PARALLEL IMPLEMENTATION OF MODIFIED GREEDY ALGORITHMS WITH OPENMP

RENEA CHOWDHURY SHORMEE

UNIVERSITI KEBANGSAAN MALAYSIA

PARALLEL IMPLEMENTATION OF MODIFIED GREEDY ALGORITHMS WITH OPENMP

RENEA CHOWDHURY SHORMEE

PROJECT SUBMITTED IN PARTIAL FULFILMENT FOR THE DEGREE OF REQUIREMENTS FOR MASTER OF COMPUTER SCIENCE

FACULTY OF INFORMATION SCIENCE AND TECHNOLOGY
UNIVERSITI KEBANGSAAN MALAYSIA
BANGI

2018

PELAKSANAAN SELARI ALGORITMA TAMAK YANG DIUBAHSUAI DENGAN OPENMP

RENEA CHOWDHURY SHORMEE

PROJEK YANG DIKEMUKAKAN UNTUK MEMENUHI SEBAHAGIAN
DARIPADA SYARAT MEMPEROLEHI IJAZAH SARJANA SAINS KOMPUTER

FAKULTI TEKNOLOGI DAN SAINS MAKLUMAT
UNIVERSITI KEBANGSAAN MALAYSIA
BANGI

2018

DECLARATION

I hereby declare that the work in this thesis is my own except for quotations and summaries which have been duly acknowledged.

04 July 2018

RENEA CHOWDHURY SHORMEE
GP04498

ACKNOWLEDGEMENT

First and foremost praise be to God for all his blessings for giving me patience and good health throughout the duration of this Project.

I am grateful and thankful to my supervisor Associate Professor Dr Ravie Chandren Muniyandi who took me as a student and for his patient supervision, guidance, constructive suggestions and comments during the research period until its completion. His advice and support throughout the program is invaluable. Without his tireless help, leadership, and confidence in my ability, completing this project would not have been possible. I also offer my gratitude to him for opening my mind to a new world of knowledge for Parallel Computation, opportunities and experiences, and giving me a better understanding throughout.

I am really grateful to my cousin brother Dr Dip Nandi who encouraged me to pursue my Master's degree and suggested me to join National University of Malaysia.

Finally, I sincerely thanks my parents for their support and prayer to complete my study. Special gratitude to my father for supporting me throughout this journey in Malaysia. I also thank my husband for his moral support all the time.

My gratitude thanks to National University of Malaysia.

ABSTRACT

Investigating the multi-core architecture is an essential issue to getting superior in parallel reenactments. However, the simulation highlights must fit on parallel programming model keeping in mind the end goal to build the execution. The main goal of this project is to choose and evaluate parallelism using OpenMP over sequential program. For this purpose, there is a portrayal of two greedy algorithms. The calculation to discover the next edge of Prim's algorithm, single source shortest way of Dijkstra's algorithm. These two calculation actualize in sequential formulation. The parallel greedy algorithms are then implemented in view of multi core processor and the speed-up ratio and efficiency of parallel greedy algorithms are tested and investigated in SGEMM GPU Kernel performance dataset with 241600 records and 18 attributes. Results show the dataset with different data sizes achieved super linear speed-up ratio and efficiency on OpenMP running 4 cores processor and reduction of the running time over sequential program. More importantly, the new implementation drastically decreases the time of execution for thread 8 for Prims algorithm for 5.16 ms to just over 1.48 ms for Dijkstra algorithm. Parallel calculation is impressively powerful for huge graph size. Parallel Programming can be an exceptionally valuable approach to work through huge informational datasets and get results about significantly faster than it had been utilized a sequential execution of a calculation. Not only it can be more effective but also it can push the architecture of the previous framework to the most extreme. This paper investigate what multi-threading and parallelism technique can improve the situation when utilizing them on a parallel issue. By utilizing these algorithm locate the most limited separation. Parallel calculation utilized for computing or finding most limited way of graph. With the assistance of graph algorithm these activities should be possible in parallel and diminish the calculation time and efficiency. General outcomes show that multi-threaded parallelism is exceptionally successful to accomplish speedup for data set based on greedy algorithms by separating the primary data set into sub-datasets to increase diversity on arrangement investigation.

