Mid-semester Examination

CS633: Parallel Computing Date: 19th February 2024

Roll number 200026

Name ABHISHEK PARDHI

PLEASE SWITCH OFF YOUR MOBILE PHONE AND KEEP IT IN YOUR BAG

- *Total time is 80 minutes. Total marks = 70.*
- *All questions are compulsory. Marks are mentioned in parenthesis. There is negative marking in Section II. There is no partial marking in programming questions.*
- *Write your name and roll number on rough sheets as well.*
- *You **must** answer all questions **on** the question paper. Ample space has been provided below each question. You may use extra sheets for rough work only. EXTRA SHEETS WILL NOT BE COLLECTED. So think before writing the final answer.*
- *The MPI function prototypes and collective message size limits are given on the last page.*
- *Read the questions carefully before answering.*
- *All clarifications must be directed to the instructor only.*
- *Extra points will not be given for being verbose, so answer to the point, without being abrupt.*
- *You will be awarded **2 marks (bonus)** for neatness.*
- *DO NOT OPEN THE QUESTION PAPER BEFORE THE SCHEDULED TIME. IF YOU ARE FOUND DOING SO, 5 MARKS WILL BE DEDUCTED AND YOU WILL LOSE 5 MINUTES.*

Marks (To be filled in by the instructor)

1	2	3	4	5	6	7	8	9	10	~~11~~ 12	~~12~~ 13	II	Neatness	Total
3	5	3	4	4	4	4	0	5	0	10	4.5	7	1	54.5

I pledge to answer all questions by myself without taking any external help.

_____ _____ **(Sign your name)**

Section-I

Marks are in parenthesis.

I-1. There is no partial marking for this question.

```
#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
  int BUFFER = atoi (argv[1]);
  int arr[BUFFER];
  int myrank, size;
  MPI_Status status;
  double start_time;

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

  start_time = MPI_Wtime ();
  for (int ij=0; ij<10; ij++) {
   if (myrank < size-1)
    MPI_Send(arr, BUFFER, MPI_INT, size-1, myrank, MPI_COMM_WORLD);
   else
   {
    int count, recvarr[size][BUFFER];
    for (int i=0; i<size-1; i++)
      MPI_Recv(recvarr[i], BUFFER, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
   }
  }
  if (myrank == size-1) printf ("time= %lf\n", MPI_Wtime() - start_time);

  MPI_Finalize();
  return 0;
}
```

Match the output for the below three executions of the above code with the below three text boxes (which show the output). Write the correct option (A/B/C) below each text box. (3)

(A) `for i in `seq 1 5`; do mpirun -np 4 ./a.out 100; done`
(B) `for i in `seq 1 5`; do mpirun -np 8 ./a.out 100; done`
(C) `for i in `seq 1 5`; do mpirun -np 12 ./a.out 100; done`

```
time= 0.001180
time= 0.000138
time= 0.000126
time= 0.000105
time= 0.000097
```

A

```
time= 0.021519
time= 0.016031
time= 0.004027
time= 0.033644
time= 0.001333
```

C

```
time= 0.000308
time= 0.023562
time= 0.000307
time= 0.000308
time= 0.000286
```

B

3

I-2. There is no partial marking for this question.

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
  int rank, size, color, N=3, newrank, newsize, max;
  MPI_Comm newcomm;

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

  color = rank%N;

  MPI_Comm_split (MPI_COMM_WORLD, color, rank, &newcomm);
  MPI_Comm_rank (newcomm, &newrank);
  MPI_Reduce (&rank, &max, 1, MPI_INT, MPI_MAX, 0, newcomm);
  MPI_Bcast (&max, 1, MPI_INT, 0, newcomm);
  printf ("%d %d %d %d\n", rank, newrank, max, color);

  MPI_Finalize();
  return 0;
}
```

What is the output of the program for **mpirun -np 10 ./a.out | sort -k4n** (Write in sorted order of the 4th column). (5)

0 0 9 0
3 1 9 0
6 2 9 0
9 3 9 0
1 0 7 1
4 1 7 1
7 2 7 1
2 0 8 2
5 1 8 2
8 2 8 2

✓ (5)

I-3.

