Parallelization of Graph Isomorphism using OpenMP

Vijaya Balpande
Research Scholar
GHRCE, Nagpur
Priyadarshini J L College of Engineering, Nagpur

Anjali Mahajan
Professor
Priyadarshini Institute of Engineering and Technology, Nagpur

ABSTRACT
Advancement in computer architecture leads to parallelize the sequential algorithm to exploit the concurrency provided by multiple core on single chip. Sequential programs do not gain performance from multicore. For multicore architectures, OPENMP and MPI are application programming interfaces. They can be used for parallelization of codes. For shared memory architecture OPENMP is used, whereas for distributed memory architecture MPI is used .In this paper, analysis of the performance of parallel algorithm over sequential algorithm is done. In particular, Graph Isomorphism problem based on vertex invariants is considered and parallelized using OpenMP. We demonstrate the performance of Graph Isomorphism using variable size graphs and parallelize it using vertical tiling technique on multicore architecture. Our previous work shows, sequential implementation of modified algorithm based on vertex invariants using Euclidian vector performs better than existing algorithm of Graph Isomorphism based on vertex invariants. To analyze the performance of parallel implementation, we present practical experiments with randomly generated graphs.

Keywords
Graph Isomorphism, Vertex Invariant, Euclidean vector, OpenMP, MPI.

1. INTRODUCTION
In shared memory architecture, system has more than one processor on a single chip. With the advancement in technology of processor architecture and due to advent of multicore architecture many researchers from various fields are concentrating in this area.

In distributed memory architecture, each processor has its own memory. Message Passing Interface (MPI) [2][3] is an API which allows distributed processors to communicate message with each other. MPI provides functions for programmers to distribute data, synchronize and create virtual topologies for communication among processes.

OpenMP and MPI are the two programming paradigms for parallelization. Both of them provide high performance with different approaches. OpenMP is used for shared memory architecture whereas MPI is used for distributed memory systems. OpenMP has simple interface which can be used to parallelize the loops. OpenMP [1] is an application program interface for shared memory programming model. Generally OpenMP directives are easy to use for converting sequential code to parallel code. In shared memory model, most of the thread handling is done by compiler which reduces the code complexity. Programmer has flexibility of declaring the variable as private or shared and which part of sequential code to be parallelized. Due to this OpenMP is widely used for parallelizing the sequential code for shared memory architecture.

Graph isomorphism is a way of matching the two graphs whether they are equivalent or not. There is complete structural equivalence between the two graphs. They differ only in the names of vertices and edges. In Graph Isomorphism there is one to one mapping of vertices and edges of two graphs. It is highly studied problem in research field and graph theory. Graphs are widely used in real life applications to represent the structure of objects, e.g. applications like molecules, images, networks. Graph isomorphism problem is extensively applied in many applications in various fields such as data mining [4], pattern recognition [5], information retrieval [6], chemistry [7].Most researchers believe that GI problem is not NP-complete. As there is no polynomial solution for GI it is not known to be in P or to be NP- complete.

Graph matching can be done in two ways. In the first case given two graphs, graph matching is done for isomorphism or to find if one graph is subgraph of the other graph. Another method is, given a database of graph called model graph, an input graph is matched against it to detect the graph or subgraph isomorphism.

Graphs are generally representing information using vertices and edges. Graph consisting of thousands of nodes and vertices requires large processing time. Sequential processing of larger graph is very time consuming [8][5] and matching of two graphs is computationally expensive process .Therefore, there is a need of methodology that could reduce the computational time while matching the graph. In this paper, parallelization of sequential graph isomorphism algorithm [8] which is referred as Algorithm A1 is compared with modified algorithm which is referred as Algorithm A2. Specifically, an evaluation of sequential and parallel implementation of existing algorithm A1 and algorithm A2 is provided in terms of execution time.

In the remaining paper, section II describes the basic definitions and notations. Section III describes the basic idea of the algorithm A1. In section IV parallelization of algorithm A1 and modified algorithm A2 is described. In Section V experimental results were given. Section VI specifies the conclusion.

2. BASIC DEFINITIONS
2.1 Definition1
In[8],the graph isomorphism is expressed as: Given two graphs G1=(V1,E1) and G2=(V2,E2) ,if there exist 1 to 1 mapping function f from V1to V2 such that (i, j) ∈ E1, if and only if (f(i) ,f(j)) ∈ E2.The function f is called an isomorphism from G1 to G2. If the two graphs isomorphic to each other, it is denoted by G1 ≅ G2.

2.2 Definition2
The identity matrix M of order n x n is represented as

$$m_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Where, m is an element of M on the i^{th} row and j^{th} column.

2.3 Definition3

A permutation matrix is obtained from the identity matrix by any row and column permutation. If M1 and M2 are the two matrices of graph G1 and G2 respectively M1 is said to be isomorphic to M2 if there exists 1 to 1 mapping of function f from the rows of M1 to rows of M2 and from column of M1 to column of M2. The function f is called an isomorphism from M1 to M2. In other words, M1 is isomorphic to M2 if and only if there exists the permutation matrices P1 and P2 satisfying the following relation