ABSTRAK

Menyiasat seni bina multi-core adalah isu penting untuk mendapatkan simulasi selari yang lebih baik. Walau bagaimanapun, sorotan simulasi mesti sesuai dengan model pengaturcaraan selari dengan mengambilkira matlamat akhir untuk membina pelaksanaan. Matlamat utama projek ini adalah memilih dan menilai pelaksanaan selari dengan menggunakan OpenMP melalui program berurutan. Untuk tujuan ini, terdapat dua algoritma tamak yang digunakan. Pengiraan untuk mencari kelebihan seterusnya dengan algoritma Prim, dan sumber tunggal cara terpendek dengan algoritma Dijkstra. Kedua-dua perhitungan ini berlaku dalam rumusan berurutan. Algoritma tamak selari kemudian dilaksanakan memandangkan pemproses berbilang teras dan nisbah kelajuan dan kecekapan algoritma tamak selari diuji dan disiasat dalam dataset prestasi Kernel SGEMM Kernel dengan rekod 241600 dan 18 atribut. Keputusan menunjukkan dataset dengan saiz data yang berbeza yang mencapai nisbah laju dan kecekapan super linear dan kecekapan pada OpenMP yang menjalankan 4 teras pemproses dan pengurangan waktu berjalan melalui program berurutan. Lebih penting lagi, pelaksanaan baru secara drastik menurunkan masa pelaksanaan untuk thread 8 untuk algoritma Prims untuk 5.16 milisaat kepada lebih dari 1.48 milisaat untuk algoritma Dijkstra. Pengiraan selari sangat mengagumkan untuk ukuran graf yang besar. Pemprograman selari boleh menjadi satu pendekatan yang sangat berharga untuk bekerja melalui dataset maklumat yang besar dan mendapatkan hasil yang lebih cepat daripada yang telah digunakan untuk melaksanakan satu perhitungan berjadual. Bukan sahaja ia boleh menjadi lebih berkesan tetapi juga boleh mendorong seni bina rangka kerja sebelumnya kepada yang paling melampau. Kertas kerja ini menyiasat teknik-teknik multi-threading dan parallelism yang dapat memperbaiki keadaan ketika menggunakannya pada masalah selari. Dengan menggunakan algoritma ini, pengasingan yang paling terhad. Pengiraan selari digunakan untuk pengkomputeran atau mencari cara graf yang paling terhad. Dengan bantuan algoritma graf, aktiviti-aktiviti ini boleh dilakukan secara selari dan mengurangkan masa pengiraan dan kecekapan. Hasil umum menunjukkan bahawa paralelisme multi-threaded sangat berjaya untuk mencapai kelajuan untuk menetapkan algoritma tamak berdasarkan data dengan memisahkan data utama yang ditetapkan ke dalam sub-dataset untuk meningkatkan kepelbagaian pada penyelidikan pengaturan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF ABBREVIATIONS

API	Application Programming Interface
CUDA	Compute Unified Device Architecture
MPI	Message Passing Interface
MST	Minimum Spanning Tree
NUMA	Non-Uniform Memory Access
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing
PVM	Parallel Virtual Machine
UMA	Uniform Memory Access

CHAPTER I

INTRODUCTION

1.1 RESEARCH BACKGROUND

Finding the shortest distance for all objects in a graph is a common task in solving many day to day and logical issues. The algorithm for finding the shortest path, discover their application in numerous fields, for example, Google maps, routing protocol and so on. There are two algorithms for finding the most limited way and single source shortest way, utilizing two algorithms Prims algorithm and Dijkstra's algorithm. The shortest way calculation is the fundamental algorithm for research hotspot. To improve the searching the best use of shortest path is to implement the parallelism. It’s use as in the parallel algorithm. Otherwise in serial implementation it is very difficult to improve its performance for Dijkstra’s algorithm(Cao et al. 2009).

