Report of Assignment 5

-Abhinav Poddar

-EP19BTECH11023

Problem Statement: This assignment aims to implement the graph coloring algorithm in parallel using threads and synchronizes them with the help of locks.

Techniques for Locking Used:

1) **Coarse Locking:** Here there is one common lock CL for all boundary vertices. For coloring any boundary vertex bvx, in one of its partitions, thread Thi has obtained the lock on CL. Assigns a color to bvx in a greedy manner by looking at all the neighbors of bvx. Since Thi has obtained the lock on CL, it does not have to worry about synchronization with any of the threads that are responsible for any of the bvi's neighbors.
2) **Fine- Grain Locking:** Here each vertex has an individual lock also denoted as a fine-grained lock. When the thread Thi wishes to color a boundary vertex bvx, it locks all the neighbors of bvx and bvx as well. It obtains the locks in an increasing (or decreasing) order of vertex ids to avoid deadlocks. After obtaining the locks, it assigns a color to bvx after looking at the colors of vx's neighbors in a greedy manner as

explained above. Again, a vertex that is uncolored (having a color of ⫰) is ignored.

Input Procedure:

I have used adjacency Matrix style to represent the graph in the program. So. the input format is given below:

Number of Partitions   Number of Vertices

// Matrix for the Graph of size V x V.

Libraries Used

1) **#include <iostream>:** It is for basic I/O operations
2) **#include <mutex> :** For using mutex locks in Fine and Coarse Graining.
3) **#include <algorithm>:** For sorting the vector during Fine Graining as it uses array of mutex locks.
4) **#include <fstream>:** For File operations
5) **#include <chrono>:** For Calculating time of execution of a function
6) **#include <set>:** To use Data Structure set.
7) **#include <thread>:** For threads.

Implementation of the Algorithm:

First, I divided the problem set into parts. One creating the greedy algorithm for Graph coloring and second implementing locks to help us in synchronization.

Graph coloring uses a greedy approach in each vertex will check its neighbors and it will take the color which does not overlap with any of its neighbors. To implement this just keep Boolean array called available and assign it as false. Keep a condition that whenever a neighbor is colored change available to true. Indicating the colors is occupied and take the least index of available array and assign it to the vertex. After that reset, the available array and repeat.

For Multithreading only in case of External vertices we must create the lock for synchronization. Other than that, we can use the same algorithm on each partition for coloring.

In case of External Thread:

1) **Coarse Grain Locking**: I used a single lock **mulock** I locked the boundary vertex can implemented the algorithm when the color is allotted then I unlock the vertex.
2) **Fine Grain Locking**: I used an array of locks called **LOCK.** I Locked all the neighbors and vertex itself and keeping their track in Locked Array if one of the locks is not successful then I unlock all the locks and try again by which deadlock can be avoid and locking in increasing manner. After that I implement the algorithm.

