Operating Systems-II

Report-Asgn-5

Task:

The goal of this assignment is to implement the graph coloring algorithm in parallel using threads and synchronizes them with the help of locks.

Approach and Implementation:

1. I have defined a Vertex in the graph with the following information.

```
typedef struct vertex{
#ifdef FINE
   pthread_mutex_t lock;
#endif
   int vertexId;
   int partitionId;
   node *next;
   int color;
   bool boundaryVertex;
} vertex;
```

2. For efficiency in storage, I have modeled the graph as a list of vertices in the adjacency list form even though it is given as an adjacent matrix.

3. As given in the question, I have partitioned vertices into K partitions, where the size of each partition is N/K. and if there are any extra remaining vertices whose partition is not assigned yet, I have randomly assigned them to any of the K partitions. We perform this partition in order from vertex 1 to vertex N.

4. For every vertex, I have checked whether the given vertex is a boundary vertex by traversing the adjacency list and checking the partition Ids.

```
for (int i = 1; i <= vertices; i++)
        {
            node *ptr = g[i].next;
```

```
        while (ptr != NULL)
        {
            int j = ptr->id;
            if (g[j].partitionId != g[i].partitionId)
            {
                g[i].boundaryVertex = true;
                break;
            }
            ptr = ptr->next;
        }
    }
```

5. After the pre-processing is all done, K threads are created and thread function colorVertices is passed inside it along with the thread number.

6. Next, in the colorVertices function, I have checked which of the vertices is an internal vertex and which is a boundary vertex with the following condition,

```
for (int i = 1; i <= vertices; i++)
   {
       partition_id = g[i].partitionId;
       if (partition_id == tid)
       {
           if (g[i].boundaryVertex == 0)
               internal.push_back(&g[i]);
           else
               boundary.push_back(&g[i]);
       }
   }
```

7. For every internal vertex, I have Checked the colors of adjacent vertices and assigned the least possible color different from the adjacent vertices.

8. For external Vertex locking comes into the picture, Coarse and fine-grained only differ in the number of locks and the locking mechanism used. Once a thread acquires a lock the way of assigning the color to the vertex is the same in both

methods.

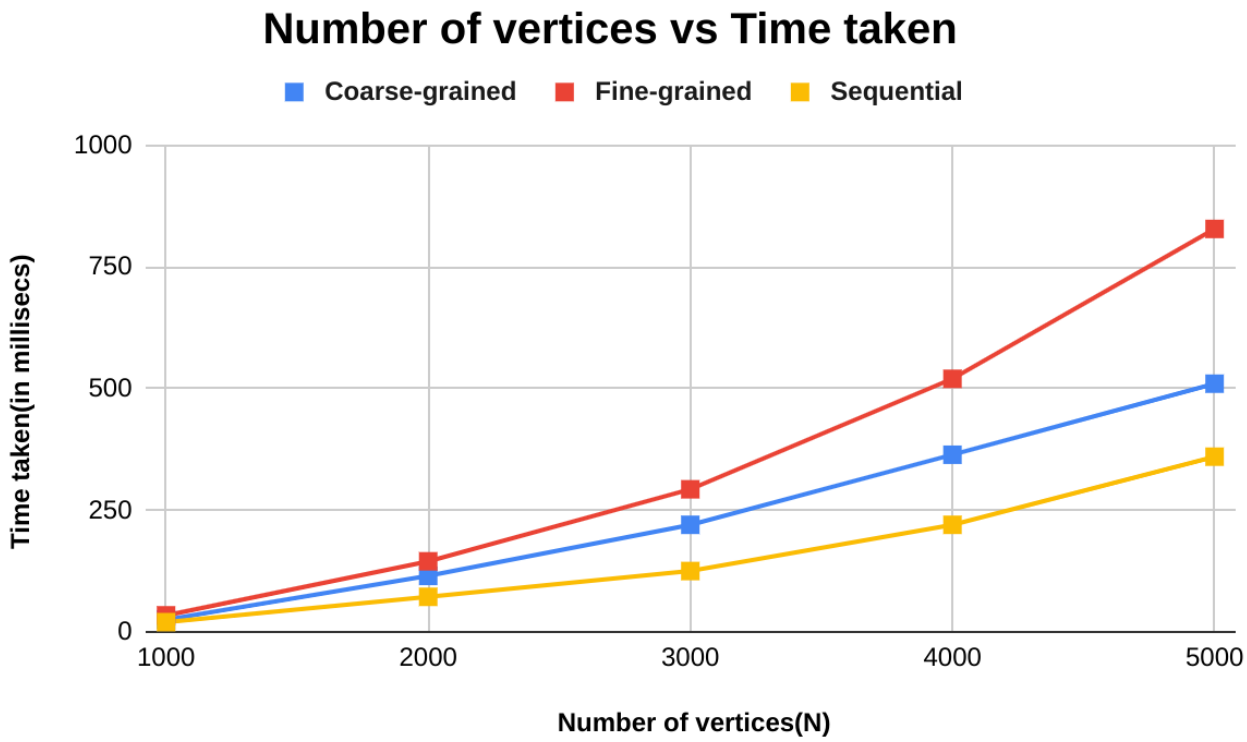
9. In the COARSE grained locking scheme, we need just one lock and in the FINE grained lock, we acquire the locks in increasing order so that deadlock won't occur.

