GPU PROFILING 기법을 통한 DEEP LEARNING 성능 최적화 기법 소개

Gwangsoo Hong, Solution Architect, ghong@nvidia.com
AGENDA

Introduction to Nsight Systems
Profiling from CLI
NVTX (NVIDIA Tools Extension)
Deep learning Optimization
Getting Started Resources
INTRODUCTION TO NSIGHT SYSTEMS
HOW TO SPEED-UP NETWORK

- Use the latest GPU and more GPU?
- Wait NVIDIA to release the new GPU or newer library?
- Optimize Neural Network Computing or Something
  - Need to understand your network operation

We are talking about this
PROFILING NEURAL NETWORK

Profiling with cProfiler + Snakeviz

I've found this helpful when profiling pytorch code: jiffyclub.github.io/snakeviz/
It takes cProfile outputs and gives much nicer viz

TENSORFLOW PROFILER
https://github.com/tensorflow/profiler-ui

- Show timeline and can trace network
- Still difficult to understand
NVIDIA PROFILING TOOLS

Nsight Systems
Nsight Compute

Nsight Visual Studio Edition
Nsight Eclipse Edition

NVIDIA Profiler (nvprof)
NVIDIA Visual Profiler (nvvp)

CUPTI (CUDA Profiling Tools Interface)
NSIGHT PRODUCT FAMILY

- Standalone Performance Tools
  - Nsight Systems  system-wide application algorithm tuning
  - Nsight Compute  Debug/optimize specific CUDA kernel
  - Nsight Graphics  Debug/optimize specific graphics
- IDE plugins
  - Nsight Eclipse Edicion/Visual Studio  editor, debugger, some perf analysis

Workflow

Nsight Systems

Nsight Compute

Nsight Graphics
Overview

- Profile **System-wide** application
  - Multi-process tree, GPU workload trace, etc
- Investigate your workload across multiple CPUs and GPUs
  - CPU algorithms, utilization, and thread states
  - GPU streams kernels, memory transfers, etc
  - NVTX, CUDA & Library API, etc
- Ready for Big Data
  - docker, user privilege (linux), cli, etc
Processes and threads

CUDA and OpenGL API trace

cuDNN and cuBLAS trace

Kernel and memory transfer activities

Multi-GPU

Thread/core migration

Thread state
NVTX Tracing
TRANSITIONING TO PROFILE A KERNEL

Dive into kernel analysis
NVIDIA NSIGHT COMPUTE
Next Generation Kernel Profiler

- Interactive CUDA API debugging and kernel profiling
- Fast Data Collection
- Graphical and multiple kernel comparison reports
- Improved Workflow and Fully Customizable (Baselining, Programmable UI/Rules)
- Command Line, Standalone, IDE Integration
- Platform Support
  - OS: Linux(x86, ARM), Windows, OSX (host only)
  - GPUs: Pascal, Volta, Turing
PROFILING GPU APPLICATION
Focusing GPU Computing

How to measure

- GPU Profiling
- CPU/GPU Tracing
- Application Tracing

Low GPU Utilization
- Low SM Efficiency
- Low Achieved Occupancy
- Memory Bottleneck
- Instructions Bottleneck

- Too few threads
- Register limit
- Large shared memory
- Cache misses
- Bandwidth limit
- Access pattern
- Arithmetic
- Control flow

NVIDIA (Visual) Profiler / Nsight Compute

NVIDIA Supports them with cuDNN, cuBLAS, and so on
PROFILING GPU APPLICATION
Focusing System Operation

How to measure

- GPU Profiling
- CPU/GPU Tracing
- Application Tracing

Low GPU Utilization

- Low SM Efficiency
- Low Achieved Occupancy

- Memory Bottleneck
- Instructions Bottleneck

CPU-Only Activities

Memcopy Latency

Kernel Launch Latency

- CPU Computation
- I/O

Job Startup / Checkpoints

Nsight System / Application Tracing
NSIGHT SYSTEMS PROFILE

Profile with CLI

APIs to be traced

$ nsys profile -t cuda,osrt,nvtx,cudnn,cublas \
  -o baseline.qd strm -w true python main.py

Name of output file
Show output on console
Application command

Automatic conversion of .qd strm temp results file to .qd rep format if converter utility is available.

