Channabasaveshwara Institute of Technology
(Affiliated to VTU, Belgavi & Approved by AICTE, New Delhi)
(**NAAC Accredited & ISO 9001:2015 Certified Institution)**
NH 216 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Information Science & Engineering

LAB MANUAL (2025–26)

PARALLEL COMPUTING (BCS702)

Name :

Usn:

Experiments

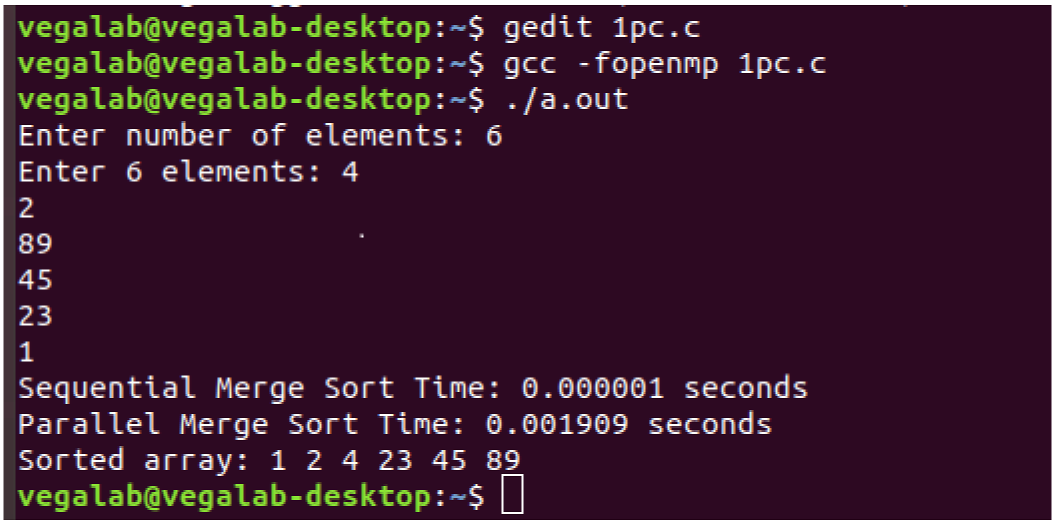
1. Write a OpenMP program to sort an array on n elements using both sequential and parallel mergesort(using Section). Record the difference in execution time.
2. Write an OpenMP program that divides the Iterations into chunks containing 2 iterations, respectively (OMP_SCHEDULE=static,2). Its input should be the number of iterations, and its output should be which iterations of a parallelized for loop are executed by which thread.
   For example, if there are two threads and four iterations, the output might be the following:
   a. Thread 0 : Iterations 0 −− 1
   b. Thread 1 : Iterations 2 −− 3
3. Write a OpenMP program to calculate n Fibonacci numbers using tasks.
4. Write a OpenMP program to find the prime numbers from 1 to n employing parallel for directive. Record both serial and parallel execution times.
5. Write a MPI Program to demonstration of MPI_Send and MPI_Recv.
6. Write a MPI program to demonstration of deadlock using point to point communication and avoidance of deadlock by altering the call sequence
7. Write a MPI Program to demonstration of Broadcast operation.
8. Write a MPI Program demonstration of MPI_Scatter and MPI_Gather
9. Write a MPI Program to demonstration of MPI_Reduce and MPI_Allreduce (MPI_MAX,MPI_MIN, MPI_SUM, MPI_PROD)

INDEX

1.Write a OpenMP program to sort an array on n elements using both sequential and parallel mergesort (using Section). Record the difference in execution time.

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
void merge(int arr[ ], int left, int mid, int right)
{
        int n1 = mid - left + 1;
        int n2 = right - mid;
        int L[n1], R[n2];
        for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
        for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2)
        {
                if (L[i] <= R[j])
                arr[k++] = L[i++];
                else
                arr[k++] = R[j++];

