BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)

CS G532 High Performance Heterogeneous Computing

Lab#2

Note: Please use programs under *Code* directory supplied with this sheet. Do not copy from this sheet.

The lab has the following objectives:
Practice programs for MPI.

1. MPI communication primitives

Basic functions to send and receive data in MPI are MPI_Send and MPI_Recv. Consider the following program given in mpi_hello.c

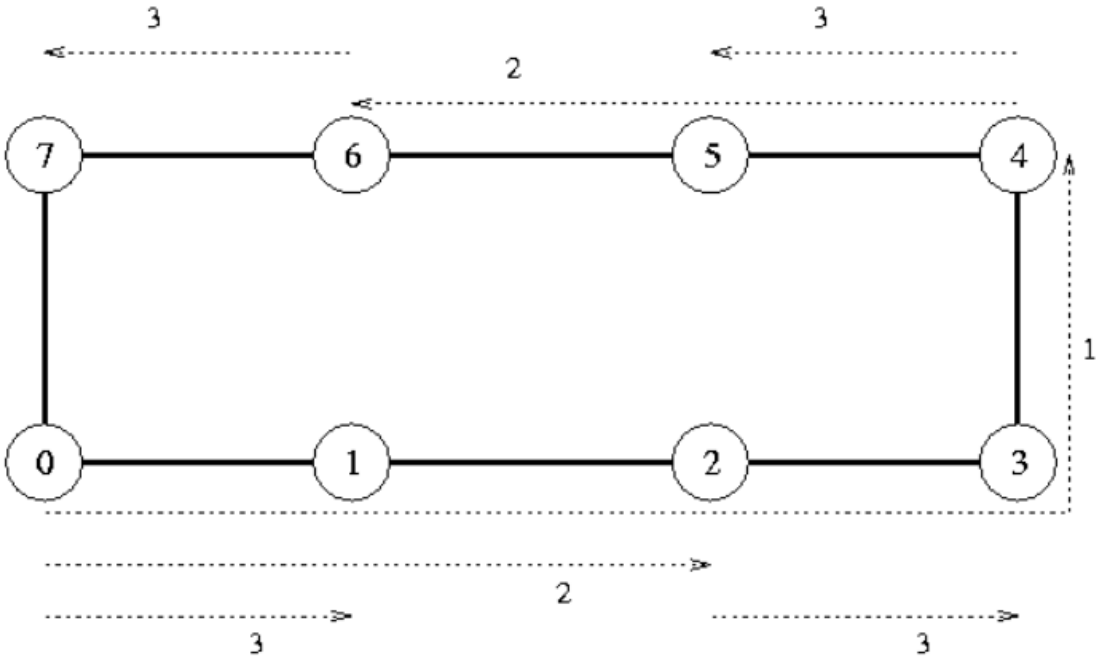
```
1.  int main(void) {
2.      char       greeting[MAX_STRING];  /* String storing message */
3.      int        comm_sz;               /* Number of processes    */
4.      int        my_rank;               /* My process rank        */
5.
6.      /* Start up MPI */
7.      MPI_Init(NULL, NULL);
8.
9.      /* Get the number of processes */
10.     MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
11.
12.     /* Get my rank among all the processes */
13.     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
14.
15.     if (my_rank != 0) {
16.        /* Create message */
17.        sprintf(greeting, "Greetings from process %d of %d!",
18.              my_rank, comm_sz);
19.        /* Send message to process 0 */
20.        MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
21.              MPI_COMM_WORLD);
22.     } else {
23.        /* Print my message */
24.        printf("Greetings from process %d of %d!\n", my_rank, comm_sz);
25.        for (int q = 1; q < comm_sz; q++) {
26.           /* Receive message from process q */
27.           MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
```

```
28·            0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
29·         /* Print message from process q */
30·         printf("%s\n", greeting);
31·      }
32·   }
33·
34·   /* Shut down MPI */
35·   MPI_Finalize();
36·
37·   return 0;
38·} /* main */
```

In the above program, all processes send a message to rank 0 process. Rank 0 processes receives all and prints them.

Q?

1. Compile the above program using `mpicc mpi_hello·c` Run `mpiexec -n 4 a·out`
2. In the current program, rank 0 process is getting strings sequentially in the order of process rank. Modify the above program so that rank 0 process can receive from any process but comm_sz times. How can rank 0 process who is sending message? [Hint: use MPI_Status variable instead of MPI_STATUS_IGNORE]
3. Modify the above program for: All non-root processes send two messages with tags “A” and “B”. Rank 0 should receive tag ‘B’ first and then tag ‘A’ messages.
4. Change the above program such that every process sends a message to its right neighbour in a ring network.
5. Change the above program such that a process sends a message to a process with rank equal to (own_rank + (comm_sz/2^itr_no)). There is log(comm_sz) iterations. Not all processes are active in the beginning iterations.



