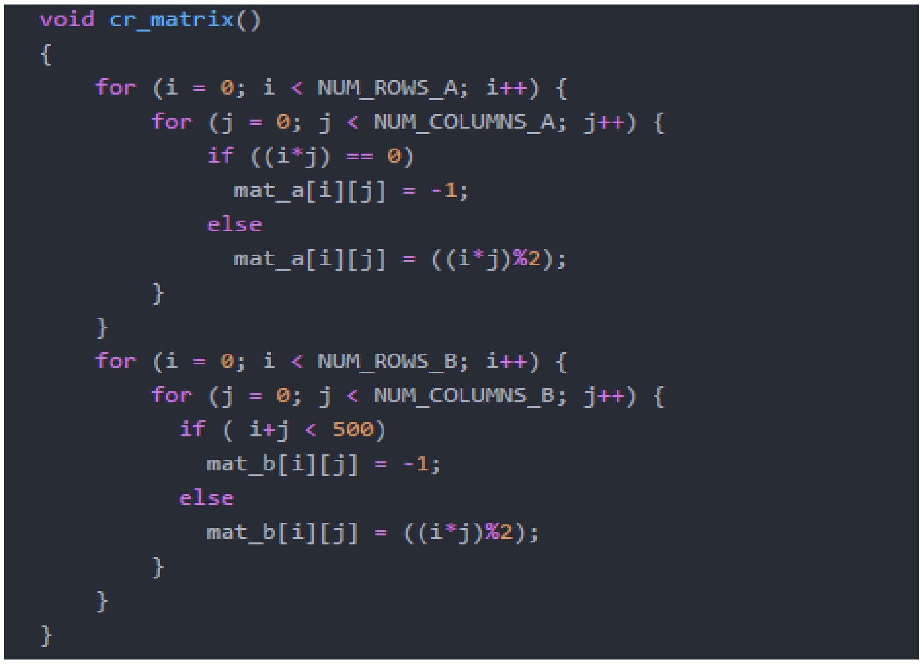
Problem 3.1

This project concerns the design and analysis of algorithm(s) of multiplying a pair of large square matrices A and B on a parallel computer with any interconnection networks available to you. You may consider A and B having $N \times N$ elements on the machine with P processing cores where $A_{ij}, B_{ij} \in (-1, 1]$ random floating-point numbers in the given interval. You may generate such random numbers using any of your favorite method(s).

Report :

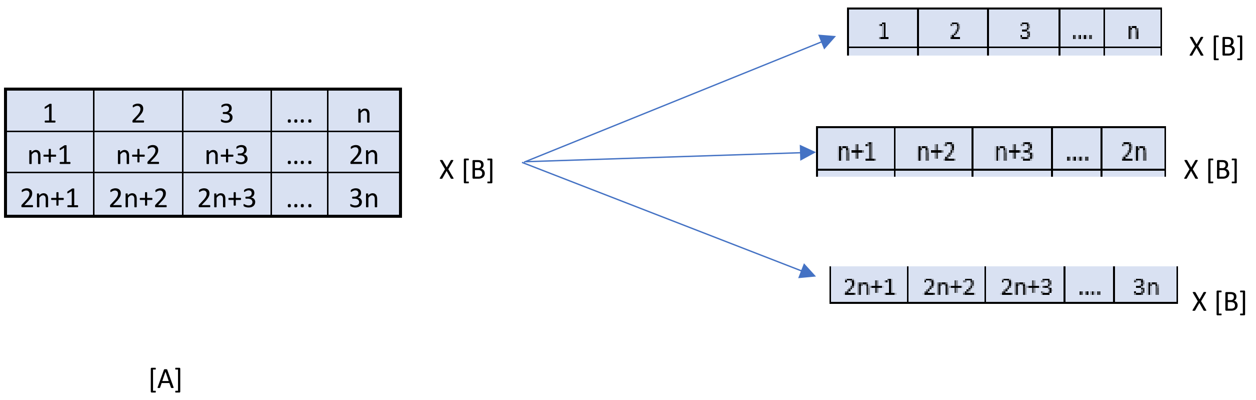
1) An algorithm to multiply 2 matrices was devised and implemented. The algorithm involves the following steps,

