Computer Engineering &
Software Systems Program

Ain Shams University
Faculty of Engineering

Lab Assignment 2

Under Supervision of

Dr. Tamer Mostafa

&

Dr Mirvet Al-Qutt

&

Eng. Beshoy

Submitted By:

Ahmed Khaled Saad Ali Mekheimer
ID: 1809799

1. Problem Statement

We want to implement Vector by Matrix Multiplication using MPI, to have a good performance from using parallel processes running.

2. Solution

Using Collective Communication APIs in MPI, MPI_Scatter & MPI_Gather is the primary point to solve such problem.

As it will help distribute values of Matrix on each process having a row of Matrix, then each process will multiply the row it has by Vector in a local sum variable then we gather back the local sum variables to a Result array.

Flow of code:

1. Initialize MPI Environment.
2. We will have all arrays defined dynamically to have their dimensions (rows or cols) flexible.
3. After defining Matrix, Vector and Result array, we will have Local_Matrix_row which is assigned for each process to take one "row" of size cols & Multiply it by Vector later on we will Gather all processes local values in Result Array.
4. local_sum is assigned for each process to calculate summation of (Local_Matrix_row values multiplied by Vector).

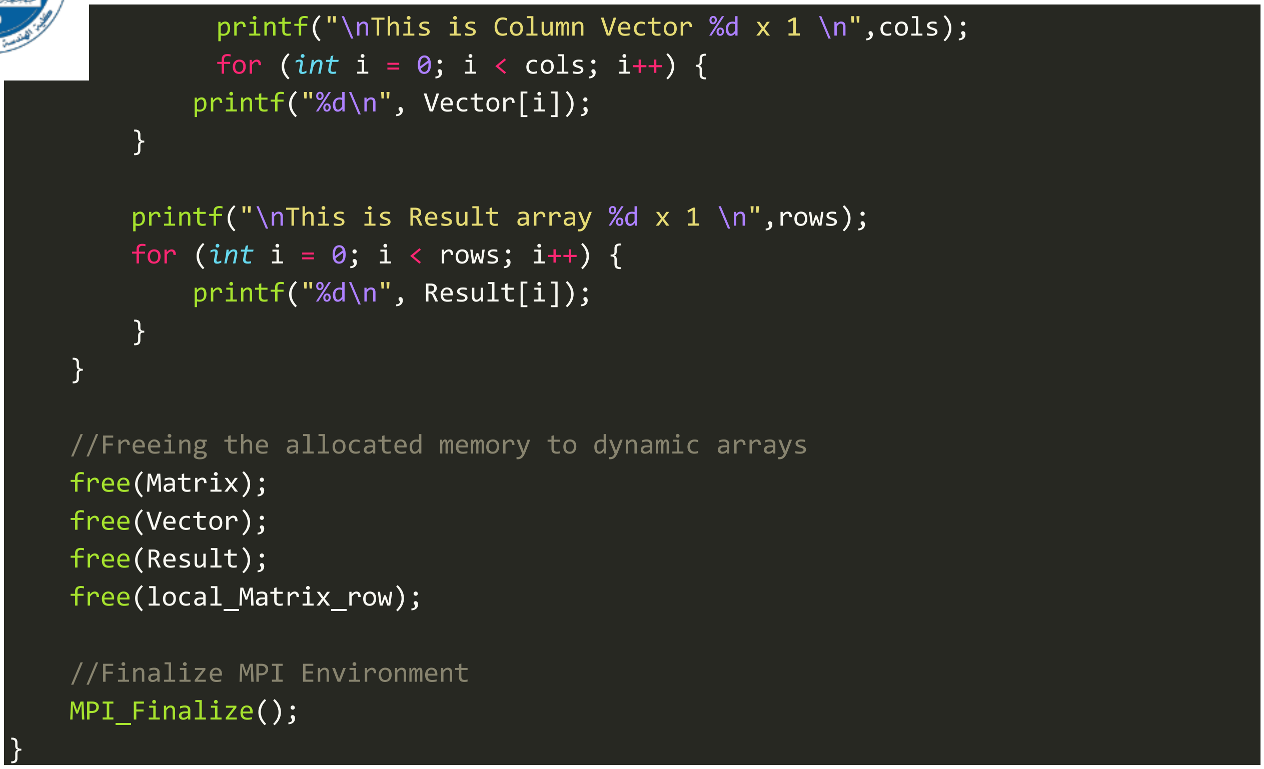
5. Just for fast testing, we will assign values of Vector & Matrix values in a for loop, but Matrix will only be assigned at Rank 0 process because later on it will be the Root process that Scatters data across processes.

