MPI4cuML 0.5 User Guide

HIGH-PERFORMANCE DEEP LEARNING TEAM
http://hidl.cse.ohio-state.edu

NETWORK-BASED COMPUTING LABORATORY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
THE OHIO STATE UNIVERSITY

Last revised: November 10, 2022

Contents

1 Overview of the MPI4cuML Project

MPI4cuML is a custom version of the cuML machine learning library that contains high-performance MPI communication backend for cuML. The MPI backend in MPI4cuML uses mpi4py over the MVAPICH2-GDR library and targets modern HPC clusters built with GPUs.

MPI4cuML provides point-to-point asynchronous I/O communication *coroutines*, which are non-blocking concurrent operations defined using the `async`/`await` keywords from the Python's `asyncio` framework. If there are any questions, comments or feedback regarding this software package, please post them to panda@cse.ohio-state.edu.

2 Features

High-level features of MPI4cuML are listed below. New features and enhancements are marked as (NEW).

- (NEW) Based on cuML 22.02.00
  - (NEW) Include ready-to-use examples for KMeans, Linear Regression, Nearest Neighbors, and tSVD
- (NEW) MVAPICH2 support for RAFT 0.5
  - (NEW) Enabled cuML's communication engine, RAFT, to use MVAPICH2-GDR backend for Python and C++ cuML applications
    - KMeans, PCA, tSVD, RF, LinearModels
  - (NEW) Added switch between available communication backends (MVAPICH2 and NCCL)
- Built on top of mpi4py over the MVAPICH2-GDR library
- Tested with
  - Mellanox InfiniBand adapters (FDR and HDR)
  - Various multi-core platforms (AMD and Intel)
  - (NEW) NVIDIA A100, V100, and P100 GPUs

3 Setup Instructions

3.1 Prerequisites

1. Install Miniconda `https://docs.conda.io/en/latest/miniconda.html`, which is a free minimal installer for the conda package manager. The following commands download and install latest version of Miniconda3 for Python 3.x:

```
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

$ chmod a+x Miniconda3-latest-Linux-x86_64.sh

$ ./Miniconda3-latest-Linux-x86_64.sh -b -p $PWD/miniconda3
```

2. After installing the conda package manager, create a new conda environment named mpi4cuml (suggested name). A tutorial on managing conda environments can be seen at https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html.

```
$ conda create -n mpi4cuml python=3.8

$ conda activate mpi4cuml
```

3. Make sure that relevant/required modules are loaded in the environment. One of the required modules is the CUDA toolkit (11.2 in this case). A list of all installed modules on the system can be seen as follows:

```
$ module list
Currently Loaded Modulefiles:
 1) cuda/11.2
```

4. Download the MVAPICH2-GDR library from http://mvapich.cse.ohio-state.edu. Detailed installation instructions are available from http://mvapich.cse.ohio-state.edu/userguide/gdr/. Please become familiar with advanced configurations of the MVAPICH2-GDR software—this is important to get maximum performance from the hardware system. This guide assumes that an environment variable $MV2_HOME points to the installation directory of the MVAPICH2-GDR software.

```
$ export MV2_HOME=/path/to/MVAPICH2-GDR/install/directory
```

It is important to make sure that MVAPICH2-GDR is the MPI library used with MPI4cuML. Other installations of MPI libraries can result in errors. In order to avoid such situations, it is important to make sure that the correct version of the MVAPICH2-GDR library is available in the $PATH variable. This can be checked as follows:

```
$ $MV2_HOME/bin/mpiname -a
```

If the $PATH is set correctly, the output of the following command should match the previous command output.

```
$ mpiname -a
```

5. Download and install the mpi4py library. Details about mpi4py can be seen at https://mpi4py.readthedocs.io. Download the software as follows:

```
$ git clone https://github.com/mpi4py/mpi4py.git

$ cd mpi4py
```

As part of the installation, the configuration file (mpi.cfg) in the root directory of mpi4py software needs to be updated to point to the MVAPICH2-GDR library. It is recommended to make a new section—in the mpi.cfg file—for MVAPICH2-GDR as follows:

```
# MVAPICH2-GDR
# ------------
[MVAPICH2-GDR]
mpi_dir = /path/to/MVAPICH2-GDR/install/directory
mpicc = %(mpi_dir)s/bin/mpicc
mpicxx = %(mpi_dir)s/bin/mpicxx
include_dirs = %(mpi_dir)s/include
library_dirs = %(mpi_dir)s/lib
runtime_library_dirs = %(library_dirs)s
```

