INTEGRATION OF DALI WITH TENSORRT ON XAVIER

Josh Park (joshp@nvidia.com), Manager - Automotive Deep Learning Solutions Architect at NVIDIA
Anurag Dixit(anuragd@nvidia.com), Deep Learning SW Engineer at NVIDIA
Backgrounds
## Backgrounds

### Massive amount of computation in DNN

<table>
<thead>
<tr>
<th>layer name</th>
<th>output size</th>
<th>18-layer</th>
<th>34-layer</th>
<th>50-layer</th>
<th>101-layer</th>
<th>152-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>$112 \times 112$</td>
<td>$3 \times 3$ max pool, stride 2</td>
<td>$7 \times 7$, $64$, stride 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv2, x</td>
<td>$56 \times 56$</td>
<td>$3 \times 3$, $64$</td>
<td>$3 \times 3$, $64$</td>
<td>$1 \times 1$, $64$</td>
<td>$3 \times 3$, $64$</td>
<td>$1 \times 1$, $64$</td>
</tr>
<tr>
<td>conv3, x</td>
<td>$28 \times 28$</td>
<td>$3 \times 3$, $128$</td>
<td>$3 \times 3$, $128$</td>
<td>$1 \times 1$, $128$</td>
<td>$3 \times 3$, $128$</td>
<td>$1 \times 1$, $128$</td>
</tr>
<tr>
<td>conv4, x</td>
<td>$14 \times 14$</td>
<td>$3 \times 3$, $256$</td>
<td>$3 \times 3$, $256$</td>
<td>$1 \times 1$, $256$</td>
<td>$3 \times 3$, $256$</td>
<td>$1 \times 1$, $256$</td>
</tr>
<tr>
<td>conv5, x</td>
<td>$7 \times 7$</td>
<td>$3 \times 3$, $512$</td>
<td>$3 \times 3$, $512$</td>
<td>$1 \times 1$, $512$</td>
<td>$3 \times 3$, $512$</td>
<td>$1 \times 1$, $512$</td>
</tr>
<tr>
<td>FLOPs</td>
<td>$1.8 \times 10^9$</td>
<td>$3.6 \times 10^9$</td>
<td>$3.8 \times 10^9$</td>
<td>$7.6 \times 10^9$</td>
<td>$1.13 \times 10^9$</td>
<td>$3 \times 10^9$</td>
</tr>
</tbody>
</table>

### GPU: High Performance Computing Platform

<table>
<thead>
<tr>
<th>Tesla V100 PCIe</th>
<th>Tesla V100 SXM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Architecture</td>
<td>NVIDIA Volta</td>
</tr>
<tr>
<td>NVIDIA Tensor Cores</td>
<td>640</td>
</tr>
<tr>
<td>NVIDIA CUDA Cores</td>
<td>5,120</td>
</tr>
<tr>
<td>Double-Precision Performance</td>
<td>7 TFLOPS</td>
</tr>
<tr>
<td>Single-Precision Performance</td>
<td>14 TFLOPS</td>
</tr>
<tr>
<td>Tensor Performance</td>
<td>112 TFLOPS</td>
</tr>
<tr>
<td>GPU Memory</td>
<td>16 GB HBM2</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>900 GB/sec</td>
</tr>
<tr>
<td>ECC</td>
<td>Yes</td>
</tr>
<tr>
<td>Interconnect Bandwidth*</td>
<td>32 GB/sec</td>
</tr>
<tr>
<td>System Interface</td>
<td>PCIe Gen3</td>
</tr>
<tr>
<td>Form Factor</td>
<td>PCIe Full Height/Length</td>
</tr>
<tr>
<td>Max Power Consumption</td>
<td>250 W</td>
</tr>
<tr>
<td>Thermal Solution</td>
<td>Passive</td>
</tr>
<tr>
<td>Compute APIs</td>
<td>CUDA, DirectCompute, OpenCL*, OpenACC</td>
</tr>
</tbody>
</table>

### SW Libraries

- **TensorRT**
- **DALI**
- **cuDNN**
- **CUDA**
- **CUDA Driver**
- **OS**
- **HW with GPUs**

---

NVIDIA DRIVE AGX Platform

Xavier - aarch64 based on SoC w/ CPU + GPU + MEM

iGPU
8 Volta SMs
512 CUDA cores
64 Tensor Cores
20 TOPS INT8, 10 TOPS FP16
CUDA Compute Capability 7.2
NVIDIA TensorRT
NVIDIA TensorRT - Programmable Inference Accelerator

- Optimize and Deploy neural networks in production environments
- Maximize throughput for latency critical apps with optimizer and runtime
- Deploy responsive and memory efficient apps with INT8 & FP16 optimizations
- Accelerate every framework with TensorFlow integration and ONNX support
- Run multiple models on a node with containerized inference server
TensorRT 5 supports Turing GPUs

- Optimized kernels for mixed precision (FP32, FP16, INT8) workloads on Turing GPUs
- Control precision per-layer with new APIs
- Optimizations for depth-wise convolution operation
How TensorRT Works?

