

# Operating Systems-2

## Report: Assignment5

Name: Naitik Malav

Roll No.: CS19BTECH11026

1. **About Graph:** I have taken input from the file *input\_params.txt*, in which graph is generated by attached source code *randomgraph.cpp*. Generated graph is symmetric. I have selected a random number which is less than  $n^2$  and inside the for loop from 0 to  $n^2$  again randomly generated  $x, y$  and set  $\text{matrix}[x][y]=1$ , as well as  $\text{matrix}[y][x]=1$  (symmetric) which means there an edge between  $x$  and  $y$ .

```
20     long int r = rand()%(n*n);
21     for(long int i=0; i<r; i++) {
22         int x, y;
23         while(1) {
24             x=rand()%n;
25             y=rand()%n;
26
27             if(x!=y)
28                 break;
29         }
30
31         matrix[x][y] = 1;
32         matrix[y][x] = 1;
33     }
34 }
```

2. Now after taking input I have declared-

- i. node \*AdjList[n+1] – AdjList[i] represents adjacency list associated with  $i^{\text{th}}$  vertex. I am going to use indexes 1 to n so that's why I have declared n+1 node pointers.

```
struct node { //each node of graph
    int vertex;
    int type; //0 for internal vertex, 1 for external vertex
    int partitionIndex; //means this vertex lies in which partition
    node *next; //pointer to next node of class
};
```

- ii. list \*Partition[k+1] – Partition[i] has the information about all the vertices which are lying in partition  $P_i$ . Basically Partition[i]->vertex is the

linked list which stores all such vertices. I am going to use indexes 1 to n so that's why I have declared n+1 list pointers.

```
struct list{    //used for partition lists
    struct node *vertex;
    list *next;
};
```

- iii. int color[n+1], bool available[n+1] about color information and availability of colors of the  $i^{\text{th}}$  vertex. I am going to use indexes 1 to n so that's why I have declared n+1 node pointers.

3. After that I have declared an object of class Info which carries all relevant information using pointers and it can be used to pass all the relevant information in the thread function.

```
class Info {
public:
    struct node **AdjList; //pointer to array AdjList[n+1]
    struct list **Partition; //pointer to array Partition[k+1]
    int *color; //pointer to array color[n+1]
    bool *available; //pointer to array available[k+1]
};
```

```
Info object; //will pass into thread function namely coarseGrained
object.AdjList = AdjList;
object.Partition = Partition;
object.color = color;
object.available = available;
```

4. Uniform random partitioning of Graph: Randomly generated a number between 1 to k and then inserted a vertex from 1 to n in Partition[random number].

```
srand(time(NULL));
for(int i=1; i<=n; i++) {
    int r = rand()%(k) + 1; //random number between 1 to k
    Partition[r] = insert(Partition[r], AdjList[i]); //inserting a vertex into randomly generated partition
    AdjList[i]->partitionIndex = r; //set partitionIndex equal to r, will use it in future
}
```

5. In the globally declared *ExternalVerticesList*, I have stored all the vertices of graph which are boundary/external vertices with the help of *getExternalVertices* function.

**About *getExternalVertices* function:** Inside it comparing the values of partitionIndex of a vertex with it's neighbours. Rest of the part is according to question.

6. Then declaring *vector<thread> Thread* and pushing each thread into thread function along with class Info *object*.

Thread function in Coarse-Grained algorithm is : coarseGrained

Thread function in Fine-Grained algorithm is : fineGrained

**About *coarseGrained*:** (Pseudocode)

```
for all thread Thread[i] |  $i \in \{1, \dots, k\}$  do:
    for each  $v \in \text{Partition}[i]$  |  $v$  is an internal vertex in Partition[i] do:
         $\text{color}(v) \leftarrow \min\{\text{colors}[n+1] - \text{color}(\text{adjacent}(v))\}$ 
    end for
    for each  $v \in \text{Partition}[i]$  |  $v$  is a boundary vertex in Partition[i] do:
        Lock List (ExternalVerticesList)
         $\text{color}(v) \leftarrow \min\{\text{colors}[n+1] - \text{color}(\text{adjacent}(v))\}$ 
        Unlock List (ExternalVerticesList)
    end for
end for
```

**About *fineGrained*:** (Pseudocode)

```
for all thread Thread[i] |  $i \in \{1, \dots, k\}$  do:
    for each  $v \in \text{Partition}[i]$  |  $v$  is an internal vertex in Partition[i] do:
         $\text{color}(v) \leftarrow \min\{\text{colors}[n+1] - \text{color}(\text{adjacent}(v))\}$ 
    end for
    for each  $v \in \text{Partition}[i]$  |  $v$  is a boundary vertex in Partition[i] do:
```

Inserting nearby vertices of  $v$  which are external/boundary vertices into ExternalList.

getting all locks for neighbours and vertex and if the vertex gets all locks then only continue.

