

Assignment 2 Report

Varun Gopal

CO22BTECH11015

The main objective of the program is to develop a multi-threaded solution using the concept of thread affinity to find the square of matrix using the matrix multiplication algorithm.

This program is implemented using the creation of multiple threads and assigning each thread some rows of elements to calculate. The main function distributes all the n rows of the matrix among k threads for each method in a different way. As the threads get created, we first assign some of the threads, specific CPU cores (BT threads among the K threads). This concept is known as Thread affinity and this helps in reducing cache coherence overload and improvising cache utilization. After the completion of all threads for each method, the parent outputs the result of each method and time taken in their respective output files.

Specifications of the System Used:-

Processor:- Intel i5 12500H; Cores:-8; OS:-Ubuntu LTS

Explanation of Code: -

First 'N' (Number of rows), 'K' (Number of threads), 'C' (Number of CPU Cores), 'BT' (Number of Bounded Threads out of K) and the corresponding $N \times N$ matrix (a) are read from the input file{inp.txt}. Now we find the square of the matrix using matrix multiplication. We use a function named dot frequently which computes the dot product of the row and the column passed to it. To do this We first create an array to store the ids of the K threads. These are used by the main function to identify each thread and wait till all the threads finish their execution. This distribution is done by distributing the rows uniformly among the threads. To Make the algorithm run faster we divide the total number of elements to be computed in the C matrix viz. $N \times N$ among K threads. We follow different distribution schemes for different methods:

Distribution for Method 1(Chunk): - Thread 1 gets 1 to N/K rows, Thread 2 gets $N/K+1$ to $2N/K$ and so on. The remaining $N\%K$ rows are allotted to the last thread. More Theoretically, Thread K gets $(K-1)N/K+1$ to N numbers. Where all the arithmetic is performed in terms of integers.

Distribution for Method 2(Mixed): - Thread 1 gets 1, $k+1$, $2*k+1$...rows, Thread 2 gets 2, $k+2$, $2*k+2$ rows and so on. The remaining $N\%K$ rows are allocated to the last thread same as in the case of method 1.

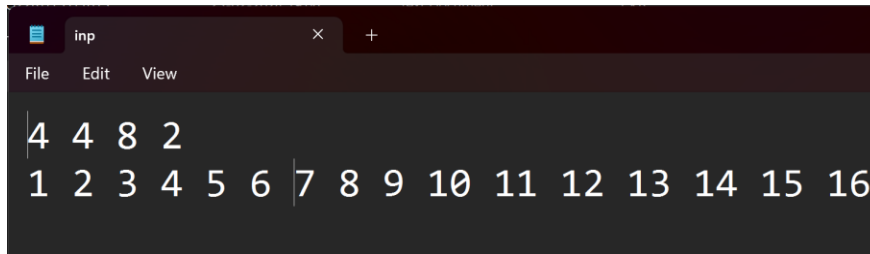
After assigning the rows to each thread, Thread affinity concept is implemented. As soon as a Thread gets created, It sets its Affinity to a particular CPU core. This process takes place by using the `pthread_setaffinity_np()` function and passing the thread id of the respective thread. This process is

repeated for 'BT' threads out of the K threads that are created. These 'BT' threads are uniformly assigned thread affinity over the 'C' CPU Cores.

The input matrix is squared and printed in their respective output files along with the time taken to perform computations for each method on the top.

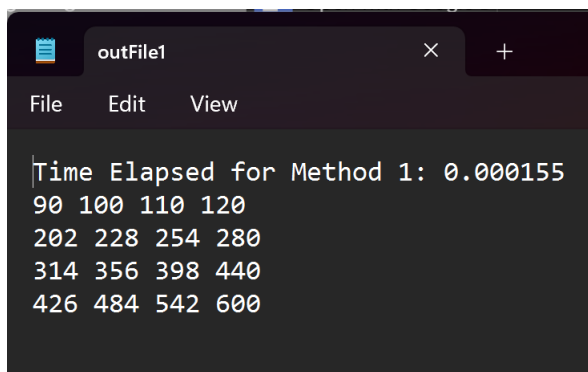
Illustration of the code for N=4 , K=4, C=8,BT=2 is given with matrix being 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 in the row major order.