Function in Program

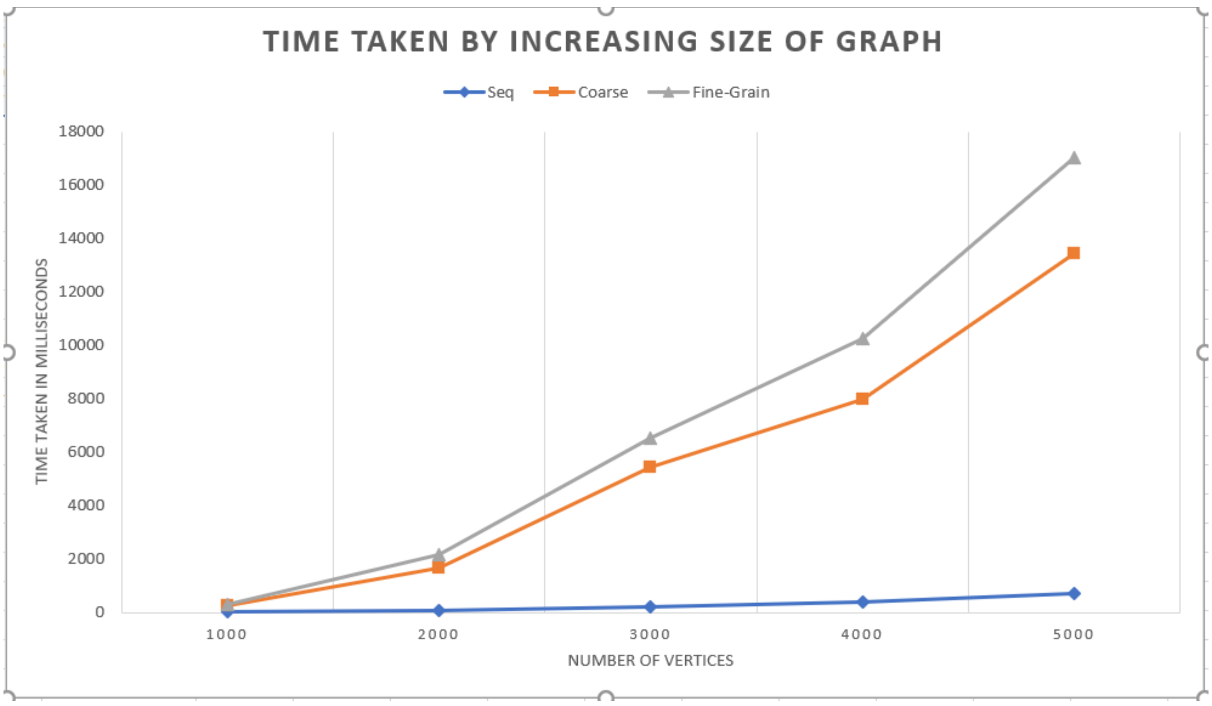
- **Neighbor(vertex):** Returns list of neighbors in sorted order.

- **Find (set, value):** Tell if the value is in set of not
- **Printarray ():** Writes the result in the file
- **TypeVertex(G,set,vertex):** Checks whether vertex is internal or external.
- **Partition(part,vertices):** Divide's vertices into partitions.
- **CoarseLocking(G,s):** For using Coarse Technique.
- **FineLocking(G,s):** For using FineGrain Technique.
- **Seq(G):** Sequential Greedy Algorithm for Coloring.

  I used 900 vertices only because my laptop starts to getting lagging and copy matrix to file becomes very difficult it is only for Plot 3 and Plot 4.