        }
        while (i < n1)
                arr[k++] = L[i++];
        while (j < n2)
                arr[k++] = R[j++];
}
// Sequential Merge Sort
void mergeSortSequential(int arr[ ], int left, int right)
{
        if (left < right)
        {
                int mid = left + (right - left) / 2;
                mergeSortSequential(arr, left, mid);
                mergeSortSequential(arr, mid + 1, right);
                merge(arr, left, mid, right);
        }
}
// Parallel Merge Sort
void mergeSortParallel(int arr[ ], int left, int right)
{
        if (left < right)
        {
                int mid = left + (right - left) / 2;
                #pragma omp parallel sections
                {
                #pragma omp section
                mergeSortParallel(arr, left, mid);
                #pragma omp section
                mergeSortParallel(arr, mid + 1, right);
                }
                merge(arr, left, mid, right);
        }
}
int main( )
{
        int n;
        printf("Enter number of elements: ");
        scanf("%d", &n);
        int *arr1 = (int *)malloc(n * sizeof(int));
        int *arr2 = (int *)malloc(n * sizeof(int));
        printf("Enter %d elements: ", n);
        for (int i = 0; i < n; i++)
        {
                scanf("%d", &arr1[i]);
                arr2[i] = arr1[i]; // Copy for parallel sorting
        }
        double start, end;

        // Sequential Sort
        start = omp_get_wtime( );
        mergeSortSequential(arr1, 0, n - 1);
        end = omp_get_wtime( );
        printf("Sequential Merge Sort Time: %f seconds\n",end-start);

        // Parallel Sort
        start = omp_get_wtime( );
        mergeSortParallel(arr2, 0, n - 1);
        end = omp_get_wtime( );

        printf("Parallel Merge Sort Time: %f seconds\n",end-start);
        printf("Sorted array: ");
        for (int i = 0; i < n; i++)
        printf("%d ", arr1[i]);
        printf("\n");
        free(arr1);
        free(arr2);
        return 0;
}
```

OUTPUT:

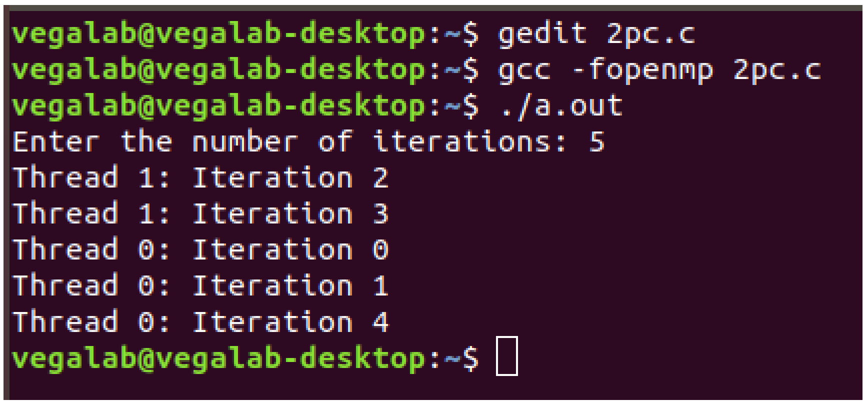


```
vegalab@vegalab-desktop:~$ gedit 1pc.c
vegalab@vegalab-desktop:~$ gcc -fopenmp 1pc.c
vegalab@vegalab-desktop:~$ ./a.out
Enter number of elements: 6
Enter 6 elements: 4
2
89
45
23
1
Sequential Merge Sort Time: 0.000001 seconds
Parallel Merge Sort Time: 0.001909 seconds
Sorted array: 1 2 4 23 45 89
vegalab@vegalab-desktop:~$
```

2.Write an OpenMP program that divides the Iterations into chunks containing 2 iterations, respectively (OMP_SCHEDULE=static,2). Its input should be the number of iterations, and its output should be which iterations of a parallelized for loop are executed by which thread. For example, if there are two threads and four iterations, the output might be the following:
a. Thread 0: Iterations 0 – 1
b. Thread 1: Iterations 2 – 3

```
#include <stdio.h>
#include <omp.h>
int main( )
{
        int num_iterations;
        printf("Enter the number of iterations: ");
        scanf("%d", &num_iterations);
        #pragma omp parallel
        {
                #pragma omp for schedule(static,2)
                for (int i = 0; i < num_iterations; i++)
                {
                        printf("Thread %d: Iteration %d\n", omp_get_thread_num(), i);
                }
        }
        return 0;
}
```

OUTPUT:

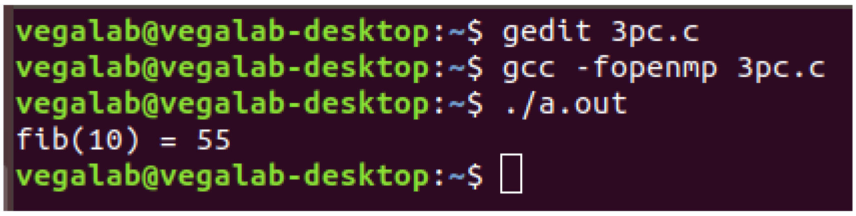


```
vegalab@vegalab-desktop:~$ gedit 2pc.c
vegalab@vegalab-desktop:~$ gcc -fopenmp 2pc.c
vegalab@vegalab-desktop:~$ ./a.out
Enter the number of iterations: 5
Thread 1: Iteration 2
Thread 1: Iteration 3
Thread 0: Iteration 0
Thread 0: Iteration 1
Thread 0: Iteration 4
vegalab@vegalab-desktop:~$ 
```

3.Write a OpenMP program to calculate n Fibonacci numbers using tasks.

