S9925: FAST AI DATA PREPROCESSING WITH NVIDIA DALI

Janusz Lisiecki, Michał Zientkiewicz, 2019-03-18
S9925: FAST AI DATA PRE-PROCESSING WITH NVIDIA DALI

Janusz Lisiecki, Michał Zientkiewicz, 2019-03-18
THE PROBLEM
CPU BOTTLENECK OF DL TRAINING

CPU : GPU ratio

Half precision arithmetic, multi-GPU, dense systems are now common (DGX1V, DGX2)

Can’t easily scale CPU cores (expensive, technically challenging)

Falling CPU to GPU ratio:

- DGX1V: 40 cores, 8 GPUs, 5 cores/ GPU
- DGX2: 48 cores, 16 GPUs, 3 cores/ GPU
CPU BOTTLENECK OF DL TRAINING

Complexity of I/O pipeline

AlexNet
2012
256x256 image

ResNet 50
2015
480p image

Random resize
Color augment
224x224 crop and mirror
Training

224x224 crop and mirror
Training
CPU BOTTLENECK OF DL TRAINING

In practice

When we put 2x GPU we don’t get adequate perf improvement

Training speed, ResNet50, MXNet container 18.09, b=256

Goal: 2x

<table>
<thead>
<tr>
<th>[img/s]</th>
<th>8GPU</th>
<th>16GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DGX1V32G</td>
<td>DGX2</td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Higher is better
CPU BOTTLENECK OF DL TRAINING

In practice

When we put 2x GPU we don’t get adequate perf improvement

Training speed, ResNet50, MXNet container 18.09, b=256

Goal: 2x

Reality: < 2x

![Graph showing performance comparison between 8GPU and 16GPU across two models, DGX1V32G and DGX2.]
DALI TO THE RESCUE
WHAT IS DALI?
High Performance Data Processing Library
DALI RESULTS

RN50 MXNet

Training speed, ResNet50, MXNet container, b=256

- Native pipeline 19.02

<table>
<thead>
<tr>
<th>GPU Configuration</th>
<th>img/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGX1V32G (8 GPUs)</td>
<td>8</td>
</tr>
<tr>
<td>DGX2 (16 GPUs)</td>
<td>16</td>
</tr>
</tbody>
</table>

Higher is better
DALI RESULTS

RN50 MXNet

Training speed, ResNet50, MXNet container, b=256

- Native pipeline 19.02
- DALI 0.3 + 18.09
- DALI 0.7 + 19.02

Higher is better

![Graph showing performance comparison between different configurations.](image-url)
DALI RESULTS

RN50 PyTorch

Training speed, ResNet50, PyTorch container, b=256

- Native pipeline 19.02
- DALI 0.7 + 19.02

Higher is better

<table>
<thead>
<tr>
<th>GPU Configuration</th>
<th>DGX1V32G</th>
<th>DGX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 GPU</td>
<td>6000</td>
<td>14000</td>
</tr>
<tr>
<td>16 GPU</td>
<td>6000</td>
<td>14000</td>
</tr>
</tbody>
</table>
DALI RESULTS

RN50 TensorFlow

Training speed, ResNet50, TensorFlow container, b=256

- Native pipeline 19.02
- DALI 0.7 + 19.02

![Bar chart showing performance comparison between Native pipeline and DALI for 8 and 16 GPUs on DGX1V32G and DGX2. Higher is better.](chart.png)
DALI RESULTS - MLPERF

Perfect scaling

Training time - ResNet50 [minutes]
- DGX1V
- DGX2
- DGX2-H

Training time - SSD [minutes]
- DGX1V
- DGX2
- DGX2-H

https://mlperf.org/results
INSIDE DALI
DALI: CURRENT ARCHITECTURE

Loader → Decode → Images → Resize → Augment → Support OP → Training

Data → Decode → Images → Resize → Augment → Labels

CPU → Mixed → GPU
HOW TO USE DALI

Define Graph

Instantiate operators

```python
def __init__(self, batch_size, num_threads, device_id):
    super(SimplePipeline, self).__init__(batch_size, num_threads, device_id)
    self.input = ops.FileReader(file_root = image_dir)
    self.decode = ops.nvJPEGDecoder(device = "mixed", output_type = types.RGB)
    self.resize = ops.Resize(device = "gpu", resize_x = 224, resize_y = 224)
```

Define graph in imperative way

```python
def define_graph(self):
    jpegs, labels = self.input()
    images = self.decode(jpegs)
    images = self.resize(images)
    return (images, labels)
```

Use it

```python
pipe.build()
images, labels = pipe.run()
```
HOW TO USE DALI

Define Graph

Instantiate operators

```python
def __init__(self, batch_size, num_threads, device_id):
    super(SimplePipeline, self).__init__(batch_size, num_threads, device_id)
    self.input = ops.FileReader(file_root = image_dir)
    self.decode = ops.nvJPEGDecoder(device = "mixed", output_type = types.RGB)
    self.resize = ops.Resize(device = "gpu", resize_x = 224, resize_y = 224)
```

