PARALLEL BREADTH FIRST SEARCH USING MPI

Course: CSE 708
Presenter: Venkata Bala Vamsi
Instructor: Dr. Russ Miller
CONTENTS:

1. Overview of BFS
2. Applications of BFS
3. Sequential Approach to BFS
4. The Necessity of Parallelization
5. Parallel Implementation of BFS
6. Results
Overview of BFS

- Given a source node, BFS performs a Level Order traversal of the graph with respect to the source node.
- Explores all the vertices in the current level before moving on to exploring vertices in the next level.

BFS Traversal: S A B C D
Applications of BFS

1. Shortest Path
2. Cycle Detection
3. Finding connected components
4. Network Broadcast
Sequential approach to BFS

```cpp
int visited[all_nodes] = {0};
queue<int> Q;
Q.push(source_node);
visited[source_node] = 1;

while(Q is not empty){
    int current_node = Q.front();
    Q.pop();
    for (each adj_node of current_node){
        if (visited[adj_node] == 0){
            Q.push(adj_node);
            visited[adj_node] = 1;
        }
    }
}
```
The Necessity of Parallelization

- **Memory Constraints**: Large Graphs can exceed memory capacity of a single processor.
- **High Computational Demand**: BFS is computationally intensive because it explores each vertex and edge of the graph.
Sequential Implementation:

```cpp
int visited[all_nodes] = {0};
queue<int> Q;
Q.push(source_node);
visited[source_node] = 1;

while(Q is not empty){
  int current_node = Q.front();
  Q.pop();
  for each adj_node of current_node{
    if(visited[adj_node] == 0){
      Q.push(adj_node);
      visited[adj_node] = 1;
    }
  }
}
```

High level Parallel Implementation:

```cpp
vector<bool> visited(n, false);
vector<int> frontier; // nodes in current level
vector<int> next_frontier; // nodes in next level

int curr_level = 0;
frontier.push_back(source_node);
visited[source_node] = true;

while(!frontier.empty()){ //for(int node : frontier){
  for(int adj_node : adj[node]){ //for(int node : frontier){
    if(visited[adj_node]){ //if(visited[adj_node]){
      visited[adj_node] = true;
      next_frontier.push_back(adj_node);
    }
  }
}

frontier = next_frontier;
next_frontier.clear();
curr_level++;
```
Graph nodes are distributed evenly across the available processors.

Each Processor maintains its own:

- **Local adjacency list**, which corresponds to the subset of vertices it is responsible for.
- **Vertex levels**, specifically for the vertices it owns, denoting their distances from the source vertex.
- **Current frontier**, which is a list of vertices it owns that are to be explored at the current level of the algorithm.
- **Next frontier**, which comprises the vertices it owns that will be explored in the subsequent level of the algorithm.
Processor 0:
Levels = { 0: -1, 1: -1, 2: -1}
FS = {0}
ALL to ALL communication
Levels = { 0: 0, 1: -1, 2: -1}
FS = {1}
ALL to ALL communication
Levels = { 0: 0, 1: 1, 2: -1}
FS = {2}
ALL to ALL communication
Levels = { 0: 0, 1: 1, 2: 2}
FS = {}
ALL to ALL communication

Processor 1:
Levels = {3: -1, 4: -1, 5: -1}
FS = {0}
ALL to ALL communication
Levels = {3: -1, 4: -1, 5: -1}
FS = {1}
ALL to ALL communication
Levels = {3: 1, 4: -1, 5: -1}
FS = {2}
ALL to ALL communication
Levels = {3: 1, 4: -1, 5: 2}
FS = {3}
ALL to ALL communication
Levels = {3: 1, 4: 3, 5: 2}
FS = {}
ALL to ALL communication

Processor 2:
Levels = {6: -1, 7: -1, 8: -1}
FS = {0}
ALL to ALL communication
Levels = {6: -1, 7: -1, 8: -1}
FS = {1}
ALL to ALL communication
Levels = {6: -1, 7: -1, 8: -1}
FS = {2}
ALL to ALL communication
Levels = {6: 2, 7: -1, 8: -1}
FS = {3}
ALL to ALL communication
Levels = {6: 2, 7: -1, 8: -1}
FS = {4}
ALL to ALL communication
Levels = {6: 2, 7: 4, 8: 3}
FS = {}
ALL to ALL communication

Levels = {6: 2, 7: 4, 8: 3}
FS = {7}
ALL to ALL communication
Levels = {6: 2, 7: 4, 8: 3}
FS = {}
Slurm.sh:

#!/bin/sh
#SBATCH --ntasks-per-node=1
#SBATCH --nodes=16
#SBATCH --time=00:03:00
#SBATCH --job-name=bfs_sparse_5000_vertices_16:1_nodes
#SBATCH --output=bfs_sparse_5000_vertices_16:1_nodes.out
#SBATCH --mem=5000M
#SBATCH --partition=general-compute
#SBATCH --qos=general-compute
#SBATCH --mail-type=END
#SBATCH --mail-user=vatukuri@buffalo.edu
#SBATCH --cluster=ub-hpc
#SBATCH --exclusive

module load intel
module list
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
source /util/academic/intel/20.2/compilers_and_libraries_2020.2.254/linux/mpi/intel64/bin/mpivars.sh
mpicxx -o parallel_bfs_mpi parallel_bfs_mpi.cpp
srun -n 16 ./parallel_bfs_mpi
Results for 5120 sparse graph vertices

Execution Time vs No:of Processors for 5120 sparse graph

Speedup vs No:of Processors for 5120 sparse graph

1 task per node

Speedup = Sequential Time / Parallel Execution Time
Results for 7520 sparse graph vertices

1 task per node

Speedup = Sequential Time / Parallel Execution Time
References:

- [https://people.eecs.berkeley.edu/~aydin/sc11_bfs.pdf](https://people.eecs.berkeley.edu/~aydin/sc11_bfs.pdf) [Parallel Breadth-First Search on Distributed Memory Systems]

- [https://www.youtube.com/watch?v=wpWvCabHqQU](https://www.youtube.com/watch?v=wpWvCabHqQU) [Distributed BFS Algorithm, IIT Delhi July 2018]

- [https://docs.ccr.buffalo.edu/en/latest/](https://docs.ccr.buffalo.edu/en/latest/)

Thank You. Questions?