Parallel Graph Coloring Using OpenMP

Ayush Bhandari-1811T209
Information Technology
National Institute of Technology
Surathkal,Karnataka 575025
Email:.ayushbhandari8909@gmail.com

Abhishek Kaswan-181IT201
Information Technology
National Institute of Technology
Surathkal,Karnataka 575025
Email:abkaswan@gmail.com

Siddharth Pokharna - 181IT146
Information Technology
National Institute of Technology
Surathkal,Karnataka 575025
Email:spokharna2002@gmail.com

Abstract— In large-scale parallel applications a graph coloring is often carried out to schedule computational tasks.Graph coloring problem is to assign colors to certain elements of a graph subject to certain constraints. The algorithm operates in an iterative fashion, in each round vertices are speculatively colored based on limited information, and then a set of incorrectly colored vertices, to be recolored in the next round, is identified.Parallel speedup is achieved in part by reducing the frequency of communication among processors.

INTRODUCTION

The graph coloring problem deals with assigning each vertex of the graph with a color such that the adjacent vertices have distinct colors. Many applications have been shown to be reducible to the graph coloring problem. Some of these include :-

1) register allocation in the back-end of a compiler.

2) scheduling of multiple tasks to resources under various constraints in the CPU.

In many parallel scientific computing applications computational dependencies are modeled using a graph, and a coloring of the vertices of the graph is used as a subroutine to identify independent tasks that can be performed concurrently. In such cases, the computational graph is often distributed among the processors, and hence the coloring itself needs to be performed in parallel. For these applications, fast greedy coloring algorithms that work well in practice are often preferred over slower local improvement heuristics that might use fewer colors.

This project deals with the parallelization of such fast greedy coloring algorithms and presents an efficient parallel coloring algorithmic scheme designed for distributed memory parallel computers.

The basic idea in the algorithm is to partition the graph among the available processors and let each processor be responsible for the coloring of the vertices assigned to it

PROBLEM STATEMENT

The graph coloring problem deals with assigning each vertex of the graph with a color such that the adjacent vertices have distinct colors.It is NP-complete to determine if a given graph admits k-coloring (except for k=1 or k=2).

The focus of our task is to implement algorithms to solve the Distance-1 graph coloring problem, i.e. coloring of the vertex set such that any pair of adjacent vertices receive distinct colors. Graph coloring is a hard problem to compute solutions for, hence our aim is to implement greedy heuristics that utilize parallel approach.

OBJECTIVE

1)To study parallelism in graph coloring algorithm by using Distance 1 coloring .
2)Analysis of no. of colors used and the time required with variation of densities of graph as well as varying the number of threads in process.
3) Plot the graph for the performance comparison with different graphs and different no.of thread used.

LITERATURE SURVEY

1. Parallel graph coloring on multi cores CPU by Per Nomann
Study on parallelism on graph coloring algorithms and creating benchmarks for different graph coloring methods including D1coloring ,Luby Jones , D2coloring,Jones Plassaman,etc.

2. **A comparison of different parallel graph coloring algorithms by JR Allwright, R. Bordawekar, C.L Martin**
Comparing different graph coloring algorithms on varying no .of vertices i.e from 256 -16384.
D1 coloring approaches in different ways i.e linked list,trees,forbidding color transversal .

3. **Parallel Graph Coloring for Manycore Architecture by Mehmet Deveci, Erik G Boman, Karen D Devine, and Sivasankaran Rajamanickam**
Identifying different regions for parallelism in graph coloring and analysis of result with varying no.of threads in program and factors affecting results as size of graph,threads.

METHODOLOGY

1.SEQUENTIAL ALGORITHM

The size of the graphs are huge, the algorithm follows a greedy approach instead of dynamic programming, therefore the total number of colors used in the graph may result in a suboptimal number. In this approach, all the vertices are iterated over one-by-one to find appropriate color that satisfies distance-1 coloring conditions.

On each iteration (i.e. for every vertex), the smallest color that is not used by its neighboring vertices is assigned to the vertex. Therefore, this implantation favors a first-fit manner for assigned colors.

In order to find the smallest available color, neighbors of the vertex are visited, and a Boolean array is used to store this information. Then, by iterating over this Boolean smallest unused color can be easily determined. By this approach, we can determine the smallest unused color in O(k) time complexity, where k represents the number of neighbors of the vertex.

