**End-semester Examination (Semester 2023-24-2)**

**CS633: Parallel Computing  Date: 30th April 2024  Time: 8:00 – 11:00 AM**

**Roll number** 200026

**Name** ABHISHEK PARDHI

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

- *Total time is 180 minutes. Total marks = 100.*
- *All questions are compulsory. Marks are mentioned in parenthesis. There is no partial marking in programming questions.*
- *Write your name and roll number on extra 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 not be awarded any marks in case of unclear/illegible response or overwriting.*
- *All typos will be considered as errors. So please revise your answers.*
- *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 |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 4 | 3 | 2 + 4 | 6 | 7 | 4+4 | 3+2+3 | 8 |
| | | | | | | | | | |
| 11 | 12 | 13 | 14 | 15 | Total | | | | |
| 15 | 3+4 | 4+4 | 4 | 4+5 | 100 | | | | |
| | | | | | | | | | |

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

ABHISHEK PARDHI **(Signature – write your name)**

1. There is no partial marking for this question.

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

int main( int argc, char *argv[])
{
 int myrank, size, sendval[2], maxval[2];
 MPI_Status status;
 MPI_Init( &argc, &argv );
 MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
 MPI_Comm_size( MPI_COMM_WORLD, &size );
 for (int i=0; i<2; i++)
  sendval[i] = myrank+i;
 MPI_Allreduce(sendval, maxval, 2, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
 printf ("%d maxval[1]=%d\n", myrank, maxval[1]);
 MPI_Finalize();
 return 0;
}
```

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

0 maxval[1] = 10

1 maxval[1] = 10

2 maxval[1] = 10

3 maxval[1] = 10

2. Match the output. There is no partial marking. (3 marks)

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
#define UL 100000

int main (int argc, char *argv[])
{
 int myrank, size, val[UL], mval[UL];
 MPI_Status status;
 double sTime, eTime, time;

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

 for (int i=0; i<UL; i++)
  val[i] = myrank+i;

 sTime = MPI_Wtime();
 MPI_Allreduce(val, mval, UL, MPI_INT, MPI_MAX, MPI_COMM_WORLD);
 eTime = MPI_Wtime()-sTime;

 MPI_Reduce (&eTime, &time, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
 if (!myrank) printf ("%lf\n", time);

 MPI_Finalize();
 return 0;
}
```

Executions

A. for i in `seq 1 3` ; do mpirun -np 4 ./a.out ; done
B. for i in `seq 1 3` ; do mpirun -np 32 -hosts host1,host2,host3,host4 ./a.out ; done

| |
|---|
| 0.014647 |
| 0.015682 |
| 0.033032 |

B

| |
|---|
| 0.001837 |
| 0.001789 |
| 0.003058 |

A

3. Write the output of **mpirun -np 3 ./a.out 100**. There is no partial marking. (4 marks)

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

int main (int argc, char *argv[])
{
 int N=atoi(argv[1]), myrank, size, count=0;
 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) {
  MPI_Send(arr, N, MPI_INT, 1, 1, MPI_COMM_WORLD);
  MPI_Send(arr, N/2, MPI_INT, 2, 2, MPI_COMM_WORLD);
 }
 else
 {
  MPI_Recv(recvarr, N, MPI_INT, 0, myrank, MPI_COMM_WORLD, &status);
  MPI_Get_count (&status, MPI_INT, &count);
 }

 printf ("%d %d\n", myrank, count);
 MPI_Finalize();
 return 0;
}
```

0 0

1 100

2 50

4.

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

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

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

 for (i=0; i<count; i++)
    buf[i] = myrank + i*i;

 for (i = 0; i< 50; i++)
  MPI_Bcast(buf, count, MPI_INT, 0, MPI_COMM_WORLD);

 MPI_Finalize();
 return 0;
}
```

Information from the IPM profiles are specified below for two executions A and B. Each job was run on 4 nodes of HPC2010. Mention the relation (<, >, =) between the integers X and Y with reason. (3 marks)

A. mpirun -machinefile hostfile -np 32 ./a.out X

| # | [time] | [count] | <%wall> |
|---|---|---|---|
| # MPI_Bcast | 0.12 | 1600 | 1.58 |

