Group 6

ID: 2016B1A70703P Name: Mayank Jasoria
ID: 2016B4A70487P Name: Puneet Anand

Gaussian Elimination

Execution Requirements

The program expects a single command line argument specifying the name of the input file. The structure of the file should be such that the first line specifies the number of equations. Thereafter, each line contains the coefficients of the variables of an equation, and the value of that equation, all present aas space separated values. For example. For the following system off equations:
3x + 4y - 2z = 5
2x + 1.6y + 4.4z = 3.9
7x - 3y + z = 1.2

The input file should be as follows:
3
3 4 -2 5
2 1.6 4.4 3.9
7 -3 1 1.2

Sample compilation, assuming mpich as the platform:
mpicc gaussian_parallel.c
mpiexec -n <num_processors> ./a.out <intput_file>

Here, num_processors should not exceed the number of equations, and must be a minimum of two.

Design

The algorithm comprises three major phases.

1. Partitioning of the data - After the input is read, the data is partitioned among all processes row-wise, such that each process receives one or more rows of data in a cyclic manner. Thus, a process with rank i will receive all rows having indices `index` mod `num_processes` = i. This cyclic partition scheme helps to reduce the cost due to waiting time, by providing a more even distribution of workload.

2. Gaussian Elimination along with partial pivoting, to compute the upper triangular matrix - This phase follows a pipelined communication architecture, such that each process receives a row of data only from its preceding process, and sends any data (either a received row, or one of its rows) only to the next process, by rank. When a process receives a row, it first communicates

ahead to the next waiting process, then performs elimination of all rows that it contains whose indices (according to the original matrix) are greater than that of the received row. When it is ascertained that some row of the current process can be used to perform elimination on other rows (this is when row index number of 0s lie before the element of that row whose index matches the row index), it first places a pivot, then performs a division of the pivot element with all other elements of the same row, then communicates this row forward.

3. Back substitution - This phase is similar to the previous phase in pipelined communication design, except that the order is reversed, and instead of complete rows, only the values of pivot elements of rows are communicated upward. Using this technique, the values of all variables are computed such that the system of equations can be satisfied.

Analysis of the algorithm:

Assuming,
Number of processes = p
Number of elements to be sorted = n
[assuming that $p < n$ and p > 1]

The algorithms is considered in 3 phases:

1. Partitioning of data
2. Gaussian Elimination (along with partial pivoting)
3. Back Substitution

Partitioning of data
Assuming that the given coefficient matrix A is present in the root process. Since we are doing a cyclic 1-D partitioning, Input is taken in the required order so as to exploit the benefits of Scatterv. Corresponding entries of the vector b is also partitioned.

Time complexity:

Gaussian Elimination (along with partial pivoting)
This phase translates the system of equations $Ax = b$ into $Ux = y$, which is parallelly solvable using back substitution.

At each step:
Each Processor follows the following sequence of actions:

1. Send the item that need to be communicated to the next processor in the pipeline.
    a. Do the elimination
    b. Find the pivot position (the column having the element with maximum absolute value)
    c. Bring the largest element to the pivot by exchanging the column entries of that row.

d. Perform division with the pivot
e. Communicate the index interchanged with the pivot and the row corresponding to the active part further in the pipeline.

2. Wait for receiving the input from the process preceding in the pipeline.

Number of subtractions and multiplications = O(n)*n/p if all the rows are present in the active part. Also it sends data to the next processor in the pipeline in O(n) time. Computation decides the complexity rather than the communication.

