Basic Parallel Programs

Owen Morehead

May 2 2022

1 Running the Programs

A `README.md` file is included which contains the commands necessary to compile and run each of the five programs below. Brief instructions are also included here. To compile each of the programs, the command used is:

`mpif90 -o program program.f90` where the name of the program, `program.f90`, is the name provided in each section title below, and the name of the executable, `program`, was chosen to be the exact same name as the program without the `.f90`. After the executables are made (they are also attached in the homework submission), they can be run using the command:

`mpirun \np num_procs program`

where `program` is again the name of the executable made from compiling above, and the number of processors used to run the program, `num_procs` varies by program and is stated in the `README.md` file and also under each section below for the specific output contained in this deliverables report.

2 Hello World – helloWorld_mpi.f90

This program writes out "Hello" from each processor, stating the processor rank and the size of the comm world. The output below is from executing the following command in `Grape`:
`mpirun -np 6 helloWorld_mpi`. Indeed, we see that the output is non-deterministic.

```
-bash-4.1$ mpirun -np 6 helloWorld_mpi
 Hello from processor           5 out of           6
 Hello from processor           0 out of           6
 Hello from processor           2 out of           6
 Hello from processor           4 out of           6
 Hello from processor           3 out of           6
 Hello from processor           1 out of           6
```

Figure 1: Screenshot of the terminal output which shows the result from the helloWorld_mpi.f90 program in parallel using 6 processors.

3 Simple Send-Receive – ssr_mpi.f90

This program sends an array of real values $(1.0, 2.0, 3.0)$ from one processor to another using the standard MPI send and receive. The output below is from executing the following command in `Grape`:
`mpirun -np 2 ssr_mpi`

```
[-bash-4.1$ mpirun -np 2 ssr_mpi
 Processor:            0 sent    1.0000000       2.0000000       3.0000000
 Processor:            1 received    1.0000000       2.0000000       3.0000000
```

Figure 2: Screenshot of the terminal output which shows the result from the ssr_mpi.f90 program in parallel using 2 processors.

4 Ping-Pong – ping_pong_mpi.f90 and ping_pong_simp_mpi.f90

This program sends data backwards and forwards between two processors. I chose to change the data after each "rally" (one rally consists of a send and a receive of the same data on each processor) such that the value of 1 is added after each rally. This allows one to visualize the order in which the send and receives are happening. The output below is from executing the following commands in `Grape`:

```
mpirun -np 2 ping_pong_mpi
mpirun -np 2 ping_pong_simp_mpi
```

where both programs produce the same results. The first program does it using the non-blocking commands, `MPI_ISEND()` and `MPI_IRECV()`, and the `simp` program does it using the blocking commands, `MPI_SEND()` and `MPI_RECV()`. Thus, the `simp` version is coded such that the root processor (ping) calls `MPI_SEND()` first followed by `MPI_RECV()`, and the other processor (pong) calls `MPI_RECV()` first followed by `MPI_SEND()`. This avoids any deadlock problems. The output of the two versions of the program are shown below respectively.

```
[-bash-4.1$ mpirun -np 2 ping_pong_mpi
 Message from Ping to Pong:            1
 Message from Ping to Pong:            2
 Message from Ping to Pong:            3
 Message from Ping to Pong:            4
 Message from Ping to Pong:            5
 Message from Ping to Pong:            6
 Message from Ping to Pong:            7
 Message from Ping to Pong:            8
 Message from Ping to Pong:            9
 Message from Ping to Pong:           10
 Message from Ping to Pong:           11
 Message from Ping to Pong:           12
 Ping-Pong Game Over.           12  total rallies achieved.
 Message from Pong to Ping:            1
 Message from Pong to Ping:            2
 Message from Pong to Ping:            3
 Message from Pong to Ping:            4
 Message from Pong to Ping:            5
 Message from Pong to Ping:            6
 Message from Pong to Ping:            7
 Message from Pong to Ping:            8
 Message from Pong to Ping:            9
 Message from Pong to Ping:           10
 Message from Pong to Ping:           11
 Message from Pong to Ping:           12
```

Figure 3: Screenshot of the terminal output which shows the result from the ping_pong_mpi.f90 program in parallel using 2 processors. This program utilizes the non-blocking MPI_ISEND() and MPI_IRECV() commands.