```
#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
  int BUFFER = atoi (argv[1]);
  int arr[BUFFER];
  int myrank, size;
  MPI_Status status;
  double start_time;

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

  start_time = MPI_Wtime ();
  for (int ij=0; ij<10; ij++) {
   if (myrank < size-1)
    MPI_Send(arr, BUFFER, MPI_INT, size-1, myrank, MPI_COMM_WORLD);
   else
   {
    int count, recvarr[size][BUFFER];
    for (int i=0; i<size-1; i++)
      MPI_Recv(recvarr[i], BUFFER, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
   }
  }
  printf ("Rank %d: time= %lf\n", myrank, MPI_Wtime () - start_time);

  MPI_Finalize();
  return 0;
}
```

Write two most important overall observations (with brief explanations) from the below two executions (runs): (4)

Run 1

```
mpirun -np 6 -hosts csews1:3,csews2:3 ./a.out 10000 | sort -k2n
Rank 0 on csews1: time= 0.002967
Rank 1 on csews1: time= 0.005490
Rank 2 on csews1: time= 0.003733
Rank 3 on csews2: time= 0.000796
Rank 4 on csews2: time= 0.000712
Rank 5 on csews2: time= 0.012079
```

Run 2

```
mpirun -np 6 -hosts csews1:3,csews2:3 ./a.out 1000000 | sort -k2n
Rank 0 on csews1: time= 1.006452
Rank 1 on csews1: time= 1.051192
Rank 2 on csews1: time= 1.089486
Rank 3 on csews2: time= 0.023303
Rank 4 on csews2: time= 0.023936
Rank 5 on csews2: time= 1.109516
```

Observation 1:

Processes with ranks 3 & 4 are taking way lesser time to communicate than 0,1,2 because 3, 4 & 5 are mapped on the same host csews2, leading to intra node communication, which is faster than inter node comm. for ranks 0,1&2.

2

Observation 2:

Processes 0,1,2 takes almost same time for completion because they are present on the same host. They all require only 1 hop to communicate with process 5. Similarly, 3 & 4 also takes almost same time.

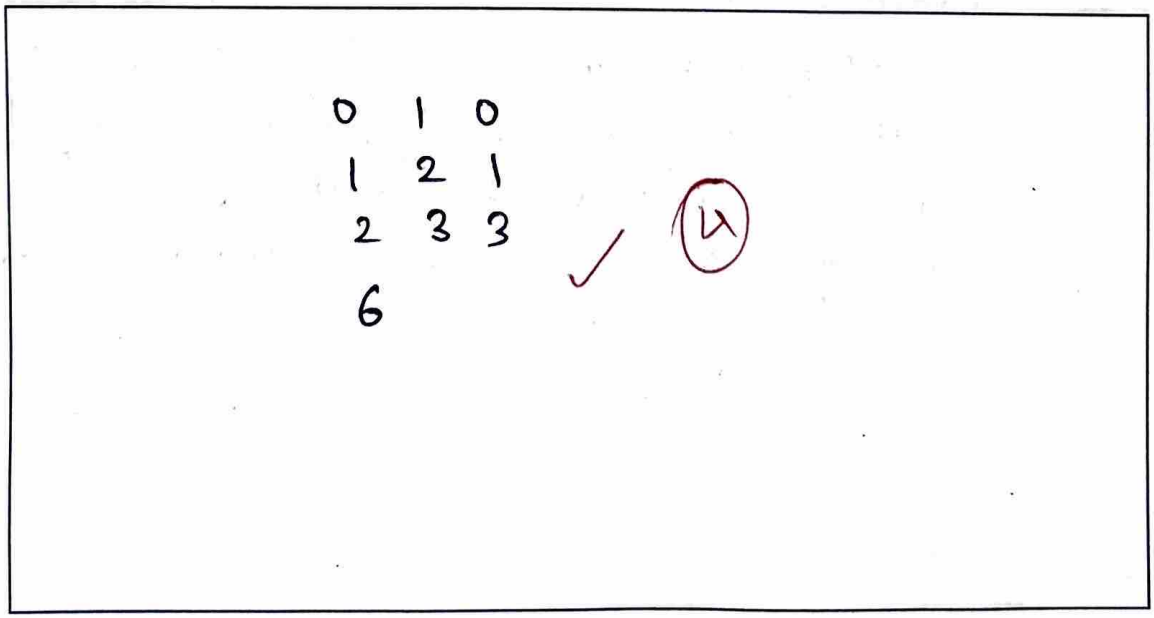
1

2+1 = 3

I-4. There is no partial marking for this question.