Input File: inp.txt

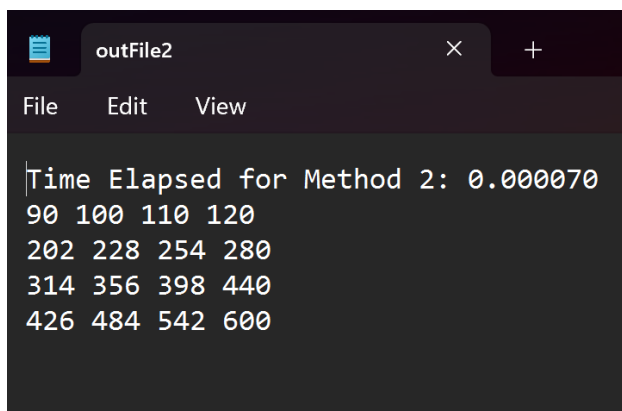


```
inp
File Edit View
4 4 8 2
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

Output Files:- outFile1.txt, outFile2.txt



```
outFile1
File Edit View
Time Elapsed for Method 1: 0.000155
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
```



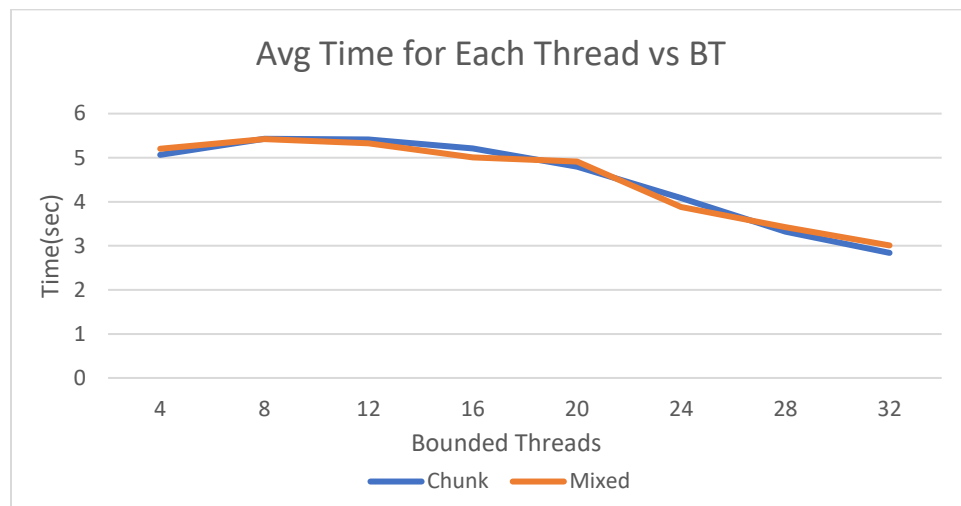
```
outFile2
File Edit View
Time Elapsed for Method 2: 0.000070
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
```

Analysis of Output:

The running times of the program were measured by varying K, BT for each execution using the 2 methods. It is a slight improvement over created multi-threaded program and letting OS to assign cores by load balancing. This method helps to reduce the time slightly by improving cache utilization and doesn't require the threads to load the cache content repeatedly as they get assigned newer cores. Previously the threads were assigned cores by the OS by Load balancing, during which if a thread gets assigned a core different from its previous execution, It has to load the memory (Cache) again.

Below is the analysis of how the running time of the program changes with variations in BT(Number of Bounded Threads) for the 2 methods: -

1.Time vs Number of Bounded Threads, BT:



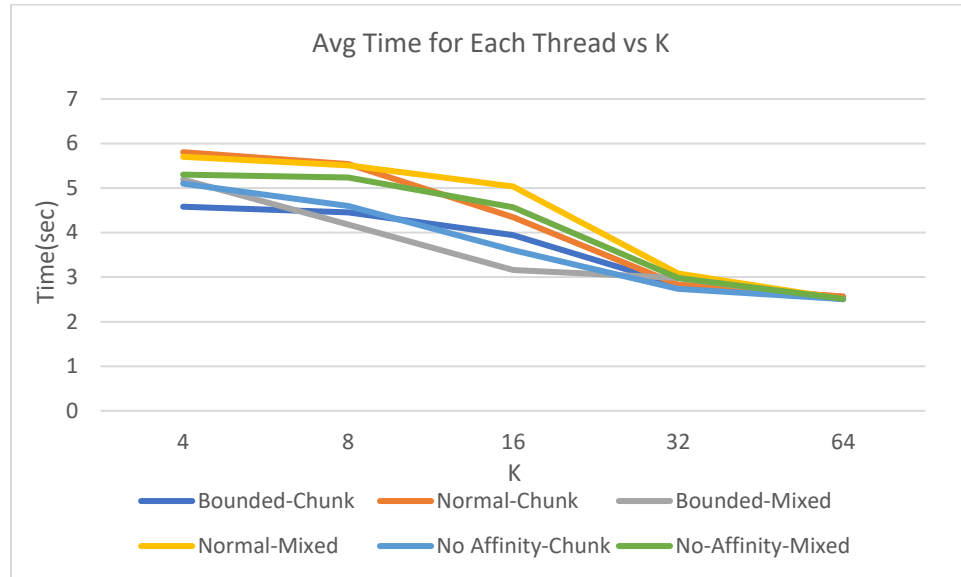
N	K	C	BT	Chunk	Mixed
1024	32	8	4	5.066984	5.199376
1024	32	8	8	5.428814	5.421065
1024	32	8	12	5.412407	5.327143
1024	32	8	16	5.211741	5.007079
1024	32	8	20	4.795032	4.913813
1024	32	8	24	4.081604	3.879744
1024	32	8	28	3.320602	3.424337
1024	32	8	32	2.838515	3.008224

From the above graph, for a fixed number of rows(N=1024), Number of threads(K=32), Cores(C=8), The running times of the program decreases for both the methods as the number of bounded threads increases. This happens due to the fact that threads which assign themselves to a specific core have