```
#include <stdio.h>
#include <omp.h>
int fib(int n)
{
        int i, j;
        if (n<2)
                return n;
        else
        {
                #pragma omp task shared(i) firstprivate(n)
                i=fib(n-1);
                #pragma omp task shared(j) firstprivate(n)
                j=fib(n-2);
                #pragma omp taskwait
                return i+j;
        }
}
int main( )
{
        int n = 10;
        omp_set_dynamic(0);
        omp_set_num_threads(4);

        #pragma omp parallel shared(n)
        {
        #pragma omp single
        printf ("fib(%d) = %d\n", n, fib(n));
        }
}
```

OUTPUT:

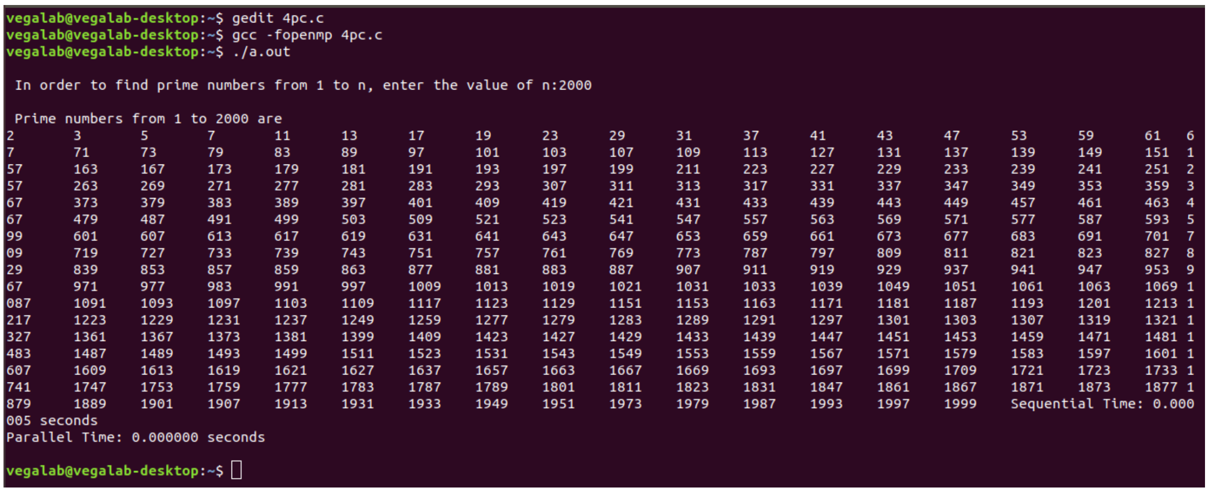


```
vegalab@vegalab-desktop:~$ gedit 3pc.c
vegalab@vegalab-desktop:~$ gcc -fopenmp 3pc.c
vegalab@vegalab-desktop:~$ ./a.out
fib(10) = 55
vegalab@vegalab-desktop:~$ 
```

4.Write a OpenMP program to find the prime numbers from 1 to n employing parallel for directive. Record both serial and parallel execution times.

