DS 7347
High-Performance Computing (HPC) and Data Science
Session 27

Robert Kalescky
Adjunct Professor of Data Science
HPC Research Scientist

July 26, 2022

Research and Data Sciences Services
Office of Information Technology
Center for Research Computing
Southern Methodist University

Outline

Session Questions

What are bandwidth and latency? Provide a nontechnical example.

Message Passing Interface (MPI)

Motivation

- MPI is a specification of what the interface should look like and what it should do
  - Separate processes on each node communicate by sending and receiving data over a network
  - MPI can be used for parallelism on a single node as well
- An MPI implementation is a set of libraries that allow for multiple nodes to be used together via message passing
- Many higher-level languages and libraries support or use MPI
- MPI has become the industry standard for distributed-memory programming

MPI Implementations

- Open source
  - MPICH
  - OpenMPI
  - MVAPICH
- Closed source
  - Intel MPI (based on MPICH)
  - Mellanox HPC-X MPI (based on OpenMPI)

Compiling MPI Programs

Depending on your programming language and the specific MPI implementation, these wrapper scripts can have different names

- C++: mpicxx or mpic++
- C: mpicc
- Fortran 90/95/2003: mpif90
- Fortran 77: mpif77

Running MPI Programs

- Running MPI batch jobs on ManeFrame is almost identical to running serial and OpenMP batch jobs. However, when running MPI jobs, we must tell the queueing system a few additional pieces of information:
  1. How many total nodes we want to reserve on the machine?
  2. How many total cores do we want to reserve on the machine?
  3. How do you want to distribute tasks on each node?
  4. How many MPI tasks do you actually want to run?
- We have two key ways to control execution of parallel batch jobs:
  - Controlling how the job is reserved
  - Controlling how the MPI job is executed

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mpi.h"

double random_double(void);
double get_hits(uint64_t points);
```

Listing 1: Includes and function prototypes

Estimating π

```
35 double random_double(void) {
36    double n = rand();
37    return n / (double)RAND_MAX;
38 }
39
40 double get_hits(uint64_t points) {
41    uint64_t hits = 0;
42    for (uint64_t i = 0; i < points; i++) {
43       if (pow(random_double(), 2) + pow(random_double(), 2) < 1) {
44          hits++;
45       }
46    }
47    return hits;
48 }
```

Listing 2: Function definitions.

Estimating π

```
9  int main(int argc, char *argv[]) {
10    if (argc != 2) {
11       printf("Requires number of points.");
12       return 1;
13    }
14    int num_tasks, rank, rc;
15    uint64_t n, points, task_hits, hits;
```

Listing 3: Argument check and variable initialization.

```
16      rc = MPI_Init(&argc, &argv);
17      if (rc != 0) printf("Problem with MPI initialization.");
18      rc = MPI_Comm_size(MPI_COMM_WORLD, &num_tasks);
19      if (rc != 0) printf("Problem with getting size of MPI_COMM_WORLD.");
20      rc = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
21      if (rc != 0) printf("Problem with getting process rank.");
```

Listing 4: Initialize MPI, get the global number of tasks, and get the process rank.

Estimating π

```
22      n = (uint64_t)atoi(argv[1]);
23      points = (uint64_t)floor((double)n / (double)num_tasks);
24      if (rank == 0) points += (uint64_t)(n % num_tasks);
25      task_hits = get_hits(points);
26      rc = MPI_Reduce(&task_hits, &hits, 1, MPI_UINT64_T, MPI_SUM, 0,
    ↪  MPI_COMM_WORLD);
27      if (rc != 0) printf("Problem with reduction.");
```

Listing 5: Get number of hits per rank via summation reduction.

```
28      if (rank == 0) {
29         printf("Estimated Pi: %f\n", 4.0 * (double)hits / (double)n);
30      }
31      rc = MPI_Finalize();
32      if (rc != 0) printf("Problem with finalization.");
33  }
```

Listing 6: If rank zero, report estimation of π. All ranks finalize and exit.

Estimating π

```
#!/bin/bash
#SBATCH -J mpi_pi             # Job name
#SBATCH -o mpi_pi_%j.out      # Output file name
#SBATCH -p development        # Queue (partition)
#SBATCH -N 2                  # Nodes
#SBATCH --ntasks-per-node=2   # Tasks/node
#SBATCH --mem=6G              # Memory
#SBATCH -t 5                  # Time limit
```

Listing 7: Reguest compute resources.

