Project Assignment: 1
MPI

Akhil Tummala
940913-8774
Aktu16@student.bth.se

Madhukar Enugurthi
950607-7537
Maen16@student.bth.se

Introduction:

Message passing Interface (MPI) is a standard message passing library based on the accord of MPI forum. The main goal of MPI is to provide efficient and flexible message passing widely used for writing message passing programs [1].

In MPI model, one process data is moved from its address space to another process through cooperation's on each process. It's basically designed for distributed memory [1].

Implementation:

Task 1: Blocked Matrix-Matrix Multiplication.

The matrix multiplication is only performed when the number of columns in the matrix A is equal to the number of rows in the matrix B. In this task, matrix multiplication is performed on two matrices A and B of size 1024*1024. To reduce the complexity for multiplication, the matrix B is transposed (interchanging the rows and columns). So, the multiplication is done using the rows. This matrix multiplication task is performed on 1,2,4 and 8 nodes. The work division for the matrix multiplication differs from one and another.

Initially, MPI is initialized in the main function. Then for 1 node, first the matrix is being initialized and then based on the matrix multiplication algorithm provided in the project manual, the matrix multiplication is performed without any work division since all the work is done on the single node.

For 2 nodes, the matrix multiplication is performed on 2 different nodes which are master and worker. Here first matrix is being initialized. Once the matrix initialization is done then the work division is done where the matrix A is divided into two equal parts (512*1024) as A0 and A1 and Matrix B (1024*1024) is not divided (Fig.1). Now using A0 and whole B matrix data matrix multiplication is performed in the master node, but before that the data of A1 and Whole B matrix is sent to the worker using the MPI_Send() function to perform the matrix multiplication in parallel. The data which is sent to the receiver is received using MPI_Recv() function. Once the data is received then the matrix multiplication is performed using A1 and B matrix data. After completion of the multiplication, the obtained result is sent back to the master node using the MPI_Send() function. The execution time for the matrix multiplication is calculated by checking the difference between start time and end time. This is done using the MPI_time() function. The start time is noted after the matrix initialization and the end time is noted after receiving the data from the worker nodes.

Master: A0 * B

Worker: A1 * B

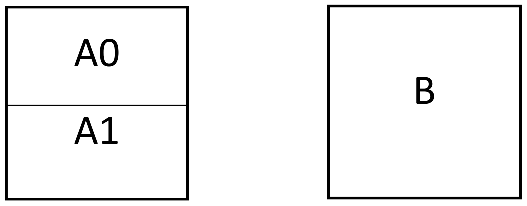


Fig.1.

For 4 nodes, the matrix multiplication is performed on 4 different nodes which are master and worker1, worker2, worker3. Here first matrix is being initialized. Once the matrix initialization is done then the work division is done where the matrix A is divided into 4 equal parts (512*512) as A0, A1, A2, A3 and Matrix B is divided into two equal parts B0 and B1 (512 * 1024) (Fig.2) . Now using A0 and B0 matrix data matrix multiplication is performed in the master node, but before that the data of A1, A2, A3, B0 and B1 matrix is sent to the workers using the MPI_Send() function to perform the matrix multiplication in parallel. The data which is sent to the receiver is received using MPI_Recv() function. Once the data is received then the matrix multiplication is performed. After completion of the multiplication, the obtained results is sent back to the master node using the MPI_Send() function. The execution time for the matrix multiplication is calculated by checking the difference between start time and end time. This is done using the MPI_time() function. The start time is noted after the matrix initialization and the end time is noted after receiving the data from the worker nodes.

Master: A0 * B0	Worker1: A1 * B1
Worker2: A2 * B0	Worker3: A3 * B1

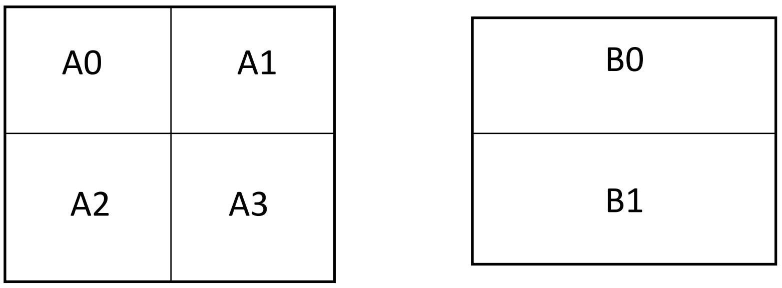


Fig.2.

