Abstract: One of the well-known property of graph is graph coloring. Any two vertices of a graph are different colors such that they are adjacent to each other. The objective of this paper is to analyse the behavioral performance of Tabu Search method through serial and parallel implementations. We explore both parallel and serial Tabu search algorithm for graph coloring with arbitrary number of nodes.

Index Terms: Algorithm analysis, Graph coloring, Parallel processing, Tabu search.

I. INTRODUCTION

A graph is an illustrated representation of different sets of objects and the links between these objects. These links together give an abstract representation of relationships. In graph theory, the objects are called vertices, whereas the links in between them are called edges. A vertex is defined as an entity and on the other hand the edge, is said to be the relationship or association between these two entities. Vertex coloring is the most common graph coloring problem. The problem starts off simply, you have m colors and need to find a way to color the vertices of your graph in such a manner, where no two adjacent vertices connected by an edge have the same color. The minimum number of colors that one completes coloring a Graph G, is called the chromatic number. Other examples of graph coloring types include Edge Coloring – no vertex is incident to two edges, which have the same color, and Face Coloring – Geographical Map Coloring. However, we choose to avoid using these for our analysis as we are planning to work with vertices which are easier to calculate and visualize when speaking about a classroom arrangement or any such application. In addition, the aforementioned coloring techniques can be transformed into Vertex Coloring using various matrix operations. Various algorithms are used to find the Chromatic number for any Graph. They use different approaches and provide varying results depending on the graph size. In this paper, we evaluate the behavioral performance of Tabu Search method through serial and parallel implementations. The number of cores and graph size would be determining factors for each algorithm. The structure of the paper as follows: Section 2 presents the relative study of the work. Section 3 discusses the graph coloring algorithm and section 4 produces the implementation results and its discussion. Section 5 conclude the paper.

Revised Manuscript Received on July 09, 2019.

Rutanshu Jhaveri, School of CSE, VIT University, Vellore, India.
Narayanan Prasanth, School of CSE, VIT University, Vellore, India.
K.Jayakumar, School of CSE, VIT University, Vellore, India.
K.Navaz, Dept. of CSE, Annamacharya Institute of Technology and Sciences Tirupathi, India

II. RELATED WORKS

Allwright et al. [1] presented some parallel graph coloring algorithms dependent on understood sequential heuristic calculations, and contrast them and some existing parallel algorithms. These calculations are actualized on both SIMD and MIMD parallel models and tried for speed, effectiveness, and for shading arbitrary triangulated networks and diagrams from sparse matrix. Buhua Chen et al. [2] introduced a new parallel genetic algorithm to take care of the Graph coloring problem (GCP) in view of Computer Unified Device Architecture (CUDA). All the operators such as initialization, crossover, mutation and selection are designed to be parallel in threads. Additionally, the execution of their algorithms is contrasted and alternate algorithms utilizing benchmark charts, and exploratory outcomes demonstrates that their calculation merges significantly more rapidly than different calculations and accomplishes focused execution for solving GCP. Frank et al. [3], the paper portrays another graph coloring algorithm, the Recursive Largest First (RLF) coloring algorithm. Adding onto RLF, an assortment of existing coloring methods are introduced and their execution on a wide scope of test data is contrasted with that of the RLF algorithm. Additionally portrayed is a methodology for producing arbitrary graphs with known chromatic number. The presence of such a technique, until now ailing in the test writing, gives a standard strategy to testing the exactness of graph coloring algorithms. Gend Lal et al. [4] describes an efficient algorithm for GCP colouring problem using less number of colours. The proposed scheme is applicable for all types of graphs. The algorithm divides the neighbours of a vertex into two categories N-type and V-type, and checks the colour filled in V-type neighbor before filling in the current vertex. Algorithm selects a colour from the list of colours, K every time from the beginning of the list colour so that it can make use less number of colours. They also compare their results with those obtained using genetic algorithms, Brown’s algorithm and other heuristics algorithms. Michael Elkin et. al. [5] initiate the study of combinatorial algorithms for Distributed Graph Coloring problems. In a distributed setting a communication network is modeled by a graph $G = (V, E)$ of maximum degree $\Delta$. The vertices of $G$ host the processors, and communication is performed over the edges of $G$. The goal of distributed vertex coloring is to color $V$ with $(\Delta + 1)$ colors such that any two neighbors are colored with distinct colors. Currently, efficient algorithms for vertex coloring that require $O(\Delta + \log^* n)$ time are based on the algebraic algorithm of Linial that employs set-systems. Evtugnev et. al. [6] shown that certain sequential coloring algorithm heuristics like largest-first (LF), smallest-last (SL), and
Parallel and Serial Graph Coloring Implementations with Tabu Search Method

Any algorithm that can color the set of vertices in parallel such that no two vertices are in parallel, then it is termed as Parallel Graph Coloring Algorithm. One of the well-known algorithm for graph coloring problems is Tabu search method [13].