```
10  echo $SLURM_JOB_PARTITION
11
12  module purge
13  module load nvhpc-22.2       # Alternatively nvhpc-21.2, nvhpc-21.9
```

Listing 8: Setup environment.

```
15  mpicc -o mpi_monte_carlo_pi mpi_monte_carlo_pi.c
16
17  srun mpi_monte_carlo_pi
```

Listing 9: Build the executable and run.

MPI for Python

Installing `mpi4py`

- `mpi4py` should be installed with `pip` when possible
- `Conda` installed versions will likely be built with an unoptimized version of MPI
- There can be significant performance impacts, especially for communication bound applications

Minimal example of **mpi4py** using **Conda**

```
# load version of Python that has Conda in it
module load python/3

# create a virtual environment named conda_mpi4py that uses Python 3.9
# and installs mpi4py
conda create -p $HOME/conda_mpi4py mpi4py python=3.9
```

Minimal example of mpi4py using pip

```
# load Intel compilers, which include a version of Python and an MPI compiler
module load intel

# create a virtual environment named venv_mpi4py
python -m venv $HOME/venv_mpi4py

# upgrade pip
pip install --upgrade pip

# make sure we're using the correct MPI compiler
export MPICC=$(which mpicc)

# install mpi4py, the flags --no-binary :all: --compile
# tell pip not to use a precompiled version
pip install --no-binary :all: --compile mpi4py
```

Bandwidth Test

We'll use a simple bandwidth test available here
https://github.com/felker/mpi4py_benchmark

Important commands:

Import the `mpi4py` libraries

```
from mpi4py import MPI
```

Creating a communicator and getting ranks:

```
comm = MPI.COMM_WORLD
myid = comm.Get_rank()
numprocs = comm.Get_size()
```

Create send and receive buffers

```
# create the message. [data_buffer size type]
if (id == 0):
        s_msg = [s_buf, size, MPI.BYTE]
        r_msg = [r_buf,    4, MPI.BYTE]
elif (if == 1):
         s_msg = [s_buf, 4, MPI.BYTE]
         r_msg = [r_buf,    size, MPI.BYTE]
```

Send and receive requests

```
if (id == 0):
        requests = comm.Isend(s_msg, 1, 100)
        MPI.Request.Waitall(requests)
        comm.Recv(r_msg, 1, 101)
elif (id == 1):
        requests = comm.Irecv(r_msg, 0, 100)
        MPI.Request.Waitall(requests)
        comm.Send(s_msg, 0, 101)
```

Running with Conda mpi4py

```
#!/bin/bash
#SBATCH --partition=standard-mem-s    # use standarm memory short queue
#SBATCH -N 2                          # request 2 nodes
#SBATCH -o conda_mpi4py_test.txt      # specify output location
#SBATCH --ntasks-per-node=1           # request 1 task on each node
#SBATCH -t 00:05:00                   # request 5 minutes
#SBATCH --mem=1G                      # request 1 GB

# load modules
module purge
module load python/3

# activate the environment
eval "$(conda shell.bash hook)"
conda activate $HOME/conda_mpi4py

# run test
mpirun python osu_bw.py
```

Running with venv mpi4py

```
#!/bin/bash
#SBATCH --partition=standard-mem-s  # use standarm memory short queue
#SBATCH -N 2                        # request 2 nodes
#SBATCH -o pip_mpi4py_test.txt      # specify output location
#SBATCH --ntasks-per-node=1         # request 1 task on each node
#SBATCH -t 00:05:00                 # request 5 minutes
#SBATCH --mem=1G                    # request 1 GB

