THE PROBLEM
CPU BOTTLENECK OF DL TRAINING

CPU : GPU ratio

- Multi-GPU, dense systems are more common (DGX-1V, DGX-2)

- Using more cores / sockets is very expensive

- CPU to GPU ratio becomes lower:
  - DGX-1V: 40 cores / 8, 5 cores / GPU
  - DGX-2: 48 cores / 16, 3 cores / GPU

The diagram shows the CPU-Computing perf trend over years.
CPU BOTTLENECK OF DL TRAINING

Complexity of I/O pipeline

Alexnet

256x256 image → 224x224 crop and mirror → Training

ResNet 50

480p image → Random resize → Color augment → 224x224 crop and mirror → Training
CPU BOTTLENECK OF DL TRAINING

Increased complexity of CPU-based I/O pipeline

Higher GPU to CPU ratio

Throughput vs. Time

GPU

CPU
Frameworks have their own I/O pipelines (often more than 1!)
Lots of duplicated effort to optimize them all
Training process is not portable even if the model is (e.g. via ONNX)
LOTS OF FRAMEWORKS
Lots of effort

Optimized I/O pipelines are not flexible and often unsuitable for research

```python
train = mx.io.ImageRecordIter(
    path_imgrec = args.data_train,
    path_imgidx = args.data_train_idx,
    label_width = 1,
    mean_r = rgb_mean[0],
    mean_g = rgb_mean[1],
    mean_b = rgb_mean[2],
    data_name = 'data',
    label_name = 'softmax_label',
    data_shape = image_shape,
    batch_size = 128,
    rand_crop = True,
    max_random_scale = 1,
    pad = 0,
    fill_value = 127,
    min_random_scale = 0.533,
    max_aspect_ratio = args.max_random_aspect_ratio,
    random_h = args.max_random_h,
    random_s = args.max_random_s,
    random_l = args.max_random_l,
    max_rotate_angle = args.max_random_rotate_angle,
    max_shear_ratio = args.max_random_shear_ratio,
    rand_mirror = args.random_mirror,
    preprocess_threads = args.data_nthreads,
    shuffle = True,
    num_parts = 0,
    part_index = 1)
```

image, _ = mx.image.random_size_crop(image, (data_shape, data_shape), 0.08, (3/4., 4/3.))
image = mx.nd.image.random_flip_left_right(image)
image = mx.nd.image.to_tensor(image)
image = mx.nd.image.normalize(image, mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
return mx.nd.cast(image, dtype), label

Inflexible  fast  flexible  slow
SOLUTION: ONE LIBRARY

- Centralize the effort
- Integrate into all frameworks
- Provide both flexibility and performance
DALI: OVERVIEW
DALI

- Flexible, high-performance image data pipeline
- Python / C++ frontends with C++ / CUDA backend
- Minimal (or no) changes to the frameworks required
- Full pipeline - from disk to GPU, ready to train
- OSS (soon)
Data pipeline is just a (simple) graph
GPU OPTIMIZED PRIMITIVES

High performance, GPU optimized implementations
GPU ACCELERATED JPEG DECODE
DALI with nvJPEG

Hybrid approach to JPEG decoding - can move fully to GPU in the future
SET YOUR DATA FREE

DALI

LMDB (Caffe, Caffe2)
RecordIO (MXNet)
TFRecord (TensorFlow)
List of JPEGs (PyTorch, others)

Use any file format in any framework
BEHIND THE SCENES: PIPELINE
One pipeline per GPU
The same logic for multithreaded and multiprocess frameworks
PIPELINE
Overview

Single direction
3 stages
CPU -> Mixed -> GPU
PIPELINE
Overview

Simple scheduling of operations
Operations processed per-sample in a thread pool
Batched processing of data
A bridge between CPU and GPU
Per-sample input, batched output
Used also for batching CPU data (for CPU outputs of the pipeline)
EXECUTOR

