HPML LAB 4

Name: Sree Gowri Addepalli
NetID : sga297

Configurations:

srun --cpus-per-task=10 --gres=gpu:k80:8 --mem=120GB --reservation=zhang --partition=k80_8 --pty /bin/bash

Q1. Computational Efficiency w.r.t Batch Size (5pt)

Sr No.	Batch Size	Time(Seconds)
1.	32	46.355837
2.	128	18.280495
3.	512	5.201220

Reason:

Code is run on single GPU until batch size 2048, which is where “memory cannot hold the batch size anymore” or “Runtime Error: CUDA out of memory” comes and it breaks.

On Model-k80 GPU 01, it takes 46.355837 seconds to train one epoch (w/o data-loading) using mini-batch size 32, 18.280495 seconds using batch-size 128, 5.201220 seconds using batch-size 512.

When a batch size is large, it reduces the number of parameters that need to be updated for training the model which is why it takes less time to train. There is an increase in speed due to GPU parallelism which internally uses linear algebra libraries which uses vectorization internally for vector and matrix operations which improves computation speed way better even if memory is used.

Q2. Speedup Measurement (5pt)

	Batch-size	32 per GPU	Batch-size	128 per GPU	Batch-size	512 per GPU
	Time(Sec)	Speedup	Time (Sec)	Speedup	Time (Sec)	Speedup
1 GPU	62.661095	1	16.937219	1	4.946042	1
2 GPU	44.222406	1.41695	12.256314	1.38191	4.017061	1.18592
4 GPU	28.341740	2.21091	8.272321	2.04745	3.616366	1.367682

Here the running time includes cpu- gpu time, gradient and weight calculation and no data I/O.

It is weak scaling here as total compute is fixed but elements per process to compute need to vary since the number of processes increases.

Scaling is the used for building high performance computing systems that allows to use multiple systems and utilize computation power in parallel when the resources utilization is increased, and it is governed by Amdahl's law.

Strong scaling measures speedup for a fixed problem size with respect to the number of processors. So, while strong scaling, every gpu gets the same input batch.

For example, while using 4 GPUs with 32 images and another 128 images and one GPU, speedup is 2.21.

It is observed that strong scaling causes slowdown while increasing the GPUs that makes a slowdown for constant batch size for which strong scaling has results which are worse. With the same data size and increase in number of GPUs, time increases when it should decrease, which is why strong scaling is worse.

Q3.1 Computation vs Communication (5pt)

	Batch-size	**32 per GPU**	**Batch-size**	**128 per GPU**	**Batch-size**	**512 per GPU**
	computation	**communication**	**computation**	**communication**	**computation**	**communication**
2 GPU	40.171962	16.9940435	11.689425	2.5491775	3.216890	0.61628
4 GPU	27.699833	16.11087375	7.622034	3.05191025	2.277797	0.977492

Computation time is calculated as done in Q1 where compute time includes CPU-GPU transferring and calculation(training) time except the time taken for data loading.

Communication Time: (Time to compute in multi gpu -Time to compute in single gpu/Number of GPUs in multi gpu environment)

It means that communication time is the difference between the time taken to run in one epoch in a multi gpu environment setting minus the time taken to compute in a single gpu per epoch/4.

For e.g.: 27.699833 - 46.355837/4 = 27.699833 - 11.58895925 = 16.11087375

Q3.2 Communication bandwidth utilization (10pt)

	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-GPU	(1* 4 * 11173962* 50000) / (16.9940435 *64)	(1*4*11173962*50000) /(2.5491775* 256)	(1*11173962*4*50000) /(1024*0.61628)
4-GPU	(1.5*4 * 11173962*50000) /(16.11087375 * 128)	(1.5*4*11173962*50000)/(3.05191025*512)	(1.5*11173962*4*50000) /(0.977492 * 2048)

	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-GPU	**2.05475708297**	3.42449978964	3.54127093711
4-GPU	**1.62554643801**	2.14529354504	1.67450049703

1) First, list the formula to calculate how long does it take to finish an all reduce:

It is the same as the communication time calculated as in 3.1

Communication Time: (Time to compute in multi gpu -Time to compute in single gpu/Total Number of Gpus in multi gpu setup.)

2) Second, list the formula to calculate the bandwidth utilization:

Formula: 2 * (N-1)/N * 4 * Number of Trainable parameters * 50000/ (batch size * number of GPUs * communication time)

As per Lemma 4 [1], to perform an all-reduce operation on X items with itsize bytes item size on N processes, there exists at least one process that must communicate a total of at least ⌈2 × (N-1)/ N × X⌉ × itsize bytes data assuming the minimum N unit for communication is an item.

N is the number of GPUs.
X is the number of trainable parameters for this model.
itsize - number of bytes to store each trainable parameter. (which is 4 here, as it is float type.)

Trainable parameters are of type: Parameters are of type “torch. float32” which is 32-bit floating point i.e., 4 bytes.

No. of training images = 50000

Bytes to GB – bytes/2^30

For one epoch, number of iterations = Total number of training images/ (batch size * number of GPUs used) = Total number of training images/ Total batch size

So, **bandwidth utilization (Gb/s)= (⌈2 × (N-1)/ N × Number of trainable parameters x ((total number of training images/ (batch size * number of GPUs used))) x 4⌉/2^30)(GB) / Communication Time.(Sec)**

References:

[1] P Patarasuk and X Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," J. Parallel Distrib. Comput., vol. 69, pp. 117–124, 2009.

Q4.1 Accuracy when using large batch (5pts)

Current lab:

Largest batch size: 512.
Average Accuracy: 46.164000
Average Epoch Loss: 1.449112

Lab2 c6 SGD

Largest batch size: 512.
Average Accuracy: 49.78
Average Epoch Loss: 1.3860

As seen the training accuracy has decreased when batch size has increased.

