Parallelization Of Traveling Salesman Problem Using Rocks Cluster

Izzatdin Abdul Aziz, Low Tan Jung, and Mazlina Mehat

***Abstract*—The Traveling Salesman Problem (TSP) is a well known Nondeterministic Polynomial (NP) problem. A number of sequential algorithms have been well defined to solve TSP. However in this paper the authors demonstrate an alternative way of solving TSP with parallelism by modifying Prim's algorithm and integrate the modified algorithm with a Random algorithm making the whole process more computational extensive in visiting all the nodes/cities. This paper discussed on converting the sequential algorithm into a parallel version to be computed in a High Performance Computer (HPC) using Message Passing Interface (MPI) libraries running on ROCKS open source systems. Outcome of load balancing and cluster performance is also discussed.**

***Index Terms*— High Performance Computing (HPC), Message Passing Interface (MPI), ROCKS Cluster, Traveling Salesman Problem (TSP).**

I. Introduction

The Travelling Salesman Problem (TSP) is famous as a resource extensive algorithm due to its complex computation. Given a number of cities representing vertices and edges representing paths, this algorithm would compute the shortest path for a person to make a round trip visiting all the cities only once.

This algorithm has been applied in many engineering and technological application [1],[2],[3] such as electronic circuitry design, neuron chain, oil and gas transportation application, DNA research and protein folding. The importance of gaining the shortest path in these computationally complex applications has led to the optimality of operation thus achieving high-quality and accurate results.

It is fairly clear that TSP algorithm needs to run in a high performance computing facility in order to achieve comparable result within diminutive time.

It is known to the mathematical and computing community that TSP is a time consuming algorithm in getting to the shortest path for few to enormous vertices. The aims of this paper are to produce a parallel algorithm of TSP using Message Passing Interface (MPI) and to measure the performance on ROCKS Cluster in terms of load distribution and CPU performance.

The ROCKS is an open source application that allows networked computing terminals to perceive each other as one cluster working in unison to solve a common problem [4],[5].

It is not difficult for one to predict that the more the number of computing nodes used for a fix number of vertices would results in a shorter completion time.

The data used for this study is 500 cities to represent vertices in x:y axis. To illustrate the complexity of the situation, the number of possible path a salesmen could possibly take is $P = n!$, where n factorial indicates the number of cities and P number of possibilities. For the case of three cities, six permutations representing viable paths are possible. When considering the case of 20 cities or 20! in which there are over 2.4 billions permutations. If we exhaustively searched through all the possible path traversals at a rate of 500 million per second, it would take over 150 years to examine them all [6]. Therefore, imagine how immense a TSP problem could be when dealing with 500 cities or 500!

II. ROCKS CLUSTER

Twenty machines were used to form the cluster. Each was off-the-shelf Intel based machine with dual P3-733 MHz processors and came with 512 MB memory. However it should be noted here that only five machines/nodes (ten processors) were actually used in the performance measurement.

All these machines were connected to a Fast Ethernet 100Mbps switch at which there exist a head node that acted as the master node. The master node has multiple network interfaces [7].

ROCKS Cluster Distribution is a Linux distribution intended for high-performance computing clusters. ROCKS was initially based on Red Hat Linux distribution. ROCKS includes many tools such as MPI, which are not part of CentOS but are integral components that make a group of computers into a cluster [8].

III. TSP Pseudocode and Algorithm

There could be various ways to achieve parallelism in solving TSP. The authors chose to apply modifications to Prim's algorithm and integrate it with Random algorithm making it more computational extensive in visiting all the nodes. Prim's algorithm was chosen so that the process will visit all vertices before master can consolidate the result to determine the shortest path. The Random algorithm was

Manuscript received on September 27, 2007.

Izzatdin Abdul Aziz is with Universiti Teknologi PETRONAS, Tronoh , 31750, Perak, Malaysia (phone: 605-3687473; fax: 605-3656180; e-mail: izzatdin@petronas.com.my).

Low Tan Jung is with Universiti Teknologi PETRONAS, Tronoh, 31750, Perak, Malaysia (e-mail: lowtanjung@petronas.com.my)

Mazlina Mehat is with Universiti Teknologi PETRONAS, Tronoh, 31750, Perak, Malaysia (e-mail: mazlinamehat@petronas.com.my)

picked for its ability to randomly initiate the process for each computing node or slave in order to discover the shortest path.

