Problem # 1

To calculate the sum of given numbers in parallel:
The following program calculates the sum of numbers from 1 to 1000 in a parallel fashion while executing on all the cluster nodes and providing the result at the end on only one node. It should be noted that the print statement for the sum is only executed on the node that is ranked zero (0) otherwise the statement would be printed as much time as the number of nodes in the cluster.

```
#include<iostream.h>
#include<mpi.h>

int main(int argc, char ** argv)
{ int mynode, totalnodes; int sum,startval,endval,accum;
MPI_Status status;
MPI_Init(argc,argv);

MPI_Comm_size(MPI_COMM_WORLD, &totalnodes);
MPI_Comm_rank(MPI_COMM_WORLD, &mynode);
sum = 0;
startval = 1000*mynode/totalnodes+1;
endval = 1000*(mynode+1)/totalnodes;
for(int i=startval;i<=endval;i=i+1)
     sum = sum + i;
     if(mynode!=0)
           MPI_Send(&sum,1,MPI_INT,0,1,MPI_COMM_WORLD);
else
           for(int j=1;j<totalnodes;j=j+1)
           {
               MPI_Recv(&accum,1,MPI_INT,j,1,MPI_COMM_WOR
          LD,&status);
           sum = sum + accum;
           }
     if(mynode == 0) cout << "The sum from 1 to 1000 is:
" <<  sum <<   endl;
MPI_Finalize();
}
```

1. Code the above example program in C that calculates the sum of numbers in parallel on different numbers of nodes. Also calculate the execution time.
[Note: You have to use time stamp function to also print the time at begging and end of parallel code segment]

Output: (On Single Node)

Execution Time:

Output: (On Two Nodes)

Execution Time:

Speedup:

Output: (On Four Nodes)

Execution Time:

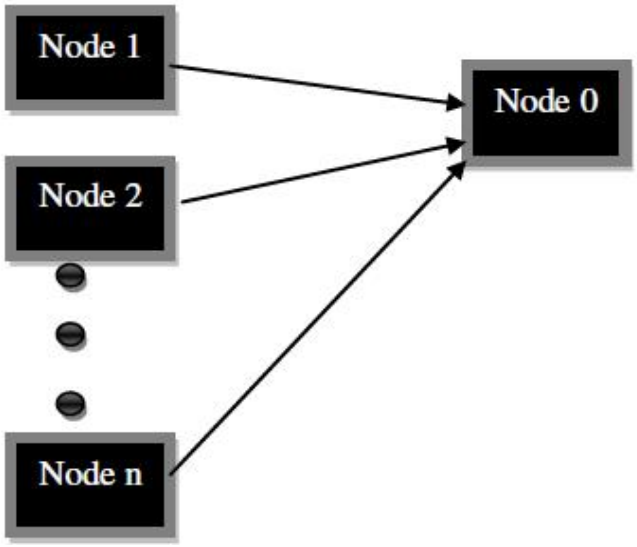
Speedup:

Output: (On Sixteen Nodes)

Execution Time:

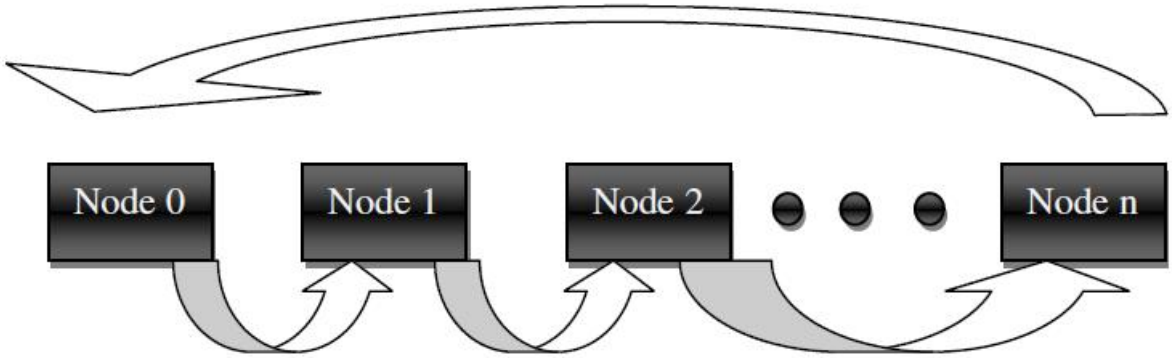
Speedup:

2. Suppose you are in a scenario where you have to transmit an array buffer from all other nodes to one node by using send/ receive functions that are used for intra-process synchronous communication. The figure below demonstrates the required functionality of the program.



