ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 9, September 2013

A New Approach of Performance Analysis of Certain Graph Algorithms

M. F Mridha[1], Mohammad Manzurul Islam[2], Syed Mohammad Oliur Rahman [3]

Assistant Professor, CSE, University of Asia Pacific, Dhaka, Bangladesh [1]

Faculty of Engineering and IT, University of Technology Sydney (UTS), Sydney, Australia [2]

CSE Department, University of Development Alternative, Dhaka, Bangladesh [3]

Abstract: Computer Network based problems often require searching a node from another and finding a path from one node to another. To solve this we use graph algorithms. Solving these problems takes a lot of time and knowledge when solved manually. For this purpose graph algorithms where devised and solving these problems became easier but the time taken to solve these problems using the algorithms sequentially takes a lot of time. So to make the time consumed to be less we devised the parallel version of these algorithms and tested. In this paper, we present a new parallel Prim algorithm targeting SMP with shared address space.

Keywords: CUDA/C, prim's algorithm, bread first search, Speedup

I. Introduction

In our case the problem for searching we solve using Breadth First Search algorithm in parallel while for finding the Minimum Spanning Tree (MST) we use prim's algorithm.

MST problem has applications in network organization, touring problems, VLSI routing problem, partitioning data points into clusters and various other fields. There exist many serial and parallel algorithms for the MST problem. The first
serial algorithm for finding MST was given by Borůvka [4]. Other two commonly used algorithms are Kruskal's algorithm and Prim's algorithm [3]. Most of the existing Parallel algorithms are based on Borůvka's algorithm. Examples are Chung et. al. [5] and Chong et. al. [6]. Recently a hybrid approach (of Prim and Borůvka) was used by Bader et. al. [7]. There are several parallel formulations of Prim's algorithm [7, 8]. In this paper we design and implement a new Parallel Prim algorithm for the MST problem targeting SMP with shared memory. We use multiple threads to run algorithm in parallel. A traversal refers to a systematic method of exploring all the vertices and edges in a graph. The ordering of vertices using a breadth first search (BFS) is of particular interest in many graph problems. Theoretical analysis in the random access machine (RAM) model of computation indicates that the computational work performed by an efficient BFS algorithm would scale linearly with the number of vertices and edges, and there are several well-known implementations of BFS algorithms. However, efficient RAM algorithms do not easily translate into good performance on current computing platforms. This mismatch arises due to the fact that current architectures lean towards efficient execution of regular computations with low memory footprints, and heavily penalize memory-intensive codes with irregular memory accesses. Graph traversal problems such as BFS are by definition predominantly memory access-bound, and these accesses are further dependent on the structure of the input graph, thereby making the algorithms irregular.

A fundamental property that we use in our parallel prim's algorithm is the Cut property of MST. For any cut C in the graph, the edge with the smallest weight in the cut belongs to all MSTs of the graph. Such a minimum weight cut edge for a cut is called a light edge. If there are multiple edges with the same smallest cost, at least one of them will be in the MST. In this paper we design and implement a new Parallel Prim algorithm for the MST problem targeting SMP with shared memory. We use multiple threads to run algorithm in parallel. The algorithm non- deterministically chooses a node and sets it as root. Each thread starts growing a tree in parallel by colouring the nodes with a unique colour (called its id). When a collision occurs (a thread likes to add a node that belongs to another tree), one of the thread sends a signal to other and we merge these trees using a MergeTree operation. We force the tree grown by the thread with larger-id to merge with tree grown by a thread with smaller-id. Eventually, thread 0 will have the MST. The threads are assumed to have the capability to send asynchronous signals to each other.