Fig. 1 is the proposed TSP with parallelism using modified version of Prim's and integration of Random algorithm.

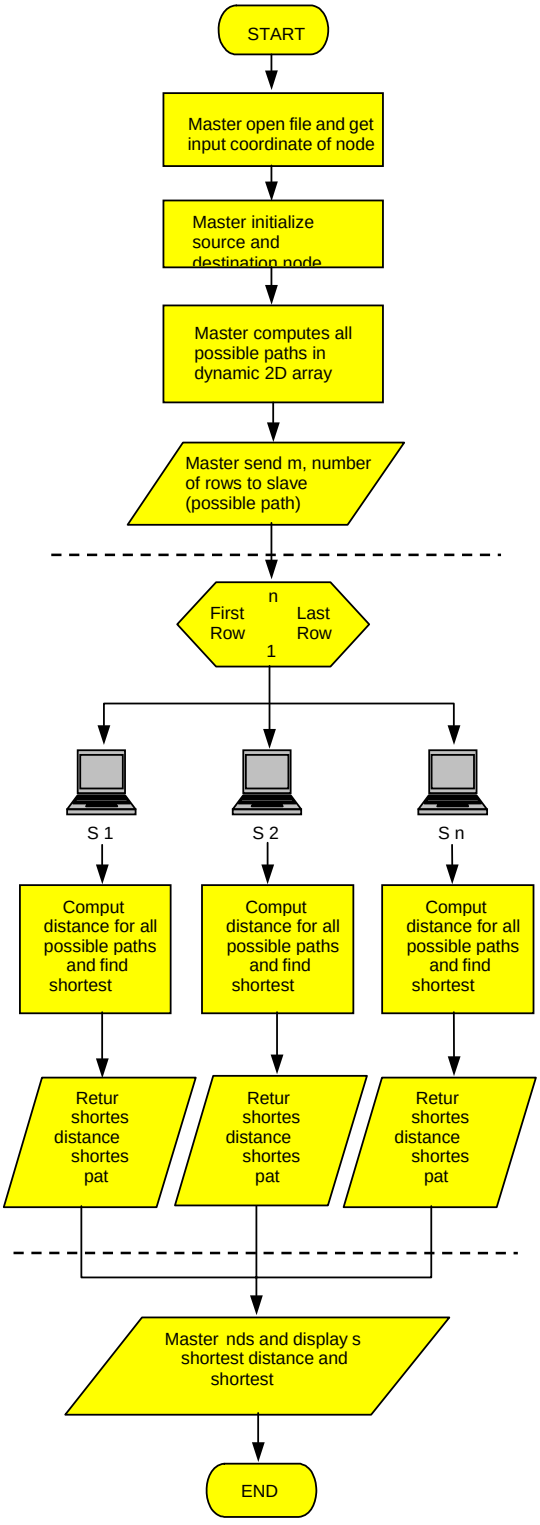

Fig. 1: Proposed parallel algorithm of TSP

IV. Decomposition

In decomposition or job distribution the algorithm works by having the Master creating a linked-list structure in a sorted order from shortest to farthest distance in a 2D matrix. Then, the Master transforms the 2D matrix to a single-dimension array to be broadcasted using MPI_Bcast (this array acts as a map for the entire slave). The Master then sends a random starting point (num_nodes) for the slave as shown in Fig. 2.

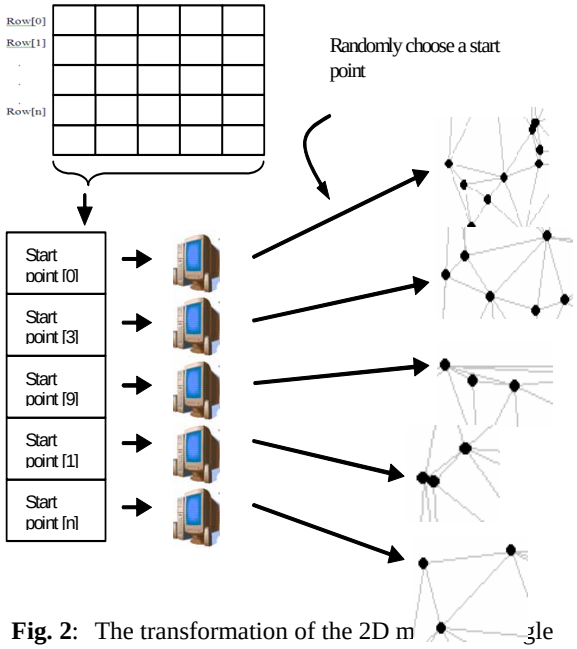

Fig. 2: The transformation of the 2D m gle dimensional arrays.

