HPC - Exercise 5

Eduardo Gonnelli

08 July 2019

1 Introduction

The aim of this report is to initialize a distributed identity matrix of size (N X N) employing the MPI. The global coordinates will be transformed to local tasks and given the number of processes and the dimension of the identity matrix, each process must allocate and set its own portion of the matrix. The Figure 1 shows the procedure for matrix decomposition among the processes.

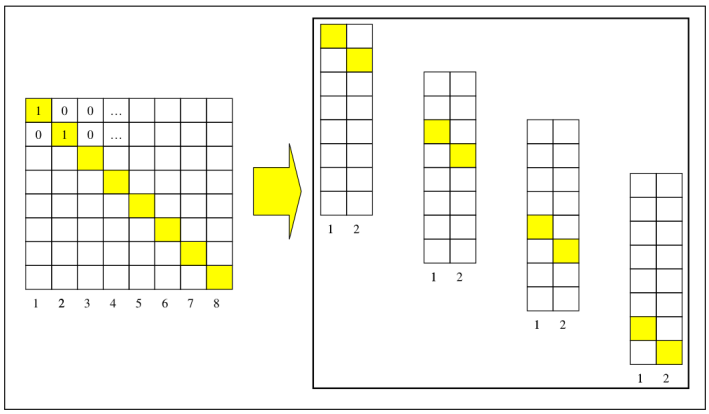

Figure 1: Identity Matrix Decomposition

2 Results

The Figure 2 shows the output of the program *matrix_id_mpi.o*. It can be seen that the number of processes was maintained constant (*-n 4*) and the size of the identity matrix (*N*) was set as 2, 4, 5 and 8.

```
[egonnell@cn07-08 ex05]$ mpirun -n 4 ./matrix_id_mpi.o 2
 1 0
 0 1
[egonnell@cn07-08 ex05]$ mpirun -n 4 ./matrix_id_mpi.o 4
 1 0 0 0
 0 1 0 0
 0 0 1 0
 0 0 0 1
[egonnell@cn07-08 ex05]$ mpirun -n 4 ./matrix_id_mpi.o 5
 1 0 0 0 0
 0 1 0 0 0
 0 0 1 0 0
 0 0 0 1 0
 0 0 0 0 1
[egonnell@cn07-08 ex05]$ mpirun -n 4 ./matrix_id_mpi.o 8
 1 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0
 0 0 1 0 0 0 0 0
 0 0 0 1 0 0 0 0
 0 0 0 0 1 0 0 0
 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 1
```

Figure 2: Some square identity matrices with different sizes (2, 4, 5 and 8).

The Figure 3 shows the output of the program *matrix_id_mpi.o* for the matrix size (N) equal 10. Due to this fact, the output was printed in a binary file called (*binary_matrix_i.txt*). It is possible to access the content of the text file typing the command *cat <Filename>*.

```
[egonnell@cn01-16 ex05]$ mpirun -n 4 ./matrix_id_mpi.o 10

 Matrix size (N >= 10)  DETECTED !!!
 Saving the result in the Binary file --> binary_matrix_i.txt

[egonnell@cn01-16 ex05]$ cat binary_matrix_i.txt
 1 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 1
```

Figure 3: Example of a identity matrix with size N equal 10. the result was saved in a binary file.

The Figure 4 shows the output of the program *matrix_id_mpi.o* maintaining the matrix size constant and varying the number of processes. The processes employed were: *2, 4, 8, 12 ,and 16*. All the results were the same. Probably the execution time would be different, however its measure is not the focus of this report.

```
[egonnell@cn07-08 ex05]$ mpirun -n 2 ./matrix_id_mpi.o 4
 1 0 0 0
 0 1 0 0
 0 0 1 0
 0 0 0 1
[egonnell@cn07-08 ex05]$ mpirun -n 4 ./matrix_id_mpi.o 4
 1 0 0 0
 0 1 0 0
 0 0 1 0
 0 0 0 1
[egonnell@cn07-08 ex05]$ mpirun -n 8 ./matrix_id_mpi.o 4
 1 0 0 0
 0 1 0 0
 0 0 1 0
 0 0 0 1
[egonnell@cn07-08 ex05]$ mpirun -n 12 ./matrix_id_mpi.o 4
 1 0 0 0
 0 1 0 0
 0 0 1 0
 0 0 0 1
[egonnell@cn07-08 ex05]$ mpirun -n 16 ./matrix_id_mpi.o 4
 1 0 0 0
 0 1 0 0
 0 0 1 0
 0 0 0 1
```

