Space Trip Planner
Project Report

A Comparison of Parallel and Sequential Breadth First Search using Astronomy Data

Group Members:
Tyler Hilbert
Tylor Beatty
Matt Nemeth

Last Edited: 12/6/2020
Submitted: 12/6/2020

Table of Contents

1. Title Page: Space Trip Planner Project..……1
2. Table of Contents……..……..2
3. Introduction: An Explanation of the Problem and Dataset……..….3
4. High Level Design and Examination of the Parallelized Sections……6
5. Results……..…8
6. Division of Work and Self Assessment……...10
7. References……..12
8. Code Appendix……...12

List of Tables

1. *Table A.* Comparison of Parallel and Sequential BFS Performance……....8

List of Figures

1. *Figure A.* Visual representation of the graph used for testing……...4
2. *Figure B.* High Level Flow Diagram……...6
3. *Figure C.* Example execution of parallel BFS algorithm……7
4. *Figure D.* Text Diagram for the parallel BFS algorithm……...7
5. *Figure E.* Graph of Parallel and Sequential BFS Runtimes……...9

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 first becomes feasible, we will have a maximum distance that can be traveled. For example, we may design space ships 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'll have to make stops at other galaxies along the way.

Our team set out to write a program 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:

Right Ascension (RA) - This is the right ascension value. It is used to calculate the angle from Earth to the galaxy

Declination Value (DEC) - This is the declination value. It is also used to calculate the angle from Earth to the galaxy.

Light Years - This is the measurement in light years from Earth to the galaxy.

Using these 3 values, the distance between 2 observed galaxies can be measured.

Right Ascension:

$$degrees\ (\theta)\ =\ 15(h + \frac{m}{60} + \frac{s}{3600})$$

Where:

h is hour measurement of ra
m is minute measurement of ra
s is second measurement of ra

Declination Value:

$$degrees\ (\phi)\ =\ deg + \frac{m}{60} + \frac{s}{3600}$$

Where:

m is minute measurement of dec

s is second measurement of s

Galactic Position on the Cartesian Plane:

$$x = r * cos(\theta) * cos(\phi)$$
$$y = r * sin(\theta) * cos(\phi)$$
$$z = r * sin(\phi)$$

Where:

r is the distance in light years from Earth

Distance Between 2 Galaxies Given the Cartesian Coordinates

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

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 was made. The edges show the galaxies that are less than 1 light year apart.

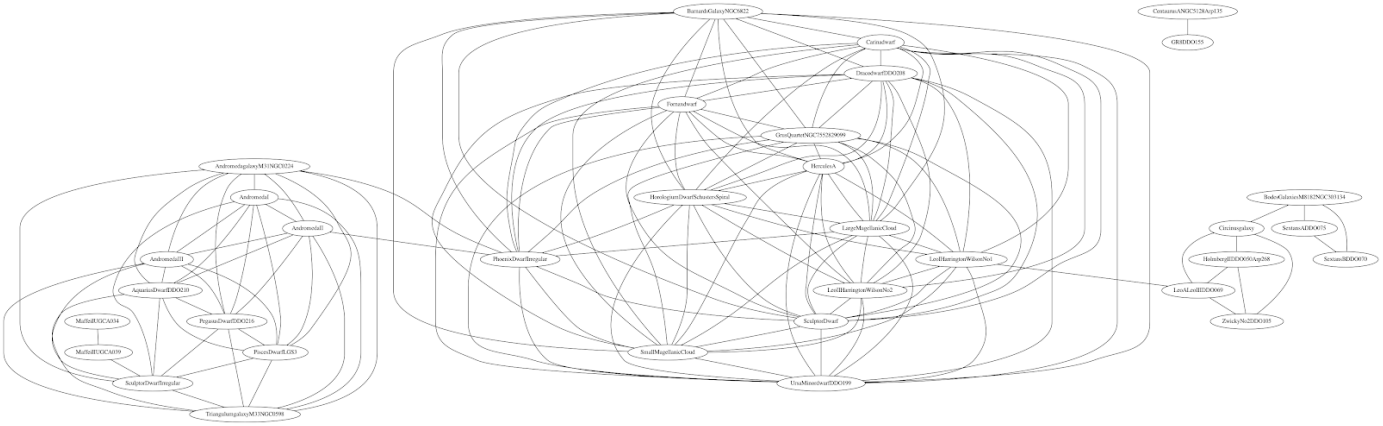

Figure A. Visual representation of the graph used for testing

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 Breadth First Search Algorithm, the time complexity is O(|V|+|E|) which we can consider as O(n), where V is Vertices and E is 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 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.