II. RELATED WORK

There are several parallel implementations of Prim's algorithm. Kumar et. al. [8] pointed out that the main outer while loop of serial Prim is very difficult to run in parallel. But one can find nearest outside node in parallel by Min-Reduction and also the update-keys step can be done in parallel. The adjacency matrix is partitioned in a 1-D block fashion. (Each processor has n × n/P of the adjacency matrix and n/p of the Key Array). Each processor finds the locally nearest node, a global min reduction is done and main thread adds the nearest node to the tree and the row entry of this node in adjacency matrix is broadcast to all processors. Gonina et. al. [9] follows a very similar algorithm but instead of adding one node to the current tree, their algorithm tries to add more nodes to the tree in every pass by doing some extra computation. The algorithm finds locally K nearest outside nodes and global Min-Reduction is done to obtain globally closest K nodes. The algorithm then iterates through the list to find out whether they are valid or not.

The main point to note here is that in both parallel formulations of Prim's algorithm they are growing a single tree. Bader et. al. [7] came up with a nondeterministic shared memory algorithm which uses a hybrid approach of Borůvka and Prim algorithm. Each processor chooses a root node and grows tree in similar fashion of serial Prim approach and

when the tree finds a nearest node that doesn't belong to any other tree it can add the node, whereas if the node belongs to another tree then it must stop growing and start with a new root. In the end, we get different connected components (which are trees) and some isolated vertices. No two trees share a vertex because merging was avoided. Now Find-Min step of Borůvka Algorithm is used to shrink each of the components into a super node.

Sequential Implementations of Algorithms

To find a shortest path from s to v, we start at s and check for v among all the vertices that we can reach by following one edge, then we check for v among all the vertices that we can reach from s by following two edges, and so forth. BFS is analogous to a group of searchers exploring by fanning out in all directions, each unrolling his or her own ball of string. When more than one passage needs to be explored, we imagine that the searchers split up to explore all of them; when two groups of searchers meet up, they join forces (using the ball of string held by the one getting there first). In BFS, we want to explore the vertices in order of their distance from the source. It turns out that this order is easily arranged: use a (FIFO) queue instead of a (LIFO) stack. We choose, of the passages yet to be explored, the one that was least recently encountered.

Our first MST method, known as Prim's algorithm, is to attach a new edge to a single growing tree at each step. Start with any vertex as a single-vertex tree; then add V_1 edges to it, always taking next, the minimum weight edge that connects a vertex on the tree to a vertex not yet on the tree.

A. *Parallel Implementation of BFS*

PBFS uses layer synchronization to parallelize breadth firstsearch of an input graph G.

Let v0 ∈ V(G) be the source vertex, and define layer d to be the set Vd ⊆ V(G) of vertices at distance d from v0.

Thus, we have V0 ={v0}. Each iteration processes layer d by checking all the neighbors of vertices in Vd for those that should be added to Vd+1

```
PBFS( G,v0)
parallel for each vertex v ∈ V(G)−{v0}
    v. dist = ∞
v0. dist = 0
d = 0
V0 = BAG-CREATE()
BAG-INSERT(V0,v0)
while ¬BAG-IS-EMPTY(Vd )
    Vd+1 = new reducer BAG-CREATE()
    PROCESS-LAYER(revert Vd ,Vd+1,d)
    d = d+1

PROCESS-LAYER(in-bag, out-bag,d)
if BAG-SIZE(in-bag) < GRAINSIZE
    for each u ∈ in-bag
        parallel for each v ∈ Adj[u]
            if v. dist == ∞
                v. dist = d+1  //benign race
                BAG-INSERT(out-bag,v)
    return
new-bag = BAG-SPLIT(in-bag)
spawn PROCESS-LAYER(new-bag, out-bag,d)
PROCESS-LAYER(in-bag, out-bag,d)
sync
```

Also note that the levels are considered here which means the height vice searching for the required node. We consider the height because the when the graph is dense at each level we have 2^{d-1} nodes where d is the depth of the binary tree or the level of the binary tree but if the tree is scarce then the number of nodes will be less or equal to 2^{d-1} nodes at each level meaning more levels are in network graph.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 9, September 2013

Parallel Implementation of Prim's

```
begin
1. For each vertex, set the colour attribute to -1.
2. Create threads with unique thread-ids.
3. For each thread i and node v, set status[i] [v] = WHITE and KeyArray[i][v] = ∞
4. Child threads will run MST Algo in parallel.
4. Wait for termination of all threads.
5. Combine result of all threads.
end
```