```
#include <stdio.h>
#include <omp.h>
int main( )
{
        long int prime[1000], i, j, n;
        // Prompt user for input
        printf("\n In order to find prime numbers from 1 to n, enter the value of n:");
        scanf("%ld", &n);
        // Initialize all numbers as prime (set all to 1)
        for(i = 1; i <= n; i++)
        {
                prime[i] = 1;
        }
        // 1 is not a prime number
        prime[1] = 0;
        // Sieve of Eratosthenes with parallelization
        for(i = 2; i * i <= n; i++)
        {
                #pragma omp parallel for
                for(j = i * i; j <= n; j = j + i)
                {
                        if(prime[j] == 1)
                        {
                                prime[j] = 0;
                        }
                }
        }
        // Print prime numbers
        printf("\n Prime numbers from 1 to %ld are\n", n);
        for(i = 2; i <= n; i++)
        {
                if(prime[i] == 1)
                {
                        printf("%ld\t", i);
                }
        }
        double start, end;
        // Sequential
        start = omp_get_wtime( );
        prime[i];
        end = omp_get_wtime( );
        printf("Sequential Time: %f seconds\n",end-start);
        // Parallel
        start = omp_get_wtime( );
        prime[i];
        end = omp_get_wtime( );
        printf("Parallel Time: %f seconds\n",end-start);
        printf("\n");
}
```

OUTPUT:

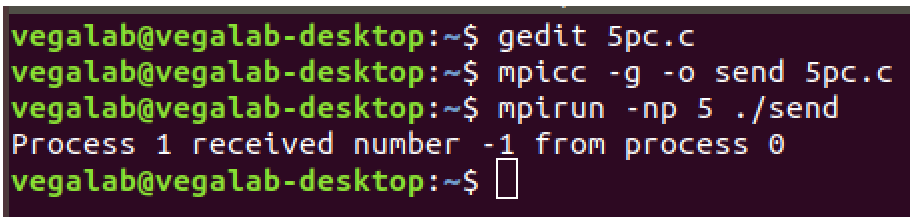


```
vegalab@vegalab-desktop:~$ gedit 4pc.c
vegalab@vegalab-desktop:~$ gcc -fopenmp 4pc.c
vegalab@vegalab-desktop:~$ ./a.out

 In order to find prime numbers from 1 to n, enter the value of n:2000

 Prime numbers from 1 to 2000 are
2       3       5       7       11      13      17      19      23      29      31      37      41      43      47      53      59      61   6
7       71      73      79      83      89      97      101     103     107     109     113     127     131     137     139     149     151  1
57      163     167     173     179     181     191     193     197     199     211     223     227     229     233     239     241     251  2
57      263     269     271     277     281     283     293     307     311     313     317     331     337     347     349     353     359  3
67      373     379     383     389     397     401     409     419     421     431     433     439     443     449     457     461     463  4
67      479     487     491     499     503     509     521     523     541     547     557     563     569     571     577     587     593  5
99      601     607     613     617     619     631     641     643     647     653     659     661     673     677     683     691     701  7
09      719     727     733     739     743     751     757     761     769     773     787     797     809     811     821     823     827  8
29      839     853     857     859     863     877     881     883     887     907     911     919     929     937     941     947     953  9
67      971     977     983     991     997     1009    1013    1019    1021    1031    1033    1039    1049    1051    1061    1063    1069 1
087     1091    1093    1097    1103    1109    1117    1123    1129    1151    1153    1163    1171    1181    1187    1193    1201    1213 1
217     1223    1229    1231    1237    1249    1259    1277    1279    1283    1289    1291    1297    1301    1303    1307    1319    1321 1
327     1361    1367    1373    1381    1399    1409    1423    1427    1429    1433    1439    1447    1451    1453    1459    1471    1481 1
483     1487    1489    1493    1499    1511    1523    1531    1543    1549    1553    1559    1567    1571    1579    1583    1597    1601 1
607     1609    1613    1619    1621    1627    1637    1657    1663    1667    1669    1693    1697    1699    1709    1721    1723    1733 1
741     1747    1753    1759    1777    1783    1787    1789    1801    1811    1823    1831    1847    1861    1867    1871    1873    1877 1
879     1889    1901    1907    1913    1931    1933    1949    1951    1973    1979    1987    1993    1997    1999    Sequential Time: 0.000
005 seconds
Parallel Time: 0.000000 seconds

vegalab@vegalab-desktop:~$ 
```

5.Write a MPI Program to demonstration of MPI_Send and MPI_Recv.

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv)
{
        // Initialize the MPI environment
        MPI_Init(NULL, NULL);

        // Find out rank, size
        int world_rank;
        MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

        int world_size;
        MPI_Comm_size(MPI_COMM_WORLD, &world_size);

        // We are assuming at least 2 processes for this task
        if (world_size < 2)
        {
                fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
                MPI_Abort(MPI_COMM_WORLD, 1);
        }
        int number;
        if (world_rank == 0)
        {
                // If we are rank 0, set the number to -1 and send it to process 1
                number = -1;
                MPI_Send(
                /* data = */ &number,
                /* count = */ 1,
                /* datatype = */ MPI_INT,
                /* destination = */ 1,
                /* tag = */ 0,
                /* communicator = */ MPI_COMM_WORLD);
        } else if (world_rank == 1)
        {
                MPI_Recv(
                /* data = */ &number,
                /* count = */ 1,
                /* datatype = */ MPI_INT,
                /* source = */ 0,
                /* tag = */ 0,
                /* communicator = */ MPI_COMM_WORLD,
                /* status = */ MPI_STATUS_IGNORE);
                printf("Process 1 received number %d from process 0\n", number);
        }
        MPI_Finalize( );
}
```