B. mpirun -machinefile hostfile -np 32 ./a.out Y

| # | [time] | [count] | <%wall> |
|---|---|---|---|
| # MPI_Bcast | 155.30 | 1600 | 95.67 |

X < Y

Since it takes more time to broadcast a larger message size. (time $\propto$ n)

5. Output of the below code is shown for two executions.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include "mpi.h"
 4
 5 int main (int argc, char *argv[])
 6 {
 7       int numprocs, rank, arraySize, local, i;
 8       arraySize = atoi (argv[1]);
 9       double time=0.0, max_time=0.0;
10       int array[arraySize], Local=1, Global=0;
11       MPI_Init(&argc,&argv);
12       MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
13       MPI_Comm_rank(MPI_COMM_WORLD, &rank);
14       local = arraySize/numprocs; //assume arraySize is divisble by numprocs
15       for (i=local*rank; i<local*(rank+1); i++)
16             array[i] = rank*i+10;
17       for (i=local*rank; i<local*(rank+1); i++)
18             Local *= array[i];
19       if (numprocs == 1)
20             Global = Local;
21       else
22             MPI_Reduce (&Local, &Global, 1, MPI_INT, MPI_PROD, 0, MPI_COMM_WORLD);
23       if (!rank) printf ("%d\n", Global);
24       MPI_Finalize();
25 }
```

A. mpirun -np 2 ./a.out 4 → Output is **15600**
B. mpirun -np 2 ./a.out 12 → Output is **-1237491712**

a) Is the output of A correct? (2 marks)
b) Is the output of B correct? If not, is there a bug in the code? What is the bug? Which line numbers need to be changed and write the changes clearly along with the line numbers. (4 marks)

a) Yes, since Local = 100 for rank=0 & Local = 156 for rank=1

b) NO, the bug in the code is in the line 10 & 23 & 22
Since we are multiplying very large numbers while doing Reduce, the output which should be stored in "Global" is of the order of $\approx 10^{14}$ which is out of limit of "int" datatype. we should change "int" to "long" on line 10 & line 23 change "%d" to "%ld"
& For line 22, change "MPI_INT" to "MPI_LONG"

6. Fill in the blanks. You cannot add any new code. 0 marks will be awarded for syntax error. There is no partial marking. (6 marks)

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

int main (int argc, char *argv[])
{
 char hostname[64], cnode[20];
 int rank, size, len, color, newrank, val, i, j=-1;
 MPI_Comm newcomm;

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

 // get processor name
 MPI_Get_processor_name(hostname, &len);

 // get node number
 for (i=0; i<len; i++)
  if (isdigit(hostname[i]))
    cnode[++j]=hostname[i];
 cnode[++j]='\0';
 sscanf(cnode, "%d", &color); // numeric part of the hostname

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

 MPI_Reduce (&rank, &val, 1, ____MPI_INT____, ____MPI_MAX____, 0,
 ____newcomm____);

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

 MPI_Finalize();
 return 0;
}
```

Output of code for **mpirun -np 8 -hosts csews1:4,csews2:4 ./a.out** is:

0 3
4 7

7. There is no partial marking for this question. (7 marks)

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include "mpi.h"

4.  int main (int argc, char *argv[])
5.  {
6.  int myrank, size, recvcount;
7.  MPI_Status status;
8.  MPI_Datatype newtype;
9.  MPI_Init(&argc, &argv);
10. MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
11. int N = atoi (argv[1]);
12. int data[N][N];
13. int count = N;
14. int blocklengths[N], displacements[N];
15. for (int i=0; i<N; i++)
16.       blocklengths[i] = i+1, displacements[i] = i*N;
17. for (int i=0; i<N; i++)
18.     for (int j=0; j<N; j++)
19.           data[i][j]=0;
20. MPI_Type_indexed (count, blocklengths, displacements, MPI_INT, &newtype);
21. MPI_Type_commit (&newtype);
22. if (myrank == 0)
23. {
24.  for (int i=0; i<N; i++)
25.    for (int j=0; j<N; j++)
26.      data[i][j]=i+j+1;
27.  MPI_Send(data, 1, newtype, 1, 99, MPI_COMM_WORLD);
28. }
29. else if (myrank == 1) {
30.  MPI_Recv(data, 1, newtype, 0, 99, MPI_COMM_WORLD, &status);
31.   for (int i=0; i<N; i++) {
32.      for (int j=0; j<N; j++)
33.          printf ("%d ", data[i][j]);
34.      printf ("\n");
35.   }
36. }
37. MPI_Type_free (&newtype);
38. MPI_Finalize();
39. return 0;
40. }
```

The desired output of the above code for **mpirun –np 2 ./a.out 4** is