$$M1 = P1\ M2\ P2 \qquad (2)$$

3. EXISTING ALGORITHM

In [8], vertex invariant property is used to test graph isomorphism. The vertex invariant property is described with the help of number of vertices, labels of vertices, label of edges and degree of vertices. Vertices having same label and same degree are grouped together and they form one group. Based on these groups, graph matrices are created. Using these matrices permutation operation is performed. Permutation is carried out to change the position of vertices and edges in graph. After performing permutation operation, the structure of original graph should match with permuted graph to detect graph isomorphism. For example, graph G1 has five vertices with each vertex having vertex label and degree. Vertex v_1 has label a and degree 2 which can be represented as $v_1\{a,2\}$.Similarly other nodes can be represented as $v_2\{b,3\}$,$v_3\{a,2\}$,$v_4\{a,2\}$ and $v_5\{b,3\}$.By applying vertex invariant property for graph G_1, vertices can be grouped together as group $g_1\{v_1,v_3,v_4\}$ and $g_2\{v_2,v_5\}$. Graph G1 has 5 vertices and total number of permutation matrices required to be computed are 5! = 120. By using permutation matrices, graph is represented as a decision tree. As size of decision tree is directly proportional to permutation matrices, it requires large amount of storage. The vertex invariant property of graph is used to group the vertices of graph so as to reduce the size of decision tree as compared to [9]. Based on vertex invariant property, Graph G1 requires 3! X 2! = 12 permutation matrices that reduces the size of the decision tree. The technique is similar to breadth pruning technique which reduces the size of decision tree remarkably by maintaining the time complexity as almost equivalent.

Decision tree is the most widely used method for inductive conclusion and simple method for knowledge representation. The basic idea [9] of the isomorphism algorithm is that all possible permutation of adjacency matrix of each of the model graph was computed offline and the permutation matrices were represented as decision tree. The matrix of input graph is matched to those of adjacency matrices in the decision tree which are identical to it. At run time, permutation matrices of the input graph are matched with the preprocessed matrices of model graph to detect the graph or subgraph isomorphism.

Decision tree is represented by using the row-column element of each permutation matrix of the model graph [8][9]. A row-column element xi of n x n matrix is a vector and is represented as xi = (y1i, y2i … yii, yi (i-1) …... yi1). The representation of an adjacency matrix A by its row–column element is illustrated in figure 2. The x_1, x_2 … are the row column element of matrix x. A root node is present at the top of decision tree. At each level of the decision tree the classification is done by comparing the row column element of permutation matrix. In the starting, the classification is done by comparing the first row-column element of the input graph by the first row-column element xi of each permutation matrix. At the nth level of decision tree the classification is carried out by comparing the row-column element xn of the permutation matrices. Graph G3 has 3 vertices and therefore it has 3! = 6 permutation matrices. The row column element of the 6 permutation matrices are then organized as a decision tree [8].

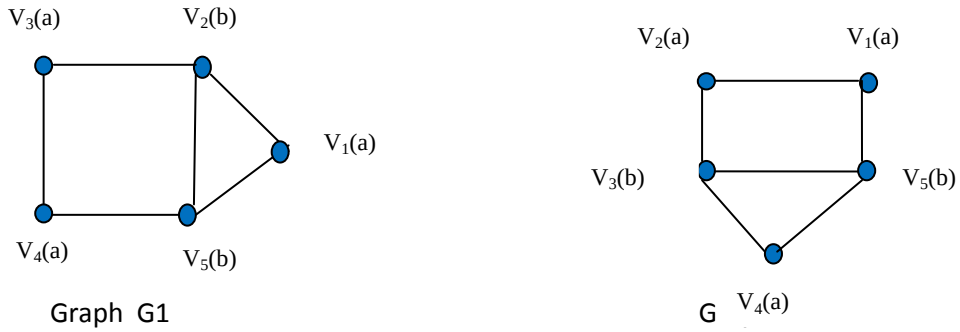

Figure 1: Graph G1 and Graph G2

The decision tree formed is of exponential size depending on the number of vertices and requires huge amount of storage if the number of vertices increases. A graph with n vertices has n! permutation matrices. A row-column element at level n of decision tree would be n! at the worst case. Existing algorithm A1 based on decision tree under the constraint of vertex invariant needed to compute permutation matrices of input graph and compared with the preprocessed permutation matrices of the model graph. The comparison is carried out element by element. Time required for comparing the matrices is more as it needs to compare n x n elements of permutation matrices.

In the proposed algorithm A2 we are using the vertex invariant property like the existing algorithm to find the count of permutation matrices. In our approach instead of computing the permutation matrices for the entire graph we compute the count of permutation matrix. Based on the count of permutation matrices we are computing only one sequence of permutation matrix. For this matrix, compute the Euclidian vector of the input graph and the model graph. If the Euclidian vector of the input graph matches with the Euclidian vector of the model graph, graph isomorphism is detected and the two graphs are said to be isomorphic to each other. Thus by comparing Euclidian vector of matrices we are reducing the time complexity as compared to existing algorithm.

