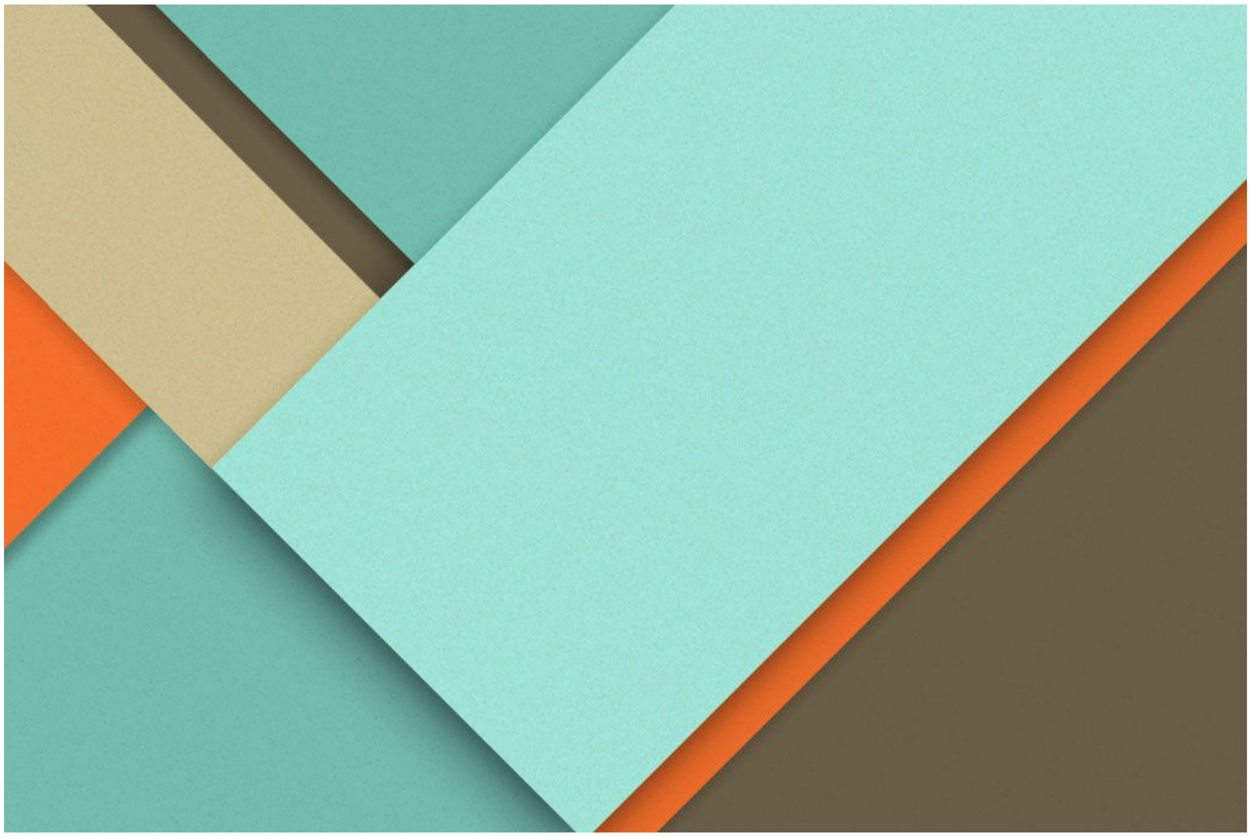


Threads

Name: Mohamed Hesham Ibrahim Alkhaligy

ID: 47

Overview

Implementing two popular algorithms as multi-threaded ones.

Goals

1. Introducing threads concepts and POSIX threads library.
1. Introducing processes and their nesting.

Specifications

I - Matrix Multiplication

It is required to implement two variations of this algorithm:

a. The computation of each element of the output matrix happens in a thread.

b. The computation of each row of the output matrix happens in a thread.

II - Merge Sort

Merge sort is an O (n log n) comparison-based sorting algorithm. It is a divide and conquer algorithm.

It is required to implement it using Pthreads. Each time the list is divided; two threads are created to do merge-sort on each half separately. This step is repeated recursively until each sub-list has only one element.

Code Organization:

The project consists of three classes; “matrix_multiplication” , “merge_sort” and “main”. The “matrix_multiplication” class implements two different variations of matrix multiplication algorithm as multi-threaded ones. The “merge-sort” class implements merge sort algorithm as multi-threaded. While the “main” class is only to initialize the the two classes.

Major Functions:

Class: “matrix_multiplication”

- **“multiplicationInitialize”** : Initializes the basic matrix multiplication functionalities as reading the two matrices form a file,, calling the two functions which implements two variations of the matrix multiplication algorithm as multi-threaded ones,, calculating the time elapsed for each variation and finally writing the result of each variation and the time elapsed in a file .
- **“elementMatrixMultiplication”** : Computes the matrix multiplication given two arrays in which it calls elementMatrixMultiplicationThread function as a thread to compute each element of the output matrix. The function returns the time elapsed to compute the matrix multiplication of the two given matrices.
- **“rowMatrixMultiplication”** : Computes the matrix multiplication given two arrays in which it calls rowMatrixMultiplicationThread function as a thread to compute each row of the output matrix. The function alsor returns the time elapsed to compute the matrix multiplication of the two given matrices.

Performance Comparison:

Element Thread VS Row Thread Computation

Row Thread Computation is faster most of the time because in Element Thread Computation the number of threads is greater than that in case of Row Thread Computation, hence, OS is going to spend all its time

managing and switching between those threads rather than doing the calculations. But if the number of elements is equal to the number of cores (CPUs), Element Thread Computation will be faster than Row Thread Computation. In most cases Row Thread Computation will be faster since the average number of cores is lower than the number of threads.

