NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA SURATHKAL
DEPARTMENT OF INFORMATION TECHNOLOGY
IT 301 Parallel Computing LAB 9

P Akshara - 181IT132

1. Non-Blocking Send and Receive

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size, myrank, x, i;
MPI_Status status;
MPI_Request request;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)
{
x = 10;
MPI_Isend(&x, 1, MPI_INT, 1, 20, MPI_COMM_WORLD, &request); // Tag is different at receiver.
printf("Send returned immediately\n");
}
else if (myrank == 1)
{
printf("Value of x is : %d\n", x);
MPI_Irecv(&x, 1, MPI_INT, 0, 25, MPI_COMM_WORLD, &request);
printf("Receive returned immediately\n");
}
MPI_Finalize();
return 0;
}
```

Output

```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpicc q1.c -o q1
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 2 ./q1
Send returned immediately
Value of x is : 0
Receive returned immediately
```

a) In non blocking send, processing continues immediately without waiting for the message to be copied out from the application buffer to system buffer. Blocking send returns only after it is safe to modify the application buffer (either data received or sent to system buffer)
b) MPI_Wait() blocks till the non blocking operations complete. Due to mismatch in tag, deadlock occurs.

```
14      MPI_Isend(&x, 1, MPI_INT, 1, 20, MPI_COMM_WORLD, &request); // Tag is different at receiver.
15      MPI_Wait(&request, &status);
16      printf("Send returned immediately\n");
17  }
18  else if (myrank == 1)
19  {
20      printf("Value of x is : %d\n", x);
21      MPI_Irecv(&x, 1, MPI_INT, 0, 25, MPI_COMM_WORLD, &request);
22      MPI_Wait(&request, &status);
23      printf("Receive returned immediately\n");
24  }
25  MPI_Finalize();
26  return 0;
27  }
```

```
TERMINAL  PROBLEMS 2  OUTPUT  DEBUG CONSOLE                    1: mpiexec

(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpicc q1.c -o q1
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 2 ./q1
Send returned immediately
Value of x is : 32594
```

2. Demonstration of Broadcast operation : MPI_Bcast().

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size, myrank, x;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)
{
scanf("%d", &x);
}
MPI_Bcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD);
printf("Value of x in process %d : %d\n", myrank, x);
MPI_Finalize();
return 0;
}
```

Output

With root as 1, garbage values for x. Taking the root to be process with rank 0

```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpicc q2.c -o q2
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 2 ./q2
3
Value of x in process 0 : 3
Value of x in process 1 : 3
```

3. Demonstration of MPI_Reduce with Sum Operation

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size, myrank, i, x, y;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
x = myrank; // Note the value of x in each process.
MPI_Reduce(&x, &y, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
if (myrank == 0)
{
printf("Value of y after reduce : %d\n", y);
}
MPI_Finalize();
return 0;
}
```

Output

```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 5 ./q2
Value of y after reduce : 10
```

4. Demonstration of MPI_Gather():

```
#include <mpi.h>
#include<stdio.h>
int main(int argc, char *argv[])
{
int size, myrank, x = 10, y[5], i;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Gather(&x, 1, MPI_INT, y, 1, MPI_INT, 0, MPI_COMM_WORLD);
if (myrank == 0)
{
for (i = 0; i < size; i++)
printf("\nValue of y[%d] in process %d : %d\n", i, myrank, y[i]);
}
MPI_Finalize();
return 0;
}
```

Output

```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpicc q4.c -o q2
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 2 ./q2

Value of y[0] in process 0 : 10

Value of y[1] in process 0 : 10
```

5. Demonstration of MPI_Scatter()

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int size, myrank, x[8], y[2], i;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0)
{
printf("Enter 8 values into array x:\n");
for (i = 0; i < 8; i++)
scanf("%d", &x[i]);
}
MPI_Scatter(x, 2, MPI_INT, y, 2, MPI_INT, 0, MPI_COMM_WORLD);
for (i = 0; i < 2; i++)
printf("\nValue of y in process %d : %d\n", myrank, y[i]);
MPI_Finalize();
return 0;
}
```

