BFS

Start simple. Grow Complex.

Rationale

- In HPC, search comes up in various problems. Some problems that involve search:
  - Internet
  - Social Networks
  - Protein Modeling
- These applications must be run at scale. The search algorithm must be able to handle graphs with various properties- varying sparsity or high-degree nodes.
- I’ve chosen BFS as a search algorithm to focus my project because of its guarantee to find the shortest path in an unweighted graph

BFS- Methodology Overview

- Development has been a continuous process but I've split my development into 2 iterations
  - Standalone BFS (No Path Information)
  - BFS with Path Information (Igraph, External Graphs such as opte-out.lgl)
- The strategy for completing the second iteration is built off the work in the first iteration

Standalone BFS - Methodology

- Creation of CSR Format = generated graphs each iteration in this compressed format
  - Percent = Allows user to vary the edge to vertex ratio in the graph
  - No. of edges = No. of Nodes * No. of Nodes * Percent
- After broadcasting graph information to other processes, I implemented the following pseudocode.

```
create levelBuf -> 0 to no_of_nodes;
While (new Vertex Visited - nVV):
        Run BFS on Frontier
        MPI_Allgather(nVV)
        If (compare)
                MPI_Alltoall(level,my_rank)
                aggregate(levelBuf, my_rank)
                MPI_Allgather(nVV)
```

Standalone BFS - Problems

- Scale was too small = At most, I can only do ~46,000 nodes.
- Graph generation was expensive = At higher levels of nodes, the graph generation took the majority of time
- Graph generation was structured = Most graphs had consistent edge to vertex ratio
- No path information = Could determine if the nodes were connected but not the path from one node to another

BFS with Path Information and IGRAPH

- To address these problems, I shifted away from creating graphs manually and focused on using Igraph - a powerful graph library- to generate graphs
    - IGRAPH handled two of these problems - scale and graph generation
    - I was able to make graphs with up to 50 Million nodes
- This speedup allowed me to focus on other important information such as shortest path information between two nodes
- I imported an external graph (opte-out.lgl) to analyze graphs with different structure

BFS with Path - Methodology

- Similar to Standalone, I worked with CSR formats for the graphs. However, the pseudocode is different this time around.

```
create levelBuf -> 0 to no_of_nodes;
create parent -> 0 to no_of_nodes;
create open -> 0 to no_of_nodes;
create closed -> 0 to no_of_nodes;
While (new Vertex Visited - nVV):
        Run BFS on Frontier
        MPI_Allgather(nVV)
        If (compare)
                MPI_Alltoall(open,my_rank)
                MPI_Alltoall(closed,my_rank)
                MPI_Alltoall(levelBuf,my_rank)
                MPI_Alltoall(parent,my_rank)
                update_open(open, my_rank)
                update_closed(closed, my_rank)
                aggregate(levelBuf, my_rank)
                aggregate_parent(parent, levelBuf, my_rank)
                MPI_Allgather(nVV)
```

BFS- Results

Problem Size	BFS 1 Serial	BFS 1 MPI	BFS 1 CUDA	BFS 1 MPI CUDA	BFS 2 MPI	BFS 2 CUDA	BFS 2 MPI CUDA
10000	2	1	37	97	1	60	156
100000					24	61	1443
1000000					225	235	1285

BFS - conclusions

- Bfs is a powerful algorithm for search
  - Can be scaled to large graph problems such as Internet and Social Media Networks
- Can be parallelized with MPI + CUDA - although the speedup is not linear
- IGRAPH can take away the heavylifting when making CSR graphs
- Path information adds to the time but provides insight into shortest path

BFS -Future Direction

- Implement BFS to interesting graph structures such as N Puzzle
- Spread out to different variations of BFS such as Bi-directional BFS
- Implement other search algorithms to compare time cost