1. Profile your distributed training experiments conducted in Lab #2 further to gain insights about performance. Profiling analysis should include:
2. Breakdown of computation and communication time
3. Identification of any specific communication pattern, (e.g. Broadcast, All-to-all, or something other)
4. Factors that affect the amount of computation. (e.g. batch size)
5. Factors that affect the amount of communication performed (e.g. model type)
6. Investigate opportunities for performance improvement for your chosen DL framework. This is an open research question, so you can explore different options to accelerate the training time.

A few example options are provided below:

* 1. Overlapping of communication and computation
  2. Using multiple CUDA streams for computation and communication
  3. Finding out the best set of hyper-parameters like learning rate to train quickly
  4. Using different communication libraries to improve performance (e.g. OpenMPI vs. MVAPICH2)

We explored several possibility within the structure of pytorch + horovod + MPI.

**Overlapping.** Due to the limitation of horovod library, we cannot operate in some very low-level mechanism, such as the operating each thread’s communication and computation. However, this is not necessarily a disadvantage. The primary benefit of horovod is that we can make the distributed training work without manually setting up the tedious options. Without a good optimization strategy, the performance is not guarantee to be better than the default.

**Multiple CUDA streams.** For the same reason, horovod does not support manipulating CUDA streams. However, outside the horovod library in the native pytorch CUDA API, it does support CUDA streams setting. A CUDA stream is a linear sequence of execution that belongs to a specific devices. One normally does not need to create one explicitly: by default, every device will use their own default stream. For the case of our experiments, it is also the case.

**Hyper-parameters.**

**MPI version.** The default MPI release we are using is mvapich2/2.3.1-gpu, which is the MPI release over InfiniBand and other high-speed interfaces. We compared the Gated CNN model’s training performance over these two MPI releases and the results are shown in Table 1.

|  |  |  |
| --- | --- | --- |
|  | Mvapich2/2.3.1-gpu | Openmpi/4.0.1 |
| Total training time (secs) | 391 | 371 |
| Computation (secs) | 287 | 244 |
| Communication (secs) | 104 | 127 |

Table 1 Training performance's over the two MPI versions.

In general, mvapich2 can achieve lower communication cost compared to vanilla openmpi 4.0.1; however, the overall computation time and the total training time are much smaller for openmpi 4.0.1, which is anti-intuitive from the experience with the results derived by Tensorflow. We think the reason is because the unique feature of pytorch, which is a define-by-run framework.