PyProf: Automating End-to-End PyTorch Profiling

https://github.com/dlacceleration/pyprof

Aditya Agrawal†, Marek Kolodziej††

† Now at Google       †† Now at Uber ATG

# Work done at Nvidia
Acknowledgements

- Michael Carilli
- Alex Settle
- Carl Case
- Natalia Gimelshein
- Bryan Catanzaro
- Jie Jiang
- Andrew Huang
- Sandeep Behera
- Kevin Stephano
- other early adopters.
Aditya is a computer architect and Deep Learning performance engineer. He analyzes and optimizes Deep Learning network performance on a variety of frameworks (PyTorch, TensorFlow etc.) and architectures (GPU, TPU etc.). He was part of the MLPerf team at Nvidia.

Marek is a Tech Lead Manager for GPU Systems on Uber ATG’s Autonomy Team. He has a decade of experience as a machine learning engineer, accelerating distributed algorithms on heterogeneous clusters. While at Nvidia, he optimized deep learning framework backends (TF, MXNet, PyTorch) for training and inference on platforms ranging from data center (Tesla) to embedded (Tegra).
Outline

• Motivation & Tool Introduction.
• Basic usage.
• Advanced usage.
• Demo.
Challenges we faced as DL analysts

Start by reading a N page paper. If we are lucky,

- There is a block diagram with layer attributes, tensor shapes and datatype.
- The implementation is the same as the description.
- The network does not use other networks as submodules.

Current profilers e.g. NVprof and NSight Systems provide no information about

- Layer parameters, tensor shapes, data types.
- Call stack i.e. file name, line number.
- Direction e.g. fprop, bprop, loss, optimizer.
- Flops, bytes, tensor core usage per kernel.
What does a DL analyst want?

For any network, quickly obtain a table like this:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Direction</th>
<th>Call Trace</th>
<th>Op</th>
<th>Parameters</th>
<th>Kernel</th>
<th>Silicon Time</th>
<th>Thread Id</th>
<th>Device Id</th>
<th>Stream Id</th>
<th>Grid Dim</th>
<th>Block Dim</th>
<th>Flops</th>
<th>Bytes</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SelfAttention</td>
<td>fprop</td>
<td>attn.py: 23, ...</td>
<td>Linear</td>
<td>MNK, fp16</td>
<td>volta_s884...</td>
<td>200</td>
<td>23</td>
<td>1</td>
<td>7</td>
<td>x,y,z</td>
<td>x,y,z</td>
<td>1000</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>Block_1a</td>
<td>fprop</td>
<td>block.py: 43,</td>
<td>Conv</td>
<td>NCHWKPQRS, fp32</td>
<td>cudnn_...</td>
<td>130</td>
<td>23</td>
<td>1</td>
<td>7</td>
<td>x,y,z</td>
<td>x,y,z</td>
<td>2000</td>
<td>400</td>
<td>1</td>
</tr>
<tr>
<td>Hadamard</td>
<td>fprop</td>
<td>net.py: 73, ...</td>
<td>mul</td>
<td>T=(128,256), fp16</td>
<td>pointwise...</td>
<td>110</td>
<td>23</td>
<td>1</td>
<td>7</td>
<td>x,y,z</td>
<td>x,y,z</td>
<td>10</td>
<td>2000</td>
<td>0</td>
</tr>
</tbody>
</table>

