Implementation of a parallel method for the graph coloring problem using MPI and verification of the number of processors on it

Conference Paper · May 2013

DOI: 10.1109/IKT.2013.6620033

3 authors, including:

Hassan Jafari
Somaliland University of Technology, SUTECH (Hargeisa, Somaliland)
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Raouf Khayami
Shiraz University of Technology
32 PUBLICATIONS **555** CITATIONS
SEE PROFILE

Implementation of a parallel method for the graph coloring problem using MPI and verification of the number of processors on it

H. Jafari, A. Alinezhad, S.R. Khayami
Department of Computer Engineering and IT, Shiraz University of Technology
Shiraz, Iran
{Ha.jafari, A.Alinezhad, Khayami}@sutech.ac.ir

Abstract—**In this paper, a parallel version based on a pipeline of innovative methods for graph coloring in a sequential mode is presented. The results of the execution time for parallel implementation in the C++ language using the Message Passing Interface (MPI) on a multi-processor machine is presented and discussed. This problem also has been implemented in a fully sequential and recursive mode, in the same environment, and the results of comparing them to parallel implementation have been reported.**

Keywords- graph coloring; parallel algorithm; message passing interface; sequential first fit algorithm

I. INTRODUCTION

Since the graph coloring is a well-known and important problem and also solve of it in the polynomial time order is not possible, and also due to its numerous applications in science and industry, finding a quick and proper coloring of a graph, often is a crucial step in the development of efficient parallel algorithms for many applications in science and engineering. For example in computing derivatives [1], testing of printed circuits [2], parallel computing of numerical problems [3], allocating registers [4] and optimization problems [5, 6, 7] graph coloring can be used. Thus, many works have been done by the researchers to improve the time order of it.

Graph coloring problem is so that there is a number of colors and a number of vertices, where the vertices are linked together by edges. Each vertex may have some or all colors. We want to color the vertices with available colors (vertices that are linked together) such that they haven't the same color.

The idea of this paper is to introduce a parallel version of a known sequential graph coloring that can be implemented by using of a programming language, for example C++, on a multi-processor system using the MPI. Message Passing Interface is a library of subprograms that can be invoked by some programming languages, such as FORTRAN77, C, and Java. Message passing interface's task is to communicate between computers and is a standard for parallel programming.

The rest of this paper is organized as follows: Section 2 discusses related works. Section 3 briefly introduces first fit algorithm and illustrative example. In section 4 the parallelization of this method is described. In section 5 the implementation of parallel method by using pipelining in C++ on a multi-processor system is presented. Section 6 presents some experiments to show the comparisons and concludes the paper.

II. RELATED WORKS

Among the works that have been done in this area the paper [8] entitled "Scalable Parallel Graph Coloring Algorithms" can be referred. In this paper, parallel version of graph coloring problem has been studied. In this paper, two methods to solve the problem is described that in the first method a simple and fast heuristic which is very suitable for shared memory programming is used that cause approximately a linear speed up in PRAM model. Also, in this paper another way to improve the number of used colors is presented. This method has been implemented using Open MP. Experiments on a SGI Cray Origin 2000 supercomputer with a very large graph and special number of tests to validate the theoretical analysis of the time are made. The first algorithm is a fast method of maximum $\Delta+1$ color (Δ is the degree of the graph) that can provide graph coloring. The relatively fully graphs, the number of colors used by this method increased if the number of processors increases.

The second method is relatively slower, but speed up is acceptable, and in comparing with the first algorithm improves the number of used colors. Also despite increasing the number of processors, the number of colors used is more stable than the first method. According to tests, the number of colors used in this method is comparable with sequential IDO[1] method. This case is one of the

[1] Incident Degree Ordering

cases that using Open MP better than message passing, that is discussable. The implementation of the algorithm using message passing is a complicated task, while using Open MP makes view of a lot of communications hidden.