CUDA - GPU kernel
osrt - OS runtime
nvtx - NVIDIA Tools Extension
cudnn - CUDA Deep NN library
cublas - CUDA BLAS library

https://docs.nvidia.com/nsight-systems/#nsight_systems/2019.3.6-x86/06-cli-profiling.htm
NSIGHT SYSTEMS PROFILE

No NVTX

- Difficult to understand → no useful
NVTX (NVIDIA TOOLS EXTENSION)
NVTX ANNOTATIONS

```python
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
```
import torch.cuda.nvtx as nvtx

def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        nvtx.range_push("Batch " + str(batch_idx))
        nvtx.range_push("Copy to device")
        data, target = data.to(device), target.to(device)
        nvtx.range_pop()
        nvtx.range_push("Forward pass")
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        nvtx.range_pop()
        nvtx.range_pop()
from cupy.cuda import nvtx

def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        nvtx.RangePush("Batch " + str(batch_idx))
        nvtx.RangePush("Copy to device")
        data, target = data.to(device), target.to(device)
        nvtx.RangePop()
        nvtx.RangePush("Forward pass")
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        nvtx.RangePop()
        nvtx.RangePop()
NSIGHT SYSTEM PROFILE

NVTX range marker tip

- NVTX for data loading (data augmentation)

```python
for batch_idx, (data, target) in enumerate(train_loader):
    nvtx.range_push('Data loading')
    (data, target) = train_loader.next()
    nvtx.range_pop()
```
BASELINE PROFILE

- MNIST Training: 89 sec, <5% utilization
- CPU waits on a semaphore and starves the GPU!
GPU is idle during **data loading**

Data is loaded using a single thread. This starves the GPU!
OPTIMIZE SOURCE CODE

- Data loader was configured to use 1 worker thread

  \[\text{kwargs} = \{'\text{num\_workers}' : 1, '\text{pin\_memory}' : \text{True} \text{ if use\_cuda else } \{}\]\n
- Let’s switch to using 8 worker threads:

  \[\text{kwargs} = \{'\text{num\_workers}' : 8, '\text{pin\_memory}' : \text{True} \text{ if use\_cuda else } \{}\]
AFTER OPTIMIZATION

- Time for data loading reduced for each bath

Reduced from 5.1ms to 60us for batch 50
DEEP LEARNING OPTIMIZATION
OPTIMIZATION STRATEGY FOR DL

- **Algorithm optimization**
  - **Tensor Cores**
    - Training: Automatic Mixed Precision
    - Inference: TensorRT

- **Data pipeline optimization**
  - **DALI (Data loading Library)**

![Diagram](image)

- Available in **Volta** and **Turing** architecture GPUs
- 125 Tflops in FP16 vs. 15.7 Tflops in FP32 (**8x** speed-up)
- Optimized **4x4x4** dot operation (GEMM)
TRAINING OPTIMIZATION CASE
NVTX TAGGING
BERT in PyTorch

```python
loss = model(input_ids, segment_ids, input_mask, start_positions, end_positions)

...

if args.fp16:
    optimizer.backward(loss)
else:
    loss.backward()
if (step + 1) % args.gradient_accumulation_steps == 0:
    if args.fp16:
        # modify learning rate with special warm up BERT uses
        # if args.fp16 is False, BertAdam is used and handles this automatically
        lr_this_step = args.learning_rate * warmup_linear.get_lr(global_step, args.warmup_proportion)
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr_this_step
    optimizer.step()
    optimizer.zero_grad()
    global_step += 1
```
NVTX TAGGING
BERT in PyTorch

```python
nvtx.range_push("Batch " + str(step))
nvtx.range_push("Forward pass")
loss = model(input_ids, segment_ids, input_mask, start_positions, end_positions)
nvtx.range_pop()

...  
nvtx.range_push("Backward pass")
if args.fp16:
    optimizer.backward(loss)
else:
    loss.backward()
if (step + 1) % args.gradient_accumulation_steps == 0:
    if args.fp16:
        # modify learning rate with special warm up BERT uses
        # if args.fp16 is False, BertAdam is used and handles this automatically
        lr_this_step = args.learning_rate * warmup_linear.get_lr(global_step, args.warmup_proportion)
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr_this_step
    optimizer.step()
    optimizer.zero_grad()
    global_step += 1
nvtx.range_pop()
nvtx.range_pop()
```

forward pass

backward pass

Batch %d
ALGORITHM OPTIMIZATION - TRAINING

Single Precision (FP32) Training for BERT

Getting API List

~60% GEMM

~460 ms

https://github.com/NVIDIA/apex
ALGORITHM OPTIMIZATION - TRAINING

Automatic Mixed Precision (FP32 + FP16) Training for BERT

- 2.1x Speed up (~460ms vs. ~222ms)

