Name: *Souham Biswas* ***CWID***: *A20365242* ***Email***: *sbiswas7@hawk.iit.edu*

CS 597 – Report 7

Finally circumvented Loss NaN issue on cooley; x_upscale & y_upscale values were not set before calling cuDNN initializer.

Encountered NaN loss upon using convolutional layers for CIFAR-10. Apparently large initial weights were being initialized which caused gradients to explode too early. Countered by using uniform distribution from -.05 to +.05

Implemented AlexNet (stripped down version for CIFAR-10); model is converging with training speed of about 0.0009 s per batch of 128 images of dimension 32 x 32 x 3; the network architecture is defined as follows –

Input → Conv0 (5x5 kernel, stride=1, pad=2, 32 feature maps) → Conv1 (5x5 kernel, stride=1, pad=2, 32 feature maps) → Conv2(5x5 kernel, stride=1, pad=2, 64 feature maps) → FullyConnected0 (64 neurons) → FullyConnected1 (32 neurons) → FullConnected2 (Softmax layer with 10 neurons)

A constant learning rate of 0.001 was used with an L2 regularization strength of 0.004 and 0.9 momentum.

The training experiences performance bottlenecks while moving data between host and GPU. According to internet, this is more of a windows problem. Investigating this issue.

Started with MPI implementation; created some dummy node to node communication patterns mirroring the use-cases in a data-parallel training environment. A sample output of the distribution benchmarks is presented as follows –

```
---->Start time = 1489166518.491115
Gradient Fetch Time = 0.013477
#Worker_Gradients = 4
Gradient Avg time = 0.003210
Global Step 0, Learning Rate = 0.400000
Weight Update time = 0.000320
Local Step 0, Loss = 5.883390
------>End time = 1489166519.363505
Time spent = 0.872390
Weight distribution time = 0.010717
```

This implementation does not include neural network training; placeholder operations have been incorporated to simulate training process.

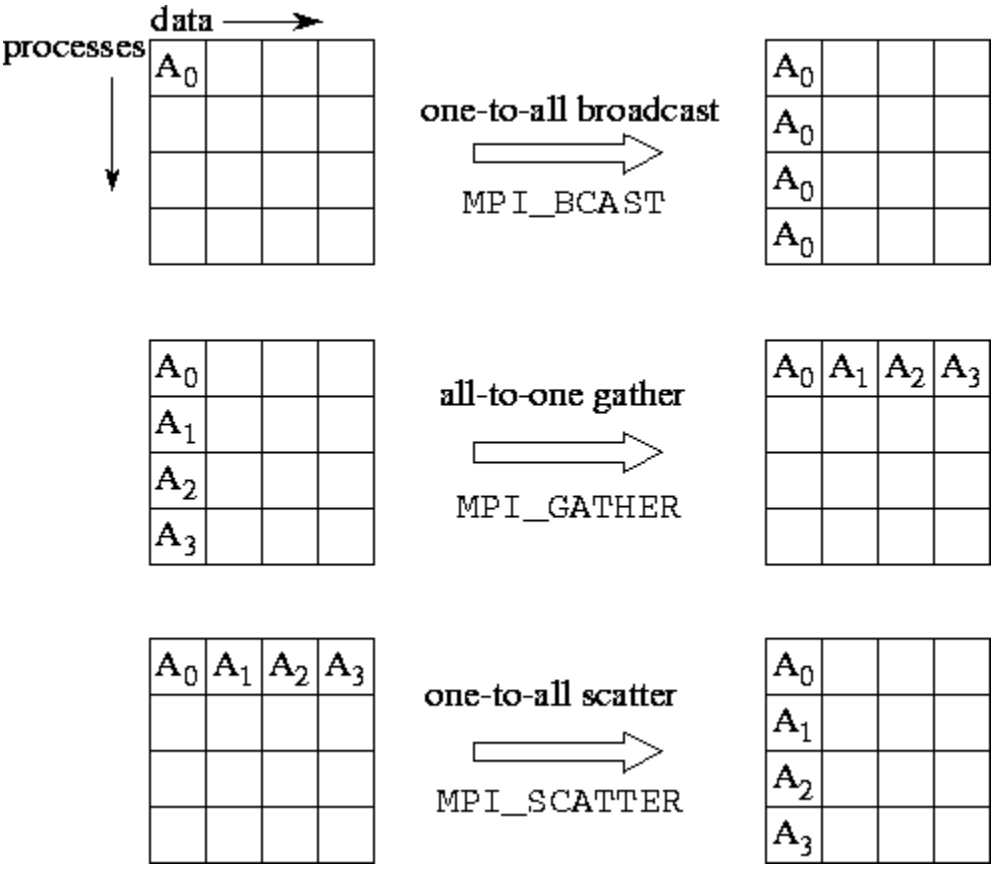
Basically, dummy weight matrices having similar dimensions as the weight matrices from the actual AlexNet (smaller version) were initialized with random values. These matrices are technically held at what is called the “parameter shard”. These are distributed to the “worker shards” which perform a dummy gradient computation operation where each worker shard generates matrices having the same dimensions as the input weights and random values, which are in this context, the “gradients”.

Name: *Souham Biswas* ***CWID***: *A20365242* ***Email***: *sbiswas7@hawk.iit.edu*

CS 597 – Report 7

The parameter shard then performs an all-to-one gather operation and gets the gradients from the workers and averages them. The averaging is currently being done using the CPU; this will be moved to the GPU once the GPUDirect functionality is enumerated and its usage is understood.

Once the gradients have been averaged, these gradients are then subtracted (weight update operation) from the weight values stored at the parameter shard and the updated weights are redistributed to the workers using a one-to-all broadcast operation.



Future work includes performing these operations along with training and leveraging presence of multiple GPUs on a single node to further parallelize the training (preferably using NCCL).