```
1 2 3 4
0 3 4 5
0 0 5 6
0 0 0 7
```

However, the above code is giving the output as

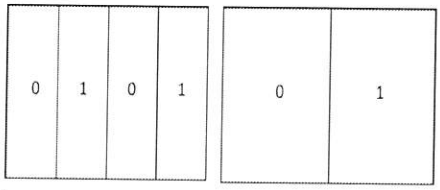
```
1 0 0 0
2 3 0 0
3 4 5 0
4 5 6 7
```

What is the bug in the code, fix the bug, write the rectified line(s) along with the line number(s).

The bug in the code is in line 16. It should be:

→ blocklengths [i] = N - i , displacements [i] = (N+1) x i ;

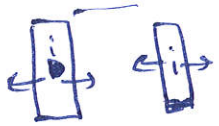
8. The left figure illustrates 1D block cyclic domain decomposition, the right figure illustrates 1D block decomposition for 2 processes (0 and 1). Assume N x N domain and P processes. Derive the communication to computation ratio for nearest neighbor 2D communication (5-point stencil) for (a) 1D block cyclic domain decomposition (b) 1D block domain decomposition. Assume an even number of processes. (4+4 marks)




(a) Communication = 4N

Computation = $N^2/P$

$\frac{\text{communication}}{\text{computation}} = \frac{4P}{N}$




for ith process 4 send/recv will take place per row → 4N total

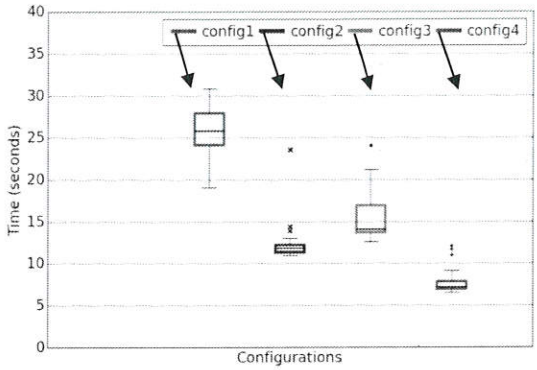
(b) Communication = 2N

Computation = $N^2/P$

$\frac{\text{Communication}}{\text{computation}} = \frac{2P}{N}$


for ith process 2 send/recv will take place per row → 2N total

9. The collective I/O times (from HPC2010) are shown in the graph for four configurations (30 runs each). HPC2010 has Lustre file system. These four configurations are shown in column 1 of the table (in a different order). 'cb' refers to the number of aggregator nodes per compute node, 'SS' refers to the Lustre stripe size, and 'SC' refers to the Lustre stripe count. (a) Fill in the missing configuration numbers in column 2 of the table (b) State the reason(s) for your response to (a). (c) Mention two observations from the plot. (3 + 2 + 3 marks)




