MPI Program Execution

Sample Programs - Explanations

MPI Program Execution

- How to program distibuted-memory systems using message-passing.
- In message passing programs, a program running on one core-memory pair is usually called a **process**, and 2 processes can communicate by calling functions:
  - One process calls a send function and
  - the other calls a receive function
- The implementation of message-passing is called Message-Passing Interface(MPI)

MPI Program Execution: Simple **MPI** Program

```
/*

Setting the path as :

------------------------

export PATH="$PATH:/home/
$USER/.openmpi/bin"

export
LD_LIBRARY_PATH="$LD_LIBRARY_P
ATH:/home/$USER/.openmpi/lib/"


$ cd MPI_Prog/


*/
```

```
#include <stdio.h>

#include <mpi.h>


  main(int argc, char **argv)

  {

    int ierr;


    ierr = MPI_Init(&argc, &argv);

    printf("Hello world\n");


    ierr = MPI_Finalize();

  }
```

MPI Program Execution

- Output:

```
$ mpicc hello.c -o hello

$ mpirun -np 2 ./hello


Hello world

Hello world
```

MPI Program Execution: Simple C program

```
#include <stdio.h>
int main(void)
{
printf(“hello, world”);
return 0;
}
```

- Lets write a program IIIr to “hello, world” that makes some use of MPI.
- Instead of having each process simply print a message, we'll designate one process to do the output, and the other processes will send it messages, which it will print.

MPI Program Execution

- In parallel programming, its common for the processes to be identified by nonnegative integer ranks.
- So if there are p processes, the processes will have ranks 0,1,2, ... p-1.
- For our parallel “hello, world”, let's make process 0 the designated process, and the other processes will send it messages.

MPI Program Execution

```
#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char** argv)

{

  int q;

  // Initialize the MPI environment

  MPI_Init(NULL, NULL);

  // Find out rank, size

  int world_rank;  // number the processes

  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

  int world_size; // Tot number the threads

  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```

```
// We are assuming at least 2 processes for this task

  if (world_size < 2) {

    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);

    MPI_Abort(MPI_COMM_WORLD, 1);

  }
```

MPI Program Execution

```
char greeting[50];
  if (world_rank != 0)
  {
    // If we are rank != 0, send the greeting to process id !=0
    sprintf(greeting,"Greeting... from process %d of %d!",world_rank,world_size);
    MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
  }
```

```
 else
  {
    printf("Greetings from process %d of %d!\n",world_rank,world_size);
    for(q=1;q<world_size;q++)
    {
      MPI_Recv(greeting, 50, MPI_CHAR, q, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE
      printf("%s\n",greeting);
    }
  }  // end of else


MPI_Finalize();
  return 0;
} // end of main()
```

MPI Program Execution

```
char greeting[50];
  if (world_rank != 0)
  {
    // If we are rank != 0, send the greeting to process id !=0
    sprintf(greeting,"Greeting... from process %d of %d!",world_rank,world_size);
    MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
  }
```

Dest: rank of the process that should recv the msg

tag: nonnegative int. 0=> messages to be printed 1=> messages to be used in a computation

communicator: collection of processes that can send messages to each other. If 2 processes are using diff communicator, msg cant be sent and recvd

```
 else
  {
    printf("Greetings from process %d of %d!\n",world_rank,world_size);
    for(q=1;q<world_size;q++)
    {
      MPI_Recv(greeting, 50, MPI_CHAR, q, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
      printf("%s\n",greeting);
    }
  }  // end of else

MPI_Finalize();
  return 0;
} // end of main()
```

MPI Program Execution

```
char greeting[50];

  if (world_rank != 0)

  {

    // If we are rank != 0, send the greeting to process id !=0

    sprintf(greeting,"Greeting... from process %d of %d!",world_rank,world_size);

    MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);

  }
```

```
 else

  {

    printf("Greetings from process %d of %d!\n",world_rank,world_size);

    for(q=1;q<world_size;q++)

    {

      MPI_Recv(greeting, 50, MPI_CHAR, q, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

      printf("%s\n",greeting);

    }

  }  // end of else


MPI_Finalize();

  return 0;

} // end of main()
```

Source: specifies the process from which the message should be received

tag: match the tag arg of the msg being sent

MPI Program Execution

- Output:

```
$ mpicc send_recv.c -o send_recv
$ mpirun -np 2 ./send_recv

Greetings from process 0 of 2!
Greeting... from process 1 of 2!


or


$ mpirun -np 4 ./send_recv
Greetings from process 0 of 4!
Greeting... from process 1 of 4!
Greeting... from process 2 of 4!
Greeting... from process 3 of 4!
```

- MPI_Comm_rank and MPI_Comm_size are first used to determine the world size along with the rank of the process.

- In the else statement, process zero initializes greeting message with process id 0

- As you can see in the if statement, process ranks(which are !=0) is calling MPI_Send to send the greeting. Message received at the MPI_Recv also