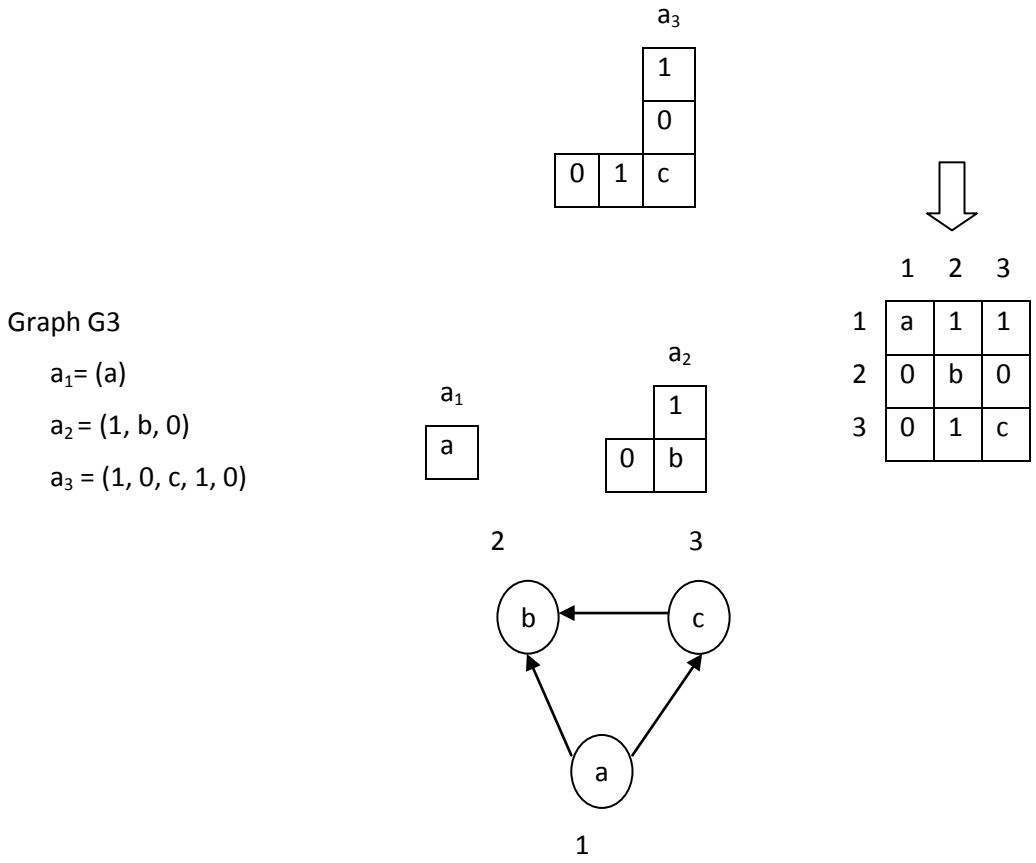

Figure 2: Graph G3 Row-column element

4. PARALLELIZATION OF GRAPH ISOMORPHISM

Parallelization can be implemented as task parallelism or data parallelism. In order to parallelize the sequential algorithm, either task or data can be divided between processors. Task parallelization is carried out by breaking the job and assigning different part of job to specific processor whereas in data parallelization all processor performs same job on different portions of data. Based on type of algorithm and requirement, task decomposition or data decomposition can be applied.

For parallelization of existing algorithm A1 and our algorithm A2, data decomposition parallelization is applied using OpenMP implementation. Here data is divided using vertical tiling mechanism.

Algorithmic process is dependent on the values of Euclidean vectors which in turn dependent on the values of adjacency atrix. For n number of nodes in a graph, the size of adjacency matrix is n*n and the size of Euclidean vector is only n. The size of adjacency matrix increases with the increase in number of nodes. For example**:** for n=1000 (number of nodes), Size of A1*A2 = 1000 * 1000 i.e. Memory for 1000000 elements are required to store all the elements of adjacency matrix whereas only 1000 elements are required for a Euclidean vector. Huge memory is required to hold all elements of adjacency matrix. To avoid this problem of memory, tiling mechanism is used. In this mechanism instead of holding all the elements of adjacency matrix in memory it divides the adjacency matrix in a tiled format and calculates each tile one by one.

Algorithm A2:

Input: Graph G1, G2

Output: To check G1 and G2 are isomorphic or not.

Basic Steps:

1. Calculate Adjacency Matrix A1 and A2 of Graph G1 and G2 respectively.
2. Calculate the values of Euclidean Vector E1 and E2 of graph G1 and G2 respectively.
3. Sort the Euclidean values E1 and E2.
4. Compare both values E1 and E2.
5. Graphs are isomorphic, if values of E1 and E2 are equal.

For parallelization of algorithm, in tiling mechanism we divide adjacency matrix into vertical tiled format as shown in figure 3.

Instead of calculating complete adjacency matrix at a time, the matrix is divided into different vertical tiles as shown in Figure 3, where T0, T1 ….T4 are vertical tiles. In sequential execution, calculation of Euclidean value of each tile is done one by one. In parallel execution by using OpenMP, outer loop is dynamically scheduled with n threads. Iteration of outer loop performs execution in different thread and once the execution of one thread is completed it will dynamically jump to another iteration on which no thread is assigned. For example, if there are 4 threads then calculation of 4 tiles are performed in parallel with these threads.

	v1	v2	v3	v4	v5
v1	x	1	0	0	0
v2	1	x	1	0	1
v3	0	1	x	1	0
v4	0	0	1	x	1
v5	1	1	0	1	x
	T0	T1	T2	T3	T4

Figure 3: Vertical Tiling

5. EXPERIMENTAL RESULTS

Experiments were carried out for both directed and undirected randomly generated graph .The experiment environment is: Intel(R) Core(TM) i3 CPU 540 @ 3.07GHz, Speed: 1,995.00 MHz, Cores: 4, 1.8 GB RAM, Free memory: 426.4 MB (+ 759.7 MB Caches) Free swap: 1.8 GB with Linux (open suse 11.3) OS.

The algorithm is implemented in c++ language. For each experiment we generated one or more model graph. Experimental results were obtained for both labeled and unlabeled graph.

