Felicity Escarzaga
CS 599 HPC
Assignment 1

Activity #1:

An if statement was used to determine if the rank was odd or even; the even ranks sent first and odds received first to keep from deadlock. This was put in a for loop for 5 iterations. The output is shown below:

```
act1.out

Rank: 3, my buffer: 2, my counter: 10


Rank: 4, my buffer: 5, my counter: 25


Rank: 0, my buffer: 1, my counter: 5


Rank: 2, my buffer: 3, my counter: 15


Rank: 7, my buffer: 6, my counter: 30


Rank: 6, my buffer: 7, my counter: 35
            Q2

Rank: 9, my buffer: 8, my counter: 40
                                          Q1

Rank: 5, my buffer: 4, my counter: 20


Rank: 8, my buffer: 9, my counter: 45


Rank: 1, my buffer: 0, my counter: 0
```

Fig. 1: Activity #1 output

Q1: What does process rank 5's counter store at the end of the computation?

- As shown in fig. 1, the counter for rank 5 was 20 at the end.

Q2: How many process ranks are used in the script above?

- The jobscript provided specified 10 ranks with the parameter: `--ntasks = 10`.
- This is also displayed in fig. 1 with 9 ranks displayed.

Felicity Escarzaga
CS 599 HPC
Assignment 1

Activity #2:

If statements were used to separate rank 0, even ranks, and odd ranks. In the case of rank 0, it would send to rank 1 and receive from the last rank (`nproc-1`). In the case of odd ranks, they would receive first from the rank before them (`my_rank-1`), and even ranks would send first to the rank after them (`(my_rank+1)%nprocs`). The use of `%nprocs` was to have the last rank send to rank 0. The above was put in a loop for 10 iterations.

Originally, I would have used `(my_rank-1)%nprocs` in the receive to include rank 0 in the even ranks but c does not compute mod of -1 the way I thought it would.

The output is shown below:

act2.out

```

Rank: 1, my buffer: 0, my counter: 0


Rank: 4, my buffer: 3, my counter: 30


Rank: 5, my buffer: 4, my counter: 40


Rank: 2, my buffer: 1, my counter: 10


Rank: 3, my buffer: 2, my counter: 20


Rank: 0, my buffer: 5, my counter: 50
```

Q3

Fig. 2: Activity #2 output

Q3: What does process rank 5's counter store at the end of the computation?

- As shown in fig. 2, the counter for rank 5 was 40 at the end.

Felicity Escarzaga
CS 599 HPC
Assignment 1

Activity #3:

All ranks sent at the same time to `(my_rank+1)%nprocs`. An if statement was only needed to separate rank 0 which received from `nprocs-1`, and all other ranks which received from `my_rank-1`. Output in fig. 3.

```
act3.out

Rank: 1, my buffer: 0, my counter: 0


Rank: 2, my buffer: 1, my counter: 10


Rank: 3, my buffer: 2, my counter: 20


Rank: 4, my buffer: 3, my counter: 30


Rank: 5, my buffer: 4, my counter: 40


Rank: 0, my buffer: 5, my counter: 50

```

Fig. 3: Activity #3 output

Q4: Comparing Programming Activities #2 and #3, which was easier to implement? Explain.

- Activity #3 was easier than act Activity #2 because using `MPI_Isend` prevents deadlock when all ranks send at the same time.

Felicity Escarzaga
CS 599 HPC
Assignment 1

Activity #4

Rank 0 sends its counter to the first `next_rank` determined by generateRandomRank. It broadcasts the `current_rank` and the `next_rank` to all other ranks.

Each `current_rank` after rank 0, receives the counter and then tells rank 0 its `current_rank` and `next_rank`. Rank 0 broadcasts the new `current_rank` and new `next_rank;` then the `current_rank` sends its new counter to the `next_rank`.

Fig. 5 is a visual diagram of the process that may make more sense than the verbal description.

Finally, activity #4 was not run on monsoon because a single node could not run 50 tasks and using 2 nodes took too much time.

act4.out
```
Master: first rank: 14
My rank: 14, old counter: 0
My rank: 14, new counter: 14
My rank: 14, next rank to recv: 39
My rank: 39, old counter: 14
My rank: 39, new counter: 53
My rank: 39, next rank to recv: 37
My rank: 37, old counter: 53
My rank: 37, new counter: 90
My rank: 37, next rank to recv: 27
My rank: 27, old counter: 90
My rank: 27, new counter: 117
My rank: 27, next rank to recv: 23
My rank: 23, old counter: 117
My rank: 23, new counter: 140
My rank: 23, next rank to recv: 26
My rank: 26, old counter: 140
My rank: 26, new counter: 166
My rank: 26, next rank to recv: 5
My rank: 5, old counter: 166
My rank: 5, new counter: 171
My rank: 5, next rank to recv: 4
My rank: 4, old counter: 171
My rank: 4, new counter: 175
My rank: 4, next rank to recv: 11
My rank: 11, old counter: 175
My rank: 11, new counter: 186
My rank: 11, next rank to recv: 36
My rank: 36, old counter: 186
My rank: 36, new counter: 222
My rank: 36, next rank to recv: 34
```

