CSE4001 PDC Lab: MPI

Environment

- MobaXterm

(or)

- Cygwin

(or)

- Linux environment

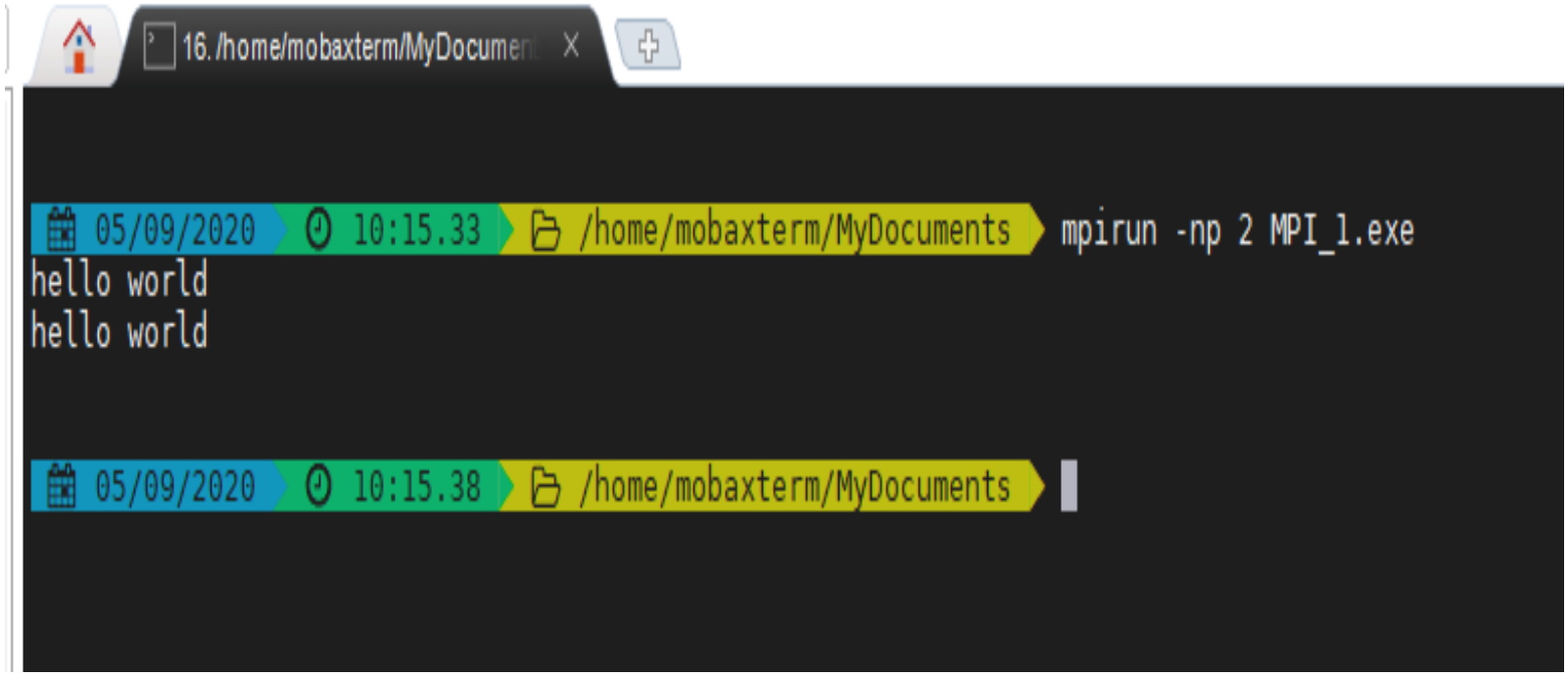
- Here we will use Mobaxterm

Install openmpi

- apt-get install openmpi, MPICH, MPICH2
- apt-get install libopenmpi-devel

Program to test MPI Installation

```
#include<stdio.h>
#include <mpi.h>
main()
{
printf("hello world\n");
}
```

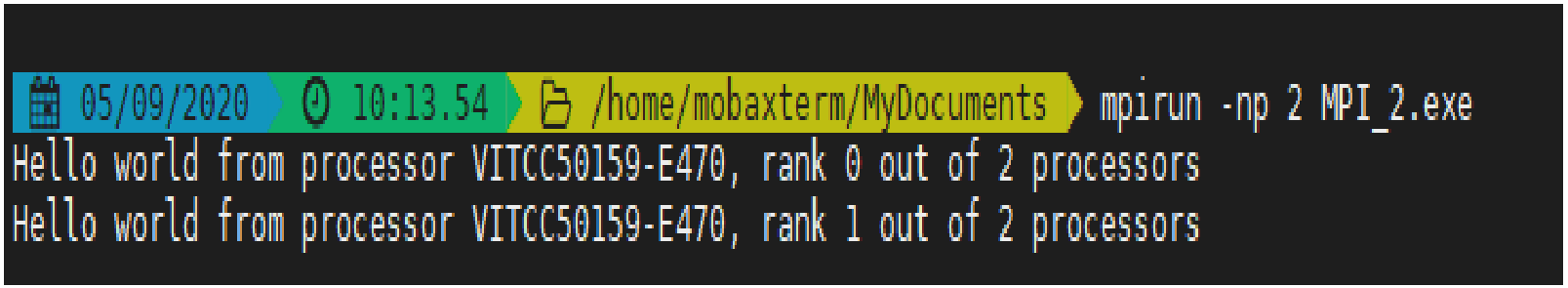

16. /home/mobaxterm/MyDocumen
05/09/2020 10:15.33 /home/mobaxterm/MyDocuments mpirun -np 2 MPI_1.exe
hello world
hello world
05/09/2020 10:15.38 /home/mobaxterm/MyDocuments

