

**Note :** I used both adjacency matrix and adjacency list to improve efficiency  
**Common algorithm for coloring a particular vertex(v) for both course grained and fine grained and implementation details**

**This algorithm goal is to color a vertex v**

We will have local array named available(`available[noofvertices]`), this array is to keep track of whether a particular color is available is or not and we will have array named color(`color[noofvertices]`), this array will tell us a color for vertex for suppose if we want Color of vertex 2, then `color[2]` will be color of vertex 2,

**Note: the colors will vary from 1 to no of vertices, because in worst case we may color every vertex a different color**

Step 1) we will make all colors as available

Step 2) for all vertices z, if there is an edge between vertex v and vertex z then we will make the Color for which vertex z is assigned as unavailable

Step 3) now we will loop through an available array from starting location i.e from  $i = 0$  to  $i = \text{no of vertices}$ , whenever color is available we will stop looping, and we will assign that color to vertex z

**Algorithm for coarse lock and implementation details**

Every thread we receive its partition of vertices in the form of list,

**Algorithm for checking whether a vertex(v) is internal or not:**

Set `count` to be 0

For all vertices z in the partition of threads, if there is an edge between vertex v and z

Then increment `count` by 1

we will have an array named `edges` which is declared globally to keep track of number of neighbours of vertex v

, `edges[v]` will give the total number of neighbors of vertex v

If that `count` is equal to `edges[v]`

Then vertex v is internal

Else

Vertex v is external

**Note: we will have semaphore named `mutex` which is declared globally, and it is initialized to 1 by using `sem_init(&mutex, 0, 1)`**

We will go through every vertex in list, if that vertex is internal vertex then we will

directly color that vertex using above algorithm, if that vertex is external then we will

Keep a common lock (by using `sem_wait(&mutex)`) and we will color a vertex using above algorithm, after coloring a vertex we will unlock (by using `sem_post(&mutex)`)

## Algorithm for fine grained lock and implementation details

Every thread we receive its partition of vertices in the form of `list`,

**Note: we will have array of semaphores named `lock[no of vertices]` which is declared globally, and all are initialized to 1 by using `sem_init(&lock[i],0,1)`**

We will go through every vertex(`v`) in `list`,

if that vertex(`v`) is internal vertex then we will

    directly color that vertex using above algorithm,

if that vertex(`v`) is external then

    We will go through adjacency list of vertex `v`,

    for every vertex `z` in adjacency list of vertex `v` we will keep a lock by using

`sem_wait(&lock[z])`

    Color a vertex `v`

    for every vertex `z` in adjacency list of vertex `v` we will unlock by using

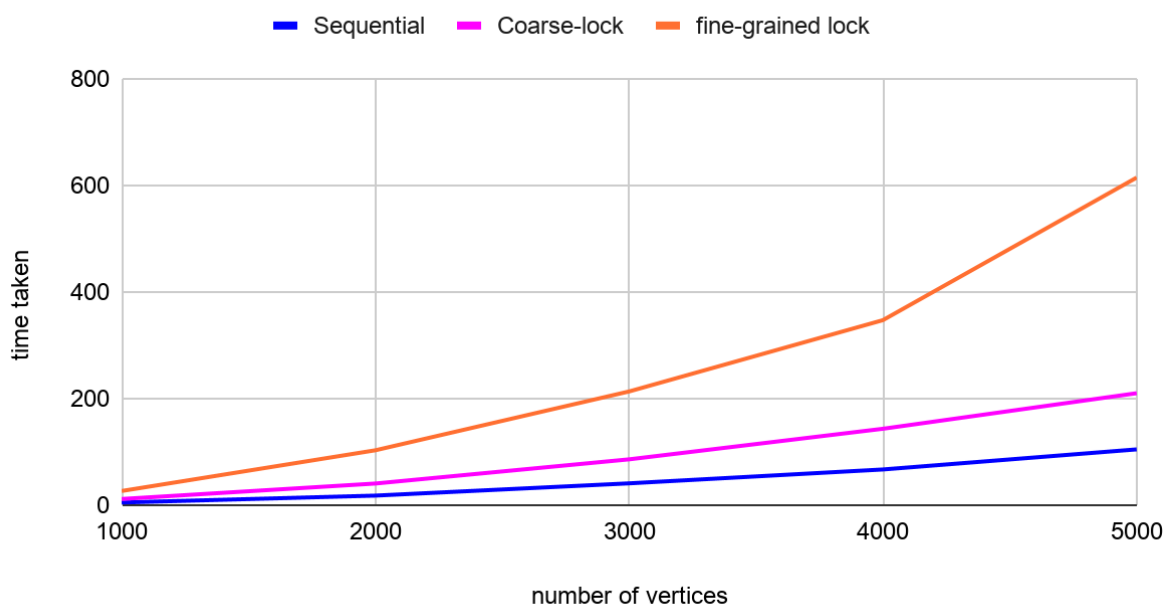
`sem_post(&lock[z])`

**Note: in adjacency list for every vertex `v`, we will keep that list in decreasing order and we will also include that vertex `v` in list**