| I/O parameters (32 processes on 4 nodes of HPC2010) | Corresponding configuration# |
|---|---|
| File size = 8 GB, cb = 1, SS = 1M, SC = 1 | config 2 |
| File size = 8 GB, cb = 2, SS = 2M, SC = 8 | config 4 |
| File size = 16 GB, cb = 1, SS = 1M, SC = 1 | config 1 |
| File size = 16 GB, cb = 2, SS = 2M, SC = 8 | config3 |

9 (b) The lowest time should be taken by the config. which has lower file size and higher number of aggregators. Therefore filesize = 8GB & cb = 2 should be config 4. The remaining two config (1 & 2) can be deduced by just looking at the file size (the only difference in the two configs) and time for config 1 is more hence it should have taken more filesize for I/O → 16 GiB.

9 (c) (1) Increasing the number of aggregators from 1 to 2 decreases the collective I/O time. This can be seen by comparing config. 1 & 3 (or 2 & 4).

(2) Collective I/O time increases as file size increases. This can be seen by comparing config. 1 & 2. (or config 3 & 4).

10. Fill up the communication matrix based on the communications in the following code snippet. Assume the binomial tree algorithm for MPI_Gather. The code is invoked for 8 processes. Assume size of integer = 4 bytes, size of double = 8 bytes. There is no partial marking. (8 marks)

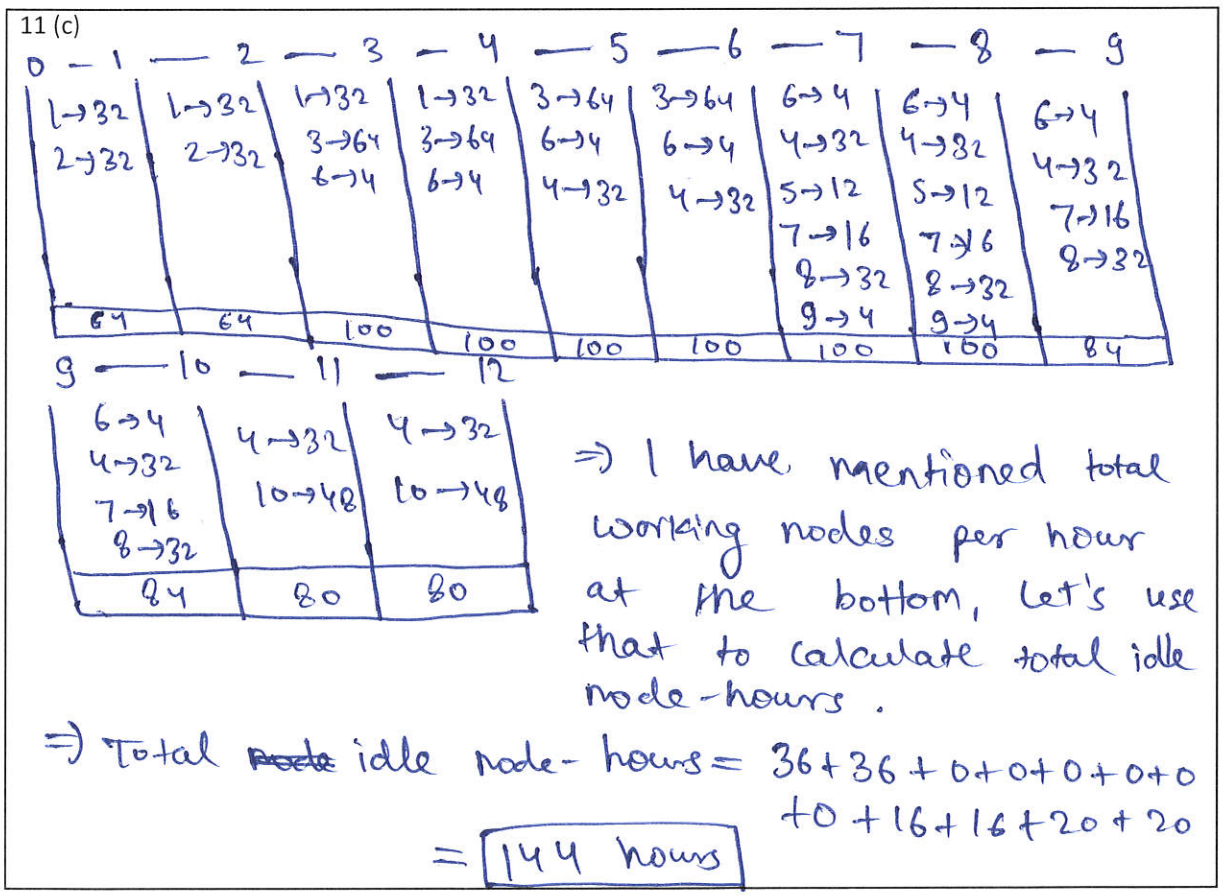
```
int arrSize = 128;
double message[arrSize];

