INDIAN INSTITUTE OF TECHNOLOGY GOA

NSM Nodal Center for Training in HPC and AI

Tutorial: MPI "HELLO WORLD" PROGRAM

Authors:

1. **CHODAVARAPU AISHWARYA**, **Project Associate**
2. **Prof. SHARAD SINHA**

MPI "Hello World" Program on Param Vidya HPC

Objective:

This tutorial will show you how to create, compile, and run a simple MPI "Hello World" program on the Param Vidya HPC cluster. We will create a parallel "Hello World" program using MPI (Message Passing Interface), where multiple processes will execute concurrently. One process will be designated to handle the output by collecting messages from other processes and printing them.

Program Explanation:

MPI programs use ranks to identify processes. If there are p processes, they will have ranks ranging from 0 to p-1. In our program, process 0 will be the designated process that receives messages from other processes and prints them.

‘hello_world.c’ code:

```
#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.

    double start_time = MPI_Wtime();

    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);
    // 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();

    double end_time = MPI_Wtime();

    printf("Total execution time: %f seconds\n", end_time - start_time);
}
```

Code Explanation:

1. **Initialization:**
    - MPI_Init(NULL, NULL): Initializes the MPI environment. This function must be called before any other MPI functions.

2. **Number of Processes:**
    - MPI_Comm_size(MPI_COMM_WORLD, &world_size): Retrieves the total number of processes in the communicator MPI_COMM_WORLD.
    - A communicator holds a group of processes that can communicate with each other. All message passing calls in MPI must have a specific communicator to use the call with. MPI_COMM_WORLD is the default communicator that contains all processes available for use.

3. **Rank of the Process:**
    - MPI_Comm_rank(MPI_COMM_WORLD, &world_rank): Determines the rank (ID) of the calling process within the communicator.

4. **Processor Name:**
    - MPI_Get_processor_name(processor_name, &name_len): Retrieves the name of the processor on which the process is running.

5. **Print Message:**
    - The program prints a message from each process, displaying the processor name, rank, and total number of processes.

6. **Finalization:**
    - MPI_Finalize(): Cleans up the MPI environment. This should be called after all MPI-related operations are complete.

Compilation and Execution:

First you can compile and run the program on your local machine to observe how the code works. To compile and run the program on your local machine, follow these steps:

1. **Install OpenMPI:**
    - Ensure that OpenMPI is installed on your local machine.

2. **Compile the Program:**
    -
    ```
    mpicc -o mpi hello_world.c
    ```
    - mpicc: It is a script that acts as a MPI wrapper for the C compiler. Its primary function is to facilitate the compilation process by automatically specifying the locations of necessary header files and libraries required for linking with the object file. This simplifies the task of compiling programs by hiding these details for the user.

3. **Run the Program:**
    -
    ```
    mpirun -np <number_of_processes> mpi
    ```
    - mpirun: Launches the MPI program. The mpirun command is used to launch multiple instances of a program across different processor cores. It can also specify which core each instance should run on. Once the processes are running, the MPI implementation ensures they can communicate with each other effectively.

    - The -np option specifies the number of processes to run. For example, to run with 4 processes, we would use the following command:

    ```
    mpirun -np 4 mpi
    ```

    - Then the output would look like this:

    ```
    $ mpirun -np 4 mpi
    Hello world from processor pop-os, rank 3 out of 4 processors
    ```

```
Hello world from processor pop-os, rank 2 out of 4 processors
Hello world from processor pop-os, rank 0 out of 4 processors
Hello world from processor pop-os, rank 1 out of 4 processors
```

- If you exceed the processor limit on your local machine, then the terminal might display a message similar to this:

```
$ mpirun -np 9 mpi
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 9
slots that were requested by the application:

  mpi

Either request fewer procs for your application, or make more slots
available for use.
```

Steps to Run the Program on the Param Vidya HPC Cluster:

To run the MPI program on the Param Vidya HPC cluster, follow these steps:

