

Practical No 8

PRN: 22520005

Name: Aftab Imtiyaj Bhadgaonkar

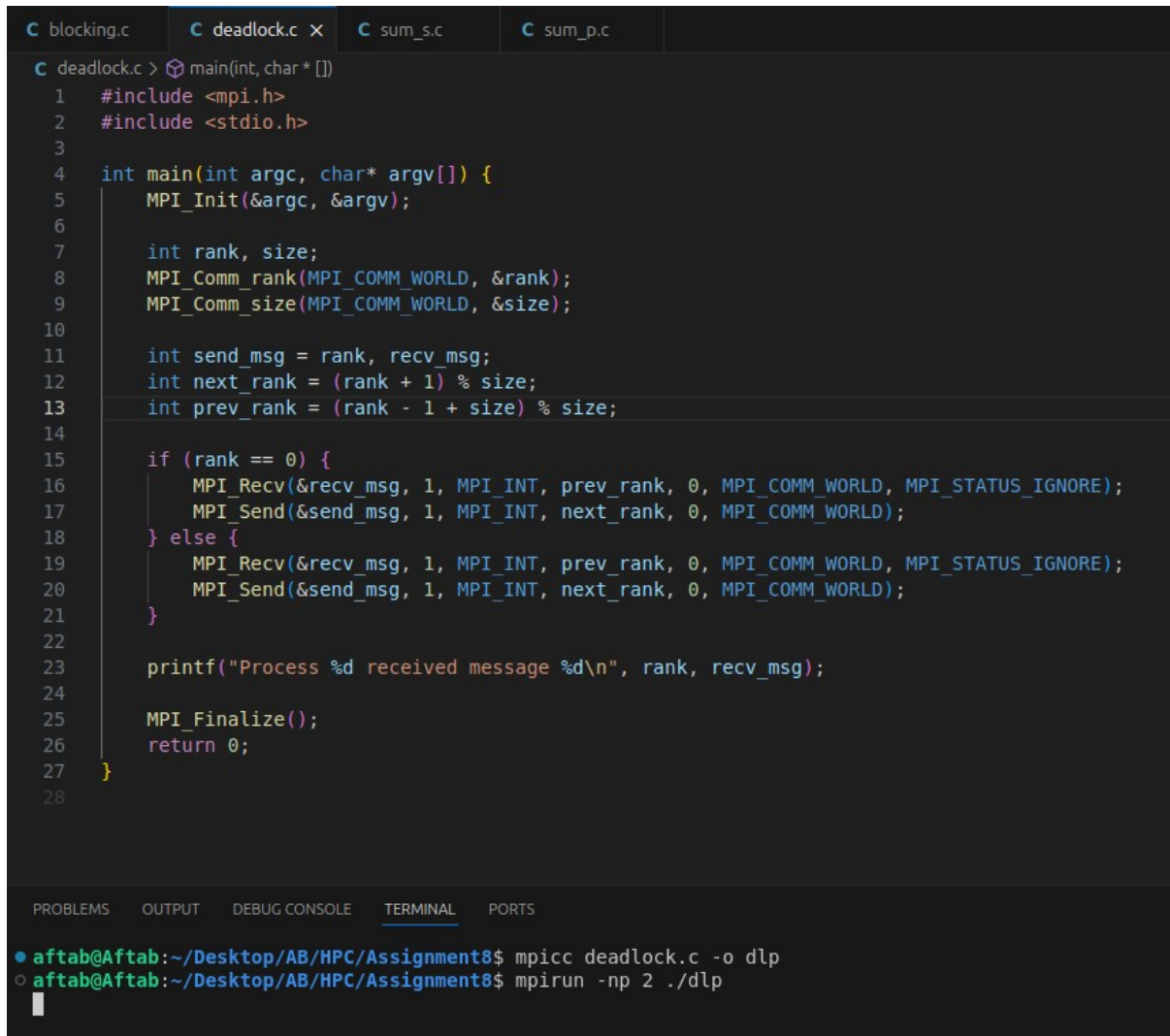
Batch: B6

Course: High Performance Computing Lab

Title of practical:

Q1: Implement a MPI program to give an example of Deadlock.

