBoard files.</td>
<td></td>
</tr>
<tr>
<td>--suppress_tb_files</td>
<td>Suppress TensorBoard output files.</td>
<td></td>
</tr>
</tbody>
</table>
Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS GUIDE IS PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do 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 guide. NVIDIA does not accept any 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 guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, cuDNN, DALI, DIGITS, DGX, DGX-1, DGX-2, DGX Station, DLProf, Jetson, Kepler, Maxwell, NCCL, Nsight Compute, Nsight Systems, NvCaffe, PerfWorks, Pascal, SDK Manager, Tegra, TensorRT, TensorRT Inference Server, Tesla, TF-TRT, and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2020 NVIDIA Corporation. All rights reserved.

www.nvidia.com