srand(time(NULL));
for (int i = 0; i < arrSize; i++)
  message[i] = (double)rand();

double recvMessage[arrSize * numtasks];
MPI_Gather(message, arrSize, MPI_DOUBLE, recvMessage, arrSize, MPI_DOUBLE, root,
MPI_COMM_WORLD);
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1024 | 2048 | | 4096 | | | |
| 1024 | | | | | | | |
| 2048 | | | 1024 | | | | |
| | | 1024 | | | | | |
| 4096 | | | | | 1024 | 2048 | |
| | | | | 1024 | | | |
| | | | | 2048 | | | 1024 |
| | | | | | | 1024 | |

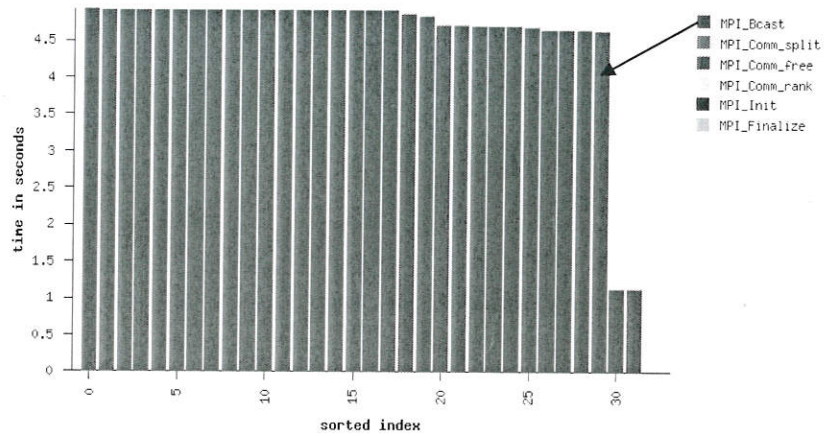
11. The details of jobs submitted to a supercomputer are shown in the table. The submit time is given in the second column. The supercomputer has 100 nodes and uses FCFS with backfilling scheduling policy. Deduce the following for the jobs: (a) Start time and (b) Wait time of each job. Write in columns 5 and 6. (c) Derive the total idle node-hours from start to end (time when the last job exits). (15 marks)

| Job ID | Submit time | Requested # nodes | Requested wall-clock time (hours) | Start time 11 (a) | Wait time (hours) 11 (b) |
|---|---|---|---|---|---|
| 1 | 00:00 | 32 | 4 | 00:00 | 0 |
| 2 | 00:00 | 32 | 2 | 00:00 | 0 |
| 3 | 00:00 | 64 | 4 | 02:00 | 2 |
| 4 | 01:00 | 32 | 8 | 04:00 | 4 |
| 5 | 01:00 | 12 | 2 | 06:00 | 6 |
| 6 | 02:00 | 4 | 8 | 02:00 | 2 |
| 7 | 02:00 | 16 | 4 | 06:00 | 6 |
| 8 | 03:00 | 32 | 4 | 06:00 | 6 |
| 9 | 03:00 | 4 | 2 | 06:00 | 6 |
| 10 | 04:00 | 48 | 2 | 10:00 | 10 |

11 (c)