Locking all vertices in nearby external vertices in increasing order of vertex ids

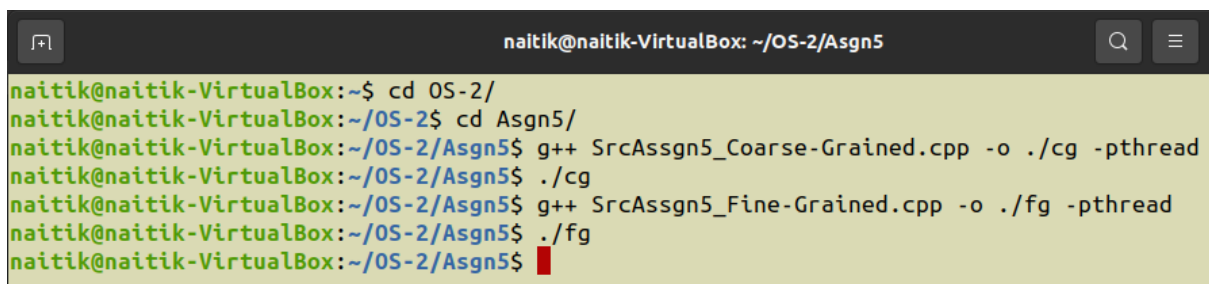
$\text{color}(v) \leftarrow \min\{\text{colors}[n+1] - \text{color}(\text{adjacent}(v))\}$

Unlock all vertices

end for

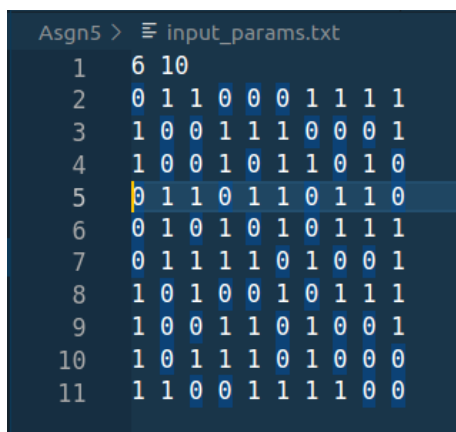
end for

## 7. Compilation Screenshot:



```
naitik@naitik-VirtualBox: ~/OS-2/Asgn5
naitik@naitik-VirtualBox:~$ cd OS-2/
naitik@naitik-VirtualBox:~/OS-2$ cd Asgn5/
naitik@naitik-VirtualBox:~/OS-2/Asgn5$ g++ SrcAsgn5_Coarse-Grained.cpp -o ./cg -pthread
naitik@naitik-VirtualBox:~/OS-2/Asgn5$ ./cg
naitik@naitik-VirtualBox:~/OS-2/Asgn5$ g++ SrcAsgn5_Fine-Grained.cpp -o ./fg -pthread
naitik@naitik-VirtualBox:~/OS-2/Asgn5$ ./fg
naitik@naitik-VirtualBox:~/OS-2/Asgn5$
```

## 8. Sample I/O: $n=10, k=6$



```
Asgn5 > input_params.txt
1 6 10
2 0 1 1 0 0 0 1 1 1 1
3 1 0 0 1 1 1 0 0 0 1
4 1 0 0 1 0 1 1 0 1 0
5 0 1 1 0 1 1 0 1 1 0
6 0 1 0 1 0 1 0 1 1 1
7 0 1 1 1 1 0 1 0 0 1
8 1 0 1 0 0 1 0 1 1 1
9 1 0 0 1 1 0 1 0 0 1
10 1 0 1 1 1 0 1 0 0 0
11 1 1 0 0 1 1 1 1 0 0
```

(Input file: input\_params.txt)

(Symmetric Matrix – without self loops, i.e  $i$  not equal to  $j$ )

a) output\_Coarse-Grained.txt

```
Asgn5 > ≡ output_Coarse-Grained.txt
1 Coarse Lock
2 No. of colors used: 5
3 Time taken by algorithm using coarse grained lock is: 955 microseconds
4
5 Colors:
6 V1-2, V2-3, V3-1, V4-4, V5-1, V6-2, V7-3, V8-5, V9-5, V10-4,
7
```

