Running From The Night: Calculating The Lunar Magellan Route in Parallel

Authors: Kevin Fang and Nikolai Stefanov

1 URL

thesnakefang.github.io

2 Summary

We will create a parallel application using MPI in C++ which uses lunar surface imagery from the Lunar Reconnaissance Orbiter (LRO) to calculate the shortest safe path between two locations on the moon on-the-fly. This program will take into account factors such as weather, max terrain change, and the current daylight position on the lunar surface.

3 Background

The Magellan Route is a proposed route of constant daylight on the lunar surface and has huge implications for lunar infrastructure. If traversing in a straight line at about 5 degrees from the pole, then you could theoretically complete this route by traveling under a mile per hour. The application will read large amounts of data from multiple images from LRO. LRO provides us with images of anywhere from 40m to 120m resolution per pixel, so finding a coarse path from those data sets might be useful.

To realistically traverse this route, we need to consider different constraints. Firstly, we should not travel in darkness. Secondly, we should limit how much our surface changes. Thirdly, we need to be able to be fast enough in case we run into faults. We lastly have to consider that any route we choose will mean that we are changing the terrain (in terms of shadowed regions) later on. Therefore, any parallelism we choose is going to involve synchronization that we get to fine-tune.

We feel parallelism is important for this application because such a route-finding algorithm could be used during future lunar missions should it be fast enough. While ideally the route would be planned before landing on the lunar surface, nothing ever quite works out that well. Potential issues with the rover could change course or even the low resolution of the images can fail to make out hazards that the rover can't traverse, therefore another path will be required to be computed quickly. This is especially the case in a mission where the

sun can't set – imagine a lunar version of Speed. The need then would be to rapidly find new paths which balance risk-taking with max speed.

4 Challenges

Creating long-distance routes based on LRO data has several parallelization challenges. The most basic is the large amount of data we are working with. For path computation between different chunks of the moon, determining how the processor will choose what data to load in is important – when we reach a corner of the current map file, do we load in all three maps adjacent to the corner? Managing the map files and pre-calculating or other management of data – e.g. we could have different granularities of memory access (SegTree) which we manage and update across different processors. Given that paths close to each other have strong locality, one possibility is to have communication across processors calculating areas near each other or have each processor take completely different directions based on some initial estimation (modifications on the A* algorithm).

Another difficulty is calculating hazards and terrain in parallel – we will want to use parallelism to determine what patches/elements of the map are non-suitable for traversal, but calculating every single rock or crater $>$ 1m in diameter on the surface of the moon would be far too inefficient. Likely our system will be calculating hazard difficulty in parallel to determining likely future paths and identifying future possible hazards based on that path, then communicating that back to the processors that are calculating hazards. This improves locality (we'd be accessing and calculating data in the close surroundings of a rover which moves relatively slowly on a contiguous path).

A final difficulty is the changing parameters – e.g. what time of the lunar day the rover starts of in, the max elevation change we've determined is acceptable, or how big of a rock/crater is an actual issue versus something that might be safe to cross. All of these will affect the level of communication to computation, potential for divergent execution, and expected runtime. Given our program will be able to calculate new routes on-the-fly based on our risk parameters, there is a possibility we will have to reconsider rapidly route and risk-tolerance depending on unforeseen circumstances that delay the rover.

5 Resources

We will use the LRO image data set found here: `https://ode.rsl.wustl.edu/moon/productsearch`. We will be starting from wire annealing a bit, though the .img format and image specific implementations may require heavier rework of the code for Assignment 4 than we are expecting. The biggest thing we might need is a way to store a lot of the data,

but we should be fine on just the GHC machines and ideally PSC machines to get more data points. We may also reference Nate Otten's PhD thesis for background information: `https://www.ri.cmu.edu/app/uploads/2018/01/thesis_otten.pdf`

6 Goals and Deliverables

Plan to Achieve:

- Develop a Parallel Pathfinding Algo
  - Implement basic pathfinding algorithm (e.g., A*) in a parallel environment for the lunar Magellan route.
  - Utilize sparse matrix calculations for determining slope constraints in a map
  - Integrate lunar terrain data (elevation, obstacles) into the algorithm.
  - Optimize parallel algorithm for load balancing and data locality.
  - Goal: Achieve a speedup of at least 6x (versus sequential version) with 8 or more processors. Seems reasonable as most of the time taken by the program should be parallelizable.
- Achieve $<$ 1h Comp Time
  - Profile and identify bottlenecks in the algorithm. Figure out how to avoid long wait times when sending and receiving signals or memory accesses.
  - Implement optimizations based on profiling results (e.g., reduce communication overhead, enhance data caching).
  - Goal: Complete route calculation for the entire lunar surface between two points within 60 minutes on an 8-core system.
- Create and Present Results
  - Collect and analyze timing data for computation, communication, and synchronization. Create graphs and visualizations as needed.
  - Compare performance with different numbers of processors and various parallelization strategies. Calculate effectiveness as environmental factors change.
  - Goal: Create a poster with charts, collected data, and tables which is capable of convincingly demonstrating our work and results.

Hope to Achieve:

- Demo Creation
  - Develop a GUI to visualize the lunar surface and the calculated route.
  - Implement interactive features to allow users to select start and end points, and adjust constraints (e.g., max terrain change).
  - Goal: Have something that classmates at the poster demonstration can interact with and set their own waypoints and visually see the calculated route.
- Dynamic & Accurate Shadow
  - Develop a model to simulate the movement of shadows on the lunar surface over time. Bonus points if its synched to a time input for what the actual shadows on the moon would look like then.
  - Integrate the shadow model into the route calculation, allowing for dynamic updates of the path.
  - Goal: Demonstrate the ability to recalculate a previously-calculated route within 10 minutes in response to significant changes in shadow patterns (time change).

7 Platform Choice

We want to use MPI in C++ for our project because it provides fine-grained control over the parallel processes and data communication, which is crucial for explicitly managing what information about the current environment situation and calculated path data is being passed between different processors. MPI's flexibility in managing inter-process communication will allow us to efficiently distribute the workload and synchronize the data across different stages of the route calculation, and it's a tool we both have experience with and find interesting.

8 Schedule

Week	Milestone
Week 1	Convert LRO .img files into usable data formats and adapt the wire annealing code for initial path calculations.
Week 2	Modify the annealing code to incorporate permanently shadowed regions as constraints in the path calculation.
Week 3	Incorporate (sequentially) other images to apply further constraints, finishing parallel across paths and across image sections.
Week 4	Complete the parallelization across different images, ensuring that the algorithm can handle multiple constraints simultaneously.
Week 5	Implement parallelism across time, Design an encroaching shadow representing the night the rover is ”running away from” and changing environmental conditions.
Week 6	Finalize all experiments, analyze timing results, and prepare a demonstration for the poster session. Create all required charts, diagrams, tables.