For all graphs i have taken number of vertices in the range 1000 to 5000 ,because for  $10^4$  i am getting segmentation fault

### Graph 1

Time taken vs number of vertices



If there are too much less internal vertices then there will be no use of coarse lock ,sequential will perform better than coarse lock because we are having common lock  
If we are having less internal vertices then coarse lock algorithm will be also sequential  
Because one thread has to wait till other finishes coloring ,we will have also extra Overhead in creating threads in coarse-lock

So, finally if there are very less number of internal vertices then sequential will perform better Than coarse lock

This number of internal vertices will depend on the input graph and the way we are partitioning, this partitioning will be random

In the graph also sequential wins over coarse-lock this is due to number of internal Vertices

If the graph is connected or if we have very less number of disconnected graphs, then

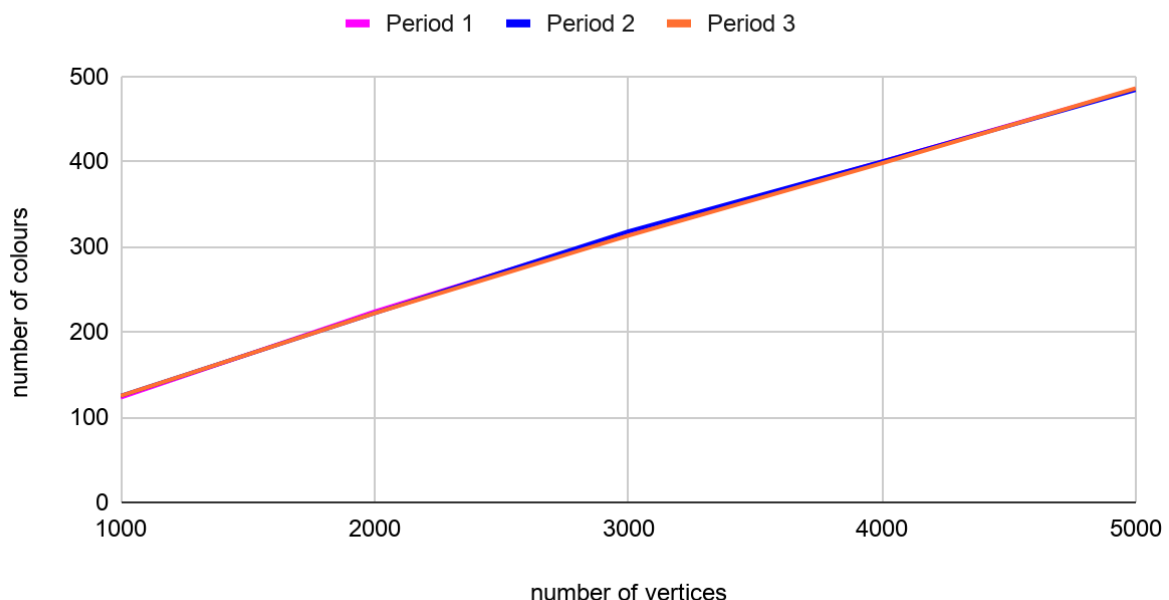
In the case of fine-grained lock also there will be no parallelism because when we are coloring vertex  $v$  we will lock all its neighbours and vertex  $v$ , for suppose take another vertex  $z$  if  $z$  is neighbour of one of neighbour of vertex  $v$ , then  $z$  will also wait because the neighbour of  $z$  is waiting like that if we have connected graph the same thing may happen, Indirectly fine grained lock is behaving same as coarse lock because there are high chances That when vertex is acquiring rest of all has to wait, and there will be more overhead in Fine grained lock for creating more semaphores and for calling `sem_wait` and `sem_post` Function many times, so finally coarse lock will be better than fine grained lock if we Have more connectedness in graph

So, our graph may be more connected because of that we got better performance for Coarse lock compared to fine grained lock

We showed that sequential is better than coarse lock and coarse lock is better than fine grained, so sequential will be better than fine grained lock