## b) output\_Fine-Grained.txt

```
Asgn5 > ≡ output_Coarse-Grained.txt
1 Coarse Lock
2 No. of colors used: 5
3 Time taken by algorithm using coarse grained lock is: 955 microseconds
4
5 Colors:
6 V1-2, V2-3, V3-1, V4-4, V5-1, V6-2, V7-3, V8-5, V9-5, V10-4,
7
```

## Graphs:

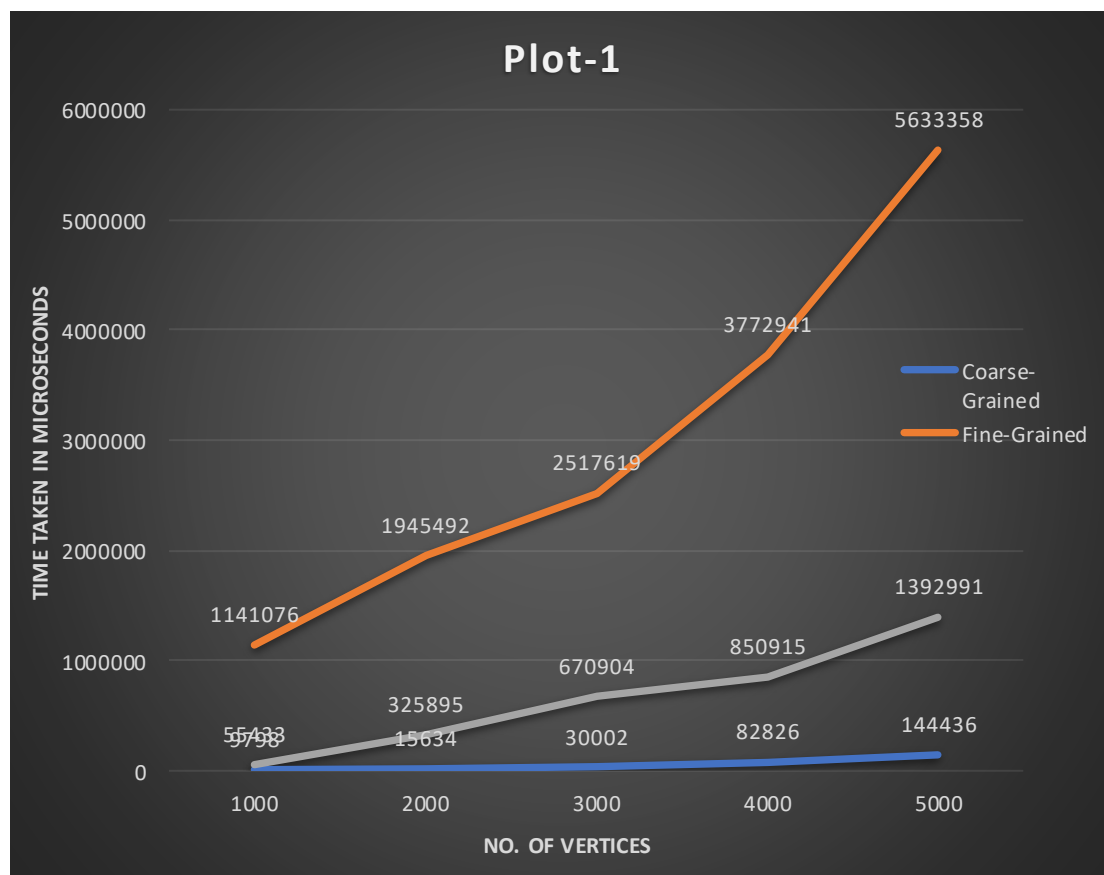
**Graph1:** Time Taken v/s Number of Vertices. Here number of vertices varies from 1000 to 5000. I have taken value of  $k=5$ . Time taken is in microseconds.

### Graph1-Analysis:

1. Position of graph curves: Coarse-Grained lies always below, then after sequential which is just above coarse-grained and below fine-grained.
2. Rate of increase in time taken in fine grained is more than coarse-grained, than in sequential and slowest in coarse-grained.
3. For all of them time taken is rises if number of vertices are increases.