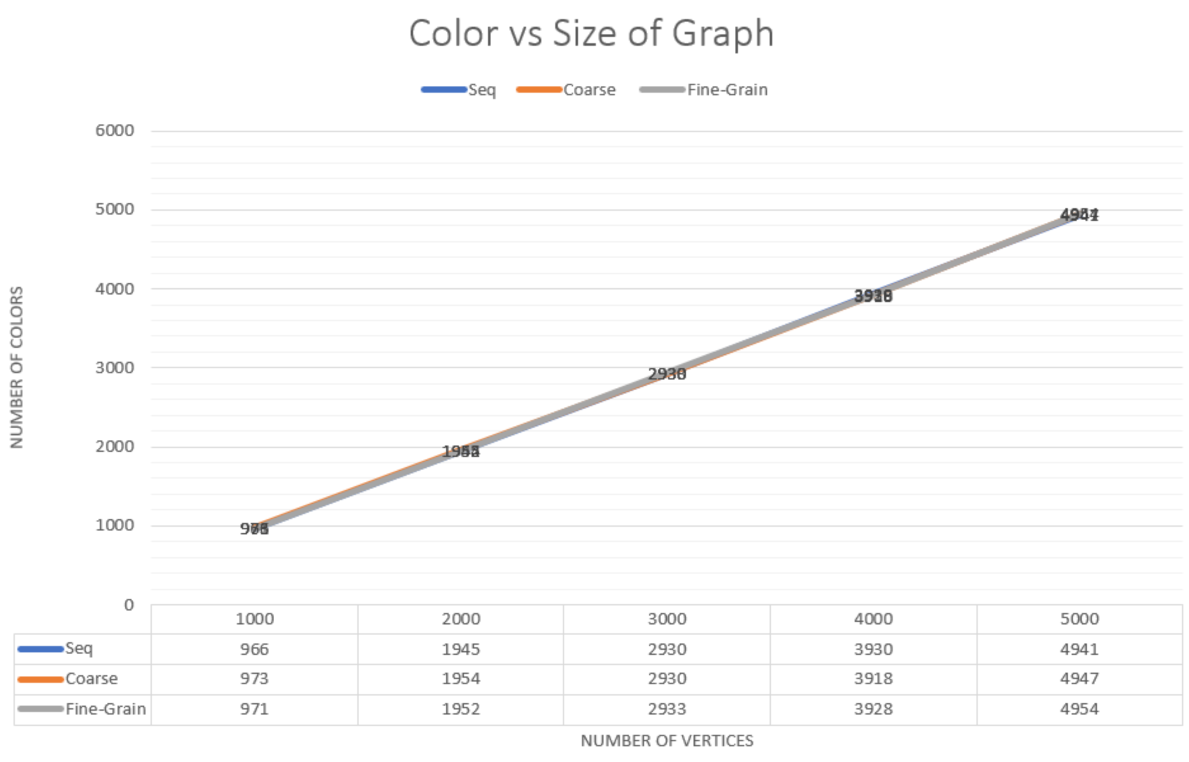
Respective Graphs of the Algorithm

- **Plot 1**



1) As size in increase also execution also increases but this drastic increase is due to poor optimization in synchronization having large number of locks.

