A Comparison of Sequential vs GPU Optimized Breadth First Search using Astronomy Data

Tyler Hilbert, *MEng, Comp. Eng.,* Tylor Beatty, *BS, Comp. Eng.,* Matt Nemeth, *BS, Comp. Eng.*

***Abstract*—This paper compares the run-time of a breadth first search algorithm running on a CPU to a breadth first search algorithm that is optimized to run on a GPU using CUDA. The dataset used consists of a list of astronomy measurements taken from Earth. A graph of galaxies connected by a distance below a threshold was created from this dataset. Then, the two algorithms were tested for run-time efficiency using the constructed graph.**

I. An Explanation of the Problem and Dataset

INTERGALACTIC space travel is technology of the future. Inevitably, we will find a way to travel between galaxies. One of the difficulties that comes with this new ability is path planning. When intergalactic space travel becomes feasible, we will have a maximum distance that can be traveled. For example, we may design spaceships that can travel 1 light year before needing to refuel. But what if you want to travel to a galaxy that is 5 light years away? You will have to make stops at other galaxies along the way.

Our team set out to write a program[1] that can compute the optimal path when travelling across the universe. Our program attempts to find the path with the fewest stops between 2 galaxies. The purpose of this is to stop the least amount of times to refuel. You can think of it as Google Maps for space travel, getting you to your destination galaxy the fastest.

The first challenge is determining the distance between each galaxy. To do this, we relied only on observations that can be made from Earth. Two datasets were stitched together to create a dataset with the following fields:

1) **Right Ascension (RA)** – Used to calculate the angle from Earth to the galaxy (ascension).
2) **Declination Value (DEC)** – Used to calculate the angle from Earth to the galaxy (declination).
3) **Light Years** – Light years from Earth to the galaxy.

Using these 3 values, the distance between 2 observed galaxies can be measured by using the following equations.

Right Ascension:

$$\text{degrees}(\Theta) = 15\left(h + \frac{m}{60} + \frac{s}{3600}\right) \tag{1}$$

where h is the hour measurement, m is the minute measurement, and s is the second measurement.

Declination Value:

$$\text{degrees}(\Phi) = deg + \frac{m}{60} + \frac{s}{3600} \tag{2}$$

[1]Code: https://github.com/Tyler-Hilbert/SpaceTripPlanner

Galactic Position on the Cartesian Plane:

$$x = r\cos(\Theta)\cos(\Phi) \tag{3}$$
$$y = r\sin(\Theta)\cos(\Phi) \tag{4}$$
$$x = r\sin(\Phi) \tag{5}$$

where r is the distance in light years from Earth.

Distance between 2 Galaxies Given the Coordinates:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{6}$$

Using the equations above, we calculated the distances between galaxies. A graph was created with edges between the galaxies that were within the travelling distance of the spaceship. For example, we assumed that our spaceship can travel 1 light year before needing to stop at a galaxy for a refuel break. Using that assumption, the following graph shown in Figure 1 was made. The edges show the galaxies that are less than 1 light year apart.

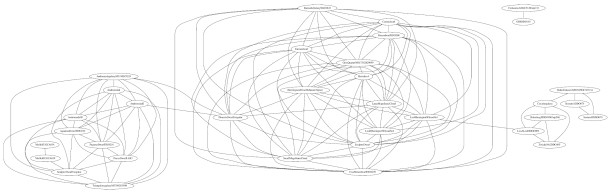

Fig. 1. Visual Representation of the Testing Graph.

It was decided that the overall distance travelled was less costly than the number of stops. The determination of this metric meant that the goal would be to minimize the number of stops. After careful discussion, it was found that a breadth first search would be an ideal algorithm for this problem. As some background, a breadth first search is a searching algorithm that looks outwards 1 node at a time from a source. As a result of this fact, it will always find a solution that has the minimal amount of in-between nodes in a unweighted graph. For a sequential BFS, the time complexity is $O(|V| + |E|)$ which we can consider as $O(n)$ where V is the vertices and E is the edges. Using a parallel breadth first search algorithm, our work complexity value is $O(n^2)$ and our step complexity value is $O(n)$. Using a parallel algorithmic approach, we are able to improve the speed of completion because we are able to use individual threads to check certain paths instead of returning to a point and checking another path on the same thread.

