CS633A: Parallel Computing
Users Online : 21

Quiz 2

Q.1

```
// this is a psuedo code, ignore syntax errors

// 'rank' is the rank of the process
N = 4;
color = rank%N;

//  int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm * newcomm)
Split MPI_COMM_WORLD using 'color', output is newcomm, key is 'rank'

'newrank' is the rank in 'newcomm'
'newsize' is the size of 'newcomm'

//int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
if (newrank == newsize-1 and newsize > 1)
   then send 'newrank' (i.e. 1 integer) to 0 using MPI_Send (communicator = newcomm, tag=color)

// int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status * status)
else if (newrank == 0 and newsize > 1)
   then recv 1 integer in 'newrank' from newsize-1 using MPI_Recv (communicator = newcomm, tag=color)

printf ("%d %d %d\n", rank, newrank, newsize);
```

Max. score: 4; Neg. score: 1; Your score: -1

- For mpirun -np 500 ./code, the output will contain a line "0 124 125"
- For mpirun -np 12 ./code, the 'newrank' of 8 processes are 1
- For mpirun -np 500 ./code, the output will contain a line "12 3 125"
- For mpirun -np 56 ./code, the value of 'newsize' is 14
- For mpirun -np 57 ./code, the 'newrank' of 'rank=56' is 14

Q.2

```
// Don't worry about semi-colons, this is just a psuedo-C code!

// myrank is the global rank of the process
int color = myrank%4;
int newrank, newsize;

MPI_Comm newcomm;

//  int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm * newcomm)
MPI_Comm_split (MPI_COMM_WORLD, color, myrank, &newcomm);

MPI_Comm_rank (newcomm, &newrank);
MPI_Comm_size (newcomm, &newsize);

printf("Old %d New %d\n", myrank, newrank);
```

Max. score: 4; Neg. score: 1; Your score: 4

- If you run the code on 102 processes, Total number of groups = 26
- If you run the code on 100 processes, Total number of groups = 25
- If you run the code on 102 processes, Old rank 88 will have new rank 22

CS633A: Parallel Computing
Users Online : 21

```
// Don't worry about semi-colons, this is just a psuedo-C code!

  char message[20], recvmessage[100];
  int myrank;
  MPI_Status status;
  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
  if (myrank == 0)    /* code for process 0 */
  {
    strcpy(message,"Hello, there");
    MPI_Send(message, 5, MPI_CHAR, 1, 99, MPI_COMM_WORLD);
  }
  else if (myrank == 1)  /* code for process 1 */
  {
    MPI_Recv(recvmessage, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status);
    printf("%s\n", recvmessage);
  }
  MPI_Finalize();
```

Max. score: 3; Neg. score: 1; Your score: 3

mpirun -np 3 ./code will print “Hello” once

mpirun -np 1 ./code will print “Hello” once

mpirun -np 2 ./code will print “Hello, there” once

mpirun -np 2 ./code will print “Hello” twice

mpirun -np 2 ./code will print “Hello” once

Q.4

```
// Don't worry about semi-colons, this is just a psuedo-C code!

  int N=atoi(argv[1]), myrank, size, count;
  int arr[N], recvarr[N];
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);

  if (myrank == 0)
    // int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
    MPI_Send(arr, N, MPI_INT, 1, 99, MPI_COMM_WORLD);
  else
  {
    // int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status * status)
    MPI_Recv(recvarr, 2*N, MPI_INT, 0, 99, MPI_COMM_WORLD, &status);
    // int MPI_Get_count(const MPI_Status * status, MPI_Datatype datatype, int *count)
    MPI_Get_count (&status, MPI_INT, &count);
    printf ("%d %d\n", myrank, count);
  }

  MPI_Finalize()
```

Max. score: 3; Neg. score: 1; Your score: 3

mpirun -np 1 ./code 2 will give error

CS633A: Parallel Computing
Users Online : 21

mpirun -np 3 ./code 2 will print 3 2

mpirun -np 2 ./code 2 will print 1 2

mpirun -np 2 ./code 2 will block

Q.5

```
int main( int argc, char *argv[])
{
  int N=atoi(argv[1]), myrank, size, count;
  int arr[N], recvarr[N];
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);

  if (myrank == 0)
    // int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
    MPI_Send(arr, N, MPI_INT, 1, 99, MPI_COMM_WORLD);
  else
  {
    // int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status * status)
    MPI_Recv(recvarr, 2*N, MPI_INT, 0, 99, MPI_COMM_WORLD, &status);
    // int MPI_Get_count(const MPI_Status * status, MPI_Datatype datatype, int *count)
    MPI_Get_count (&status, MPI_INT, &count);
    printf ("%d %d\n", myrank, count);
  }

  MPI_Finalize();
  return 0;
}
```

Max. score: 3; Neg. score: 1; Your score: 3

mpirun –np 2 ./code 10 will block

mpirun –np 30 ./code 10 will block

mpirun –np 2 ./code 10 will run fine

mpirun –np 30 ./code 10 will run fine

Q.6

```
//int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
//int MPI_Ssend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm
comm)
//int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm,
MPI_Status * status)

Assume that myrank is the rank of a process in MPI_COMM_WORLD

Even ranks
-----------
MPI_Ssend 40 bytes to the next consecutive rank in MPI_COMM_WORLD, using the tag "myrank"
MPI_Send 400 bytes to the next consecutive rank in MPI_COMM_WORLD, using the tag "myrank*2"


Odd ranks
-----------
MPI_Recv 40 bytes from the previous rank in MPI_COMM_WORLD, using the tag "myrank"
MPI_Recv 400 bytes from the previous rank in MPI_COMM_WORLD, using the tag "myrank*2"


Select all correct answers.
```

CS633A: Parallel Computing
Users Online : 21

place rank 13 and rank 14 on the same node

The above code will work for 400 processes if the tag is changed to "myrank-1" and "(myrank-1)*2" in the 2 MPI_Recv calls respectively.

mpirun -np 50 -hosts csews10,csews11,csews12,csews13,csews14 ./code (the above code) will place rank 3 on csews13 and rank 4 on csews14

The above code will work for 200 processes if the tag is changed to "myrank+1" and "(myrank+1)*2" in the 2 MPI_Send calls respectively.

The above code will work for even number of processes if the sender and receiver are placed on the same node, even if the tags do not match for a send-receive pair.

Q.7 //int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
//int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status * status)

Rank 0

MPI_Send (data, 10, MPI_INT, 1, 20, MPI_COMM_WORLD)
MPI_Send (data, 10, MPI_INT, 1, 20, MPI_COMM_WORLD)

Rank 1

MPI_Recv (data, 10, MPI_INT, 0, 20, MPI_COMM_WORLD, &status)
MPI_Recv (data, 10, MPI_INT, 0, 20, MPI_COMM_WORLD, &status)

The above code snippet is not safe and it will not run successfully.

Max. score: 2; Neg. score: 0.5; Your score: 2

false

true

Q.8 "mpirun -np 100 -hosts host1:50,host2:49 ./parallelsum"

Max. score: 2; Neg. score: 0.5; Your score: -0.5

will run on 100 CPU cores

will spawn 100 processes

will give error

will run on 99 CPU cores

Score: 13.5