2. Collective computation

The following program evaluates an integral for given intervals on function f.

```
1·  #include<mpi·h>
2·  #include<stdio·h>
3·
4·  double func(double x) {
5·    return (double)x * x;
6·  }
7·
8·  double Trap(double a, double b, int n, double h) {
9·    double area = (func(a) + func(b)) / 2·0;
10·   for (int i = 1; i <= n - 1; ++i) {
11·     double x = a + i * h;
12·     area += func(x);
13·   }
14·   area *= h;
15·   return area;
16· }
17·
18· int main() {
19·   int my_rank, comm_sz, n = 1024, local_n;
20·   double a = 0·0, b = 3·0, h, local_a, local_b;
21·   double local_int, total_int;
22·   int source;
23·   MPI_Init(NULL, NULL);
24·   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
25·   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
26·   h = (b - a) / n; /* h is the same for all processes */
27·   local_n = n / comm_sz; /* So is the number of trapezoids */
28·   local_a = a + my_rank * local_n * h;
29·   local_b = local_a + local_n * h;
30·   local_int = Trap(local_a, local_b, local_n, h);
31·   if (my_rank != 0) {
32·
33·     MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
34·
35·   }
36·   else {
37·
38·     total_int = local_int;
39·     for (source = 1; source < comm_sz; source++) {
40·       MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0, MPI_COMM_WORLD,
41·
```

```
42·         MPI_STATUS_IGNORE);
43·
44·       total_int += local_int;
45·
46·     }
47·   }
48·   if (my_rank == 0) {
49·
50·     printf("With n = %d trapezoids, our estimate\n", n);
51·     printf("of the integral from %f to %f = %·15e\n",
52·       a, b, total_int);
53·
54·   }
55·   MPI_Finalize();
56·   return 0;
57·}
```

Q?

1. Compile the program with `mpicc point2point·c`. Run `mpiexec -n 4 a·out` . Add MPI_Wtime at line no 24 and 54 and print the time taken. Measure time with respect to increasing number of processes 1 2 3 4 …. Check the speedup achieved. Compute efficiency as well.
2. Change the above program so that variables a, b and n are dynamically read from user in rank 0 and shared with other processes. Can we read from standard input in non-root processes?
3. Can line nos 39-44 be replaced with MPI_Reduce? Change your program accordingly? Is there any difference in execution time n=1 2 3 4 5 6 ?
4. What should we do if all processes need to know the total integral?

3. Collective Communication Routine

This is a program to distribute rows of an array to separate processes.

```
1·  #include<mpi·h>
2·  #include<stdio·h>
3·
4·  #define SIZE 4
5·
6·  int main(int argc, char* argv[]) {
7·    int numtasks, rank, sendcount, recvcount, source;
8·    float sendbuf[SIZE][SIZE] = {
9·      {1·0, 2·0, 3·0, 4·0},
10·     {5·0, 6·0, 7·0, 8·0},
11·     {9·0, 10·0, 11·0, 12·0},
```

```
12·     {13·0, 14·0, 15·0, 16·0}
13·   };
14·   float recvbuf[SIZE];
15·   MPI_Init(&argc, &argv);
16·   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
17·   MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
18·   if (numtasks == SIZE) {
19·     // define source task and elements to send/receive, then perform collective scatter
20·     source = 1;
21·     sendcount = SIZE;
22·     recvcount = SIZE;
23·     MPI_Scatter(sendbuf, sendcount, MPI_FLOAT, recvbuf, recvcount, MPI_FLOAT, source,
    MPI_COMM_WORLD);
24·     printf("rank= %d Results: %f %f %f %f\n", rank, recvbuf[0], recvbuf[1], recvbuf[2],
    recvbuf[3]);
25·   }
26·   else
27·     printf("Must specify %d processors· Terminating·\n", SIZE);
28·   MPI_Finalize();
29·}
```

Q?

1. Compile and run the program with `mpirun -np 4 ·/a·out` or `mpiexec -n 4 a·out`. Why was Scatter used here instead of Bcast?
2. Suppose we need to send first two rows to P0 and next two rows to P1. Modify above program in this regard.
3. Suppose SIZE is not equal to number of processes. Says number of processes is 3. Use MPI_Scatterv. Modify the given program.