Screenshot:



```
C blocking.c C deadlock.c X C sum_s.c C sum_p.c
C deadlock.c > main(int, char * [])
1  #include <mpi.h>
2  #include <stdio.h>
3
4  int main(int argc, char* argv[]) {
5      MPI_Init(&argc, &argv);
6
7      int rank, size;
8      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
9      MPI_Comm_size(MPI_COMM_WORLD, &size);
10
11     int send_msg = rank, recv_msg;
12     int next_rank = (rank + 1) % size;
13     int prev_rank = (rank - 1 + size) % size;
14
15     if (rank == 0) {
16         MPI_Recv(&recv_msg, 1, MPI_INT, prev_rank, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
17         MPI_Send(&send_msg, 1, MPI_INT, next_rank, 0, MPI_COMM_WORLD);
18     } else {
19         MPI_Recv(&recv_msg, 1, MPI_INT, prev_rank, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
20         MPI_Send(&send_msg, 1, MPI_INT, next_rank, 0, MPI_COMM_WORLD);
21     }
22
23     printf("Process %d received message %d\n", rank, recv_msg);
24
25     MPI_Finalize();
26     return 0;
27 }
28

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpicc deadlock.c -o dlp
○ aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpirun -np 2 ./dlp
```

Q2. Implement blocking MPI send & receive to demonstrate Nearest neighbor exchange of data in a ring topology.

```
C blocking.c x C deadlock.c C sum_s.c C sum_p.c
C blocking.c > main(int, char *[])
1  #include <mpi.h>
2  #include <stdio.h>
3
4  int main(int argc, char* argv[]) {
5      MPI_Init(&argc, &argv);
6
7      int rank, size;
8      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
9      MPI_Comm_size(MPI_COMM_WORLD, &size);
10
11     int send_msg = rank;
12     int rcv_msg;
13
14     int next_rank = (rank + 1) % size;
15     int prev_rank = (rank - 1 + size) % size;
16
17     MPI_Send(&send_msg, 1, MPI_INT, next_rank, 0, MPI_COMM_WORLD);
18     MPI_Recv(&rcv_msg, 1, MPI_INT, prev_rank, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
19
20     printf("Process %d received message %d from process %d\n", rank, rcv_msg, prev_rank);
21
22     MPI_Finalize();
23     return 0;
24 }
25
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpicc blocking.c -o b
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpirun ./b
Process 4 received message 3 from process 3
Process 5 received message 4 from process 4
Process 0 received message 5 from process 5
Process 1 received message 0 from process 0
Process 2 received message 1 from process 1
Process 3 received message 2 from process 2
○ aftab@Aftab:~/Desktop/AB/HPC/Assignment8$
```

Q3. Write a MPI program to find the sum of all the elements of an array A of size n. Elements of an array can be divided into two equals groups. The first $[n/2]$ elements are added by the first process, P0, and last $[n/2]$ elements the by second process, P1. The two sums then are added to get the final result.

