Multi-GPU Training in PyTorch with Code (Part 2): Data Parallel

Anthony Peng · Follow
Published in Polo Club of Data Science | Georgia Tech
7 min read · Jul 7, 2023

In Part 1, we successfully trained a ResNet34 on CIFAR10 using a single GPU. In this article, we will explore how to launch the training on multiple GPUs using Data Parallel (DP).

Before jumping into DP, I would like to discuss the setting of my machine. I have 8 GPUs and below shows the topology between each pair of GPUs. Only GPU 0&1, 2&3, 4&5, and 6&7 have high-speed NVLinks. All other data has to traverse the PCIe. This info will be useful while we are debugging DP below.

```
$ nvidia-smi topo -m
        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    CPU Aff
GPU0     X      NV4     NODE    NODE    SYS     SYS     SYS     SYS     0-31,64
GPU1    NV4      X      NODE    NODE    SYS     SYS     SYS     SYS     0-31,64
GPU2    NODE    NODE     X      NV4     SYS     SYS     SYS     SYS     0-31,64
GPU3    NODE    NODE    NV4      X      SYS     SYS     SYS     SYS     0-31,64
GPU4    SYS     SYS     SYS     SYS      X      NV4     NODE    NODE    32-63,9
GPU5    SYS     SYS     SYS     SYS     NV4      X      NODE    NODE    32-63,9
GPU6    SYS     SYS     SYS     SYS     NODE    NODE     X      NV4     32-63,9
GPU7    SYS     SYS     SYS     SYS     NODE    NODE    NV4      X      32-63,9
```

```
Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect
between NUMA nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect
between PCIe Host Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge
(typically the CPU)
  PXB  = Connection traversing multiple PCIe bridges (without
traversing the PCIe Host Bridge)
  PIX  = Connection traversing at most a single PCIe bridge
  NV#  = Connection traversing a bonded set of # NVLinks
```

Multi-GPU Data Parallel

1. *What is data parallel?*
   The idea of data parallel is copying the same model weights into multiple workers and assigning a fraction of data to each worker to be processed at the same time.

2. *How is it implemented in PyTorch?*
   DP splits the input across the specified devices by chunking in the batch dimension. In the forward pass, the module is replicated on each device, and each replica handles a portion of the input. During the backward pass, gradients from each replica are summed into the original module. For more info, please check the official API.

It's just one line of code to wrap up your model in DP.

```
model = torch.nn.parallel.DataParallel(model)
```

TrainerDP — Wrapping up our single model with DP. Note we set *gpu_id = "cuda"* so that DP can use all available GPUs. TrainerSingle is defined in Part 1 of this series.

```
from torch.nn.parallel import DataParallel
```

```
class TrainerDP(TrainerSingle):
    def __init__(
        self,
        model: nn.Module,
        trainloader: DataLoader,
        testloader: DataLoader,
    ):
        self.gpu_id = "cuda"
        super().__init__(self.gpu_id, model, trainloader,
testloader)

        self.model = DataParallel(self.model)

    def _save_checkpoint(self, epoch: int):
        ckp = self.model.state_dict()
        model_path = self.const["trained_models"] /
f"CIFAR10_dp_epoch{epoch}.pt"
        torch.save(ckp, model_path)
```

Main function. It's exactly the same code as using a single GPU.

```
def main_dp(final_model_path: str):
    const = prepare_const()
    train_dataset, test_dataset = cifar_dataset(const["data_root"])
    train_dataloader, test_dataloader = cifar_dataloader_single(
        train_dataset, test_dataset, const["batch_size"]
    )
    model = cifar_model()
    trainer = TrainerDP(
        model=model,
        trainloader=train_dataloader,
        testloader=test_dataloader,
    )
    trainer.train(const["total_epochs"])
    trainer.test(final_model_path)
```

Experiments

```
if __name__ == "__main__":
    final_model_path = Path("./trained_models/CIFAR10_dp_epoch14.pt")
    main_dp(final_model_path)
```