Define graph in imperative way

```python
def define_graph(self):
    jpegs, labels = self.input()
    images = self.decode(jpegs)
    images = self.resize(images)
    return (images, labels)
```

Use it

```python
pipe.build()
images, labels = pipe.run()
```
How to Use DALI

Define Graph

Instantiate operators

```python
def __init__(self, batch_size, num_threads, device_id):
    super(SimplePipeline, self).__init__(batch_size, num_threads, device_id)
    self.input = ops.FileReader(file_root = image_dir)
    self.decode = ops.nvJPEGDecoder(device = "mixed", output_type = types.RGB)
    self.resize = ops.Resize(device = "gpu", resize_x = 224, resize_y = 224)
```

Define graph in imperative way

```python
def define_graph(self):
    jpegs, labels = self.input()
    images = self.decode(jpegs)
    images = self.resize(images)
    return (images, labels)
```

Use it

```python
pipe.build()
images, labels = pipe.run()
```
HOW TO USE DALI

Define Graph

Instantiate operators

```python
def __init__(self, batch_size, num_threads, device_id):
    super(SimplePipeline, self).__init__(batch_size, num_threads, device_id)
    self.input = ops.FileReader(file_root = image_dir)
    self.decode = ops.nvJPEGDecoder(device = "mixed", output_type = types.RGB)
    self.resize = ops.Resize(device = "gpu", resize_x = 224, resize_y = 224)
```

Define graph in imperative way

```python
def define_graph(self):
    jpegs, labels = self.input()
    images = self.decode(jpegs)
    images = self.resize(images)
    return (images, labels)
```

Use it

```python
pipe.build()
images, labels = pipe.run()
```
HOW TO USE DALI

Use in PyTorch

**PyTorch DataLoader**

```python
train_loader = torch.utils.data.DataLoader(...)
prefetcher = data_prefetcher(train_loader)
input, target = prefetcher.next()
i = -1
while input is not None:
    i += 1
    (...)
    input, target = prefetcher.next()
```

**DALI iterator**

```python
dali_pipe = TrainPipe(...)
train_loader = DALIClassificationIterator(dali_pipe)

for i, data in enumerate(train_loader):
    input = data[0]["data"]
    (...)
    target = data[0]["label"]).squeeze()
    (...)
```
HOW TO USE DALI

Use in MXNet

**MXNet Dataloader and DataBatch**

```python
train_data = SyntheticDataIter(...)

for i, batches in enumerate(train_data):
    data = [b.data[0] for b in batches]
    label = [b.label[0].as_in_context(b.data[0].context)
for b in batches]
    (....)
```

**DALI iterator**

```python
dali_pipes = [TrainPipe(...) for gpu_id in gpus]
train_data = DALIClassificationIterator(dali_pipe)

for i, batches in enumerate(train_data):
    data = [b.data[0] for b in batches]
    label = [b.label[0].as_in_context(b.data[0].context)
for b in batches]
    (....)
```
**How to Use DALI**

**Use in TensorFlow**

---

**TensorFlow Dataset**

```python
def get_data():
    ds = tf.data.Dataset.from_tensor_slices(files)
    ds.define_operations(...)
    return ds
```

```
classifier.train(input_fn=get_data, ...)
```

---

**DALI TensorFlow operator**

```python
def get_data():
    dali_pipe = TrainPipe(...)
    daliop = dali_tf.DALIIterator()
    with tf.device("/gpu:0"):
        img, labels = daliop(pipeline=dali_pipe, ...)
    return img, labels
```

```
classifier.train(input_fn=get_data, ...)
```
NEW USE CASES
Use operators in the DALI graph:

images = self.paste(images, paste_x = px, paste_y = py, ratio = ratio)
bboxes = self.bbpaste(bboxes, paste_x = px, paste_y = py, ratio = ratio)
crop_begin, crop_size, bboxes, labels = self.prospective_crop(bboxes, labels)
images = self.slice(images, crop_begin, crop_size)
images = self.flip(images, horizontal = rng, vertical = rng2)
bboxes = self.bbflip(bboxes, horizontal = rng, vertical = rng2)
return (images, bboxes, labels)
VIDEO

Video Pipeline Example

Instantiate operator:

```
self.input = ops.VideoReader(device="gpu", filenames=data, sequence_length=len)
```

Use it in the DALI graph:

```
frames = self.input(name="Reader")
output_frames = self.Crop(frames)
return output_frames
```
VIDEO
Optical Flow Example

Instantiate operator:

```python
self.input = ops.VideoReader(file_root = video_files, sequence_length = len, step = step)
self.opticalFlow = ops.OpticalFlow()
self.takeFirst = ops.ElementExtract(element_map = [0])
```

Use it in the DALI graph:

```python
frames = self.input()
flow = self.opticalFlow(frames)
first = self.takeFirst(frames)
return first, flow
```
MAKING LIFE EASIER
MORE EXAMPLES
Help you get started

ResNet50 for PyTorch, MXNet, TensorFlow
How to read data in various frameworks
How to create custom operators
Pipeline for the detection
Video pipeline
More to come...