## Graph 2

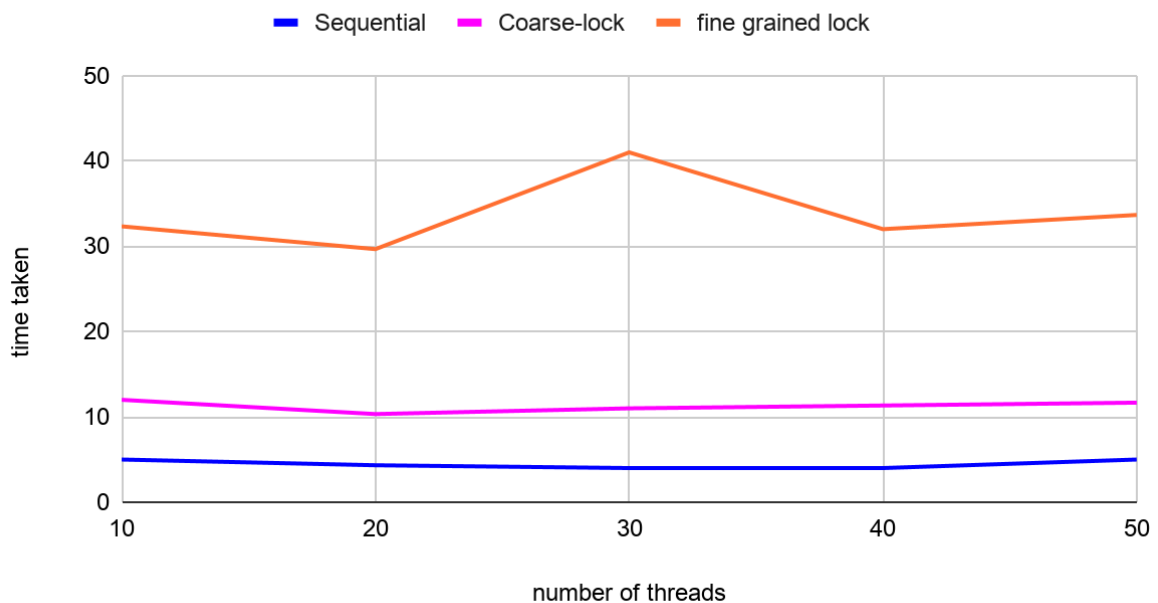
### number of colours vs number of vertices



The number of colors is almost same for sequential, coarse-lock, fine-grained, because we will Just change order of coloring in greedy algorithm, as number of vertices increases number Of colors also increases because as vertices increases we need more colors

Graph 3

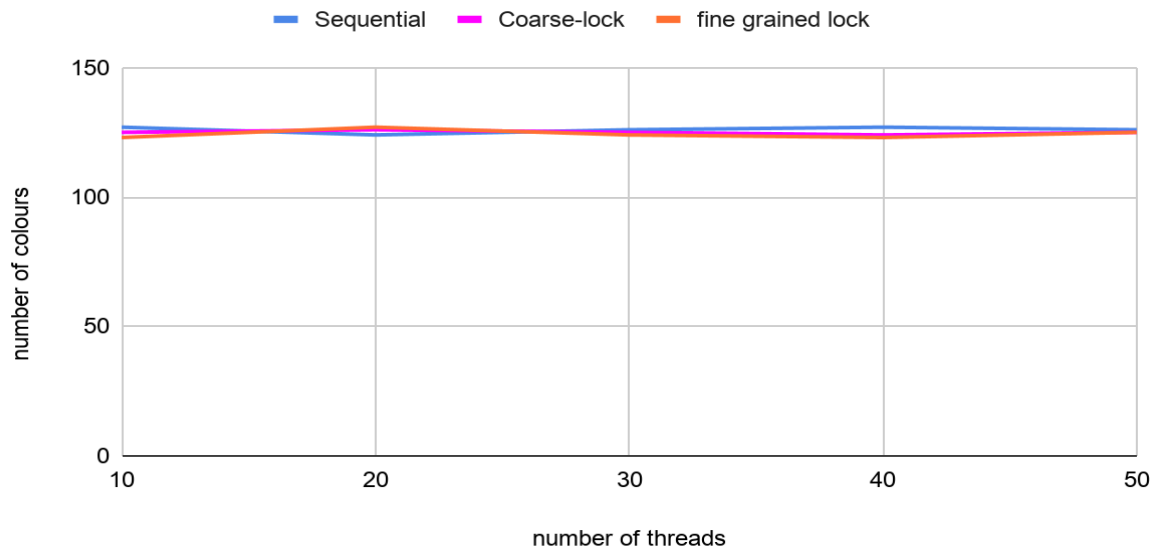
### time taken vs number of threads



There will be no use if the number of threads created is greater than number of cores Available,i am having four cores so there will no use of creating number of threads greater than 4 ,for sequential we are almost getting constant because there will be no threads In sequential but there are some fluctuations due to nature of CPU,for Coarse grained and fine grained if we increase number of threads the time taken should increase because there will be more overhead in creating threads ,but there are also We getting fluctuations due to nature of CPU In this graph also sequential is better than coarse-lock,fine grained and coarse lock is better Than fine grained due to same reason which was told for [graph1](#)

Graph 4

### number of colours vs number of threads



The number of colors is almost same for sequential,coarse-lock,fine-grained,because we will Just change order of coloring in greedy algorithm,the increase in number of threads will Not have any effect on colorings because number of vertices will remain constant