As soon as the slave receives the initial random start, it compares the distances of itself with all the connected nodes. Once the shortest edge is found, it will go by that edge and marks the current node flag to 1 so that it would not revisit nor consider that node again. It is worth noted here that this marking mechanism does not exist in the original Prim's algorithm.

Each node that was visited will be stored in an array. The first element of the array is the minimal path. Then this array is sent to the Master for comparison with other processors' results. Note that the send and receive process uses a Non-blocking method.

V. Problems Encountered

There were two weaknesses encountered in the algorithm proposed. Firstly the array map broadcasted by the Master can be large. This might overwhelm the slave buffer. One possible solution is to break the array map into smaller chunks and reassemble these chunks once they arrives at the slaves. Secondly, the use of linked-list to add and delete nodes by the Master may slow down the processing speed due to memory allocation and de-allocation by the compiler. A possible solution to this problem is by having a fix-size array instead of building up a dynamic linked list.

Nevertheless the strength is the linked-list which allows

the dynamic linked list tree to be built up only once by the master. So this could optimize the execution of the program by not forcing the slave to iteratively building the linked list, thus avoiding redundancy for slave to visit a particular node more than once if the nodes have been flagged to 1.

VI. Results and Discussions

Two variables were investigated upon execution of the parallel TSP algorithm. The variables are load balance and CPU performance (processing time). The master processor resides in the High Performance Computer (HPC) is used to disseminate the jobs to each and every slave for computation as according to the written algorithm.

A. Load Balancing

From the experiment it can be seen that in Fig. 3, to certain extent an ideal load balancing is achieved. Through out the test, the system is able to maintain 75% of CPU loads.

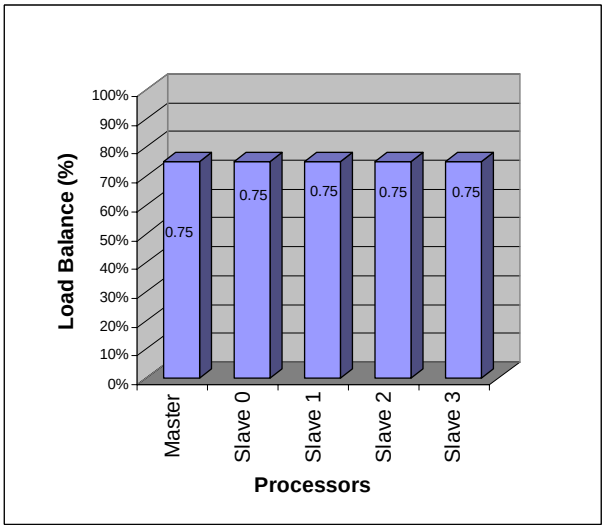

Fig. 3. Load balancing for 5 processors

Towards the end of processing time, the load has reduced to an average of 10% as depicted in Fig. 4. This is due to reduction of all results from slave processors back to the master processor to decide which possibilities of the shortest path is to be concluded.

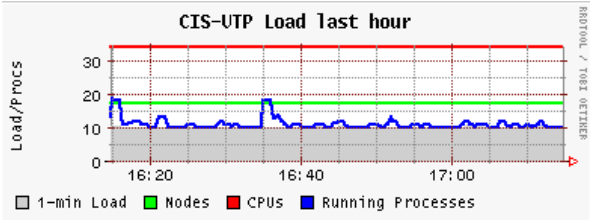

Fig. 4: Load balance towards ending of the process

B Cluster Performance

In Fig. 5, the master processor maintains a consistent performance of 100%, however the slave maintains a consistent performance rate of 75%. This is due to delegation of tasks by the Master, which at the same time performs same computation as the other nodes or slaves.