Another paper has been worked on the parallelization of graph coloring is [9]. The paper entitled "Graph Coloring on Coarse Grained Multicomputers". In this paper, an efficient and scalable algorithm is proposed can coloring a graph with at most Δ+1 color. The algorithm is presented in two forms: randomized and deterministic. In this paper, it is shown that the time order of average case of randomized form is O(|E|/p) and the communication cost order of worst-case of deterministic form is O(|E|).

In many papers graph coloring algorithm is used for other applications, such as to divide the matrix. For example, in the papers [4], [5] and [6] these methods have been introduced. A comparison of parallelization methods of graph coloring has been done in [11]. Also in [12] graph coloring problem using hybrid parallelism, shared memory and distributed, has been expressed.

III. THE FIRST FIT SEQUENTIAL ALGORITHM

In the section, sequential version of first fit algorithm for solving the graph coloring problem is presented. For a graph G = (V, E) with vertices V = $\{v_1, \ldots, v_n\}$ and edge E ⊆ V * V, a function f: V → {1, ..., k}, v ∈ V → f (v) is called a coloring function (vertex) of graph G, if for every pair of vertices u, v ∈ V, u ≠ v: $(u, v) \in E \Rightarrow f(u) \neq f(v)$. If the value of f (v) is called vertex color, graph coloring means that each color should be assigned to a vertex v ∈ V, such that neighbor vertices have different colors. Lowest k that f: V → {1, ..., k} colors the graph G = (V, E) is called the chromatic number of a graph[10].

Graph coloring is a problem that should be solved in different problems such as scheduling, register allocation, print circuit testing, etc. Unfortunately, finding a minimum number of colors for coloring an arbitrary graph (as known so far) in polynomial time complexity is not solvable. Therefore the goal of coloring algorithms called heuristic is to find an answer to the question although cannot primarily provide an optimal solution but its running time has polynomial complexity.

First fit algorithm is a well-known heuristic algorithm to color the vertices of a graph with polynomial time complexity. This algorithm requires the vertices of the input graph be sorted. After that, the first color (color with number of one) is assigned to the first vertex, while the others are processed in the sequential form and the color at each vertex can be assigned to it, if there is not any interference with previously painted ones.

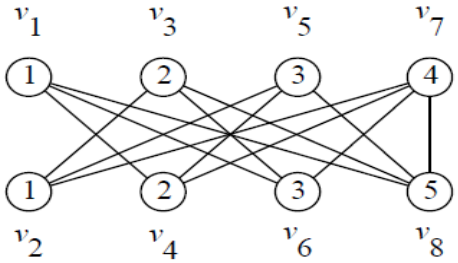

Figure 1. First fit algorithm for 8 vertices graph

Fig. 1 shows a graph with 8 vertices, that coloring algorithm has produced five colors for coloring it. Name of each vertex has been written outside the circle and the color generated by the algorithm has been written inside the circle. Initial sorting of the vertices has been done by using of the index of vertices, i.e. ith vertex in the sequence isv_i.

IV. PARALLELIZATION OF FIRST FIT ALGORITHM

Although first fit algorithm due to its sequential nature, often is called inherently sequential, but here a parallel version is presented that achieves proper speed up on a parallel machine in practice.

A. *Basics*

To get an idea of how the proposed parallel algorithm work, possible implementations of first fit algorithm is reviewed. Coloring the vertices v_i can be implemented in two main phases:

1. A list of all the possible colors for vertex v_i is determined, i.e. the colors that used forv_j, that $j < i$ and v_i is the neighbor of them are excluded. It could be implemented by using of a logical array L_i that is called possibility list; so: $L_i[k] = false \iff \exists v_j; j < i, (v_i, v_j) \in E, f(v_j) = k.$
2. The minimum possible number of colors for vertex v_i is determined, i.e. finding smallest entry inL_i such that $L_i[k] = true$ and assigning color k to vertexv_i.

