National University of Computer and Emerging Sciences

FAST School of Computing **Spring-2022** **Islamabad Campus**

CS-3006: Parallel and Distributed Computing

Serial No:
Sessional Exam-I
Total Time: 1 Hour
Total Marks: 50

Saturday, 12th March, 2022

Course Instructors

Muhammad Aleem, Muhammad Arshad Islam, Shujaat Hussain, Qaiser Shafi, Humera Sabir

_____
Signature of Invigilator

_____ _____
Student Name Roll No. Section
Signature

DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.

Instructions:

1. Attempt on question paper. Attempt all of them. Read the question carefully, understand the question, and then attempt it.
2. No additional sheet will be provided for rough work. Use the back of the last page for rough work.
3. If you need more space write on the back side of the paper and clearly mark question and part number etc.
4. After asked to commence the exam, please verify that you have **three? (3?)** different printed pages including this title page. There are a total of **5?** questions.
5. Calculator sharing is strictly prohibited.
6. Use permanent ink pens only. Any part done using soft pencil will not be marked and cannot be claimed for rechecking.

	Q-1	**Q-2**	**Q-3**	**Q-4**	**Total**
Marks Obtained					
Total Marks	**12**	**20**	**08**	**10**	**50**

Question-1 [12 Marks]

Please completely fill the box ☐ for the correct answer like this ☒ and not like this ☒ or this ☒. Cutting, overwriting, multiple answers, or not correctly filling the box would be considered incorrect. There is no negative marking.

NOTE: Any MCQ option not marked here (even if it is marked in the next pages) will not be checked.

	A	B	C	D	E
01	☐	☐	☐	☐	☐
02	☐	☐	☐	☐	☐
03	☐	☐	☐	☐	☐
04	☐	☐	☐	☐	☐
05	☐	☐	☐	☐	☐
06	☐	☐	☐	☐	☐
07	☐	☐	☐	☐	☐
08	☐	☐	☐	☐	☐
09	☐	☐	☐	☐	☐
10	☐	☐	☐	☐	☐
11	☐	☐	☐	☐	☐
12	☐	☐	☐	☐	☐

1. Which of the following parallel architecture is most suited for the data-parallel applications:

A. Gird
B. Cluster
C. SIMD
D. Cloud
E. MISD

2. Which type of the parallel application has better potential for load balancing:

A. Coarse-grained
B. Fine-grained

C. Task parallel
D. Applications with triangular dependencies

3. Which benchmark is officially used to rank the top supercomputers?
   A. NAS Parallel benchmarks
   B. SPEC2000 benchmarks
   C. LINPACK benchmark
   D. SPEChpc benchmarks
   E. All of the above

4. Which is the most used platform for top500 super-computers?
   A. IaaS
   B. PaaS
   C. Linux
   D. MS AZURE platform
   E. Custom or proprietary platforms

5. In MPI_RECV call which of the followings MUST be provided?
   A. Rank of the sender
   B. Tag of the message
   C. Type of the data
   D. All of the above
   E. Status structure

6. Which parallel computing architecture provides a better quality of service with respect to the *availability* aspect?
   A. Cluster
   B. Grid
   C. Cloud
   D. Internet-of-Things
   E. GPU Cluster

7. Cyclic domain decomposition is more suitable if the cluster is
   A. Homogeneous
   B. Heterogeneous
   C. Not suitable for cluster-based computation
   D. Data-size is too small
   E. None of the above

8. MIMD can be used as
   A. SIMD and SISD
   B. SISD and MISD
   C. MISD and SIMD
   D. All of above
   E. None of above

9. Which of the following programming environments support SPMD computing model:
   A. PThreads
   B. MPI

C. OpenMP
D. None of the above

10. Which of the following is not an implementation of MPI specification:
A. MPICH
B. LAM
C. OpenMPI
D. OpenMP
E. B and D