- Available from NVprof / Nsight Systems.
- PyProf: Intercept PyTorch calls and obtain the call trace, op and parameters.
- PyProf: Calculate direction, flops, bytes and Tensor Core (TC) usage.
- PyProf: User annotation (optional).
<table>
<thead>
<tr>
<th>Layer</th>
<th>Direction</th>
<th>Op</th>
<th>Params</th>
<th>Kernel</th>
<th>S1(ns)</th>
<th>FLOPs</th>
<th>Bytes</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1_x</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=23, C=3,H=224,W=224,K=64,P=112,Q=112,R=7,7,S=1,cudnn:gemm::computeOffsKern...</td>
<td>2176</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>conv1_x</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=23, C=3,H=224,W=224,K=64,P=112,Q=112,R=7,7,S=1,volta_fp16_scdnn_fp16_128x64_re...</td>
<td>668991</td>
<td>7552892928</td>
<td>61032832</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>conv1_x</td>
<td>fprop</td>
<td><em>add</em></td>
<td>T=[(1),(1),f32],elementwise_kernel</td>
<td>2848</td>
<td>1</td>
<td>8</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>conv1_x</td>
<td>fprop</td>
<td>batch_norm</td>
<td>T=[(32,64,112,112),f32],batch_norm_collect_statistics_kern...</td>
<td>424863</td>
<td>205520896</td>
<td>205520896</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>conv1_x</td>
<td>fprop</td>
<td>batch_norm</td>
<td>T=[(32,112,112),f32],batch_norm_transform_input_kern...</td>
<td>1151360</td>
<td>205520896</td>
<td>205520896</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>conv1_x</td>
<td>fprop</td>
<td>relu</td>
<td>T=[(32,64,112,112),f32],elementwise_kernel</td>
<td>1174048</td>
<td>25690112</td>
<td>12760448</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>conv1_x</td>
<td>fprop</td>
<td>max_pool2d</td>
<td>T=[(32,64,112,112),float16],max_pool_forward_nchw</td>
<td>189216</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Conv1</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=32,C=64,H=65,W=65,K=64,P=56,Q=56,R=1,S=1,ph,cudnn:gemm::computeOffsKern...</td>
<td>1472</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Conv1</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=32,C=64,H=65,W=65,K=64,P=56,Q=56,R=1,S=1,ph,volta_fp16_s884cudnn_fp16_256x128_ldg...</td>
<td>85344</td>
<td>822083584</td>
<td>25698304</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>BN1</td>
<td>fprop</td>
<td><em>add</em></td>
<td>T=[(1),(1),f32],elementwise_kernel</td>
<td>1760</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>BN1</td>
<td>fprop</td>
<td>batch_norm</td>
<td>T=[(32,64,56,56),f32],batch_norm_collect_statistics_kern...</td>
<td>125984</td>
<td>51380224</td>
<td>51380224</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>BN1</td>
<td>fprop</td>
<td>batch_norm</td>
<td>T=[(32,64,56,56),f32],batch_norm_transform_input_kern...</td>
<td>39328</td>
<td>51380224</td>
<td>51380224</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>RelU</td>
<td>fprop</td>
<td>relu</td>
<td>T=[(32,64,56,56),f32],elementwise_kernel</td>
<td>44736</td>
<td>6422528</td>
<td>25690112</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Conv2</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=32,C=64,H=65,W=65,K=64,P=56,Q=56,R=1,S=3,ph,cudnn:gemm::computeOffsKern...</td>
<td>37664</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Conv2</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=32,C=64,H=65,W=65,K=64,P=56,Q=56,R=1,S=3,ph,cudnn:gemm::computeOffsKern...</td>
<td>2784</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Conv2</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=32,C=64,H=65,W=65,K=64,P=56,Q=56,R=1,S=3,ph,volta_hmmaImplicit_gemm_fprop_ft32_nhwc...</td>
<td>147967</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Conv2</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=32,C=64,H=65,W=65,K=64,P=56,Q=56,R=1,S=3,ph,cudnn:gemm::computeOffsKern...</td>
<td>36192</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>BN2</td>
<td>fprop</td>
<td><em>add</em></td>
<td>T=[(1),(1),f32],elementwise_kernel</td>
<td>1760</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>BN2</td>
<td>fprop</td>
<td>batch_norm</td>
<td>T=[(32,64,56,56),f32],batch_norm_collect_statistics_kern...</td>
<td>124384</td>
<td>51380224</td>
<td>51380224</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>BN2</td>
<td>fprop</td>
<td>batch_norm</td>
<td>T=[(32,64,56,56),f32],batch_norm_transform_input_kern...</td>
<td>39136</td>
<td>51380224</td>
<td>51380224</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>RelU</td>
<td>fprop</td>
<td>relu</td>
<td>T=[(32,64,56,56),f32],elementwise_kernel</td>
<td>44928</td>
<td>6422528</td>
<td>25690112</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Conv3</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=32,C=64,H=65,W=65,K=256,P=56,Q=56,R=1,S=1,ph,cudnn:gemm::computeOffsKern...</td>
<td>1856</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Conv3</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=32,C=64,H=65,W=65,K=256,P=56,Q=56,R=1,S=1,ph,volta_fp16_s884cudnn_fp16_256x128_ldg...</td>
<td>158656</td>
<td>3288334336</td>
<td>64258048</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>BN3</td>
<td>fprop</td>
<td><em>add</em></td>
<td>T=[(1),(1),f32],elementwise_kernel</td>
<td>1760</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>BN3</td>
<td>fprop</td>
<td>batch_norm</td>
<td>T=[(32,256,56,56),f32],batch_norm_collect_statistics_kern...</td>
<td>163328</td>
<td>205520896</td>
<td>205520896</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>BN3</td>
<td>fprop</td>
<td>batch_norm</td>
<td>T=[(32,256,56,56),f32],batch_norm_transform_input_kern...</td>
<td>155488</td>
<td>205520896</td>
<td>205520896</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Downsampler</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=32,C=64,H=56,W=56,K=256,P=56,Q=56,R=1,S=1,ph,cudnn:gemm::computeOffsKern...</td>
<td>1472</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Downsampler</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=32,C=64,H=56,W=56,K=256,P=56,Q=56,R=1,S=1,ph,volta_fp16_s884cudnn_fp16_256x128_ldg...</td>
<td>157888</td>
<td>3288334336</td>
<td>64258048</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Downsampler</td>
<td>fprop</td>
<td><em>add</em></td>
<td>T=[(1),(1),f32],elementwise_kernel</td>
<td>1760</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Downsampler</td>
<td>fprop</td>
<td>batch_norm</td>
<td>T=[(32,256,56,56),f32],batch_norm_collect_statistics_kern...</td>
<td>163520</td>
<td>205520896</td>
<td>205520896</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Downsampler</td>
<td>fprop</td>
<td>batch_norm</td>
<td>T=[(32,256,56,56),f32],batch_norm_transform_input_kern...</td>
<td>153952</td>
<td>205520896</td>
<td>205520896</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Residual</td>
<td>fprop</td>
<td><em>add</em></td>
<td>T=[(32,256,56),(32,256,56),f32],elementwise_kernel</td>
<td>202143</td>
<td>38535168</td>
<td>154140672</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>RelU</td>
<td>fprop</td>
<td>relu</td>
<td>T=[(32,256,56,56),f32],elementwise_kernel</td>
<td>173536</td>
<td>25690112</td>
<td>102760448</td>
</tr>
<tr>
<td>conv2_x</td>
<td>Bottleneck</td>
<td>Conv1</td>
<td>fprop</td>
<td>conv2d</td>
<td>N=32,C=256,H=56,W=56,K=64,P=56,Q=56,R=1,S=1,ph,cudnn:gemm::computeOffsKern...</td>
<td>1472</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Salient Features