1. **Log in to the HPC Machine:**
   - Linux systems provide a built-in SSH client, so there is no need to install any additional package. Use SSH to log in to the HPC cluster. You'll need the appropriate credentials and network access.

```
ssh username@hpc_address
```

   - For example, to connect to the PARAM Vidya Login Node,

```
ssh username@paramvidya.iitgoa.ac.in -p <port number>
```

2. **Transfer hello_world.c to the Cluster:**
   - Use scp to transfer the source code file from your local machine to the cluster:

```
scp –P 4422 -r /path/to/directory/hello_word.c <your username>@paramvidya.iitgoa.ac.in:<path to directory on HPC where to save the data>
```

- Otherwise, you can also create a hello_world.c file after logging in to the HPC machine using commands like nano, vim etc.

3. **Load the MPI module:**
   - First you need to load the mpi compiler in the mpi environment before the program runs.
   - To list the modules available on the hpc machine, use the below command:

```
module avail
```

   - Identify the correct name of the openmpi module from the list (in my case it is openmpi3/3.1.4). Then, load that module with the command:

```
module load openmpi3/3.1.4
```

   - This will load the openmpi module on the HPC machine.

4. **Compile the Program:**
   -

```
mpicc -o mpi mpi_hello_world.c
```

5. **Run the Program:**
   -

```
mpirun -np <number_of_processes> mpi
```

   - For example, to run with 4 processes, we would use the below command.

```
mpirun -np 4 mpi
```

   - Then the output would look like this:

```
Hello world from processor paramvidya.iitgoa.ac.in, rank 2 out of 4 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 0 out of 4 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 1 out of 4 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 3 out of 4 processors
```

- Since the limit of the number of processors in our hpc machine is 48, we can run the program across 48 processors.

```
Hello world from processor paramvidya.iitgoa.ac.in, rank 10 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 36 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 42 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 44 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 24 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 29 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 30 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 32 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 0 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 2 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 4 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 5 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 8 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 15 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 17 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 26 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 28 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 40 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 43 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 6 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 12 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 16 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 18 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 20 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 25 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 35 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 41 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 1 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 3 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 7 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 11 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 13 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 14 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 19 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 21 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 22 out of 48 processors
```

```
Hello world from processor paramvidya.iitgoa.ac.in, rank 23 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 27 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 31 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 33 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 34 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 37 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 38 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 39 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 45 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 46 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 47 out of 48 processors
Hello world from processor paramvidya.iitgoa.ac.in, rank 9 out of 48 processors
```

6. **Create a SLURM Job Script:**
   - On the cluster, create a job script file named test_hello_world.sh. This script will automate the job submission process.

```
vim test_hello_world.sh
```

   - Insert the following content into the script file:

```
#!/usr/bin/bash

#SBATCH -N 1                # Specifies number of nodes
#SBATCH --job-name=testHelloWorld   # Name the job as
'testHelloWorld'
#SBATCH --output=job.%J.out_node_48
#SBATCH --error=job.%J.err_node_48
#SBATCH --ntasks-per-node=48 # Request n cores or task per node
#SBATCH --mem-per-cpu=4GB   # Request 4GB RAM per core
#SBATCH --time=06:50:20 # maximum duration of the run
#SBATCH --partition=standard # specifies queue name

module load gnu8/8.3.0
module load openmpi3/3.1.4
#module list                # will list modules loaded by default. In our case,
it will be GNU8 compilers and OpenMPI3 MPI libraries

pwd                         # prints current working directory
date                        # prints the date and time

mpicc -o mpi mpi_hello_world.c
mpirun mpi        # run the MPI job
```

7. **Submit the Job Script:**
    - Use the sbatch command to submit the job script to the SLURM scheduler:

```
sbatch test_hello_world.sh
```

8. **Check Job Status:**
    - Use squeue to check the status of your job. This will show the current jobs in the queue for your user.

```
squeue -u username
```

9. **View Output:**
   Once the job completes, view the output files to see the results. The output file will be named according to the #SBATCH --output directive, with %J replaced by the job ID.