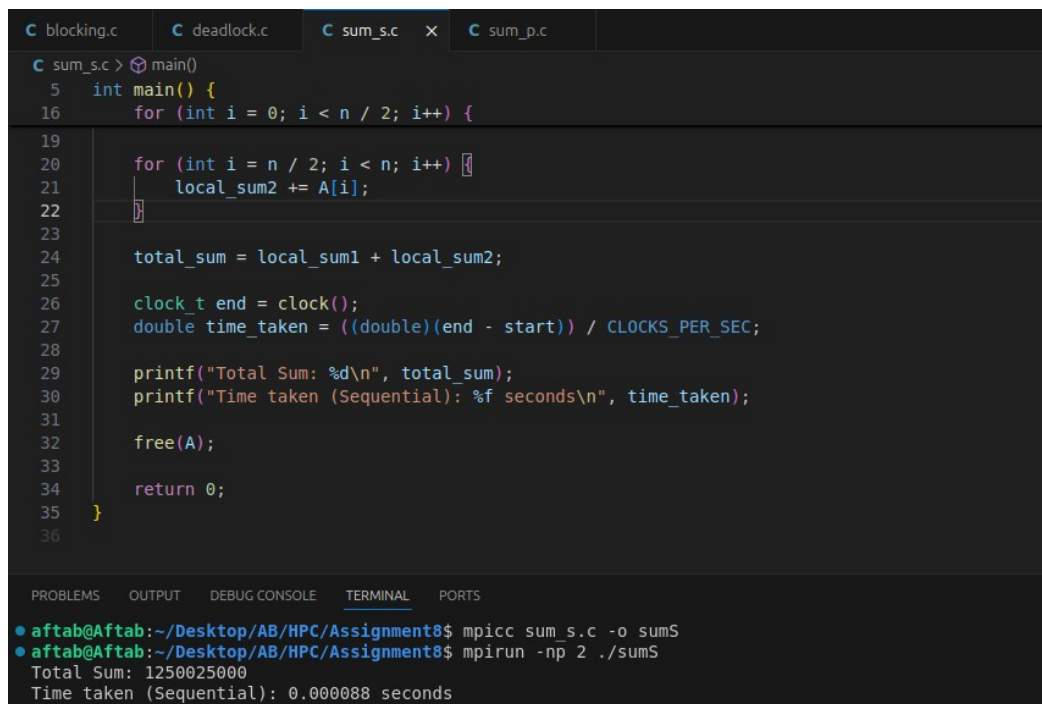
a) Sequential Code:

```
#include <stdio.h>

#include <stdlib.h>

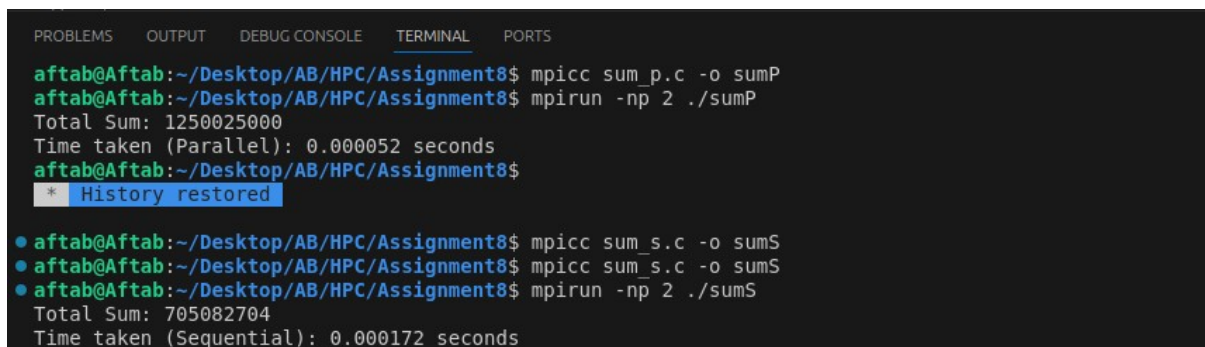
#include <time.h>

int main() {
    int n = 50000;
    int *A = (int*) malloc(n * sizeof(int));
    int local_sum1 = 0, local_sum2 = 0, total_sum = 0;
    for (int i = 0; i < n; i++) {
        A[i] = i + 1;
    }
    clock_t start = clock();
    for (int i = 0; i < n / 2; i++) {
        local_sum1 += A[i];
    }
    for (int i = n / 2; i < n; i++) {
        local_sum2 += A[i];
    }
    total_sum = local_sum1 + local_sum2;
    clock_t end = clock();
    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Total Sum: %d\n", total_sum);
    printf("Time taken (Sequential): %f seconds\n", time_taken);
    free(A);
    return 0;
}
```



```
C blocking.c C deadlock.c C sum_s.c X C sum_p.c
C sum_s.c > main()
5 int main() {
16 for (int i = 0; i < n / 2; i++) {
19
20     for (int i = n / 2; i < n; i++) {
21         local_sum2 += A[i];
22     }
23
24     total_sum = local_sum1 + local_sum2;
25
26     clock_t end = clock();
27     double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
28
29     printf("Total Sum: %d\n", total_sum);
30     printf("Time taken (Sequential): %f seconds\n", time_taken);
31
32     free(A);
33
34     return 0;
35 }
36

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpicc sum_s.c -o sumS
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpirun -np 2 ./sumS
Total Sum: 1250025000
Time taken (Sequential): 0.000088 seconds
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpicc sum_p.c -o sumP
aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpirun -np 2 ./sumP
Total Sum: 1250025000
Time taken (Parallel): 0.000052 seconds
aftab@Aftab:~/Desktop/AB/HPC/Assignment8$
* History restored
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpicc sum_s.c -o sumS
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpicc sum_s.c -o sumS
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpirun -np 2 ./sumS
Total Sum: 705082704
Time taken (Sequential): 0.000172 seconds
```