As seen in Q1, when we increase the batch size, lesser is the training time due to parallelism by GPU. Batch size is large we take larger steps and lose generalization. For a fixed number of epochs, larger batch sizes take fewer steps. While in smaller batch size, we might get convergence but that may not be global optima and we may be stuck around local optima. We can see that training loss also increases.

Reference:
https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

Q4.2 How to improve training accuracy when batch size is large (5pts)

Batch size is one of the most important hyperparameters to tune in modern deep learning systems. Practitioners often want to use a larger batch size to train their model as it allows computational speedups from the parallelism of GPUs. However, it is well known that too large of a batch size will lead to poor generalization.

On the one extreme, using a batch equal to the entire dataset guarantees convergence to the global optima of the objective function. However, this is at the cost of slower, empirical convergence to that optima.

On the other hand, using smaller batch sizes have been empirically shown to have faster convergence to “good” solutions. This is intuitively explained by the fact that smaller batch sizes allow the model to “start learning before having to see all the data.” The downside of using a smaller batch size is that the model is not guaranteed to converge to the global optima.

Some remedies are:

1. **Linear Scaling Rule**: When the minibatch size is multiplied by k, multiply the learning rate by k. But we can recover the lost test accuracy from a larger batch size by increasing the learning rate in this fashion as per [2]. The linear scaling rule can help us to not only match the accuracy between using small and large minibatches, but equally importantly, to largely match their training curves, which enables rapid debugging and comparison of experiments prior to convergence.

2. It also means larger batch sizes either makes larger gradient steps or smaller batch sizes for the same number of samples seen and variance increases with batch size. The size of the update depends heavily on which samples are drawn from the dataset. On the other hand, using small batch size means the model makes updates that are all about the same size. The size of the update only weakly depends on which samples are drawn from the dataset. Better solutions can be far away from the initial weights and if the loss is averaged over the batch then large batch sizes simply do not allow the model to travel far enough to reach the better solutions for the same number of training epochs. One can compensate for a larger batch size by increasing the number of epochs so that the models can find faraway solutions as per [1].
3. **Gradual warmup** - As per [2], the author presents an alternative warmup that gradually ramps up the learning rate from a small to a large value. This ramp avoids a sudden increase of the learning rate, allowing healthy convergence at the start of training. In practice, with a large minibatch of size kn, we start from a learning rate of η and increment it by a constant amount at each iteration such that it reaches $\hat{\eta} = k\eta$ after 5 epochs (results are robust to the exact duration of warm up). After the warmup, we go back to the original learning rate schedule.

References:

[1] https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

[2] P Goyal, P Doll´ar, R. B Girshick, P Noordhuis, L Wesolowski, A Kyrola, A Tulloch, Y Jia, and K He, “Accurate, large minibatch SGD: training imagenet in 1 hour,” CoRR, vol. abs/1706.02677, 2017.

Q5. Distributed Data Parallel (5pt)

Data Parallel is single-process, multi-thread, and only works on a single machine, while Distributed Data Parallel is multi-process and works for both single- and multi- machine training. Thus, even for single machine training, where your data is small enough to fit on a single machine, Distributed Data Parallel is expected to be faster than Data Parallel. Distributed Data Parallel also replicates models upfront instead of on each iteration and gets Global Interpreter Lock out of the way.

As per the references below, epoch id is used in manual seed for shuffling the data in batch in Distributed Data parallel while Data Parallel doesn't use it. So, in Data parallel, at a time the same epoch id is used while Distributed data parallel, we need the epoch ID for reproducibility of the data when using manual seed.

Reference:

[1] https://pytorch.org/tutorials/intermediate/ddp_tutorial.html
[2] https://pytorch.org/docs/stable/_modules/torch/utils/data/distributed.html

Q6. What are passed on network ? (5pt)

No, gradients are not the only messages that are communicated across learners. Batch norms are also communicated. Batch-size is an important hyper-parameter of the model training. Larger batch sizes often converge faster and give better performance. As per reference paper **[1]** for a minibatch size n over which the BN statistics are computed is a key component of the loss: if the per worker minibatch sample size n is changed, it changes the underlying loss function L that is optimized. More specifically, the mean/variance statistics computed by BN with different n exhibit different levels of random variation. Batch norm parameters don't have a gradient, but they are required for the loss term calculation.

Reference:
[1] P Goyal, P Doll´ar, R. B Girshick, P Noordhuis, L Wesolowski, A Kyrola, A Tulloch, Y Jia, and K He, “Accurate, large minibatch SGD: training imagenet in 1 hour,” CoRR, vol. abs/1706.02677, 2017.

Q7. What if we only communicate gradients ? (5pt)

Yes, it is enough to communicate only gradients across 4 GPUs. As per reference paper **[1]** for a minibatch size **n** over which the BN statistics are computed is a key component of the loss: if the

per worker minibatch sample size n is changed, it changes the underlying loss function L that is optimized. More specifically, the mean/variance statistics computed by BN with different n exhibit different levels of random variation. If n is fixed as in our case for a batch size of 512, the loss function doesn't change, so such a parameter should not be communicated, not only for the sake of communication but also for maintaining the same underlying loss function being optimized. Also, batch norms for the multi gpu setup can be evaluated from the initial GPU.

Reference:

[1] P Goyal, P Doll´ar, R. B Girshick, P Noordhuis, L Wesolowski, A Kyrola, A Tulloch, Y Jia, and K He, “Accurate, large minibatch SGD: training imagenet in 1 hour,” CoRR, vol. abs/1706.02677, 2017.