
# *Introduction to Parallel Programing Techniques*

## *Deferred Assessment*

Professor: Stylianos Sygletos

Nestor Edgar Sandoval Alaguna
200243856

# Assignment 2

**Exercise – 1 [2 points]**

Write an MPI program where each of p processes, generates 20 random integers in the range [-100, 100]. Then, process 0 must print the single largest integer generated and the rank(s) of the process(es) containing it. Use of MPI_MAXLOC is not allowed. Run examples with p>=8 to demonstrate the correct implementation of your code.

**Code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
void rand_generator(int n, int rnd_array[], int my_rank);
int main(int argc, char* argv[])
{
int comm_sz, my_rank;
int i;int n = 20;
int rnd_array[20];
int local_m, * global_m;
MPI_Comm comm;
MPI_Init(NULL, NULL);
comm = MPI_COMM_WORLD;
MPI_Comm_size(comm, &comm_sz);
MPI_Comm_rank(comm, &my_rank);
// Create n random integers
srand(my_rank);
rand_generator(n, rnd_array, my_rank);
// Calculate local maximum
local_m = rnd_array[0];
for (i = 1; i < n; i++)
if (rnd_array[i] > local_m)
local_m = rnd_array[i];
// Extract local maximum from each process
global_m = (int*)malloc(comm_sz * sizeof(int));
MPI_Gather(&local_m, 1, MPI_INT, global_m, 1, MPI_INT,
0, MPI_COMM_WORLD);
if (my_rank == 0)
{
int global_l = global_m[0];
for (i = 1; i < comm_sz; i++)
if (global_m[i] > global_l)
global_l = global_m[i];
// Print
printf("The biggest integer is: %d\n", global_l);
for (i = 0; i < comm_sz; i++) {
if (global_m[i] == global_l)
printf("Rank %d has the maximum\n", i);
}
}
free(global_m);
MPI_Finalize();
return 0;
}
void rand_generator(int n, int rnd_array[], int my_rank)
{
```

```
int i,t;
for (i = 0; i < n; i++)
rnd_array[i] = (rand() % 200) - 100;
printf("For rank %d numbers are: [", my_rank);
for (i = 0; i < n; i++)
printf("%d,", rnd_array[i]);
printf("]\n");
}
```

```
C:\Users\Nestor\Downloads\Parallel programming\assignment2>mpiexec -n 9 ex111.exe
For rank 5 numbers are: [-46,-7,-45,-51,-40,-70,27,-51,-28,-60,14,21,33,-24,-74,7,63,7,50,-69,]
For rank 7 numbers are: [-39,-78,-85,-60,21,16,52,94,27,44,-82,-91,-58,-99,-58,-85,-37,89,-88,12,]
For rank 3 numbers are: [-52,96,-6,-9,-69,77,2,3,-83,68,-91,-67,-77,52,78,99,-37,-75,-11,18,]
For rank 6 numbers are: [-42,-27,19,45,-94,89,-60,22,-84,-8,66,65,4,-45,-82,-55,-87,64,81,56,]
For rank 2 numbers are: [-55,-84,98,95,-16,-50,90,-69,5,16,57,89,-48,-94,53,-71,-87,67,-42,61,]
For rank 1 numbers are: [-59,-33,34,0,69,24,-22,58,62,-36,5,45,-19,-73,61,-9,95,42,-73,-64,]
For rank 4 numbers are: [-49,45,59,86,45,4,-85,76,60,-80,-39,-23,62,30,-98,37,13,-18,19,-26,]
For rank 8 numbers are: [-36,70,11,3,-65,-57,-3,-33,-29,95,-30,-47,-87,79,-34,53,-19,-53,-58,-31,]
For rank 0 numbers are: [-62,19,-62,-63,-45,97,65,-15,-50,-88,-47,0,42,-19,37,21,45,-15,97,-20,]
The biggest integer is: 99
Rank 3 has the maximum