slightly lower execution times than the normal ones. As we increase the number of Bounded threads, The effect of thread affinity can be seen more significantly on the execution times.

Method 1 and Method 2 show similar execution times as Number of Bounded threads varies.

2. Avg. Time for each Thread vs Number of Threads, K:



N	K	C	BT	Bounded-Chunk	Normal-Chunk	Bounded-Mixed	Normal-Mixed	No Affinity-Chunk	No-Affinity-Mixed
1024	4	8	2	4.581773	5.806229	5.191175	5.702549	5.099875	5.302272
1024	8	8	4	4.454033	5.539826	4.178923	5.512176	4.598546	5.23501
1024	16	8	8	3.945686	4.345844	3.164719	5.036403	3.608164	4.568892
1024	32	8	16	2.826305	2.827186	2.980168	3.085706	2.740522	2.985045
1024	64	8	32	2.560332	2.56591	2.500812	2.508044	2.507521	2.515364

The above graph shows the average execution time for each thread as the Number of threads(K) varies. At each instant the number of bounded threads is chosen to be K/2. Now It is observed from the graph that, as K (Number of threads) increases, The average execution time for each type of thread decreases. This is obvious due to the fact that as K increases, the degree of parallelization increases.

Moreover, bounded threads seem to have a slightly lower execution time on an average for a given K as they have improved Cache management reducing Cache coherency redundancies.

Summary of the Report:

The Assignment expects us to find the square of the matrix using multithreading with various different ways of distribution of data. As an improvement to the traditional multithreading, Here some threads are assigned some specific CPU core which improves their performance by allowing them to reduce the time taken for caching the data.