CS2523
Programming Assignment – 1

DESIGN OF THE PROGRAMS

Generating the inputs

For generating the inputs required for the programs, I have created a program named ***inputGenerator()***;

It depends on the parameter ***n***, upon which it creates a randomly generated ***n*n*** matrix having numbers from 1 to 100 (can be changed).

It prints ***n***, ***k***, and the matrix in order onto the input file to be used by the main programs.

Common things in the main programs

The main programs read the parameters ***n*** and ***k*** from the input file, and then dynamically allocates memory for the two ***n*n*** matrices. Afterwards, they read the values of the matrix from the input file and put it into the 2-D matrix ***A***. Then they create the transpose of ***A*** and store it into the 2-D matrix ***A_trans***.

To store the arguments to be passed into the ***runner()*** function, they create the structure arguments.

The ***dot()*** function gives the dot product of two 1-D arrays. It is used to find the elements of ***A_sq*** by multiplying a row of ***A*** with a column of ***A_trans***.

To implement multithreading, they create an array of threads, with the thread at each index having a unique set of arguments.

The time required to perform the matrix multiplication is counted using the ***chrono*** header file.

Then the output matrix ***A_sq*** is printed onto the output file.

The files are closed and the memory for the 2-D arrays is deallocated and the programs end.

CHUNKS

Each thread receives a chunk (group) of rows to calculate. The chunks are of uniform size ***n/k***.

There are ***k*** chunks and hence each chunk is allotted a thread. So, each thread finds ***n/k*** rows of ***A_sq***.

Therefore, each thread receives rows ***i*n/k*** to ***(i+1)*n/k -1***

where ***i*** goes from 0 to ***k***-1.

The arguments passed to the ***runner()*** function are :

1. ***start***- starting index(row number) of the chunk.
2. ***end*** – ending index(row number) of the chunk.

MIXED

Each thread receives a set of rows of size ***n/k***. In this case, the index of the rows received by a thread differ by the number of threads ***k***,

i.e., each thread receives row number ***i***, ***i+k***, ***i+2*k***, ….. , ***i + (n/k -1)*k***

where ***i*** can go from 0 to ***k***-1.

The arguments passed into the ***runner()*** function are:

1. ***start*** - the start index of the set of rows
2. ***max*** – the max index a row can have+1 (***n***)
3. ***jump*** – the number by which the index of rows jumps within a thread (***k***)

MIXED-CHUNKS

The matrix is divided into $\boldsymbol{k^2}$ chunks, i.e., each chunk has size $\boldsymbol{n/k^2}$. Each thread receives $\boldsymbol{k}$ chunks. These chunks are distributed among the threads using the mixed method described earlier, having a jump size of $\boldsymbol{n/k}$;

Therefore, each thread receives rows $\boldsymbol{i}$, $\boldsymbol{i+1}$, $\boldsymbol{i+2}$, …. , $\boldsymbol{i+(n/k^2 -1)}$;

$\boldsymbol{i+ n/k}$, $\boldsymbol{i+ n/k +1}$,…, $\boldsymbol{i+ n/k + (n/k^2-1)}$;

.

.

.

$\boldsymbol{i+ (k-1)*n/k}$, $\boldsymbol{i+ (k-1)*n/k +1}$, …, $\boldsymbol{i+ (k-1)*n/k + (n/k^2 -1)}$

where $\mathbf{i = 0, n/k^2, 2* n/k^2, …. , (k-1)*n/k^2}$

Special Case: In case $\boldsymbol{k^2}$ is greater than $\boldsymbol{n}$, $\boldsymbol{n/k^2 = 0}$. To handle this, I have hardcoded the chunk size to be 1.

