Distributed Training on HAL System

Horovod

Horovod was originally developed by Uber to make distributed deep learning fast and easy to use, bringing model training time down from days and weeks to hours and minutes. With Horovod, an existing training script can be scaled up to run on hundreds of GPUs in just a few lines of Python code. Once Horovod has been configured, the same infrastructure can be used to train models with any framework, making it easy to switch between TensorFlow, PyTorch, MXNet, and future frameworks as machine learning tech stacks continue to evolve.

/home/shared/distributed-deep-learning-on-hal/05_horovod_tensorflow_mnist.py

```
import tensorflow as tf

# Choose the right backend
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model...
loss = ...
opt = ...

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes during initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)
```

The runscript is shown as follows

```
#!/bin/bash
#SBATCH --job-name="horovod_tensorflow_mnist"
#SBATCH --output="horovod_tensorflow_mnist_%j.out"
#SBATCH --partition=gpu
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=40
#SBATCH --cores-per-socket=20
#SBATCH --threads-per-core=4
#SBATCH --sockets-per-node=2
#SBATCH --mem-per-cpu=1200
#SBATCH --export=ALL
#SBATCH --gres=gpu:v100:4
#SBATCH --time=4:00:00

module load opence

mpirun -n 8 python 05_horovod_tensorflow_mnist.py
```

Apex

NVIDIA-maintained utilities to streamline mixed precision and distributed training in Pytorch. Some of the code here will be included in upstream Pytorch eventually. The intention of Apex is to make up-to-date utilities available to users as quickly as possible.

/home/shared/distributed-deep-learning-on-hal/03_pytorch_distributeddataparallel.py

```
def train(gpu, args):
    # calculate rank
    rank = args.nr * args.gpus + gpu
    # choose the nccl as backend
    dist.init_process_group(backend='nccl', init_method='env://', world_size=args.world_size, rank=rank)
    torch.manual_seed(0)
    model = ConvNet()
    # set device for each rank
    torch.cuda.set_device(gpu)
    # assign model to CUDA
    model.cuda(gpu)
    batch_size = 128
    # define loss function (criterion) and optimizer
    criterion = nn.CrossEntropyLoss().cuda(gpu)
    optimizer = torch.optim.SGD(model.parameters(), 1e-4)
    # wrap the model
    model = nn.parallel.DistributedDataParallel(model, device_ids=[gpu])
    ...
```

The runscript is shown as follows

```
#!/bin/bash
#SBATCH --job-name="pytorch_distributeddataparallel"
#SBATCH --output="pytorch_distributeddataparallel_%j.out"
#SBATCH --partition=gpu
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=40
#SBATCH --cores-per-socket=20
#SBATCH --threads-per-core=4
#SBATCH --sockets-per-node=2
#SBATCH --mem-per-cpu=1200
#SBATCH --export=ALL
#SBATCH --gres=gpu:v100:4
#SBATCH --time=4:00:00
#SBATCH --reservation=uiuc_32

export MASTER_PORT=12340
export WORLD_SIZE=8

echo "NODELIST="${SLURM_NODELIST}
master_addr=$(scontrol show hostnames "$SLURM_JOB_NODELIST" | head -n 1)
export MASTER_ADDR=$master_addr
echo "MASTER_ADDR="$MASTER_ADDR

module load opence/1.3.1

python 03_pytorch_distributeddataparallel.py
```