6. Calling MPI_Scatter, Rank 0 process will scatter from Matrix number of elements = cols for each process in its local_Matrix_row Which means that we scatter one row for each process.

7. Each process will calculate summation of (its row values multiplied by Vector) in its local_sum. local_sums now represent values of Result array.
8. Calling MPI_Gather, we will gather local_sum of each process in Result array. Each process puts its local_sum in Result array at Rank 0 process which is the only process with Result array.
9. Printing Matrix, Vector and Result array.
10. Freeing the allocated memory to dynamic arrays.
11. Finalize MPI Environment.

3. Code

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

                  // WARNING //
//Number of processes (size) entered must be equal to rows
//Because each process takes one row from Matrix Array
//Assigning size > rows leads to "HEAP CORRUPTION DEBUGGING ERROR"

#define rows 5
#define cols 4

int main(int argc, char** argv) {
    //Initialize MPI Environment
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    //Using Dynamic arrays to be flexible in rows & cols values

    int* Matrix = (int*)malloc(rows * cols * sizeof(int));  //Matrix rowsxcols
    int* Vector = (int*)malloc(cols * sizeof(int));         //Vector cols
    int* Result = (int*)malloc(rows * sizeof(int));         //Result rows


    //local_Matrix_row is for each process to take one "row" of size cols &
Multiply it by Vector
    //Then Gather all processes local values in Result Array
    int* local_Matrix_row = (int*)malloc(cols * sizeof(int));

    //local_sum for each process to calculate summation of
(Local_Matrix_row  values multiplied by Vector)
    int local_sum = 0;

    //Assigning values for Vector
    //Vector is defined by all processes because they all will need it to
multiply it with the row they have from Matrix
    for (int i = 0; i < cols; i++) {
        Vector[i] = i;
    }
    //Assigning values for Matrix
    //Matrix must be assigned by one process (we chose rank=0), which will then
be the root process that scatters its rows among other processes
    if (rank == 0) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                Matrix[i * cols + j] = 2;
            }
        }
    }

    //Rank 0 process will scatter from Matrix number of elements = cols for each
process in its local_Matrix_row
    //Which means that we scatter one row for each process
    MPI_Scatter(Matrix, cols, MPI_INT, local_Matrix_row, cols, MPI_INT, 0,
MPI_COMM_WORLD);



    //Following is executed by all processes
    //Each process will calculate summation of (its row values multiplied by
Vector) in its local_sum
    for (int i = 0; i < cols; i++) {
        local_sum =local_sum+ (local_Matrix_row[i] * Vector[i]);
    }



    //local_sums now represent values of Result array
    //We will gather local_sum of each process in Result array
    //Each process puts its local_sum in Result array at Rank 0 process
    //Rank 0 process is the only one with Result array
    MPI_Gather(&local_sum, 1, MPI_INT, Result, 1, MPI_INT, 0, MPI_COMM_WORLD);

