Programming Assignment 5
Implementing Graph Coloring Algorithm using locks
Submission Date: 12th April 2021, 9:00 PM

Goal: This assignment aims to implement the graph colouring algorithm in parallel using threads and synchronises them with the help of locks.

Details: Graph colouring is an assignment of labels traditionally called "colours" to each vertex in a given undirected graph. A graph colouring must have a unique property: given two adjacent vertices, i.e., such that there exists an edge between them, they must not share the same colour. In this assignment, you have to implement a parallel graph colouring algorithm based on locks in c++. Greedy colouring is one of the sequential algorithms for graph colouring. In this method, vertices are processed in the order. Each vertex is assigned a colour that is not already used by any of its adjacent vertices. This algorithm generally doesn't use the minimum number of possible colours.

In this assignment, you have to implement a parallel greedy graph colouring algorithm where each thread is assigned a partition. You consider the number of threads k as one of the parameters. All the vertices (n being the total number of vertices) in the graph are randomly partitioned into p partitions and each partition is assigned to a thread. Note that unless mentioned otherwise, you can assume that p is equal to k.

In a partition, there are two kinds of vertices:

a. Internal Vertex: Vertex having all the adjacent vertices in the same partition.
b. External Vertex: Vertex having one or more than one adjacent vertex in another partition.

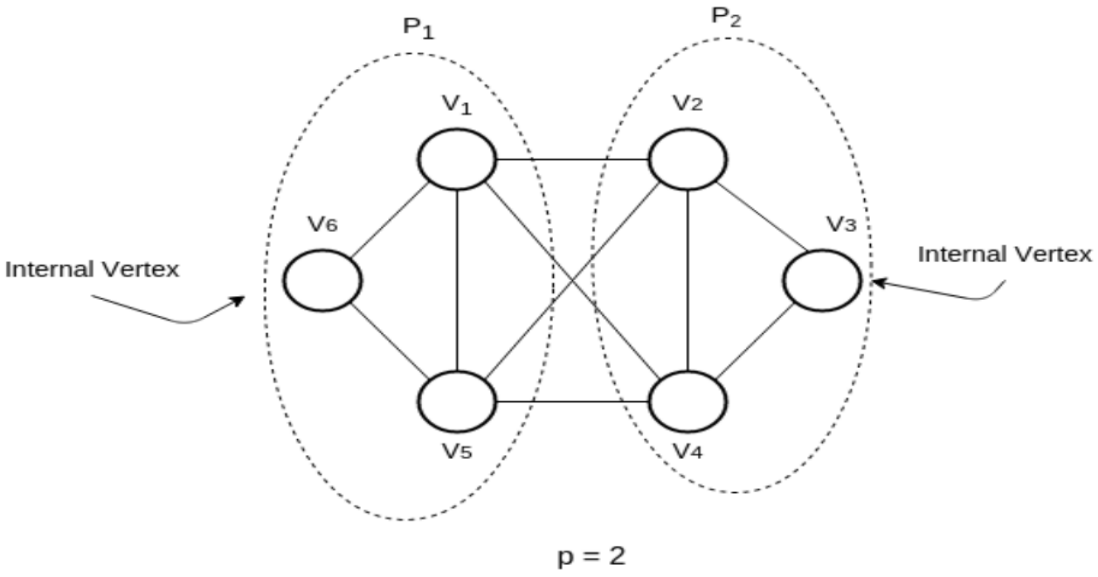
The figure below illustrates it. In the given figure graph is partitioned into two ($p = 2$) partitions, P1 and P2. Here V6 is the internal vertex of P1, similarly, V3 is the internal vertex of P2. V1 and V5 are the external vertices of partition P1 as the adjacent vertices of these V1 and V5 lie in a different partition. Similarly, V2 and V4 are the external vertices of the partition of P2.

In this algorithm, the thread to which a partition is assigned is responsible for the colouring of vertices in its partition. A vertex Vi is said to be in **conflict** with Vj if they are neighbours and have the same colour.

Initially, all the vertices are uncoloured. We denote the colour of such a vertex as ↓ (bottom). A thread Thi colours as follows:

I. Internal vertices: First checks for conflicts of internal vertices within the partition as by definition there can't be any conflict of these nodes with vertices in other partitions. It considers each internal vertex vi and assigns it with a colour that is not the same as any of the neighbours which is the greedy colouring. If any neighbour of vi, say vj, has the colour ↓then vj is ignored. As mentioned above, vi is coloured using a simple greedy approach.

II. External vertices: With the case of external vertices which have neighbours on a different partition, the thread Thi needs to synchronize with other threads to check for conflicts and resolve them to correctly colour all the vertices in its partition. This is achieved as described below.



Colouring Boundary Vertices using Locks

There are two ways of colouring boundary vertices using locks:

1. **Coarse-Grained Lock:** Here there is one common lock CL for all boundary vertices. For colouring any boundary vertex bvx, in one of its partitions, thread Thi has obtained the lock on CL. Assigns a colour to bvx in a greedy manner by looking at all the neighbours 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 neighbours.

