scientific report: Jacobi method implementation on MS_MPI

Ammar Ali Master Student

Department Of Information Technology And Programming,

Business Information System,ITMO University, Saint Petersburg RUSSIA

(Dated: March 10, 2020)

I. INTRODUCTION

In this report, we will present the Jacobi method to solve linear equations. It is known that the Jacobi method has a complexity of O[n^2] for each iteration and the number of iterations is related to the number of coefficients and to the required accuracy. We will use the Microsoft Messaging Passing Interface (MPI) library to implement the Jacobi method based on multi-threading instead of running on one thread. Then we will discuss how the multi-threading will affect the run-time.

II. DISCUSSION

A. Jacobi method

In numerical linear algebra, the Jacobi method is an iterative algorithm for determining the solutions of a strictly diagonally dominant system of linear equations. Each diagonal element is solved for, and an approximate value is plugged in. The process is then iterated until it converges. Let $Ax = b$ be a system of linear equations, where:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

If we decompose A into three matrices D,L,U where D is the diagonal component ,L the lower triangle and U the upper triangle. Then we can obtain the solution iteratively according to the relation.

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.$$

A sufficient (but not necessary) condition for the method to converge is that the matrix A is strictly or irreducibly diagonally dominant. Strict row diagonal dominance means that for each row, the absolute value of the diagonal term is greater than the sum of absolute values of other terms:

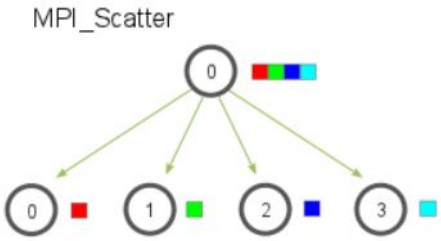
$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|.$$

From the above it was shown that for each iteration we need to calculate a vector-matrix multiplication which needs O[N^2] complexity.

III. IMPLEMENTATION

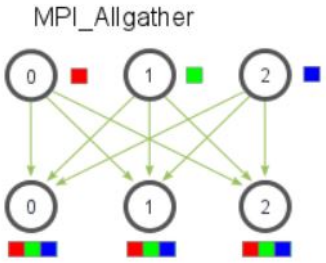
The idea is to split the matrix into parts and each thread will compute a part of the new vector in each iteration then we will gather the new vector from all threads and go to the second iteration. in each step it is also necessary to check the convergence "if the accuracy is more that a threshold value" to stop the infinite loop. Also it is important to check the sufficient condition of the convergence before starting any iteration to avoid the non convergence situations which holds when the absolute value of a diagonal element is bigger than the summation of all other elements on the same row. and also this will be done using multi threading. We will use MS-MPI library to implement the algorithm. To do that when we generate a random matrix we send data from one process to all other processes in a communicator. to do that we are using MPI_Scatter[**?**]. as well it was used when the vector

FIG. 1: send data from one process to all other processes in a communicator



is generated. then to modify the new vector for each operation to calculate the distance and check for the convergence we need to gather the information form the processes.So, Given a set of elements distributed across all processes, MPI_Allgather will gather all of the elements to all the processes[**?**].

