Lab2

Threads

Name Sohayla Mohammed

ID 32

Matrix Multiplication

Problem Statement

It is required to implement two variations of matrix multiplication:

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.

For both variations,it's required to compute the elapsed time for each of them and compare them.

Objectives

- Understanding threads concepts.

Design and Implementation

Tools & Libraries Used

- Coded with c++.
- Libraries

- pthread

Data Structures

- vector<int>
  - Holds the size of the input matrix.
- vector<vector<int>>
  - Holds the input matrices .

Built in

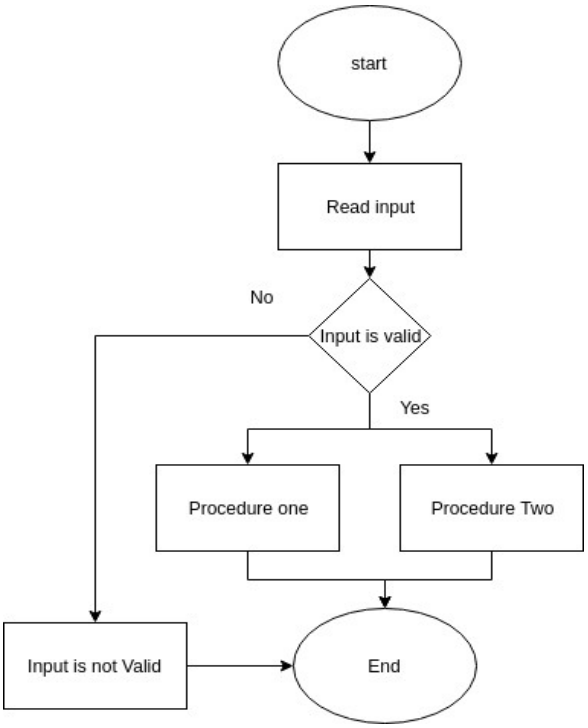
- Strings
  - getline()
- Threads
  - pthread_create(), pthread_join()
- Synchronization
  - mutex

Assumptions

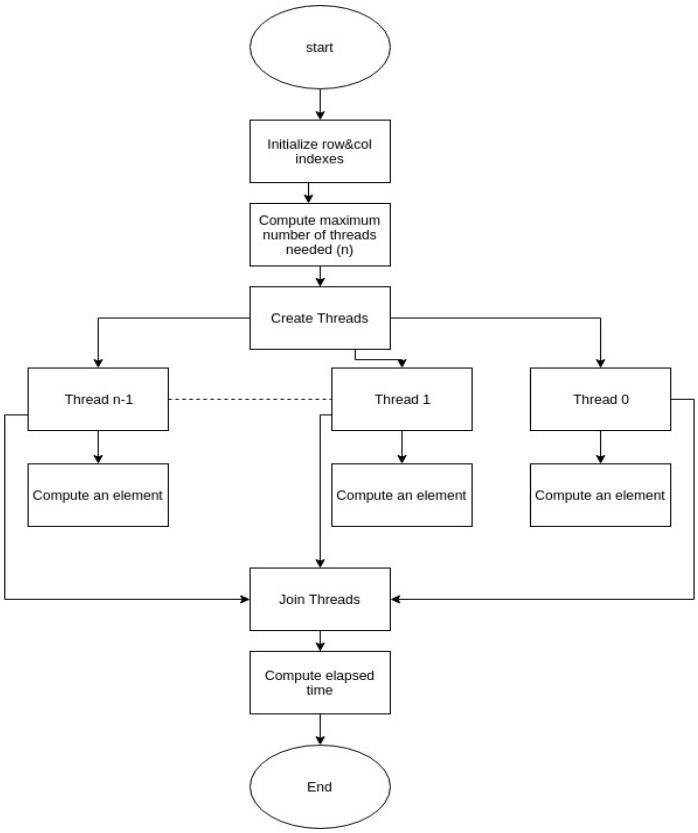
- The input is given in the format shown in the “matrix-readme.txt” file.
- The output file contains
  - result matrices from the two procedures.
  - The elapsed time of each procedure.