- \texttt{volta_fp16_s884gemm} indicates for using the tensor cores.

\url{https://github.com/NVIDIA/apex}
ALGORITHM OPTIMIZATION - TRAINING

APEX ADAM Optimizer Optimization

- ADAM optimizer
  - Low Utilization

- APEX ADAM optimizer
  - High utilization

https://github.com/NVIDIA/apex

132 ms

8.5 ms
INFERENTIAL OPTIMIZATION CASE
ALGORITHM OPTIMIZATION - INFERENCE

TensorFlow Single Precision (FP32) vs. half Precision (FP16)

- **2.4x** Speed up (~27.0ms vs. ~11.0ms)

Single Precision (FP32)

Tensor Cores APIs

Half Precision (FP16)

27.0 ms

11.0 ms
ALGORITHM OPTIMIZATION - INFERENCE

TensorRT Single Precision (FP32) vs. Half Precision (FP16)

- **5.3x** Speed up (~6.4ms vs. ~1.2ms)

Single Precision (FP32)

Half Precision (FP16)

Tensor Cores APIs

224x224, Resnet 50, bathsize 128, V100 16G 1EA in trtexec
DATA PIPELINE OPTIMIZATION

Naïve data augmentation pipeline

FileReader → RandomResizedCrop → RandomHorizontalFlip → Normalize

ImageNet, Resnet 50, batchsize 256, V100 32G 1EA in Pytorch

3.3 ms
DATA PIPELINE OPTIMIZATION

DALI (Data Loading Library)

FileReader → JPEGDecoder → RandomResizedCrop → RandomHorizontalFlip → Normalize

4.7x Speed up (3.3ms vs. ~0.7ms)

ImageNet, ResNet 50, batchsize 256, V100 32G 1EA in Pytorch
OPTIMIZATION STRATEGY FOR DL

Reminding

- Algorithm optimization
  - Tensor Cores
    - Training: Automatic Mixed Precision
    - Inference: TensorRT

- Data pipeline optimization
  - DALI (Data loading Library)

\[ D = \begin{bmatrix}
A_{1,1} & A_{1,2} & A_{1,3} \\
A_{2,1} & A_{2,2} & A_{2,3} \\
A_{3,1} & A_{3,2} & A_{3,3}
\end{bmatrix}
+ \begin{bmatrix}
B_{1,1} & B_{1,2} & B_{1,3} \\
B_{2,1} & B_{2,2} & B_{2,3} \\
B_{3,1} & B_{3,2} & B_{3,3}
\end{bmatrix}
\]

- Available in Volta and Turing architecture GPUs
- 125 Tflops in FP16 vs. 15.7 Tflops in FP32 (8x speed-up)
- Optimized 4x4x4 dot operation (GEMM)

DALI example pipeline
GETTING STARTED RESOURCES
LEARN MORE

• Nsight System
  • https://developer.nvidia.com/nsight-systems

• Official Documentation (Nsight System developer guide)
  • https://docs.nvidia.com/nsight-systems/

• Nsight System Blog
  • https://devblogs.nvidia.com/nsight-systems-exposes-gpu-optimization/
LEARN MORE DURING AI CONFERENCE

- 17:20 ~ 18:00 Track2
  - 효율적인 Deep Learning 서비스 구축을 위한 핵심 애플리케이션 - NVIDIA TensorRT Inference Server by NVIDIA 정소영 상무
RELATED SESSIONS

Automatic Mixed Precision (AMP) for training optimization
- 13:00 - 13:40 Track1
  - Tensor Core를 이용한 딥러닝 학습 가속을 쉽게 하는 방법
    (Getting more DL Training Acceleration using Tensor Cores and AMP) by NVIDIA 한재근 과장