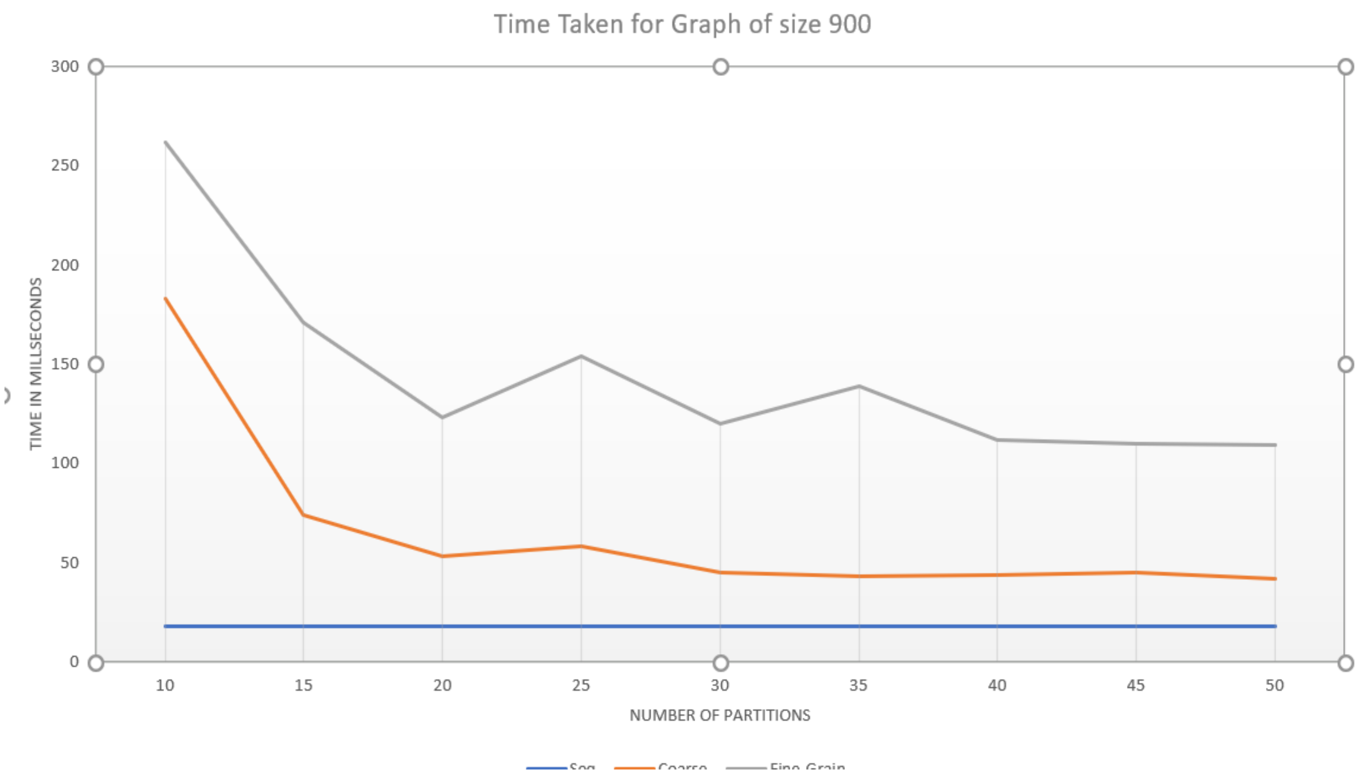
- **Plot 2**



	1000	2000	3000	4000	5000
Seq	966	1945	2930	3930	4941
Coarse	973	1954	2930	3918	4947
Fine-Grain	971	1952	2933	3928	4954

1) Number of Color is almost as we increase the size of graph 2
2) It follows linear relationship that why data set is provided.

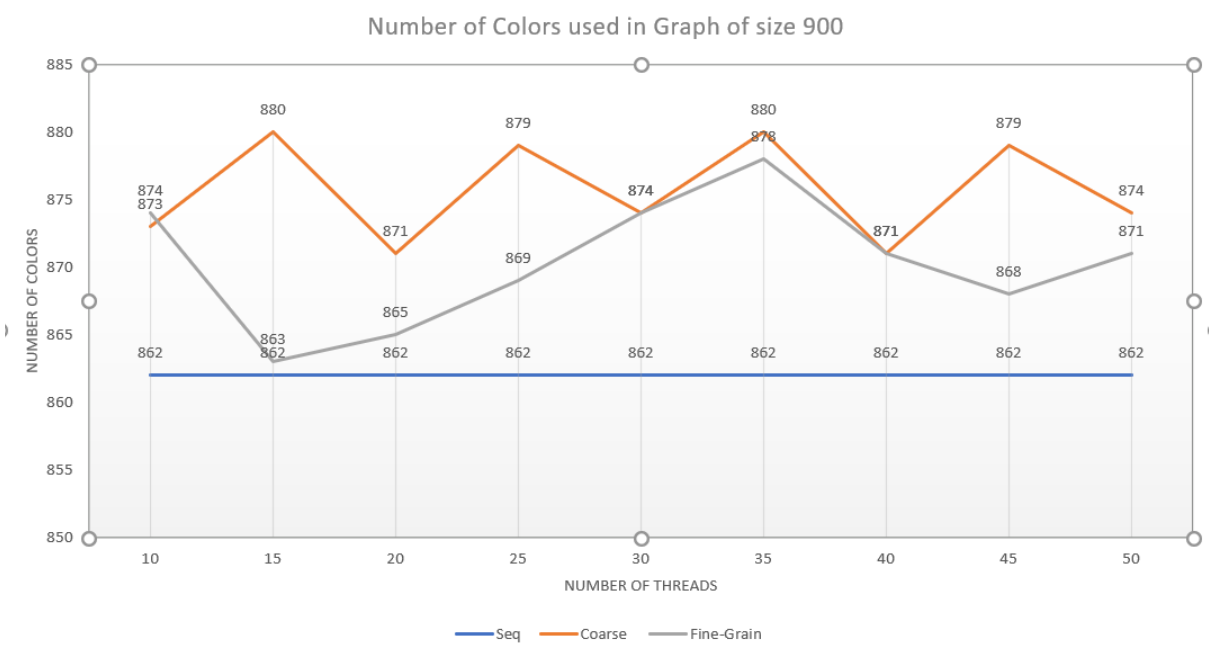
- **Plot3**



1) As we can see as threads increases the time of execution decreases.

2) Coarse technique takes less time but rate of decreasing for Fine Grain is more.

- **Plot 4**



1)Fine grain came nearer to sequential execution but it has high variability.

Observation:

1) Use of recursive_mutex is recommended as normal lock is fast and cause problem. In which output on console will be (Illegal Instruction).
2) Use of set is better than vector for storing partitions as search time in set is O(1) which can optimize the process.
3) Program in Fine Grain can be more optimize if we only lock neighbors which are external vertices rather than locking all of them.
4) Kindly wait during execution for higher size of Graphs.
5) For plot 1 and 2 number of threads where 10.