In sequential algorithm implementation, it requires long time to discover the most shortest distance if all sets of vertices are in the graph. So it is troublesome assignment to locate the most nearest node from source to goal. Consequently utilize the parallel algorithm or parallel processing takes less time than the sequential execution. And it is easy to calculate and find the most nearest node or path for algorithms or numerous different applications. Both of the greedy algorithm problem starts initially considering source as A to all other vertices in datasets. These two algorithm Prims and Dijkstra’s algorithm are most important to find the shortest path. The graph solving problems increment in measure, effective parallel most limited way handling ends up imperative as computational and memory prerequisites increment (Awari 2017).

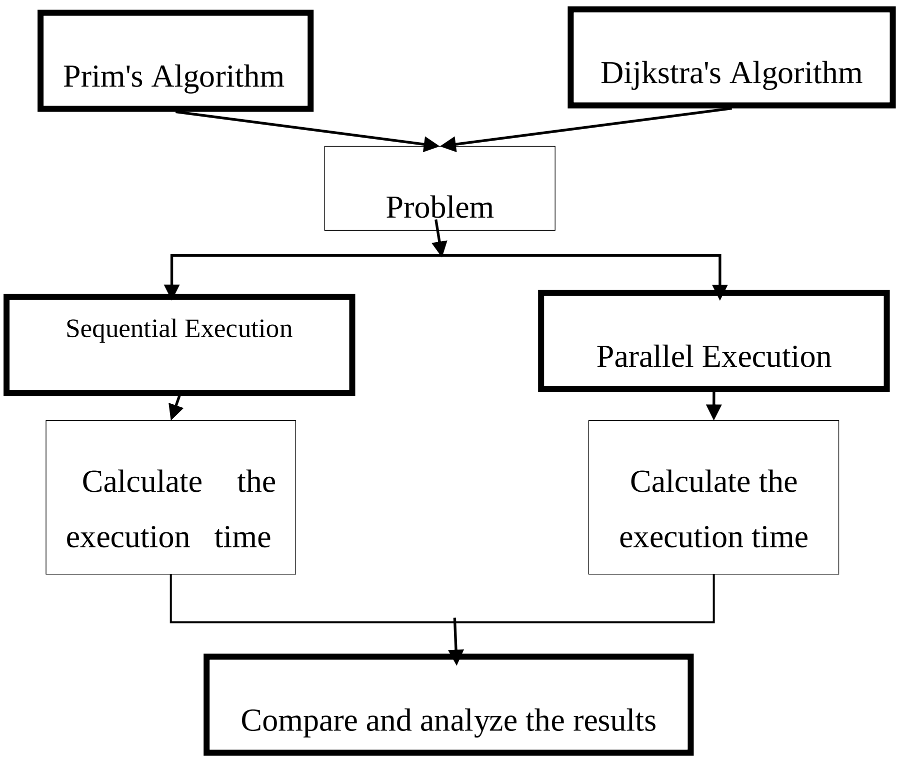
Prim's Algorithm
Dijkstra's Algorithm
Problem
Sequential Execution
Parallel Execution
Calculate the execution time
Calculate the execution time
Compare and analyze the results

Figure 1.1 Overview of proposed work

For large structure or framework it requires long time to perform their tasks and this is the reason why the parallelization used to perform operation in less time. Execution for the task in less time to diminish the proficiency and speedup factor utilizing the OpenMP and furthermore utilize the parallel Prims algorithm, parallel Dijkstra algorithm(Awari 2017). The overview of the proposed work has been shown in figure 1.1. Parallel computers can be generally classified as Multi-Core and Multiprocessor. A core is the piece of the processor which performs perusing and executing of the guideline. However as the name implies, Multicore processors are made out of in excess of one core. An extremely regular case would be a dual core processor. The upside of a multicore processor over a single core one is that the multi-core processor can either utilize the two cores to achieve a solitary undertaking or it can traverse threads which separated tasks between the two of its cores, with the goal that it requires double measure of investment it would take to execute the assignment than it would on a single core processor. Multicore processors can likewise execute various assignments at a single time. Execution is the action of gathering the data about the execution attributes of a program(Pathare & Kulkarni 2014).

This chapter shows a review about the research. It identifies the research issues and the requirement for doing such research. This chapter additionally illuminates research conditions by obviously characterizing research scope and methodologies together with the setting up of the research objectives.