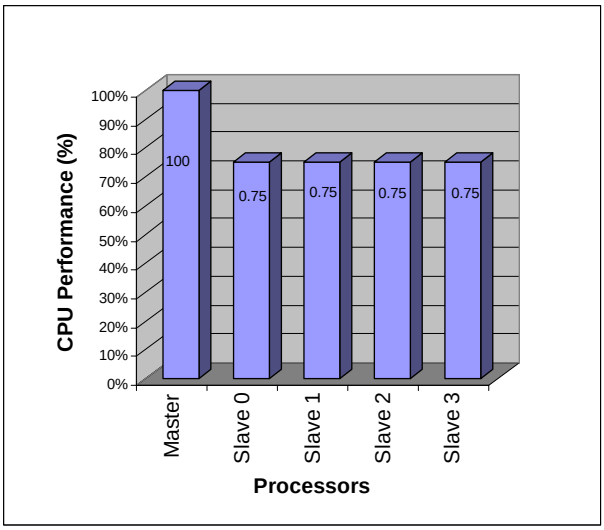

Fig. 5. CPU performance for 5 processors

It is best to remind that the cluster performance measured in this paper refers to the CPU usage of each computing nodes.

VII. Conclusion

A parallel algorithm was developed by modifying the Prim's version to speed up the TSP processing time. This was possible because TSP was solved in unison by a collection of processors. The primary goal here is to address the common problem of a sequential algorithm that consumes huge amount of time when dealing with massive processing needs or massive data set.

It can be seen that by having the ROCKS running on Beowulf cluster we can effectively disseminate the tasks through out the cluster thereby performing computation via the developed parallel algorithm with reasonable performance and load balancing.

The proposed algorithm can be easily applied to tasks that demands high computation needs such as weather forecasting, DNA simulation or analysis, etc. In view of the possibilities of a task downloaded to the cluster for processing may takes more than a day, we recommend that the power supply to the whole cluster needs to be properly backed-up. This is because any power failure shall cause the cluster to process the whole task all over again.

The authors' cluster setup does not come with a roll-back recovery capability. Therefore, will be good to incorporate some form of recovery mechanisms in the cluster for the sake of recovering any possible lost data/results in the event of unforeseen system or cluster crash.

References

[1] P. Borovska,(2006,June). Solving the Travelling Salesman Problem in Parallel by Genetic Algorithm on Multicomputer Cluster. *International Conference on Computer Systems and Technologies 2006*. [Online].

Available:
http://ecet.ecs.ru.acad.bg/cst06/index.php?cmd=dPage&pid=cpr

[2] K.Bryant, A. Benjamin, "Genetic Algorithms and Traveling Salesmen Problem", *Thesis*, Harvey Mudd College, Dept. of Mathematics, 2000.

[3] M. Manfrin, M. Birattari, T. Stutzle, and M. Dorigo, (2006, April). Parallel Ant Colony Optimization for the Traveling Salesman Problem, *ANTS 2006: Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence*. [Online]. Available: http://iridia.ulb.ac.be/supp/IridiaSupp2006-001/index.html

[4] G. Bruno, M. J. Katz, F. D. Sacerdoti and P. M. Papadopoulos, (2004, July). Rolls: Modifying a Standard System Installer to Support User-Customizable Cluster Frontend Appliances. *IEEE International Conference on Cluster Computing (CLUSTER'04)*, pp. 421-430, [Online]. Available: http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/cluster/2004/8694/00/8694toc.xml&DOI=10.1109/CLUSTR.2004.1392641

[5] F. D. Sacerdoti, S. Chandra, and K. Bhatia, (2004, September).Grid Systems Deployment & Management Using Rocks. *IEEE International Conference on Cluster Computing (CLUSTER'04)* , pp. 337-345, [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1392631

[6] M. Lacy .(2001,March 1). *An Introduction to Genetic Algorithms In Java*, [Online]. Available: http://java.sys-con.com/read/36224.htm

[7] D. Adhipta, I. A. Aziz, L. T. Jung, N. S. Haron,(2006, August). Performance Evaluation on Hybrid Cluster: The Integration of Beowulf and Single System Image, *The 2nd Information and Communication Technology Seminar (ICTS*).

[8] S. Meacham, (2007) SCI: Delivering Cyberinfrastructure: From Vision to Reality [Online]. Available http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0438741,