```
#include <stdio.h>
#include "mpi.h"

int main (int argc, char *argv[])
{
  int rank, size;

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

  int countAll[size], displArray[size], displ=0, count = rank+1;
  MPI_Gather (&count, 1, MPI_INT, countAll, 1, MPI_INT, 0, MPI_COMM_WORLD);

  if (!rank) {
     for (int i=0; i<size; i++)
       displArray[i] = displ, displ += countAll[i];
     for (int i=0; i<size; i++)
       printf ("%d %d %d\n", i, countAll[i], displArray[i]);
     printf ("%d\n", displ);
  }

  MPI_Finalize();
  return 0;
}
```

Write the output of the program for **mpirun –np 3 ./a.out | sort –k1n** (Write the output in sorted order of column 1) (4)



I-5. There is no partial marking for this question.

```
#include <stdio.h>
#include "mpi.h"

int main (int argc, char *argv[])
{
  int rank, size, data;
  MPI_Status status;
  MPI_Request request;

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

  data = rank/2;
  if (rank < size/2)
     MPI_Send (&data, 1, MPI_INT, rank+size/2, rank, MPI_COMM_WORLD);
  else {
     MPI_Irecv (&data, 1, MPI_INT, rank-size/2, rank-size/2, MPI_COMM_WORLD,
&request);
     MPI_Wait (&request, &status);
  }

  printf ("%d %d\n", rank, data);

  MPI_Finalize();
  return 0;
}
```

Write the output for **mpirun -np 6 ./a.out | sort -k1n** (Write in sorted order). (4)

0 0
1 0
2 1
3 0
4 0
5 1

✓ (4)

I-6. There is no partial marking for this question.

```
#include <stdio.h>
#include "mpi.h"

int main (int argc, char *argv[])
{
  int rank, size, color, N=2, newrank, newsize;
  MPI_Comm newcomm;
  MPI_Status status;
  MPI_Request request;

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  color = rank%N;

  MPI_Comm_split (MPI_COMM_WORLD, color, rank, &newcomm);
  MPI_Comm_rank (newcomm, &newrank);
  MPI_Comm_size (newcomm, &newsize);

  if (newrank == newsize-1 && newsize > 1)
     MPI_Send (&newrank, 1, MPI_INT, 0, color, newcomm);
  else if (newrank == 0 && newsize > 1) {
     MPI_Irecv (&newrank, 1, MPI_INT, newsize-1, color, newcomm, &request);
     MPI_Wait (&request, &status);
  }

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

  MPI_Finalize();
  return 0;
}
```

Write the output for **mpirun -np 6 ./a.out |sort -k1n** (Write in sorted order) (4)

0 2
1 2
2 1
3 1
4 2
5 2

✓ (4)

I-7. There is no partial marking for this question.

```
#include <stdio.h>
#include "mpi.h"

int main (int argc, char *argv[])
{

  int rank, size, color, root=0;
  int sendbuf, recvbuf;

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

  sendbuf = rank;
  MPI_Reduce(&sendbuf, &recvbuf, 1, MPI_INT, MPI_MAX, root, MPI_COMM_WORLD);
  color = recvbuf;
  recvbuf = rank;
  printf ("%d: color = %d recvbuf = %d\n", rank, color, recvbuf);

  MPI_Finalize();

  return 0;

}
```

Write the output (sort by column 1) of the program for `mpirun -np 4 ./a.out |sort -k1n` (4)

0 : color = 3 recvbuf = 0

1 : color = 0 recvbuf = 1

2 : color = 0 recvbuf = 2

3 : color = 0 recvbuf = 3

✓

4

I-8. There is no partial marking for this question.