# load modules
module purge
module load intel

# activate environment
source $HOME/venv_mpi/bin/activate

# run test
srun python osu_bw.py
```

Example Output

```
# MPI Bandwidth Test
# Size [B]    Bandwidth [MB/s] (pip)    Bandwidth [MB/s] (conda)
65536                        10231.65                     1502.74
131072                       10927.86                     1837.19
262144                       11226.46                     2112.31
524288                       11558.91                     2187.85
1048576                      11815.73                     2234.20
2097152                      11926.97                     2239.13
4194304                      11996.23                     2008.81
8388608                      11816.54                     1826.22
16777216                     11337.77                     1806.65
33554432                     11202.05                     1781.83
67108864                     11122.19                     1818.52
134217728                    11073.34                     1710.29
```

NVIDIA Collective Communications Library (NCCL)

NCCL

- NCCL is a communication library designed primarily for inter-GPU communications
- It is not (currently) a stand alone parallel programming framework
- Generally, NCCL uses is very similar to MPI

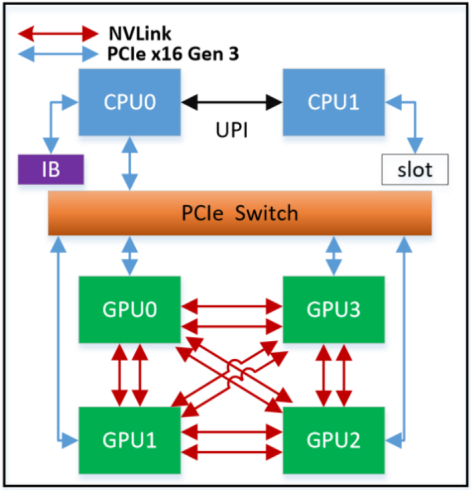

Figure 1: An example of a possible multi-GPU node configuration

NCCL Collective Operations

```
1  // AllReduce
2  ncclResult_t ncclAllReduce(const void* sendbuff, void* recvbuff, size_t count,
   ↪  ncclDataType_t datatype, ncclRedOp_t op, ncclComm_t comm, cudaStream_t stream)
3
4  // Broadcast
5  ncclResult_t ncclBroadcast(const void* sendbuff, void* recvbuff, size_t count,
   ↪  ncclDataType_t datatype, int root, ncclComm_t comm, cudaStream_t stream)
6
7  // Reduce
8  ncclResult_t ncclReduce(const void* sendbuff, void* recvbuff, size_t count,
   ↪  ncclDataType_t datatype, ncclRedOp_t op, int root, ncclComm_t comm,
   ↪  cudaStream_t stream)
9
10 // AllGather
11 ncclResult_t ncclAllGather(const void* sendbuff, void* recvbuff, size_t sendcount,
   ↪  ncclDataType_t datatype, ncclComm_t comm, cudaStream_t stream)
12
13 // ReduceScatter
14 ncclResult_t ncclReduceScatter(const void* sendbuff, void* recvbuff, size_t
   ↪  recvcount, ncclDataType_t datatype, ncclRedOp_t op, ncclComm_t comm,
```

NCCL Point to Point Operations

Note, these are non-blocking.

```
1  // Send
2  ncclResult_t ncclSend(const void* sendbuff, size_t count, ncclDataType_t datatype,
   ↪  int peer, ncclComm_t comm, cudaStream_t stream)
3
4  // Receive
5  ncclResult_t ncclRecv(void* recvbuff, size_t count, ncclDataType_t datatype, int
   ↪  peer, ncclComm_t comm, cudaStream_t stream)
```

General Flow

```
1  // initialize MPI
2  int myRank, nRanks, localRank = 0;
3  MPI_Init(&argc, &argv);
4  MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
5  MPI_Comm_size(MPI_COMM_WORLD, &nRanks);
6
7  // initialize NCCL
8  int nDev = 4;
9  int devs[4] = { 0, 1, 2, 3 };
10 ncclComm_t comms[4];
11 ncclCommInitAll(comms, nDev, devs);
```

General Flow

```
1  // allocate and initializing device buffers
2  float** sendbuff = (float**)malloc(nDev * sizeof(float*));
3  float** recvbuff = (float**)malloc(nDev * sizeof(float*));
4  cudaStream_t* s = (cudaStream_t*)malloc(sizeof(cudaStream_t)*nDev);
5
6
7  for (int i = 0; i < nDev; ++i) {
8          cudaSetDevice(i);
9          cudaMalloc(sendbuff + i, size * sizeof(float));
10         cudaMalloc(recvbuff + i, size * sizeof(float));
11         cudaMemset(sendbuff[i], 1, size * sizeof(float));
12         cudaMemset(recvbuff[i], 0, size * sizeof(float));
13         cudaStreamCreate(s+i);
14 }
```

General Flow