OUTPUT :



```
vegalab@vegalab-desktop:~$ gedit 5pc.c
vegalab@vegalab-desktop:~$ mpicc -g -o send 5pc.c
vegalab@vegalab-desktop:~$ mpirun -np 5 ./send
Process 1 received number -1 from process 0
vegalab@vegalab-desktop:~$
```

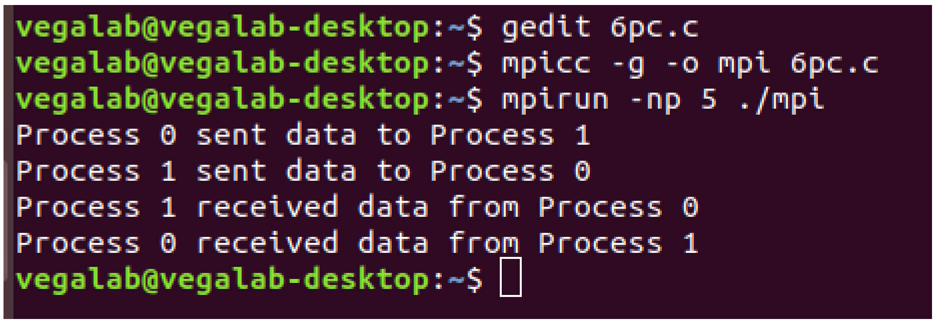
6.Write a MPI program to demonstration of deadlock using point to point communication and avoidance of deadlock by altering the call sequence.

Demonstration of deadlock

Code:

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv)
{
        int rank, size, data = 0;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        if (size < 2)
        {
                printf("This program requires at least 2 processes.\n");
                MPI_Abort(MPI_COMM_WORLD, 1);
        }
        if (rank == 0)
        {
        // Rank 0 sends first, then receives
                MPI_Send(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
                printf("Process 0 sent data to Process 1\n");
                MPI_Recv(&data,    1,    MPI_INT,    1,    0,    MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
                printf("Process 0 received data from Process 1\n");
        } else if (rank == 1)
        {
                // Rank 1 sends first, then receives
                MPI_Send(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
                printf("Process 1 sent data to Process 0\n");
                MPI_Recv(&data,    1,    MPI_INT,    0,    0,    MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
                printf("Process 1 received data from Process 0\n");
        }
        MPI_Finalize( );
        return 0;
}
```

OUTPUT:

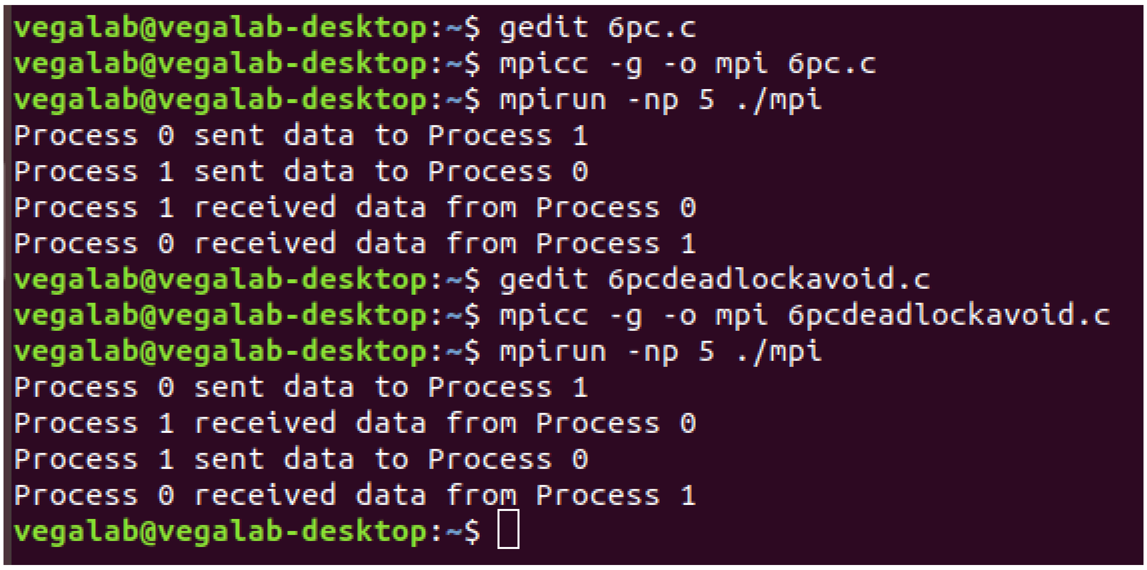


