Parallel Computing/Programming Assignment #3: Point-2-Point Reduction vs. Collective Reduction

Christopher D. Carothers
Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street
Troy, New York U.S.A. 12180-3590
Email: chrisc@cs.rpi.edu or chris.carothers@gmail.com

February 16, 2023

DUE DATE: Friday, March 3rd, 2022, 11:59 p.m.

1 Assignment Description

For this first MPI assignment, your are to develop a point-2-point message version of the `MPI_reduce` operation and compare the performance of that version you develop to the collective version, `MPI_reduce` across a variable number of MPI rank configurations.

1.1 `MPI_P2P_reduce`

Here, you will create a function called `MPI_P2P_Reduce` that takes the same arguments as `MPI_reduce` except that your implementation will only perform the `MPI_SUM` operation and the final reduction result goes to MPI rank 0. So, please ignore the MPI Operation and target arguments to your `MPI_P2P_Reduce`. The core algorithm forms a binary tree of the MPI ranks. The algorithm becomes:

```
MPI_P2P_reduce(...)
{
   1.  Each rank computes sum over local data array.
   2a. Compute pairwise sums using MPI_Isend/Irecv
       between MPI ranks at a stride of 1:
          0 and 1, 2 and 3, 4 and 5, ....n-2 and n-1
       with result going to lower rank id
   2b. Compute pairwise sums using MPI_Isend/Irecv
       between MPI ranks at a stride of 2:
          0 and 2, 4 and 6, ....n-4 and n-2.
       with result going to lower rank id
```

```
   2c. Compute pairwise sums using MPI_Isend/Irecv
       between MPI ranks at a stride of 4:
          0 and 4, 8 and 12, ...n-8 and n-4.
       with result going to lower rank id
   2d, e, ... keep going until Rank 0 has the final sum result.
}
```

As a specific example, consider a 16 MPI rank configuration. There will be $log_2(16)$ steps after the calculation of the local sum at each MPI rank. Note, for below the pair (x,y) means that MPI rank x performs a `MPI_Irecv` for sum data that was sent via `MPI_Isend` from MPI rank y. Upon receipt, MPI rank x will add rank y's value to it running sum.

1. Rank pair lists: (0,1), (2,3), (4,5), (6,7), (8,9), (10,11) ,(12,13) ,(14,15)
2. Rank pair lists: (0,2), (4,6), (8,10), (12,14)
3. Rank pair lists: (0, 4), (8, 12)
4. Rank pair lists: (0, 8)
5. Final sum: 0

Note, that in the case of rank 0 (and other even ranks), pay careful attention to not "over add" previous sums into an already existing sum when you use rank 0 (and other even ranks) result in other subsequent sum calculations.

Use the AiMOS's `clock_now` function from Assignment 2 to measure the number of cycles this point-2-point reduce operation took as well as the MPI collection `MPI_reduce`. Exclude data allocation and initialization in your timing measurements. To use this function make sure you include the following header file on AiMOS and use per the example below.

```
                .
                .
                .
    start_cycles= clock_now();
    MPI_P2P_Reduce(.....);
    end_cycles= clock_now();

    time_in_secs = ((double)(end_cycles - start_cycles)) /
                   clock_frequency;
```

Note, that the `clock_frequency` is 512,000,000 cycles per second. The outputs for the collective and point to point versions should be the sum answer from Rank 0 and the time in seconds (one space between numbers). The first output is for the point2point version (on a separate line and the second is the collective reduce version.

1.2 Data for Reducing

. The data will consist of a single long array of 1 billion (e.g., $2^{30} = 1,073,741,824$) entries of type `MPI_LONG_LONG`. This array will be split equally across ranks and it's data value will be initialized to it's global position index. For example, $bigarray[0] = 0$ while $bigarray[999999999\%elements\_per\_rank] = 999999999$.

This initialization is to be deterministic across all MPI rank configurations. In terms of the answer it is easy to compute via the equation $N * N - 1/2$ which is 576,460,751,766,552,576.

The output of your program (from MPI Rank 0 only!), must print the above correct answer in order to receive full credit for this assignment for BOTH the MPI_Reduce and MPI_P2P_Reduce implementations.

Note, that each AiMOS compute node has 512 GB of RAM. A 32 rank, 1 node run will than have 32 arrays of 32K entries each. So, you should have plenty of RAM for all the experiments.

1.3 Experiments

You will conduct a strong scaling experiment using "AiMOS", the CCI IBM DCS/AC922 supercomputer system. Here, you will have *upto* 32 MPI rank per AiMOS compute-node. For 64, 128 and 256 rank runs, you will need to allocate proportionally more compute nodes per below.

- Run 1 billion entry using 2 MPI ranks (1 AiMOS compute node)
- Run 1 billion entry using 4 MPI ranks (1 AiMOS compute node)
- Run 1 billion entry using 8 MPI ranks (1 AiMOS compute node)
- Run 1 billion entry using 16 MPI ranks (1 AiMOS compute nodes)
- Run 1 billion entry using 32 MPI ranks (1 AiMOS compute nodes)
- Run 1 billion entry using 64 MPI ranks (2 AiMOS compute nodes)
- Run 1 billion entry using 128 MPI ranks (4 AiMOS compute nodes)
- Run 1 billion entry using 256 MPI ranks (8 AiMOS compute nodes)

In your report, you will plot the execution time of each reduction approach as a function of the number of MPI ranks and suggest a reason why the shape of the performance curve for both implementations and why one is faster than the other.

To launch the above experiments, you will need to use the `sbatch` SLURM job scheduling command. Class lecture will review how to use this command as well it is documented on the CCI Wiki. For using Spectrum-MPI, make sure use the `slurmSpectrum.sh` example script to launch your MPI jobs from the CCI Wiki (see: `https://secure.cci.rpi.edu/wiki/Slurm/`.

2 HAND-IN INSTRUCTIONS

Both the report in PDF format and MPI/C-code and headerfiles are to be submitted using the Class Grading System, `submitty.cs.rpi.edu`.