After updating the mpi.cfg configuration file, install mpi4py with following commands:

```
$ python setup.py build --mpi=MVAPICH2-GDR

$ pip install .
```

3.2 Download

The latest version of MPI4cuML package can be downloaded from http://hidl.cse.ohio-state.edu/download/hidl/cuml/. This contains the following items:

1. The cuml folder that is an enhanced version of the cuML library with new examples and documentation—this is what we refer to as MPI4cuML
2. The raft folder that is an enhanced version of the RAFT library with MPI communication backend
3. The envs folder containing available conda environments
4. This user-guide

3.3 Unpacking and Creating Conda Environment

1. Unzip the unified MPI4cuML and MPI4RAFT distribution tarball:

   ```
   $ wget http://hidl.cse.ohio-state.edu/download/hidl/cuml/mpi4cuml-0.5.tar.gz

   $ tar -xzvf mpi4cuml-0.5.tar.gz
   ```

2. Change directory to envs:

   ```
   $ cd mpi4cuml-0.5/conda_envs
   ```

3. Install pre-requisite libraries including build tools, Python (version 3.8 here), Cython, CUDA tookit (version 11.2 here), Dask packages, RAPIDS libraries, and array packages. The conda environment may be installed as follows:

```
$ conda env update -n mpi4cuml --file=mpi4cuml_dev_cuda11.2.yml
```

4. Add conda static and dynamic libs to environment:

```
$ export LIBRARY_PATH=/path/to/miniconda3/envs/mpi4cuml/lib:$LIBRARY_PATH
$ export LD_LIBRARY_PATH=/path/to/miniconda3/envs/mpi4cuml/lib:$LIBRARY_PATH
```

3.4 Installing Base MPI4RAFT Package

1. Change directory to raft:

```
$ cd mpi4cuml-0.5/raft
```

2. Install RAFT package at C++ layer:

```
$ PARALLEL_LEVEL=16 ./build.sh
```

3. Install RAFT package at Python layer:

```
$ cd python
$ python setup.py build_ext --inplace
$ python setup.py install
```

4. At this time, RAFT with all its dependencies should be installed. Verify this by running the following commands:

```
$ conda list

$ conda list | grep raft
```

3.5 Installing Base MPI4cuML Package

As part of this step, an enhanced version of the cuML library with ready-to-use multinode examples and updated documentation commands:

1. Change directory to unpacked MPI4cuML:

```
$ cd mpi4cuml-0.5/cuml
```

2. Install cuML package at Python and C++ layers:

```
$ RAFT_PATH=path_to_raft CUDA_HOME=path_to_cuda_home PARALLEL_LEVEL=16 ./build.sh
```

3. At this time, MPI4cuML with all its dependencies should be installed. Verify this by running the following commands:

```
$ conda list

$ conda list | grep cuml
```

4 Basic Usage Instructions

The MPI4cuML package contains sample scripts at both the Python and C++ layers to test and benchmark performance of MPI4cuML. These applications include:

- Python layer: RF, KMeans, Nearest Neighbors, Linear Regression, tSVD
- C++ layer: KMeans

4.1 Python Layer: RF, KMeans, and Nearest Neighbors

These benchmarks generate synthetic (random) data via native cuML functions `make_blobs` and `make_classification`. Each application's `fit()` function is then applied to this random data. These benchmarks were created from the general template found in the cuML repository https://github.com/rapidsai/cuml/blob/branch-0.18/notebooks/kmeans_mnmg_demo.ipynb.

Write the hosts files. This file contains names of the compute nodes, with GPUs, that will execute the parallel cuML program.

```
$ cd mpi4cuml-0.5/cuml/mnmg_scripts/

$ vim hosts
.. write name of the compute nodes ..

$ cat hosts
machine1
machine2
machine3
machine4
```

Users will need to customize the hosts file as per their environment. This benchmark can be executed as follows (Note that LD_PRELOAD is a single line without spaces):

```
$ LD_PRELOAD=$MV2_HOME/lib/libmpi.so:$CUML_HOME/cpp/build/libcuml++.so:\
  $CUML_HOME/python/cuml/dask/cluster/mv2_support.cpython-37m-x86_64-linux-gnu.so
  $MV2_HOME/bin/mpirun_rsh -export-all -np 4 -hostfile hosts MV2_USE_CUDA=1
      MV2_USE_GDRCOPY=1 MV2_GPUDIRECT_GDRCOPY_LIB=/opt/gdrcopy2.0/lib64/libgdrapi.so
      python kmeans_mnmg.py
```

This command is starting the execution of the cuML application using MPI—this is enabled by the MPI4cuML package. A total of 4 MPI processes are started with process 0 and 1 assuming the role of Dask scheduler and client respectively. All other processes with ranks greater than 1 becoming Dask workers. Here processes 2 and 3 are Dask workers. The `LD_PRELOAD` environment variable is required for Python applications to run correctly with the MVAPICH2-GDR library.