11. SIMD represents an organization that:
A. Includes many processing units under the supervision of a common control unit
B. vector supercomputer and MIMD systems
C. a pipeline based processor organization
D. All of the above

12. A processor performing fetch or decoding of different instruction during the execution of another instruction is called:
A. SIMD processor
B. Cache-coherent processor
C. Pipeline based processor
D. Uniform Memory Access processor
E. None of the above

Question-2 [20 Marks]

Consider the following MPI codes and determine the output (results shown on the screen when executed):

```
// CODE-1
// NOTE: This MPI program will be executed using 4 Processes

      int main(int argc, char* argv[])
      {
         int rank, size, tag = 89, val, i, root=0, other;
         int A[]={4,2,5,3};

         MPI_Init(&argc,&argv);
         MPI_Comm_size(MPI_COMM_WORLD,&size);
         MPI_Comm_rank(MPI_COMM_WORLD,&rank);
         val = A[rank];
         int FV = val;
         if (rank == root)
         {
            for (i = 0; i < size; ++i)
            {
                 if (i!=root)
               {
                  MPI_Recv(&other,1,MPI_INT,i,tag, MPI_COMM_WORLD,
      MPI_STATUS_IGNORE);
                  FV *= other;
               }
            }
            printf("\nFinal value: %d\n",FV);
         }
         else
            MPI_Send(&val, 1, MPI_INT, root, tag, MPI_COMM_WORLD);

         MPI_Finalize();

        return 0;

} //end of main
```

Output:

120

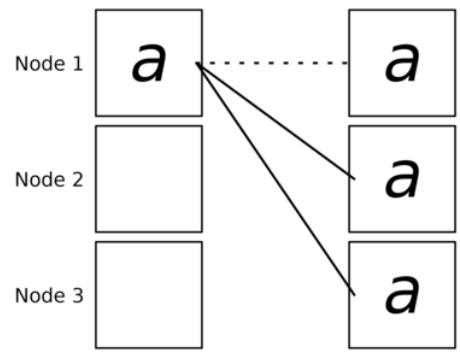
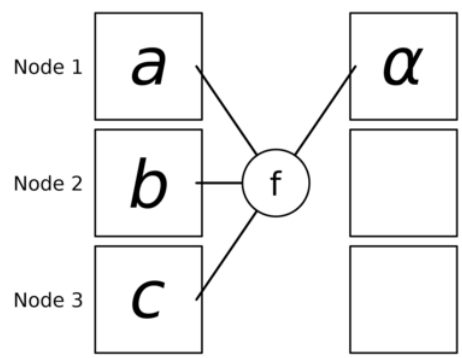
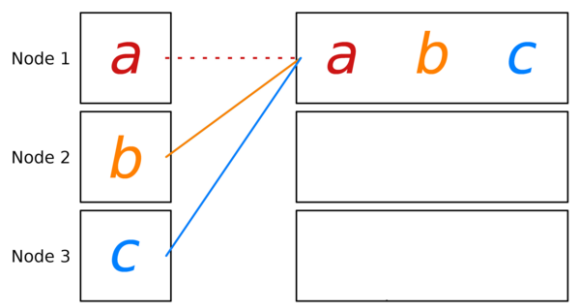
```
// CODE-1
// NOTE: This MPI program will be executed using 2 Processes

#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include<string.h>

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

    int numtasks, rank, dest, source, count,namelen;
    char data[15];
    char processorName[10];
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(processorName, &namelen);

    if (rank == 0) {

strcpy(data, "Hello SectionA");
    MPI_Send(data, 15, MPI_CHAR, 1, 1, MPI_COMM_WORLD);
strcpy(data, "Hello SectionB");
MPI_Send(data, 15, MPI_CHAR, 1, 2, MPI_COMM_WORLD);
strcpy(data, "Hello SectionC");
MPI_Send(data, 15, MPI_CHAR, 1, 3, MPI_COMM_WORLD);
strcpy(data, "Hello SectionD");
MPI_Send(data, 15, MPI_CHAR, 1, 4, MPI_COMM_WORLD);

    } else {
        MPI_Recv(data, 15, MPI_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("processor %s having rank %d of %d received message: %s\n",
processorName, rank, numtasks, data);
        MPI_Recv(data, 15, MPI_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("processor %s having rank %d of %d received message: %s\n",
processorName, rank, numtasks, data);


    }
    MPI_Finalize();


        return 0;

} //end of main
```

