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

February 25, 2019

DUE DATE: Tuesday, March 12th, 2019

1 Assignment Description

For this 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. 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 {\tt 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 {\tt 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 {\tt 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 Irecv for sum data that was sent via 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

Use the Blue Gene/Q's "GetTimeBase()" function 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 the Blue Gene/Q and use per the example below.

```
#define BGQ 1 // when running BG/Q, comment out when running on mastiff

#ifdef BGQ
#include<hwi/include/bqc/A2_inlines.h>
#else
#define GetTimeBase MPI_Wtime
#endif

double time_in_secs = 0;
double processor_frequency = 1600000000.0;
unsigned long long start_cycles=0;
unsigned long long end_cycles=0;
          .
          .
          .
start_cycles= GetTimeBase();
MPI_P2P_Reduce(.....);
end_cycles= GetTimeBase();

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

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.

Note, that each BG/Q node has 16 GB of RAM. A 64 rank, 1 node run will than have 64 arrays of 16,777,216 entries each. However, since each array entry is an 8 byte quantity, it will consume 134,217,728 bytes per rank which is about 1/2 the available memory on each rank. So, use your memory sparingly!! Otherwise, your 1 node runs won't complete due to insufficient memory resources.

1.3 Experiments

You will conduct a strong scaling experiment using "AMOS", the CCI Blue Gene/Q system. Here, you will have 64 MPI rank per Blue Gene/Q node.

- Run 1 billion entry 64 MPI ranks (1 BG/Q nodes)
- Run 1 billion entry 128 MPI ranks (2 BG/Q nodes)
- Run 1 billion entry 256 MPI ranks (4 BG/Q nodes)
- Run 1 billion entry 512 MPI ranks (8 BG/Q nodes)
- Run 1 billion entry 1024 MPI ranks (16 BG/Q nodes)
- Run 1 billion entry 2048 MPI ranks (32 BG/Q nodes)
- Run 1 billion entry 4096 MPI ranks (64 BG/Q nodes)
- Run 1 billion entry 8192 MPI ranks (128 BG/Q 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.

2 HAND-IN INSTRUCTIONS

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