We examine the time complexity experimentally; the time required for graph isomorphism detection using vertex invariant in the existing algorithm A1 is more as compared to our algorithm A2 which is implemented using vertex invariant and Euclidean vector.

For the directed labeled graph, when the number of vertices more than 500, the existing algorithm fails to perform the number of permutations and unable to detect graph isomorphism. In our proposed algorithm A2, Euclidean vector is computed and it is able to detect graph isomorphism for the graph having vertices more than 1000.

Experimental result for the undirected labeled and unlabelled graph also gives better result than the existing algorithm. The existing algorithm fails to compute permutations and unable to detect graph isomorphism for the graph having vertices more than 30 in undirected labeled graph and more than 20 in unlabelled graph.

Experimental result for directed and undirected graph for Algorithm A1 and Algorithm A2 are shown in figures. Limitation of Algorithm A2 is it can detect only graph isomorphism and fails to detect subgraph isomorphism.

6. CONCLUSION

We conclude that we have successfully parallelize an existing algorithm A1 based on decision tree under the constraint of vertex invariant with OpenMP and studied the performance with respect to time for large size randomly generated graphs. We could also parallelize modified algorithm A2 based on Euclidean vector with OpenMP and studied the performance with respect to time for large size randomly generated graphs.

A lot of experiments were performed using OpenMP on algorithm A1 and A2 and we conclude that depending upon size of graph, type of graph and using vertical tiling mechanism on our architecture, algorithm A2 shows better performance than algorithm A1.In future we are trying to reduce the space complexity of parallel algorithm A2.

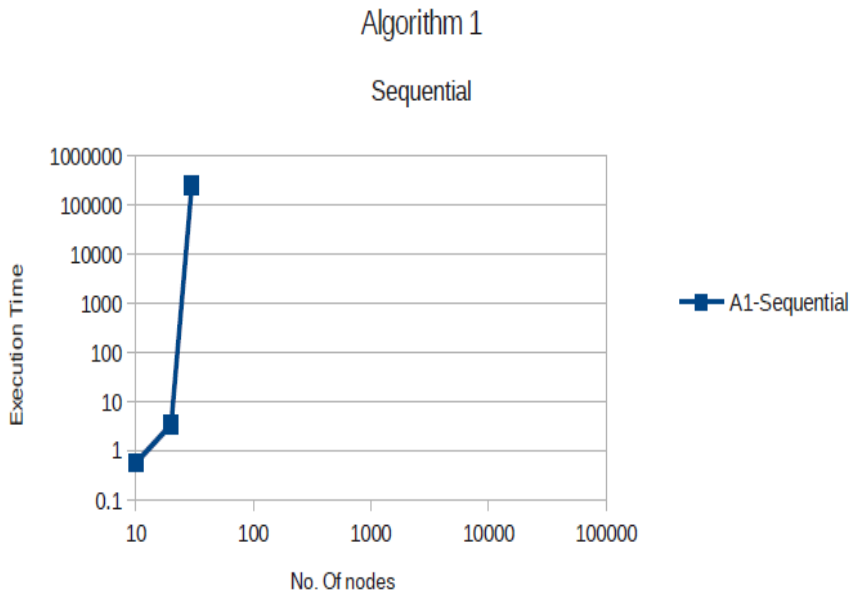

Figure 4: Time required for graph isomorphism detection for undirected labeled graph using sequential algorithm A1

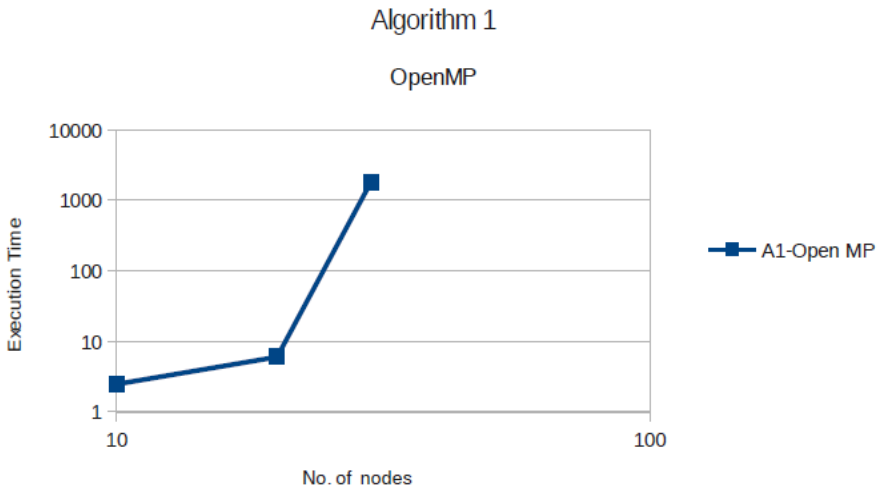

Figure 5: Time required for graph isomorphism detection for undirected labeled graph using parallel algorithm A1