II. High Level Design and Examination of the Parallelized Sections

This project contains three parts. The first is gathering the data from [1] and [2]. The next is a Python3 script used for preprocessing the "GalaxiesWithDistances.csv" file. This is where all calculations are done to create a graph with all the galaxies that are within the specified distance. The preprocessing script then formats the graph data and generates both a CUDA and C++ file that contains graph initialization data. These generated data structures then need to be copied and pasted into the correct CUDA or C++ BFS file. This was done because it was determined that it would be quicker to develop with the assumption that the preprocessor could run entirely separate from the CUDA/C++ programs to alleviate the need for any variable input into CUDA/C++. Various resources were used for technical background [3]–[8].

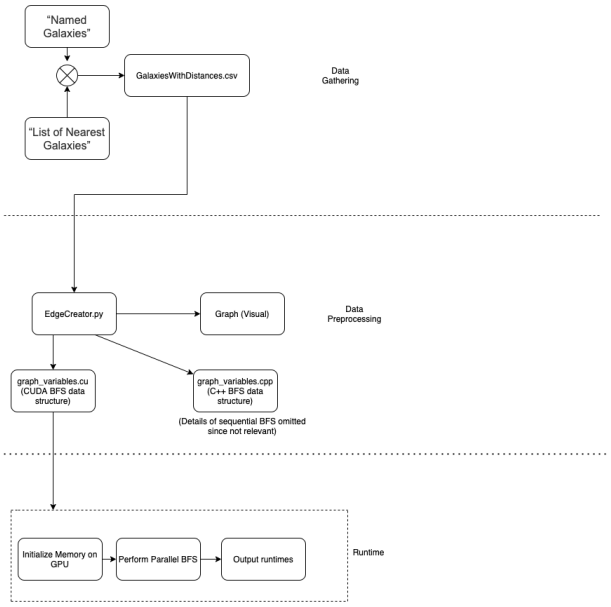

Fig. 2. High Level Flow Diagram.

In Figure 3, an example of the parallelism the algorithm achieves is given. Each step further from the chosen source node is processed in parallel. Depending on the choice of graph, the algorithm can provide large performance improvements. However, for very linear graphs with few branches, it would be expected to run similarly to a sequential algorithm since computations will not be able to be done in parallel if there are not enough edges. A logarithmic improvement was expected assuming that each node has many edges.

Algorithm 1 is the pseudocode for the algorithm's implementation. The first loop is done on the CPU sequentially and continues until the GPU kernel returns true for a completion flag. The body of this loop is the GPU kernel and is the code that is implemented in parallel for each marked, unvisited node. A technical challenge that was encountered with this code was the potential race condition for the processing of a neighboring/child node, specifically in the context of writing the previous path of nodes. This issue was mitigated using an atomic compare-and-swap instruction, ensuring the node could only ever be processed a single time.

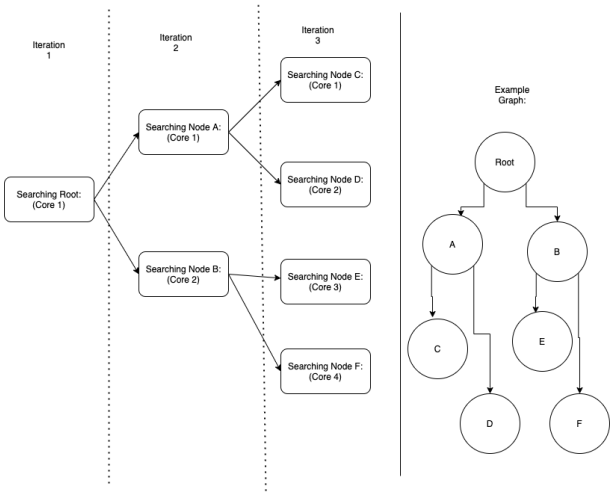

Fig. 3. Example Execution of Parallel BFS Algorithm.

```
Frontier[0] <- true
Loop until done
    Done <- true
    For each node=true in Frontier
        Mark as visited
        Remove from frontier list
        For each child node of Frontier Node
            If has not been visited
                Set distance from source to Frontier Node + 1
                Set ID string list to FronterNode_ID_List + Frontier_Node_ID
                Add to frontier list
                Done <- false
```

Fig. 4. Text Diagram for Parallel BFS Algorithm.

Figure 4 shows an example of pseudocode for the algorithm's implementation. The first loop is done on the CPU sequentially and continues until the GPU kernel returns true for a completion flag. The body of this loop is the GPU kernel and is the code that is implemented in parallel for each marked, unvisited node. A technical challenge that was encountered with this code was the potential race condition for the processing of a neighboring/child node, specifically in the context of writing the previous path of nodes. This issue was mitigated using an atomic compare-and-swap instruction, ensuring the node could only ever be processed a single time.