A. Tabu Search Method

Tabu search, created by Fred W. Glover in 1986 and formalized in 1989, is a local scan strategy utilized for scientific improvement. Nearby inquiries take a potential answer for an issue and check its prompt neighbors (that is, arrangements that are comparable aside from a couple of minor subtleties) in the desire for finding an enhanced arrangement. Nearby hunt strategies tend to wind up stuck in imperfect locales or on levels where numerous arrangements are similarly fitted. Tabu pursuit improves the execution of these procedures by utilizing memory structures that depict the visited arrangements or client gave sets of guidelines. In the event that a potential arrangement has been recently visited in a certain short-term period or on the off chance that it has damaged a standard, it is set apart as "unthinkable" with the goal that the calculation does not think about that plausibility more than once.

B. Performance Study

We started with the analytical study of parallel processing on Tabu Search Algorithm. Based on our hypothesis, we assumed that parallel processing would be much faster and cause the coloring to have a shorter duration. Pj2 library [14] was used for parallelizing the algorithms on certain class of graph. Parallel Java 2 (PJ2) is an API and middleware for parallel programming in 100% Java on multicore parallel computers, cluster parallel computers, hybrid multicore cluster parallel computers, and GPU accelerated parallel computers [14]. It also includes a lightweight map-reduce framework.

Parallel Processing

The Tabu search algorithm implemented with several factors into consideration such as number of nodes in test input, number of cores for each coloring process, whether the task file was for parallel or sequential graph coloring. Total time (ms) taken for the given graph is measured by the number of nodes and cores. Below are the screenshots which depict the obtained results for different node size. Then, we tabulated these results and tried to find relationships with various parameters aforementioned.
5-Node Graph

Core = 1, Node number = 5,
Time Taken = 15ms, Colors required = 3
Fig. 1 provides the result of the graph coloring using Tabu search method whose graph node size is 5. Total number of cores used during implementation is 1 and total number of color required is 3. The above scenario took 8ms for the completion of the coloring process.

Cores = 2, Node number = 5,
Time Taken = 8ms, Colors required = 3
Fig. 2 provides the result of the same graph coloring process which is implemented with 3 cores and total number of color required is 3. It also took 8ms for the completion of the coloring process.

Cores = 4, Node number = 5, Time Taken = 13ms, Colors required = 4

Fig. 3 provides the result of the graph coloring using Tabu search method whose graph node size is 5. Total number of cores used during implementation is 4 and total number of color required is 4. The above scenario took 13ms for the completion of the coloring process. We did not exceed number of cores more than four because then we were getting anomalous results with negative numbers as colors.

Fig. 1 Total time taken for graph coloring with node size 5 number of core used for implementation is 1

Fig. 2 Total time taken for graph coloring with node size 5 number of cores used for implementation is 3

Fig. 3 Total time taken for graph coloring with node size 5 number of core used for implementation is 4

Fig. 4 A 10-Node Graph

Fig. 1.5 depicts a graph with 10 nodes. In the subsequent section, performance of graph coloring (with node size 10) through parallel processing is computed and the results are depicted as screenshots.

Core = 1, Node number = 10,
Time Taken = 103ms, Colors required = 3

Cores = 4, Node number = 5, Time Taken = 13ms, Colors required = 4
Parallel and Serial Graph Coloring Implementations with Tabu Search Method

Fig. 5 Total time taken for graph coloring with node size 10 number of core used for implementation is 1

Fig. 5 shows the result of the graph coloring with node size 10 and is implemented with 1 core. Total number of color required is 3 and takes 103ms for the completion of the coloring process. Fig. 6 shows the result of the graph coloring with node size 10 and is implemented with 2 cores. Total number of color required is 3 and takes 105ms for the completion of the coloring process.