```
1  // Do some NCCL communication. Group API is required when using
2  // multiple devices per thread
3  ncclGroupStart();
4  for (int i = 0; i < nDev; ++i)
5           ncclAllReduce((const void*)sendbuff[i], (void*)recvbuff[i], size,
   ↪  ncclFloat, ncclSum,        comms[i], s[i]);
6  ncclGroupEnd();
7
8
9  // Make sure operations are synchronized by waiting for stream to finish
10 for (int i = 0; i < nDev; ++i) {
11         cudaSetDevice(i);
12         cudaStreamSynchronize(s[i]);
13 }
```

General Flow

```
1  // free device buffers
2  for (int i = 0; i < nDev; ++i) {
3          cudaSetDevice(i);
4          cudaFree(sendbuff[i]);
5          cudaFree(recvbuff[i]);
6  }
7
8
9  // finalizing NCCL
10 for(int i = 0; i < nDev; ++i)
11         ncclCommDestroy(comms[i]);
12
13 // finalize MPI
14 MPI_Finalize()
```

Example on M2

Run a simple benchmark on M2 from
https://github.com/NVIDIA/nccl-tests

```
# get repository
git clone git@github.com:NVIDIA/nccl-tests.git

# change directory to the repository
cd nccl-tests

# load NVHPC module
module load nvhpc-21.9

# build with MPI enabled
make MPI=1
↪  CUDA_HOME=/hpc/applications/nvidia/hpc_sdk/2021_21.9/Linux_x86_64/21.9/cuda/11.4/
```

Run on 4 P100 nodes

```
#!/bin/bash
#SBATCH --partition=gpgpu-1     # use gpgpu-1 queue
#SBATCH -N 4                    # request 4 nodes
#SBATCH -o p100_nccl_test.txt   # specify output location
#SBATCH --ntasks-per-node=1     # request 1 task on each node
#SBATCH -t 00:05:00             # request 5 minutes
#SBATCH --mem=10G               # request 10 GB
#SBATCH --gres=gpu:1            # request 1 gpu per task

# load modules
module purge
module load nvhpc-21.9

srun ./build/all_reduce_perf -b 8 -e 128M -f 2 -g 1
```

Run on 4 P100 nodes

```
# nThread 1 nGpus 1 minBytes 8 maxBytes 134217728 step: 2(factor) warmup iters: 5 iters: 20 validation:
#
#                                                     out-of-place                       in-place
#       size         type   redop     time   algbw   busbw  error     time   algbw   busbw
#        (B)                             (us)  (GB/s)  (GB/s)            (us)  (GB/s)  (GB/s)
           8         float     sum    23.75    0.00    0.00  1e-07    22.92    0.00    0.00
          16         float     sum    23.53    0.00    0.00  3e-08    24.04    0.00    0.00
          32         float     sum    23.03    0.00    0.00  3e-08    23.15    0.00    0.00
          64         float     sum    23.24    0.00    0.00  3e-08    23.88    0.00    0.00
         ...           ...     ...      ...     ...     ...    ...      ...     ...     ...
     8388608         float     sum   1461.1    5.74    8.61  2e-07   1463.1    5.73    8.60
    16777216         float     sum   2892.3    5.80    8.70  2e-07   2886.7    5.81    8.72
    33554432         float     sum   5733.4    5.85    8.78  2e-07   5733.4    5.85    8.78
    67108864         float     sum    11448    5.86    8.79  2e-07    11440    5.87    8.80
   134217728         float     sum    22765    5.90    8.84  2e-07    22753    5.90    8.85
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 3.1374
#
```

Run on 2 V100s on 2 nodes

```
#!/bin/bash
#SBATCH --partition=v100x8  # use the v100x8 queue
#SBATCH -N 2                # request 2 nodes
#SBATCH -o v100_2x2.txt     # specify output location
#SBATCH --ntasks-per-node=1 # request 1 task on each node
#SBATCH -t 00:05:00         # request 5 minutes
#SBATCH --mem=10G           # request 10 GB
#SBATCH --gres=gpu:2        # request 2 gpus per task

