PYTORCH DISTRIBUTED – ARULNITHI P (12210280)

In this first phase I have implemented Distributed data parallel in multiple GPUs at a single machine. I have used MNIST dataset and convolution model.

```
48
49
50  def ddp_setup():
51      init_process_group(backend="nccl")
52
53  class Trainer:
54      def __init__(
55          self,
56          model: torch.nn.Module,
57          train_data: DataLoader,
58          optimizer: torch.optim.Optimizer,
59          save_every: int,
60          snapshot_path: str,
61      ) -> None:
62          self.gpu_id = int(os.environ["LOCAL_RANK"])
63          self.model = model.to(self.gpu_id)
64          self.train_data = train_data
65          self.optimizer = optimizer
66          self.save_every = save_every
67          self.epochs_run = 0
68          self.snapshot_path = snapshot_path
69          if os.path.exists(snapshot_path):
70              print("Loading snapshot")
71              self._load_snapshot(snapshot_path)
72
73          self.model = DDP(self.model, device_ids=[self.gpu_id])
74
```

Line 73 DDP initialize the distributed data parallel approach in the multiple systems available. Models' replicas are shared.

Line 51 before the initializing of the DDP we need to create the process group and at for backend communication I have used nickle.(Nvidia distributed communication control protocol for CUDA gpus)

Dataloder

```
137
138 def prepare_dataloader(dataset: Dataset, batch_size: int):
139     return DataLoader(
140         dataset,
141         batch_size=batch_size,
142         pin_memory=True,
143         shuffle=False,
144         sampler=DistributedSampler(dataset)
145     )
```

Line 144 Distributed samples divides the data set into chunks and distributes to multiple gpus available.

The default batch size is 32. In single Gpus 1875 data given per epoch and in 2 gpu machines 938 and in 4 gpu machines 469.

Fault tolerance

I have used torchrun for fault tolerance it makes it easy for differencing number of GPUs, it sets up the world size and ID number of the GPU. It helps us to load the last saved snapshot and resume the operation from there.

Convolution neutral network model.

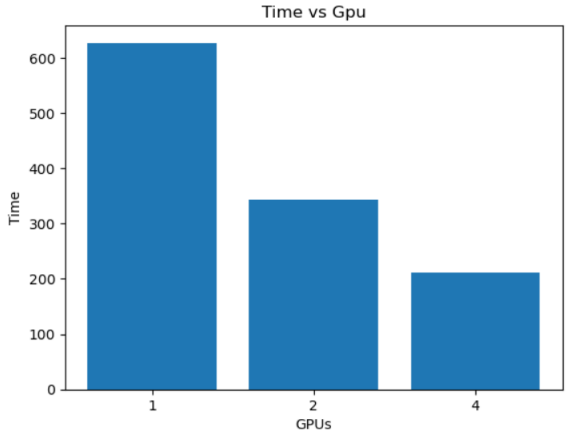
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)
```

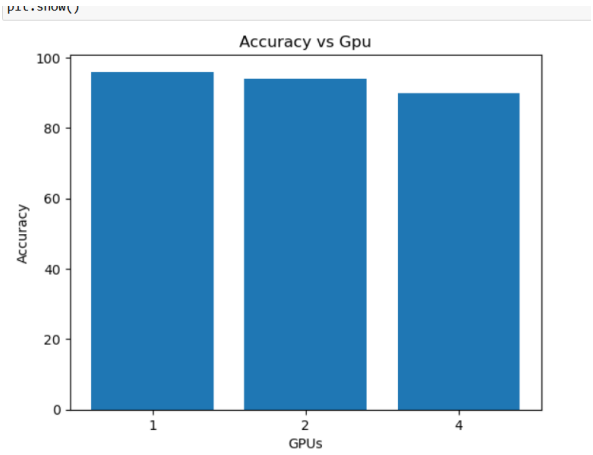
Experimental setup

I have run the model in Nvidia RTX 3080 GPUs. I have VAST.ai GPU renting service for my experiments.

Result



As we can see that time taken to train the model has significantly reduced.



As the number of GPUs increases the Accuracy has dropped a bit. With 4 GPUs we can significantly see that there is drop of 6 % compared to single GPU