

Parallel and Distributed Computing
CSE4001
Fall Semester 2020-21

Lab Assignment 6

ISHAAN OHRI
18BCE0265

Aim:

Consider a suitable instance that has MPI routines to assign different tasks to different processors.

For example, parts of an input data set might be divided and processed by different processors, or a finite difference grid might be divided among the processors available. This means that the code needs to identify processors. In this example, processors are identified by rank - an integer from 0 to total number of processors.

1. Implement the logic using C
2. Build the code
3. Show the screenshots with proper justification

Note.

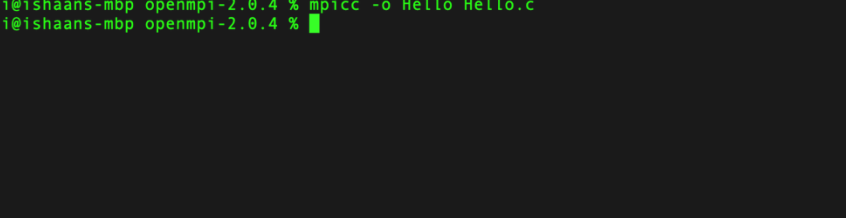
Compile and run with:

```
mpicc -o Hello Hello.c  
mpirun -np 4 ./Hello
```

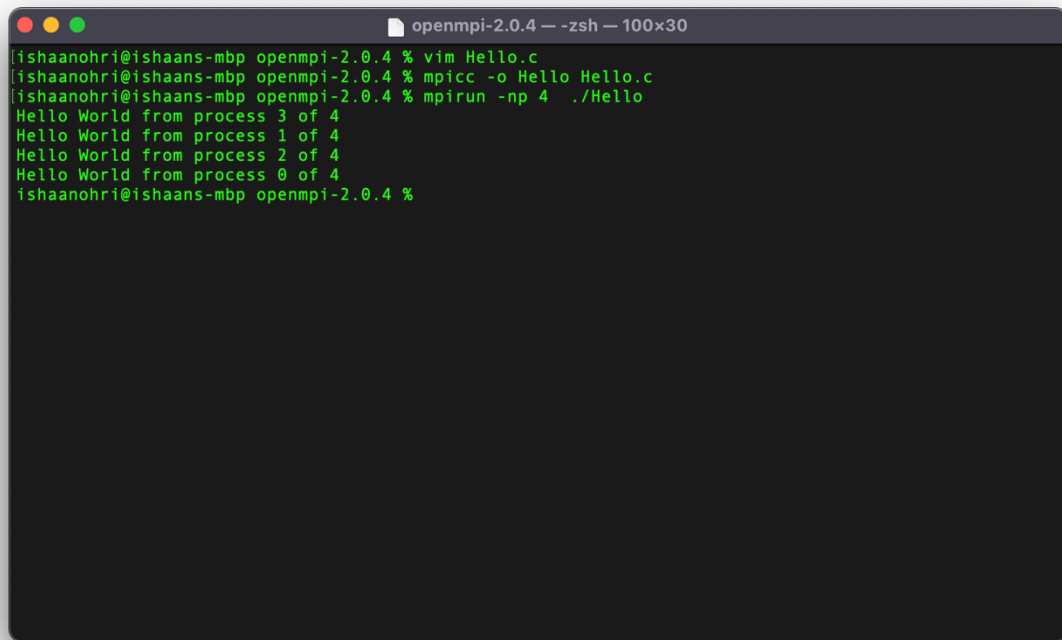
Source Code:

[illegible]

Execution:



```
openmpi-2.0.4 — zsh — 102x26
[ishaanohri@ishaans-mbp openmpi-2.0.4 % mpicc -o Hello Hello.c
[ishaanohri@ishaans-mbp openmpi-2.0.4 %
```

A terminal window titled 'openmpi-2.0.4 — zsh — 100x30' showing the execution of an MPI program. The user 'ishaanohri' is at a machine named 'ishaans-mbp'. The commands executed are: 'vim Hello.c', 'mpicc -o Hello Hello.c', and 'mpirun -np 4 ./Hello'. The output shows four processes printing 'Hello World from process X of 4' where X is 3, 1, 2, and 0 respectively. The prompt returns to 'ishaanohri@ishaans-mbp openmpi-2.0.4 %' after the execution.

```
openmpi-2.0.4 — zsh — 100x30
[ishaanohri@ishaans-mbp openmpi-2.0.4 % vim Hello.c
[ishaanohri@ishaans-mbp openmpi-2.0.4 % mpicc -o Hello Hello.c
[ishaanohri@ishaans-mbp openmpi-2.0.4 % mpirun -np 4 ./Hello
Hello World from process 3 of 4
Hello World from process 1 of 4
Hello World from process 2 of 4
Hello World from process 0 of 4
ishaanohri@ishaans-mbp openmpi-2.0.4 %
```

Result:

From this experiment, after implementing the program I learnt the following:

The command `MPI_Init(NULL, NULL)` initializes the MPI execution environment with the two arguments being pointer to the number of arguments and the pointer to the argument vector.

The command `MPI_Comm_rank()` is used to determine the rank of the calling process in the communicator. It takes the communicator as an argument.

The command `MPI_Comm_size()` is used to determine the size of the calling process in the communicator. It takes the communicator as an argument.

The command `MPI_Finalize()` is used to terminate the MPI execution environment. All processes must call this routine before exiting. The number of processes running after this routine is called is undefined.

The above experiment was conducted and all results along with the source code have been attached above in the document. The experiment was assisted by Dr Deebak. I thank sir for his assistance.