Introduction:

About Breadth First Search - Algorithm for exploring the edges and vertices in a graph, given a source. Some of the existing algorithms for the same:

1. Serial Breadth First Search

2. Centralized Queues Parallel Breadth First Search

3. Decentralized Parallel Breadth First Search

4. Distributed randomized work stealing Parallel Breadth First Search

There is also another approach by MIT [2] and the following employs a special data structure known as bags.

Parallel Breadth-First Search - The IDEA

Source : Ref [3]

The idea of doing BFS in parallel is that, in principal, you can process all the vertices on a single level at the same time. That is, once you've found all the level-1 vertices, you can do a parallel loop that explores from each of them to find level-2 vertices. Thus, the parallel code will have an important sequential loop over levels, starting at 0.

In the parallel code, it's possible that when you're processing level i, two vertices v and w will both find the same level-i+1 vertex x as a neighbor. This will cause a data race when they both try to set level[x]=i+1, and also when they each try to set parent[x] to themselves. But if you're careful this is a "benign data race" -- it doesn't actually cause any problem, because there's no disagreement about what level[x] should be, and it doesn't matter in the end whether parent[x] turns out to be v or w. To avoid the segmentation errors because of data races,we employ locks. There are also lock free implementations of the algorithms mentioned above which use further modifications of queue structures and deliver superior efficiency.

Implemented algorithms -

1. Centralized Queues with Locks and distributed scheduling of work (minor modifications to the traditional algorithm)

2. Distributed randomized work stealing with Locks

Outline of the algorithms - paper [1]

System Used-

Intel E7

Number of Processor Cores = 32

Operating System: Linux (CentOS release 6.4, 2.6.32 Kernel

Processor spec: 32x2.13GHz Intel Xeon E7-4830 Processor Cores

RAM = 256GB

Codes -

pbfs.cpp (Centralized Queue Parallel Breadth First Search)

Main.cpp (Distributed randomized work stealing approach)

Version of gcc - 4.9.2

Compiling - g++ "filename" -o "OutputName" -fcilkplus -std=c++11

Note: No optimization flags have been used as shown in the implementations in the paper, to test the efficiency of algorithm without compiler effects.

Results and Conclusions-

Results - Document.pdf in the folder

In the two algorithms that have been implemented, parallel bfs with work stealing approach has the better efficiency.The algorithms work well with sparse graph data too. All test cases have been checked for connected components and ensured that there are no disjoint set of graphs. Correctness of the algorithms are ensured by checking if every node in the graph is visited or not.

Performance evaluation: There are few drawbacks for the algorithm. In the case of small data sets and less number of processors, the process is costly and performs with less or equal efficiency as the centralized queues parallel breadth first search algorithm. Work stealing is only efficient when the data size is comparable with a good number of processors. The use of locks also reduce the efficiency and scalability to a great extent. Also the overhead of locks increases at a faster rate in the centralized queue implementation compared to lock-based work stealing approach. Hence the efficiency of centralized queue implementation drops compared to the workstealing approach when the size and processors number increase.The current implementations are scalable only till 16 cores because of the use of locks. Though the lock wait time is O(1) in the case of work steal using try_locks, the centralized approach wait time can go over to O(p), p = number of processors. The lockfree implementations of centralized implementation is found to be scalable till 20 cores and the workstealing approach to be scalable till 32 cores [1].

References -

1. "Avoiding Locks and Atomic Instructions in Shared-Memory Parallel BFS

Using Optimistic Parallelization" 2013 IEEE 27th International Symposium on Parallel & Distributed Processing

2. "A Work-Efficient Parallel Breadth-First Search Algorithm (or How to Cope with the Nondeterminism of Reducers)"

Charles E. Leiserson; Tao B. Schardl

3. UCSB - http://www.cs.ucsb.edu/~gilbert/cs140/old/cs140Win2011/bfsproject/bfs.html