So, for coloring a graph with n vertices, following operations will be done:

color $(L_i, v_i); i = 1, \ldots, n$ and

build $(L_i, v_j); i = 1, \ldots, n$, $j = 1, \ldots, i - 1.$

These operations cannot be performed completely in parallel because of the time dependence in accessing to possible lists:

1. For all$j < i$, build (L_i, v_j) should be before color (L_i, v_i).

2. Color (L_i, v_i) must be executed before build (L_j, v_i) for$j < i$.

B. First parallel method

This operation has been distributed on n processors and a parallel algorithm that totally needs 2n-1 steps has been adopted.

1. For $1 < i < n$ during the odd steps 2i — 1 the action color (L_i, v_i) is executed on processorP_i;
2. During the even steps 2i for $1 \leq i \leq n$ the actions build (L_{2i-j+1}, v_j) will be executed on processors P_j with$1 \leq j \leq i$.

This algorithm can also be derived from the recurrences implied by the time dependencies [10]. Fig. 2 shows the distribution of the actions over 5 processors for a graph with 5 vertices.

C. General parallel method

In the previous method, there should be a processor per each vertex. The general approach presented here enables to use any number of processor$P_1, \ldots, P_N$ $(1 \leq N \leq n)$. In this case, each processor is assigned to a sub graph with n / N vertex.

The execution of the generalized parallel algorithm is illustrated in fig. 3. It can be see clearly the partition of the graph into N blocks with n / N vertices each. Each processor now has to color all the vertices of its corresponding block under consideration of the possibility lists prepared on the previous processors. This action is done by a procedure named color. The action build (L_i, v_j) used in the generalized version performs the exclusion of the colors of all vertices $v_j = \{v_{1+\frac{(j-1)n}{N}}, \ldots, v_{\frac{jn}{N}}\}$ contained in the jth sub graph from the possibility list L_i of vertex v_i which will be colored later by another processor.

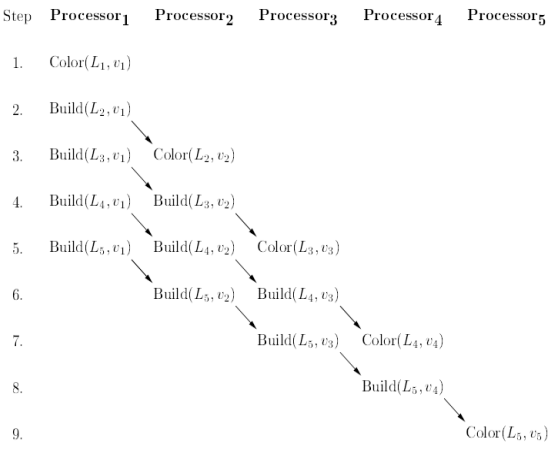

Figure 2. Parallel first fit

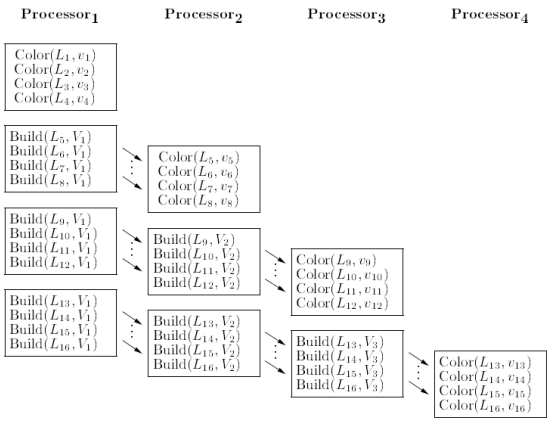

Figure 3. General parallel method

Pipeline behavior of the algorithm due to result of the flow of control over possibility lists is visible in the picture. Compared to the first method, the last stage of the pipeline algorithm in this method cannot be empty.