C:\Users\Nestor\Downloads\Parallel programming\assignment2>
```

**Exercise – 2 [4 points]**

Write your own implementation of MPI_Bcast using MPI_Send and MPI_Recv. Your routine, myMPI_Bcast, should take the same arguments as the original MPI_Bcast; that is:
int myMPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root,
MPI_Comm comm);
Subsequently, write an MPI program where the root process initializes a vector of three elements with random values and uses your routine to broadcast the contents of the vector to the other processes. Identify and run appropriate examples to demonstrate the correct implementation of your code.

**Code:**

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


int myMPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root,MPI_Comm
comm);
int myMPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root,MPI_Comm
comm) {
int my_rank;
int comm_sz;
int i;
MPI_Comm_size(comm, &comm_sz);
MPI_Comm_rank(comm, &my_rank);
if (my_rank == root) {
for (i = 1; i < comm_sz; i++) {

MPI_Send(buffer, count, datatype, i, 0, comm);

}
```

```
}
else {
MPI_Recv(buffer, count, datatype, root, 0, comm, MPI_STATUS_IGNORE);
}
}


int main(int argc, char* argv[]) {
int my_rank;
int comm_sz;
int local_n;
int rand_vec[3];
int i;
int n = 3;
MPI_Comm comm;
MPI_Init(NULL, NULL);
comm = MPI_COMM_WORLD;
MPI_Comm_size(comm, &comm_sz);
MPI_Comm_rank(comm, &my_rank);
srand(my_rank);
if (my_rank == 0) {
for (i = 0; i < n; i++)
rand_vec[i] = rand() % 5;
printf("The rank %d has the generated vector: [", my_rank);
for (i = 0; i < n; i++)
printf("%d,", rand_vec[i]);
printf("]\n");
myMPI_Bcast(rand_vec, 3, MPI_INT, 0, comm);
}
else {
for (i = 0; i < n; i++) {
myMPI_Bcast(rand_vec, 3, MPI_INT, 0, comm);
printf("The rank %d has received the vector: [", my_rank);
for (i = 0; i < n; i++)
printf("%d,", rand_vec[i]);
printf("]\n");
}
}
MPI_Finalize();
}
```

```
C:\Users\Nestor\Downloads\Parallel programming\assignment2>mpiexec -n 9 ex22-.exe
The rank 4 has received the vector: [8,9,8,]
The rank 6 has received the vector: [8,9,8,]
The rank 3 has received the vector: [8,9,8,]
The rank 7 has received the vector: [8,9,8,]
The rank 2 has received the vector: [8,9,8,]
The rank 1 has received the vector: [8,9,8,]
The rank 0 has the generated vector: [8,9,8,]
The rank 5 has received the vector: [8,9,8,]
The rank 8 has received the vector: [8,9,8,]

C:\Users\Nestor\Downloads\Parallel programming\assignment2>
```

**Exercise – 3 [4 points]**

Write your own implementation of MPI_Reduce using MPI_Send and MPI_Recv. Your routine, myMPI_Reduce, will have the following signature:
int myMPI_Reduce(const void *sendbuf, void *recvbuf, int count, int root,MPI_Comm comm);

This should be a simplified version of the original MPI_Reduce which assumes double precision data (double) and performs the equivalent to the reduction operation MPI_MIN.
Subsequently, write an MPI program where each process generates one random value and uses your routine to obtain the minimum among them.

**Code:**

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <mpi.h>
int myMPI_Reduce(int* sendbuf, int* recvbuf, int count, MPI_Datatype datatype, int
root, MPI_Comm comm){
int my_rank, comm_sz;
MPI_Comm_rank(comm, &my_rank);
MPI_Comm_size(comm, &comm_sz);
if (my_rank != root){
MPI_Send(sendbuf, count, datatype, root, 0, comm);
}
else{
for (int i = 1; i < comm_sz; i++)
{
MPI_Recv(recvbuf, count, datatype, i, 0, comm, MPI_STATUS_IGNORE);
if (*recvbuf < *sendbuf){
*sendbuf = *recvbuf;
}
}
}
return 0;
}
int main(int argc, char** argv){
int sendbuf, recvbuf;
int my_rank, comm_sz;
MPI_Comm comm;
comm = MPI_COMM_WORLD;
MPI_Init(NULL, NULL);
MPI_Comm_rank(comm, &my_rank);
MPI_Comm_size(comm, &comm_sz);
srand(my_rank + 1 + time(NULL));
sendbuf = rand() % 100;
printf("The processor %d has the number %d\n", my_rank, sendbuf);
myMPI_Reduce(&sendbuf, &recvbuf, 1, MPI_INT, 0, MPI_COMM_WORLD);
if (my_rank == 0)
{
printf("The minimum of all of them %d", sendbuf);
}
printf("\n");
MPI_Finalize();
return 0;
}
```

```
C:\Users\Nestor\Downloads\Parallel programming\assignment2>mpiexec -n 9 ex33.exe
The processor 4 has the number 93

The processor 7 has the number 3

The processor 5 has the number 96

The processor 6 has the number 0

The processor 3 has the number 90

The processor 8 has the number 6

The processor 1 has the number 83

The processor 2 has the number 87

The processor 0 has the number 80
The minimum of all of them 0

C:\Users\Nestor\Downloads\Parallel programming\assignment2>
```