Details on various options, passed to the MVAPICH2-GDR library, are as follows:

- `-np 4` — specifies the total number of parallel processes started by the MPI library

- `-hostfile hosts` — specifies the names of machines where to start parallel processes in the "hosts" file
- `MV2_USE_CUDA=1` — ensures that MVAPICH2-GDR library supports GPU-to-GPU communication
- `MV2_USE_GDRCOPY=1` — turns on the GDRCopy protocol
- `MV2_GPUDIRECT_GDRCOPY_LIB` — specifies path to the GDRCopy dynamic library
- `MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER` — selects a socket-level scatter process binding policy

This is not an exhaustive list of options that can be passed to the MVAPICH2-GDR library. Details on these—and other flags for the MVAPICH2-GDR library—can be seen at: http://mvapich.cse.ohio-state.edu/userguide/gdr/. As mentioned earlier, it is important to configure the MVAPICH2-GDR installation correctly to ensure optimal performance.

The <application-name>_mnmg.py scripts have some configuration parameters that can be modified, in the source-code, by the users. These include `HOSTS`, which specifies the hostfile created above, and the hyperparameters for each application. Please note that the benchmark runtime is highly dependent on the hyperparameters, GPU architecture, and interconnect. Therefore, the user should expect to perform some tuning before collecting reasonable benchmark runtimes.

4.2 C++ Layer: KMeans

This benchmark provides an example of distributed KMeans via MVAPICH2-GDR at the C++ layer of cuML. This benchmark is adapted from the cuML package repository https://github.com/rapidsai/cuml/tree/branch-0.18/cpp/examples/kmeans. This application must first be built as follows:

```
$ cd mpi4cuml-0.5/cuml/cpp/examples/mg_kmeans

$ cmake .. -DCUML_LIBRARY_DIR=/path/to/directory/with/libcuml.so
    -DCUML_INCLUDE_DIR=/path/to/directory/with/kmeans/kmeans_c.h

$ make
```

The application can then be executed with tiny test input as follows:

```
$ LD_PRELOAD=$MV2_HOME/lib/libmpi.so:$CUML_HOME/cpp/build/libcuml++.so
    $MV2_HOME/bin/mpirun_rsh -export-all -np 4 -hostfile hosts MV2_USE_CUDA=1
    MV2_USE_GDRCOPY=1 MV2_GPUDIRECT_GDRCOPY_LIB=/opt/gdrcopy2.0/lib64/libgdrapi.so
    ./mg_kmeans_example
```

KMeans can also be run with a larger test dataset from Kaggle (https://www.kaggle.com/c/homesite-quote-conversion/data):

```
$ unzip all.zip
$ ./prepare_input.py [train_file=train.csv] [test_file=test.csv] [output=output.txt]
Reading Input from train_file = train.csv and test_file = test.csv
Training dataset dimension: (260753, 299)
Test dataset dimension: (173836, 298)
```

```
Output dataset dimension: (260753, 298)
Wrote 77704394 values in row major order to output output.txt
```

Once the dataset has been prepared, it can be run with KMeans with the following command

```
$ LD_PRELOAD=$MV2_HOME/lib/libmpi.so:$CUML_HOME/cpp/build/libcuml++.so
    $MV2_HOME/bin/mpirun_rsh --export-all -np 4 -hostfile=hosts MV2_USE_CUDA=1
    MV2_USE_GDRCOPY=1 MV2_GPUDIRECT_GDRCOPY_LIB=/opt/gdrcopy2.0/lib64/libgdrapi.so
    ./mg_kmeans_example -num_rows 260753 -num_cols 298 -input output.txt

Reading input with 260753 rows and 298 columns from output.txt.
Run KMeans with k=10, max_iterations=300
  num_pts inertia
0 18615 7.749915e+12
1 18419 7.592070e+12
2 30842 1.815066e+13
3 31247 1.832832e+13
4 31272 1.887647e+13
5 18362 7.749335e+12
6 31028 1.821217e+13
7 31040 1.869879e+13
8 18652 7.681686e+12
9 31276 1.877210e+13
Global inertia = 1.418115e+14
```

5 Troubleshooting with MPI4cuML

If you are experiencing any problems with , please feel free to contact us by sending an email to hidl-discuss@lists.osu.edu.