If the picture of fig. 3 interpreted as a state/time diagram it is recognized that only roughly half of the possible computing resources are used by this algorithm. Therefore the speedup achieved is not expected to significantly exceed half of the number of processors used, but the resulting efficiency of about 50% is not too bad for this type of algorithm.

V. IMPLEMENTATION

The implementation of this method has been done by using Message Passing interface in C++. At first the program is implemented in such way that the number of processors is equal to vertices. Each vertex is assigned to a processor, so the processor has to color corresponding vertex. This early information of each processor in the server including their colors, communications between processors, etc. is sent to them.

Except first processor, each one is waiting to receive updated information from previous one. After that, each according to its color array that has been updated by the predecessor of it starts to color corresponding vertex. After considering the color of its vertex, the array of next vertex will be updated and sends the data to the next processor. Then similarly the array of successor vertices will be updated to the end by it. The first processor after receiving information from the server starts to coloring. If the latest processor succeeded to color its vertex notify the previous one and this notification continued to the first processor.

It should be notified that except latest processor the others after their coloring and sending updated information to the next processors continue to its work until it receive finish message from its successor or finishing its colors. If one processor couldn't to color its corresponding vertex notifying its predecessor and this action performs to the server.

According to the above it can be seen that the graph coloring has been implemented in parallel mode. But the issue here is the communication overhead compared to the computation cost which cannot be easily neglected. Because of the version of the Java implementation thread was used, there wasn't a high price for communication, although it is also has its cost (see [10]). But in this implementation communication cost is high.

So (as described in section 3.3), the number of vertex assigned to each processor increased to a set of vertices to decrease the communication overhead and causing the parallelism more useful. This implementation has one stage more than the previous method. Thus, after each processor has finished coloring of a vertex updating the color array of its remaining vertices. After all of these it updates the color array of its successors and sends their updated information to them. Then it continues to color its remaining vertices.

A. *Execution environment*

The system configuration that has been used is as follow:

CPU: Core i7, 2 GHz
RAM: 8 GB
OS: Windows 7

Implementation environment for message passing interface is in C++ in Visual studio 2010. The number of colors of each vertex, number of each color, number of communications of each vertex and the number of each vertex in each communication is selecting uniformly and randomly.

Purpose of this paper is enabling parallel graph coloring problem by using message passing interface in multiprocessor environment and evaluating this method with different number of processors. This test measures the running time of the program. So the workload of the system is the vertices and the communications (edges) between them. System parameter is the number of processors in the parallel implementation. Parameters of the workload are the total number of vertices, total number of colors, number and colors number of each vertex, how many and number of the vertices associated with each vertex and the number of iterations.

It is noteworthy that the number of colors per vertex should be more than 35 percent of all colors. The number of connections per vertex is also should be a maximum of 40% of the total vertices.

VI. RESULTS AND CONCLUSION

In this section, differences between run time of these parallel methods within different number of processors has been verified. Also run time of graph coloring in sequential and recursive mode has been presented. The results are presented in fig.4. This test has been made for 15, 25, 45, 70, 90 and 150 vertices and for 8, 12, 20, 30, 45 and 85 colors, correspondingly. Each test of related to number of vertices has been repeated 15 times and the average reported in the fig. 4.

As can be seen from the fig. parallelism for graphs with less number of vertices is not better than sequential mode and by increasing the number of vertices performance of parallelization becomes better. In case of 45vertices, because of randomly selection of number of vertices and number of edges, run time is less than 15 and 25 vertices case.

As can be expected the main issue is the communication overhead between processors in MPI environment that implicate higher run time in case of less vertices.

If processors exchange their information between all the others the performance becomes so improper. So, exchanging information has been controlled and decreased. Then information that not change in run time has been exchanged one time that cause to improve run time considerably. This can be expressed as bellow:

$$t_{comm} = t_{start\ up} + mt_{data}$$

where t_{comm} is the communication time between processors in parallel method. Also $t_{start\ up}$ and t_{data} are preparation time and data transfer time between processor, respectively. m indicates exchange times of information between processors, so decrease of it cause to decrease of communication overhead.