Note: Due to scaling coarse-grained is looking like straight line but it's not a straight line. It's actually increasing. Check out the values for more clarity.

number of vertices(n)	coarse-grained (in us)	fine-grained (in us)	Sequential (in us)
1000	9798	1141076	55433
2000	15634	1945492	325895
3000	30002	2517619	670904
4000	82826	3772941	850915
5000	144436	5633358	1392991

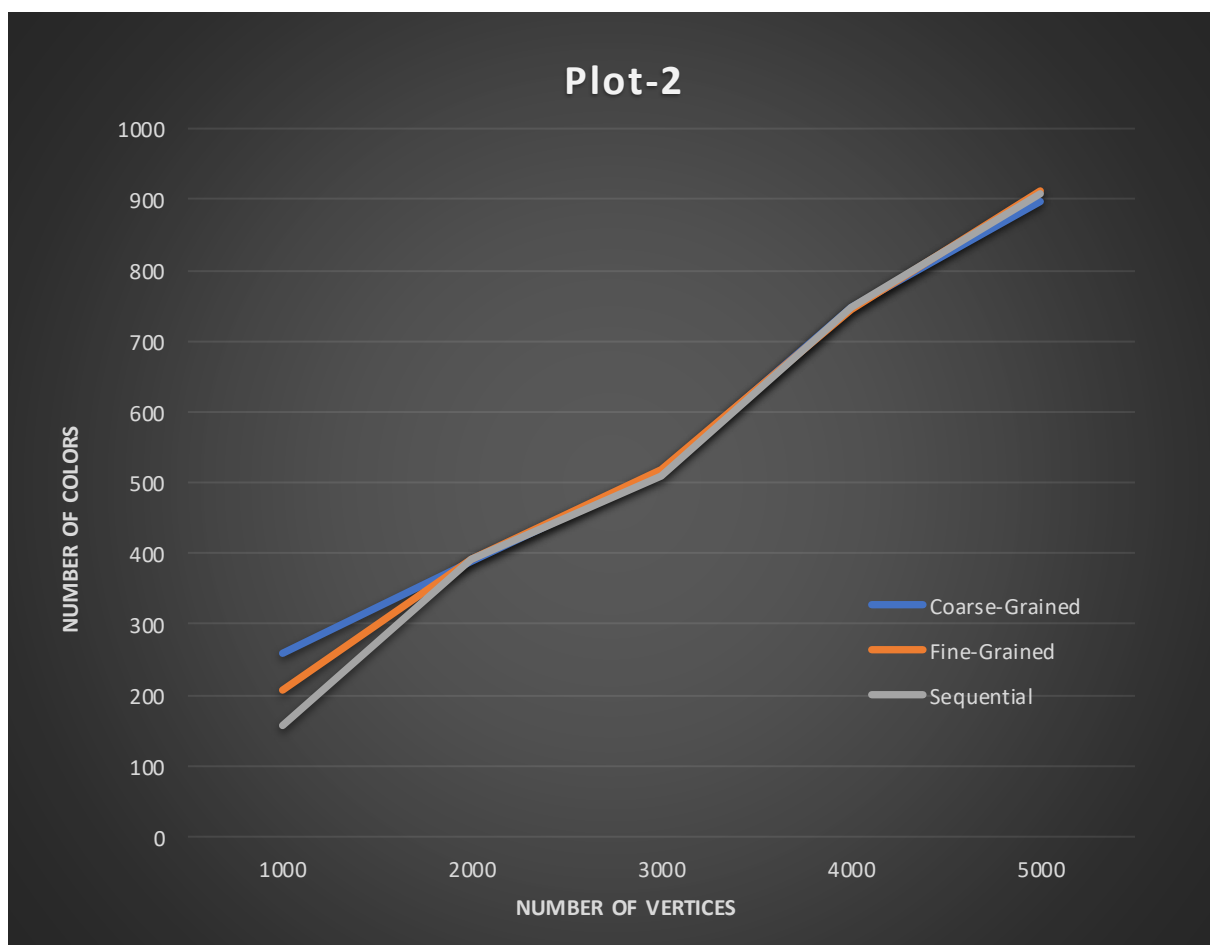


**Graph2:** Number of colors v/s Number of Vertices. Here number of vertices varies from 1000 to 5000. I have taken value of  $k=5$ .

**Graph2-Analysis:**

1. Number of colors used are almost same for every method for a particular value of number of vertices.
2. As the number of vertices increases means in our random graph degree also increases so the number of colors used are also increases.

number of vertices	Coarse-Grained (colors used)	Fine-Grained (colors used)	Sequesntial (colors used)
1000	259	207	157
2000	390	391	393
3000	515	519	508
4000	748	744	747
5000	897	912	908

