MPI REPORT

Rosen Sasov Mohit Vellanki

Task 1

Parallel version of Matrix Multiplication

For the first task, we implemented the checkerboard matrix multiplication using Message Passing Interface (MPI). We implemented the parallel version on 1,2,4 and 8 cores and compared the execution times with the sequential version of matrix multiplication.

Implementation

After initializing the matrices to their default values, we transposed the second matrix (Matrix b in our case) . This is done to achieve a speedup which is due to the less number of looping statements required. The process can be completed quicker because the indexes are the same for both the matrices.

For transposing, we have used two methods. One of them is the normal method using the temporary variable and the other is by transposing the matrix without using a third or temporary variable.

An example for transposing using the first method is described below. The values of α and β are swapped using the following set of three equations and a temporary variable γ .

$$\alpha = \beta$$
$$\beta = \gamma$$
$$\gamma = \alpha$$

And in the second method the values of α and β are swapped using the following set of three equations without a temporary variable.

$$\alpha = \alpha + \beta$$
$$\beta = \alpha - \beta$$
$$\alpha = \alpha - \beta$$

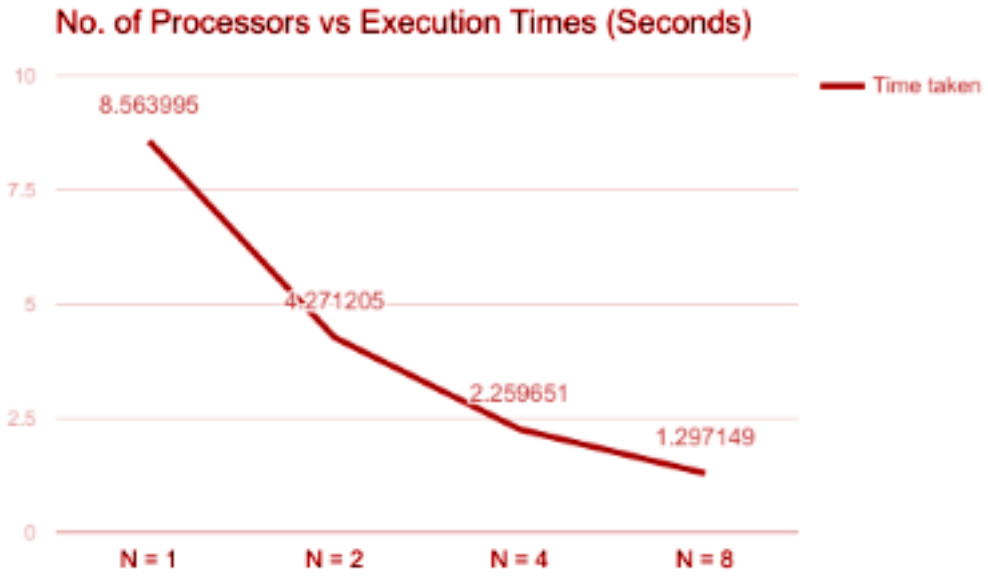
We noticed a slight difference in the execution times when we used the second method and hence implemented it rather than the normal way of transposing. We parallel executed the matrix multiplication by dividing the work among n nodes, out of which 1 is the master node and n-1 are the slave nodes. The data we are distributing in this matrix multiplication task is the rows and columns of the matrices that are to be multiplied . They are divided among the master and slaves by calculating the size of rows and columns per number of nodes which is obtained by dividing the number of rows and columns with the number of processors.

We are using offsets for locating the areas in which master and slaves will work. The master is always receiving offset 0 for rows and columns (the first element of the matrix) and in that way, depending on the size, the working area of the master is determined.

The offset is the first element which will be accessed by slave, then we are incrementing the offset with the calculated size per node and we receive in that way the whole area of the matrix which is going to be calculated by certain slave.

Execution Times

The execution time of the sequential version of matrix multiplication was calculated using the noticed to be 61.55 seconds.We have noticed that the execution times have decreased as expected when executed parallelly on more processors. These execution times (in seconds) have been shown in the graph below for N = 1, 2, 4 and 8 processors.



Task 2

Parallel version of Laplace Approximation using the SOR algorithm

For the second task, we implemented the Successive Over Relaxation (SOR) algorithm to calculate the Laplace approximation using Message Passing Interface (MPI). We

implemented the parallel version on 2,4 and 8 cores and compared the execution times with the sequential version of the Laplace approximation.

In the sequential version provided, the matrix elements were shifted to the left side and upwards to ensure that the border elements have all the required neighbouring elements. But we wanted to calculate the cell content without shifting and hence we have transmitted all the contents (N+2) of a required row to a node.

Initially the work to be done by the slaves is calculated with the help of the total number of nodes and the number of processors. After the calculation, the work is distributed to the slaves by the worker.

Number of processors	**Execution Time (In seconds)**
2	
4	
8	