So for parallelization in message passing interface within multiprocessor environment two points should be considered:

1. Number of processors to solve a problem must choose according to the problem. This number should be selected so that the computational overhead of the processor be more than of the communication overhead between processors. Otherwise parallelism will not improve the run time of the solution.

2. Message passing interface must be

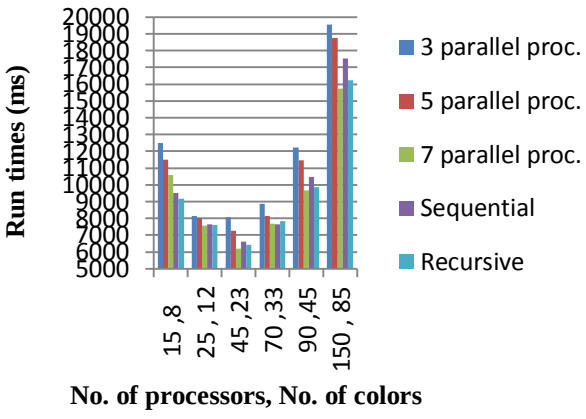

Figure 4. Run time comparisons

used to address problems with large number of entries (number of vertices and etc.).

References

[1] A. H. Gebremedhin, F. Manne, and A. Pothen, "What color is your jacobian? Graph coloring for computing derivatives," SIAM Review, vol. 47, no. 4, pp. 629–705, 2005.

[2] M. Garey, D. Johnson, and H. So, "An application of graph coloring to printed circuit testing," Circuits and Systems, IEEE Transactions on, vol. 23, no. 10, pp. 591–599, Oct. 1976.

[3] A. H. Gebremedhin, F. Manne and A. Pothen, "Graph coloring in optimization," Revisited January 2003.

[4] G. J. Chaitin, "Register allocation & spilling via graph coloring," SIGPLAN Not., vol. 17, pp. 98–101, Jun. 1982.

[5] T. F. Coleman and J. J. More, "Estimation of sparse Jacobian matrices and graph coloring problems," SIAM Journal on Numerical Analysis, vol. 1, no. 20, pp. 187–209, 1983.

[6] A.H. Gebremedhin, F. Manne and A. Pothen, "Parallel distance-k coloring algorithms for numerical optimization," In B. Monien and R. Feldmann (Eds.): Proceedings of the 8th International Euro-Par Conference, Paderborn, Germany, August 2002, volume 2400 of Lecture Notes in Computer Science, pages 912-921. Springer-Verlag 2002.

[7] A.H. Gebremedhin, I.Gu_erin Lassous, J. Gustedt and J.A. Telle, "PRO: a model for parallel resource-optimal computation," In Proceedings of HPCS'02, 16th Annual International Symposium on High Performance Computing Systems and Applications, Moncton, Canada, pages 106-113. IEEE Computer Society Press 2002.

[8] A. H. Gebremedhin and F. Manne, "Scalable parallel graph coloring algorithms," Concurrency: Practice and Experience, 2000.

[9] A. H. Gebremedhin, I.Gu_erin Lassous, J. Gustedt and J.A. Telle, "Graph coloring on coarse grained multicomputers," Discrete Applied Mathematics, to appear.

[10] T. Umland, "Parallel graph coloring using java," Architectures: Languages and patterns, 1998.

[11] J. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, and C. Martin, "A comparison of parallel graph coloring algorithms," Northeast Parallel Architectures Center at Syracuse University (NPAC), Tech. Rep. SCCS-666, 1994.

[12] Sariyüce, A.E., Saule, E., Catalyurek, U.V., "Scalable hybrid implementation of graph coloring using MPI and OpenMP," pages 1744-1753, IEEE Computer Society Press 2012.