Output

```
$ CUDA_VISIBLE_DEVICES=6,7 python main.py
Files already downloaded and verified
Files already downloaded and verified
-------------------------------------------------------------------------------
[GPUcuda] Epoch  0 | Batchsize: 128 | Steps: 391 | LR: 0.1000 | Loss: 2.0683 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch  1 | Batchsize: 128 | Steps: 391 | LR: 0.1000 | Loss: 1.4562 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch  2 | Batchsize: 128 | Steps: 391 | LR: 0.1000 | Loss: 1.2057 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch  3 | Batchsize: 128 | Steps: 391 | LR: 0.1000 | Loss: 0.9887 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch  4 | Batchsize: 128 | Steps: 391 | LR: 0.0100 | Loss: 0.8248 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch  5 | Batchsize: 128 | Steps: 391 | LR: 0.0100 | Loss: 0.5498 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch  6 | Batchsize: 128 | Steps: 391 | LR: 0.0100 | Loss: 0.4616 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch  7 | Batchsize: 128 | Steps: 391 | LR: 0.0100 | Loss: 0.3924 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch  8 | Batchsize: 128 | Steps: 391 | LR: 0.0100 | Loss: 0.3204 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch  9 | Batchsize: 128 | Steps: 391 | LR: 0.0010 | Loss: 0.2484 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch 10 | Batchsize: 128 | Steps: 391 | LR: 0.0010 | Loss: 0.1450 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch 11 | Batchsize: 128 | Steps: 391 | LR: 0.0010 | Loss: 0.1216 |
-------------------------------------------------------------------------------
[GPUcuda] Epoch 12 | Batchsize: 128 | Steps: 391 | LR: 0.0010 | Loss: 0.1070 |
--------------------------------------------------------------------------------
[GPUcuda] Epoch 13 | Batchsize: 128 | Steps: 391 | LR: 0.0010 | Loss: 0.0951 |
--------------------------------------------------------------------------------
[GPUcuda] Epoch 14 | Batchsize: 128 | Steps: 391 | LR: 0.0001 | Loss: 0.0856 |
[GPUcuda] Test Acc: 74.7900%
```

As mentioned above, GPU 6&7 has the NVLink. Let's run the same code using a group of GPUs with at least one PCIe connection, e.g., GPU 5&6.

```
$ CUDA_VISIBLE_DEVICES=5,6 python main.py
Files already downloaded and verified
Files already downloaded and verified
--------------------------------------------------------------------------------
[GPUcuda] Epoch  0 | Batchsize: 128 | Steps: 391 | LR: 0.1000 | Loss: nan | Acc
--------------------------------------------------------------------------------
[GPUcuda] Epoch  1 | Batchsize: 128 | Steps: 391 | LR: 0.1000 | Loss: nan | Acc
```

...

The code execution speed is significantly slower as expected since GPU 5&6 traverses through PCIe. The full output is omitted since the loss is NaN in the first epoch. This is unexpected since we are launching the same code. The issue is also reported in the PyTorch forum.