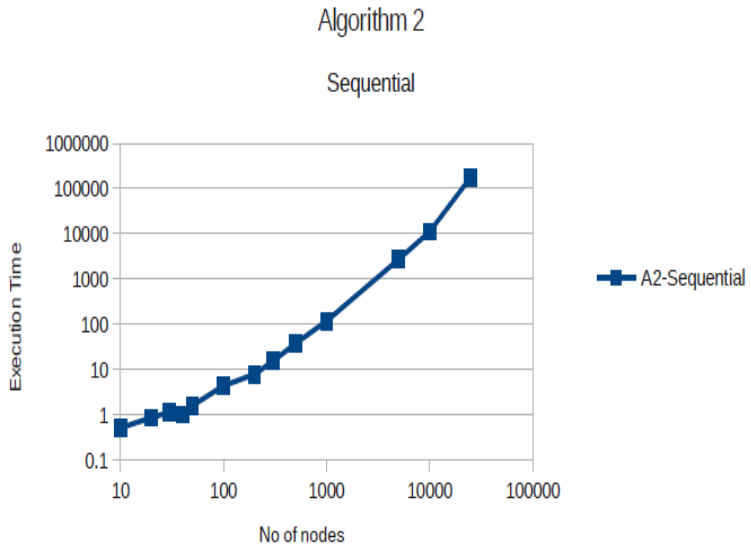

Figure 6: Time required for graph isomorphism detection for undirected labeled graph using sequential algorithm A2

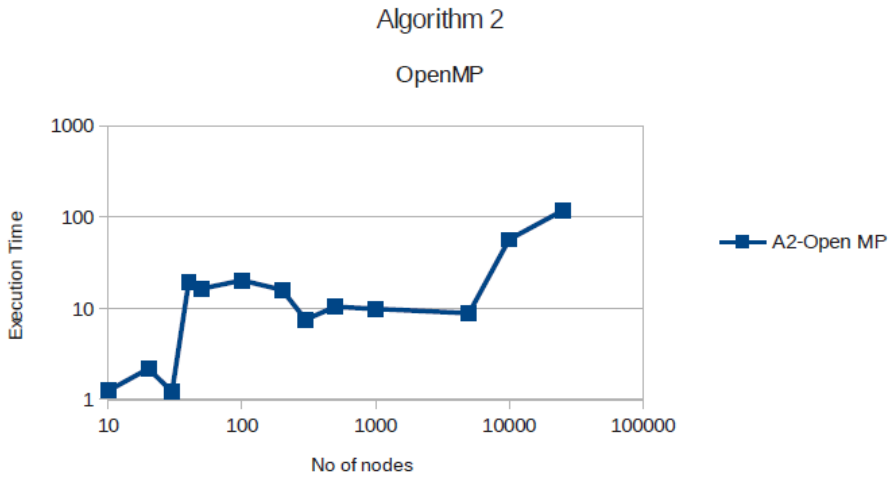

Figure 7: Time required for graph isomorphism detection for undirected labeled graph using parallel algorithm A2

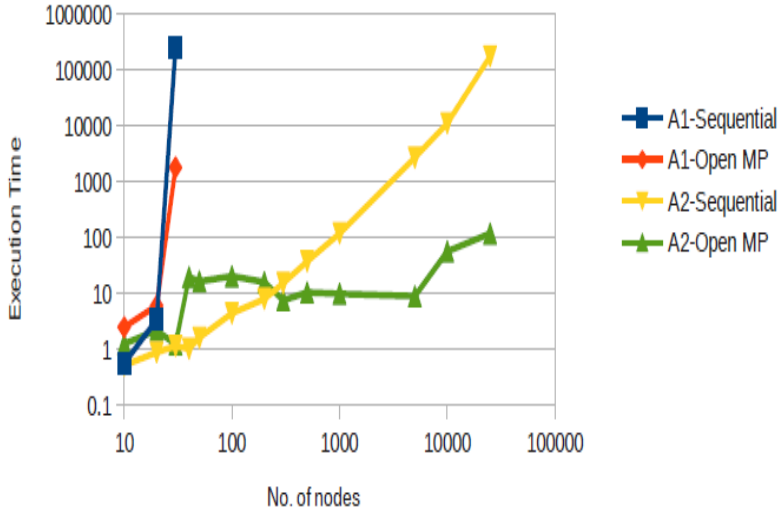

Figure 8: Performance comparison of sequential and parallel algorithm A1 and algorithm A2 for undirected labeled graph

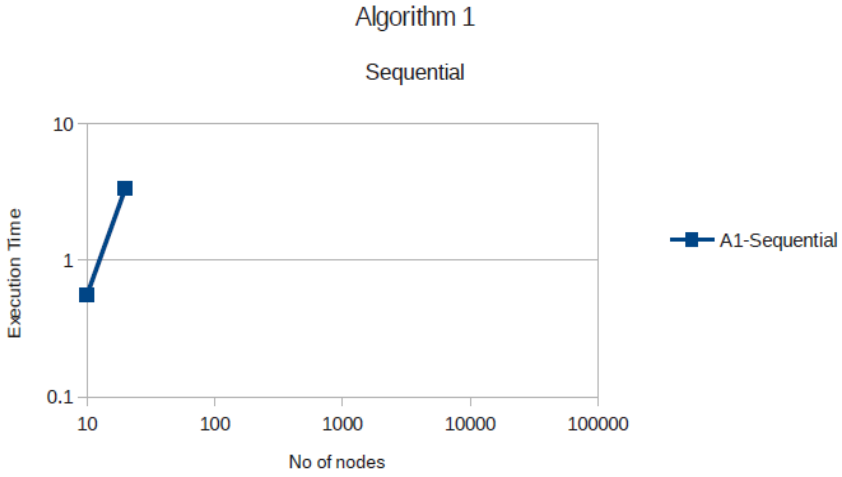

Figure 9: Time required for graph isomorphism detection for undirected unlabeled graph using sequential algorithm A1.