Pipelining the pipeline

CPU, Mixed and GPU stages need to be executed serially

But each batch of data is independent...
EXECUTOR

Pipelining the pipeline

Each stage is asynchronous

Stages of given batch synchronized via events
OPERATORS

Gallery
USING DALI
import dali
import dali.ops as ops

class HybridRN50Pipe(dali.Pipeline):
    def __init__(self, batch_size, num_threads, device_id, num_devices):
        super(HybridRN50Pipe, self).__init__(batch_size, num_threads, device_id)
        # define used operators

    def define_graph(self):
        # define graph of operations
EXAMPLE: RESNET-50 PIPELINE

Defining operators

```python
def __init__(self, batch_size, num_threads, device_id, num_devices):
    super(HybridRN50Pipe, self).__init__(batch_size, num_threads,
                                         device_id)
    self.loader = ops.Caffe2Reader(path=lmdb_path, shard_id=dev_id,
                                    num_shards=num_devices)
    self.decode = ops.HybridDecode(output_type=dali.types.RGB)
    self.resize = ops.Resize(device="gpu", resize_a=256,
                              resize_b=480, random_resize=True,
                              image_type=types.RGB)
    self.crop = ops.CropMirrorNormalize(device="gpu",
                                         random_crop=True, crop=(224, 224),
                                         mirror_prob=0.5, mean=[128., 128., 128.],
                                         std=[1., 1., 1.], output_layout=dali.types.NCHW)
```
def define_graph(self):
    jpeg, labels = self.loader(name="Reader")
    images = self.decode(jpeg)
    resized_images = self.resize(images)
    cropped_images = self.crop(resized_images)
    return [cropped_images, labels]
**EXAMPLE: RESNET-50 PIPELINE**

*Usage: MXNet*

```python
import mxnet as mx
from dali.plugin.mxnet import DALIIterator

pipe = HybridRN50Pipe(128, 2, 0, 1)
pipe.build()
train = DALIIterator(pipe, pipe.epoch_size("Reader"))

model.fit(train,
           # other parameters
)
```
EXAMPLE: RESNET-50 PIPELINE

Usage: TensorFlow

```python
import tensorflow as tf
from dali.plugin.tf import DALIIterator

pipe = HybridRN50Pipe(128, 2, 0, 1)
serialized_pipe = pipe.serialize()
train = DALIIterator()

with tf.session() as sess:
    images, labels = train(serialized_pipe)
    # rest of the model using images and labels
    sess.run(....)
```
EXAMPLE: RESNET-50 PIPELINE

Usage: Caffe 2

```python
from caffe2.python import brew

pipe = HybridRN50Pipe(128, 2, 0, 1)
serialized_pipe = pipe.serialize()

data, label = brew.dali_input(model, ["data", "label"],
                                serialized_pipe=serialized_pipe)

# Add the rest of your network as normal
conv1 = brew.conv(model, data, "conv1", ...)
```
PERFORMANCE
PERFORMANCE
I/O Pipeline

Throughput, DGX-2, RN50 pipeline, Batch 128, NCHW
PERFORMANCE
End-to-end training

End-to-end DGX-2, RN50 training - MXNet, Batch 192 / GPU
NEXT STEPS
def define_graph(self):
    images, masks = self.loader(name="Reader")
    images = self.decode(images)
    masks = self.decode(masks)

    # Apply identical transformations
    resized_images, resized_masks = self.resize([[images, masks]], ...)
    cropped_images, cropped_masks = self.crop([[resized_images, resized_masks]], ...)

    return [cropped_images, cropped_masks]
NEXT: MORE FORMATS

What would be useful to you?

PNG

Video frames
NEXT++: MORE OFFLOADING

Fully GPU-based decode

HW-based via. NVDEC

Transcode to video
SOON: EARLY ACCESS

Looking for:

General feedback

New workloads

New transformations

Contact: Milind Kukanur
mkukanur@nvidia.com
ACKNOWLEDGEMENTS

Trevor Gale
Andrei Ivanov
Serge Panev
Cliff Woolley
DL Frameworks team @ NVIDIA