III. Results

To evaluate the parallel performance of the algorithm, a comparison was made between a parallel and a sequential BFS. The run-times of these algorithms were compared using different starting points from the same galaxy dataset. The following equation was used to calculate the performance increases of the parallel algorithm over the sequential algorithm:

$$\text{Performance} = 100\left(\frac{S-P}{S}\right) \tag{7}$$

TABLE I
TABLE COMPARISON OF PARALLEL AND SEQUENTIAL BFS

	Parallel BFS Average (ms)	**Sequential BFS Average** (ms)	**Performance Increase** (%)
00	2.029	8.612	77.000
10	1.572	6.230	74.000
20	1.599	6.750	76.000
25	1.798	7.286	75.000

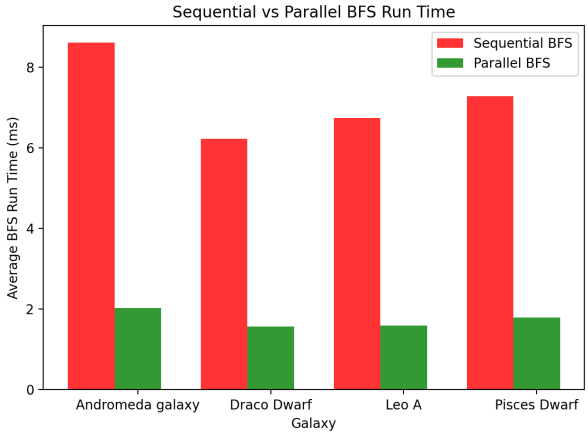

Fig. 5. Graph of Parallel and Sequential BFS Run-times.

where P is parallel run-time and S is sequential run-time.

Figure 5 shows the performance increase of the parallel over the sequential BFS algorithms. For the galaxy dataset used in this report, there was around a 75% performance increase. Please note that the performance increase is dependent on the dataset used. This dataset has 34 nodes and 272 edges. The ratio of edges to nodes is 8:1. One would expect the performance increases to decrease while the ratio gets closer to 1:1 and increase as the ratio approaches ∞:1. Also note that the performance will start to get diminishing returns as the ratio approaches ∞:1 due to limitations in the number of cores and other hardware constraints.

IV. CONCLUSION

Overall, the amount of improvement the parallel BFS showed over the sequential BFS was significant and what one would hypothesize to expect. The reason for this performance increase is that a parallel BFS algorithm can compute frontiers being explored that are the same distance away at the same time, while a sequential BFS algorithm has to explore each one separately. This means that the parallel algorithm can examine a larger number of galaxies than the sequential algorithm in the same amount of time. One possible improvement to this paper would involve running the algorithm on a dataset where the spaceship is able to travel different distances. For example, 5 light years instead of 1 light year. This would give a better representation of how the performance of the algorithm will scale. Another possible improvement would be to do the preprocessing on CUDA instead of Python. Theoretically, the edges could be calculated during run-time in parallel instead of as a preprocessor script that runs sequentially. This was not implemented for this paper because it was assumed that all preprocessing of the galaxy data had already been performed and thus did not need to be optimized.

REFERENCES

[1] B. F. Madore, "Named Galaxies," [Online]. Available: https://ned.ipac.caltech.edu/level5/CATALOGS/naga.html. [Accessed: 06-20-2020].
[2] Wikipedia, "List of Nearest Galaxies," [Online]. Available: https://en.wikipedia.org/wiki/list_of_nearest_galaxies. [Accessed: 06-20-2020].
[3] R. Kaszuba, "Implementation of breadth first search on GPU with CUDA Driver API," [Online]. Available: https://github.com/rafalk342/bfs-cuda.
[4] P. Harish, P.J. Narayanan, "Accelerating Large Graph Algorithms on the GPU Using CUDA," In: S. Aluru, M. Parashar, R. Badrinath, V.K. Prasanna High Performance Computing. Springer, Berlin. 2007.
[5] J. Sanders, "CUDA by Example: an Introduction to General-Purpose GPU Programming," In: Addison-Wesley/Pearson Education. 2015.
[6] S. Srinivas, "Implementing Breadth First Search in CUDA," [Online]. Available: https://siddharths2710.wordpress.com/2017/05/16.
[7] Udacity, "BFS Try #1 - Intro to Parallel Programming," [Online]. Available: https://youtu.be/SktpTf2MoIw. [Accessed: 06-20-2020].
[8] Udacity, "Merrills Linear-Complexity BFS on GPUs Part 1 – Intro to Parallel Programming," [Online]. Available: https://youtu.be/Dq0ImVxQsRo.