High Level Design and Examination of the Parallelized Sections

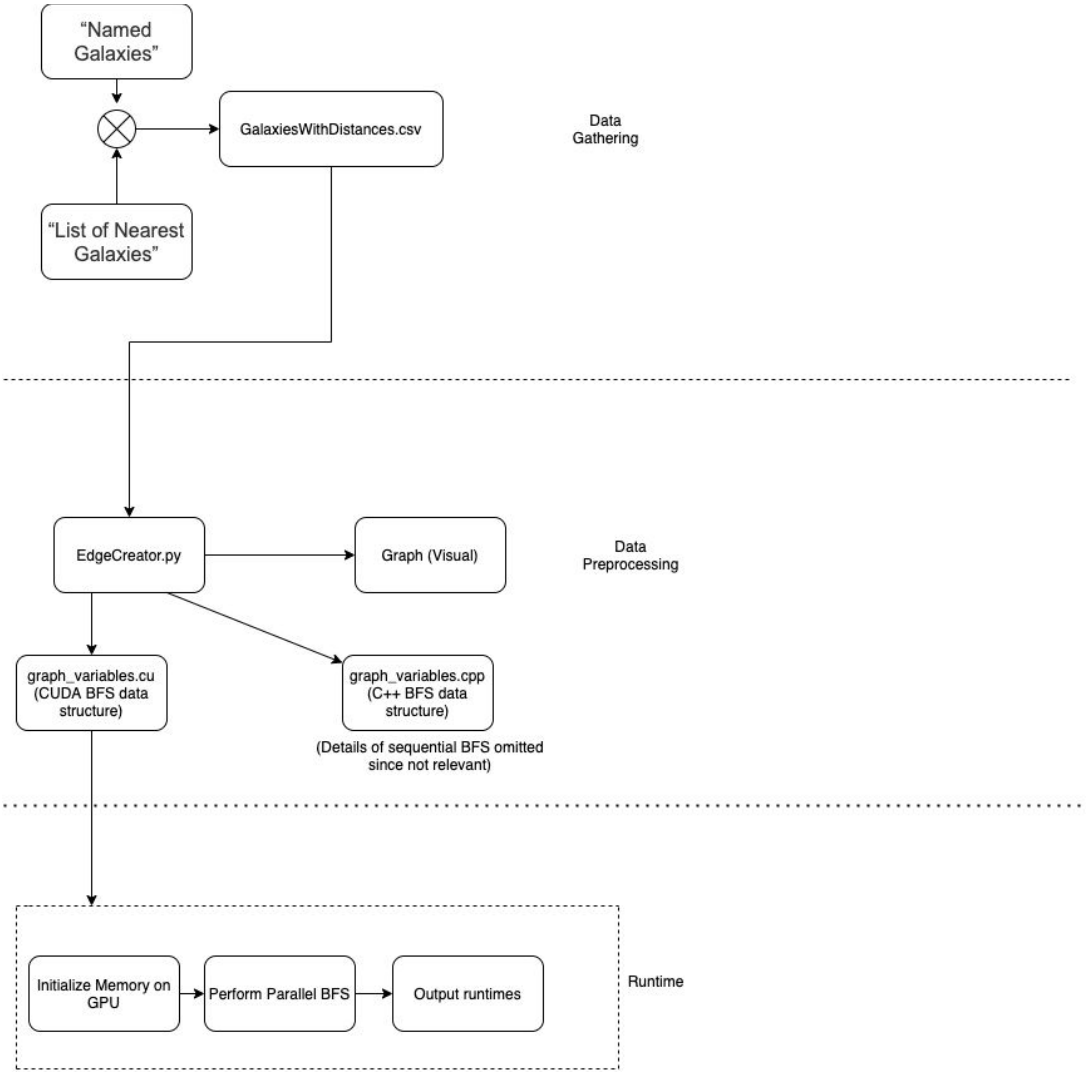

Figure B. High Level Flow Diagram

This project contains 3 parts. The first is gathering the data from the sources provided in the references [1] and [8]. The next is a Python3 script used for preprocessing of the GalaxiesWIthDistances.csv. This is where all calculations are done to create a graph with all galaxies that are within the specified distance. The preprocessing script then formats the graph data and generates both a CUDA and C++ file that contain 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++.

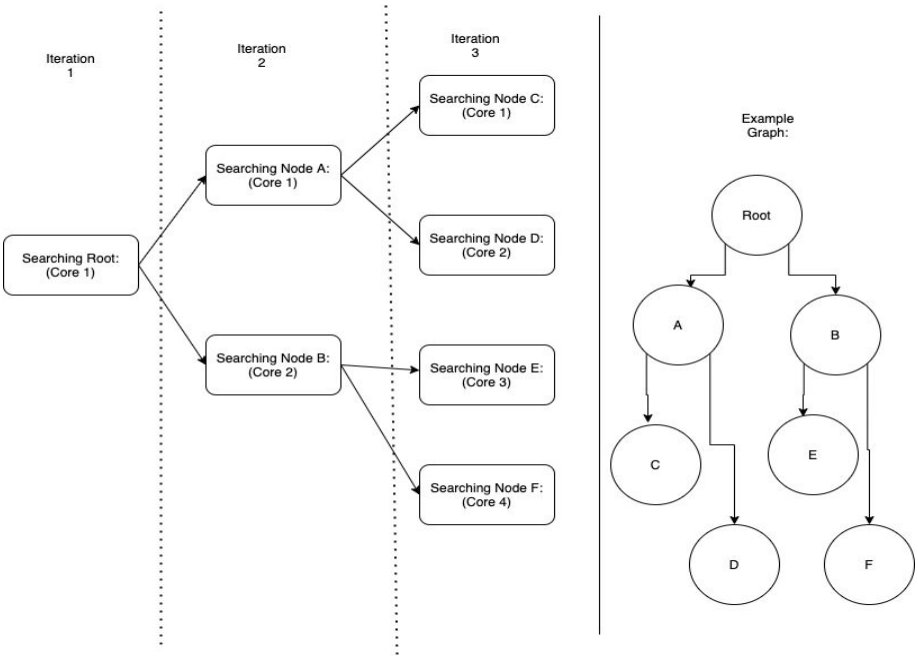

Figure C. Example execution of parallel BFS algorithm

In *Figure C*, an example of the parallelism the algorithms 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 very large performance improvements. However, for very linear graphs with few branches, it would be expected to run similarly to a sequential algorithm due to the fact that computations won't be able to be done in parallel if there aren't enough edges. A logarithmic improvement was expected assuming that each node has many edges.

```
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
```

Figure D. Text Diagram for the parallel BFS algorithm

Figure D 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.

Results

To evaluate parallel performance of the algorithm a comparison was made between a parallel and a sequential BFS algorithm. The run times of the 2 algorithms were compared using different starting points from the same galaxy dataset.

(parallel_runtime - sequential_runtime) / sequential_runtime * 100%

The equation above was used to calculate the performance increases of the parallel algorithm over the sequential algorithm.

Start Node	Parallel BFS Average (ms)	Sequential BFS Average (ms)	Performance increase
Andromeda galaxy (00)	**2.029**	**8.612**	**324%**
Draco dwarf (DDO 208) (10)	**1.572**	**6.230**	**296%**
Leo A / Leo III / DDO 069 (20)	**1.599**	**6.750**	**322%**
Pisces Dwarf (LGS 3) (25)	**1.798**	**7.286**	**305%**