```
vegalab@vegalab-desktop:~$ gedit 6pc.c
vegalab@vegalab-desktop:~$ mpicc -g -o mpi 6pc.c
vegalab@vegalab-desktop:~$ mpirun -np 5 ./mpi
Process 0 sent data to Process 1
Process 1 sent data to Process 0
Process 1 received data from Process 0
Process 0 received data from Process 1
vegalab@vegalab-desktop:~$
```

Deadlock avoidance

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv)
{
        int rank, size, data = 0;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        if (size < 2)
        {
                printf("This program requires at least 2 processes.\n");
                MPI_Abort(MPI_COMM_WORLD, 1);
        }
        if (rank == 0)
        {
                // Rank 0 sends first
                MPI_Send(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
                printf("Process 0 sent data to Process 1\n");
                MPI_Recv(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
                printf("Process 0 received data from Process 1\n");
        }
        else if (rank == 1)
        {
                // Rank 1 receives first
                MPI_Recv(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
                MPI_STATUS_IGNORE);
                printf("Process 1 received data from Process 0\n");
                MPI_Send(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
                printf("Process 1 sent data to Process 0\n");
        }
        MPI_Finalize( );
        return 0;
}
```

OUTPUT:

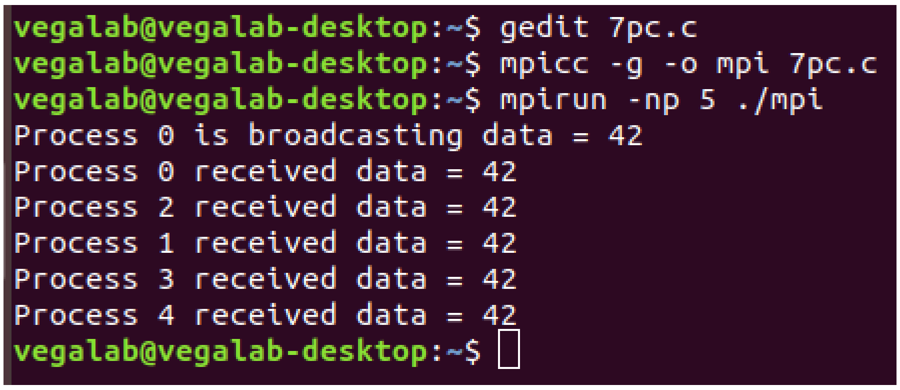


```
vegalab@vegalab-desktop:~$ gedit 6pc.c
vegalab@vegalab-desktop:~$ mpicc -g -o mpi 6pc.c
vegalab@vegalab-desktop:~$ mpirun -np 5 ./mpi
Process 0 sent data to Process 1
Process 1 sent data to Process 0
Process 1 received data from Process 0
Process 0 received data from Process 1
vegalab@vegalab-desktop:~$ gedit 6pcdeadlockavoid.c
vegalab@vegalab-desktop:~$ mpicc -g -o mpi 6pcdeadlockavoid.c
vegalab@vegalab-desktop:~$ mpirun -np 5 ./mpi
Process 0 sent data to Process 1
Process 1 received data from Process 0
Process 1 sent data to Process 0
Process 0 received data from Process 1
vegalab@vegalab-desktop:~$ 
```