For 8 nodes, the matrix multiplication is performed on 8 different nodes which are master and worker1, worker2, worker3, worker4, worker5, worker6, worker7. Here first matrix is being initialized. Once the matrix initialization is done then the work division is done where the matrix A is divided into 8 equal parts (256*512) as A0, A1, A2, A3, A4, A5, A6, A7 and Matrix B is divided into two equal parts B0 and B1 (512 * 1024) (Fig.3). Now using A0 and B0 matrix data matrix multiplication is performed in the master node, but before that the data of A1, A2, A3, A4, A5, A6, A7, B0 and B1 matrix is sent to the workers using the MPI_Send() function to perform the matrix multiplication in parallel. The data which is sent to the receiver is received using MPI_Recv() function. Once the data is received then the matrix multiplication is performed. After completion of the multiplication, the obtained results is sent back to the master node using the MPI_Send() function. The execution time for the matrix multiplication is calculated by checking the difference between start time and end time. This is done using the MPI_time() function. The start time is noted after the matrix initialization and the end time is noted after receiving the data from the worker nodes.

Master: A0 * B0	Worker1: A1 * B1
Worker2: A2 * B0	Worker3: A3 * B1
Worker4: A4 * B0	Worker5: A5 * B1
Worker6: A6 * B0	Worker7: A7 * B1

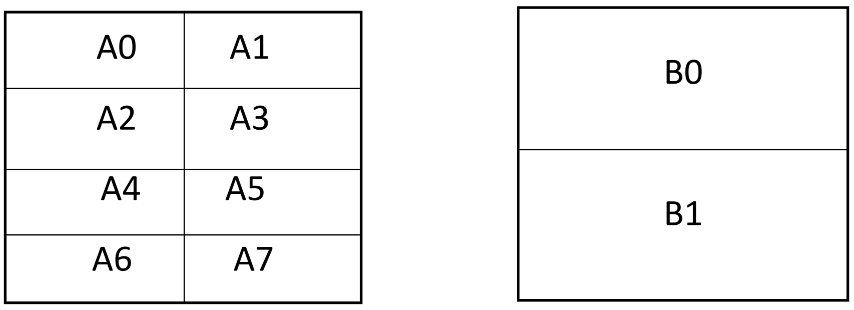


Fig.3.

Measurements:

1. Sequential Matrix Multiplication:

   Execution Time: 61.4 Sec

2. Blocked (2D) Matrix Multiplication (Parallel version):

   1 Node Execution Time: 8.7 Sec

   2 Nodes Execution Time: 4.3 Sec

   4 Nodes Execution Time: 2.1 Sec

   8 Nodes Execution Time: 1.3 Sec

Based on the execution times of the sequential matrix multiplication and MPI-based blocked matrix multiplication, we can say that MPI-based model performed the multiplication faster than the sequential version.

Speedup:

Nodes	Speedup
2	14.2
4	29.2
8	47.2

Table.1.

Task 2: Laplace Approximation

In this task, Laplace approximation is performed for a matrix of size 2048 * 2048. This Laplace approximation task is performed on 1,2,4 and 8 nodes. The work division for the Laplace approximation differs from one node and another.

Initially, MPI is initialized in the main function. Then for 1 node, first the matrix is being initialized and then based on the sequential Laplace approximation algorithm provided in the project manual, the Laplace approximation is performed without any work division since all the work is done on the single node.

For 2 nodes, the Laplace approximation is performed on 2 different nodes which are master and worker. Here main matrix A is being initialized in Master. Once the matrix initialization is done then the work division is done. First a Matrix B is being created to store the data from matrix A to perform the Laplace approximation. Here B matrix is initialized for every node. After this the matrix A (whole matrix including the border values) is divided into two equal parts A0(for master node) and A1(for worker node) and stored in matrix B(Fig.4) using the MPI_Scatter function (Fig.4). The fig.4. shows the data division for size 8*8.

To perform the Laplace approximation, we need the all the surrounding elements. So, the last row of the node 0 and first node of node 1 are repeatedly updated for each iteration. This updation is done using the MPI_Send() and MPI_Recv() functions. After calculating the elements average value using the surrounding elements and relaxation factor, the value obtained here is verified to find the maximum sum. Then using this max sum, the pervious iteration sum is verified to check whether to finish the iterations or not. Since the calculations are done in both the nodes, the maxi value has to be highest value of the nodes. To obtain the maxi value, MPI_Allreduce is used to find the max maxi value. This process is repeated for both the even elements and odd elements for the matrix in 2 nodes.