• Network: Analyze any network e.g. Torchvision, MLPerf, BERT, GPT, Waveglow, Tacotron2.
• Fast: Analyze any network in 10 min e.g. entire Transformer inference with ~ 200,000 kernels.
• Coverage: Supports lot of layers e.g. Conv, GEMM, Pointwise, Reduction, Loss, Optimizer etc.
• Low effort: About 5 lines of instrumentation.
• Plug & Play: No changes to PyTorch.
Testimonials

“For me PyProf was the fastest way to analyze the sequence of GPU kernels that get executed during beam search. The availability of high level information, such as GEMM dimensions made it much easier to understand what was going on.”

-- User 1

“I used PyProf to profile GPT2 in a single GPU system. It took less than 10 minutes to set up and provided deep insights such as what layers are launched, what are the compute bottlenecks and most importantly program trace of the specific performance-limiting kernel. Thanks, PyProf team!”

-- User 2
Testimonials

“Deep learning models like Transformer language translation or BERT language models can have on the order of **800 to 1000 kernels** in a training step. While there is a repetitive pattern to the kernels, you likely have to be an expert in cuDNN, cuBLAS, and PyTorch kernel naming conventions to decipher the difference in kernels over such a large pool of kernels. PyProf, out-of-the-box, allows the model writer to see kernel times in the context of their model giving them better **instant feedback** on hot spots in their model that they otherwise might ignore given the high bar of analysis effort. Having done the analysis with and without PyProf, I saw my **time commitment shrink** from a day or more to more like an hour or two!”

-- User 3

“PyProf **reduced the time** it takes to analyze our neural network workloads by an **order of magnitude**. Its **modular software design** has allowed us to integrate it to our workflow, as a result it had a positive impact for **several teams** at once.”