The following algorithm gives the code to be executed by the threads.

```
1.    Choose root node non-deterministically
      If all nodes are visited return
      flag= true;
      while(flag==true)

2.    find the nearest node 'minnode' with status[i][minnode]
=GRAY
      if no node can be found return
      lock minnode

3. 1  if color[minnode] = -1 then
      block all signals
      color[minnode] = i
      status[i][minnode] = BLACK
      unlock minnode
      append minnode to treelist of 'i'

      3.1.1  for all neighbours 'v' of minnode
               if status[i][v] = WHITE
                    status[i][v] = GRAY
                    append v to treelist of 'i'
               else if status[i][v] = GRAY
                    KeyArray[i][v] = min(value[i][v],
                    AdjMat[minnode][v] );
             end for
             unblock all signals

3.2   else if color[minnode] != i then
      j = color[minnode];

      3.2.1  if i < j then
                    send signal -1 to j
                    wait till thread 'j' accepts signal
                    and executes signal handler
                    ( Mergetree(i,j) )
                    unlock minnode and kill j
      3.2.2  else if i > j then
                    send signal -2 to j
                    wait till thread 'j' accepts signal
                    and executes signal handler
                    ( Mergetree(j,i) )
                    unlock minnode and kill j
      3.2.3  else
                    unlock minnode
                    continue;
3.3   else if color[minnode] == i then
             continue;
      end while
end while
end
```

B. *Experiment*

The experiment was conducted in the CAPPLAB on GPU's Tesla C2075 (14 SMs, 14x32 = 448 cores).

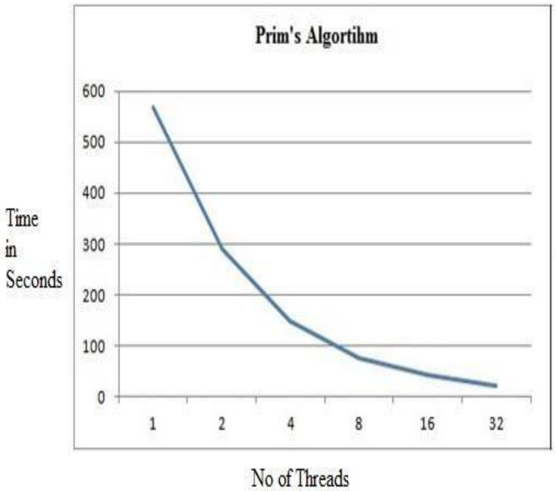
- Input: Adjacency Matrix of the Graph
- Sequential Breadth First Search
- Parallel Breadth First Search
- Sequential Prim's
- Parallel Prim's
- Analyze the output
- Speed up Computation

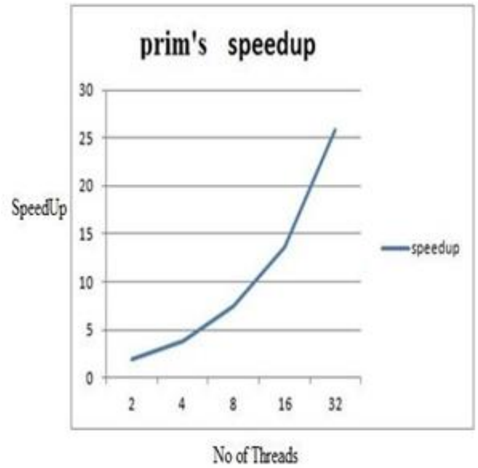
The results obtained in the lab that are given below.