1.2 PROBLEM STATEMENT

With the rapid improvement of urban communities, congested road turned into a concerning issue. Along these lines the Intelligent Traffic System is developing rapidly and the shortest path optimization is an important part of this problem. This issue has been the research hotspot for long time and for the sequential shortest path optimization, individuals have gotten numerous research comes about and applied in many applications(Cao et al. 2009).

Big data mostly comes from people's day-to-day activities and Internet-based companies. Big data represents content and cloud computing is an environment that can be used to perform tasks on big data. Nonetheless, the two concepts are connected. In fact, big data can be processed, analyzed, and managed on cloud. Parallel algorithms can be implemented in the cloud-computing environment to reduce computation time, memory usage and I/O overhead for generating frequent item sets(Reyes-Ortiz et al. 2015).Moreover, several single source shortest path algorithms and minimum spanning tree have been computed in order to resolve this issue, to saves time using parallel process to quickly focus only on the results of attentiveness. For this study, the parallel computation is a proficient method to enhance the greedy algorithms containing large data.

1.3 RESEARCH QUESTION

The study aimed to answer the following Research Questions (RQ):

1) What is the suitable programming process to execute the simulation in less amount of time for large datasets of greedy algorithms?

2) How can the proposed parallel process is going to influence the performance of two greedy algorithms?

1.4 RESEARCH OBJECTIVES

This research aim to achieve the following objectives:

1) To propose and implement the programming process between sequential and parallel programming that required the less execution time for large datasets.
2) To investigate and evaluate the performance of parallel process using OpenMP over sequential programming of two greedy algorithms.

1.5 RESEARCH METHODOLOGY

The research methodology of this study has been distributed into three stages. To begin with stage 1 is concentrating on the sequential algorithm process including the implementation, execution of time, with the scenario of datasets.

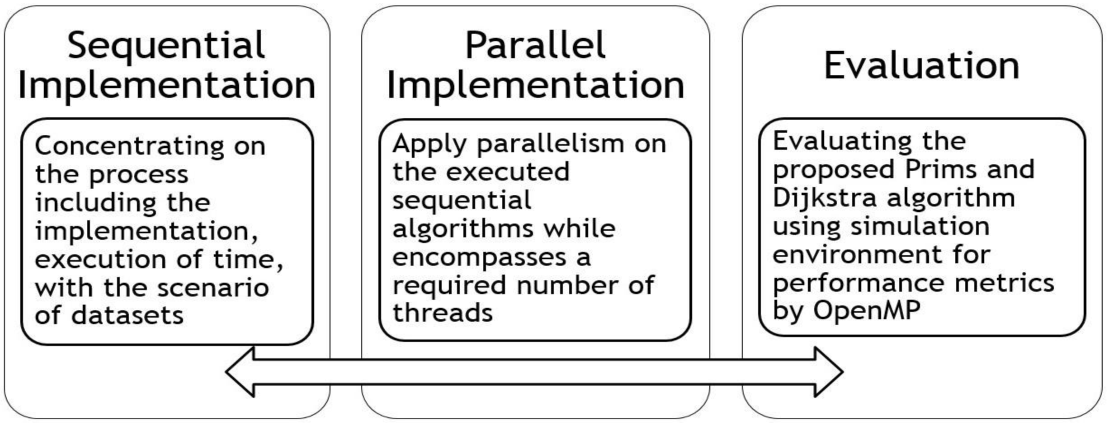
Sequential Implementation
Concentrating on the process including the implementation, execution of time, with the scenario of datasets
Parallel Implementation
Apply parallelism on the executed sequential algorithms while encompasses a required number of threads
Evaluation
Evaluating the proposed Prims and Dijkstra algorithm using simulation environment for performance metrics by OpenMP

Figure 1.2 Research Methodology

While, the second stage is focusing on the parallelism procedure for the executed sequential greedy algorithms while encompasses a required number of threads and applying such process with observing the outcomes for those threads. Lastly, the third stage will focus on the assessment of the proposed process by comparison.Here, the Figure 1.2 demonstrates the research methodology of this research.