2.PARALLEL ALGORITHM

Graph coloring is a hard problem to compute solutions for, hence our aim is to implement greedy heuristics that utilize parallelism.

One of the most costly operations of this algorithm is the atomic operations.critical regions should be used because the main loop termination condition is dependent on it. but there is another mechanism to traverse all vertices and check if there is at least one vertex without a color assigned to it. And in parallel we made slight changes where we used parallel methods and operators on the main functions like accuracy(), performColoring(), assignColors(), detectConflicts() and forbidColors().

We present this suitable greedy approach for dealing with the Graph Coloring problem on a shared memory programming model which means that threads communicate only by writing to and reading from the shared memory. We begin by assuming that all the vertices in the graph are assigned a unique id's from {1, 2, . . . , |V |}. Initially, the graph G = (V, E) has been preprocessed by partitioning it uniformly into p partitions where p is the number of threads. Then the vertices with their corresponding id's in {1, . . . , |V |/p} are assigned to partition V1, {|V |/p + 1, . . . , 2|V |/p} get assigned to partition V2 and so on until Vp. It seems only fair that the graph partitioning preprocessing time be included in the overall time taken for graph coloring.

Hence, using this random heuristic based partitioning helps us bring down the overall time taken for coloring.

Therefore, we do not use a graph partitioning software for minimizing the crossing edges between various partitions because it effectively increases the coloring time.

The vertices in each partition/block can be classified into:- internal vertices (whose all the neighbouring vertices lie in the same partition) and boundary vertices (those who have neighbours belonging to other partitions). Each thread is responsible for proper coloring of all the vertices in its partition. The subsequent subsections present algorithms using the First Fit Coloring strategy, which assigns each vertex the least legal color available.

Region of Parallelism

1. Assigning colors to vertices.
2. Detecting conflicts between adjacent vertices.
3. Forbid the coloring.

IMPLEMENTATION

1) Conversion of .mtx file into readable format of different types of input files.

Input Files:

Table 1: Size of the different input files graphs in terms of nodes and edges

S.no	File	Nodes	Edges
1	**af_shell10**	**1508065**	**25582130**
2	**bone010**	**98665**	**35339115**
3	**coPapersDBLP**	**540486**	**15245729**
4	**nlpkktp120**	**3542400**	**46651696**

2)Create sequential algorithm code using d1_coloring and greedy approach.

3)Create parallel algorithm code using Sequential code as follows:

Algorithm 1: Parallel Graph Coloring for

Result: color c i for ∀ v i ∈ V
while ∃ v i ∈ V not colored do
assignColors();
#pragma omp barrier
detectConflicts();
#pragma omp barrier
forbidColors();
#pragma omp barrier
allColored();
end

---

4)Store the output results in form of .txt files

5) Plot the graphs of the results

Parallel Region:-

1. assignColor():
#pragma omp parallel for is used.

2. detectConflicts():
#pragma omp parallel for schedule(guided) ,
#pragma omp barrier

3. forbidColors():
#pragma omp parallel for collapse()

RESULT AND ANALYSIS

RESULT:

A: af_shell10 **B: bone010**
C: coPapersDBLP **D: nlpkktp120**

In all tables time taken in ms and data entered is mean of each entry

i)Sequential

Table 2: Time taken by various functions in graph coloring and total time taken in sequential program

S.No	Type	A	B	C	D
1	Assign	159	327	407	84
2	Detect	50	73	37	95
3	Forbid	0	0	0	0
4	Colors	15	39	337	2
5	Total Time	576	906	658	894

ii)Parallel (best thread performance)

Table 3: Time taken by various functions in graph coloring and total time taken in parallel implementation.

S.No	Type	A	B	C	D
1	Assign	23	55	128	60
2	Detect	20	23	15	76
3	Forbid	0	0	0	0
4	Colors	25	45	337	4
5	Total Time	377	497	254	748

iii)Improvement due to parallel implementation

Table 4: Performance improvement in various functions due to parallelism(i.e parallel time - sequential time).