Functionality

Overall



- Upon starting the program:
  - Reads data from file “input.txt”
    - If file does not exist the program is terminated.
    - If the format of the file is wrong the program is terminated
    - If the input is not valid for matrix multiplication, the program is terminated.
  - When the input is valid
    - first call is to procedure one which will compute each element in a thread.
    - Second call is to procedure two which will compute each row in a thread.
  - The output matrix and elapsed time from each procedure is written to “output.txt” file.
- In the first procedure :
  - Row and Column indices are initialized to zero.
  - Max number of threads is the multiplication of the number of rows in first matrix and the number of columns in the second matrix.
  - An array of threads is created.
  - Each thread is created to compute an element in the matrix.
  - All threads are joined later.



- An elapsed time is calculated from the beginning until the last thread is joined.

- In the second procedure :
  - The same as first but
    - The maximum number of threads is equal to the number of rows in the first matrix.
    - Each thread computes a row in the matrix.

Functions

- **void** begin(string filename);
  - Begins the execution of the program.
  - Parameters :
    - string filename denotes the name of the input file.
  - Return type : void
- **void** readFileMult(string filename);
  - Reads data from input file to datastructures.
  - Parameters :
    - string filename denotes the name of the input file
- Two Helper functions :
  - Act as a bridge between pthread_create() and the needed function.

- **void** matrixMult1();
  - Represents procedure one, which creates a thread for each element computed.
  - Parameters None.
  - Return type void.

- **void** matrixMult2();
  - Represents procedure twp, which creates a thread for each row computed.
  - Parameters None.
  - Return type void.

- **void*** computeAnElement(**void**);
  - Computes each element from multiplying the two matrices.
  - Parameters NONE.
  - Return type *void (auto)

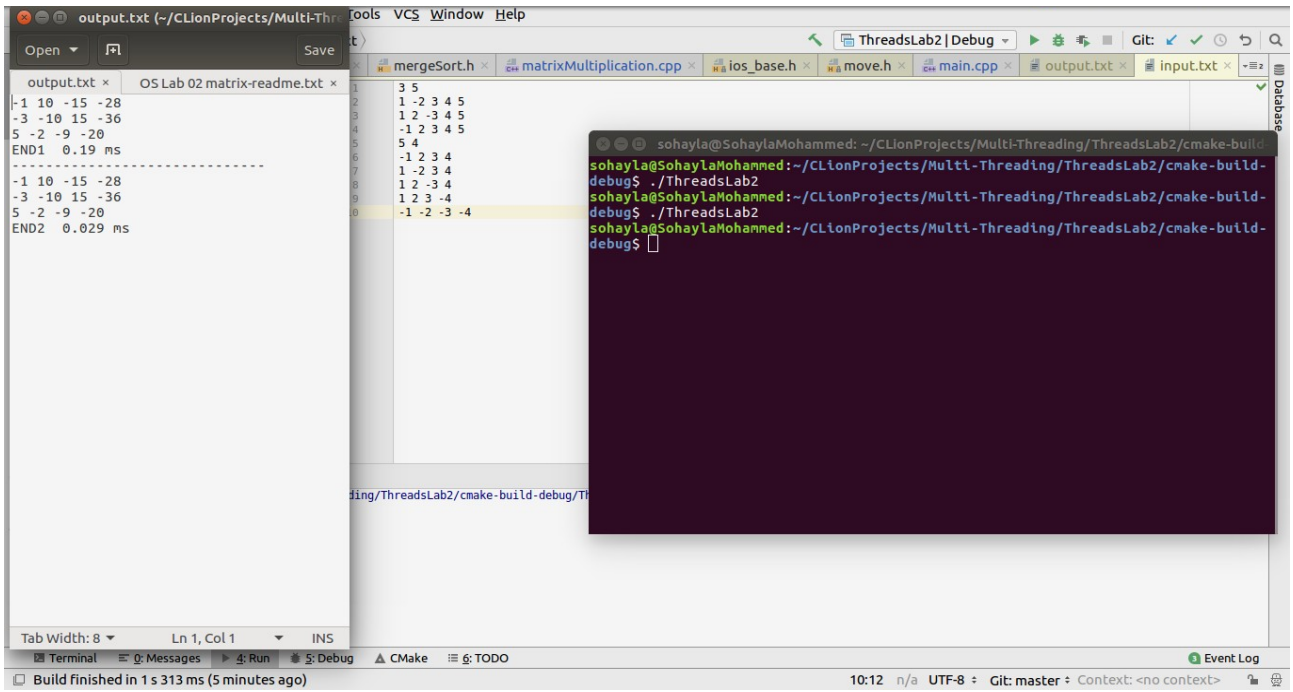
- **void*** computeARow(**void**);
  - Computes each row from multiplying the two matrices.
  - Parameters NONE.