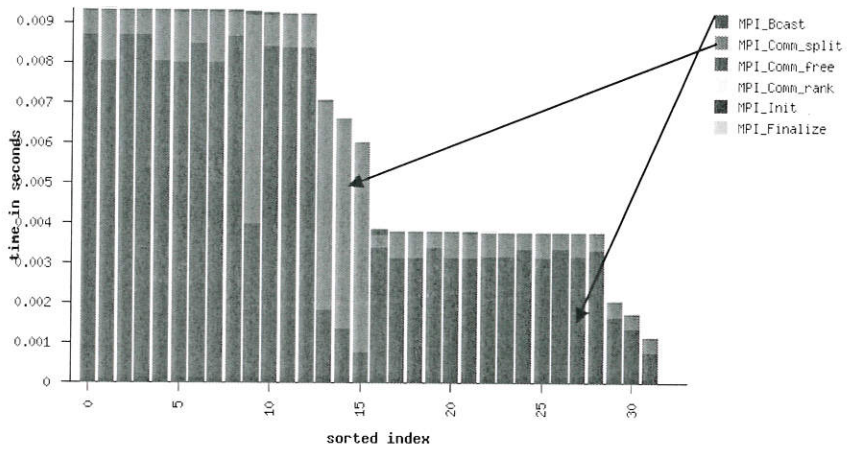


12.



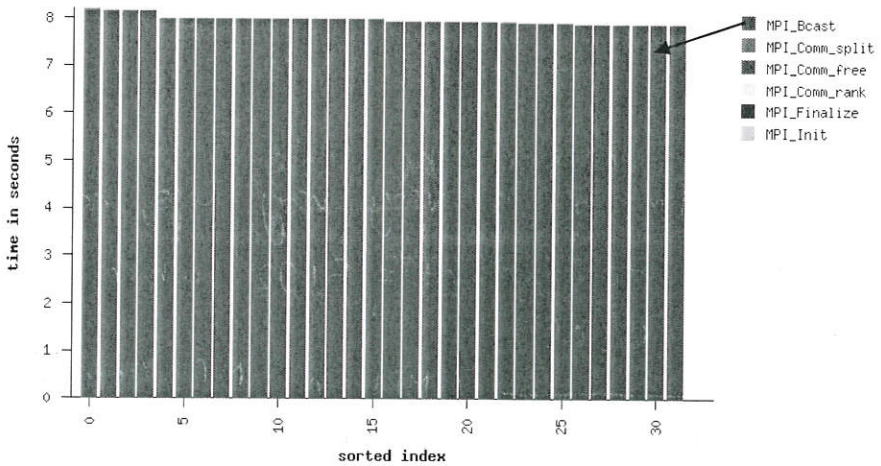

A

| MPI function name | %MPI |
|---|---|
| MPI_Bcast | 99.97 |
| MPI_Comm_split | 0.03 |




B

| MPI function name | %MPI |
|---|---|
| MPI_Bcast | 79.28 |
| MPI_Comm_split | 19.97 |




C

| MPI function name | %MPI |
|---|---|
| MPI_Bcast | 99.99 |
| MPI_Comm_split | 0.01 |

Three IPM profiles (A, B, C) are shown corresponding to three executions of the below code. All executions are performed on 4 nodes of HPC2010 supercomputer with linear (i.e. sequential/default) placement of ranks on each node. (a) Write the corresponding figure/profile number for the three run commands in the below table and (b) State two reasons for your response in (a). (3+4 marks)

| Run command | Corresponding figure/profile number (from previous page) |
|---|---|
| mpirun -np 32 ./a.out 8192 2 | B |
| mpirun -np 32 ./a.out 8388608 2 | C |
| mpirun -np 32 ./a.out 8388608 8 | A |

```
#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"
int main (int argc, char *argv[])
{
  int myrank, *buf;
  int i, count = atoi(argv[1]), mod = atoi(argv[2]);
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  MPI_Comm newcomm;
  buf = (int *) malloc (count * sizeof(int));
  for (i=0; i<count; i++)
      buf[i] = myrank + i*i;
  int color = myrank%mod;
  int newrank, newsize;
  MPI_Comm_split (MPI_COMM_WORLD, color, myrank, &newcomm);
  MPI_Comm_rank(newcomm, &newrank);
  for (i = 0; i< 50; i++)
    MPI_Bcast(buf, count, MPI_INT, 0, newcomm);
  MPI_Comm_free(&newcomm);
  MPI_Finalize();
  return 0;
}
```

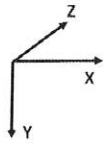
State the reasons.

12 (b)

Reason 1: Execution time for B is very very less, hence it should correspond to count = 8192 which is much less than 8388608.