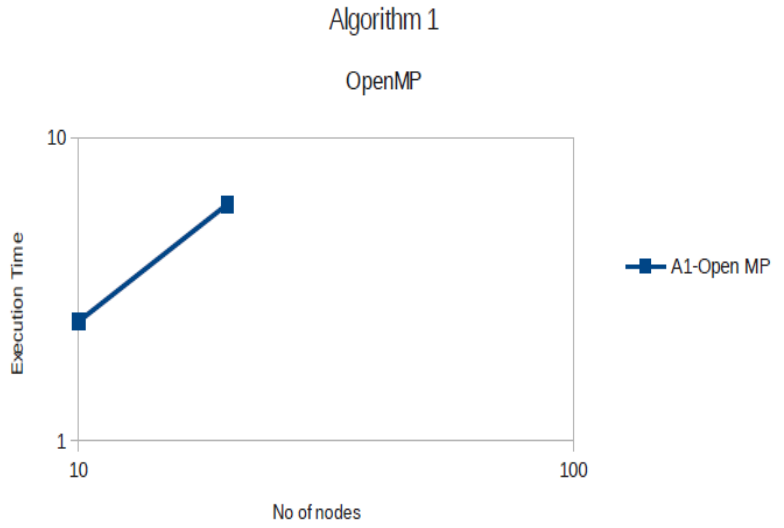

Figure 10: Time required for graph isomorphism detection for undirected unlabeled graph using parallel algorithm A1.

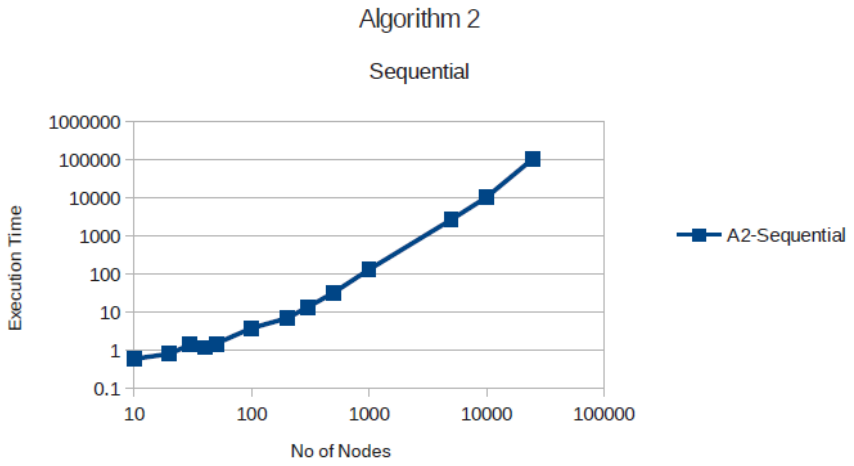

Figure 11: Time required for graph isomorphism detection for undirected unlabeled graph using sequential algorithm A2.

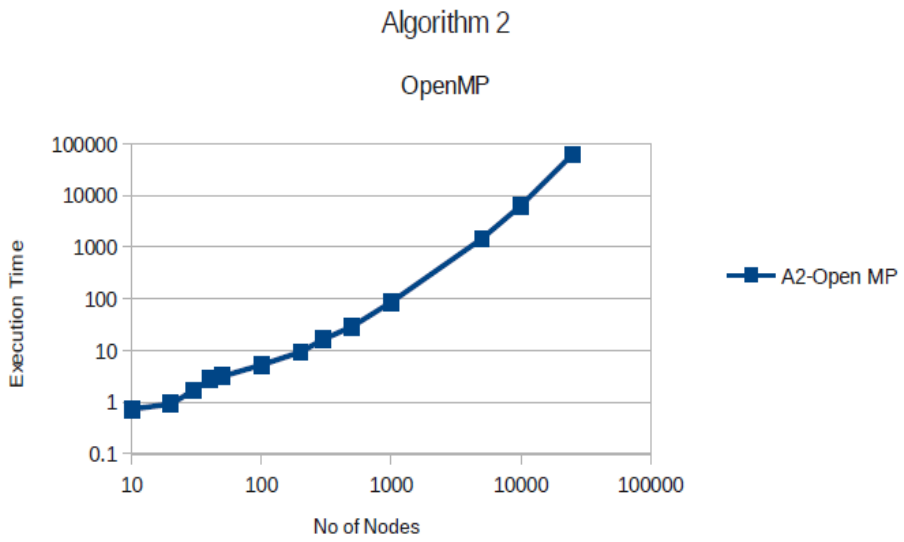

Figure 12: Time required for graph isomorphism detection for undirected unlabeled graph using parallel algorithm A2.

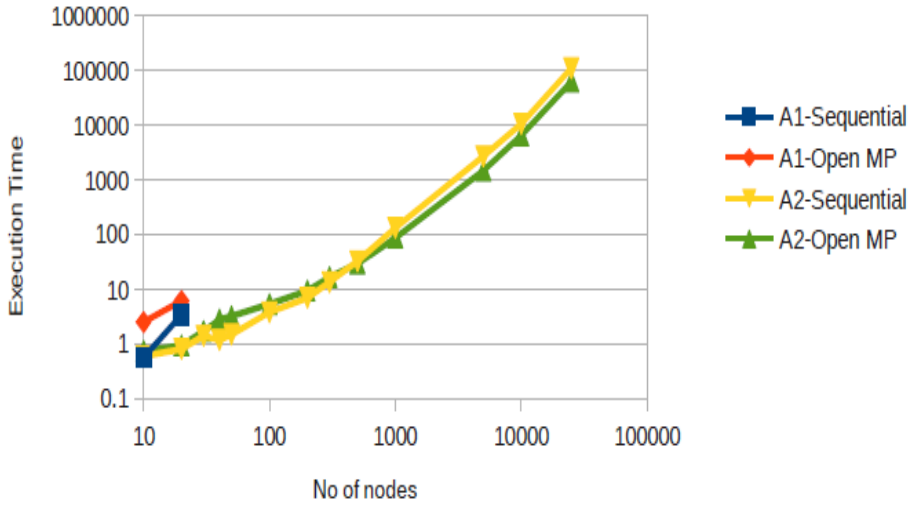

Figure 13: Performance comparison of sequential and parallel algorithm A1 and algorithm A2 for undirected unlabeled graph

