BREADTH FIRST SEARCH USING 1 D PARTITION

Course: CSE 633 Parallel Algorithms
Presenter: Shalini Agarwal
Instructor: Dr. Russ Miller
CONTENTS

• About BFS
• Applications of BFS
• Sequential BFS implementation
• Why parallel BFS?
• Parallel BFS implementation
• Future Work Status
• Execution Results
• References
Bread First Search (BFS)

- Algorithm for traversing or searching graph data structures.
- A traversal refers to a systematic method of exploring all the vertices and edges in a graph.
- Explores the vertices at the current level before proceeding to the next level.
- Extra memory is needed to keep track of the child nodes (vertices) encountered but not yet explored.

Applications of BFS

- Shortest Path: Used to find the shortest path between two vertices in an unweighted graph.
- Social Networking: Used to find the shortest path between two users in a social network. Also, can be used to find the connected components in the network.
- Game Theory: Used to find the shortest path to reach the goal state in games such as Chess, Checkers.
- Peer-to-Peer Networks: Used to find the all the neighboring nodes in peer to peer networks like BitTorrent.
- Web Crawlers: Crawlers build search index using BFS. They start from the source page and continue to follow all the links from the source.
- GPS Navigation System: BFS is used to find all the neighboring locations.
Sequential BFS implementation

- Create an empty queue and add the starting vertex to the queue.
- Create a visited set to keep track of the visited vertices.
- Mark the starting vertex as visited and add it to the visited set.

Refer: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9269471/
• While the queue is not empty, do the following:
  a. Dequeue a vertex from the queue.
  b. For each adjacent vertex of the dequeued vertex that
     is not already visited, do the following:
     i. Mark the adjacent vertex as visited and add it to
        the visited set.
     ii. Enqueue the adjacent vertex to the queue.
• Repeat the previous step until the queue becomes
  empty.

Refer: [https://en.wikipedia.org/wiki/Parallel_breadth-first_search](https://en.wikipedia.org/wiki/Parallel_breadth-first_search)
Why parallel BFS?

- Improved performance: Parallel BFS improves performance by processing multiple nodes in parallel.
- Scalability: It is highly scalable and can handle large graphs or trees efficiently.
- Concurrency: Parallel BFS allows for concurrent exploration, minimizing idle time and maximizing resource utilization.
- Load balancing: This ensures efficient utilization of computational resources.
- Flexibility: Parallel BFS can be implemented in various ways, making it flexible to adapt to different hardware configurations.
Parallel BFS implementation

• Similar to sequential BFS implementation, but instead of checking the queue of vertices sequentially, we implement this in parallel across all the vertices at the same level.

• A level-synchronous strategy that relies on a simple vertex-based partitioning of the graph.

• Each processor \( (p_i) \) with distributed memory will oversee \( n/p \) vertices or graph nodes. \((n = \text{number of vertices}; \ p = \text{number of processors})\)

\[
n = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8\}
\]
\[
p = \{p_1, p_2, p_3, p_4\}
\]
\[
n/p = 8/4 = 2 \text{ i.e. 2 vertices for each processor}
\]
Parallel BFS implementation

• The processor will store partitioned vertices in a 1 D array structure where each vertex will have a row of outgoing edges represented by destination vertex index.

• Frontier will store the vertices which are at the same distance from the source vertex.

• Next Frontier will contain the unexplored neighbors of the vertices from the Frontier.
Parallel BFS implementation

- A neighbor vertex from one processor may be stored in another processor; hence each processor needs to communicate to those processors about the traversal status.
- Each processor should also receive communication from all other processors to construct the next frontier.
- This requires an all-to-all communication after every step of analyzing the frontier.
- The algorithm ends when the global size of the frontier is zero.

```c
BFS_distributed_ID (local G = (V, E), vertex s)
{
    frontier = {}; next_frontier = {}
    curr_level = 0
    for all v belongs to V:
        level[v] = -1;
        if owner(s) = curr_rank:
            level[s] = 0
            frontier.add(s)
    while True
    {
        // contains the local vertices in the current frontier
        for u belongs to frontier:
            for v belongs to neighbor(u):
                w = owner(v)
                buffer[w].add(v)
        // send & receive buffers to the respective processors
        All to all v (buffer, receive_buffer)
        for all p = [0 .... numRank - 1]:
            next_frontier.merge(receive_buffer[p])
        frontier = {}
        for v belongs to next_frontier:
            if level[v] == -1:
                level[v] = curr_level + 1
                frontier.add(v)
        next_frontier = {}
        curr_level ++
        size = frontier.size()
        AllReduce(size); //sum
        if (size == 0):
            break out of the while loop;
    }
}
Future Work Status

• [Completed] Irregularity in execution time needs investigation for potential code or execution environment errors.

• [Completed] Larger graphs with more vertices need to be tested to ensure scalability.

• [Completed] Test the algorithm on a higher number of processors for performance evaluation.

• [Completed] Calculate the speed up of the parallel BFS algorithm compared to the sequential BFS algorithm.

• [Completed] Troubleshoot the Slurm script to make it functional.
Execution Results: Running Time

Number of graph vertices: 100
Number of nodes: 125

Number of graph vertices: 500
Number of nodes: 125
Execution Results: Running Time

Number of graph vertices: 1000
Number of nodes: 125

Number of graph vertices: 1500
Number of nodes: 125
Execution Results: Speed Up

\[
\text{Speed Up} = \frac{\text{Seq. Exec. Time}}{\text{Parallel Exec. Time}}
\]

Number of Nodes = 125  
Graph Vertices = 100; 500; 1000; 1500
Execution Results: Cost

\[ \text{Cost} = \text{Execution Time} \times \text{No. of Processors} \]
References

- https://people.eecs.berkeley.edu/~aydin/sc11_bfs.pdf [Parallel Breadth-First Search on Distributed Memory Systems]
- https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1559977 [A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L]
- https://www.youtube.com/watch?v=wpWvCabHqQU [Distributed BFS Algorithm, IIT Delhi July 2018]
- https://en.wikipedia.org/wiki/Parallel_breadth-first_search
- http://ijrar.com/upload_issue/ijrar_issue_1836.pdf [Graph Traversals and its Applications]
- https://docs.ccr.buffalo.edu/en/latest/
- https://cse.buffalo.edu/faculty/miller/Courses/CSE529/Spring-2023/syllabus.html
- https://devdocs.io/c/