10. I have measured the time using the Chrono library.

Graph Analysis:

Plot 1

K = 50 and varied the number of vertices in the graph from 10^3 to $5*10^3$ in the increment of 10^3



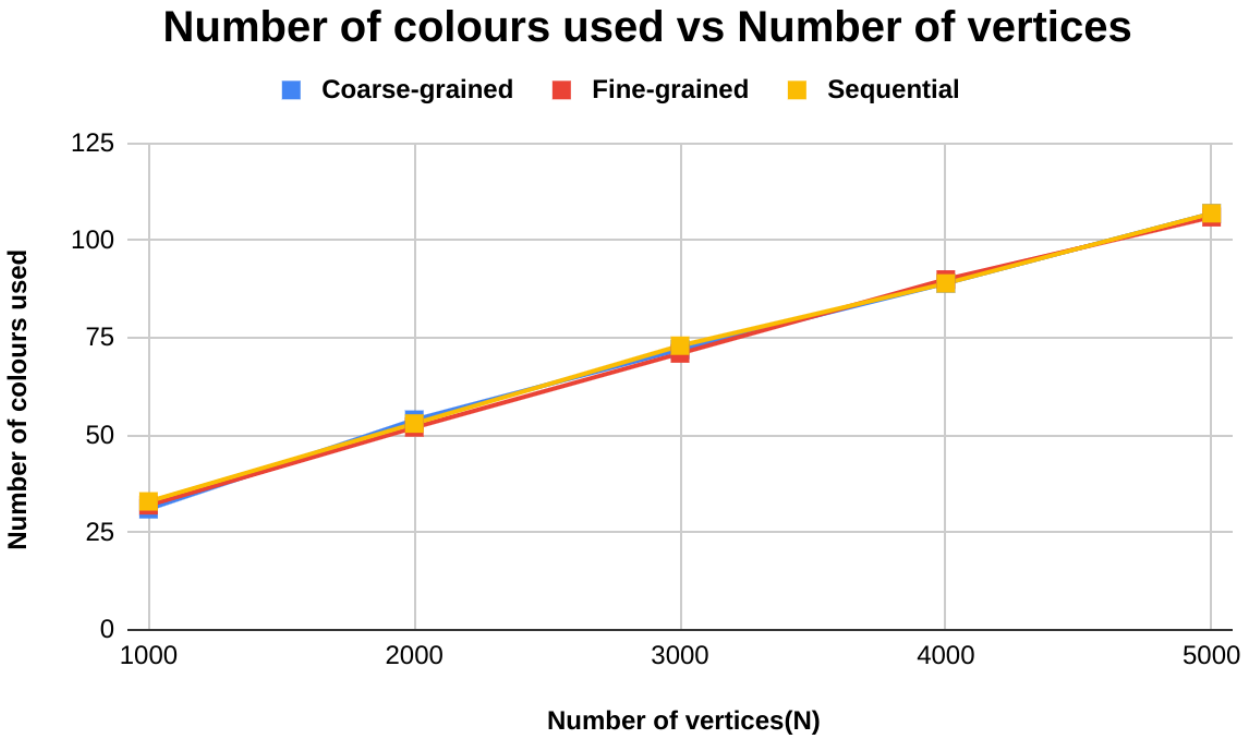
1. As the number of vertices increase, the time taken in any method will also increase as they have to color more number of vertices.