1.6 RESEARCH SCOPE

This research has been done on sequential algorithms using parallel component. The parallelism requires multicore processors with the goal that it can span threads which isolated tasks between its cores. In the space of this exploration, this study address two kinds of greedy algorithms: Prims and Dijkstra algorithms. By utilizing OpenMP, consuming or reducing the execution time for outlining the parallel programs, so large problems can be settle in less measure of time(Awari 2017).

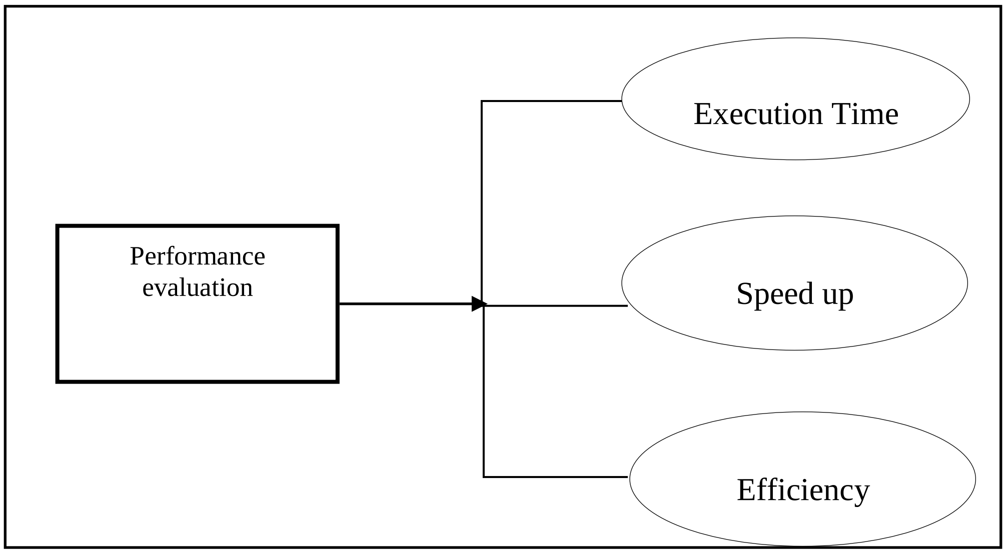

Figure 1.3 Performance evaluation

As well as compute the performance assessment parameter speedup and efficiency based on the algorithms. The work process of the proposed study is appeared in figure 1.3.

1.7 ORGANIZATION OF THE THESIS

This thesis is composed into five chapters; these parts are condensed as takes after:

Chapter I give a prologue to the research background and recognize the current problem statement regarding the OpenMP parallel procedure over sequential program.

This chapter likewise portrays the objectives, strategy and scope of the study and research significance.

Chapter II displays a literature review for greedy algorithms, Prims algorithm, Dijkstra algorithm, Parallel programming and OpenMP. At that point the related work includes the steps and mechanisms that proposed to reduce the execution time. At last, a synopsis closes the chapter.

Chapter III illustrates the methodology that was actualized to accomplish the research objectives. Especially, it portrays in points of interest the simulation environment and the parameters as to representation of the work. Likewise this chapter discusses about the point by point clarification about outline contemplations. Furthermore, the process of method for this experiment has been described over here. Here this study explain the sequential algorithm. Then the analysis of OpenMP with an example of a program. This chapter additionally has an explanation of parallel execution of both algorithms by introducing the #pragma and multi-threading features of OpenMP. At last, this chapter end with a summary of parallel workflow and techniques.

Chapter IV investigates the reenactment come about by utilizing different performance metrics, which are execution time, speed-up ratio and efficiency. Various simulation runs have been done to get the results. There are comparisons between two programming techniques using two different algorithms. Simulation results and discussion are described at the end of this chapter.

Chapter V gives a synopsis to the investigation comes about, draws some conclusions about them, research findings, inquire about significances. It additionally incorporates recommendations for the future works potential thoughts which are identified with the current research.