Documentation available online:
PLUGIN MANAGER
Adds Extensibility

Create operator

```cpp
template<>
void Dummy<GPUBackend>::RunImpl(DeviceWorkspace *ws, const int idx) {
  (...)
}
DALI_REGISTER_OPERATOR(CustomDummy, Dummy<GPUBackend>, GPU);
```

Load Plugin from python

```python
import nvidia.dali.plugin_manager as plugin_manager
plugin_manager.load_library('./customdummy/build/libcustomdummy.so')
ops.CustomDummy(...)
```
CHALLENGES
CHALLENGES

Object Detection

Data-dependent random transformation

Random crop

[Valid transformation example]

[Invalid transformation example]
CHALLENGES
Object Detection

More types of data, not only images and labels - bounding boxes as well

Previously only images were processed

Now processing of bounding boxes drives image processing
CHALLENGES

Video

Integrated NVDEC to utilize H.264 and HEVC

Samples are no longer single image - sequence (NFHWC<->NCFHW)

Reuse operators - flatten the sequence
CHALLENGES

CPU/GPU high or network traffic consumes GPU cycles

• CPU operators coverage

Sweet spot for SSD mixed pipeline - part CPU, part GPU

• Test what works best for you
CHALLENGES

Memory Consumption

DGX - “works for me”

A lot of non-DGX users started using DALI

• Want to use CPU operators
• Memory consumption on the CPU side matters
• Usability more important than speed
CHALLENGES

Memory Consumption

Multiple buffering

...but memory consumption

- Caching allocators?
- Subbatches?
CHALLENGES

Decoding Time

Significant image decoding time

- CPU decoding already pushed to the limits

Can we do better?

- nvJPEG - huge improvement
- ROI decoding
CHALLENGES

TensorFlow Forward Compatibility

PyTorch and MXNet integration

- Python API - “easy-peasy”

TensorFlow - custom operator needed

- Frequent changes to TensorFlow C++ API
- Cannot preserve forward compatibility at the binary level
- DALI TF plug-in package is now available - compile your TensorFlow DALI op
CHALLENGES
Discrepancies Between Frameworks

Bilinear filter - OpenCV vs Pillow

Bicubic filter - TensorFlow vs Pillow

https://hackernoon.com/how-tensorflows-tf-image-resize-stole-60-days-of-my-life-aba5eb093135
CHALLENGES

Discrepancies Between Frameworks

- MXNet is based on OpenCV
- PyTorch uses Pillow
- TensorFlow has its own augmentation operators

We want portability between frameworks, but what about pre-trained models?

https://github.com/python-pillow/Pillow/issues/2718
NEXT STEPS
NEW USE CASES

Medical imaging (Volumetric data)
• Performant 3D augmentations library

Segmentation?
NEW USE CASES

Extract augmentation operators in a separate library

- Inference - the same augmentation operation can be used in custom inference pipeline where full feature DALI is not required (i.e. embedded platform)
- Ability to use operator directly from Python code

```python
import nvidia.dali.standaloneOps as standaloneOps
import cv2

image = cv2.imread('test.jpg', 0)
standaloneOps.Rotate(image, device="gpu", angle=45, interp_type = types.INTERP_LINEAR)
cv2.imwrite("./img_tf.png", image)
```
DALI

Summary

- Open source, GPU-accelerated data augmentation and image loading library
  - Over 1100 GitHub stars
    - 1) Top 50 ML/DL Projects (out of 22,000 in 2018)
- Full pre-processing data pipeline ready for training and inference
- Easy framework integration
- Portable training workflows

1) https://github.com/Mybridge/amazing-machine-learning-opensource-2019
More questions? Connect with Experts Sessions: DALI Tue 19th, Wed 20th, 2pm (Expo Hall)

Meet us P9291 - Fast Data Pre-processing with DALI (Mon 18th, 6-8pm)

Attend S9818 - TensorRT with DALI on Xavier to learn about TensorRT inference workflow with DALI graphs and customer operators

We want to hear from you

Dali-Team@nvidia.com
https://github.com/NVIDIA/DALI
https://developer.nvidia.com/dali
ACKNOWLEDGEMENTS

Joaquin Anton
Trevor Gale
Andrei Ivanov
Simon Layton
Krzysztof Łęcki
Serge Panev
Michał Szołucha
Przemek Trędak
Albert Wolant
Pablo Ribalta
Cliff Woolley
DL Frameworks @ NVIDIA