Reason 2: ~~% MPI for mod = 8 shou~~ % MPI in MPI_Comm_Split for mod = 8 should be for than mod = 2 since mod=8 creates 8 groups so splitting will more time hence A (which has more % MPI in MPI_Comm_Split than C) should correspond to command with mod = 8

13. A 3D domain was written to a file in XYZ order. P processes read the file and calculate partial sum of each subdomain in parallel. The domain decomposition is 1D along the Z-axis. The C MPI code is shown below. Assume that every process reads data[start] to data[end-1] from the file in line 18. (a) Fill in the blanks in the code (lines 16 and 17). Do not add new code. (4 marks)




```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include "mpi.h"
4
5 int main (int argc, char *argv[])
6 {
7   int rank, P, sum = 0, X, Y, Z, N;
8   MPI_Init (&argc, &argv);
9   MPI_Comm_rank (MPI_COMM_WORLD, &rank);
10  MPI_Comm_size (MPI_COMM_WORLD, &P);
11  X = atoi(argv[1]);
12  Y = atoi(argv[2]);
13  Z = atoi(argv[3]); // Assume Z is divisible by P
14  N = X*Y*Z;
15  int data[N];
16  int start = X * y * Z/p * rank ;
17  int end= X * y * Z/p * (rank+1) ;
18  //Assume code for file read
19  for (int i=start; i<end; i++)
20    sum += data[i];
21  if (!rank) printf ("%d: %d %d\n", rank, start, end);
22  MPI_Finalize();
23  return 0;
24 }
```

(b) Write the output of the above code for the below runs. (2+2 marks)