    - Return type *void (auto)
- **void** writeOutputFile();
    - Writes the results into “output.txt” file.
    - Parameters None.
    - Return type void.

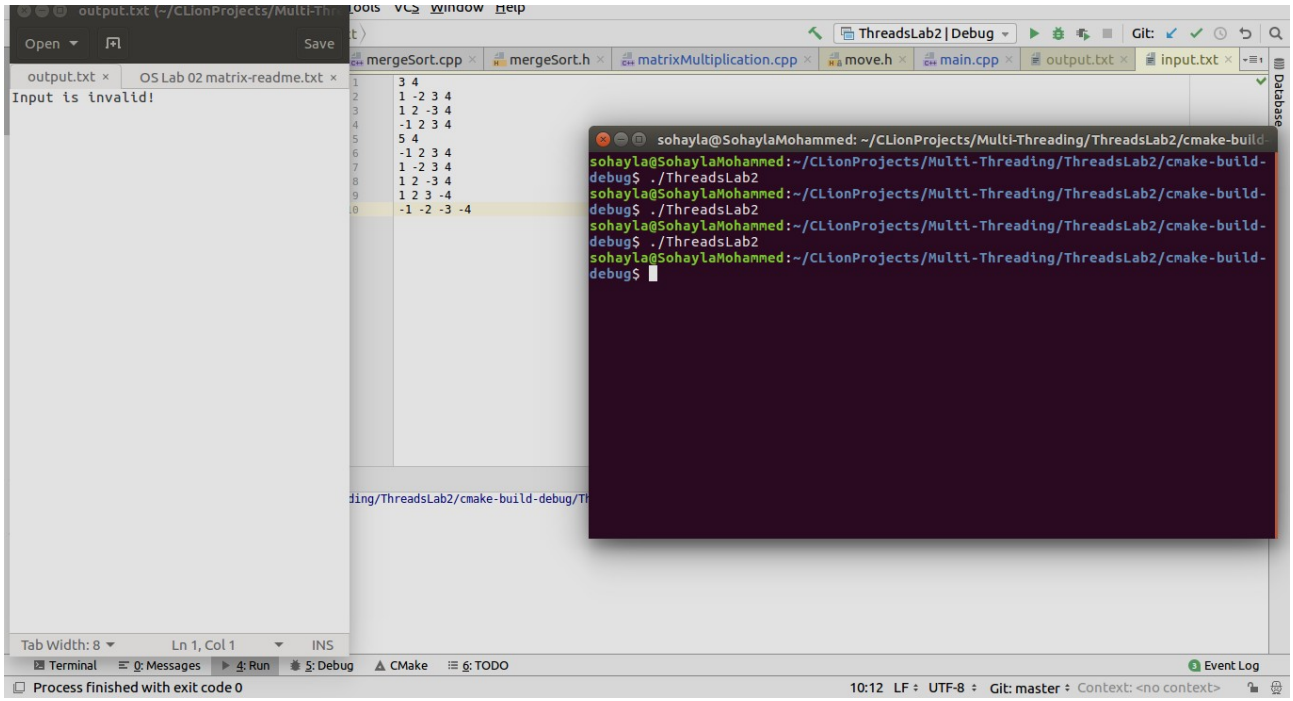
Sample Runs

Input file does not exist, output file :