S.No	Type	A	B	C	D
1	Assign	27	272	279	24
2	Detect	30	50	22	19
3	Forbid	0	0	0	0
4	Colors	10	6	0	2
5	Total Time	199	409	404	146

ANALYSIS:

i)Parallel algorithm performs better in terms of time taken then Sequential irrespective of no.of threads used in parallel.

ii)In parallel assignColor() and detectColor() have significant change in time taken whereas in case of forbidColor() time change is insignificant.

iii)No.of colors used is either equal to or greater than then Sequential region due to more collision.

Sequential Outputs:-

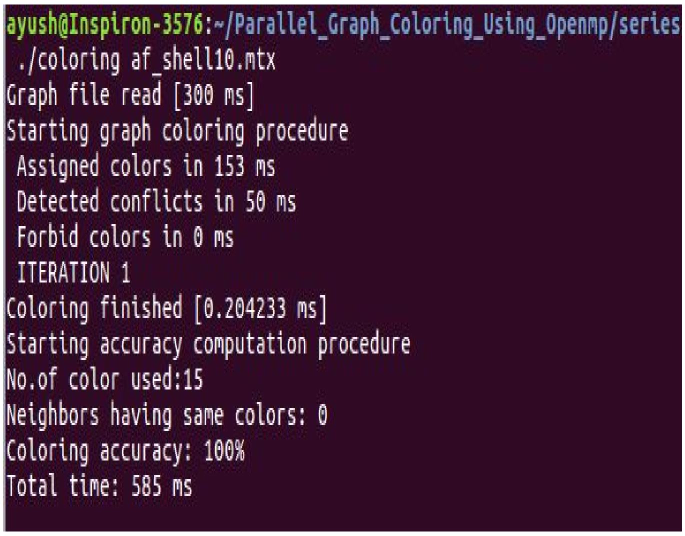

Fig1: Output using file af_shell10.mtx

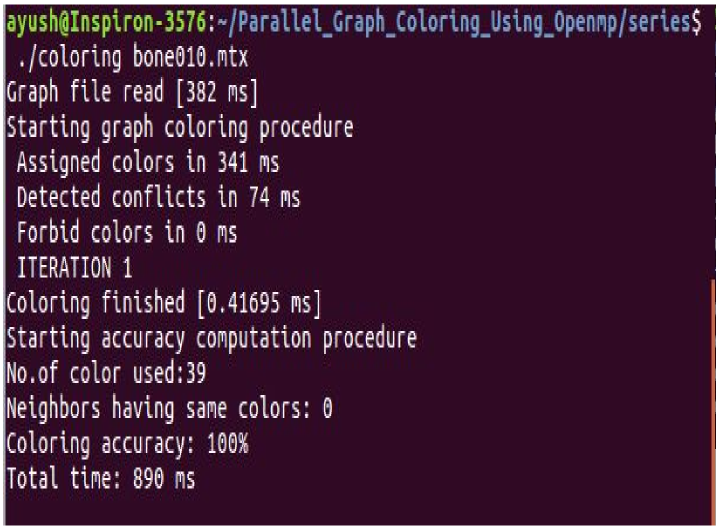

Fig2: Output using file bone010.mtx

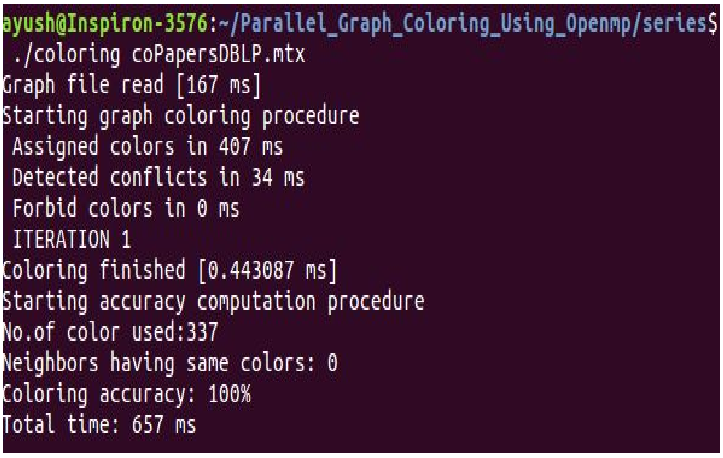

Fig3: Output using file coPapersDBLP.mtx

ii) nlpkkt120.mtx

```
ayush@Inspiron-3576:~/Parallel_Graph_Coloring_Using_Openmp/series
 ./coloring nlpkkt120.mtx
Graph file read [494 ms]
Starting graph coloring procedure
 Assigned colors in 85 ms
 Detected conflicts in 95 ms
 Forbid colors in 0 ms
 ITERATION 1
Coloring finished [0.182909 ms]
Starting accuracy computation procedure
No.of color used:2
Neighbors having same colors: 0
Coloring accuracy: 100%
Total time: 844 ms
```