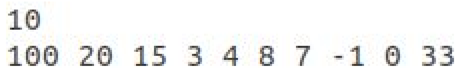
Class: “merge-sort”

- **“mergeSortInitialize”** : Initializes the basic and required functionalities as reading the array to be sorted from a file, sorting the array using merge sort algorithm as multi-threaded and finally printing out the sorted array.
- **“mergeSortThreads”** : Sorts the unsorted array using merge sort algorithm by multithreading approach.
- **“merge_partitions”** : Merges the sorted partitions into one sorted partition

Sample Runs:

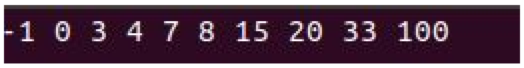
I. Merge Sort:

- File:



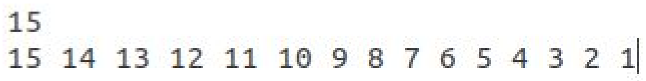
```
10
100 20 15 3 4 8 7 -1 0 33
```

- Terminal:



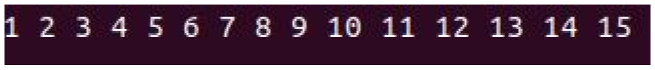
```
-1 0 3 4 7 8 15 20 33 100
```

- File:



```
15
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

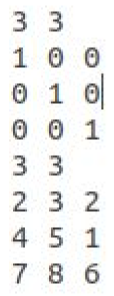
- Terminal:



```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

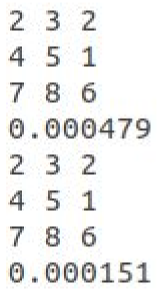
II. Matrix Multiplication:

- Input File:



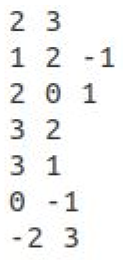
```
3 3
1 0 0
0 1 0
0 0 1
3 3
2 3 2
4 5 1
7 8 6
```

- Output File:



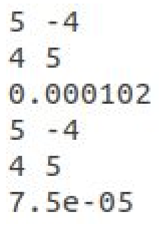
```
2 3 2
4 5 1
7 8 6
0.000479
2 3 2
4 5 1
7 8 6
0.000151
```

- Input File:



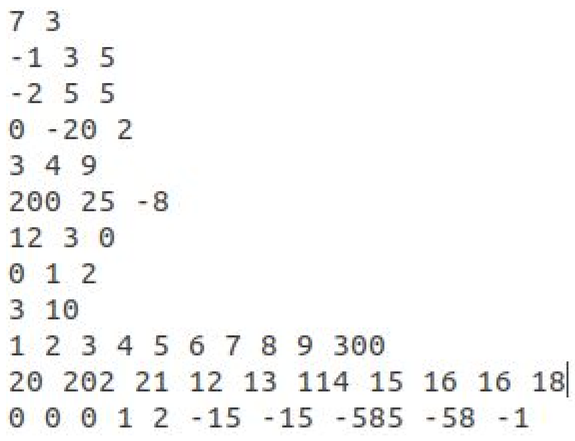
```
2 3
1 2 -1
2 0 1
3 2
3 1
0 -1
-2 3
```

- Output File:



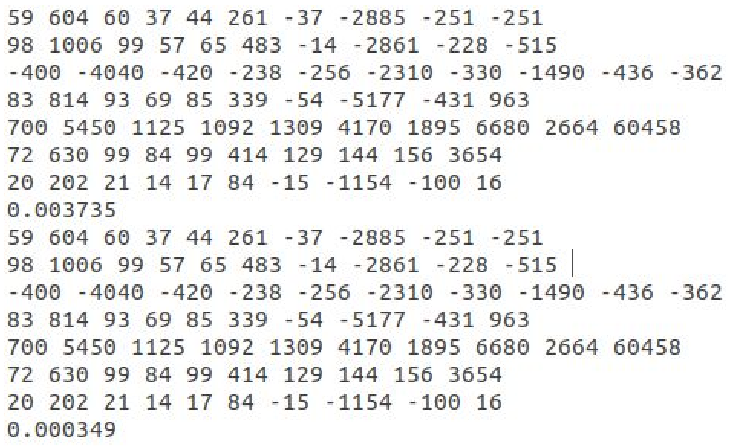
```
5 -4
4 5
0.000102
5 -4
4 5
7.5e-05
```

- Input File:



```
7 3
-1 3 5
-2 5 5
0 -20 2
3 4 9
200 25 -8
12 3 0
0 1 2
3 10
1 2 3 4 5 6 7 8 9 300
20 202 21 12 13 114 15 16 16 18
0 0 0 1 2 -15 -15 -585 -58 -1
```

- Output File:



```
59 604 60 37 44 261 -37 -2885 -251 -251
98 1006 99 57 65 483 -14 -2861 -228 -515
-400 -4040 -420 -238 -256 -2310 -330 -1490 -436 -362
83 814 93 69 85 339 -54 -5177 -431 963
700 5450 1125 1092 1309 4170 1895 6680 2664 60458
72 630 99 84 99 414 129 144 156 3654
20 202 21 14 17 84 -15 -1154 -100 16
0.003735
59 604 60 37 44 261 -37 -2885 -251 -251
98 1006 99 57 65 483 -14 -2861 -228 -515
-400 -4040 -420 -238 -256 -2310 -330 -1490 -436 -362
83 814 93 69 85 339 -54 -5177 -431 963
700 5450 1125 1092 1309 4170 1895 6680 2664 60458
72 630 99 84 99 414 129 144 156 3654
20 202 21 14 17 84 -15 -1154 -100 16
0.000349
```