Cores = 2, Node number = 10,
Time Taken = 15ms, Colors required = 3

Fig. 6 Total time taken for graph coloring with node size 10 number of core used for implementation is 2

Fig. 6 shows the result of the graph coloring with node size 10 and is implemented with 4 cores. Total number of color required is 4 and takes 7ms for the completion of the coloring process.

Cores = 4, Node number = 10,
Time Taken = 7ms, Colors required = 4

Fig. 7 Total time taken for graph coloring with node size 10 number of core used for implementation is 4

Fig. 7 shows the result of the graph coloring with node size 10 and is implemented with 1 core. Total number of color required is 4 and takes 9ms for the completion of the coloring process.

Cores = 4, Node number = 10,
Time Taken = 9ms, Colors required = 4

Fig. 8 depicts a graph with 20 nodes. Now, performance of graph coloring (for node size 20) through parallel processing is computed and the results are depicted as screenshots.

Core = 1, Node number = 20,
Time Taken = 9ms, Colors required = 4

Fig. 10 shows the result of the graph coloring with node size 20 and is implemented with 1 core. Total number of color required is 4 and takes 9ms for the completion of the coloring process.

Cores = 4, Node number = 20,
Time Taken = 7ms, Colors required = 4

Figure 8 20-NODE Graph

Retrieval Number: B1840078219/19@BEIESP
DOI: 10.35940/IJRTF.B1840079219
Fig. 9 Total time taken for graph coloring with node size 20, number of core used for implementation is 1
Cores = 2, Node number = 20,
Time Taken = 7ms, Colors required = 4

Fig. 10 Total time taken for graph coloring with node size 20, number of core used for implementation is 2
Cores = 4, Node number = 20,
Time Taken = 7ms, Colors required = 5

Tabu Search Algorithm – Sequential Processing
Below are the screenshots which shows the results of graph coloring by Tabu Search method through sequential processing. We were shocked to realize that most of the sequential processes gets completed in the time period of around 0ms-1ms.
Core = 1, Node number = 5,
Time Taken = 0ms, Colors required = 3

Fig. 11 Total time taken for graph coloring with node size 20, number of cores used for implementation is 4
Fig. 6 shows the result of the graph coloring with node size 20 and is implemented with 4 cores. Total number of color required is 5 and takes 7ms for the completion of the coloring process.
Parallel and Serial Graph Coloring Implementations with Tabu Search Method

Core = 1, Node number = 10,
Time Taken = 1ms, Colors required = 3

Core = 1, Node number = 20,
Time Taken = 0ms, Colors required = 4

Table 1 Consolidated results of Tabu parallel search

<table>
<thead>
<tr>
<th>GraphColSmp</th>
<th>Nodes In Graph</th>
<th>Cores</th>
<th>Time/ms</th>
<th>Colors Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>15</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>8</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>13</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>103</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>15</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 13 Sequential process of the graph coloring process with 10 nodes and 1 core

Fig. 14 Sequential process of the graph coloring process with 20 nodes and 1 core

Fig. 15 shows the graph of Time v/s Cores required for computation. It is clear that the time is inversely proportional to number of cores. So, more the cores for bigger graph reduces the computation time required. We have ignored the number of colors required out here since we only want to visualize the relationship between cores and time. The above bar graph considers all the parameters such as colors required, time and cores. The optimum balance between all these parameters is set when the number of core is two. In Table 2, we see that Sequential process takes almost negligible amount of time and this proves our hypothesis as false. We understood that the sequential processing may take more time because of overhead. In general, when people make sweeping statements about computer performance, there are far more variables at play here, and you can't really make that assumption. For example, inside your for loop, you are doing nothing more than Math.Pow, which the processor can perform very quickly. If this is an I/O intensive operation, requires each thread to wait for a long time, or even if it were a series of processor-intensive operations, you would get more out of parallel processing (assuming you have a multi-threaded processor). But as it is, the overhead of creating and synchronizing these threads is far greater than any advantage that parallelism might give you.
IV. CONCLUSION

In this paper, we discussed both serial and parallel implementation of Tabu search algorithm for a class of connected graphs. The performance of Tabu parallel search algorithm for graphs is comparatively better than the Tabu serial algorithm. In addition to that, the performance gets improved when the number of nodes keep increasing. On the other hand, Tabu serial search algorithm offer better performance when the size of the graph is relatively small. We can extend this study to all class of graphs with suitable restrictions.

REFERENCES