International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 5, ISSN No. 2455-2143, Pages 149-151
Published Online March – April 2016 in IJEAST (http://www.ijeast.com)

PARALLELISM OF GRAPH TRAVERSING ALGORITHM USING OPENMP

Shailendra W. Shende, Abhijit N. Pimple, Bhushan Gajbhiye, Sheetal R. Radke
Department of Information Technology,
Yeshwantrao Chavan College of Engineering,
Nagpur, India.

***Abstract*—Graph traversing is generally complex and time consuming process. There is an ever increasing need of graph algorithm that requires less amount of time for computation as these graph traversing algorithms are used in computational sciences, and social network. Because of irregular and memory intensive nature, graph applications are specifically known for their low performance on parallel computer systems. We often want to solve problem that are expressible in terms of traversal or search over a graph. The goal of graph traversal, generally, is to find all nodes reachable from a given set of root nodes. In this paper, we are trying to implement parallel versions of breadth first search (BFS) and depth first search (DFS) algorithm using OpenMP and featuring with (i) To find the speed up of proposed algorithm, (ii) to increase the efficiency of graph algorithm, (iii) to reduce the amount of time that the graph algorithm usually take, (iv) to increase the resource utilization.**

Keywords—**OpenMP, Breadth First Search (BFS), Depth First Search (DFS).**

I. INTRODUCTION

IJEAST A group of major software and hardware vendors jointly defines OpenMP as an Application Programming Interface (API). For developers of shared memory parallel application, OpenMP provides a portable, scalable model. The API supports C/C++ and FORTRAN on a wide variety of architectures. It comprised of three primary API components: compiler directives, runtime library routines, environment variables. The goals of OpenMP are standardization, lean and mean, ease of use, and portability. Multi-platform shared memory multiprocessing programming is supported by OpenMP. The main aim of parallel programming system is that OpenMP should be powerful and easy to use, and on the same time it allowing the programmer to write high performance programs. At the beginning the focus was only on numerical application which involve loops written in FORTRAN or C/C++, also it includes construct necessary to deal with more kinds of parallel algorithm. It consists of compiler directives, library routines and environment variables.

The rest of the paper is divided as follows. Section 2 describes the parallel computing and which types of algorithm parallelism supports. Section 3 shows implemented algorithms and where is the parallel execution. Section 4 compares platforms on which test were performed. Section 5 deals with the measured execution times.

II. PARALLEL COMPUTING DESCRIPTION

To solve a computational problem like: a problem is broken into discrete parts that can be solved at the same time; parallel computing is used with multiple computer resources. Each part is further divided into a series of instructions. The overall control mechanism is employed only after the instructions from each part execute on different processors. Parallel computing can be done on a single computer having multiple cores or processor. High performance is achieved due to parallel computing so that time complexity will reduce.

The computer resources in parallel computing are typically: A single computer with multiple processor or cores, Random number of such computers connected through a network. For modeling, simulating and understanding complex, real world phenomena parallel computing is much better suited than serial computing.

III. ALGORITHMS PARALLELIZATION

Algorithm 2 Parallel BFS Using OpenMP

1. for all v € V do
2. D[v] ← ∞
3. d[s] ←0, level←0, f←Φ,N← Φ
4. #pragma omp parallel
5. for 0<k<m(edge) do{
6. send f_k← Φ x //shared message
7. receive f_k← Φ //mpi communication }
8. #pragma omp while
9. while f≠ Φ do {
10. #pragma omp for

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 5, ISSN No. 2455-2143, Pages 149-151
Published Online March – April 2016 in IJEAST (http://www.ijeast.com)

11. for inspect all neighbours{
12. if(node not visited) {
13. mark d[v] ←1
14. que[F,N]
}
15. else {
16. exit_node←true
}
17. #pragma omp barrier
18. #pragma omp single
19. F.nodev ← N.node++
20. N.node ← Null
}}}

In the above algorithm, we have increased the speed of algorithm execution, for making speedy algorithm we convert the serial algorithm into parallel algorithm using openMp. In this algorithm we try to cover the maximum drawback in serial BFS algorithm and had fixed it using openMp drawback such as completeness, accuracy, perfectness and mainly focus on the speed of algorithm.

In serial BFS algorithm, execution starts from the root node and then visit to next level by level while visiting the node before travel to next level, it visit the adjacent node. Therefore, BFS algorithms require large queue and memory to stored node in it. All execution takes place serially. Therefore it require more time for execution.

Therefore, we design new algorithm i.e. parallel BFS, we have divide the all level and parallelize the process and send it to different processors for compilation and execute it parallelly, then combine all processes to form a single complete result.