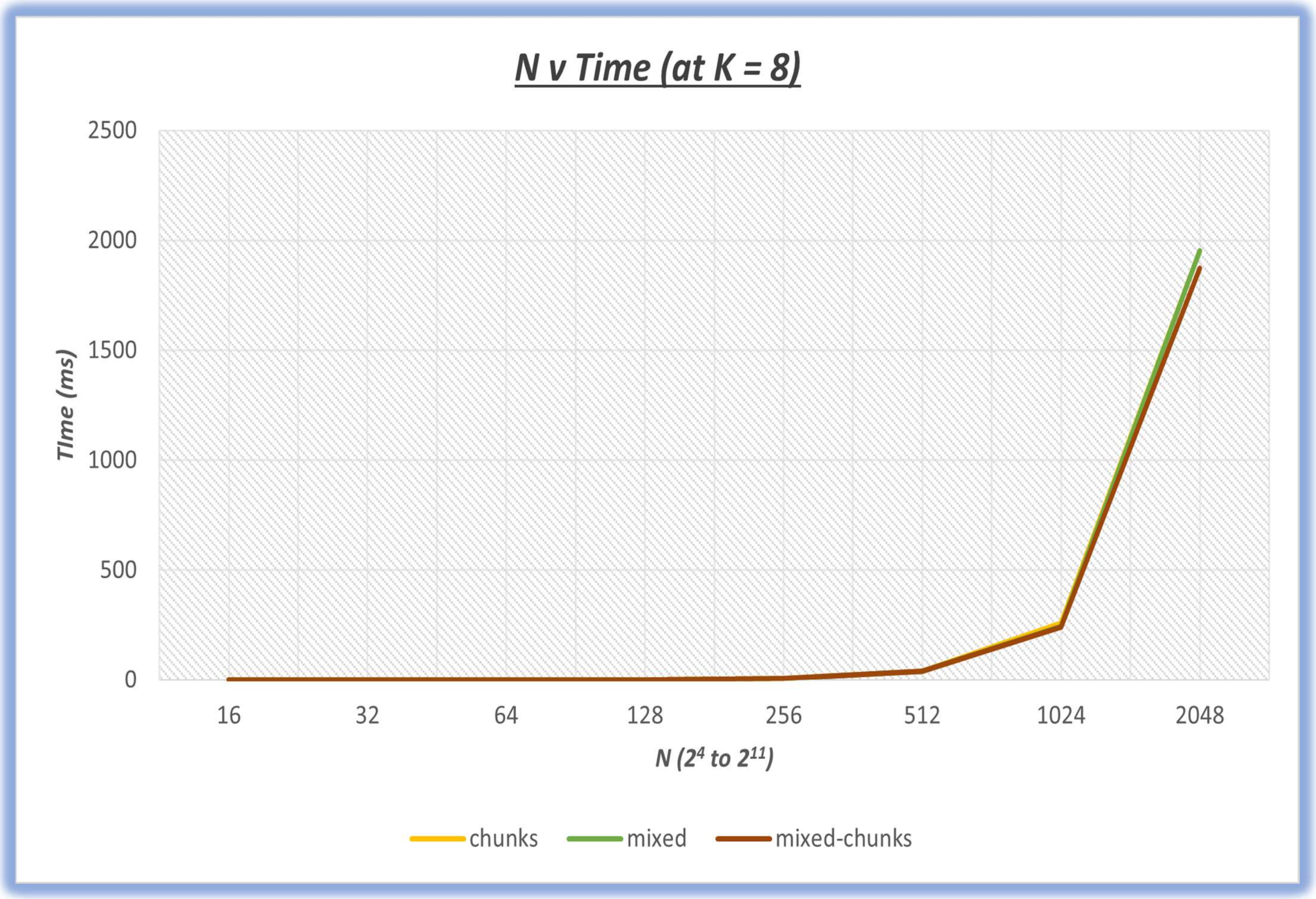
This implies that in cases where $\boldsymbol{n <= k^2}$, the ***MIXED-CHUNKS*** program will be equivalent to the ***MIXED*** program.

The arguments passed into ***the runner()*** function are:

1. ***start*** – the starting index of the chunk received by the thread
2. ***max*** – the max index a row can have+1 (***n***)
3. ***chunk_size*** – the size of the chunks ($\boldsymbol{n/k^2}$)
4. ***jump*** – the number by which the index of rows jumps within a thread (***n/k***)