So Total Time complexity = $2*(n/p) \sum_{k=0}^{n-1} (n) = O(n^3/p)$

Back Substitution
It will go in bottom up manner, the opposite if used in gaussian elimination, the values of calculated x_i's are pipelined above.
Total time complexity = O(n/p) for computation,
In the bottom up traversal of pipeline, time = O(n)

Overall Time Complexity = $O(n^2/p)$

Theoretical Speedup = $Tseq \div Tp = (2n^3/3) \div ((n^3/p)) = 2p/3$

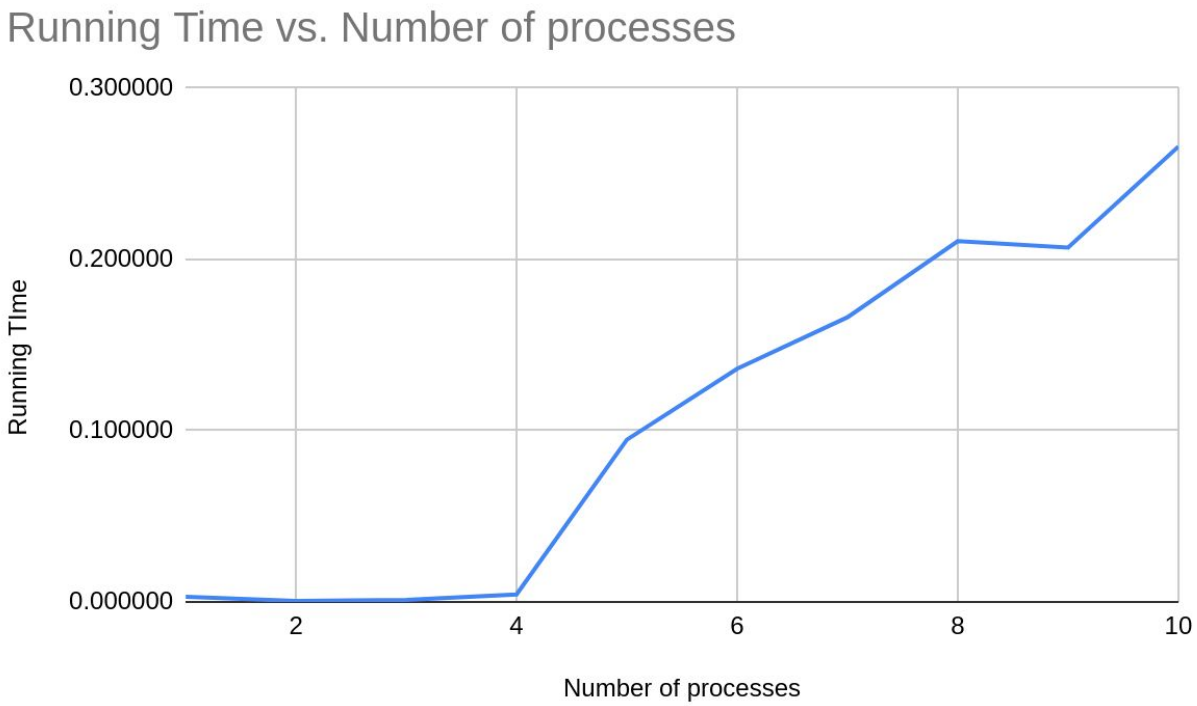
Efficiency = $Speedup / No. of Processors = 2/3$

Actual Speedup estimation:

The following table has been derived for running the algorithm from 1 to 10 processes on a dataset of 10 x 11 elements, with each row representing 10 coefficients of variables and the last element denoting the value of the expression.

Number of processes	**Run 1**	**Run 2**	**Run 3**	**Running TIme**
1	0.003002	0.003001	0.003002	0.003002
2	0.000637	0.000638	0.000394	0.000556
3	0.002495	0.000400	0.000699	0.001198
4	0.003461	0.007246	0.002372	0.004360
5	0.078218	0.102941	0.102941	0.094700
6	0.107150	0.134484	0.166840	0.136158
7	0.227966	0.137850	0.132726	0.166181

8	0.202757	0.267519	0.161612	0.210629
9	0.229296	0.201958	0.189464	0.206906
10	0.272404	0.245196	0.279885	0.265828



Although the running time would normally be expected to decrease with the number of processes, but due to a limitation on the number of processors, the average waiting time of each process increases mainly due to scheduling. With an increasing number of processors, the scheduling time continues to outweigh the benefits of parallel execution leading to the increasing time trend that is visible in the graph above.