Q1. On ResNet18 model GPU, it takes 269.804 seconds to train one epoch (w/o data-loading) using mini-batch size 32, 228.871 seconds using batch-size 128, 211.488 seconds using batch-size 512. When batch size is larger, it takes shorter time to train because arithmetic intensity leans towards compute-bound. Due to more computations over a large batch size per iteration and fewer batch retrieval, we get more time spent training inside the GPU and fewer loop iterations which results in fewer CPU-to-GPU and back overhead.

Q2.

Table 1:

	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	271.984	1	229.739	1	211.854	1
2-GPU	153.240	1.775	125.222	1.835	116.669	1.816
4-GPU	79.637	3.415	63.705	3.606	58.438	3.625

Table 1 records the training time and speedup for different batch size up to 4 GPUs. Here we are observing a weak scaling as we're increasing the problem size per iteration but keeping it proportional to the number of GPUs in use. For instance, the total batch size per iteration for a 1-GPU setup is 32 whereas it would be 64 in a 2-GPU setup. Comparing the results for strong scaling, on the other hand, is ineffective as the communication overhead would drastically hamper the performance especially for a small batch size such as 32. In effect, splitting that batch into 4 GPUs would mean that each learner only works on 8 input images, leaving little work to be done for the GPUs whereas communication cost between GPUs and to and from CPU would be more expensive. I tried implementing a strong scaling scenario whereby the total batch size is 32 split across the GPUs and my results were 274.355sec for 2-GPU and 241.768sec for 4-GPU. No noticeable speedup was observed in this case.

Q3.1.

Table 2:

	Batch-size 32 per GPU		Batch-size 128 per GPU		Batch-size 512 per GPU	
	Compute(sec)	Comm(sec)	Compute(sec)	Comm(sec)	Compute(sec)	Comm(sec)
2-GPU	152.249	17.347	124.729	10.294	116.121	10.377
4-GPU	78.910	11.459	63.170	5.952	57.686	4.814

To calculate the communication cost for a multi-GPU setup, we first record the training time for that setup and compare it to the training time for a 1-GPU setup which we had recorded for Q1. From Q1, we have the training time for only 1 GPU. Then, supposing we run it in a 2-GPU setup, the expected running time should be scaled inverse proportionally. For instance, for 1 GPU in a 32-batch size, we have

269.804. The time we expect it would take had we used 2 GPUs should be 134.902. However, the actual training time for a 2-GPU setup is 152.249. Therefore, the difference is the communication costs that account for the additional overhead when using multiple GPUs.

Q3.2.

Table 3:

	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	2.577	4.342	4.307
4-GPU	5.851	11.262	13.927

We will be following the formula stated in the paper to calculate the bandwidth. From the paper, we know that at each step, each process is sending and receiving $\frac{X*itsize}{N}$ data for a duration of N-1 steps and done twice to account for reduce-scatter and all-gather, where N is the number of processes (i.e. GPUs), X items with itsize bytes. Then, the total data moved is $2*(N-1)*\frac{X*itsize}{N}$. We know the total number of parameters to be X = 11173962. Each parameter is a float32 so itsize = 4. From table 2, we know the actual communication time. So all we need is to divide them to get the bandwidth.

Q4.1. The average training loss and training accuracy for the 5th epoch for batch size per gpu 512 on 4 GPUs is 0.6444 and 27.50%. Whereas in hw2 with batch size 128 on 1 GPU, the average loss and accuracy was 0.685 and 79.55%. We have suffered a tremendous accuracy loss with this setup.

Q4.2. Remedy 1 is hyper-parameter-free linear scaling rule – that is when the minibatch size is multiplied by k, scale the learning rate by the same amount. Remedy 2 is to use gradual warmup which is to gradually ramp up the learning rate until a certain threshold before going back to the original learning rate.

Q5. One needs to set up epoch ID because we must synch with the different processes in each node. This way if one of the nodes finishes the training earlier, it must wait for the other processes to arrive at the same epoch.

Q6. No, statistics such as mean and variance when using Batch Norm for instance is passed and synched across GPUs. This statistic can be aggregated along in a ring reduction.

Q7. No, because as seen in Q4.1, the training accuracy suffered drastically compared to a smaller batch size and 1-GPU setup form hw2. We can apply the remedies from Q4.2 and more as specified in the paper "Accurate, large minibatch SGD: training imagenet in 1 hour" by P Goyal et al. as well as batch normalization synchronized across the GPUs that requires further communication of meta data.