Supercomputing

Hands-On 4: Matrix Computation with MPI

1 Objectives

This last practice focuses on the parallel implementation of a more advanced numerical problem using the MPI environment that implements the MPI (Message Passing Interface) message passing standard. The objective of the practice is to face a more significant problem with more room for improvement. This practice consists of 2 objectives:

1. Study of the problem and implementation of the parallel version.
2. Improvement of the cyclical parallel version and experimental measurement of performance.

The practice will start from the sequential and parallel version by blocks of rows found in the cloud, and will have to be modified to include a cyclical distribution by rows.

2 Description of the problem

A system of equations $Ax = b$ has a unique solution (compatible and determined) when the coefficient matrix has a nonzero determinant. Several direct and iterative techniques for solving equations highlight the LU factorization among the natural methods.

The LU factorization consists of obtaining a pair of matrices, a lower triangular unit (all elements above the main diagonal to zero and the main diagonal itself to one) and another upper triangular (all elements above the diagonal zero and no restrictions on the main diagonal), both of the same dimension as the starting matrix.

In this way, the resolution of the system of equations is equivalent to the solution of two triangular systems, the programming of which is well known.

$$\left.\begin{array}{c} A = LU \\ Ax = b \end{array}\right\} \longrightarrow \quad LUx = b \quad \longrightarrow \left\{\begin{array}{c} Ly = b \\ Ux = y \end{array}\right. . \tag{1}$$

The calculation of the LU decomposition can be performed by Gaussian factorization, using the diagonal elements of the matrix as pivots and making zeros below the diagonal in each column.

The triangular systems can be solved by applying progressive elimination algorithms (for the case of the lower unit triangular matrix) back substitution (for the case of the upper triangular matrix).

2.1 Sequential Implementation

The student must implement the algorithms necessary to solve a system of linear equations. The three algorithms required to solve the problem explicitly (LU decomposition, resolution of the lower triangular system and resolution of the upper triangular system) are shown in Figure 1.

```
LU Decomposition
U = A;
for  k = 1, ..., n-1
  if ukk = 0 than
  lkk = 1
  for  i = k+1, ..., n
    m = uik/ukk
    lik = m
    uik = 0
    for j = k+1, ..., n
      uij = uij - m*ukj
    endfor
  endfor
endfor
lnn = 1
```

```
Progressive Elimination
% Ly = b
for i = 1, 2, ..., n
  yi = bi
  for j = 1, ..., i-1
    yi = yi - lij*yj
  endfor
endfor
```

```
Regressive Substitution
% Ux = b
for i = n , ..., 1
  xi = bi
  for j = i+1, ..., n
    xi = xi - uij*xj
  endfor
  xi = xi/uii
endfor
```

Figure 1: LU decomposition algorithms and resolution of triangular systems.

$$\begin{bmatrix} c_1 & f_2 & f_3 & f_4 & f_5 \\ c_2 & c_1 & f_2 & f_3 & f_4 \\ c_3 & c_2 & c_1 & f_2 & f_3 \\ c_4 & c_3 & c_2 & c_1 & f_2 \\ c_5 & c_4 & c_3 & c_2 & c_1 \end{bmatrix}$$

```
function [A, b] = createoep(c,f)
  n = size(c,1);
  for i=1:n
      b(i) = c(1);
      for j=1:i-1
          A(i,j) = c(i-j+1);
          A(j,i) = f(i-j+1);
          b(i) = b(i) + A(i,j);
          b(j) = b(j) + A(j,i);
      end
      A(i,i) = c(1);
  end
```

Figure 2: Example of a Toeplitz matrix and algorithm for its creation from row vectors (`f`) and column (`c`).

The three algorithms will be tested on equations with double precision real matrices generated by the Toeplitz expression, with dimensions that will vary between 64 and 1024. The algorithm speed (flops per second) should be measured for each of the two. Algorithms and represent those values on a graph.

A Toeplitz matrix is constructed from two vectors (f and c) that define the first row and column (the first value of both vectors must be the same).

The initial code tends to be rewritten in such a way that the output overwrites the input so that the coefficient matrix, after the execution of the function that performs the factorization, contains the matrices L and U, and that in solving equations, the vector of independent terms is overwritten with the solution of the system of equations. In the Figure 3 these algorithms are shown.

3 Parallel Implementation

The parallel implementation of solving a system of linear equations aims to perform several independent operations concurrently.

The most obvious parallelism occurs by rows, considering that the update of each row by the pivot is independent for each pivot (`i` loop). In this way, the coefficient matrix and the vector of independent terms could be distributed by rows, and the update carried out in parallel. This process would involve sending the pivot row to all the processes involved before starting to update their rows. Figure 4 shows a schematic of how the loops `j` are executed in parallel.