Fig 4: Output using file nlpkkt120.mtx

Parallel

For coPapersDBLP:-
i)With 2 threads

```
ayush@Inspiron-3576:~/Parallel_Graph_Coloring_Using_Openmp$
 ./coloring coPapersDBLP.mtx 2
Graph file read [78 ms]
Starting graph coloring procedure
 Assigned colors in 127 ms
 Detected conflicts in 14 ms
 Forbid colors in 0 ms
 ITERATION 1
 Assigned colors in 0 ms
 Detected conflicts in 1 ms
 Forbid colors in 0 ms
 ITERATION 2
Coloring finished [144 ms]
Starting accuracy computation procedure
No.of colors used337
 Neighbors having same colors: 0
Coloring accuracy: 100%
Total time: 249 ms
```

Fig 5: Output using OpenMP from file coPapersDBLP.mtx using 2 threads

ii)With 4 threads

```
ayush@Inspiron-3576:~/Parallel_Graph_Coloring_Using_Openmp$
 ./coloring coPapersDBLP.mtx 4
Graph file read [103 ms]
Starting graph coloring procedure
 Assigned colors in 140 ms
 Detected conflicts in 22 ms
 Forbid colors in 0 ms
 ITERATION 1
 Assigned colors in 0 ms
 Detected conflicts in 1 ms
 Forbid colors in 0 ms
 ITERATION 2
Coloring finished [164 ms]
Starting accuracy computation procedure
No.of colors used337
 Neighbors having same colors: 0
Coloring accuracy: 100%
Total time: 302 ms
```

Fig 6: Output using OpenMP from file coPapersDBLP.mtx using 4 threads

iii)with 8 threads

```
ayush@Inspiron-3576:~/Parallel_Graph_Coloring_Using_Openmp$
 ./coloring coPapersDBLP.mtx 8
Graph file read [97 ms]
Starting graph coloring procedure
 Assigned colors in 141 ms
 Detected conflicts in 19 ms
 Forbid colors in 0 ms
 ITERATION 1
 Assigned colors in 0 ms
 Detected conflicts in 1 ms
 Forbid colors in 0 ms
 ITERATION 2
Coloring finished [162 ms]
Starting accuracy computation procedure
No.of colors used337
 Neighbors having same colors: 0
Coloring accuracy: 100%
Total time: 292 ms
```

Fig 7: Output using OpenMP from file coPapersDBLP.mtx using 8 threads

Graph Plots

In graphs
x: No.of threads
y: Time taken to finish(ms)

i)Plot for bone010

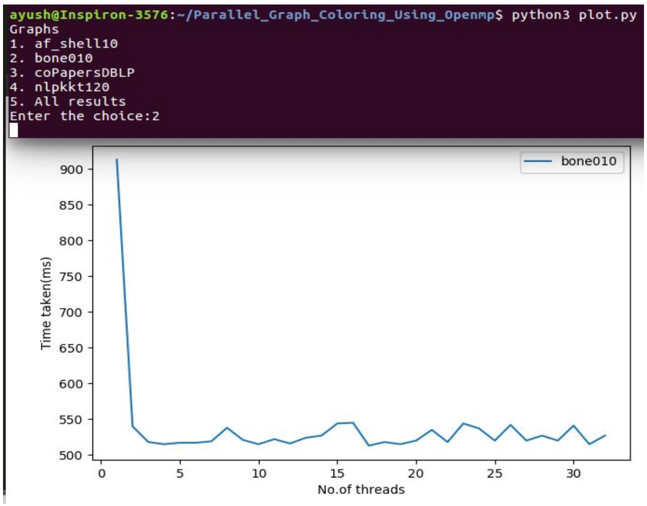

Fig8: Graph for input file bone010.mtx

ii) Plot of all input files i.e af_shell10 , bone10 , coPapersDBLP , nlpkkt1220 with best performance of each file.

