OS2 Programming Assignment 1 – Report

Efficient Matrix Squaring

<u>Design Of Program</u>

My program predominantly consists of a few things – Taking Input from a file, Partitioning numbers for each thread, Creating those threads, Evaluating the rows of Square Matrix assigned to each Thread, and printing the final assimilated output into the Out file.

My Program uses a structure calcInfo which is created for each of the individual threads – to store 1. The matrix 2. The rows which it has to evaluate 3. Size of the matrix 4. Number of rows it as to evaluate.

First – I have created an inp.txt file from which the input of N, K and matrix A is drawn. After this – the partitioning of the rows is done.

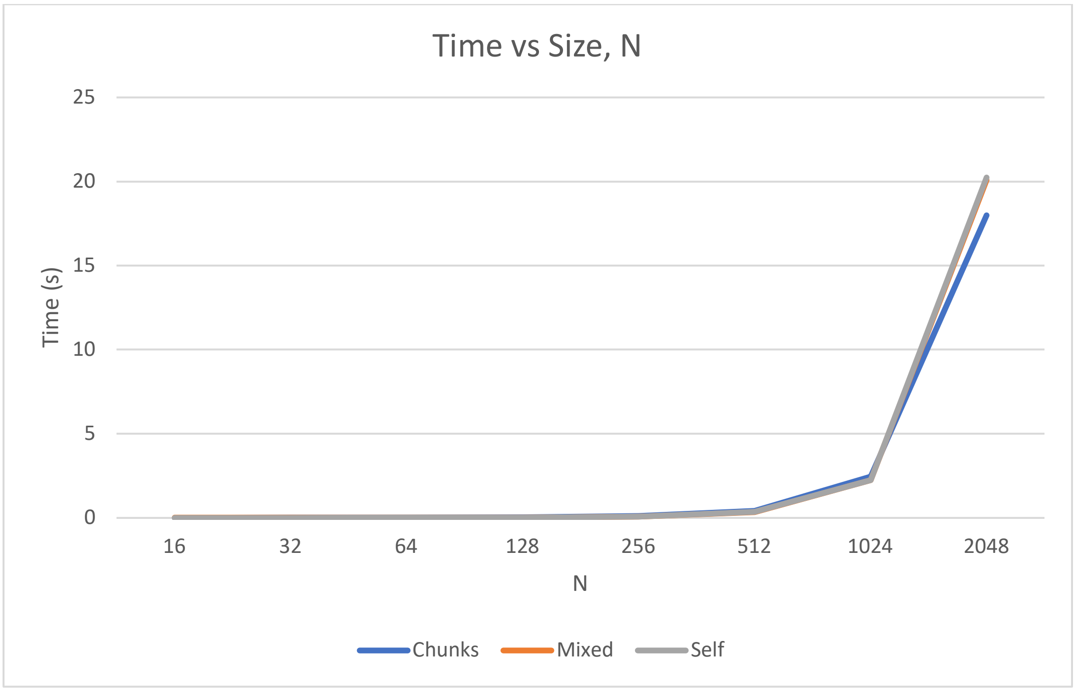
To partition the N number for K threads – I have used 3 methods – Chunk, Mixed, and Self.

1. Chunk – I have chosen the partition size here to be N / K. The first (K-1) threads have each been assigned partition size number of rows.
   If N is not perfectly divisible by K, N / K will be the floor of the actual number – which will cause us to miss a few numbers in the end. Hence for the Kth thread – I have allotted N – (K-1)*(N / K) rows.
2. Mixed – Here, I have sequentially allotted rows one by one to each thread. Once all the threads receive one row, we move back to the first thread and the process continues. We keep doing this until we reach ct == N, which means that all the rows have been distributed to some or the other thread.
3. Self – Here, I have chosen the partition Size to be (N / pow(2, K / 2)). My thought behind this was – mixed clusters. Since each of the inputs in the graph is a power of 2, 2^(K / 2) would be a good size for the chunks. Here – Each thread is sequentially allotted rows, like in mixed – but now, we allot (N / pow(2, K / 2)) rows at each time. Hence – we have made a combination of the two methods – allotting chunks of rows in a mixed fashion. Also, if in case the partition Size becomes < 1 – as in the case when 2^(K / 2) will be larger than N – I have made the partition size default to 1 – and my method lapses to the mixed form.