TensorRT for inference optimization
- 13:50 ~ 14:30 Track2
  - Deep Learning inference 가속화를 위한 NVIDIA의 기술 소개 by NVIDIA 이종환 과장
- 14:40 ~ 15:20 Track2
  - TensorRT를 이용한 OCR Model Inference 성능 최적화 by KAKAO 이현수

DALI for data pipeline optimization
- 15:40 ~ 16:20 Track1
  - GPU를 활용한 Image Augmentation 가속화 방안 - DALI by NVIDIA 한재근 과장
DEEP LEARNING PROFILER WORKFLOW

INPUT

PROFILE

CORRELATE

OUTPUT

ANALYZE

Graghdef file
generate in TensorFlow

use NSight
tools to gather
kernel and timing
profile data

Correlate
profile data
with TensorFlow model

Generate TensorFlow
event files
and detailed
reports

Analyze in
TensorBoard
or other 3rd
party tools
ARCHITECTURE

- Automates workflow
- Nsight Systems
  - Gather timeline information
  - Determines Tensor Core usage from name of kernels
- Nsight Compute
  - Detailed kernel level profiling
  - Determines Tensor Core usage from GPU program counters
- Use NVTX markers to correlate kernels with DNN graph nodes
- Any number of reports can be generated
  - TB event Files, CSV, JSON
  - Analyze with tool of your choice
DEEP LEARNING PROFILER

Command Line Example

- Example command to profile mobileNet V2 and generate a graphdef

```
$ /usr/bin/python tf_cnn_benchmarks.py --num_gpus=1 --batch_size=8 --model=mobilenet --device=gpu --gpu_indices=1 --data_name=imagenet --data_dir=/data/train-val-tfrecord-480 --num_batches=1 --use_fp16 --fp16_enable_auto_loss_scale --graph_file=/results/mobilenet_graph.pb
```

- Example Deep Learning Profiler command

```
$ dlprof --in_graphdef=/results/mobilenet_graph.pb/usr/bin/python tf_cnn_benchmarks.py --num_gpus=1 --batch_size=8 --model=mobilenet --device=gpu --gpu_indices=1 --data_name=imagenet --data_dir=/data/train-val-tfrecord-480 --num_batches=1 --use_fp16 --fp16_enable_auto_loss_scale
```

- Launching TensorBoard

```
$ tensorboard --logdir ./event_files
```
TENSORBOARD MODIFICATIONS

Start TensorBoard with NVIDIA modifications
COMPATIBILITY DETAILS

Select Compatible using Tensor Cores
COMPATIBILITY DETAILS

Select Compatible using Tensor Cores

Compatibility details and panel providing guidance and links to help with mixed precision.

Tensor Core Help:
- Please note that if there are multiple kernels being observed on single node, there are likely performing data transposes to prepare the data for efficient use by tensorscores. Such transposes themselves would not use tensorscores.
- To learn more about Tensor cores and Mixed Precision training, visit this site: https://developer.nvidia.com/tensorcores.
- You can find resources on how to train networks with mixed precision and make full use of Tensor cores for Tensorflow models at: https://docs.nvidia.com/tensorcore/tensorcore-training/index.html#training_tensortoolflow
- Visit this site to find out more about CNN examples optimized and tuned for Tensor cores provided by NVIDIA: https://developer.nvidia.com/deep-learning-examples.
OPNODES SUMMARY TAB

GPU Summary tab showing all the Nodes, compatible and using Tensor Cores
GROUP NODE SUMMARY TAB

Roll up timing metrics and Tensor Core utilization per group node
MODEL SUMMARY TABLE

Model Summary shows concise information on Tensor core usage

<table>
<thead>
<tr>
<th>Model Summary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes in Graph</td>
<td>6158</td>
</tr>
<tr>
<td>Compatible Nodes</td>
<td>112</td>
</tr>
<tr>
<td>Compatible Nodes using Tensor Cores</td>
<td>97</td>
</tr>
<tr>
<td>Total GPU Time</td>
<td>15.10 (s)</td>
</tr>
<tr>
<td>% of time in TC compatible nodes</td>
<td>15.62%</td>
</tr>
</tbody>
</table>