-- User 4
Basic Usage
Basic Usage

- Take any off the shelf PyTorch network.
- Add ~ 5 lines of instrumentation code.
- Run NVprof/Nsight Systems to generate a SQL database.
- Extract layer name, call trace, direction, operator, kernel name, tensor dims & type, silicon time etc.
- Use the operator, tensor dimensions and type to calculate flops and bytes per kernel.
PyProf: Components and Flow

- **import pyprof**: Intercept all PyTorch, custom functions and modules.
- Run NVprof/NSight Systems to obtain a SQL database.
- **parse.py**: Extract information from the SQL database.
- **prof.py**: Use this information to calculate flops and bytes.
Code Instrumentation

# examples/simple.py

```python
import torch
import torch.cuda.profiler as profiler  # Import CUDA profiler
import pyprof
pyprof.init()

with torch.autograd.profiler.emit_nvtx():  # Enable PyTorch NVTX
    for epoch in range(100):
        for iteration in range(100):
            if (epoch == 0 and iteration == 20):
                profiler.start()  # Start profiler (optional)

            ...  # Code...

            if (epoch == 0 and iteration == 25):
                profiler.stop()  # Stop profiler (optional)
```

March 25, 2020
# If you did not use profiler start/stop

```bash
$ nvprof
   -f
   -o net.sql
   python net.py
```

# Overwrite existing file
# Create net.sql

# If you used profiler start/stop

```bash
$ nvprof
   -f
   -o net.sql
   --profile-from-start off
   python net.py
```

# Profiling start/stop inside net.py
NSight Systems

$ nsys profile
  -f true
  -o net
  -c cudaProfilerApi
  -s none
  --stop-on-range-end true
  --export sqlite
  python net.py

# Overwrite existing files
# Create net.qdrep (used by Nsys)
# Control profile start/stop, like NVprof
# Don’t sample CPU (otherwise very slow)
# Export net.sql (similar to NVprof)
$ parse/pwe.py net.sql > net.dict

For each GPU kernel extract

<table>
<thead>
<tr>
<th>Tool</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVProf / NSight</td>
<td>Kernel Name</td>
<td>elementwise_kernel</td>
</tr>
<tr>
<td></td>
<td>Duration</td>
<td>44736 ns</td>
</tr>
<tr>
<td></td>
<td>Grid and block dimensions</td>
<td>(160,1,1) (128,1,1)</td>
</tr>
<tr>
<td></td>
<td>Thread Id, Device Id, Stream Id</td>
<td>23, 0, 7</td>
</tr>
<tr>
<td>+ PyProf</td>
<td>Call stack</td>
<td>resnet.py:210, resnet.py:168</td>
</tr>
<tr>
<td></td>
<td>Layer name</td>
<td>Conv2_x:Bottleneck_1:ReLU</td>
</tr>
<tr>
<td></td>
<td>Operator</td>
<td>ReLU</td>
</tr>
<tr>
<td></td>
<td>Tensor Shapes</td>
<td>[32, 64, 56, 56]</td>
</tr>
<tr>
<td></td>
<td>Datatype</td>
<td>fp16</td>
</tr>
</tbody>
</table>
Get Flops, Bytes & TC Usage

$ prof/prof.py --csv net.dict  # CSV output
$ prof/prof.py net.dict        # Space separated output
$ prof/prof.py -w 150 net.dict # Columnated output with width 150
$ prof/prof.py -c op,kernel,sil net.dict  # Space separated output with 3 cols

In addition to the previous information, for every GPU kernel obtain

- Direction (fprop, bprop).
- Flops and bytes.
- Tensor Core Usage.
Advanced Usage

(Optional)
Advanced Usage

- Layer annotation.
- Custom functions and modules.
- Extensibility.
# examples/user_annotation/resnet.py
# Use the “layer:” prefix

class Bottleneck(nn.Module):
    def forward(self, x):
        nvtx.range_push("layer:Bottleneck_{id}".format(self.id))  # NVTX push marker.

        nvtx.range_push("layer:Conv1")  # Nested NVTX push/pop markers.
        out = self.conv1(x)
        nvtx.range_pop()

        nvtx.range_push("layer:BN1")  # Use the “layer:” prefix.
        out = self.bn1(out)
        nvtx.range_pop()

        nvtx.range_push("layer:ReLU")
        out = self.relu(out)
        nvtx.range_pop()

        ...  # NVTX pop marker.