Once the partitioning is done – I have calculated the required rows of the square matrix that each Thread is allotted – using the simple Matrix Multiplication formula – Cij = Sigma(k = 0 to N-1) Aik + Bkj. Once each thread has done its job – I have first locally stored the computed rows – and then transferred them to the main Matrix C. This C is printed into the outfile.

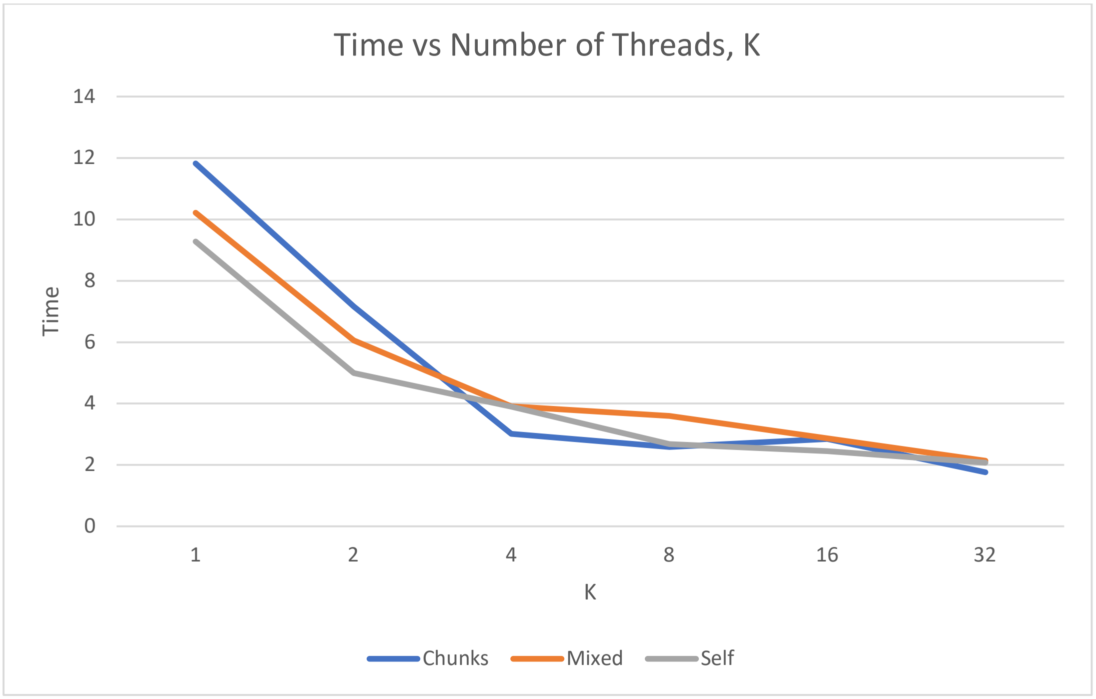
Graphs

Time vs Size



This graph represents how time increases with increasing the dimension of the square matrix. For all the 3 implemented methods, we see a rise in time. For the inputs considered – N is always perfectly divisible by K (and 2^(K / 2)). Hence – all the threads get an equal number of rows. Thus – All the 3 methods require more or less the same time. But – In Mixed and Self Method – The time required to make those partitions is quite high since the partition size is small. Thus – this overhead leads to a slight increase in their time to execute when N reaches 2048. At the end – we see time taken by Chunks < Mixed = Self

Time vs Number of Threads



In this graph – we can see that as the number of threads increases, the time taken to compute the square of the matrix decreases. The reason is that as we divide the work amongst more threads – less work has to be done per thread – and hence the time taken reduces. This pattern is more or less the same amongst all the 3 graphs. The time successively reduces as the number of threads increase. In all the 3 cases – all the 3 methods are given the same number of rows to compute. Thus, there is no significant difference in the time taken by any one particular method. Although initially, we see that the computing Time of Self < Mixed < Chunks.