HPMLlab4

Yonathan Daniel

April 2024

Q1

Table 1: Speedup Measurement for different Batch Size (Batch size per GPU: 32)

GPU	Time(sec)	Speedup
1	48.61245506070554	1
2	30.818510502576828	1
4	30.8219216464595	1

Table 2: Speedup Measurement for different Batch Size (Batch size per GPU: 128)

GPU	Time(sec)	Speedup
1	27.22366295568645	1.78566914893
2	20.923493693244424	1.47291417745
4	19.878172473981977	1.55054101109

Table 3: Speedup Measurement for different Batch Size (Batch size per GPU: 512)

GPU	Time(sec)	Speedup
1	25.94455107487738	1.04930175423
2	15.664426802657545	1.33573312045
4	15.1269	1.31409426082

Above are my result from the various batchsizes across multipe GPUs. In my case I started to notice issues when I went towards anything above 512 batchsize. At around 512 we can see that batchsize the speed up is starting to stagnate. We can see as well using a distributed architecture like DDP we can achieve better results in terms of performance. The difference in speed between 2 GPUs and 4 GPUs was minuscule.

Q2

My results for question 2 are also located in q1 as stated earlier we can see that the difference between 2 and 4 GPUs in terms of performance was small. In the 512 batchsize the 4 GPU congfiguration was actually slower but 0.5 of a second. With respect to batchsize the peformance scales strongly as the increases, and for the number of GPUs it is weak scaling as the speed up is significant but not as drastic in comparison to the batchsize increase.

Q3

3.1

Table 4: Compute and Communication time for Batch Size: 32

GPU	**Batch size per GPU: 32**	
	Compute(sec)	**Comm(sec)**
2-GPU	17.719614028930664	0.14946778118610382
4-GPU	18.085159301757812	0.20533965528011322

Table 5: Compute and Communication time for Batch Size: 128

GPU	**Batch size per GPU: 128**	
	Compute(sec)	**Comm(sec)**
2-GPU	4.26522249438477	0.049557614311409
4-GPU	4.496433734893799	0.05234416574239731

Table 6: Compute and Communication time for Batch Size: 512

GPU	**Batch size per GPU: 512**	
	Compute(sec)	**Comm(sec)**
2-GPU	1.30768883228302	0.034110888838768005
4-GPU	1.285510540008545	0.03125176951289177

I found the the compute and communication times in a similar manner to homework2 where parts of the neural network were sectioned off to compute benchmarks. The compute was for the gradient computation and the communication time was found via how long the dataloader took to load from memory.

3.2

In the Bandwidth Optimal All-reduce Algorithms for Clusters of Workstations I found the following equations that I believe were useful to compute the values required. 1. Total Communication Time:

$$T = 2 \times (N-1) \times T\left(\frac{X \times itsize}{N}\right)$$

where:

- T is the total communication time,
- N is the number of processes,
- X is the number of items,

- *itsize* is the size of each item,

2. Bandwidth Utilization:

$$BandwidthUtilization = \frac{TotalData}{TotalCommunicationTime}$$

Assuming CIFAR10 is 0.1718 GB we can say the Bandwith is

$$BW = \frac{.1718}{TotalCommunicationTime}$$

Table 7: Communication Bandwidth utilization

GPU	**Batch-size-per-GPU 32**	**Batch-size-per-GPU 128**	**Batch-size-per-GPU 512**
	Bandwidth Utilization(GB/s)	Bandwidth Utilization(GB/s)	Bandwidth Utilization(GB/s)
2	1.19089210121	3.59177903281	5.21827504529
4	0.86685642749	3.40057000576	5.69567748561

Q4

4.1

In my implementation after the fifth epoch my training loss was 0.6241 and training accuracy was 78.34% in homework 2 the results for average training loss and training acurracy were 0.7153 75.36% respectively.

4.2

The following are two methods to make sure training accuracy does not suffer as batch sizes increase.
Gradual Warmup: Implementing a gradual warmup strategy can help address optimization difficulties early in training with large minibatches. By starting with lower learning rates and gradually increasing them, the model can navigate the challenges posed by large minibatch sizes. We have done similar in this homework as well as in homework2.

Linear Scaling Rule for Learning Rate: Adopting a hyperparameter-free linear scaling rule to adjust the learning rate based on the minibatch size can also enhance training accuracy. This allows the learning rate 5o adjusted properly to accommodate the larger minibatches which can lead to improved accuracy/performance.

Q5

DDP is more hands on than DP essentially this means that there are less abstractions for the developer. This means that in a distrbuted architecture there will be some need for the developer to maintain consistency across the application. Setting the Epoch Id allows the processes to direct their attention to the specific task and maintains synchronization.

Q6

No gradients are not the only messages sent between learners. In Data Distributed Parallel from what I read some of the operations act as locking mechanisms, so worker processes do not get to far ahead or cause potential scheduling conflicts.

Q7

From what I read in Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour computing the gradients only is a sufficient method to handle the 4 GPU 512 batchsize case. Given that Gradients can be split and computed separately in this setting it would be sufficient to use this approach in the 512 batchsize case. This is because it is significantly larger than 32 and 128 size cases, so benefits of parallelism would be more apparent.