2. Fine-grained takes longer than coarse-grained since the amount of locks required to acquire a color external vertex in fine-grained is higher than the number of locks required in coarse-grained.

3. As a result, we can see in the graph that coarse-grained takes less time than fine-grained. In the case of sequential, there are no locks, and every vertex to be colored does not need to wait, hence it takes the least amount of time.

Plot-2

K = 50 and varied the number of vertices in the graph from 10^3 to $5*10^3$ in the increment of 10^3



1. It is observed that the number of colors used is more or less the same in all three algorithms.

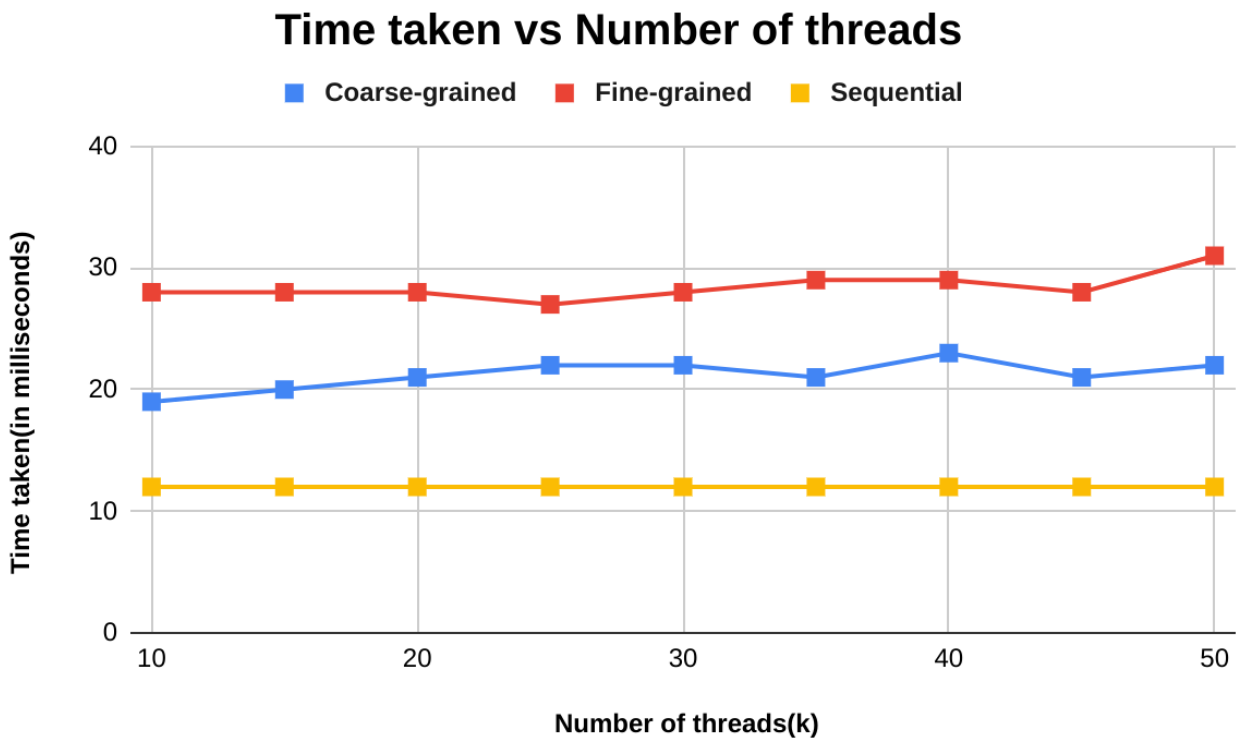
2. As the number of vertices increase, the number of colors used by the algorithm should also increase because the number of neighboring vertices is

increasing.