We verify this phenomenon from another GitHub repo of training CIFAR10 models, which also uses DP for multi-GPU training. Below we show the results of running the code on GPUs w/ and w/o NVLink. We just partially print the output of the first epoch below, which is sufficient to make observations.

```
$ CUDA_VISIBLE_DEVICES=6,7 python main.py --lr 0.1
==> Preparing data..
Files already downloaded and verified
Files already downloaded and verified
==> Building model..
```

```
Epoch: 0
[>.................................................................]
Step: 6s679ms | Tot: 0ms | Loss: 2.34
[>.................................................................]
Step: 90ms | Tot: 90ms | Loss: 2.321
[>.................................................................]
Step: 43ms | Tot: 133ms | Loss: 2.349
[>.................................................................]
Step: 41ms | Tot: 175ms | Loss: 2.353
[>.................................................................]
Step: 43ms | Tot: 218ms | Loss: 2.364
[>.................................................................]
Step: 44ms | Tot: 263ms | Loss: 2.390
[>.................................................................]
Step: 40ms | Tot: 304ms | Loss: 2.423
[=>................................................................]
Step: 42ms | Tot: 346ms | Loss: 2.412
[=>................................................................]
Step: 44ms | Tot: 391ms | Loss: 2.399
[=>................................................................]
Step: 45ms | Tot: 436ms | Loss: 2.426
[=>................................................................]
Step: 44ms | Tot: 481ms | Loss: 2.410

...

$ CUDA_VISIBLE_DEVICES=5,6 python main.py --lr 0.1
==> Preparing data..
Files already downloaded and verified
Files already downloaded and verified
==> Building model..

Epoch: 0
[>.................................................................]
Step: 7s169ms | Tot: 0ms | Loss: 2.33
[>.................................................................]
Step: 236ms | Tot: 236ms | Loss: 2.33
[>.................................................................]
Step: 203ms | Tot: 439ms | Loss: 11.9
[>.................................................................]
Step: 203ms | Tot: 643ms | Loss: 3640
[>.................................................................]
Step: 200ms | Tot: 843ms | Loss: nan
[>.................................................................]
Step: 203ms | Tot: 1s47ms | Loss: nan
[>.................................................................]
Step: 203ms | Tot: 1s251ms | Loss: nan

...
```

Apparently, the loss explodes all the way to NaN after switching to GPUs w/o NVLink. As DP is summing all the gradients on multiple GPUs, we test whether reducing lr can help the training converge.

```
$ CUDA_VISIBLE_DEVICES=5,6 python main.py --lr 0.05
==> Preparing data..
Files already downloaded and verified
Files already downloaded and verified
==> Building model..
```

```
Epoch: 0
[>................................................................]
Step: 6s842ms | Tot: 0ms | Loss: 2.29
[>................................................................]
Step: 305ms | Tot: 305ms | Loss: 2.30
[>................................................................]
Step: 203ms | Tot: 509ms | Loss: 15.5
[>................................................................]
Step: 202ms | Tot: 711ms | Loss: 6728
[>................................................................]
Step: 203ms | Tot: 914ms | Loss: 3828
[>................................................................]
Step: 203ms | Tot: 1s118ms | Loss: nan
[>................................................................]
Step: 202ms | Tot: 1s320ms | Loss: nan
[=>...............................................................]
Step: 203ms | Tot: 1s523ms | Loss: nan

...

$ CUDA_VISIBLE_DEVICES=5,6 python main.py --lr 0.001
==> Preparing data..
Files already downloaded and verified
Files already downloaded and verified
==> Building model..

Epoch: 0
[>................................................................]
Step: 6s790ms | Tot: 0ms | Loss: 2.335
[>................................................................]
Step: 224ms | Tot: 224ms | Loss: 2.315
[>................................................................]
Step: 201ms | Tot: 425ms | Loss: 4.280
[>................................................................]
Step: 200ms | Tot: 625ms | Loss: 6.375
[>................................................................]
Step: 200ms | Tot: 826ms | Loss: 7.297
[>................................................................]
Step: 201ms | Tot: 1s27ms | Loss: 11.276
[>................................................................]
Step: 200ms | Tot: 1s228ms | Loss: 13.932
[=>...............................................................]
Step: 200ms | Tot: 1s429ms | Loss: 12.478
```

```
[=>.................................................................]
Step: 203ms | Tot: 1s632ms | Loss: 11.350
[=>.................................................................]
Step: 200ms | Tot: 1s833ms | Loss: 10.503

...
```

Since we use two GPUs, we first reduce the lr by half. However, the training diverges rapidly after a few steps. Only when we reduce the lr to 0.001, which is 1/100 of the original lr, the training starts to converge, but the loss is still gradually increasing. At the time of writing, it's still unclear how this happens, and whether this is a hardware and software mismatch between DP and NVLink. In the next article (Part 3), we will bypass the issue via DDP.

Deep Learning Gpu Machine Learning

Written by Anthony Peng

18 Followers · Editor for Polo Club of Data Science | Georgia Tech

CS PhD @Georgia Tech; Homepage: shengyun-peng.github.io/