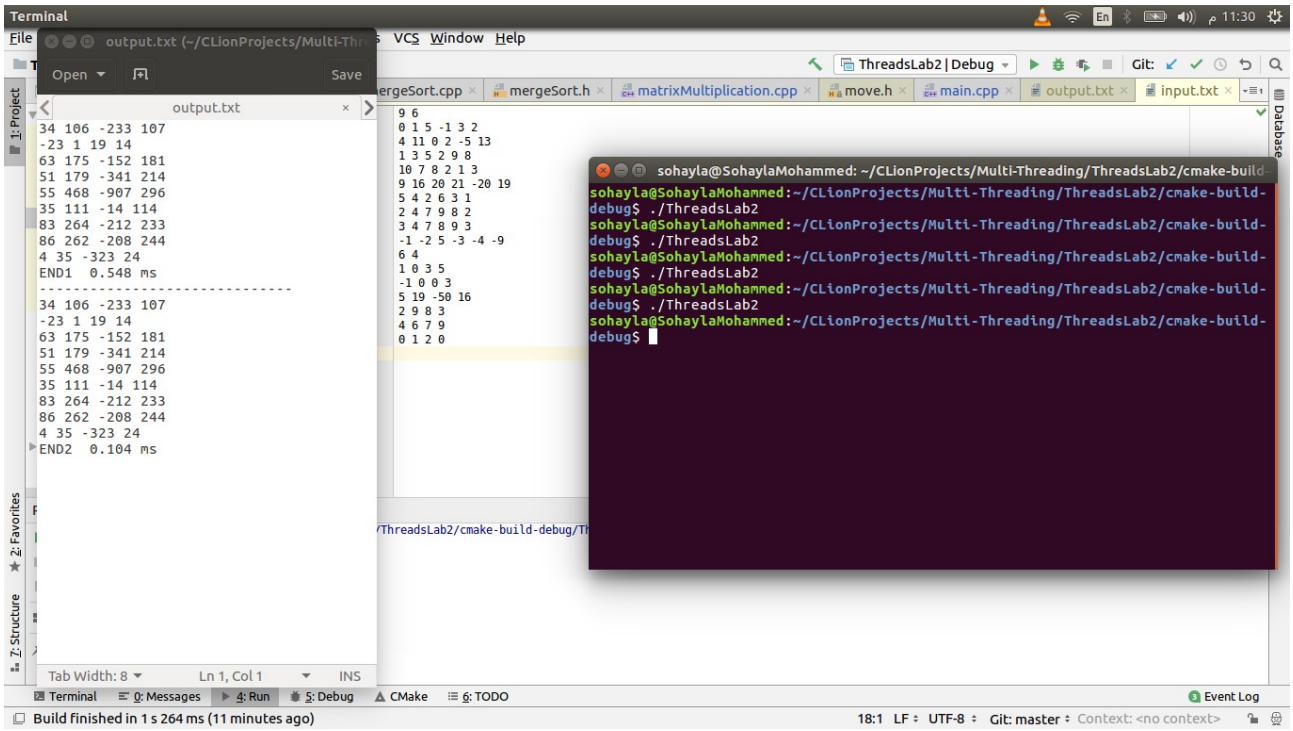
Given input :



Invalid input



Given input



Time Elapse

	Procedure one	Procedure Two
Test 1	0.19	0.029
Test 2	0.548	0.104
Test 3	1.084	0.26
Test 4	0.772	0.166
Test 5	0.424	0.035

Conclusion

- Procedure two takes less time than procedure one.
- As procedure one creates more threads, as each thread creation and termination takes more overhead time.
- Number of threads in the first procedure is equals (m*n) and in the second procedure equals (m).
- If the number of columns in the second matrix is 1, the difference between two procedures time is less.

Merge Sort

Problem Statement

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

It's 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.

Objectives

- Understanding threads concepts.

Design and Implementation

Tools & Libraries Used

- Coded with c++.
- Libraries
  - pthread

Data Structures

- vector<int>
  - Holds the input array.
  - Holds the output array.
- vector<pthreads_t>

- Holds the threads created

Built in

- Strings
  - getline()
- Threads
  - pthread_create(), pthread_join()
- Synchronization
  - mutex

Assumptions

- The input is given in the format shown in the “merge-readme.txt” file.
- The output is printed after the finishing of the program.