        return out
Custom Function

```python
# examples/custom_func_module/custom_function.py

import torch
import pyprof
pyprof.init()

class Foo(torch.autograd.Function):
    @staticmethod
def forward(ctx, in1, in2):
        out = in1 + in2
        return out
    @staticmethod
def backward(ctx, grad):
        in1_grad, in2_grad = grad, grad
        return in1_grad, in2_grad

# Hook the forward and backward functions to pyprof.
pyprof.wrap(Foo, 'forward')
pyprof.wrap(Foo, 'backward')
```

# This could be a custom C++ function.
# This could be a custom C++ function.

March 25, 2020
# examples/custom_func_module/custom_module.py

```python
import torch
import pyprof
pyprof.init()

class Foo(torch.nn.Module):
    def __init__(self, size):
        super(Foo, self).__init__()
        self.n = torch.nn.Parameter(torch.ones(size))
        self.m = torch.nn.Parameter(torch.ones(size))

    def forward(self, input):
        return self.n * input + self.m  # This could be a custom C++ function.

# Hook the forward function to pyprof.
pyprof.wrap(Foo, 'forward')
```

Custom Module
Extensibility

• For custom functions and modules, users can add flops and bytes calculation.
• Python code is easy to extend – no need to recompile, no need to change the PyTorch backend and resolve merge conflicts on every version upgrade.
Actionable Items

• NvProf / Nsight Systems tell us what the hotspots are, but not if we can act on them.

• If a kernel runs close to max perf based on FLOPs and bytes (and maximum FLOPs and bandwidth of the GPU), then there’s no point in optimizing it even if it’s a hotspot.

• If the ideal timing based on FLOPs and bytes (max(compute_time, bandwidth_time)) is much shorter than the silicon time, there’s scope for improvement.

• Tensor Core usage (conv): for Volta, convolutions should have the input channel count (C) and the output channel count (K) divisible by 8, in order to use tensor cores. For Turing, it’s optimal for C and K to be divisible by 16.

• Tensor core usage (GEMM): M, N and K divisible by 8 (Volta) or 16 (Turing)

We presented PyProf, a tool which automates end-to-end kernel-level neural network analysis for PyTorch.

Adding ~5 lines of code generates layer type, dimensions, data type, direction, layer parameters, CUDA launch information, kernel duration, FLOPs and bandwidth.

The tool is really easy to use and extend.

From weeks to minutes to actionable insights.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Direction</th>
<th>Call Trace</th>
<th>Op</th>
<th>Parameters</th>
<th>Kernel</th>
<th>Silicon Time</th>
<th>Thread Id</th>
<th>Device Id</th>
<th>Stream Id</th>
<th>Grid Dim</th>
<th>Block Dim</th>
<th>Flops</th>
<th>Bytes</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self Attention</td>
<td>fprop</td>
<td>attn.py: 23, ...</td>
<td>Linear</td>
<td>MNK, fp16</td>
<td>volta_s884...</td>
<td>200</td>
<td>23</td>
<td>1</td>
<td>7</td>
<td>x,y,z</td>
<td>x,y,z</td>
<td>1000</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>Block_1a</td>
<td>fprop</td>
<td>block.py: 43, Conv</td>
<td>NCHWKPSQR, fp32</td>
<td>cudnn_...</td>
<td></td>
<td>130</td>
<td>23</td>
<td>1</td>
<td>7</td>
<td>x,y,z</td>
<td>x,y,z</td>
<td>2000</td>
<td>400</td>
<td>1</td>
</tr>
<tr>
<td>Hadamard</td>
<td>fprop</td>
<td>net.py: 73, ...</td>
<td>mul</td>
<td>T=(128,256), fp16</td>
<td>pointwise...</td>
<td>110</td>
<td>23</td>
<td>1</td>
<td>7</td>
<td>x,y,z</td>
<td>x,y,z</td>
<td>10</td>
<td>2000</td>
<td>0</td>
</tr>
</tbody>
</table>
Repository Note

• The original code for PyProf used to be in Apex: https://github.com/NVIDIA/apex/tree/master/apex/pyprof

• We created a new repo to rapidly iterate over new features (e.g. Nsight Systems support). The latest code can be found at: https://github.com/dlacceleration/pyprof

• NVIDIA is planning to create a new home for PyProf. Our repo will point to it once it goes live.
Contact: Questions & Contributions

GitHub: https://github.com/dlacceleration/pyprof

Aditya: aditya.iitb@gmail.com, https://github.com/adityaiitb

Marek: mkolod@gmail.com, https://github.com/mkolod