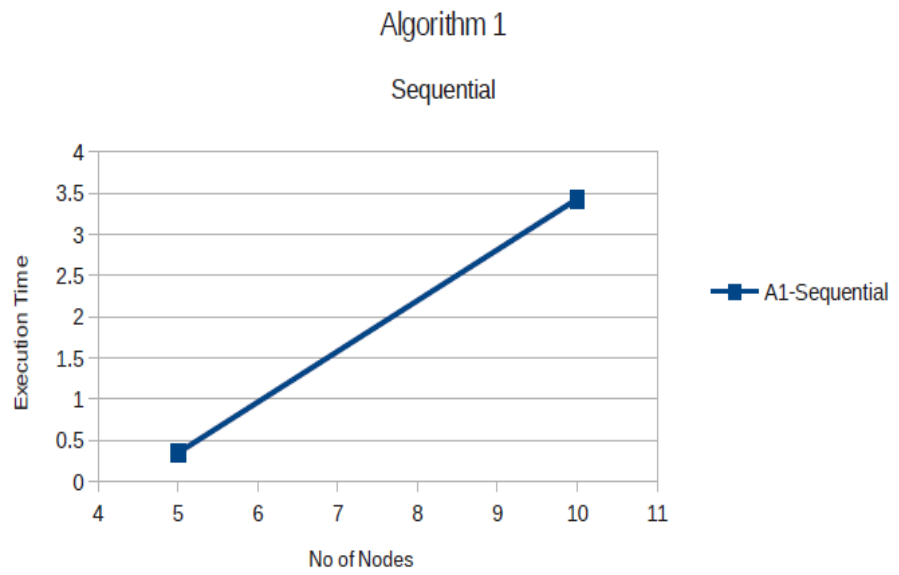

Figure 14: Time required for graph isomorphism detection for directed unlabeled graph using sequential algorithm A1.

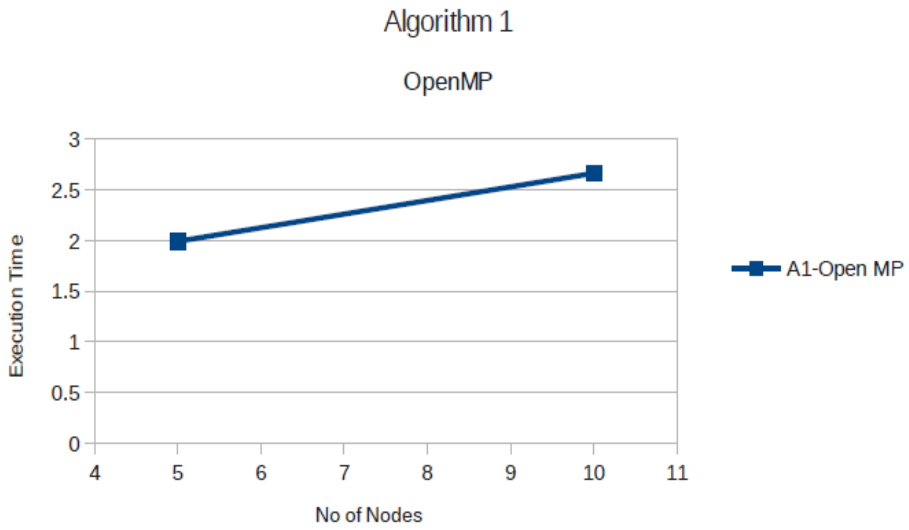

Figure 15: Time required for graph isomorphism detection for directed unlabeled graph using parallel algorithm A1.

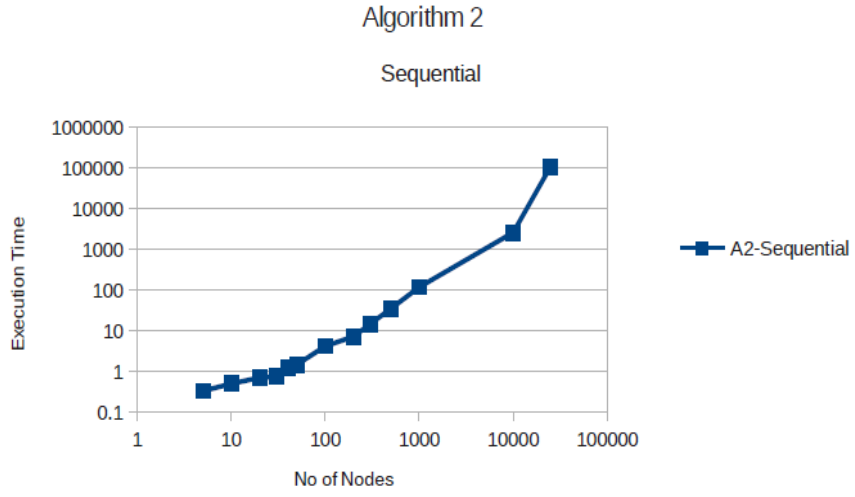

Figure 16: Time required for graph isomorphism detection for directed unlabeled graph using sequential algorithm A2.