Table A. Table Comparison of Parallel and Sequential BFS Performance

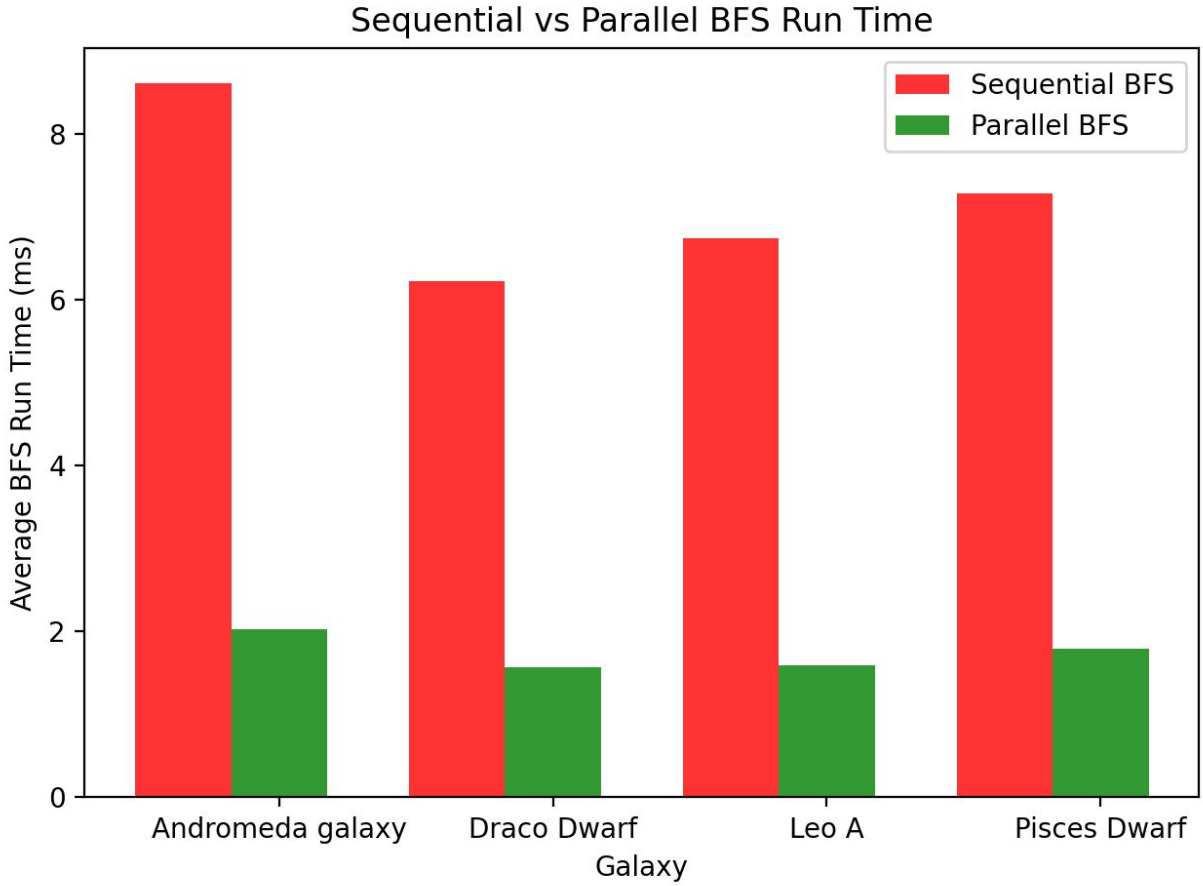

Figure E. Graph of Parallel and Sequential BFS Runtimes

Figure E and F show the performance increase of the parallel over the sequential BFS algorithms. For the galaxy dataset used in this report, there was around a 300% 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.

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 runtime 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 didn't need to be optimized.

Division of Work and Self Assessment

Below lists each individuals self-assessment describing their contributions:

Tyler Hilbert

My main responsibility was generating a dataset of galaxies that could be traveled between in a graph data structure. This needed to be in a format that was understandable by CUDA.

To do this I had to research basic astronomy measurements to figure out what data was needed to compute which galaxies (nodes) could be travelled between (edges) given a limited (and arbitrary) traveling distance. Since such a graph has not been created before, I had to figure out and find the required data. I discovered the most basic way to compute possible edges between galaxies in our hypothetical situation would be to utilize the following 3 variables for each galaxy: RA, DEC and light years from Earth. Since there was no clean dataset containing these 3 data points, I stitched together 2 different datasets to create the GalaxiesWithDistances.csv dataset.