Output

4 processes receiving 2 values each

```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpicc q5.c -o q2
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 4 ./q2
Enter 8 values into array x:
1 2 3 4 5 6 7 8

Value of y in process 0 : 1

Value of y in process 0 : 2

Value of y in process 1 : 3

Value of y in process 1 : 4

Value of y in process 2 : 5

Value of y in process 2 : 6

Value of y in process 3 : 7

Value of y in process 3 : 8
```

6. Write an MPI program to find the smallest element in a given array of size N

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

```
// No.of elements each processor works on
const long int local_size = 1e6;

int main(int argc, char **argv)
{
int rank, size,local_arr[local_size],local_min[2],global_min[2];
double start,end;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
srand(rank+1);
start = MPI_Wtime();
for (int i = 0; i < local_size; i++)
local_arr[i] = rand()%100;
local_min[0] = local_arr[0];
for (int i = 1; i < local_size; i++)
if (local_arr[i] < local_min[0])
local_min[0] = local_arr[i];
local_min[1] = rank;
MPI_Reduce(local_min, global_min, 1, MPI_2INT, MPI_MINLOC, 0, MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
end = MPI_Wtime();
if (rank == 0)
{
printf("No. of processes = %d\t Chunk size = %ld\nNo. of elements in array is %ld\nRank %d has lowest value of
%d\n", size, local_size, local_size*size, global_min[1], global_min[0]);
printf("Time taken is %f \n", end-start);
}
MPI_Finalize();
return 0;
}
```

Output

```
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 5 ./q61
No. of processes = 5     Chunk size = 200000
No. of elements in array is 1000000
Rank 0 has lowest value of 0
Time taken is 0.013439
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpicc q61.c -o q61
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 20 ./q61
No. of processes = 20    Chunk size = 50000
No. of elements in array is 1000000
Rank 0 has lowest value of 0
Time taken is 0.008890
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpicc q61.c -o q61
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 50 ./q61
No. of processes = 50    Chunk size = 2000
No. of elements in array is 100000
Rank 0 has lowest value of 0
Time taken is 0.025324
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpicc q61.c -o q61
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 50 ./q61
No. of processes = 50    Chunk size = 20000
No. of elements in array is 1000000
Rank 0 has lowest value of 0
Time taken is 0.051067
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpicc q61.c -o q61
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 25 ./q61
No. of processes = 25    Chunk size = 4000
No. of elements in array is 100000
Rank 0 has lowest value of 0
Time taken is 0.013973
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpicc q61.c -o q61
(base) akshara@akshara-VivoBook-ASUSLaptop-X530FN-S530FN:/media/akshara/DATA/NITK/Lab-Sem5/IT301 PC/Lab 9$ mpiexec -n 25 ./q61
No. of processes = 25    Chunk size = 40000
No. of elements in array is 1000000
Rank 0 has lowest value of 0
Time taken is 0.011216
```

Taking Total No. of array elements = 1000000

Sl. No.	No. of Processes	Chunk Size (local array size)	Time (s)
1	1	1000000	0.019354
2	5	200000	0.013439

3	10	100000	0.009126
4	20	50000	0.008890
5	25	40000	0.011216
6	50	20000	0.051067
7	100	10000	0.088301

From the above table, it is clear that for 10^6 array elements, having around 20 processes with chunksize as 50000 works the best. Having lesser no. of processes results in more time being spent in each process for finding the local minimum. Having greater no. of processes also increases computation time due to overheads in spawning the processes and intermediate communications.

For finding min element we can use MPI_Reduce with MPI_MINoperation. Here, I have used MPI_MINLOC to also get the rank of the processor which found the minimum element.
We can use MPI_Scatter for giving elements to every processor. But that can result in additional overhead. So here I have taken a local array, on which every processor will operate and then used MPI_Reduce.