   (a) 2 matrices A and B are created. The values of the matrices are $A_{i,j}, B_{i,j} \in (-1 , 1)$ and can be generated using a random function. In my program the matrices are generated using the function "cr_matrix()". Below is a snippet of the code,



```
void cr_matrix()
{
    for (i = 0; i < NUM_ROWS_A; i++) {
        for (j = 0; j < NUM_COLUMNS_A; j++) {
            if ((i*j) == 0)
              mat_a[i][j] = -1;
            else
              mat_a[i][j] = ((i*j)%2);
        }
    }
    for (i = 0; i < NUM_ROWS_B; i++) {
        for (j = 0; j < NUM_COLUMNS_B; j++) {
          if ( i+j < 500)
            mat_b[i][j] = -1;
          else
            mat_b[i][j] = ((i*j)%2);
        }
    }
}
```

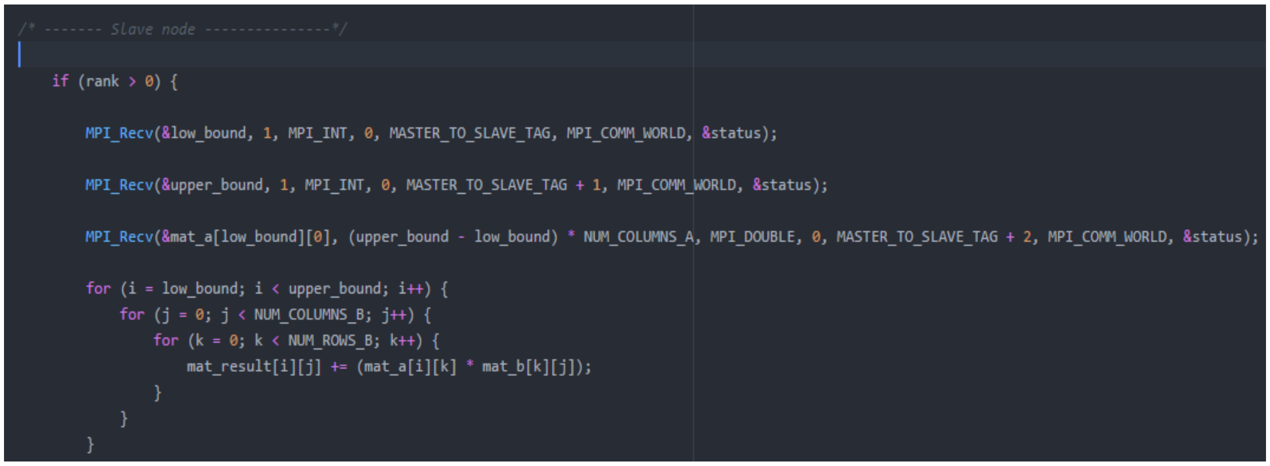
(b) The algorithm involves dividing the first matrix into equal parts and distributing them to different processors. We have 2 tags, one denoting the master and the other denoting the slaves. The master receives both the matrices, it divides the first matrix A into equal chunks of data and sends it to all slave nodes using Isend command. The matrix B is sent to all the slave nodes from the master node using broadcast command.



Master node Slave nodes

(c) All the slave nodes individually process the chunks of data that they have received through Irecv command and the size of the matrix A received depends on the total number of processors and the total number of rows in matrix A. Now each slave node has both matrix B and a part of matrix A which will be multiplied in order. Essentially the whole multiplication works on the principle of block partitioning the matrix A and multiplying with B.