Performance of the programs

Time vs Size, N

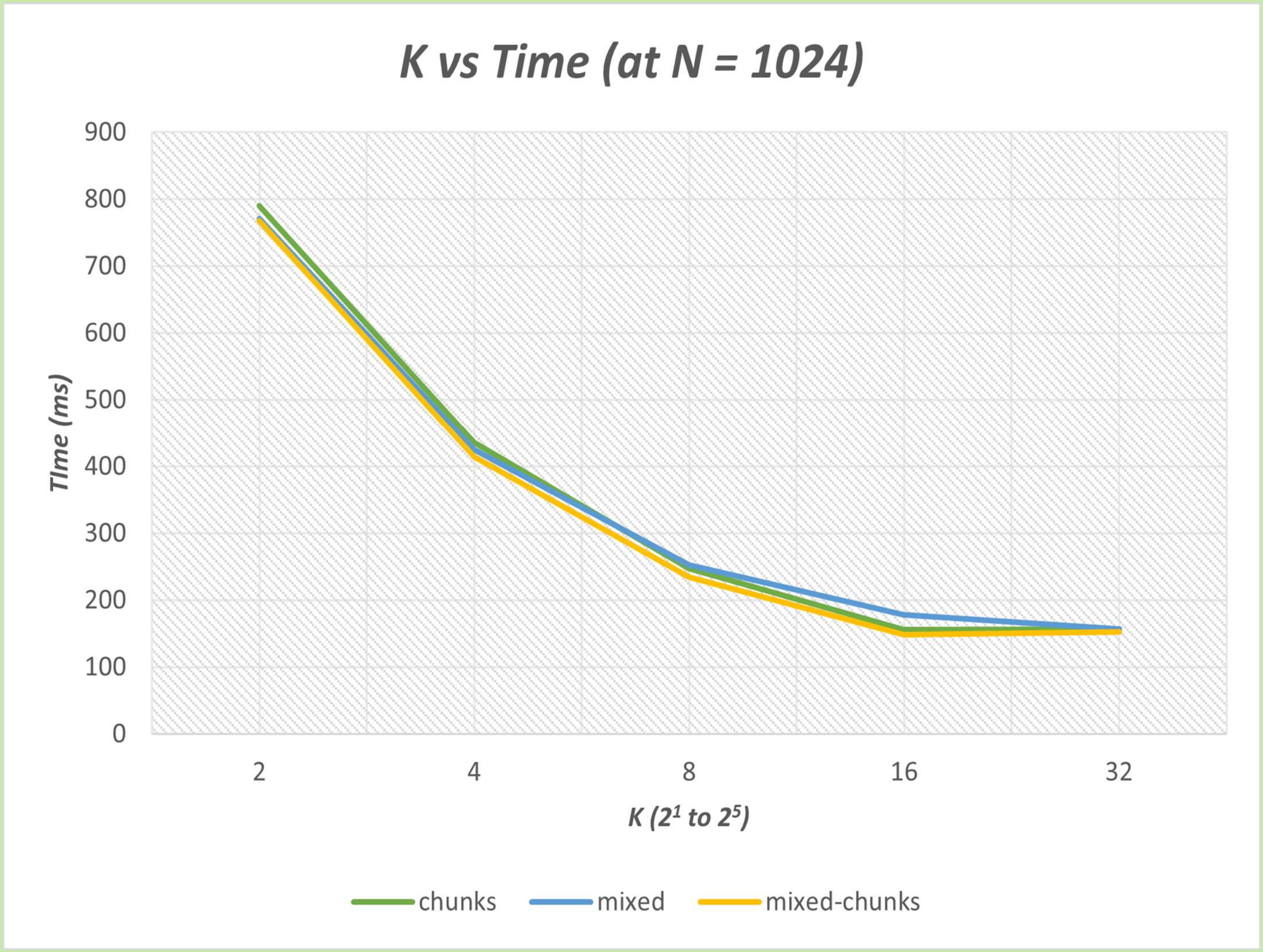


As we can see, as we increase the number of rows of the matrix to be squared, the time taken to do the computations increases exponentially.

The performances of all three methods are roughly the same, although the ***MIXED-CHUNKS*** method gives slightly faster results.

For all three methods, the time taken ranges from about 0.5ms (at ***n = 16***) to about 2sec (at ***n = 2048***).

Time vs Number of threads, K



In the case of the ***k vs time*** graph, we can see that the time taken to compute the squared matrix appears to decrease exponentially with increase in the number of threads.

In this case also, all three methods perform roughly the same, with the ***MIXED-CHUNKS*** method giving slightly better times.

Time taken ranges from around 150ms (at ***k = 32***) to 800ms (at ***k = 2***).