a) Parallel Code

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

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

```
#include <stdlib.h>
```

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

```
{
```

```
    MPI_Init(&argc, &argv);
```

```
int rank, size;

MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

if (size != 2)
{
    if (rank == 0)
    {
        printf("This program requires exactly 2 processes.\n");
    }
    MPI_Finalize();
    return 1;
}

int n = 50000;
int *A = (int *)malloc(n * sizeof(int));
int local_sum = 0, total_sum = 0;

if (rank == 0)
{
    for (int i = 0; i < n; i++)
    {
        A[i] = i + 1;
    }
}

MPI_Bcast(A, n, MPI_INT, 0, MPI_COMM_WORLD);

double start_time = MPI_Wtime();
```

```
if (rank == 0)
{
for (int i = 0; i < n / 2; i++)
{
local_sum += A[i];
}
}
else if (rank == 1)
{
for (int i = n / 2; i < n; i++)
{
local_sum += A[i];
}
}

MPI_Reduce(&local_sum, &total_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);

double end_time = MPI_Wtime();

double time_taken = end_time - start_time;

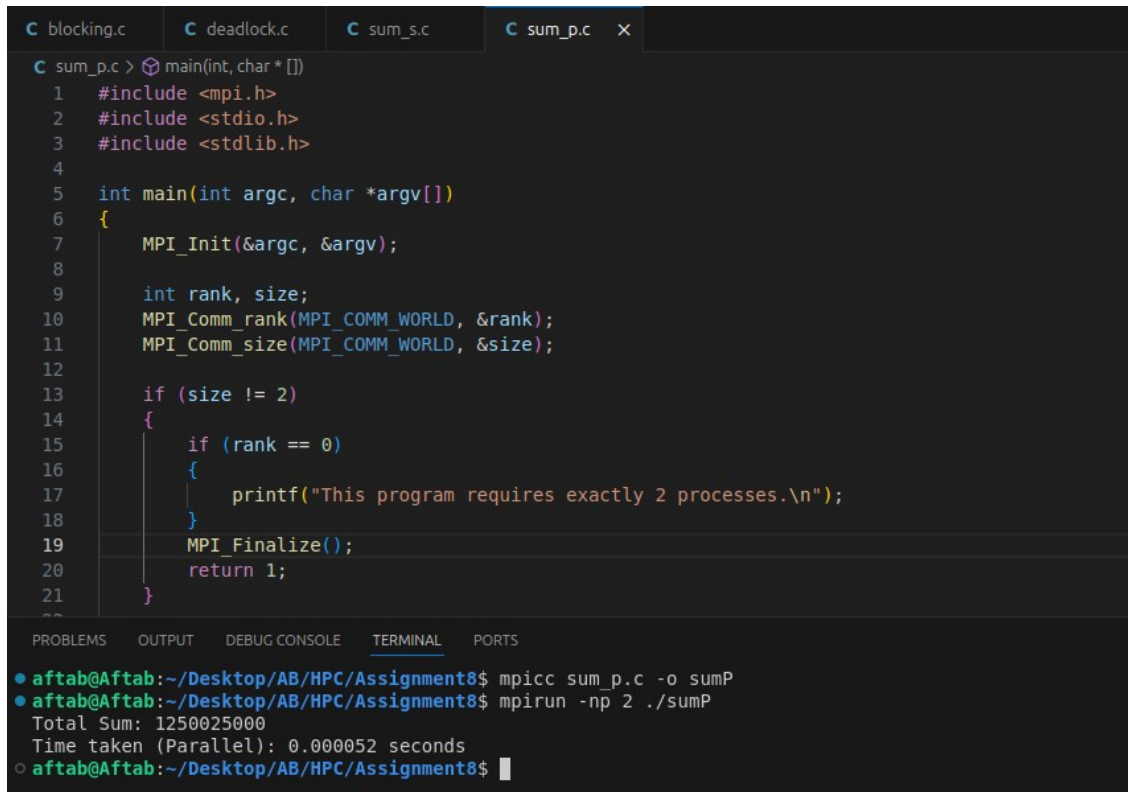
if (rank == 0)
{
printf("Total Sum: %d\n", total_sum);
printf("Time taken (Parallel): %f seconds\n", time_taken);
}

free(A);

MPI_Finalize();

return 0;
}
```

