Block Division Design

Devansh Tripathi

1 Introduction

This document explains the designing of the code used for performing Gaussian elimination in parallel using Message Passing Interface (MPI). This project was developed for comparing the algorithms for parallelizing Gaussian elimination. Two algorithms were analized- Serial and Parrallel algorithm.

In this project, we followed centralised distribution. In this distribution, we define the problem on the rank **0**. Tasks such as inputting the matrix and inputting right hand side will be accomplised by rank **0**. And then it has to be distributed to all other ranks using *MPI Send* and *MPI Recv* function.

2 Design

Since centralised distribution has been followed so rank **0** has all the data. Now rank **0** will distribute the data such as matrix and right hand side vector to other ranks using block division.

2.1 Block Division

Let's say $\boldsymbol{N}$ is order of matrix and the program is using $\boldsymbol{x}$ number of processors. Then each rank (including rank **0**) will get a block of matrix having $\boldsymbol{N/x}$ number of rows. The very first $\boldsymbol{N/x}$ number of rows will be send to rank 0 and next $\boldsymbol{N/x}$ will be send to rank 1 and so on. Hence every rank will have $\boldsymbol{(N/x) * N}$ size of matrix. Similar procedure will be followed to distribute the right hand side (rhs) and every rank (including rank 0) will have a right hand side (rhs) vector of size $\boldsymbol{(N/x) * 1}$.

2.2 Load Balancing

Load balancing simply means that every rank should have equal amount of work that they are doing and idle time for each rank should be minimized as much as possible. Block division algorithm may cause a problem in load balancing i.e. if every rank is using the previous row as pivot row then for say, rank 3 has to wait until rank 2 has done applying row operation on it's last row since for rank 3 's first row, rank 2's last row will be the pivot row.

This problem can be tackle with a different type of approach of deciding pivot row. The approach that has been followed in this code is that **instead of last row, first row of each rank should act as a pivot for the next rank. Hence as soon as rank** 0 **starts working, it will send it's first row to next rank and similarly other ranks will also send their first row to successor ranks.** For last rank, it would not send any row as pivot row. The idle time for each rank now have been minimized.

2.3 Gaussian Elimination

Now, all the ranks (including rank 0) will perform the row operations and convert the local matrix (that each rank has) to upper triangular matrix. Since local matrices are of the order $(N/x) * N$ i.e. they are rectangular matrices hence these will not convert to a perfect upper triangular matrices. But when they all are gathered at rank 0, they will constitute a perfect upper triangular matrix of order $N * N$.

2.4 Gathering of matrix

At this moment, all the rank will have a upper triangular matrix. All these ranks will send there local matrices to rank 0 and they will be gathered in the same order as they were distributed using *MPI Send* and *MPI Recv* functions. Hence, constitute a perfect upper triangular matrix of order $N * N$.

2.5 Back Substitution

Since rank 0 has the matrix in upper triangular form, it will start performing back substitution from downward and generate a solution vector. Rank 0 will also write this solution vector to a file.