FAST MIS - A SIMPLE RANDOMIZED PARALLEL ALGORITHM FOR THE MAXIMAL INDEPENDENT SET PROBLEM

REPORT ON PACKAGE

SUBMITTED BY

SWARNA S **Roll No : 18PT38**

RAMANNAMALAI RM **Roll No : 18PT28**

SUBJECT : OPERATING SYSTEMS

DEPARTMENT OF APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCES

PSG COLLEGE OF TECHNOLOGY

COIMBATORE - 641004

CONTENTS

ABSTRACT

The independent set of a graph G is defined as a set S of vertices such that for every two vertices in S, there is no edge connecting the two. Equivalently, each edge in the graph has at most one endpoint in S.

A maximal independent set is an independent set that is not a proper subset of any other independent set.In other words no vertex can be added to this set satisfying the independent property.

In this report, we explore on developing fast and efficient algorithms for finding the maximal independent set of a given graph G. Our exploration starts by developing an brute-force serial algorithm and ends in the Fast MIS algorithm. Apart from this, we introduce the concept of OpenMP API and demonstrate the implementation of the above mentioned parallel algorithm. At the end, we present you our findings and conclude henceforth.

1.1 INTRODUCTION

Finding the maximal independent set of a graph has always been a problem of interest to many researchers. It is due to the fact that unknowingly, knowledge of the maximal independent set of a graph has various real life practical applications associated with it.

Few of the applications of MIS are mentioned as follows :

1. Traffic light management

    Traffic light was discovered and established in the early 18th century in London. During those days there was no automatic control of traffic lights all over the city as to how it is now. In order to maintain such a system, the government authority people used the concept of maximal independent set to find the traffic hub for all such traffic lights in a particular area .

The network of traffic lights were constructed, treating the traffic lights as the nodes and the edges being a path or road through which we can see another traffic light. The nodes that are present in the MIS can be treated as traffic hubs. The hubs can be used to monitor the whole system. For example any power issues of traffic lights among the city can be easily resolved as we know the traffic hub. Some supervisions such as CCTV and patrols can be set at the hub to control the traffic as well as security of that area.

2. Star topology

   A star network is an implementation of a spoke–hub distribution paradigm in computer networks.A star topology is a topology for a Local Area Network (LAN) in which all nodes are individually connected to a central connection point, like a hub or a switch.

   In some other cases, all computers are connected to a single computer which acts as a head. But it is not advisable since every node depends on a particular node and bringing down that node would result in failure of connection. The concept of MIS can be used here similar to that of the traffic light problem. It will help us to select the servers or computers that are essential to monitor and safeguard our system.

Because of its vital importance, we chose the maximal independent set problem and present in this report, the various algorithms and solutions proposed to solve it. We focus in this report more on optimising and tuning up the performance of the existing solutions.

1.2 DESCRIPTION

In this project, we take the information about the graph from the user using the concept of adjacency list. An adjacency list of a vertex V is an array of vertices that are neighbours to the vertex V. Using this adjacency list and some other extra information, we find out the maximal independent set of the given graph.

Both algorithms find the maximal independent set based on the starting vertex the user gives. So the finding set is only a maximal independent set and not a maximum independent set.

1.2.1 SERIAL ALGORITHM

Apart from the adjacency list of the vertices, we first initially obtain the number of vertices and edges of the graph from the user beforehand.

In this algorithm, a binary array of size equal to the number of vertices is created and initialised to zero(0). This array is used to find out the vertices that entered the maximal independent set and those that didn't. After the termination of the algorithm below, if the value present in the binary array at index i is still zero, it means the vertex i entered the maximal independent set.

The steps of the serial algorithm is mentioned as follows :

```
For every vertex v in array of adjacency list,
    If binary array value of index v is zero,
        Then for all the neighbours of v, set the binary array value of
        that particular vertex to 1
Finally the set of elements whose binary array value is 0 is the MIS.
```

We test the above algorithm with different graphs and record the time it takes to complete.

1.2.2 PARALLEL ALGORITHM

We observe that the serial algorithm we have developed suffers from various drawbacks :

1. The serial algorithm takes a longer time for computing the MIS for a graph, as the number of vertices increases.
2. It is actually tiring for the user to provide the adjacency list of all the vertices of the graph. Basically, it is a time-consuming process and it is not of much use for the computation.

In order to remove these drawbacks and improve efficiency, we try to come up with a parallel algorithm. In this project, we have made use of a randomised parallel algorithm. Randomisation is a common technique where you generate random numbers and your algorithm works based on the numbers that are being generated.

The parallel algorithm operates in synchronous rounds, grouped into phases.
A single phase is as follows:

1. *Each node v chooses a random value r(v) ∈ [0, 1] and sends it to its neighbors.*
2. *If r(v) < r(w) for all neighbors w ∈ N(v), node v enters the MIS and informs its neighbors.*
3. *If v or a neighbor of v enters the MIS, v terminates (v and all edges adjacent to v are removed from the graph), otherwise v enters the next phase.*

If all the vertices of the graph are used in at least one round, the algorithm is terminated. Else, we enter another round with the remaining set of vertices still available.