    //Printing Matrix, Vector and Result array
    if (rank == 0) {
        printf("\nThis is Matrix array %d x %d \n", rows,cols);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                printf(" %d ", Matrix[i * cols + j]);
            }
            printf("\n");
        }
```

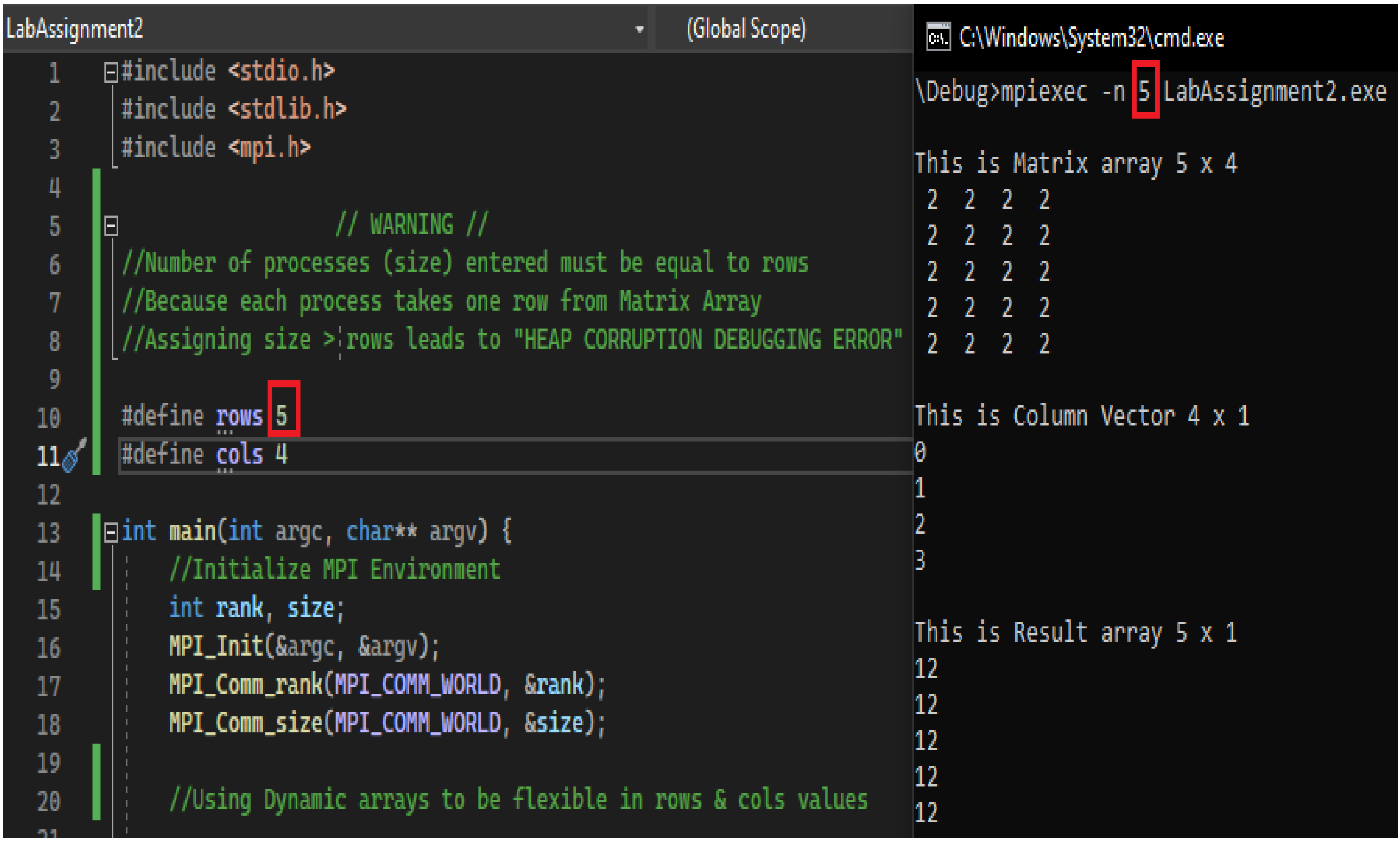


```
        printf("\nThis is Column Vector %d x 1 \n",cols);
        for (int i = 0; i < cols; i++) {
            printf("%d\n", Vector[i]);
        }

        printf("\nThis is Result array %d x 1 \n",rows);
        for (int i = 0; i < rows; i++) {
            printf("%d\n", Result[i]);
        }
    }

    //Freeing the allocated memory to dynamic arrays
    free(Matrix);
    free(Vector);
    free(Result);
    free(local_Matrix_row);

    //Finalize MPI Environment
    MPI_Finalize();
}
```

4. Output

Very Important Notes:

- Program can be run only by command not “run” in Visual Studio.
- Number of processes (size) entered in command must be equal to rows, because each process takes one row from Matrix Array.
  Assigning size > rows leads to "HEAP CORRUPTION DEBUGGING ERROR".
- Rows & cols can be changed from #define in the code to assign whatever dimensions.



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

                    // WARNING //
//Number of processes (size) entered must be equal to rows
//Because each process takes one row from Matrix Array
//Assigning size > rows leads to "HEAP CORRUPTION DEBUGGING ERROR"

#define rows 5
#define cols 4

int main(int argc, char** argv) {
    //Initialize MPI Environment
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    //Using Dynamic arrays to be flexible in rows & cols values
```

```
\Debug>mpiexec -n 5 LabAssignment2.exe

This is Matrix array 5 x 4
 2  2  2  2
 2  2  2  2
 2  2  2  2
 2  2  2  2
 2  2  2  2

This is Column Vector 4 x 1
0
1
2
3

This is Result array 5 x 1
12
12
12
12
12
```