- Layer & Tensor Fusion
- Auto-Tuning
- Precision Calibration
- Multi-Stream Execution
- Dynamic Tensor Memory
Layer & Tensor Fusion

Unoptimized Network

<table>
<thead>
<tr>
<th>Networks</th>
<th>Number of layers (Before)</th>
<th>Number of layers (After)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG19</td>
<td>43</td>
<td>27</td>
</tr>
<tr>
<td>Inception v3</td>
<td>309</td>
<td>113</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>670</td>
<td>159</td>
</tr>
</tbody>
</table>

TensorRT Optimized Network

e.g.
Kernel Auto-Tuning

- Maximize kernel performance
- Select the best performance for target GPU
- Parameters
  - Input data size
  - Batch
  - Tensor layout
  - Input dimension
  - Memory
  - Etc.
Lower precision - FP16

- FP16 matches the results quite closely to FP32
- TensorRT automatically converts FP32 weights to FP16 weights

```cpp
builder->setFp16Mode(true);
```

- To enforce that 16-bit kernels will be used when building the engine

```cpp
builder->setStrictTypeConstraints(true);
```

- Tensor Core kernels (HMMA) for FP16 (supported on Volta and Turing GPUs)
Lower Precision - INT8 Quantization

- Setting the builder flag enables INT8 precision inference.
  - builder->setInt8Mode(true);
  - IInt8Calibrator* calibrator;
  - builder->setInt8Calibrator(calibrator);

- Quantization of FP32 weights and activation tensors
  - **weights**
    - \( \text{Int8\_weight} = \text{ROUND\_To\_Nearest} \left( \text{scaling\_factor} \times \text{FP32\_weight\_in\_the\_filters} \right) \)
    - \( \text{scaling\_factor} = 127.0 \, f / \max ( | \text{all\_FP32\_weights} | ) \)
  - **activation**
    - \( \text{Int8\_value} = \begin{cases} \text{threshold} & \text{if} \ (\text{value} > \text{threshold}) \\ \text{scaling\_factor} \times \text{FP32\_value} & \text{else} \end{cases} \)
    - * Activation range unknown (input dependent) => calibration is needed

- Dynamic range of each activation tensor => the appropriate quantization scale
- TensorRT: symmetric quantization with quantization scale calculated using absolute maximum dynamic range values
- Control precision per-layer with new APIs
- Tensor Core kernel (IMMA) for INT8 (supported on Drive AGX Xavier iGPU and Turing GPUs)
Lower Precision - INT8 Calibration

- Calibration Solutions in TensorRT
  - Run FP32 inference on Calibration
  - Per Layer:
    - Histograms of activations
    - Quantized distributions with different saturation thresholds.
  - Two ways to set saturation thresholds (dynamic ranges):
    - manually set the dynamic range for each network tensor using `setDynamicRange` API
      - * Currently, only symmetric ranges are supported
    - use INT8 calibration to generate per tensor dynamic range using the calibration dataset (i.e. ‘representative’ dataset)
      - *pick threshold which minimizes KL_divergence (entropy method)

* INT8 and FP16 mode, both if the platform supports. TensorRT will choose the most performance optimal kernel to perform inference.
Plugin for Custom OPs in TensorRT 5

- Custom op/layer: op/layer not supported by TensorRT => need to implement plugin for TensorRT engine

- Plugin Registry
  - stores a pointer to all the registered Plugin Creators / look up a specific Plugin Creator
  - Built-in plugins: RPROI_TRT, Normalize_TRT, PriorBox_TRT, GridAnchor_TRT, NMS_TRT, LReLU_TRT, Reorg_TRT, Region_TRT, Clip_TRT

- Register a plugin by calling REGISTER_TENSORRT_PLUGIN(pluginCreator) which statically registers the Plugin Creator to the Plugin Registry
How can we further optimize end-to-end inference pipeline on NVIDIA DRIVE Xavier?
NVIDIA DALI
Motivation: CPU BOTTLENECK OF DL TRAINING

CPU ops and CPU to GPU ratio

- Operations are performed mainly on CPUs before the input data is ready for inference/training
- Half precision arithmetic, multi-GPU, dense systems are now common (e.g., DGX1V, DGX2)
- Can’t easily scale CPU cores (expensive, technically challenging)
- Falling CPU to GPU ratio:
  - DGX1: 40 cores, 8 GPUs, 5 cores/ GPU
  - DGX2: 48 cores↑, 16 GPUs↑ 3 cores/ GPU↓

Complexity of I/O pipeline
Data Loading Library (DALI)

High Performance Data Processing Library

“Originally on X86_64”
Why DALI?