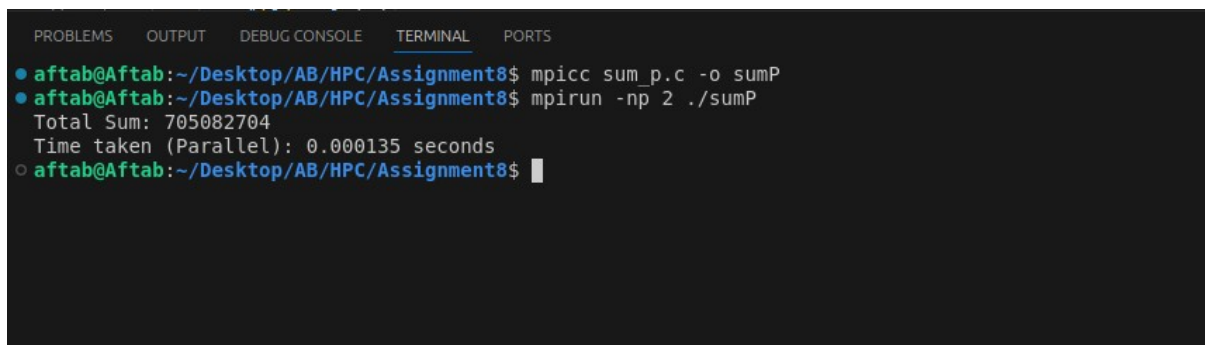
Walchand College of Engineering, Sangli
Department of Computer Science and Engineering



```
C blocking.c C deadlock.c C sum_s.c C sum_p.c X
C sum_p.c > main(int, char *[])
1  #include <mpi.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main(int argc, char *argv[])
6  {
7      MPI_Init(&argc, &argv);
8
9      int rank, size;
10     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
11     MPI_Comm_size(MPI_COMM_WORLD, &size);
12
13     if (size != 2)
14     {
15         if (rank == 0)
16         {
17             printf("This program requires exactly 2 processes.\n");
18         }
19         MPI_Finalize();
20         return 1;
21     }
22 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpicc sum_p.c -o sumP
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpirun -np 2 ./sumP
Total Sum: 1250025000
Time taken (Parallel): 0.000052 seconds
○ aftab@Aftab:~/Desktop/AB/HPC/Assignment8$
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpicc sum_p.c -o sumP
● aftab@Aftab:~/Desktop/AB/HPC/Assignment8$ mpirun -np 2 ./sumP
Total Sum: 705082704
Time taken (Parallel): 0.000135 seconds
○ aftab@Aftab:~/Desktop/AB/HPC/Assignment8$
```

Analysis:

Input Size (n)	Parallel	Sequential
50000	0.000052	0.000088
100000	0.000135	0.000172