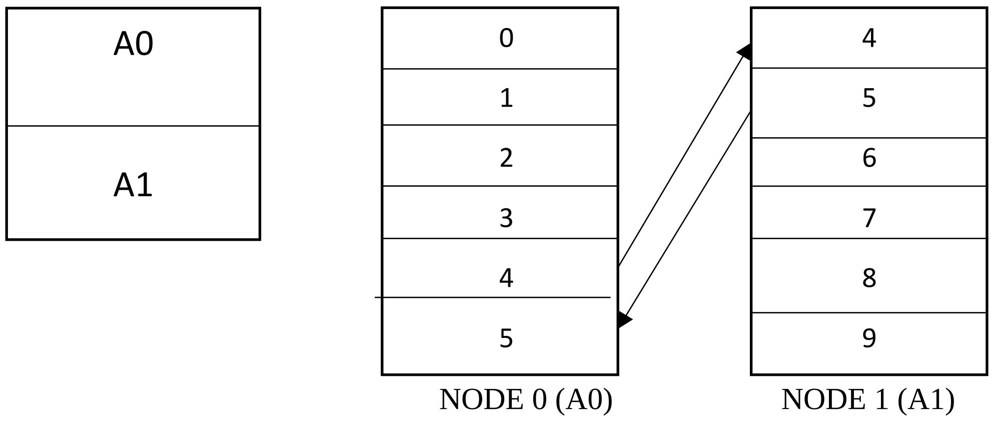
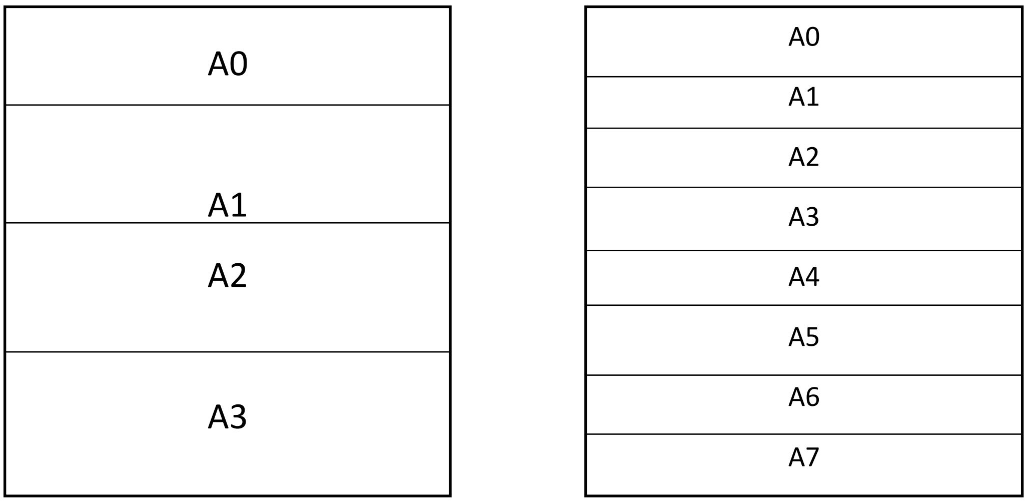


Fig.4.

After completion of the task, the obtained result is sent back to the master node using the MPI_Gather() function and stored in matrix A. The execution time for the Laplace approximation is calculated by checking the difference between start time and end time. This is done using the MPI_time() function. The start time is noted after the matrix initialization and the end time is noted after receiving the data from the worker nodes.

This same process is repeated for the 4 nodes and the 8 nodes also. The matrix division for the nodes here is also done equally(fig.5.).



4 Nodes 8 Nodes

Fig.5.

Measurements:

1. Sequential Laplace Approximation:

Execution Time: 63.4 Sec

2. Parallel Laplace Approximation:

1 Node Execution Time: 65.0 Sec
2 Nodes Execution Time: 32.6 Sec
4 Nodes Execution Time: 21.4 Sec
8 Nodes Execution Time: 19.1 Sec

Based on the execution times of the sequential Laplace Approximation and MPI-based parallel Laplace Approximation, we can say that MPI-based model performed the Laplace Approximation much faster than the sequential version.

Speedup:

Nodes	Speedup
2	1.94
4	2.96
8	3.31

REFERNCE

[1] https://computing.llnl.gov/tutorials/mpi/