```
#include <stdio.h>
#include "mpi.h"
#define BUFFER 10485760

int main( int argc, char *argv[])
{
  int arr[BUFFER] = {0};
  int myrank, size;
  MPI_Status status1, status2;
  int arr1=10, arr2=100;
  int recvarr1=0, recvarr2=0;

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

  if (myrank == 0)
  {
    MPI_Send(&arr1, 1, MPI_INT, 1, 1, MPI_COMM_WORLD);
    MPI_Send(&arr2, 1, MPI_INT, 1, 1, MPI_COMM_WORLD);
    MPI_Send(&arr1, 1, MPI_INT, 2, 2, MPI_COMM_WORLD);
    MPI_Send(&arr2, 1, MPI_INT, 2, 2, MPI_COMM_WORLD);
  }
  else
  {
    MPI_Recv(&recvarr1, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
&status1);
    MPI_Recv(&recvarr2, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
&status2);
  }
  printf ("Rank %d: %d %d\n", myrank, recvarr1, recvarr2);

  MPI_Finalize();
  return 0;
}
```

Write the output (sort by column 1) of the program for **mpirun -np 3 ./a.out | sort -k1n** (4)

Rank 0 : 0 0

Rank 1 : 10 20

Rank 2 : 10 20

I-9. There is no partial marking for this question.

```
#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"

int main( int argc, char *argv[])
{
  int BUFFER = atoi (argv[1]);
  int arr[BUFFER], recvarr[BUFFER];
  double darr[BUFFER/2], drecvarr[BUFFER/2];
  int myrank, size;
  MPI_Status status;
  double start_time, time[2], gTime[2];

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

  if (size%2 != 0) {
    printf ("Use even number of processes to run this code.\n");
    return 0;
  }

  start_time = MPI_Wtime ();
  for (int ij=0; ij<10; ij++) {
   if (myrank % 2 == 0)
    MPI_Send(arr, BUFFER, MPI_INT, myrank+1, myrank, MPI_COMM_WORLD);
   else
    MPI_Recv(recvarr, BUFFER, MPI_INT, myrank-1, myrank-1, MPI_COMM_WORLD,
&status);
  }
  time[0] = MPI_Wtime() - start_time;

  MPI_Barrier (MPI_COMM_WORLD);

  start_time = MPI_Wtime ();
  for (int ij=0; ij<10; ij++) {
   if (myrank % 2 == 0)
    MPI_Send(darr, BUFFER/2, MPI_DOUBLE, myrank+1, myrank, MPI_COMM_WORLD);
   else
    MPI_Recv(drecvarr, BUFFER/2, MPI_DOUBLE, myrank-1, myrank-1,
MPI_COMM_WORLD, &status);
  }
  time[1] = MPI_Wtime() - start_time;

  MPI_Reduce (time, gTime, 2, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
  if (!myrank) printf ("time1 = %lf: time2 = %lf\n", gTime[0], gTime[1]);

  MPI_Finalize();
```

```
  return 0;
}
```

Match the output for the below three executions of the above code with the below three text boxes (which show the output). Write the correct option (A/B/C/D) below each text box. Explain your answer. (5)

(A) `mpirun -np 8 -hosts csews10,csews20 ./a.out 10000`
(B) `mpirun -np 8 -hosts csews10:4,csews20:4 ./a.out 10000`
(C) `mpirun -np 8 -hosts csews10,csews20 ./a.out 10000000`
(D) `mpirun -np 8 -hosts csews10:4,csews20:4 ./a.out 10000000`

time1 = 0.000814: time2 = 0.000595 time1 = 0.000524: time2 = 0.000323	time1 = 0.114672: time2 = 0.111407 time1 = 0.118924: time2 = 0.110683
B	D
time1 = 13.718738: time2 = 13.903650 time1 = 13.728285: time2 = 13.716866	time1 = 0.016347: time2 = 0.014986 time1 = 0.014775: time2 = 0.014056
C	A ✓

First, C & D have large data size, so they will take more time.
second, in NN exchange, 4 adjacent processes communicate, so having adjacent processes on same node will lead to intra node comm. instead of internode comm. Hence, B & D will take lesser time since they are mapping adjacent processes on same node.

5

I-10. There is no partial marking for this question.

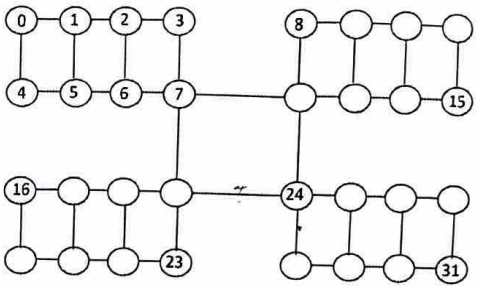
```
#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
  int myrank, *buf;
  int count = atoi(argv[1]);
  buf = (int *) malloc (count * sizeof(int));