Problem # 2

1. Write a program in which every node receives from its left node and sends message to its right node simultaneously as depicted in the following figure



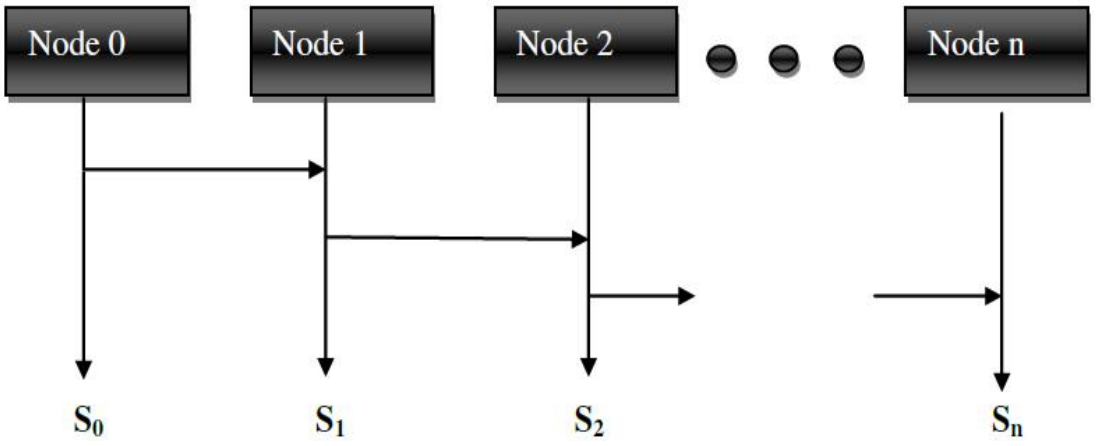
2. Write a program to calculate prefix sum S_n of 'n' numbers on 'n' processes

- Each node has two variables 'a' and 'b'.
- Initially a=node id
- Each node sends 'a' to the other node.
- 'b' is a variable that receives a sent from another node.

Problem # 3

Write a program to calculate prefix sum of ‘n’ numbers on ‘n’ processes. Use MPI_Scan to address this problem.

The above problem can be best stated with the help of following figure.



Problem # 4

Write a MPI parallel program that calculates the value of PI using integral method.

Algorithm: The algorithm suggested here is chosen for its simplicity. The method evaluates the integral of 4/(1+x*x) between -1/2 and 1/2. The method is simple: the integral is approximated by a sum of n intervals; the approximation to the integral in each interval is (1/n)*4/(1+x*x). The master process (rank 0) asks the user for the number of intervals; the master should then broadcast this number to all of the other processes. Each process then adds up every n'th interval (x = -1/2+rank/n, -1/2+rank/n+size/n,). Finally, the sums computed by each process are added together using a reduction.

Problem # 5

Consider the scenario of distributed memory as given in below requirements:

Requirement # 1: The processor 0 distributes the factorial computing to all the nodes (total number of nodes are 5).

Requirement # 2: The communication between processor 0 and 1 is blocking.

Requirement # 3: Each node computes the factorial and then returns it to processor 0.

Requirement # 4: The computed result is printed first at processor 0 and then each node prints their respective results.

a. Develop parallel version of factorial calculation program for distributed memory using MPI. The sequential version is given below to start with.
b. Also, comment on how you will develop the MPI version?

```c
#include <stdio.h>
void main(){
  int i,f=1,num;

  printf("Input the number : ");
  scanf("%d",&num);

  for(i=1;i<=num;i++)
      f=f*i;

  printf("The Factorial of %d is: %d\n",num,f);
}
```

Problem # 6

You are given two square ***integer*** matrices, A and B, of size N x N. Your task is to implement matrix multiplication using MPI. You will use MPI_Scatter and MPI_Reduce operations to distribute the workload among multiple processes and combine the results.

Task Steps:

1. Matrix Generation:
    - Generate two random square matrices, A and B, of size N x N.
2. Sequential Matrix Multiplication:
    - Implement a sequential matrix multiplication algorithm in Python without MPI.
    - Perform matrix multiplication of matrices A and B using the sequential algorithm.
    - Print the resulting matrix C, which is the product of A and B.
3. MPI Scatter:
    - Initialize MPI and get the total number of processes and the current process rank.
    - If the current process rank is 0:
        - Scatter matrix A to all other processes using MPI_Scatter.
    - If the current process rank is not 0:
        - Receive the scattered portion of matrix A using MPI_Scatter.
    - Print the received portion of matrix A on each process.
    - Scatter matrix B to all processes using MPI_Scatter.
    - Print the received portion of matrix B on each process.
4. Matrix Multiplication:
    - Perform the local matrix multiplication of the scattered portions of matrices A and B on each process.
    - Implement the matrix multiplication algorithm for the local portions of A and B.
    - Print the resulting local matrix on each process.
5. MPI Reduce:
    - Use MPI_Reduce to collect all the local matrix portions from different processes.
    - Reduce the local matrix portions using MPI_Reduce with the MPI_SUM operation.
    - On process rank 0, print the resulting matrix C, which is the product of A and B.