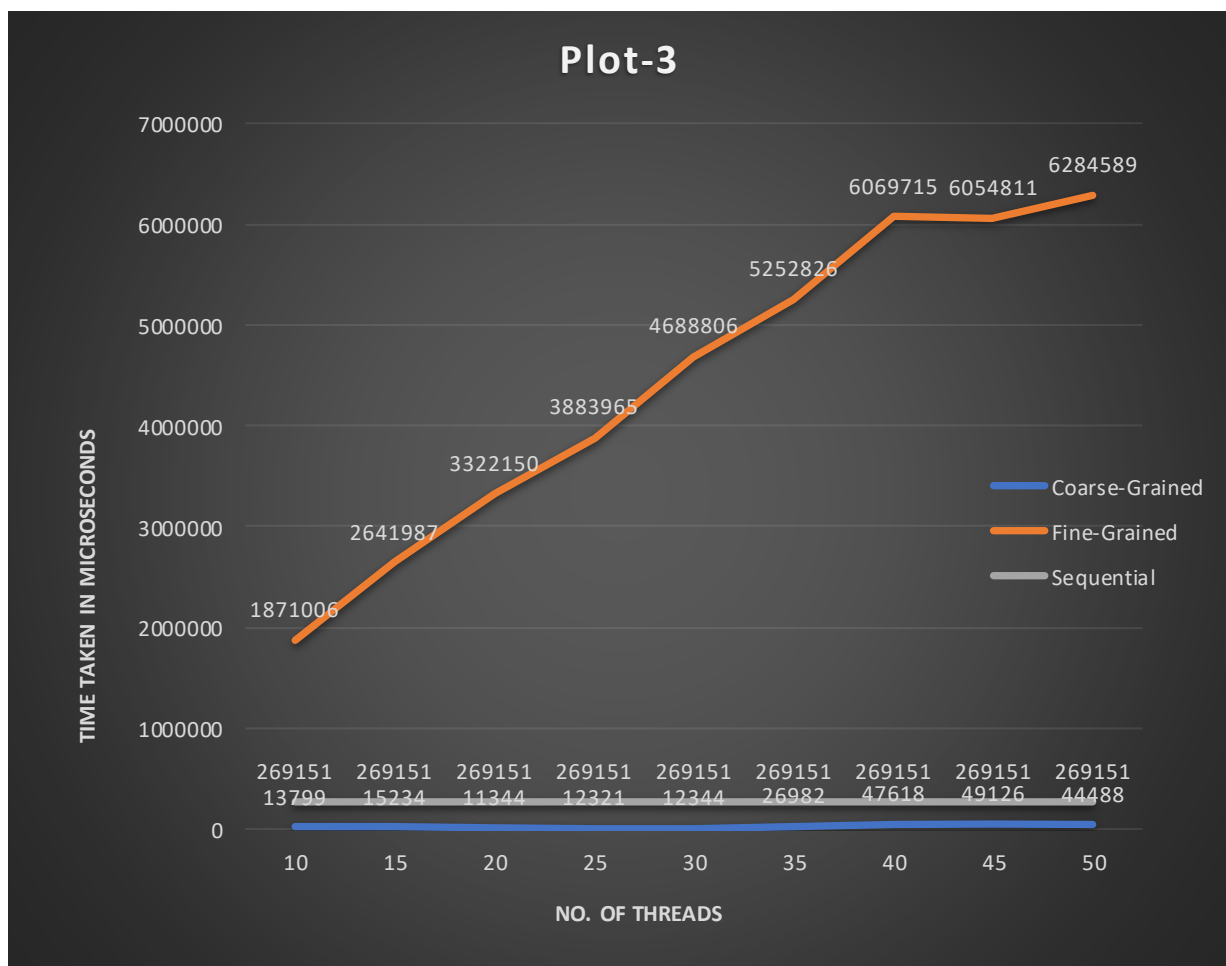


**Graph3:** Time Taken v/s Number of Threads. Here number of threads varies from 10 to 50. I have taken value of n=1000.

**Graph3-Analysis:**

1. Order of graph curves is Fine-grained is on top then sequential and then coarse-grained.
2. Sequential remains constant.

Number of threads (k = partitions = p)	Coarse-Grained (in us)	Fine-Grained (in us)	Sequential (in us)
10	13799	1871006	269151
15	15234	2641987	269151
20	11344	3322150	269151
25	12321	3883965	269151
30	12344	4688806	269151
35	26982	5252826	269151
40	47618	6069715	269151
45	49126	6054811	269151
50	44488	6284589	269151





**Graph4:** Number of colors v/s Number of Threads. Here number of threads varies from 10 to 50. I have taken value of  $n=1000$ .

**Graph4-Analysis:**

1. Almost all values are similar.
2. For coarse-grained and fine-grained it's decreasing. And for sequential it's constant.

Number of threads (k = partitions = p)	Coarse-Grained (colors-used)	Fine-Grained (colors-used)	Sequential (colors-used)
10	274	280	275
15	267	266	275
20	262	264	275
25	255	259	275
30	258	257	275
35	245	245	275
40	239	241	275
45	225	228	275
50	202	204	275