We implement this algorithm in our project using C language. We have made use of OpenMP API to implement the parallel sections of the code. Our program doesn't require the user to input the adjacency list of all the vertices. Whatever vertex the algorithm chooses during each phase, information regarding that alone is required. We test our project with different graphs and note down the running time of each.

1.3 SYSTEM CALLS USED

1.3.1 OMP_GET_WTIME()

Function provided by OpenMP to compute the running time of the program.

1.3.2 RAND()

Function to generate random numbers.

1.3.3 OPENMP CONSTRUCTS USED

#PRAGMA OMP PARALLEL FOR SHARED(VARIABLE_NAME)

This is an OpenMP construct used to parallelise the for loop. The iterations of the for loop are split between the threads. Each thread executes one iteration of the loop independently. The variable provided within the shared clause is shared among all the threads.

#PRAGMA OMP PARALLEL FOR REDUCTION(+ : VARIABLE_NAME)

This is an OpenMP construct used to parallelise the for loop. The iterations of the for loop are split between the threads. Each thread executes one iteration of the loop independently. The variable provided within the reduction clause is present in the critical section. This clause provides inbuilt synchronisation for the threads to access the critical region.

1.4 TOOLS AND TECHNOLOGY

In this project, we made use of the Linux operating systems and used the tool OpenMP API in order to parallelise our code. The OpenMP API supports multi-platform shared-memory parallel programming in C/C++ and Fortran. The OpenMP API defines a portable, scalable model with a simple and flexible interface for developing parallel applications on platforms from the desktop to the supercomputer.

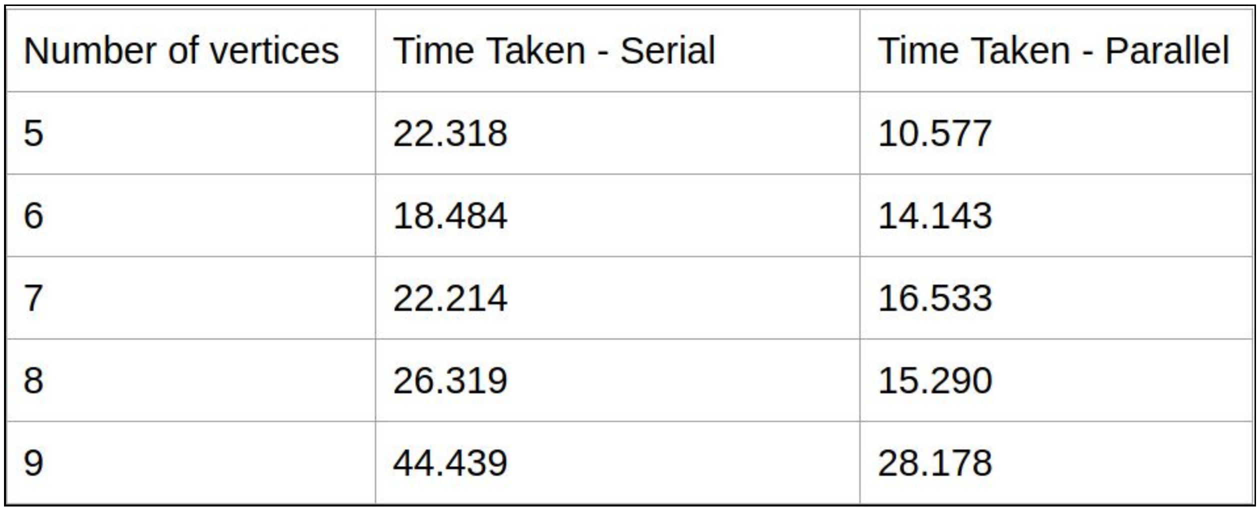
1.5 WORKFLOW

The workflow of this project is divided into seven steps as shown in the figure given below :

1. We come up with the serial algorithm for finding the MIS for a given graph.
2. Once we have developed the prototype, we code it and test our algorithm with different types of graphs. We note down the time it takes to run for each graph.
3. Now, we develop a parallel algorithm for the same using the concept of randomization.
4. We code our algorithm and again perform testing. The running times are noted down.
5. The parallel algorithm is further optimized with the help of constructs provided by the OpenMP API. We make use of threads to improve the performance.
6. The new parallelised algorithm is tested again and the running times are noted.
7. We use a performance graph to assess our findings and conclude the results obtained.

1.6 RESULTS

We have noted down the running times of the parallelized algorithm using OpenMP and our serial algorithm. The results are given below in Figure 1.