CHAPTER II

LITERATURE REVIEW

2.1 INTRODUCTION

With the advancement of Computer and data innovation, the research about graph hypothesis get an extensive variety of consideration, and an assortment of data structures and algorithms have been proposed.The shortest path algorithm is always a research hotspot in graph theory and it is the most basic algorithm. For instance, searching the shortest path is utilized to implement activity designing in IP organizes and to enhance Intelligent and Transportation Systems. But for that kind of algorithm it is extremely hard to enhance its execution. At present the algorithms for the sequential searching optimization have reached the time limitation. Subsequently the parallel computation is an effective method to enhance the performance. By putting a few constraints on the data and taking the benefit of the hardware, the execution of the algorithms can be essentially progressed(Cao et al. 2009).

2.2 FRAMEWORK OF LITERATURE REVIEW

In this chapter it describe the related work of the objectives of the project. In addition, this study present the framework of Literature Review of all research studies as shown in Figure 2.1.

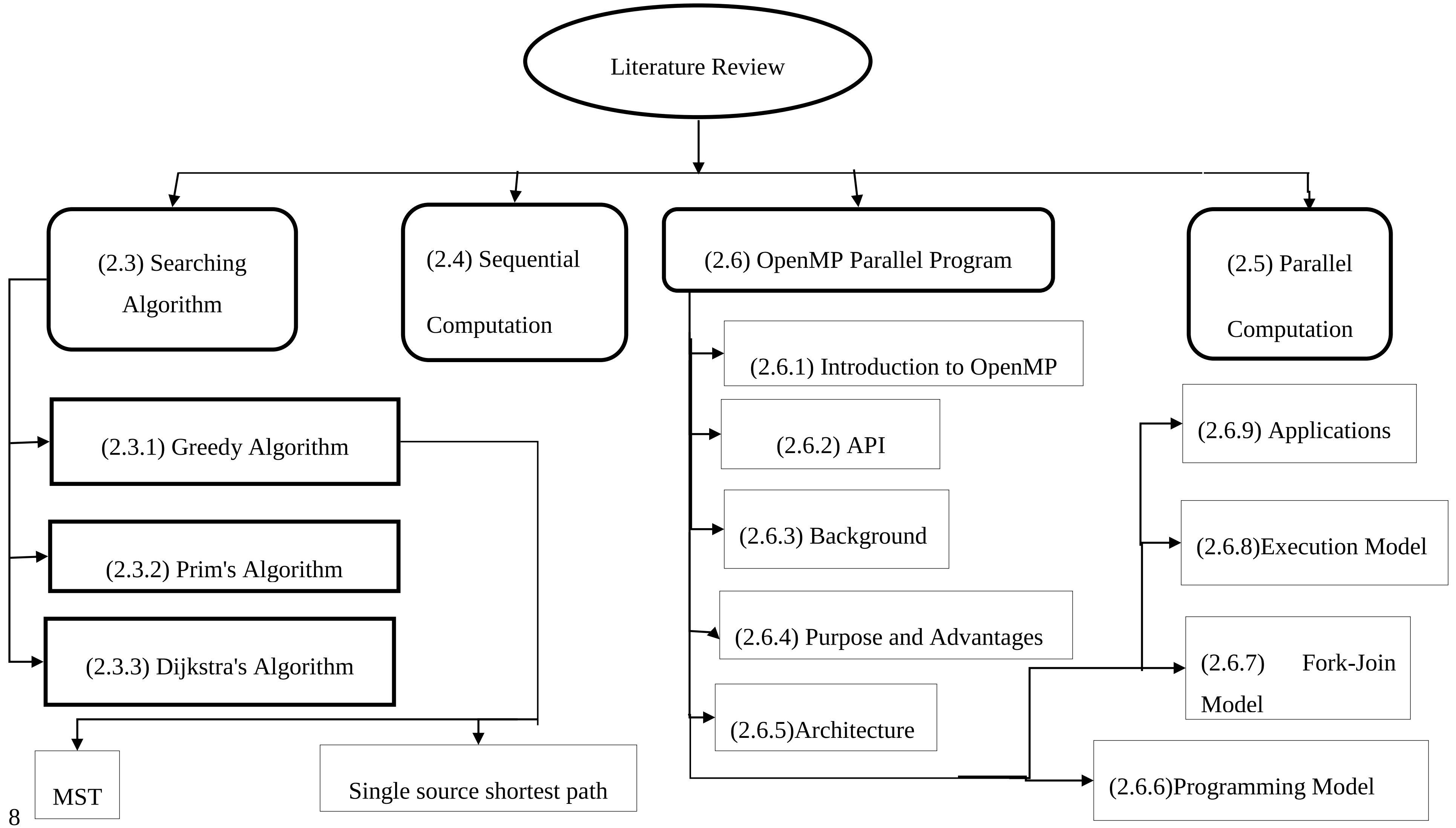
Literature Review
(2.3) Searching Algorithm
(2.4) Sequential Computation
(2.6) OpenMP Parallel Program
(2.5) Parallel Computation
(2.3.1) Greedy Algorithm
(2.3.2) Prim's Algorithm
(2.3.3) Dijkstra's Algorithm
MST
Single source shortest path
(2.6.1) Introduction to OpenMP
(2.6.2) API
(2.6.3) Background
(2.6.4) Purpose and Advantages
(2.6.5)Architecture
(2.6.9) Applications
(2.6.8)Execution Model
(2.6.7) Fork-Join Model
(2.6.6)Programming Model