Consider the program given mpi_mat_vect_mult.c. This program implements Matrix Vector implementation using MPI. Rank 0 takes input from user/random and distributed to other processes.

Q?

4. This program shows usage of broadcast, scatter, gather and allgather routines. Why do we need to call MPI_Allgather in line 434? Modify the program so that every process will have all elements of x? Measure time taken for original and modified program.

Consider program given in mpi_odd_even.c file. This is the implementation of odd-even transposition sort.

Q?

1. This program uses MPI_Sendrecv leaving the scheduling of send and receive in multiple processes to MPI runtime. Replace MPI_Sendrecv with MPI_Send and MPI_Recv functions. Does it lead to deadlock?

4. Derived Data type

Consider the program given in mpi_trap4.c This program instead of broadcasting a,b and n separately to all processes, creates a new datatype and broadcasts all together.

Q?

1. Compile and run the program.
2. Change the program so that new datatype includes not just a, b, and n but also source and destination ranks and message “hello”.

Process Topology

```
1·  #include<mpi·h>
2·  #include<stdio·h>
3·
4·  int main(int argc, char* argv[]) {
5·    int rank, size;
6·    MPI_Comm comm;
7·    int dim[2], period[2], reorder;
8·    int coord[2], id;
9·
10·   MPI_Init(&argc, &argv);
11·   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
12·   MPI_Comm_size(MPI_COMM_WORLD, &size);
13·
14·   dim[0] = 4; dim[1] = 3;
15·   period[0] = 1; period[1] = 0;
16·   reorder = 1;
17·   MPI_Cart_create(MPI_COMM_WORLD, 2, dim, period, reorder, &comm);
```

```
18·   if (rank == 5) {
19·     MPI_Cart_coords(comm, rank, 2, coord);
20·     printf("Rank %d coordinates are %d %d\n", rank, coord[0], coord[1]);fflush(stdout);
21·   }
22·   if (rank == 0) {
23·     coord[0] = 3; coord[1] = 1;
24·     MPI_Cart_rank(comm, coord, &id);
25·     printf("The processor at position (%d, %d) has rank %d\n", coord[0], coord[1],
    id);fflush(stdout);
26·   }
27·   MPI_Finalize();
28·   return 0;
29·}
```

Q?

1. This is a program which generates a virtual topology in the form of a 3x4 array with wrap around (a 2D torus). Run the above program with `mpirun -np 12 ./a.out`.
2. Can you change the dimensions of the topology?

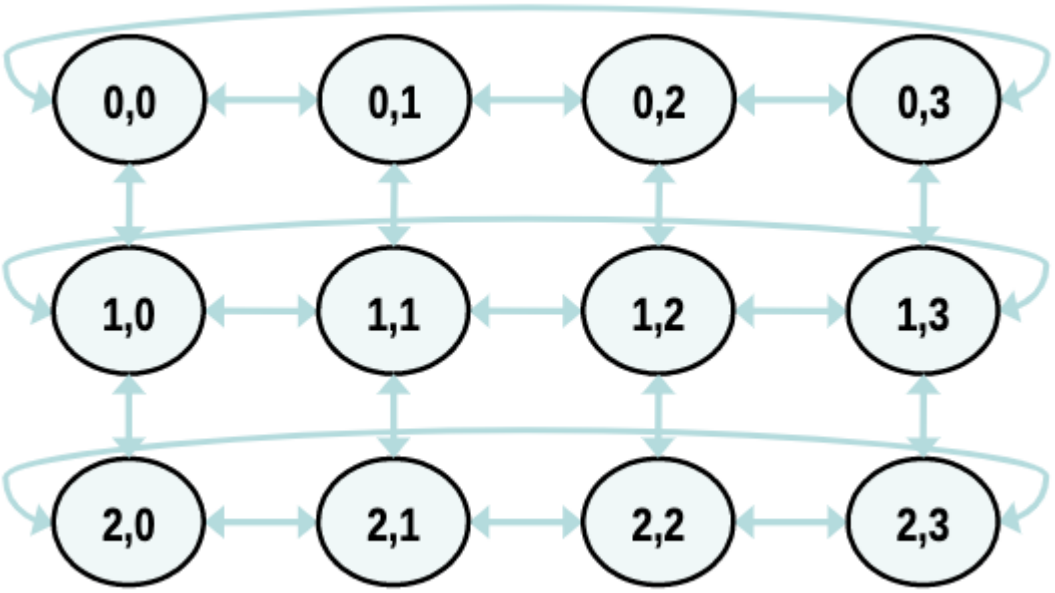


Figure 1: Source: codingame.com

End of lab