CX 4220/CSE 6220 High Performance Computing (Fall 2024)
Programming Assignment 2
Due: Oct 18

1 Problem Statement

In this assignment you will develop your own MPI communication primitives in C++. You are expected to write your code from scratch and you can only make use of basic MPI instructions MPI_Send/MPI_Isend, MPI_Recv/MPI_Irecv or MPI_Sendrecv.

2 MPI Communication Primitives

1. **Scatter -** Scatter equally distributes the given data from the root processor to all the processors in the communication network.
   - Should work on any number of processors (correctness test on 16 and 24 processors).
   - Should work with any processor as root processor.
2. **AllGather -** AllGather collects the data from all processors and the collected data is present with all the processors. The collection of data is appended according to the processor rank (0 to p - 1).
   - Should work on any number of processors (correctness test on 16 and 24 processors).
   - All processors should have all the data and sorted based on processor rank.
3. **AllReduce -** AllReduce reduces the data based on the suggested operation and the result is present in every processor. The reduction operation is done element-wise.
   - Should work on any number of processors (correctness test on 16 and 24 processors).
   - The reduction operation is "MPI_SUM"
4. **Alltoall -** Each processor sends a chunck of its data (equal size) to every other processor. The received data in every processor is appended based on the rank of the sending processor (0 to p-1).
   (a) **Arbitrary** communication style
      - Should work on any number of processors (correctness test on 16 and 24 processors).
   (b) **Hypercubic** communication style
      - Should work on 2^k processors.

3 Code Framework

3.1 Input & Output Format

Input: We will have `problem_size` and `root_rank` as two command-line arguments respectively. The processor with rank `root_rank` should scatter the data to the remaining processors. Other custom functions will not use `root_rank`.

```
0 <= root_rank < COMM_SIZE
```

Output: The program will print the output which will state the correctness of your implementations.

3.2 Download Files

Download the PA2 files from this link **(or)**
Use `git clone https://github.com/gtcse6220-fall24/pa2.git`

1. **Makefile**
2. **custom_collectives.cpp** - Develop your code in this file. This is your **submission file**. **Note:-** You need to only need to submit the **custom_collectives.cpp** file.
3. **main.cpp** - The driver file for your code. You can make changes in the driver file for your own testing and custom test cases, but make sure that the submitted 'custom_collectives.cpp' file runs with the original 'main.cpp' file
4. **custom_collectives.h** - This file contains the function definitions and you should not modify the file.
5. **autograder.sh** - This file can be used to check whether you are within the expected performance bounds. Before submitting, make sure your code passes the tests to obtain the credits.

3.2.1 Instructions

1. After downloading all the files, go through each of the files and try to understand the structure.
2. Write your implementation of all the custom functions in `custom_collectives.cpp` file.
3. Use `make` to compile the program. An executable **primitives** will be created if there are no compilation errors.
4. Request the resources (interactive node) as discussed in the review session (using `salloc`) and use `module load openmpi` to load the MPI library (which will solve the include/command not found errors, if any).

5. Then use `srun -n <num_processors> ./primitives -<flag> <problem_size> <root_rank>` to run the program.

   `<flag>` options :
   `s` for Scatter
   `g` for Allgather
   `r` for Allreduce
   `a` for Alltoall_Arbitrary
   `h` for Alltoall_Hypercubic

3.3 Deliverables

1. **custom_collectives.cpp** - Make sure this file runs with the original 'main.cpp' file.

2. **Report** - Make sure to list names of all your teammates at the very beginning of your report.

   - Please explain the working of all your custom implementations in short.
   - Do the theoretical runtime analysis for all your custom implementations which includes both computation and communication cost.
   - Plot the runtime against varying problem sizes $n = 10^4, 10^5, 10^6, 10^7$ and 10^8 for fixed number of processors $p = 16$ for each of the custom implementations.

3.4 Grading

- Code

  NOTE: We will be running your submissions on a Intel(R) Xeon(R) Gold 6226 CPU @ 2.70GHz (24 cores) node with OpenMPI Version 4.1.5

  All the performance and correctness tests will be run on the problem size $n = 10^8$. You can see all the tests and their bounds/relaxations (based on TA implementations) in the autograder.

  - Custom_Scatter - **10**
    - 2.5 points - Correctness on 16 procs (arbitrary root)
    - 2.5 points - Correctness on 24 procs (arbitrary root)
    - 5 points - Passing the runtime test on 16 procs (Less than 25 times of MPI runtime)
  - Custom_Allgather - **10**
    - 2.5 points - Correctness on 16 procs
    - 2.5 points - Correctness on 24 procs
    - 5 points - Passing the runtime test on 16 procs (Less than 3 times of MPI runtime)
  - Custom_Allreduce - **10**
    - 2.5 points - Correctness on 16 procs

        - 2.5 points - Correctness on 24 procs
        - 5 points - Passing the runtime test on 16 procs (Less than 10 times of MPI runtime)
    - Custom_Alltoall_Hypercube - **10**
        - 5 points - Correctness on 16 procs
        - 5 points - Passing the runtime test on 16 procs (Less than 10 times of MPI runtime)
    - Custom_Alltoall_Arbitrary - **10**
        - 2.5 points - Correctness on 16 procs
        - 2.5 points - Correctness on 24 procs
        - 5 points - Passing the runtime test on 16 procs (Less than 2 times of MPI runtime)
- Report -
    - Runtime analysis of all custom implementations - **10**

4 Resources

- What is a `Makefile` and how does it work?: `https://opensource.com/article/18/8/what-how-makefile`
- PACE ICE cluster guide: `https://docs.pace.gatech.edu/ice_cluster/ice-guide/`. Documentation for writing a PBS script: `https://docs.pace.gatech.edu/software/PBS_script_guide/`