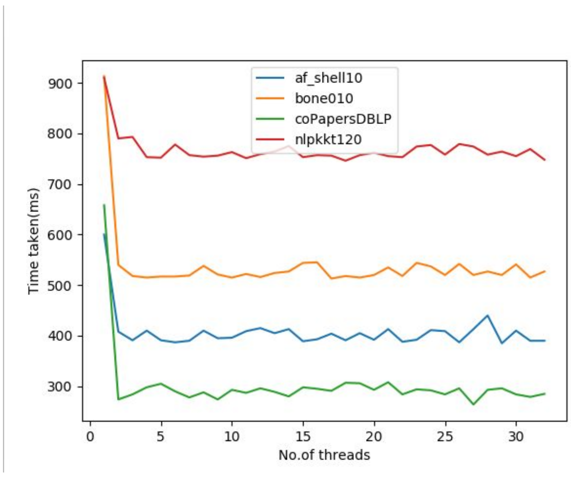

Fig 9:Graph for of all input files with varying no.of threads

INDIVIDUAL CONTRIBUTION

S.No	Name	Contribution
1	Ayush Bhandari	i)Implementation of perform coloring and isfullycolored in parallel graph coloring. ii)Testing the parallel and sequential graph coloring with different no.of threads and graph with different densities and debugging the files.
2	Siddharth Pokharna	i)Implementation of Assign color and Detect color in parallel graph coloring iii)Implement Parallel part in forbid coloring , in allcolored() function and check accuracy.
3	Abhishek Kaswan	i)Implementation of sequential for graph coloring including assign colors,forbid coloring, getnext color. ii)Change of mtx format input into readable format for the program.

FUTURE WORK

i)We could have used distance-2 graph coloring with improvements in the current algorithm. So for future work we will improve the OpenMP algorithm by using thread-private forbidden arrays and also use CUDA implementation. Parallelism on CPU will be achieved by openmp while on GPU using CUDA.

ii) Colors used in parallel algorithms are either equivalent or higher then Sequential algorithms and also in between different threads time taken is not monotonous. So,try to find more about these issues like how to use lesser colors in parallel coloring.
iii) Visualization of the project would have given better understanding on how the project works.

CONCLUSION

In this report, we covered Sequential and parallel implementations of Graph Coloring using D1 Coloring. On the evaluation results of with different graphs as well as different no .of threads we observed that the conflicts increase with concurrency in parallel program and the number of colors used also increases in the parallel implementation. Also parallel performs significantly better than Sequential but on observation from the output graphs, the change in time by increasing the number of threads is completely random and depends on the number of collisions in parallel implementation.

REFERENCES

[1] Input files:https://sparse.tamu.edu/
[2]http://www.academia.edu/2479839/Applications_of_Graph_Coloring_in_Modern_Computer_Science
[3]http://read.pudn.com/downloads161/ebook/733562/GSM/GSM_chap3.pdf
[4]A. H. Gebremedhin and F. Manne, "Scalable parallel graph coloring algorithms", *Concurrency: Practice & Experience*, vol. 12, no. 12, pp. 1131-1146, 2000.
[5]H. C. Edwards, C. R. Trott and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns", *J Parallel Distrib Comp*, vol. 74, no. 12, pp. 3202-3216, 2014.
[6]T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection", *ACM Trans Math Softw*, vol. 38, no. 1, pp. 1-25, 2011.
[7][1] Deveci, Mehmet Boman, Erik D. Devine, Karen Rajamanickam, Siva. (2016). Parallel Graph Coloring for Manycore Architectures. 892-901. 10.1109/IPDPS.2016.54.
[8]M. T. Jones and P. Plassmann, "A parallel graph coloring heuristic", *SIAM J Sci Comput*, vol. 14, no. 3, pp. 654-669, 1993.
[9]M. Luby, "A simple parallel algorithm for the maximal independent set problem", *SIAM J Comput*, vol. 15, no. 4, pp. 1036-1055, 1986.
[10]Parallel Graph Coloring for Manycore Architectures:https://ieeexplore.ieee.org/abstract/document/7516086