FIG. 2: each process are gathered in order of their rank



IV. RESULTS

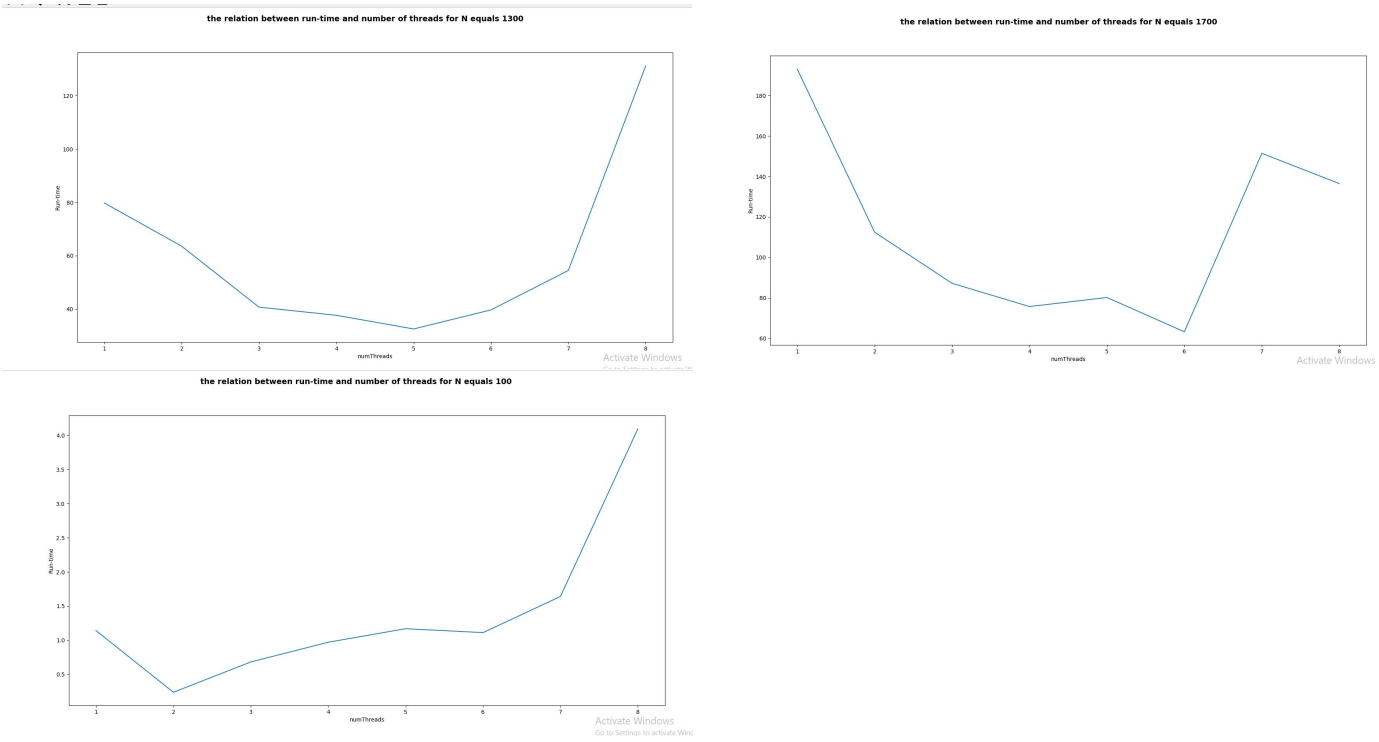
I generated 100000 random numbers between 1 and 100 and run the code on a different number of threads as shown in fig(2). The best Running Time obtained when the number of threads equals 4 and for more threads, the run-time started to increase again. Note: tor run the code please using command window go to the directory of the debug in the project folder and type:

mipexec -n 4 MPIfirst

or specify any number of threads you want. Also maybe a conflict occurs because of using $freopen$ to read from files.To solve this problem please add "_CRT_SECURE_NO_WARNINGS" to the $Project\ Properties -> C/C++ -> Preprocessor -> Preprocessor\ Definitions$.

I generated multiple matrices from dimensions of 100X100 to 2000X2000 increased by 200 so the testing matrices will have dimensions of the set (100,300,500,700 ..). The number of iterations initialized to be 300 with Error permission EPS equals to 10^{-6}. But it was shown that all matrices will converge after maximum 67 iterations as long as it is a diagonal dominated matrix. The testing matrices computed using different number of threads between 1 and 8 (Max of my laptop).And it showed the best number of threads to be used is different according to the size of matrices.

FIG. 3: run-time according to number of used threads for different matrix dimensions



As shown in the figures, the optimized number of threads related to the matrix dimensions. For small dimensions as 100 it shwoed that two threads will give the best running time by decreasing it more than the half in comparison with running on one thread. for a matrix dimension of 1700 the best performance was using 6 threads where the running time divided by three.‘ To see all figures please run the provided script.

V. REFERENCE

[1] [*https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/*].