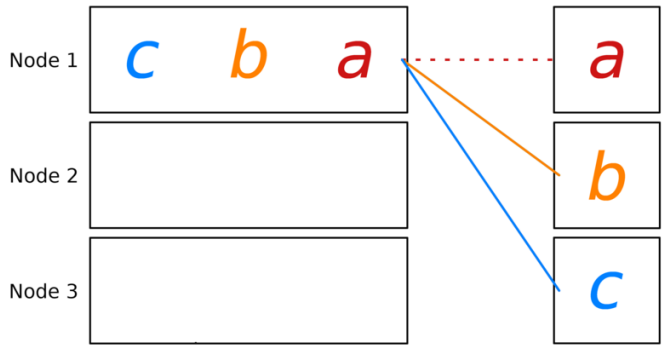
Output:

```
processor ez-VirtualBox having rank 1 of 2 received message: Hello SectionA
processor ez-VirtualBox having rank 1 of 2 received message: Hello SectionB
```

Question-3 [08 Marks]

Consider the following figures depicting communication patterns, name each of these.

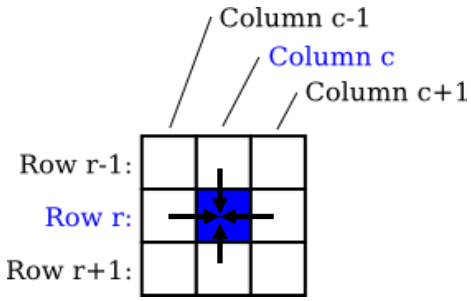
Name of Communication Operation	Communication Patterns (Senders → Receivers)
Broadcast	
Reduction	
Gather	

Scatter	

Question-4 [20 Marks]

Consider a scientific application that takes as input a square 2D array (of size 4096 by 4096) to produce the resultant matrix using the neighbor-point computation mechanism. As shown in the below diagram, each array element will be computed using 5 values:

```
A[i][j] = A[i][j] + (A[i-1][j] * A[i][j-1] / A[i+1][j] * A[i][j+1])
```



Considering the above application and its penalization, answer the following questions with Justifications <u>Do not write outside the box it will not be marked:</u>

Is this problem able to be parallelized? (Yes/No)	Yes
Justification (2 Marks): After communicating neighboring points by each process, All executions can be 100% concurrent	

Name the best suited partitioning scheme?	Row-wise block or simply Block-
Justification (2 Marks): Row or column wise distribution can be done, Equal parts division possible	

Are communications needed?	Answer: Yes
Justification (2 Marks): Adjacent rows or columns element will be communicated between two adjacent processes	

Are synchronizations will be required	Answer: Yes
Justification (2 Marks): Among adjacent processes	

Will load balancing be a concern?	Answer: NO
Justification (2 Marks): Block wise distribution contains same amount of work, so no imbalance related issues.	

MPI Syntax Sheet

int MPI_Init(int *argc, char ***argv)	int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Finalize( )	int MPI_Comm_rank(MPI_Comm comm, int *rank)
MPI_Send(void* data, int count, MPI_Datatype type, int dest, int tag, MPI_Comm comm)	MPI_Recv(void* data,int count,MPI_Datatype type, int source,int tag,MPI_Comm comm,MPI_Status* status
int MPI_Get_count(MPI_Status* status, MPI_Datatype type, int* count)	int MPI_Probe(int source, int tag, MPI_Comm comm, MPI_Status *status)