mpirun –np 8 ./a.out 4 4 8

0 : 0 16

mpirun –np 8 ./a.out 4 8 16

0 : 0 64

14.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include "mpi.h"
 4
 5 int main( int argc, char *argv[])
 6 {
 7   int myrank, size;
 8   double start_time, time, max_time;
 9   MPI_Init(&argc, &argv);
10   MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
11   MPI_Comm_size(MPI_COMM_WORLD, &size);
12   MPI_Status status[size-1];
13   MPI_Request request[size-1];
14   int BUFSIZE = atoi(argv[1]);
15   int arr[BUFSIZE];
16   if (myrank < size-1)
17    MPI_Send(arr, BUFSIZE, MPI_INT, size-1, myrank, MPI_COMM_WORLD);
18   else
19   {
20    int count, recvarr[size][BUFSIZE];
21    for (int i=0; i<size-1; i++)
22     MPI_Irecv(recvarr[i], BUFSIZE, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
&request[i]);
23   }
24   MPI_Waitall (size-1, request, status);
25   MPI_Finalize();
26   return 0;
27 }
```

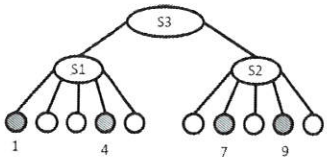
Will the above code run for **mpirun –np 4 ./a.out 100**? If not, what is the error? What is the fix and in which line number(s)? (4 marks)

NO, there's ~~one an~~ errors in the code:

① Place the code at line 24 inside the 'else' statement V so that only rank = 3 process will call waitall. (after the for loop)

② ~~change~~ ~~request~~

15. An user would like to use 8 processes to execute her molecular dynamics simulation code. The XYZ process decomposition (virtual topology) for the 3D simulation domain (NxNxN) is 2x2x2. A process communicates with its 6 nearest neighbors (maximum) in 3 dimensions (X, Y, Z) using MPI_Isend() to exchange atom attributes in the halo regions every time step (NN communication). Every process communicates 128 bytes to each of its neighbors in a time step. The 8 processes execute on node numbers 1, 4, 7, 9 of the system with tree network as shown. Assume default process placement policy. Two ranks are placed on each allocated node, i.e. ranks 0 and 1 on node 1, etc. The number of hops for intra-switch communication is 2, whereas it is 4 for inter-switch communication. Assume that no other communication-intensive job is running. (4+5 marks)



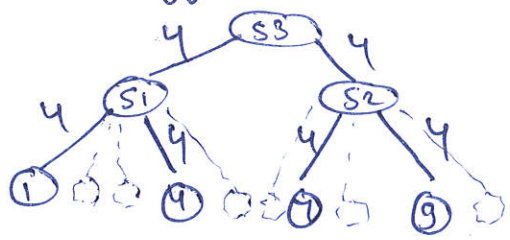


(a) Derive the maximum hop-bytes in a communication step.

(b) List and explain all potential interferences and network congestion during the NN communications.

(a) Communication between process rank 0 and 4 leads to the maximum hop-bytes because nodes 1 and 7 are the farthest away nodes in this tree and rank=0 is on node 1 and rank=4 is on node 7.
# hops b/w node 1 & 7 = 4 (∵ inter switch comm).
⇒ Hop-bytes = 4 × 128 = 512 bytes

(b) Let's see how many times each link in this tree topology is used in a singe comm. step:




comms = [0-1, 0-2, 0-4, 1-3, 1-5, 2-3, 2-6, 4-5, 4-6, 6-7]

As you can see, all of the links are used 4 times per time step, hence there is no network congestion.

## 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_Gatherv**(const void *sendbuf, int sendcount, MPI_Datatype sendtype,void *recvbuf, const int *recvcounts, const int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
18) int **MPI_Scatterv**(const void *sendbuf, const int *sendcounts, const int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount,MPI_Datatype recvtype, int root, MPI_Comm comm)
19) int **MPI_Allgatherv**(const void *sendbuf, int sendcount, MPI_Datatype sendtype,void *recvbuf, const int *recvcounts, const int *displs,MPI_Datatype recvtype, MPI_Comm comm)
20) int **MPI_Alltoallv**(const void *sendbuf, const int *sendcounts, const int *sdispls, MPI_Datatype sendtype, void *recvbuf, const int *recvcounts, const int *rdispls, MPI_Datatype recvtype, MPI_Comm comm)
21) int **MPI_Isend**(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)
22) int **MPI_Irecv**(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)
23) int **MPI_Ibcast**(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm, MPI_Request *request)
24) int **MPI_Igather**(const void *sendbuf, int sendcount, MPI_Datatype sendtype,void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm, MPI_Request *request)
25) int **MPI_Iscatter**(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm, MPI_Request *request)
26) int **MPI_Iallgather**(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm, MPI_Request *request)
27) int **MPI_Ireduce**(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm, MPI_Request *request)
28) int **MPI_Iallreduce**(const void *sendbuf, void *recvbuf, int count,MPI_Datatype datatype, MPI_Op op, MPI_Comm comm, MPI_Request *request)
29) int **MPI_Ialltoall**(const void *sendbuf, int sendcount, MPI_Datatype sendtype,void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm, MPI_Request *request)
30) int **MPI_Igatherv**(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, const int recvcounts[], const int displs[], MPI_Datatype recvtype, int root, MPI_Comm comm, MPI_Request *request)
31) int **MPI_Iscatterv**(const void *sendbuf, const int sendcounts[], const int displs[],MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm, MPI_Request *request)
32) int **MPI_Iallgatherv**(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, const int recvcounts[], const int displs[], MPI_Datatype recvtype,MPI_Comm comm, MPI_Request *request)
33) int **MPI_Ialltoallv**(const void *sendbuf, const int sendcounts[], const int sdispls[], MPI_Datatype sendtype, void *recvbuf, const int recvcounts[], const int rdispls[], PI_Datatype recvtype, MPI_Comm comm, MPI_Request *request)
34) int **MPI_Type_vector**(int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype * newtype)
35) int **MPI_Type_indexed**(int count, const int *array_of_blocklengths, const int *array_of_displacements, MPI_Datatype oldtype, MPI_Datatype *newtype)
36) int **MPI_File_read_at**(MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Status * status)
37) int **MPI_File_read_all**(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status) Input Parameters
38) int **MPI_File_open**(MPI_Comm comm, ROMIO_CONST char *filename, int amode, MPI_Info info, MPI_File * fh)
39) int **MPI_File_set_view**(MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype filetype, ROMIO_CONST char *datarep, MPI_Info info)
40) int **MPI_File_seek**(MPI_File fh, MPI_Offset offset, int whence)