Nachiappan C14148805 28th October 2018

# PARALLEL COMPUTING ASSIGNMENT

## INTRODUCTION

The Summa algorithm is used in distributed computing architectures in the form of 2D meshes, for the purpose of matrix multiplication.

The storage requirement remains constant which is one of its main advantages.

Unlike the conventional matrix multiplication algorithm;

```
for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            C[i][j] = 0;
            for (int k = 0; k < N; k++)
            {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
```

The outer product algorithm is employed where the order of execution is changed. Instead of finishing a single element (Eg: c[1][1]) in the first outer for-loop iteration, the outer product algorithm initialized all the elements in the matrix, in the first iteration.

```
for (int k = 0; k < N; k++)
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
            {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
```

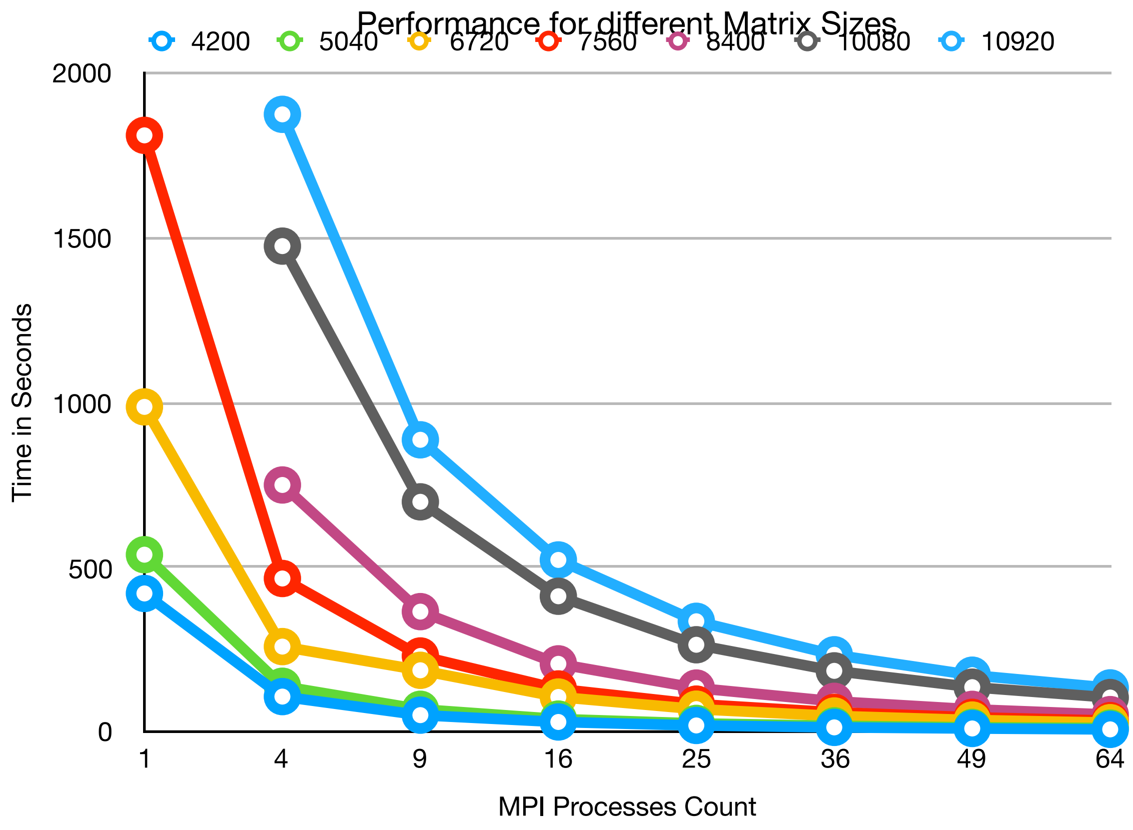
Since each node/mpi process only has a certain block of matrix data, it needs to get the appropriate block to process each step of the result matrix it is responsible for.

For this experiment, the following configuration is used for resource request from the Palmetto Cluster(with variations in the number of CPU and MPIprocs):

```
select=4:ncpus=16:mpiprocs=16:interconnect=fdr:mem=100gb,walltime=8:00
:00
```