Functionality

Overall

- Upon starting the program:
  - Reads data from file “input.txt”
    - If file does not exist the program is terminated.
    - If the format of the file is wrong the program is terminated
  - When the input is valid
    - A thread is created as the parent thread.
    - Then recursively :

        - Each array in a thread is divided into 2 until there exist a thread with only one element
        - Then each thread sort it's array then merged it with it's sibling thread
        - This procedure is done from bottom up.
        - All the threads join.
    - The output array is printed after it's done.

Functions

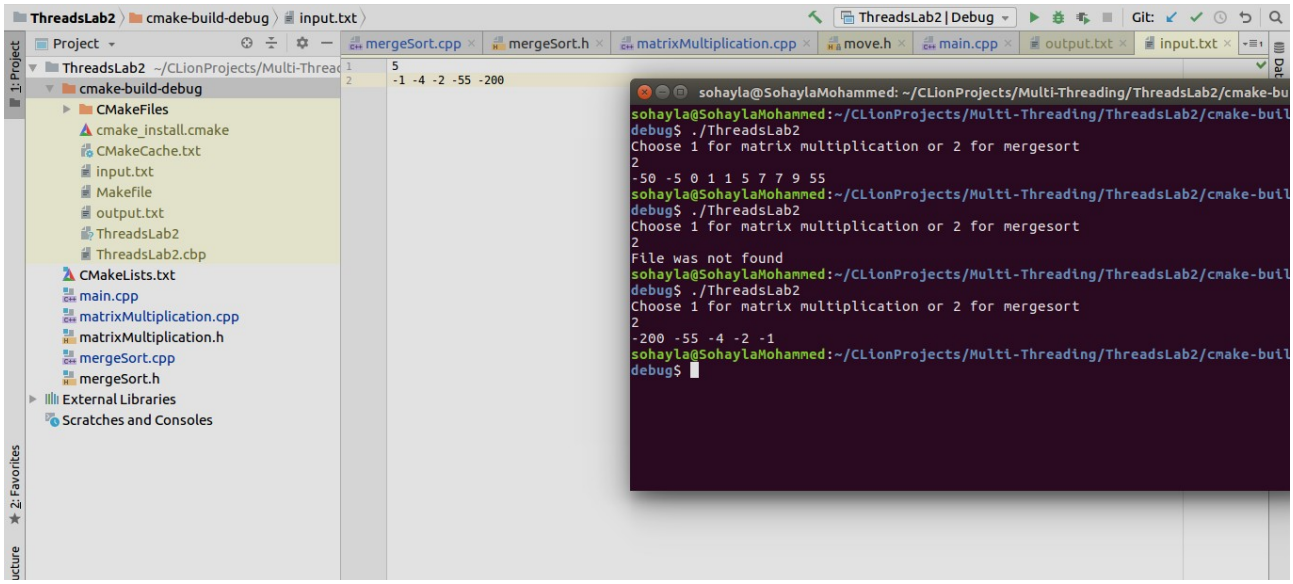
- **void** begin(string filename);
    - Begins the execution of the program.
    - Parameters :
        - string filename denotes the name of the input file.
    - Return type : void
- **void** readFileMerge(string filename);
    - Reads data from input file to datastructures.
    - Parameters :
        - string filename denotes the name of the input file
        - Return type void
- one Helper function :
    - Acts as a bridge between pthread_create() and the needed function.

- **void** mainThread();
    - Creates the parent thread which in return responsible for all the other ones.
    - Parameters NONE
    - Return type void
- **void** * mergeSortF(**void*** arg);
    - Divides the given array "arg" into two arrays, then creates two threads to merge sort each one.
    - Parameters :
        - arg represents the given sub-array in current recursive call.
    - Return type void
- **void** merge(vector<**int**>* a,vector<**int**> l, vector<**int**> r);
    - Merges two sorted arrays l & r into a.
    - Parameters :
        - l → left sorted part of the array
        - r → right sorted part of the array.
        - a → holds the merging result of l & r.

Sample Runs

No input File

Given input



Given input