Number of vertices	Time Taken - Serial	Time Taken - Parallel
5	22.318	10.577
6	18.484	14.143
7	22.214	16.533
8	26.319	15.290
9	44.439	28.178

Figure 1

We plot the table and obtain the performance graph for the serial and parallel algorithm. The graph is shown in Figure 2.

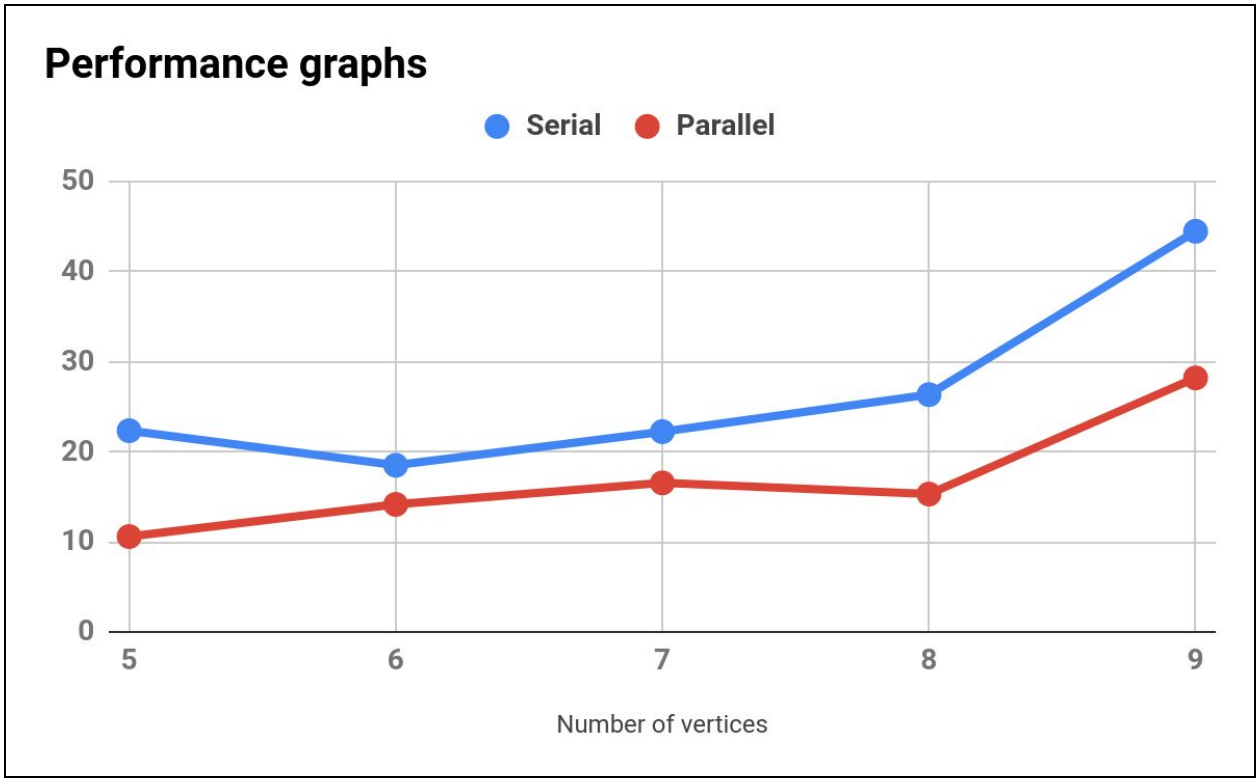


Figure 2

From the above graph, we can make out the fact that our parallel algorithm works much better compared to that of the serial version. When the number of vertices in the graph increases, the time taken by the serial algorithm to compute the MIS becomes very large. This happens due to the fact that we are iterating over all the vertices. However with the help of randomization and OpenMP, the parallel algorithm works over a smaller subset of the vertices. This has led to the improved performance as shown.

1.7 CONCLUSION

We can conclude that our parallel algorithm is much faster and efficient to compute the MIS for a given graph. Making use of the concept of randomisation has helped us to gain better performance.

However, this project is only for computing a single MIS for a given graph. It doesn't compute all the maximal independent sets of a graph. The problem of finding out all the maximal independent sets of a graph is NP-hard.

In the future, we can look forward to integrating many more algorithmic approaches to our package and tune it further up.

1.8 BIBLIOGRAPHY

1.8.1 WEBSITES

1. Tim Mattson's (Intel) "Introduction to OpenMP" (2013) on YouTube
    a. Slides: Intro_To_OpenMP_Mattson.pdf
    b. Exercise files: Mattson_OMP_exercises.zip
2. https://disco.ethz.ch/courses/podc_allstars/
3. https://www.net.t-labs.tu-berlin.de/~stefan/class02-mis.pdf

1.8.2 RESEARCH PAPERS

1. https://disco.ethz.ch/courses/podc_allstars/lecture/chapter7.pdf