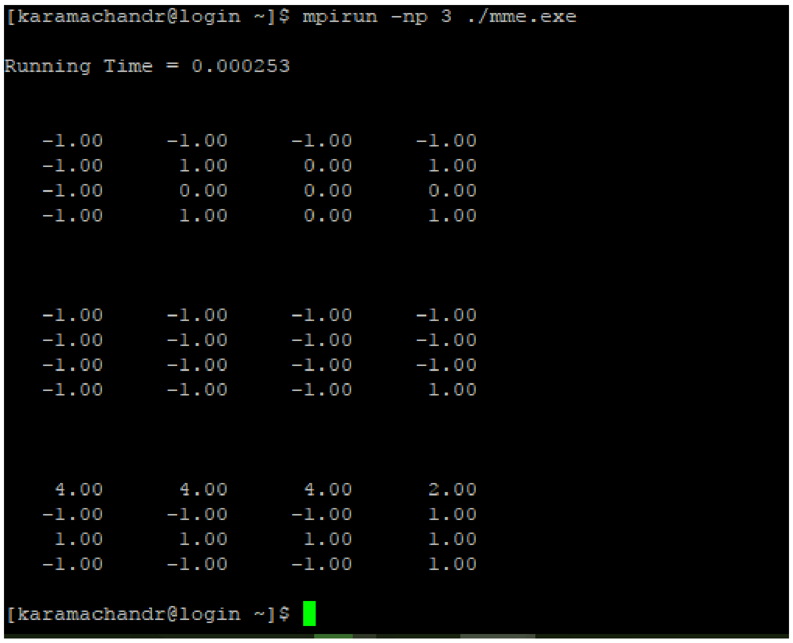
(d) The slave processing nodes individually calculates a segment of the result. The calculation in a slave node works on the principle of multiplication and accumulation(MAC) with in a row. This is kept in track by having the MAC operation within a loop bound by upper and lower bounds.



```
/* ------- Slave node ---------------*/

  if (rank > 0) {

      MPI_Recv(&low_bound, 1, MPI_INT, 0, MASTER_TO_SLAVE_TAG, MPI_COMM_WORLD, &status);

      MPI_Recv(&upper_bound, 1, MPI_INT, 0, MASTER_TO_SLAVE_TAG + 1, MPI_COMM_WORLD, &status);

      MPI_Recv(&mat_a[low_bound][0], (upper_bound - low_bound) * NUM_COLUMNS_A, MPI_DOUBLE, 0, MASTER_TO_SLAVE_TAG + 2, MPI_COMM_WORLD, &status);

      for (i = low_bound; i < upper_bound; i++) {
          for (j = 0; j < NUM_COLUMNS_B; j++) {
              for (k = 0; k < NUM_ROWS_B; k++) {
                  mat_result[i][j] += (mat_a[i][k] * mat_b[k][j]);
              }
          }
      }
```

(e) Finally, the result is sent back to the master node using same Isend Irecv non-blocking commands which gives the result.

2) The algorithm explained in the previous section was coded and implemented on the seawulf supercomputer. It was implemented in C programming language and compiled using the intel compilers for C and MPI. The execution and the results of the multiplication of two 4x4 matrices are shown below,



```
[karamachandr@login ~]$ mpirun -np 3 ./mme.exe

Running Time = 0.000253


   -1.00     -1.00     -1.00     -1.00
   -1.00      1.00      0.00      1.00
   -1.00      0.00      0.00      0.00
   -1.00      1.00      0.00      1.00



   -1.00     -1.00     -1.00     -1.00
   -1.00     -1.00     -1.00     -1.00
   -1.00     -1.00     -1.00     -1.00
   -1.00     -1.00     -1.00      1.00



    4.00      4.00      4.00      2.00
   -1.00     -1.00     -1.00      1.00
    1.00      1.00      1.00      1.00
   -1.00     -1.00     -1.00      1.00

[karamachandr@login ~]$
```

3) The results that were obtained by the above code us summarized using the following table, the table shows how the execution time varies with the processor count and the number of data inputs. All the values of time are represented in seconds and were calculated using the function MPi_Wtime() on MPI.

Execution time	Data input N		
Processors count	1400	2800	5600
14	0.538	7.168	48.692
28	0.506	5.866	46.51
56	0.667	8.536	57.275
224	2.029	32.038	68.712
Serial processor	15.69	182.363	2534.196

A serial program was written to measure the speedup of the system and it is represented as the red row in the above table. The speedup value is calculated using the formula,