Figure 2.1 Framework of literature review

2.3 SEARCHING ALGORITHM

A search algorithm is the step-by-step procedure used to find particular information among a collection of data. It is considered as a fundamental procedure in computing. In software engineering, while searching for data, the difference between a quick application and a slower one regularly lies in the utilization of the proper search algorithm.

2.3.1 Greedy Algorithm

Greedy algorithm work in stages. In each stage, a choice is made that seems to be good, without regards for future results. For the most part, this implies some local optimum is picked. This "take what you can get now" technique is the source of the name for this class of algorithms. At the point when the algorithm ends, it is hoped that the local optimum is equivalent to the global optimum. If so, at that point the algorithm is right. Otherwise, the algorithm has delivered a suboptimal arrangement. In the event that the most perfect answer isn't required, at that point straightforward greedy algorithms are now and then used to create approximate answers, as opposed to utilizing the more complicated algorithms for the most part required to produce a correct answer. There are a few real-life cases of greedy algorithms.

The most evident is the coin-changing issue. To roll out improvement in U.S. currency, it over and again dispense the largest denomination. In this manner, to give out seventeen dollars and sixty-one cents in transform, it gives out a ten-dollar charge, a five-dollar charge, two one-dollar charges, two quarters, one dime and one penny. By doing this, they are ensured to minimize the quantity of bills and coins. This algorithm does not work in every monetary framework, but rather luckily, this can demonstrate that it works in the American money related framework. Indeed, it works regardless of whether two-dollar bills and fifty-cent pieces are permitted(Fallis 2013).

Traffic problems give a case where settling on locally optimal decisions does not continuously work. For instance, amid certain rush hour times in Miami, it is best to remain off the prime streets regardless of whether they look empty, in light of the fact that traffic will grind to a halt a mile not far off, and people will be trapped. Considerably all the more stunning, it is better now and again to influence an impermanent bypass toward the path inverse your goal so as to maintain a strategic distance from all traffic bottlenecks(Fallis 2013).

A greedy algorithm settles on a locally optimal choice with the expectation that this decision will prompt a globally optimal solution. The decision made at each progression must be:

i) Feasible : Satisfy the problem's limitations

ii) Locally Optimal: Be the best nearby decision among every single decision

iii) Irrevocable: Once made, the decision can't be changed on resulting steps

Optimal solution are:

a) Change making

b) Minimum Spanning Tree (MST)

c) Single-source shortest paths

d) Huffman Algorithm

In this proposed work, this study is considering Dijkstra and Prim's algorithm. Both of them are greedy algorithms but have some different criteria. Prim's algorithm is a MST search algorithm and Dijkstra's algorithm is a single source shortest path algorithm.

a. MST

The MST of a graph is the arrangement of edges that associate each vertex contained in the original graph, such that the total weight of the edges in the tree is minimized. In spite of the fact that this problem discover its application in a few areas, it assumes a more significant part in Very Large Scale Integration (VLSI) outline and system steering. The examination on MSTs has been dynamic for quite a few years, during which various MST solvers and usage have been proposed. In numerous application spaces, for example, impromptu systems, MST-solvers are frequently required, in this manner requesting proficient executions.

b. SSSP

The single source shortest path problem is that of computing, for a given source vertex *s* and a destination vertex *t*, the weight of a path that obtains the minimum weight among all the possible paths. Dijkstra's algorithm is a graph search algorithm that solves single-source most limited way for a graph with non-negative weights. Widely utilized as network routing protocol.