Fig. 4: Activity #4 output

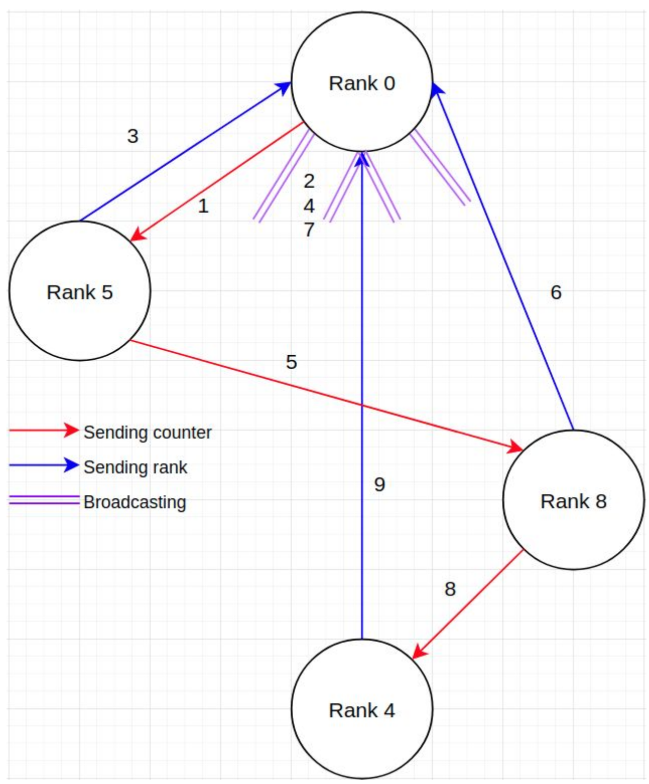

Fig. 5: Visualization of Activity #4

Felicity Escarzaga
CS 599 HPC
Assignment 1

Activity #5:

The current_rank broadcasts to all other ranks the `next_rank`, and only sends the counter to the `next_rank`. The next_rank uses `MPI_ANY_SOURCE` to receive the counter. Fig. 7 shows a visual representation of this process.

This activity was also not run on monsoon since 1 node could not run 50 tasks and 2 nodes took an unknown amount of time. If run with a smaller number of tasks the end result will not be 222.

act5.out

```
My rank: 0, next rank to recv: 14
My rank: 14, old counter: 0
My rank: 14, new counter: 14
My rank: 14, next rank to recv: 39
My rank: 39, old counter: 14
My rank: 39, new counter: 53
My rank: 39, next rank to recv: 37
My rank: 37, old counter: 53
My rank: 37, new counter: 90
My rank: 37, next rank to recv: 27
My rank: 27, old counter: 90
My rank: 27, new counter: 117
My rank: 27, next rank to recv: 23
My rank: 23, old counter: 117
My rank: 23, new counter: 140
My rank: 23, next rank to recv: 26
My rank: 26, old counter: 140
My rank: 26, new counter: 166
My rank: 26, next rank to recv: 5
My rank: 5, old counter: 166
My rank: 5, new counter: 171
My rank: 5, next rank to recv: 4
My rank: 4, old counter: 171
My rank: 4, new counter: 175
My rank: 4, next rank to recv: 11
My rank: 11, old counter: 175
My rank: 11, new counter: 186
My rank: 11, next rank to recv: 36
My rank: 36, old counter: 186
My rank: 36, new counter: 222
```

Fig. 6: Activity #5 output

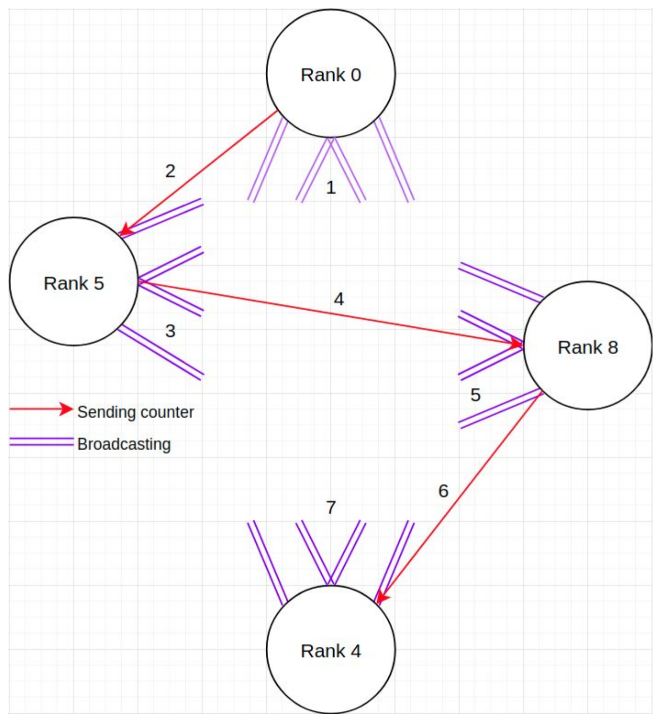

Fig. 7: Visualization of Activity #5

Q5: Comparing Programming Activities #4 and #5, which was easier to implement? Explain.

- Activity 5 was easier to implement because using `MPI_ANY_SOURCE` keeps the `current_rank` from deadlocking even if the `next_rank` doesn't know the previous rank it's receiving from.