# load modules
module purge
module load nvhpc-21.9

srun ./build/all_reduce_perf -b 8 -e 128M -f 2 -g 2
```

Run on 2 V100s on 2 nodes

```
# nThread 1 nGpus 2 minBytes 8 maxBytes 134217728 step: 2(factor) warmup iters: 5 iters: 20 validation: 1
#
#                                                     out-of-place                       in-place
#       size         type      redop     time   algbw   busbw  error     time   algbw   busbw
#        (B)                              (us)  (GB/s)  (GB/s)           (us)  (GB/s)  (GB/s)
            8     float     sum    32.30    0.00    0.00  1e-07    32.13    0.00    0.00
           16     float     sum    32.25    0.00    0.00  3e-08    32.55    0.00    0.00
           32     float     sum    837.4    0.00    0.00  3e-08    104.6    0.00    0.00
           64     float     sum    375.6    0.00    0.00  3e-08    32.43    0.00    0.00
          ...       ...     ...      ...     ...     ...    ...      ...     ...     ...
      8388608     float     sum   1444.0    5.81    8.71  1e-07   1447.4    5.80    8.69
     16777216     float     sum   2727.5    6.15    9.23  1e-07   2684.4    6.25    9.37
     33554432     float     sum   5111.7    6.56    9.85  1e-07   5120.8    6.55    9.83
     67108864     float     sum    10018    6.70   10.05  1e-07   9979.1    6.72   10.09
    134217728     float     sum    19611    6.84   10.27  1e-07    19474    6.89   10.34
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 3.17035
#
```

Run on 4 V100s on 1 node

```
#!/bin/bash
#SBATCH --partition=v100x8  # use the v100x8 queue
#SBATCH -N 1                # request 1 node
#SBATCH -o v100_4x1.txt     # specify output location
#SBATCH --ntasks-per-node=1 # request 1 task on each node
#SBATCH -t 00:05:00         # request 5 minutes
#SBATCH --mem=10G           # request 10 GB
#SBATCH --gres=gpu:4        # request 4 gpus per task

# load modules
module purge
module load nvhpc-21.9

srun ./build/all_reduce_perf -b 8 -e 128M -f 2 -g 4
```

Run on 4 V100s on 1 node

```
# nThread 1 nGpus 4 minBytes 8 maxBytes 134217728 step: 2(factor) warmup iters: 5 iters: 20 validation: 1
#
#                                                     out-of-place                       in-place
#       size         type      redop     time   algbw   busbw  error     time   algbw   busbw
#        (B)                              (us)  (GB/s)  (GB/s)           (us)  (GB/s)  (GB/s)
           8     float     sum    14.82    0.00    0.00  1e-07    14.78    0.00    0.00
          16     float     sum    15.08    0.00    0.00  3e-08    15.06    0.00    0.00
          32     float     sum    15.10    0.00    0.00  3e-08    15.16    0.00    0.00
          64     float     sum    15.31    0.00    0.01  3e-08    15.20    0.00    0.01
         ...       ...     ...      ...     ...     ...    ...      ...     ...     ...
     8388608     float     sum    307.4   27.29   40.94  2e-07    307.5   27.28   40.92
    16777216     float     sum    584.4   28.71   43.06  2e-07    584.6   28.70   43.05
    33554432     float     sum   1135.3   29.56   44.33  2e-07   1135.1   29.56   44.34
    67108864     float     sum   2251.0   29.81   44.72  2e-07   2250.5   29.82   44.73
   134217728     float     sum   4479.3   29.96   44.95  2e-07   4484.6   29.93   44.89
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 13.581
#
```

Assignments and Project

Assignments and Labs

- Complete and commit all assignments and labs by Tuesday, July 26 to receive grades
- All assignments are listed in the README
- Available for office hours by request and 30 minutes before and after Thursday's (07/21/22) session

Project

- Implement initial improvements for your three optimization targets to discuss at next Thursday's (07/28/22) session
- Available for office hours by request and 30 minutes before and after Thursday's (07/21/22) and Tuesday's (07/26/22) sessions
- Meet with each individual to discuss optimization targets