3. The number of colors obtained in the case of coarse, fine-grained depends on the order in which they acquired the locks.

Plot 3

N =10^3, varied number of threads from 10 to 50 in the increment of 5.



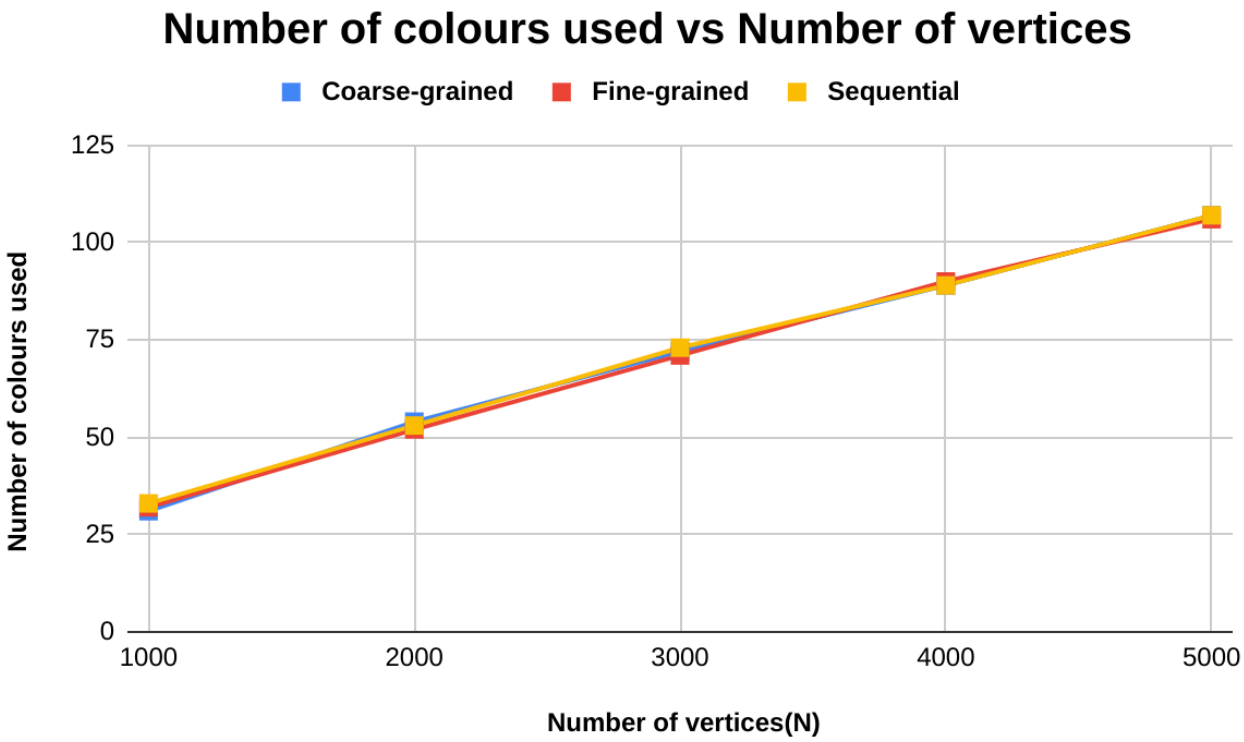
1. Here the number of vertices is fixed, hence even though the number of threads increases the time taken by sequential will not change.

2. As the number of threads increases, there is more competition for the locks hence time slightly increases.

3. Fine-grained takes longer than coarse-grained since the amount of locks required to color an external vertex in fine-grained is higher than the number of locks required in coarse-grained.

Plot 4:

N =10^3, varied number of threads from 10 to 50 in the increment of 5.



1. Here the number of vertices is fixed, hence even though the number of threads increases colors used by the sequential method should not differ by much.

2. The number of colors obtained in the case of coarse, fine-grained depends on the order in which they acquired the locks.

3. In this setting, the number of colors used is about the same in all three algorithms, with minor differences here and there.