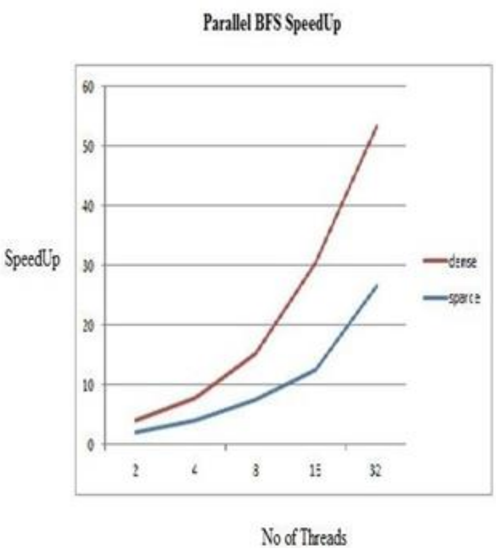
```
begin
1. For each vertex, set the colour attribute to -1.
2. Create threads with unique thread-ids.
3. For each thread i and node v, set status[i] [v] = WHITE and KeyArray[i][v] = ∞
4. Child threads will run MST Algo in parallel.
4. Wait for termination of all threads.
5. Combine result of all threads.
end
```

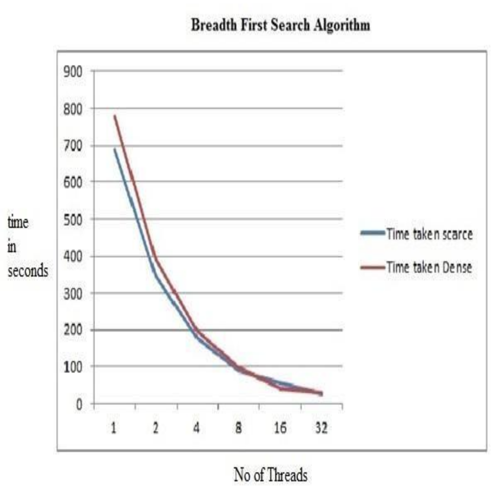
ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 9, September 2013









In future other graph based algorithms will be taken up for parallelization.

III. CONCLUSION

In this paper we presented a new parallel Prim algorithm that grows multiple trees in parallel. We made simple observations based on the cut property of the graph to grow MSTs in parallel. Our algorithm achieves reasonable speedup when it is compared with Serial Prim algorithm for dense graphs and sparse graphs. Breadth First Search and Prim's Algorithm's parallel implementations using CUDA/C was successfully done. The Speedup computed helped realize performance improvements by the use of parallel algorithms. In case of breadth first search algorithm in parallel when graph is sparse speed up is 2.0 while that of when graph is dense 1.9. As for as prim's is concerned speedup is at the minimum of 1.96 i.e 2.0 more when at least 2 threads are used.

REFERENCES

[1]. Rohit Setia, Arun Nedunchezhian, Shankar alachandran, *"A New Parallel Algorithm forMinimum Spanning Tree Problem"*, in HIPC 2009, Kochi, Kerala, India.

[2]. Robert Sedgewick and Kevin Wayne, "*Algorithms*", Addison-Wesley, 4th Edition, 2011.

[3] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms. MIT Press,Cambridge, MA.1990

[4] Otakar Borůvka. O jistem problem minimalnim (About a certain minimal problem). Prace mor. přirodověd. spol. v Brně III 3: 37– 58.1926

[5] Sun Chung and Anne Condon. Parallel implementation of Bor°uvka's minimum spanning tree algorithm.(IPPS'96)

[6] Ka Wong Chong, Yijie Han, Yoshihide Igarashi, and Tak Wah Lam. 1999. Improving the Efficiency of Parallel Minimum Spanning Tree Algorithms. Discrete Applied Mathematics. 2003

[7] David A. Bader, and Guojing Cong.Fast. Shared-Memory Algorithms for Computing the Minimum Spanning Forest of Sparse Graphs. JPDC, 2006

[8] G. Karypis A. Grama, A. Gupta and V. Kumar. Introduction to Parallel Computing.Addison Wesley, second edition, 2003.

[9] Ekaterina Gonina and Laxmikant V. Kale. Parallel Prim's algorithm on dense graphs with a novel extension. Technical Report. Department of Computer Science, University of Illinois at Urbana-Champaign. November 2007.