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

  // initialize data
  for (int i=0; i<count; i++)
      buf[i] = myrank + i*i;

  double sTime = MPI_Wtime();
  MPI_Bcast(buf, count, MPI_INT, 0, MPI_COMM_WORLD);
  double eTime = MPI_Wtime();

  printf ("%d: %lf\n", myrank, eTime - sTime);

  MPI_Finalize();
  return 0;

}
```

The above code was executed using: **mpirun -np 4 ./a.out 10000 | sort -k1n**
Which of the below output(s) are feasible for this execution? Put a tick mark above the correct text boxes or write the option numbers in the provided space. Explain your answer in brief. (4)

A	B	C	D
0: 0.000090 1: 0.000040 2: 0.000075 3: 0.000112	0: 0.000070 1: 0.000109 2: 0.000071 3: 0.000138	0: 0.000086 1: 0.000094 2: 0.000099 3: 0.000124	0: 0.000080 1: 0.000073 2: 0.000064 3: 0.000140

(B): Using Recursive Doubling, we will see that (0→1) and (2→3) happens in the last step, hence 1 & 3 should take more time in comparison to 0 & 2

I-12. A mini-cluster of 32 nodes is shown in the right-side figure. The node numbers (0- 31) are shown in the figure (XY ordering in each quarter). Two 16-process jobs (programs) are running on this cluster. Assume XY static routing on the network, i.e. message from node 0 to node 7 travels via node 3 always, similarly message from node 0 to node 5 travels via node 1 always.



10

Program A is allocated nodes numbered 0 – 15. Program A has one collective communication function MPI_Bcast. Root is rank 0. Rank 0 broadcasts **512** bytes to all ranks in MPI_COMM_WORLD.

Program B is allocated nodes 16 – 31. Program B has one collective communication function MPI_Scatter. Root is rank 0. Rank 0 scatters **100** bytes to each and every rank in MPI_COMM_WORLD.

During the execution of Programs A and B,

(a) Derive the number of bytes transferred over the network link between node 3 to node 7.
(b) Derive the number of bytes transferred over the network link between node 0 to node 4.
(c) Derive the number of bytes transferred over the network link between node 7 to node 12.
(d) Derive the number of bytes transferred over the network link between node 19 to node 24.
(e) Derive the number of bytes transferred over the network link between node 24 to node 28. (2+2+2+2+2)

(a) If we use ~~recursive~~ binomial algorithm for Bcast, then 3–7 link is used only once in comm. between 0 and 8 in first step. Hence, Ans = 512 bytes 2

(b) Similarly, 0–4 link is used only once in comm. b/w 0 and 4 in 2nd step, Hence, Ans = 512 bytes 2

(c) Ans = 512 bytes
7–12 link is used → 1st step b/w 0 and 8 2

(d) 19–24 link is used only once in ~~vector~~ halving algorithm for reduce. → 1st step b/w 16 and 24 2

(e) Ans = 800 bytes (since initially 16 will have $\frac{1600}{2}$ = 1600 bytes and in 1st step it will send half of this to 24.)

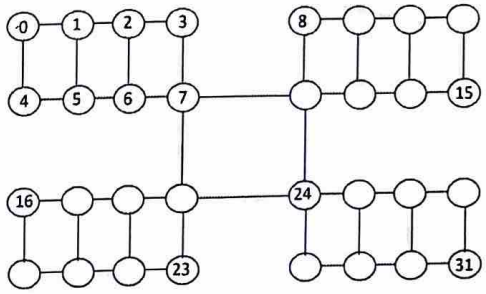
24–28 link is used only once → 2nd step b/w 24 and 28 2

Ans = 400 bytes (since comm. size at 2nd step is $\frac{n}{4} = \frac{1600}{4} = 400$)

Total = 4 1/2

I-13. A mini-cluster of 32 nodes is shown in the right-side figure. Considering this network:

(a) What is the bisection width (explain)
(b) What is the diameter (explain)
(c) Derive the total number of hops in case of Reduce (100 bytes) on 16 processes (0 – 15) executing on the two upper quarters of the cluster (1 process per node). Assume XY ordering of ranks (as shown). Derive using the communication step number. Assume that communications per step occur simultaneously. (2+2+5)



(a) Bisection width is 2. Cutting the links between nodes 7 and 19 & 12 and 24 will give us two equal halves. 2

(b) Diameter is 10. From node 0 to node 31 it takes 10 links to cross, and it is min. 2

(c)

Communication steps	# Hops
1st →	8
2nd →	8
3rd →	2
4th →	6
Total =	24

No derivation! Wrong answer

1/2

Section-II