Equivalently, the triangular systems execute the update of each of the independent terms in parallel from

```
LU DECOMPOSITION
for  k = 1, ..., n-1
  if akk = 0 then
  for  i = k+1, ..., n
    aik = aik/akk
    for j = k+1, ..., n
      aij = aij - aik*akj
    endfor
  endfor
endfor
```

```
PROGRESSIVE ELIMINATION
for i = 1, 2, ..., n
  for j = 1, ..., i-1
    bi = bi - lij*bj
  endfor
endfor
```

```
REGRESSIVE SUBSTITUTION
for i = n , ..., 1
  for j = i+1, ..., n
    bi = bi - uij*bj
  endfor
  bi = bi/uii
endfor
```

Figure 3: LU decomposition algorithms and resolution of triangular systems with overwrite.

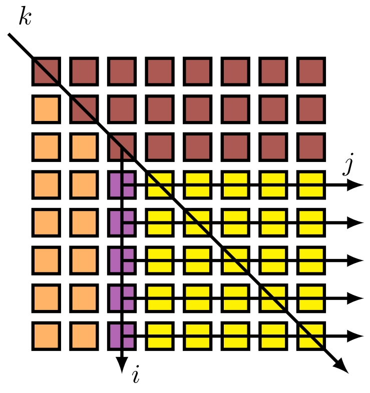

Figure 4: LU decomposition diagram where the path of the three loops is illustrated.

the new solutions obtained.

In the case of a row implementation, multiple data distributions present different results. Specifically, within the scope of this practice, we will study two possibilities:

1. Distribution by blocks of consecutive rows.
2. Cyclical distribution of the rows.

Each process must provide the pivot row if it has one or receives it from the corresponding method in another case. After that, you must update all the rows you have using the pivot row.

After factoring, the triangular systems must be solved. Similarly, in each iteration, a process will have all the information to calculate the value of the corresponding component in the vector of unknowns. This value should be propagated to all the other methods that still have pending rows so that they can update the vector of independent terms with the contributions in each step. Finally, each process must return the subvector of the unknowns calculated to the zero process.

Taking into account each of the distributions, the following considerations should be taken into account:

- **Row Blocks**. A balanced distribution of the different rows between the processors and valid for any problem size. Each processor, including 0, will be assigned a block of consecutive rows.
- **Cyclical Distribution** by Rows. A balanced distribution of the different rows between the processors and valid for any size of the problem. Each processor will be assigned a block of non-consecutive rows separated by a distance equal to the number of processors.

4 Parallel version code

The structure of the code for the parallel version is shown below. This program implements the generation of the matrix and the vector of independent terms, its distribution among the different processors through a distribution oriented to blocks of rows, the LU factorization, the resolution of the lower triangular system of equations, and the solution of the system of superior triangular equations, these last three in parallel. It also provides functions to reserve and free memory and to display matrices and vectors on the screen.

The code consists, in addition to the main function, of the following parts:

- `double **ppdGenMat(int nN, int nM)`
  The function receives the dimensions of a matrix and returns a vector of double pointers that points to the different rows of a dynamic matrix.

- `int nRellenaMat(double **ppdMat, int nN, int nM)`
  The function is given a dynamic matrix as a vector of pointers, and its two dimensions fill said matrix with the values of a Toeplitz matrix.

- `double *pdGenTI(int nN)`
  The function that, given a dimension, generates a dynamic vector of independent terms so that the solution of the system is an all-to-1 vector.

- `int nPrintMat(double **ppdMat, int nN, int nM)`
  The function prints in tabular form the content of a dynamic array whose dimensions are indicated in the arguments.

- `int nSustReg(double **ppdU, double *pdB, int nN, int nId, int nP)`
  The function that implements the backward substitution method to solve a system of equations of nN rows in which the matrix of coefficients is a singular triangular matrix. The solution is overwritten on the vector of independent terms. The function is executed on a system with nP processors, and each processor must supply the index of the process.

- `int nLiberaMat(double **ppdMat)`
  The function that frees the memory reserved for a dynamic array.

- `int nPrintVec(double *ppdVec, int nN)`
  Function that displays the content of a vector of dimension nN on the screen.

- `int nElimProg(double **ppdL, double *pdB, int nN, int nId, int nP)`
  The function that implements the progressive elimination method to solve a system of equations with nN rows in which the coefficient matrix is a unit lower triangular matrix. The solution is overwritten on the vector of independent terms. The function is executed on a system with nP processors, and each processor must supply the index of the process.

- `int nLU(double **ppdMat, int nN, int nId, int nP)`
  The function implements the LU factorization of a non-singular matrix supplied as a parameter and of dimension nN. The L and U overwrite the input matrix so that the matrix L occupies the lower triangle (except the diagonal) and U the upper triangle (including the diagonal). The function is executed on a system with nP processors, and each processor must supply the index of the process.

5 Work to be done

The behaviour should be analysed and properly justified with the comparison of cyclical parallel version and experimental measurement of performance. The solution must be prepared and delivered containing a description of the implementation, performance graphs (execution time, speedup, etc.) and the conclusions.

References

[1] F. Almeida, D. Giménez, J. M. Mantas and A. M. Vidal. *Introducción a la Programación Paralela*. Ed. Paraninfo, 2008, Spain.

[2] Forum, Message Passing Interface. *MPI: A Message-Passing Interface Standard*. University of Tennessee, 1994, USA.