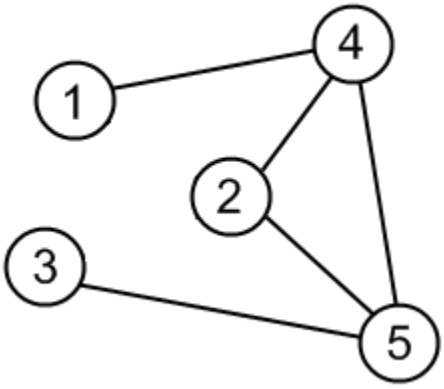
2. **Fine-Grained Locks:** Here each vertex has an individual lock also denoted as a fine-grained lock. When the thread Thi wishes to colour a boundary vertex bvx, it locks all the neighbours of bvx and bvx as well. It obtains the locks in an increasing (or decreasing) order of vertex ids to avoid deadlocks. After obtaining the locks, it assigns a

colour to bvx after looking at the colours of vx's neighbours in a greedy manner as explained above. Again a vertex that is uncoloured (having a colour of ↓) is ignored.

Input: Input to the program will be a file, named input_params.txt, consisting of the number of threads followed by the no. of vertices in the graph. From the next line, there should be the representation of the graph in adjacency matrix format. The sample input file is as follows:

```
2 5
  1 2 3 4 5
1 0 0 0 1 0
2 0 0 0 1 1
3 0 0 0 0 1
4 1 1 0 0 1
5 0 1 1 1 0
```

This is the representation of the above graph as follows:



Here 2 in the first line represents the number of threads while 5 is the number of vertices in the graph. You can use automatic scripts to generate graphs. Please note that the graph is undirected so your adjacency matrix should be symmetric.

Output: Your program should output a file name output.txt in which you should be printing the number of colours used, the time taken and the colour of each vertex for each variant. The sample output file is as follows:

```
Coarse Lock
No. of colours used: 5
Time taken by the algorithm using: 1573 Millisecond
Colours:
```

v1 - 1, v2 - 2, v3 - 1, v4 - 5, ……..

Fine-Grained Lock
No. of colours used: 5
Time taken by the algorithm using: 1457 Millisecond
Colours:
v1 - 2, v2 - 3, v3 - 1, v4 - 3, ……..

Note that the TAs will check for the correctness of the colouring of your algorithm.

Report: You have to submit the report for this assignment containing the design and the algorithm's implementation details. This report should be unequivocal to understand your algorithm and its implementation. You have to submit three graphs with analysis in this report. For that, you also need to implement the sequential algorithm for comparison.

A. **Plot1:** X-axis should be the number of vertices in the graph varying from 1 * 10^4 to 5 * 10^4 the increments of 10^4. Y-axis should demonstrate a comparison of time taken by the three algorithms: (1) Sequential Algorithm execution (2) Coarse Lock (3) Fine-Grained Lock. You should fix the number of threads for all the graphs to some k.

B. **Plot2:** It takes the same input as above and shows the colours used. The X-axis should be the number of vertices in the graph varying from 1 * 10^4 to 5 * 10^4 the increments of 10^4. Y-axis should demonstrate a comparison of the number of colours used by the three algorithms: (1) Sequential Algorithm execution (2) Coarse Lock (3) Fine-Grained Lock. You should fix the number of threads for all the graphs to some k.

C. Plot3: X-axis should be the number of threads varying from 10 to 50 with the increment of 5. Y-axis should demonstrate a comparison of time taken by the three algorithms: (1) Sequential Algorithm execution (2) Coarse Lock (3) Fine-Grained Lock. You should fix the number of vertices to 1*10^4.

D. Plot4: The input here is again similar to the case C above. X-axis should be the number of threads varying from 10 to 50 with the increment of 5. Y-axis should demonstrate a comparison of the number of colours used by the three algorithms: (1) Sequential Algorithm execution (2) Coarse Lock (3) Fine-Grained Lock. You should fix the number of vertices to 1*10^4.

Your report should also contain the analysis of the results while explaining any anomalies observed.

Deliverables: You have to submit the following:

- The source file containing the actual programs to execute name it as SrcAssgn5<Roll_NO>.pdf.
- A readme.txt that explains how to execute the program.
- The report as explained above.

Zip all the three files and name them as ProgAssgn5-<rollno>.zip. Then upload it on the google classroom page of this course by the deadline as mentioned above. Please see the instructions given above before uploading your file. Your assignment will NOT be evaluated if there is any deviation from the instructions posted there.

The policy for grading this assignment will be

- Design as described in report and analysis of the obtained results: 50%
- Execution: 40%
- Code documentation and indentation: 10%.

As mentioned before, all assignments for this course has the late submission policy of a penalty of 10% each day after the deadline.

Kindly remember that all submissions are subjected to plagiarism checks.