In BFS algorithm, in line number [4] we use directive of openMp that make algorithm parallel. In line number[5] it send all process to different processor in line number[5] it receive the all process threads from processor then carry out the further execution.in line number[7] we used directive for parallelise while loop and similarly we used for loop also then all execution take palce parallelly. Each thread execute parallelly on different processor then the executed threads are send back to root thread.

There may arise situation of overlapping and overloading of process, for overcoming this situation, in line[16] we used directive pragma barrier which synchronizes the thread as they execute and in line[17] we make use of pragma directive to transfer the executed threads in the same order as they are received from the root node.

Algorithm 4 Parallel DFS Using OpenMP

1. For all v € V do
2. d[v]=∞
3. d[s]=0,level=1,F=Φ,dN= Φ,d[v]=1
4. #pragma omp parallel
5. For 0<k<m (edge) do
{
6. Send f_k=Φ //shared message
7. Receive f_k= Φ // mpi communication
}
8. #pragma omp while
{
9. While f= Φ do
{
10. #pragma omp for
11. For n>j>1(vertices) inspect all depth node
{
12. If(node not visited [k]≠0)
{
13. Mark d[v]=1
14. Stk[d[v],dN]
15. d[v]++;
}
16. else
{
17. exit_node=true
}
18. #pragma omp barrier
19. #pragma omp single

20. F.node=d.node++
21. d.node=Null
}}}

In the above algorithm, we have increased the speed of algorithm execution, for making speedy algorithm we convert the serial algorithm into parallel algorithm using openMp. In this algorithm we try to cover the maximum drawback in serial DFS algorithm and had fixed it using openMp drawback such as completeness, accuracy, perfectness and mainly focus on the speed of algorithm.

In serial DFS algorithm, it starts the execution of the algorithm from the root node and then visit to next level by level while visiting the node before it visit the adjacent node, it visits the depth node first till the last node. Therefore, DFS algorithms require large stack and memory to store nodes in it. All execution takes place serially, therefore it require more time for execution and increase the completion.

Therefore, we design new algorithm i.e. parallel DFS, we have divided the program into different modules and parallelize the modules and pass it to different processor for compilation and execute it parallely, then combine all processes to form a single complete result accurately and having less execution time.

In DFS algorithm, on line number [3] we used directive of openMp that make the algorithm parallel. In line number[5] it send all process to different processor, in line number[6] it receives all the process threads from processors then carry out the further execution, in line number[7] we used directive for parallelise while loop and similarly we used directives for

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 5, ISSN No. 2455-2143, Pages 149-151
Published Online March – April 2016 in IJEAST (http://www.ijeast.com)

the for loop also then all executions takes palce parallelly. Each thread executes parallelly on different processor then the executed threads are send back to root thread.
There may arise situation of overlapping and overloading of process, for overcoming this situation, in line [16] we used directive pragma barrier which synchronizes the thread as they execute and in line [17] we make use of pragma directive to transfer the executed threads in the same order as they were received from the root node.

IV. MEASURED PERFORMANCE

Number of nodes	BFS			DFS		
	Serial Execution Time(TS) ms	Parallel Execution Time(PS)ms	Speed Up(TS/TP)	Serial Execution Time(TS)ms	Parallel Execution Time(PS)ms	Speed Up(TS/TP)
4	19.939	16.657	1.1970	12.8589	8.6871	1.4802
8	47.819	31.578	1.5143	23.0582	17.8416	1.2923
16	103.373	57.409	1.8006	42.3237	39.9951	1.0582
32	256.491	111.735	2.2955	327.1127	117.4268	2.7856

V. CONCLUSION

In this paper, we had parallelized the two graph traversing algorithm namely Breadth First Search and Depth First Search using OpenMP. OpenMP is used to distribute the work among different processor. As we had used different processor for a single task, the time for required to complete a single will drastically reduce as compared to serial program. Also we calculated speed ups for distinct number of nodes.

VI. REFERENCE

[1] OpenMP specifications [online]. <http://www.openmp.org/specs/>.
[2] Blaise Barney, Lawrence Livermore National Laboratory[online].
<https://computing.llnl.gov/tutorials/openMP/>
[3] Roman Mego and Tomas Fryza, “Performance of Parallel Algorithms Using OpenMP” IEEE Radioelektronika (RADIOELEKTRONIKA), 2013 23rd International Conference., pp. 236 – 239 16-17 April 2013.

[4] V. Agarwal, F. Petrini, D. Pasetto, And D>A. Bader, “Scalable graph exploration on multicore processors”,. In Proc. ACM/IEEE Conference on Supercomputing (SC10), November 2010.