7. Write a MPI Program to demonstration of Broadcast operation.

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv)
{
        int rank, size;
        int data; // The data to broadcast
        MPI_Init(&argc, &argv); // Initialize MPI
        MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Get process rank
        MPI_Comm_size(MPI_COMM_WORLD, &size); // Get total number of processes
        if (rank == 0)
        {
                data = 42; // Root process sets the data
                printf("Process %d is broadcasting data = %d\n", rank, data);
        }
        // Broadcast the data from process 0 to all other processes
        MPI_Bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
        // All processes print the received data
        printf("Process %d received data = %d\n", rank, data);
        MPI_Finalize( ); // Finalize MPI
        return 0;
}
```

OUTPUT:

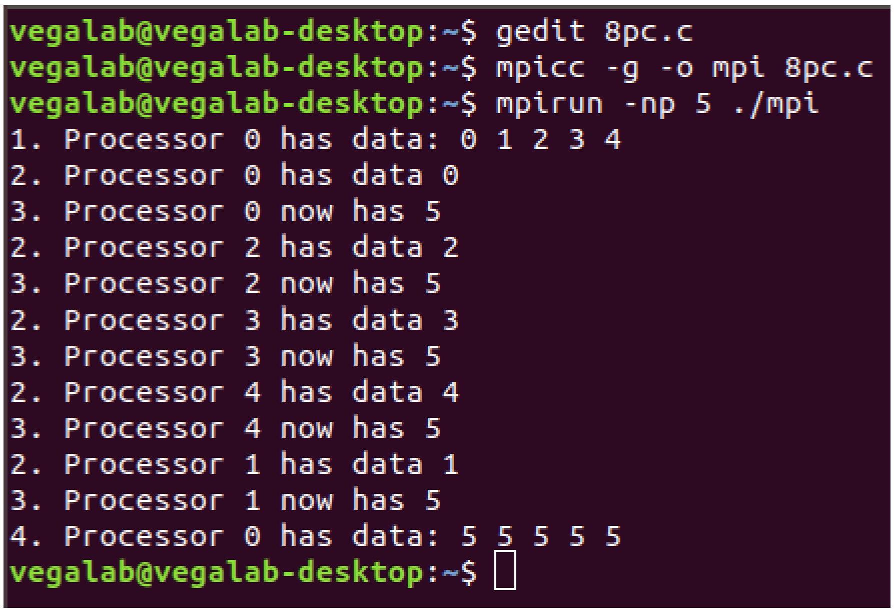


```
vegalab@vegalab-desktop:~$ gedit 7pc.c
vegalab@vegalab-desktop:~$ mpicc -g -o mpi 7pc.c
vegalab@vegalab-desktop:~$ mpirun -np 5 ./mpi
Process 0 is broadcasting data = 42
Process 0 received data = 42
Process 2 received data = 42
Process 1 received data = 42
Process 3 received data = 42
Process 4 received data = 42
vegalab@vegalab-desktop:~$
```

8. Write a MPI Program demonstration of MPI_Scatter and MPI_Gather

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv)
{
        int size, rank;
        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        int globaldata[4];/*wants to declare array this way*/
        int localdata;/*without using pointers*/
        int i;
        if (rank == 0)
        {
                for (i=0; i<size; i++)
                globaldata[i] = i;
                printf("1. Processor %d has data: ", rank);

                for (i=0; i<size; i++)
                printf("%d ", globaldata[i]);
                printf("\n");
        }
        MPI_Scatter(globaldata,1,MPI_INT,&localdata,1,MPI_INT,0, MPI_COMM_WORLD);
        printf("2. Processor %d has data %d\n", rank, localdata);
        localdata= 5;
        printf("3. Processor %d now has %d\n", rank, localdata);
        MPI_Gather(&localdata,1,MPI_INT,globaldata,1,MPI_INT, 0, MPI_COMM_WORLD);
        if (rank == 0)
        {
                printf("4. Processor %d has data: ", rank);
                for (i=0; i<size; i++)
                printf("%d ", globaldata[i]);
                printf("\n");
        }
        MPI_Finalize( );
        return 0;
}
```

OUTPUT:

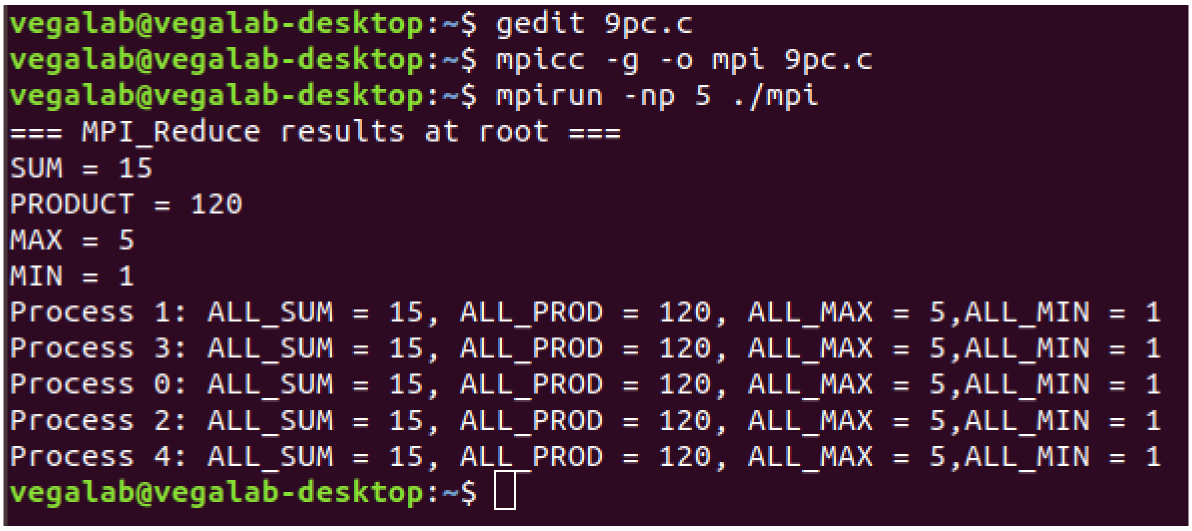


```
vegalab@vegalab-desktop:~$ gedit 8pc.c
vegalab@vegalab-desktop:~$ mpicc -g -o mpi 8pc.c
vegalab@vegalab-desktop:~$ mpirun -np 5 ./mpi
1. Processor 0 has data: 0 1 2 3 4
2. Processor 0 has data 0
3. Processor 0 now has 5
2. Processor 2 has data 2
3. Processor 2 now has 5
2. Processor 3 has data 3
3. Processor 3 now has 5
2. Processor 4 has data 4
3. Processor 4 now has 5
2. Processor 1 has data 1
3. Processor 1 now has 5
4. Processor 0 has data: 5 5 5 5 5
vegalab@vegalab-desktop:~$ 
```

9.Write a MPI Program to demonstration of MPI_Reduce and MPI_Allreduce (MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD)

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv)
{
        int rank, size;
        int value, sum, product, max, min;
        int all_sum, all_product, all_max, all_min;

        MPI_Init(&argc, &argv); // Initialize MPI
        MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Get rank
        MPI_Comm_size(MPI_COMM_WORLD, &size); // Get number of processes
        value = rank + 1; // Each process has a unique value

        // ---- MPI_Reduce ----
        MPI_Reduce(&value, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
        MPI_Reduce(&value, &product, 1, MPI_INT, MPI_PROD, 0, MPI_COMM_WORLD);
        MPI_Reduce(&value, &max, 1, MPI_INT, MPI_MAX, 0, MPI_COMM_WORLD);
        MPI_Reduce(&value, &min, 1, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);
        if (rank == 0)
        {
                printf("=== MPI_Reduce results at root ===\n");
                printf("SUM = %d\n", sum);
                printf("PRODUCT = %d\n", product);
                printf("MAX = %d\n", max);
                printf("MIN = %d\n", min);
        }

        // ---- MPI_Allreduce ----
        MPI_Allreduce(&value, &all_sum, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
        MPI_Allreduce(&value,&all_product,1,MPI_INT,MPI_PROD, MPI_COMM_WORLD);
        MPI_Allreduce(&value, &all_max, 1, MPI_INT, MPI_MAX, MPI_COMM_WORLD);
        MPI_Allreduce(&value, &all_min, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);

        printf("Process %d: ALL_SUM = %d, ALL_PROD = %d, ALL_MAX = %d, ALL_MIN =
        %d\n",rank, all_sum, all_product, all_max, all_min);
        MPI_Finalize( ); // Finalize MPI
        return 0;
}
```

OUTPUT:



```
vegalab@vegalab-desktop:~$ gedit 9pc.c
vegalab@vegalab-desktop:~$ mpicc -g -o mpi 9pc.c
vegalab@vegalab-desktop:~$ mpirun -np 5 ./mpi
=== MPI_Reduce results at root ===
SUM = 15
PRODUCT = 120
MAX = 5
MIN = 1
Process 1: ALL_SUM = 15, ALL_PROD = 120, ALL_MAX = 5,ALL_MIN = 1
Process 3: ALL_SUM = 15, ALL_PROD = 120, ALL_MAX = 5,ALL_MIN = 1
Process 0: ALL_SUM = 15, ALL_PROD = 120, ALL_MAX = 5,ALL_MIN = 1
Process 2: ALL_SUM = 15, ALL_PROD = 120, ALL_MAX = 5,ALL_MIN = 1
Process 4: ALL_SUM = 15, ALL_PROD = 120, ALL_MAX = 5,ALL_MIN = 1
vegalab@vegalab-desktop:~$ 
```