- Running DNN models requires input data pre-processing
- Pre-processing involves
  - Decoding, Resize, Crop, Spatial augmentation, Format conversions (NCHW ↔ NHWC)
- DALI supports
  - the feature to accelerate pre-processing on GPUs
  - configurable graphs and custom operators
  - multiple input formats (e.g. JPEG, LMDB, RecordIO, TFRecord)
  - serializing a whole graph (portable graph)
- Easily integrates with framework plugins and open source bindings
Integration: Our Effort on DALI

Extension to aarch64 and Inference engine

Beyond x86_64

- Extension of targeted platform to "aarch64": Drive AGX Platform

High level TensorRT runtime within DALI

- TensorRTInfer op via a plugin
## Dependency

<table>
<thead>
<tr>
<th>Components</th>
<th>On x86_64</th>
<th>On aarch64</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>4.9.2 or later</td>
<td>5.4</td>
</tr>
<tr>
<td>Boost</td>
<td>1.66 or later</td>
<td>N/A</td>
</tr>
<tr>
<td>Nvidia CUDA</td>
<td>9.0 or later</td>
<td>10.0 or later</td>
</tr>
<tr>
<td>protobuf</td>
<td>version 2.0 or later</td>
<td>version 2.0</td>
</tr>
<tr>
<td>cmake</td>
<td>3.5 or later</td>
<td>3.5 later</td>
</tr>
<tr>
<td>libnvjpeg</td>
<td>Included in cuda toolkit</td>
<td>Included in cuda toolkit</td>
</tr>
<tr>
<td>opencv</td>
<td>version 3.4 (recommended)</td>
<td>version 3.4</td>
</tr>
<tr>
<td></td>
<td>2.x (unofficial)</td>
<td></td>
</tr>
<tr>
<td>TensorRT</td>
<td>5.0 / 5.1</td>
<td>5.0 / 5.1</td>
</tr>
</tbody>
</table>
How we Integrate TensorRT with DALI?

- DALI supports custom operator in C++
- Custom operator library can be loaded in the runtime
- TensorRT inference is treated as a custom operator
- TensorRT Infer schema
  - serialized engine
  - TensorRT plugins
  - input/output binding names
  - batch size for inference
Pipeline Example of TensorRT within DALI

Newly accelerated nodes in an end-to-end inference pipeline on GPU

1. Image Decoder
2. Resize
3. NormalizePermute
4. TensorRTInfer

Decoded image → Resized Image → Normalized Image
Use Cases

Single Input, Multi Outputs

Multi Inputs, Multi Outputs

Multi Inputs, Multi Output with Post processing

iGPU + DLA pipeline
Parallel Inference Pipeline

SSD Object Detection (iGPU)

DeepLab Segmentation (DLA)

Input

Output

iGPU + DLA pipeline

Input

Pre-process

TensorRTInfer (iGPU)

TensorRTInfer (DLA)

Post-process

Output
Performance
Object Detection Model on DALI

- Model Name: SSD (Backbone ResNet18)
- Input Resolution: 3x1024x1024
- Batch: 1
- HW Platform: TensorRT Inference on Xavier (iGPU)
- OS: QNX 7.0
- CUDA: 10.0
- cuDNN: 7.3.0
- TensorRT: 5.1.1
- Preprocessing: jpeg decoding, resizing, normalizing

**DALI Pipeline**

```
CPU Preprocessing

Host Decoder -> Decoded image

Resize -> Resized Image

NormalizePermute -> Normalized Image

TensorRTInfer

GPU Preprocessing

Host Decoder -> Decoded image

Resize -> Resized Image

NormalizePermute -> Normalized Image

TensorRTInfer
```
Performance of DALI + TensorRT on Xavier

TensorRT Speedup per Precision (resnet-18)

Preprocessing Speedup via DALI
Stay Tuned!

NVIDIA DALI github: https://github.com/NVIDIA/DALI

[PR] Extend DALI for aarch64 platform: https://github.com/NVIDIA/DALI/pull/522
Acknowledgement

Special Thanks to

- NVIDIA DALI Team
  - @Janusz Lisiecki, @Przemek Tredak, @Joaquin Anton Guirao, @Michal Zientkiewicz

- NVIDIA TSE/ADLSA
  - @Muni Anda, @Joohoon Lee, @Naren Sivagnanadasan, @Le An, @Jeff Hetherly, @Yu-Te Cheng

- NVIDIA Developer Marketing
  - @Siddartha Sharma
Thank You