Once I had the dataset created, I wrote a Python script that would calculate which galaxies could be traveled between. To do this I utilized an equation that would convert each galaxy into a point on a 3D cartesian coordinate system with Earth at the center. This equation could be combined with the distance formula to calculate the distance between 2 galaxies. If the distances between the galaxies was less than the distance the hypothetical space ship could travel, an edge was created between the galaxies. Once I had figured out the required math, I implemented it in a Python script to compute the graph of galaxies.

The final part of my contributions involved getting the data from Python into CUDA. To do this I looked at the different formats data can be stored for a parallel BFS. Once I had identified the correct format for the parallel algorithm, I updated my Python script to

generate a CUDA/C++ file that had each of the graph variables already instantiated in the correct format. This prevented the need to read in a dataset in C/C++ and avoided many possible input errors.

Tylor Beatty

The team worked together, researching various implementations of Breadth First Search traversal of graph algorithms. When choosing a design implementation, preference was put towards a simple, yet highly parallel design. The group examined various sources of code implementations and research papers in order to make the decision.

After the process of choosing a good, simple design for our cuda Breadth First Search traversal of graph algorithm was complete, I was tasked with writing the code implementation of the algorithm. I created a basic pseudocode mockup and retrieved approval from the group that it was satisfactory to the agreed upon implementation. Finally, I wrote the code for both sequential (CPU) and parallel (GPU) versions of the algorithm, with the occasional input or assistance from the rest of the team. Finally, I tested the algorithm code with inputs generated by Tyler's python code.

Matt Nemeth

Our team met multiple times defining the requirements and objectives for our final project. Each team member took part in planning out what would be completed as well as giving feedback on different options our team could potentially implement. I acted as a float on this project, assisting both other team members on their respective portion of the project.

My main responsibility of this project was to understand the work and step complexity of our parallel breadth first search algorithm. Using the video series on Youtube created by Udacity, recommended to us at the start of the semester, I was able to get a better understanding of where the complexity values come from.

I was also responsible for maintaining and keeping track of resources necessary to put in our bibliography. All resources were reviewed by each team member since each member assisted in all parts of the project. Our team used a variety of resources including research articles, Youtube Videos, Websites and more.

References

1. Barry F. Madore (BFM), 2020 May 6th, “Named Galaxies”

   https://ned.ipac.caltech.edu/level5/CATALOGS/naga.html

2. Rafalk342, 2018 Jan. 24th, “bfsCUDA.cu”

   https://github.com/rafalk342/bfs-cuda/blob/master/bfsCUDA.cu

3. Pawan Harish, 2007, Accelerating Large Graph ALgorithms on the GPU Using CUDA

   https://www.nvidia.co.uk/content/cudazone/CUDABrowser/downloads/Accelerate_Large_Graph_Algorithms/HiPC.pdf

4. Sanders, Jason, et al. *CUDA by Example: an Introduction to General-Purpose GPU Programming*. Addison-Wesley/Pearson Education, 2015.
5. Sid Srinivas, 2019, “Implementing Breadth First Search in CUDA”

   https://siddharths2710.wordpress.com/2017/05/16/implementing-breadth-first-search-in-cuda/

6. Udacity, 2015 Feb. 23rd, “BFS Try #1 - Intro to Parallel Programming”

   https://www.youtube.com/watch?v=SktpTf2MoIw

7. Udacity, 2015 Feb. 23rd, “Merrills Linear-Complexity BFS on GPUs Part1 - Intro to Parallel Programming”

   https://www.youtube.com/watch?v=Dq0ImVxQsRo&list=PLGvfHSgImk4aweyWlhBXNF6XISY3um82_&index=351

8. Wikipedia, 2007 Oct 7th, “List of Nearest Galaxies”

   https://en.wikipedia.org/wiki/List_of_nearest_galaxies

Code Appendix:

https://github.com/Tyler-Hilbert/SpaceTripPlanner