1. The first graph shows the constant size plot of the execution time for 2048x2048 matrix with respect to various number of processors (1,4,16). The size is chosen as in serial implementation it took 70s to execute.

We can see that when there is 1 processor the time in row\_partition is less than canon due to the communication overhead. We see that the difference reduces and canon becomes faster when the number of processors is 4 and when number of processor is 16. We can see that the mkl\_canon algorithm is superfast as its matrix multiplication kernel is superfast and efficient. For the values obtained from the simulation I could see that for smaller matrix sizes canon algorithm starts dominating the row\_partitioning algorithm for lower number of processors also

1. The graph below shows the execution time of the scaled problem size. For row partition algorithm as per isoefficiency analysis we found that Work(W) is proportional to P^2 where P is the number of multiplication. In matrix multiplication W = N^3 where N is the size of the matrix. So for the plot of the row partition algorithm for P = (1,4,16) I have selected N = (1024,2048,4096) respectively which follows the above relations. For P’/P = 4 we get N’N = 2.5 but we approximate it to 2 hence we plotted for size 1024, 2560, 6400.

If we perform the same analysis for the canon algorithm we find that work (W) is proportional to P^1.5 where W = N^3 in matrix multiplication. So we can conclude that matrix size N should be proportional to P^0.5. Hence the size of matrix should double when I increase the number of processor from 1 to 4 and will quadruple when increased from 1 to 16. Hence the matrix size is chosen as 1024,2048,4096 for 1,4,16 processors respectively.

1. The first graph shows the constant size plot of the speed up for 2048x2048 matrix with respect to various number of processors (1,4,16).

As expected as the number of processors increases the speedup of the canon algorithm for constant size problem increases.

1. The following graph shows the plot of efficiency with respect to number of processors. The matrix size is taken to be 2048x2048. Efficiency is measured for the number of processors 4 and 16. Simulation is also done for the number of processor = 1 but calculating the efficiency for it is redundant as we are concerned with efficiency of the speed-up gained by increasing the number of processor. So the plot for P=1 is not shown in the graph.

The efficiency of the canon algorithm increases with the increase in the number of processors whereas that if the row partition slightly decreases.

1. The following graph shows the karp flatt metric for the two algorithm (row partition and canon). The size of matrix this time is taken as 1024x1024. Again the number of processor of interest are 4 and 16 as the formula for ‘e’ will give undefined value for number of process = 1.