Zhuo Xu

Mentor: Mahdieh Ghazimirsaeed

CSE5194.01

December 11, 2020

Profiling communications in HiDL applications using TAU

As the complexity and diversity of computer hardware and the elaborateness of network technologies have made the implementation of portable and efficient algorithms more challenging, the need to understand application communication patterns has become increasingly relevant. Indeed, the transfer time of data exchanges between processes of an HiDL application depends on both the software and the hardware.

In this project, I utilized TAU, a performance analysis tool framework developed by University of Oregon to profile the MPI communication. TAU provides a suite of static and dynamic tools that provide graphical user interaction and inter-operation to form an integrated analysis environment for parallel applications. The model that TAU uses to profile parallel, multi-threaded programs maintains performance data for each thread, context, and node in use by an application. The profiling instrumentation needed to implement the model captures data for C++ functions, methods, basic blocks, and statement execution at these levels.

From the profile data collected, TAU's profile analysis procedures generated a wealth of performance information. It shows the exclusive and inclusive time spent in each function with nanosecond resolution. It also shows the breakup of time spent for each node. Other data includes how many times each function was called, how many

profiled functions did each function invoke, and what the mean inclusive time per call was. Time information can also be displayed relative to nodes, contexts, and threads.

The experiments were conducted on Ohio Supercomputer Center's Owens Cluster. Each node has 28 CPU cores and 1 NVIDIA Tesla P100 GPU. For this experiment I was using Pytorch 1.7 as my deep learning framework, and Horovod 0.21. Horovod was compiled with MVAPICH2.3.4-GDR CUDA-aware Message Passing Interface. MVAPICH2.3.4-GDR was installed with CUDA 10.2.89.

During the experiment setup period, I experienced some difficulties to get the environment properly setup using Slurm for job scheduling and resource management. All of my pervious environment setup can't work properly. After realizing that Slurm job scheduler could be the problem, I switched back to Torque/Moab environment, and get the experiment environment properly set up.

In this experiment, I trained a Convolutional Neural Networks models containing two layers of Convolutional layers, two fully connected layers and a dropout layer in between. The model was fed with MNIST as testing dataset and was parallelized on 4 Owens Cluster node.

I run the model for 2 epochs each time with different training and testing batch size, the results shows that training batch size have some impact on the MPI communication pattern.

Communication profile of running the model with training batch size 128:

```
FUNCTION SUMMARY (total):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0     2:50.742     5:01.677          17       43095   17745746 .TAU application
 43.0     2:09.763     2:09.763           9           0   14418112 OpenMP_Thread_Type_ompt_thread_initial
  0.2          641          641           4           0     160443 MPI_Init_thread()
  0.2          513          513       42929           0         12 MPI_Allreduce()
  0.0           10           10           4           0       2527 MPI_Finalize()
  0.0            2            2           4           0        565 MPI_Comm_dup()
  0.0            2            2           4           0        546 MPI_Comm_split()
  0.0        0.782        0.782           8           0         98 MPI_Allgather()
  0.0        0.109        0.485           4           4        121 OpenMP_Parallel_Region at::native::rand
777/zhuoxu3614/miniconda3/envs/horovod_mv2/lib/python3.8/site-packages/torch/lib/libtorch_cpu.so} {0, 0}]
  0.0        0.376        0.376           4           0         94 OpenMP_Implicit_Task
  0.0        0.191        0.191          12           0         16 MPI_Comm_free()
  0.0        0.186        0.186          37           0          5 MPI_Bcast()
  0.0        0.112        0.112           4           0         28 MPI_Type_commit()
  0.0        0.107        0.107           8           0         13 MPI_Gather()
  0.0         0.07         0.07           4           0         18 MPI_Type_contiguous()
  0.0         0.04         0.04           4           0         10 MPI_Type_free()
  0.0        0.037        0.037           8           0          5 MPI_Gatherv()
  0.0        0.034        0.034           4           0          8 MPI_Op_create()
  0.0        0.014        0.014           4           0          4 MPI_Query_thread()
  0.0        0.011        0.011           4           0          3 MPI_Finalized()
  0.0        0.011        0.011           4           0          3 MPI_Op_free()
  0.0        0.007        0.007          20           0          0 MPI_Comm_rank()
  0.0        0.004        0.004          12           0          0 MPI_Comm_size()
  0.0        0.001        0.001           4           0          0 MPI_Type_size()
```

Communication profile of running the model with training batch size 64:

```
FUNCTION SUMMARY (total):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0     2:53.343     5:07.822          17       44657   18107227 .TAU application
 43.3     2:13.310     2:13.310           9           0   14812267 OpenMP_Thread_Type_ompt_thread_initial
  0.2          637          637           4           0     159312 MPI_Init_thread()
  0.2          515          515       44495           0         12 MPI_Allreduce()
  0.0           10           10           4           0       2550 MPI_Finalize()
  0.0            2            2           4           0        566 MPI_Comm_dup()
  0.0            2            2           4           0        546 MPI_Comm_split()
  0.0        0.783        0.783           8           0         98 MPI_Allgather()
  0.0        0.128        0.542           4           4        136 OpenMP_Parallel_Region at::native::rand
777/zhuoxu3614/miniconda3/envs/horovod_mv2/lib/python3.8/site-packages/torch/lib/libtorch_cpu.so} {0, 0}]
  0.0        0.414        0.414           4           0        104 OpenMP_Implicit_Task
  0.0        0.182        0.182          35           0          5 MPI_Bcast()
  0.0        0.152        0.152          12           0         13 MPI_Comm_free()
  0.0        0.117        0.117           7           0         17 MPI_Gather()
  0.0        0.112        0.112           4           0         28 MPI_Type_commit()
  0.0        0.071        0.071           4           0         18 MPI_Type_contiguous()
  0.0        0.043        0.043           4           0         11 MPI_Type_free()
  0.0        0.041        0.041           7           0          6 MPI_Gatherv()
  0.0        0.032        0.032           4           0          8 MPI_Op_create()
  0.0        0.013        0.013           4           0          3 MPI_Query_thread()
  0.0        0.011        0.011           4           0          3 MPI_Op_free()
  0.0         0.01         0.01           4           0          2 MPI_Finalized()
  0.0        0.009        0.009          20           0          0 MPI_Comm_rank()
  0.0        0.003        0.003          12           0          0 MPI_Comm_size()
  0.0        0.003        0.003           4           0          1 MPI_Type_size()
```

The output result shown above recorded which MPI operations are used during model training. MPI_Allreduce() has the most number of calls which is 42929 and 44495 respectively for batch size 128 and batch size 64. MPI_Allreduce() also consist most of time spent on communication. For batch size 64 it used 515 milliseconds cumulatively and for batch size 128 it used 513 milliseconds. MPI_Bcast() was called 37 times for training batch size 128, and 35 times for training batch size 64. They both used around 0.18 milliseconds for the whole training process, with 0.186 milliseconds for training batch size 128, and 0.182 milliseconds for batch size 128.