Figure 4: Running the *matrix_id_mpi.o* with the same identity matrix size N equal 4 but varying the number of processes. The outputs are the same.

References

[1] CINECA - *21st Summer School of Parallel Computing: SuperComputing, Applications and Innovation Department*
`https://www.cineca.it`

Appendices

A Codes

Listing 1: C++ code - Identity Matrix

```
#include<iostream>
#include<cstdlib>
#include<cstdio>
#include<mpi.h>

      int main(int argc,char **argv)
      {

      int rank, size;
      const int N = atoi( argv[1] );
      FILE *f = fopen( "binary_matrix_i.txt", "wb" );

                 MPI_Init( &argc, &argv );
                 MPI_Comm_rank( MPI_COMM_WORLD, &rank );
                 MPI_Comm_size( MPI_COMM_WORLD, &size );

    int rest = N % size;
    int local_N = ( N/size ) + ( ((int)rank < rest) ? 1 : 0 );
    int* A=( int* )calloc( local_N*N, sizeof(int) );
    int start = ( rank*local_N ) + ( ((int)rank < rest) ? 0: rest );

        for (int i=0; i<local_N; i++){
              A[ start + (i*N) + i ] = 1;
                                    }
        if(rank !=0){
          MPI_Send( A, N*local_N, MPI_INT, 0, 101, MPI_COMM_WORLD);
                    }
        if(rank==0){
          int* received = ( int* )malloc( N*local_N*sizeof(int) );
          MPI_Request req;
        // For loop calling the ranks
        for ( int i = 1; i < size; i++ ){

        MPI_Irecv( received,  N*local_N, MPI_INT, i, 101, MPI_COMM_WORLD, &req );

        if(N<10){
          for(int row=0; row<local_N; row++)
          {
              for (int col = 0; col < N; col++)
              {
              std::cout << "␣" << A[ col + row*N ];
              }
          std::cout << std::endl;
          }
         } else if(N>=10){
          for (int row = 0; row < local_N; row++)
           {
             for (int col = 0; col < N; col++)
             {
                fprintf(f,"␣%d", A[ col + row*N ]);
             }
             fprintf( f,"\n" );
           }
        }
                 MPI_Wait( &req, MPI_STATUS_IGNORE );
          // swap
          int* tmp = NULL;
          tmp = received;
          received = A;
          A = tmp;
          tmp = NULL;

        if( i == rest && rest != 0 )
        {
         local_N--;
        }
      // End of for loop calling the ranks
        }
          if(N<10)
            {
             for(int row=0; row<local_N; row++)
                {
                  for(int col=0; col<N; col++)
                    {
                      std::cout << "␣" << A[ col + row*N ];
                    }
                    std::cout << std::endl;
                }

             }else if(N>=10)
```

```
              {
                    for(int row = 0; row<local_N; row++)
                      {
                        for(int col=0; col<N; col++)
                            {
                             fprintf(f,"␣%d", A[ col + row*N ]);
                            }
                       fprintf( f,"\n" );
                      }
              std::cout << "\n" << "␣Matrix␣size␣(N␣>=␣10)␣␣DETECTED␣!!!" << "\n";
              std::cout << "␣Saving␣the␣result␣in␣a␣Binary␣file␣-->␣binary_matrix_i.txt" << "\n";
              std::cout << "\n";
              }

     free(received);
 }

     free(A);
                MPI_Finalize();

   fclose(f);

  return 0;
}
```