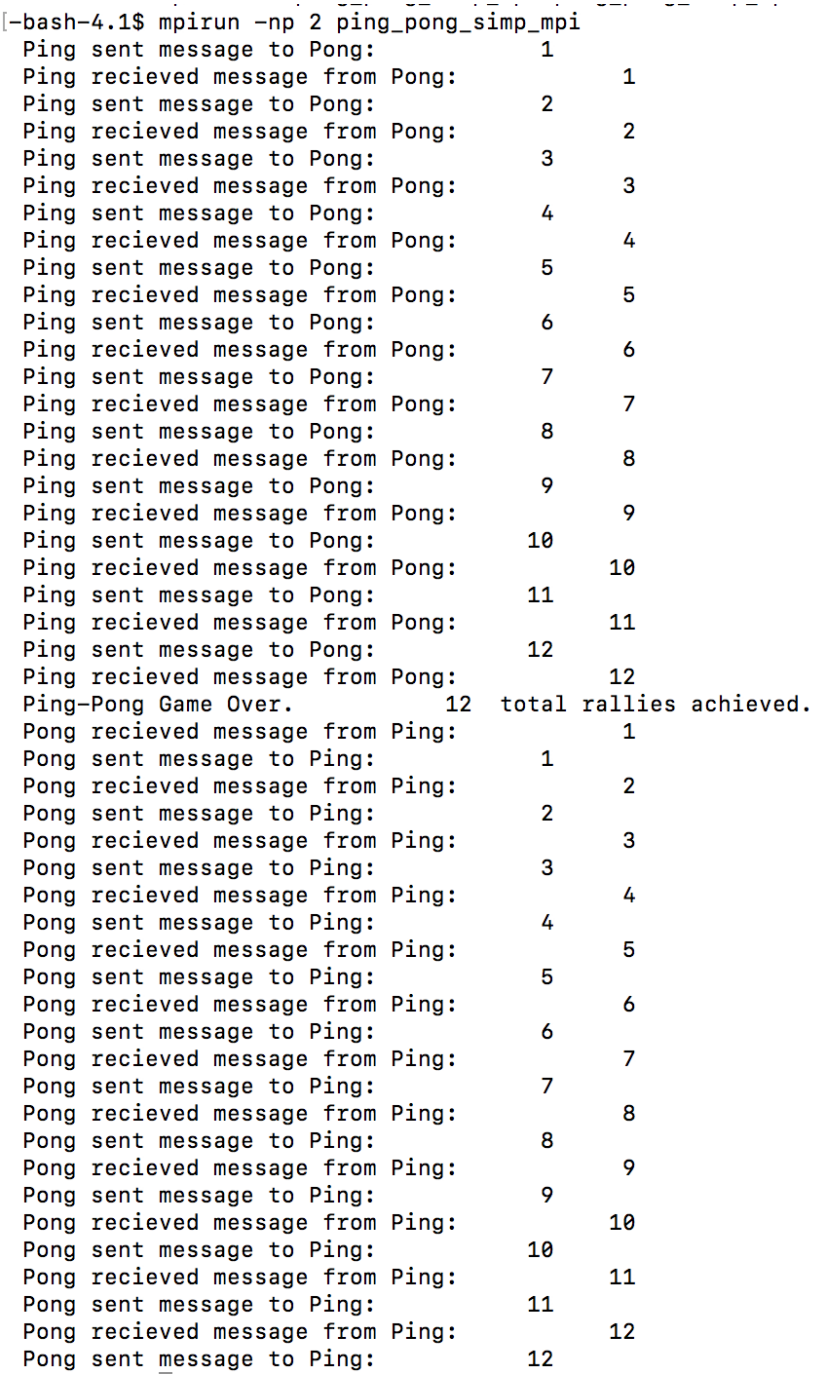

Figure 4: Screenshot of the terminal output which shows the result from the ping_pong_simp_mpi.f90 program in parallel using 2 processors. This program utilizes the blocking MPI_SEND() and MPI_RECV() commands.

5 Ring – ring_mpi.f90

This program sends some data (its processor number) around a ring of N processors either in the right or left direction depending on the user input. I originally sent the data around the ring only once, and then edited the code to send it around the ring N times (as asked in assignment description), such that each processor starts by sending its id number to the neighboring processor, and ends by sending the neighboring id number to that exact neighbor. For example, in a ring of 4 processors as I chose, processor number 0 will start by sending the value 0 to processor 1 if the Right direction is chose. It will then send the value 3 to processor 1, then the value 2, and lastly, it will send the value 1 to processor 1. Each processor in the ring does this process. The below screenshots of the output show this for both a Right and Left shift respectively. The output below is from executing the following command in `Grape`:
`mpirun -np 4 ring_mpi`

```
-bash-4.1$ mpirun -np 4 ring_mpi
 Shift data to Right or Left? Enter ['0' for Right] or ['1' for Left]
0
 Processor number            3  sent data            2  to processor number            0
 Processor number            3  sent data            1  to processor number            0
 Processor number            3  sent data            0  to processor number            0
 Processor number            3  sent data            3  to processor number            0
 Processor number            0  sent data            3  to processor number            1
 Processor number            0  sent data            2  to processor number            1
 Processor number            0  sent data            1  to processor number            1
 Processor number            0  sent data            0  to processor number            1
 Processor number            1  sent data            0  to processor number            2
 Processor number            1  sent data            3  to processor number            2
 Processor number            1  sent data            2  to processor number            2
 Processor number            1  sent data            1  to processor number            2
 Processor number            2  sent data            1  to processor number            3
 Processor number            2  sent data            0  to processor number            3
 Processor number            2  sent data            3  to processor number            3
 Processor number            2  sent data            2  to processor number            3
```