*Fill in the blanks. There are 10 questions in this section. Each question has 1 mark. There is **negative** marking of 0.5 mark. 0 marks awarded for typos.*

II-1. A new system, called PooreSau, in the CSE department of IIT Kanpur has 100 nodes (host1, ... host100). Each node has 8 cores with 32 GB RAM.

(a) You ran your code using the command: mpirun –np 40 –hosts host1,host2,host3,host4 ./a.out. The number of compute nodes this program will execute on is 4

(b) You ran your code using the command: mpirun –np 50 –hosts host1,host2,host3,host4 ./a.out. Will this program fail to run due to insufficient number of nodes? (Yes/No) No

(c) You ran your code using the command: mpirun –np 20 –hosts host1,host2,host3,host4 ./a.out. The number of processes per node is 5

II-2. The above mentioned system, PooreSau, has been designed to run at a peak throughput of 10 GFLOPS (floating point operations per second). This means that (Write True/False)

(i) A job that runs on PooreSau will take a maximum of 10 ns to run False

(ii) No job will run at more than 10 GFLOPS True

(iii) It is possible that a code that runs on PooreSau and gives a throughput of less than 10 GFLOPS cannot be optimized further False

II-3. The communication time complexity of

(i) MPI_Reduce (M processes, D bytes, C cost of compute/byte) using Recursive Doubling algorithm is $\log M (L + D/B + DC)$

(ii) MPI_Allgather (M processes, D/P bytes per process) algorithm used for large message sizes is $(M-1)L + \frac{(M-1)}{M} \times \frac{D}{B}$ (Ring Algorithm)

II-4. Select the correct option. The message size per communication step for

(i) MPI_Scatter using distance halving algorithm increases/decreases/remains constant decreases

(ii) MPI_Allgather using recursive doubling algorithm increases/decreases/remains constant increases

----------END----------

A.MPI FUNCTIONS (PROTOTYPES)

1) int **MPI_Comm_rank**(MPI_Comm comm, int *rank)
2) int **MPI_Comm_size**(MPI_Comm comm, int *size)
3) int **MPI_Comm_split**(MPI_Comm comm, int color, int key, MPI_Comm * newcomm)
4) int **MPI_Group_incl**(MPI_Group group, int n, const int ranks[], MPI_Group *newgroup)
5) int **MPI_Comm_create_group**(MPI_Comm comm, MPI_Group group, int tag, MPI_Comm * newcomm)
6) int **MPI_Comm_group**(MPI_Comm comm, MPI_Group * group)
7) int **MPI_Send**(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
8) int **MPI_Recv**(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
9) int **MPI_Barrier**(MPI_Comm comm)
10) int **MPI_Bcast**(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
11) int **MPI_Gather**(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
12) int **MPI_Scatter**(const void *sendbuf, int sendcount, MPI_Datatype sendtype,void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
13) int **MPI_Allgather**(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype,MPI_Comm comm)
14) int **MPI_Reduce**(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
15) int **MPI_Allreduce**(const void *sendbuf, void *recvbuf, int count,MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
16) int **MPI_Alltoall**(const void *sendbuf, int sendcount, MPI_Datatype sendtype,void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)
17) int **MPI_Isend**(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)
18) int **MPI_Irecv**(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)
19) int **MPI_Wait**(MPI_Request *request, MPI_Status *status)
20) int **MPI_Waitall**(int count, MPI_Request array_of_requests[], MPI_Status array_of_statuses[])
21) double **MPI_Wtime**(void)

B.MPI COLLECTIVE FUNCTION MESSAGE SIZE LIMITS

MPIR_CVAR_BCAST_SHORT_MSG_SIZE =12288
MPIR_CVAR_BCAST_LONG_MSG_SIZE =524288

MPIR_CVAR_REDUCE_SHORT_MSG_SIZE =2048
MPIR_CVAR_ALLREDUCE_SHORT_MSG_SIZE=2048

MPIR_CVAR_ALLGATHER_SHORT_MSG_SIZE=81920

MPIR_CVAR_ALLGATHER_LONG_MSG_SIZE=524288