Getting Started with MPI

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
```

…Continued

```
 // Get the name of the processor
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int name_len;
    MPI_Get_processor_name(processor_name,  &name_len);

    // Print off a hello world message
    printf("Hello world from processor %s, rank %d out of %d
processors\n",
           processor_name,  world_rank, world_size);

    // Finalize the MPI environment.
    MPI_Finalize();
}
```


05/09/2020
10:13.54
/home/mobaxterm/MyDocuments
mpirun -np 2 MPI_2.exe
Hello world from processor VITCC50159-E470, rank 0 out of 2 processors
Hello world from processor VITCC50159-E470, rank 1 out of 2 processors

Broadcasting with MPI Send and Receive Sample

```
void my_bcast(void* data, int count, MPI_Datatype datatype, int root,  MPI_Comm
communicator) {
  int world_rank;
  MPI_Comm_rank(communicator,  &world_rank);
  int world_size;
  MPI_Comm_size(communicator,  &world_size);

  if (world_rank == root) {
    // If we are the root process, send our data to everyone
    int i;
    for (i = 0; i < world_size; i++) {
      if (i != world_rank) {
        MPI_Send(data, count, datatype, i, 0, communicator);
      }
    }
  } else {
    // If we are a receiver process, receive the data from the root
    MPI_Recv(data, count, datatype, root, 0, communicator,
            MPI_STATUS_IGNORE);
  }
}
```

Compare MPI_Bcast and MPI send, receive (Complete the code)

```
for (i = 0; i < num_trials; i++) {
  // Time my_bcast
  // Synchronize before starting timing
  MPI_Barrier(MPI_COMM_WORLD);
  total_my_bcast_time -= MPI_Wtime();
  my_bcast(data, num_elements, MPI_INT, 0, MPI_COMM_WORLD);
  // Synchronize again before obtaining final time
  MPI_Barrier(MPI_COMM_WORLD);
  total_my_bcast_time += MPI_Wtime();

  // Time MPI_Bcast
  MPI_Barrier(MPI_COMM_WORLD);
  total_mpi_bcast_time -= MPI_Wtime();
  MPI_Bcast(data, num_elements, MPI_INT, 0, MPI_COMM_WORLD);
  MPI_Barrier(MPI_COMM_WORLD);
  total_mpi_bcast_time += MPI_Wtime();
}
```

MPI Non Blocked Point to Point Communication

- MPI_Isend and MPI_Irecv will not wait for the buffer data to be copied
- It will attach a pointer to the message transfer and immediately return
- Use MPI_Wait to ensure proper communication between send and receive

Sample

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])
{
   int myid, numprocs, left, right;
   int buffer[10], buffer2[10];
   MPI_Request request, request2;
   MPI_Status status;

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,
&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD, &myid);
   right = (myid + 1) % numprocs;
   left = myid - 1;
   if (left < 0)
      left = numprocs - 1;
   MPI_Irecv(buffer, 10, MPI_INT, left, 123,
MPI_COMM_WORLD, &request);
   MPI_Isend(buffer2, 10, MPI_INT, right, 123,
MPI_COMM_WORLD, &request2);
   MPI_Wait(&request, &status);
   MPI_Wait(&request2, &status);
   MPI_Finalize();
   return 0;
}
```

MPI Collective Communications

- MPI_BCAST
- MPI_BARRIER
- MPI_SCATTER
- MPI_GATHER

MPI_BCAST

```
Main()
        {
                ....
                data =1;
                MPI_Bcast(data, num_elements,
                MPI_INT, 0, MPI_COMM_WORLD);
                //print the value of data inside a if condition that
                checks the rank (other than the root)

        }
```

MPI_Scatter

- Designated root process sending data to all processes in a communicator
- MPI_Bcast → Same data
- MPI_Scatter → chunk of data from different processes

Syntax

- MPI_Scatter(void* send_data, int send_count, MPI_Datatype send_datatype, void* recv_data, int recv_count, MPI_Datatype recv_datatype, int root, MPI_Comm communicator);

Example: Root process generates random numbers and scatters it among multiple slaves

```
float create_rand_nums(int n)
        {
         float *rnd=(float *) malloc(sizeof(float)*n);
        for i= 0 to n
                        {
                rnd[i]=rand();
                        }
        return rnd;
         }
```

…Continued

```
Main()
        {
        ….
        int id;
         int p;
        MPI_Comm_rank(MPI_COMM_WORLD, &id);
        MPI_Comm_size(MPI_COMM_WORLD, &p);
              float *rand_nums = NULL;
                if (id== 0) {
                      rand_nums = create_rand_nums(int   *p);
                    }
        float *sub_rand_nums = (float *)malloc(sizeof(float) * p);
                MPI_Scatter(rand_nums, num_elements_per_proc,
MPI_FLOAT, sub_rand_nums, 1, MPI_FLOAT, 0,
        MPI_COMM_WORLD);
                  …..
                }
```

MPI_Gather

- MPI_Gather(void* send_data, int send_count, MPI_Datatype send_datatype, void* recv_data, int recv_count, MPI_Datatype recv_datatype, int root, MPI_Comm communicator);

MPI_AllGather

- It is MPI_Gather followed by MPI_Bcast