Figure 5: Screenshot of the terminal output which shows the result from the ring_mpi.f90 program in parallel using 4 processors. This output sends the data around to the Right.

```
-bash-4.1$ mpirun -np 4 ring_mpi
 Shift data to Right or Left? Enter ['0' for Right] or ['1' for Left]
1
 Processor number            3  sent data            0  to processor number            2
 Processor number            3  sent data            1  to processor number            2
 Processor number            3  sent data            2  to processor number            2
 Processor number            3  sent data            3  to processor number            2
 Processor number            0  sent data            1  to processor number            3
 Processor number            0  sent data            2  to processor number            3
 Processor number            0  sent data            3  to processor number            3
 Processor number            0  sent data            0  to processor number            3
 Processor number            1  sent data            2  to processor number            0
 Processor number            1  sent data            3  to processor number            0
 Processor number            1  sent data            0  to processor number            0
 Processor number            1  sent data            1  to processor number            0
 Processor number            2  sent data            3  to processor number            1
 Processor number            2  sent data            0  to processor number            1
 Processor number            2  sent data            1  to processor number            1
 Processor number            2  sent data            2  to processor number            1
```

Figure 6: Screenshot of the terminal output which shows the result from the ring_mpi.f90 program in parallel using 4 processors. This output sends the data around to the Left.

6 Pi – pi_mpi.f90

This program figures out the value of π by the "dartboard method" in parallel. A circular dartboard on a square background has a ration of teh areas, $\pi r^2/(2r)^2 = \pi/4$. If we throw "darts" randomly at the dartboard, we can examine the ratio of darts which land inside vs outside the circle, and then use this ratio to estimate the value of π. The output below is from executing the following command in `Grape`:
`mpirun -np 4 pi_mpi`
Additionally, I ran the program using 8 total processors and also changed around the total number of darts randomly thrown to compare the results from these varying parameters.

```
-bash-4.1$ mpirun -np 4 pi_mpi
 Processor            0  recorded        39237  hits inside the circle, out of        50000  total hits
 Processor            1  recorded        39209  hits inside the circle, out of        50000  total hits
 Processor            2  recorded        39468  hits inside the circle, out of        50000  total hits
 Processor            3  recorded        39344  hits inside the circle, out of        50000  total hits
 Approximation of pi:    3.1451600
```

Figure 7: Screenshot of the terminal output which shows the result from the pi_mpi.f90 program in parallel using 4 processors, and 50,000 total random dart throws on each processor.

```
-bash-4.1$ mpirun -np 8 pi_mpi
 Processor            3  recorded        39344  hits inside the circle, out of        50000  total hits
 Processor            4  recorded        39269  hits inside the circle, out of        50000  total hits
 Processor            7  recorded        39246  hits inside the circle, out of        50000  total hits
 Processor            1  recorded        39209  hits inside the circle, out of        50000  total hits
 Processor            2  recorded        39468  hits inside the circle, out of        50000  total hits
 Processor            5  recorded        39240  hits inside the circle, out of        50000  total hits
 Processor            0  recorded        39237  hits inside the circle, out of        50000  total hits
 Processor            6  recorded        39178  hits inside the circle, out of        50000  total hits
 Approximation of pi:    3.1419101
```

Figure 8: Screenshot of the terminal output which shows the result from the pi_mpi.f90 program in parallel using 8 processors, and 50,000 total random dart throws on each processor.

```
-bash-4.1$ mpirun -np 8 pi_mpi
 Processor            7  recorded       784993  hits inside the circle, out of      1000000  total hits
 Processor            2  recorded       785163  hits inside the circle, out of      1000000  total hits
 Processor            3  recorded       785854  hits inside the circle, out of      1000000  total hits
 Processor            5  recorded       785284  hits inside the circle, out of      1000000  total hits
 Processor            0  recorded       785902  hits inside the circle, out of      1000000  total hits
 Processor            6  recorded       786147  hits inside the circle, out of      1000000  total hits
 Processor            1  recorded       784900  hits inside the circle, out of      1000000  total hits
 Processor            4  recorded       785075  hits inside the circle, out of      1000000  total hits
 Approximation of pi:    3.1416590
```

Figure 9: Screenshot of the terminal output which shows the result from the pi_mpi.f90 program in parallel using 8 processors, and 1,000,000 total random dart throws on each processor.

We see that the approximation of π becomes better as more processors are used, and also as the total number of dart throws increases. In Figure 7, 50,000 darts were thrown on 4 processors. Thus returns an approximation, $\pi \approx 3.145160$, which is 0.11% larger than the actual value of π. In Figure 9, 1,000,000 darts were thrown on each processor, and 8 processors were used. This returns an approximation, $\pi \approx 3.1416590$, which is only 0.002% larger than the exact value!