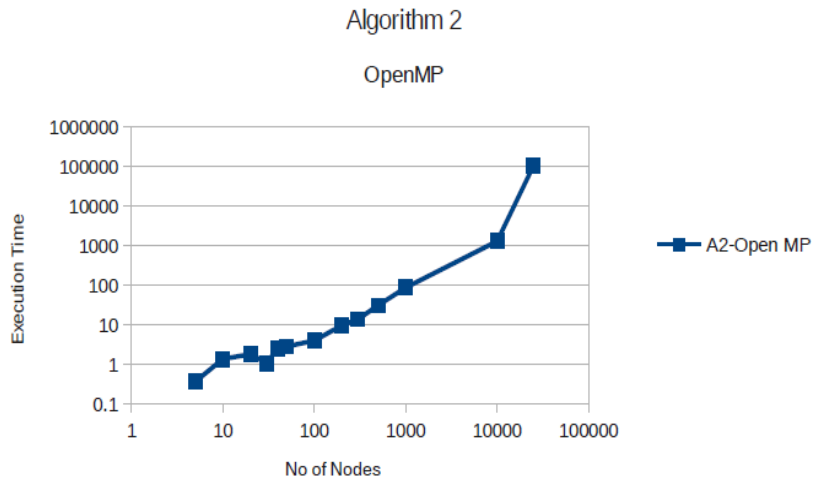

Figure 17: Time required for graph isomorphism detection for directed unlabeled graph using parallel algorithm A2.

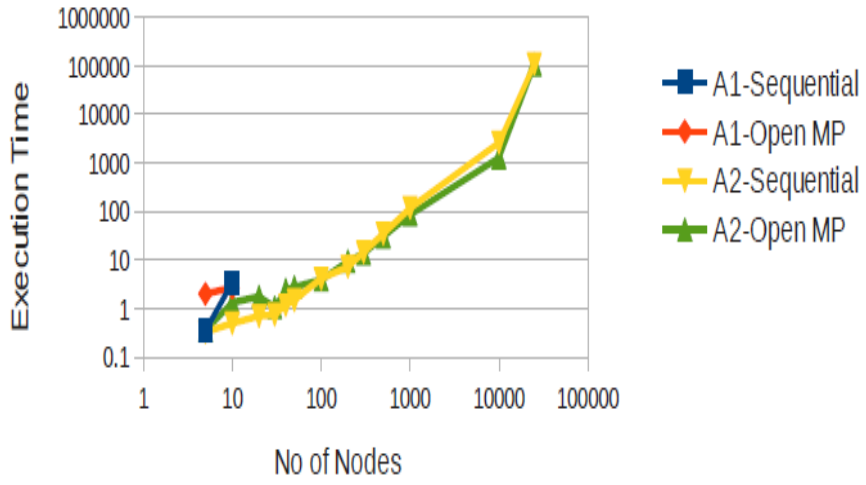

Figure18: Performance comparison of sequential and parallel algorithm A1 and algorithm A2 for directed unlabeled graph

7. REFERENCES

[1] Alejandro Duran, Marc Gonzales, and Julita Corbalán.Automatic Thread Distribution for Nested Parallelism in OpenMP. In 19th ACM International Conference on Supercomputing, pages 121–130, Cambridge, MA, USA, June 2005

[2] The Message Passing Interface (MPI) Standard http://wwwunix. mcs.anl.gov/mpi/ and http://www.mpi-forum.org

[3] Anne C. Elster and David L. Presberg, "Setting Standards for Parallel Computing: The High Performance FORTRAN and Message Passing Interface Efforts", May 1993, Theory Center SMART NODE Newsletter, Vol. 5, No.3. http://www.idi.ntnu.no/ elster

[4] T. Washio and H. Motoda. State of the art of graph-based data mining. ACM SIGKDD Explorations Newsletter, 5(1):59–68,2003

[5] D. Conte and et al. Graph matching applications in pattern recognition and image processing. In International Conference on Image Processing, volume 2, pages 21–24. IEEE, 2003.

[6] A. T. Berztiss. A backtrack procedure for isomorphism of directed graphs. Journal of the ACM, 20(3):365–377, 1973.

[7] J. Braun and et al. Molgen-cid, a canonizer for molecules and graphs accessible through the internet. Journal of Chemical Information and Computer Sciences, 44(2):542–548, 2004.

[8] Ming Qiu , Haibin Hu, Qingshan Jiang and Hailong Hu : A New Approach of Graph Isomorphism Detection based on Decision Tree IEEE, Second International workshop on Education Technology and Computer Science,2010

[9] B.T.Messmer and H.Bunke: A decision tree approach to graph and subgraph isomorphism detection Pattern Recognition 32 (1999) 1979-1998

[10] B.T.Messmer and H.Bunke: Subgraph Isomorphism in Polynomial Time. University of Bern, Institute of Computer Science and Applied Mathematics, Bern, Switzerland Technical Report IAM 1995-003, 1995

[11] Narsingh Deo: Graph Theory with Applications to Engineering and Computer Science ,Prentice Hall,Inc, 1995