In this paper, the proposed algorithms are Prim's algorithm and Dijkstra's algorithm. Dijkstra's algorithm is for single source shortest path and Prim's algorithm is for MST. The primary distinction between the two is the rule that is utilized to pick the following vertex for the tree. Prim pick the nearest vertex to any vertex in the MST. Though, Dijkstra pick the nearest vertex from the source vertex.

2.3.2 Prim's Algorithm

The algorithm was found in 1930 by mathematician Vojtech Jarnik and later independently by computer scientist Robert C. Prim in 1957. MST algorithm have been implemented in a several parallel computing devices, for example, an amalgamation of Prim's algorithm on multicore CPU chips. Sequential implementations of Prim's algorithm is very simple and its execution changes with the input graph and the data structures utilized. During the late 90s, the exploration around these MST algorithms spun around implementation points of interest to enhance the execution of the algorithms and some were appeared to be of awesome impact on the performance of the algorithms(Prim 1957). The algorithm constantly builds the measure of a tree beginning with a single vertex until the point when it spans all the vertices.

Prim's algorithm is a Greedy Algorithm. It begins with an empty spanning tree. The idea is to keep up two sets of vertices. The first set contains the vertices already included into the MST, the other set contains the vertices not yet included(Cormen et al. 2002). At each progression, it consider all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST(Fallis 2013).

Prim's algorithm is a MST algorithm that works much like Dijkstra's algorithm which improves the situation shortest path trees. Truth be told, it's even simpler. So the algorithm is the same as Dijkstra's algorithm, aside from the programmer don't include distance to the length of the edge when choosing which edge to put in next.

MST-PRIM (*G*, *w*, *r*) pseudocode:

```
1  for each $u \in G.V$
2        $u.key = \infty$ 1
3        $u.\pi =$ NIL
4  $r.key = 0$
5  $Q = G.V$
6  while $Q \neq \emptyset \pi \phi$
7        $u =$ EXTRACT-MIN($Q$)
8        for each $v \in G.Adj[u]$
9              if $v \in Q$ and $w(u, v) < v.key$
10                   $v.\pi = u$
11                   $v.key = w(u, v)$
```

Source: (Chen, n.d.)

So, at each progression of Prim's algorithm, it locate a cut of two sets, one contains the vertices officially incorporated into MST and another contains rest of the vertices and pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains effectively included vertices).

The pseudo code of Prim's algorithm has been represented in the previous page. The work flow according to this has been described below:

Step 1: Choose any vertex as a starting vertex. At that point discovered all the associated edges to that vertex and their weights. Compare between the edges and discover the minimum weights. Vertex that contain minimum weights from the beginning node should be added to the tree.

Step 2: Now it should take a look at all the nodes or vertices of the tree and find out all the edges associated with those tree nodes. This ought to likewise discover the weights of those edges and attempt to find out the minimum weights. Finally this should include the nodes with minimal weights to the tree.

Step 3: Repeat the steps until the point that all the vertices are in the tree or visited.

The figure 2.2 shows the steps of Prim's algorithm using the flow of algorithm from the pseudo code. The algorithm flow takes after:

i) Suppose, beginning from a vertex *u0* in the associated graph *G = (V,{E})*, the base weight edge *(u0, u1)* related with it is chosen and its vertices are added to the vertex set *U* of the spreading over tree.

ii) Each progression is then chosen from the edges *(u, v)* whose vertices are incorporated into *U* and alternate vertices are most certainly not. The edges are added to the edge set *TE* of the base crossing tree, while alternate vertices are added to the set *U*.

iii) Repeat this procedure until the point that all the vertices are added to the vertex set *U* of the crossing tree. The way towards developing the base crossing tree utilizing Prim algorithm is illustrated in Fig.2.2, where *V1*to*V6* are the objective focuses, and the number between the two target focuses is the weight between the two target focuses.