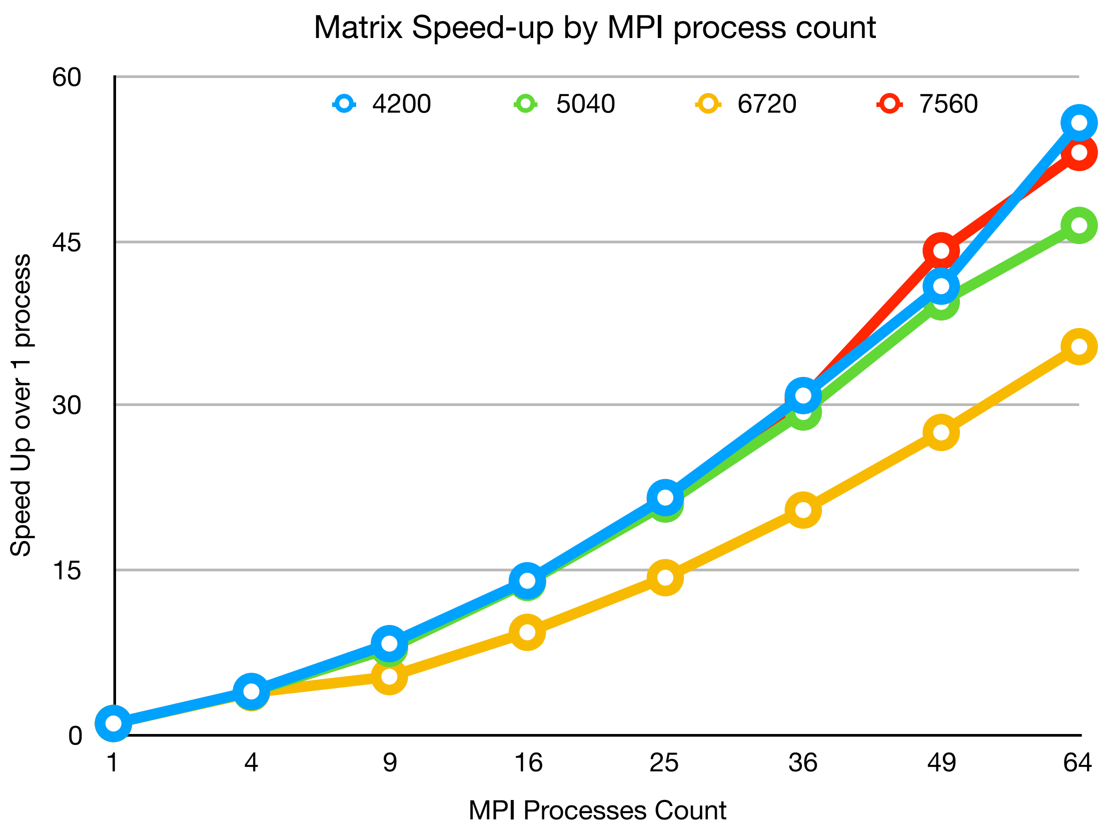
**EXPERIMENT**

A Bidiagonal Matrix is used to determine whether the output for matrix multiplication is correct.
Following this task, the algorithm is used to compute a large matrix of sizes above 4000x4000. The size is chosen as a multiple of 840 which is the LCM for the root of processor sizes we are going to use: 1, 4, 9, 16, 25, 36, 49, 64.



| MPI Processes/ Matrix Size | 1 | 4 | 9 | 16 | 25 | 36 | 49 | 64 |
|---|---|---|---|---|---|---|---|---|
| **4200** | 420.5 | 107.14 | 50.77 | 30 | 19.47 | 13.61 | 10.29 | 7.54 |
| **5040** | 538.25 | 138.58 | 68.56 | 38.84 | 25.63 | 18.31 | 13.65 | 11.6 |
| **6720** | 987.36 | 259.03 | 187.03 | 105.96 | 69.05 | 48.26 | 35.86 | 27.93 |
| **7560** | 1812.36 | 465.97 | 230.57 | 130.40 | 84.34 | 59.28 | 44.10 | 34.16 |
| **8400** |  | 750.09 | 365.28 | 206.64 | 133.10 | 92.79 | 68.38 | 52.39 |
| **10080** |  | 1475.82 | 698.84 | 412 | 264.62 | 185.32 | 135.78 | 104.2 |
| **10920** |  | 1876.19 | 887.34 | 522.35 | 335.92 | 234.19 | 172.21 | 133.61 |

We notice a huge decrease in time taken going from 1 process to 4. The time taken reduces by close to 4 times. Below is the plot of speedup compared to using one process for the matrices.



An interesting point to note is that the speed-up is roughly the same irrespective of the size of the size of the matrices. The only outlier is the 6720 matrix.

The speed-up over using one process is close to linear with increasing MPI Process count. This shows the efficient nature of the SUMMA algorithm. The broadcast operation does not appear to have a significant negative impact on time taken as MPI process size increases.

The time noted here is excluding the synchronization and re-aggregation of the result array on the root node post calculation of the result sub-matrix on each node. The time taken is likely to increase if this additional step is added.