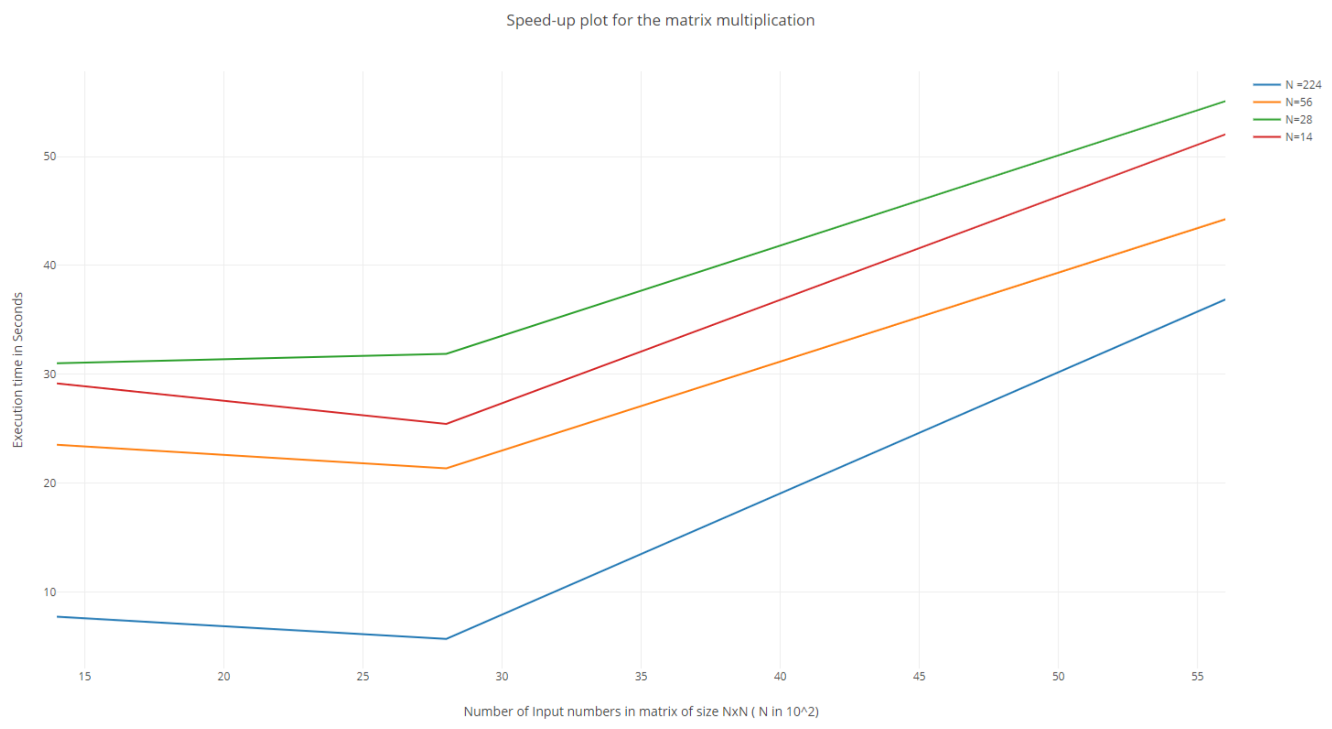
$$Speedup\ ,S(P,N) = \frac{T\ (1,N)}{T(P,N)}$$

Where P is the processor count and N is the size of the input matrix. T(1,N) represents the execution time taken by the program using the serial program T(P,N) represents the execution time taken by the program using processors in parallel.

By applying this formula to the values obtained in the above table we get the following table,

Speed up	Data input N		
Processors count	1400	2800	5600
14	29.163	25.44	52.041
28	31.007	31.881	55.086
56	23.523	21.363	44.242
224	7.732	5.692	36.878

4) The following is the speed curve obtained by plotting all the values from the above table.



Looking at the plot and the readings from the table the following inferences can be draw,

(a) The execution time increases as the size of the input matrix increases. This is in accordance with the assumptions since the number of MAC operations increase with the increase of input matrix.
(b) The efficiency the program decreases for all combination of processor count when size of the matrix is 2800. And later it gradually increases when the matrix size is increased to 5600.
(c) The value of speed up can be found greater than the number of P at several instances. Although it appears that the program handles operation well when the problem scales to a higher number and when the processor count increases, the algorithm performs very poorly. This is due to unnecessary overhead communication with in the processors. The matrix B is initially broadcasted to all the processors even though they use only a small part of it.
(d) The speed up curve is shows that when processor count is 224 the program performs poorly compared to others. The best performance is obtained when processor count is 28. However, for a input matrix of the higher size the value may degrade.
(e) Both programs use different libraries to measure the time. MM_parallel used MPI_Wtime() where as MM_serial used the inbuilt <time.h> library. Hence the results may not be very accurate.

References : https://computing.llnl.gov/tutorials/mpi/

http://mpitutorial.com/tutorials/