Parallel pathfinding for Amazon's Prime Air UAVs

Stephan Boettcher

Department of Computational Science and Engineering, Georgia Institute of Technology

Objectives

The objective of this project was to:

- Establish baseline performance of the A* pathfinding algorithm
- Determine how to generate a large number of optimal paths as fast as possible

Introduction

Amazon Prime Air[1] was announced December 2013 as a way to deliver packages within 30 minutes of ordering. This will be achieved through the use of small unmanned aerial vehicles (UAV). The Amazon Prime Air project places restrictions on which customers are able to utilize this service, usually based on a geographic radius from a supply warehouse. These UAVs currently have small payload capacities and limited range due to battery power density restraints. To achieve peak efficiency, these UAVs will need to fly from the warehouse to their delivery location following an optimal path. In a dense urban environment, such as Los Angeles as seen in Figure , airspace restrictions are fluid and subject to change over the course of a day. A pathfinding algorithm must be able to handle these changes while still being able to generate a large number of routes quickly.

Figure 1: FAA airspace surrounding Los Angeles

Methods

A single path can be generated quickly using an algorithm similar to the Parallel Ripple Search or the Fringe Search[2][3]. Unfortunately these two methods utilize all of the system computing resources. Conversely, it is possible to spawn off one path per thread and batch process the paths using a slower serial algorithm. This method results in all threads being dedicated to batch processing. Peak efficiency is achieved with a blend of these two paradigms.

The following steps were taken for analysis:

1. Generate 6 maps for benchmarking suite
2. Batch process A* algorithm
3. Parallelize and batch process A* algorithm
4. Used MPI to perform distributed parallel batch processing on A* algorithm
5. Parallelized Fringe Search method
6. Determined optimality lost by Fringe Search method

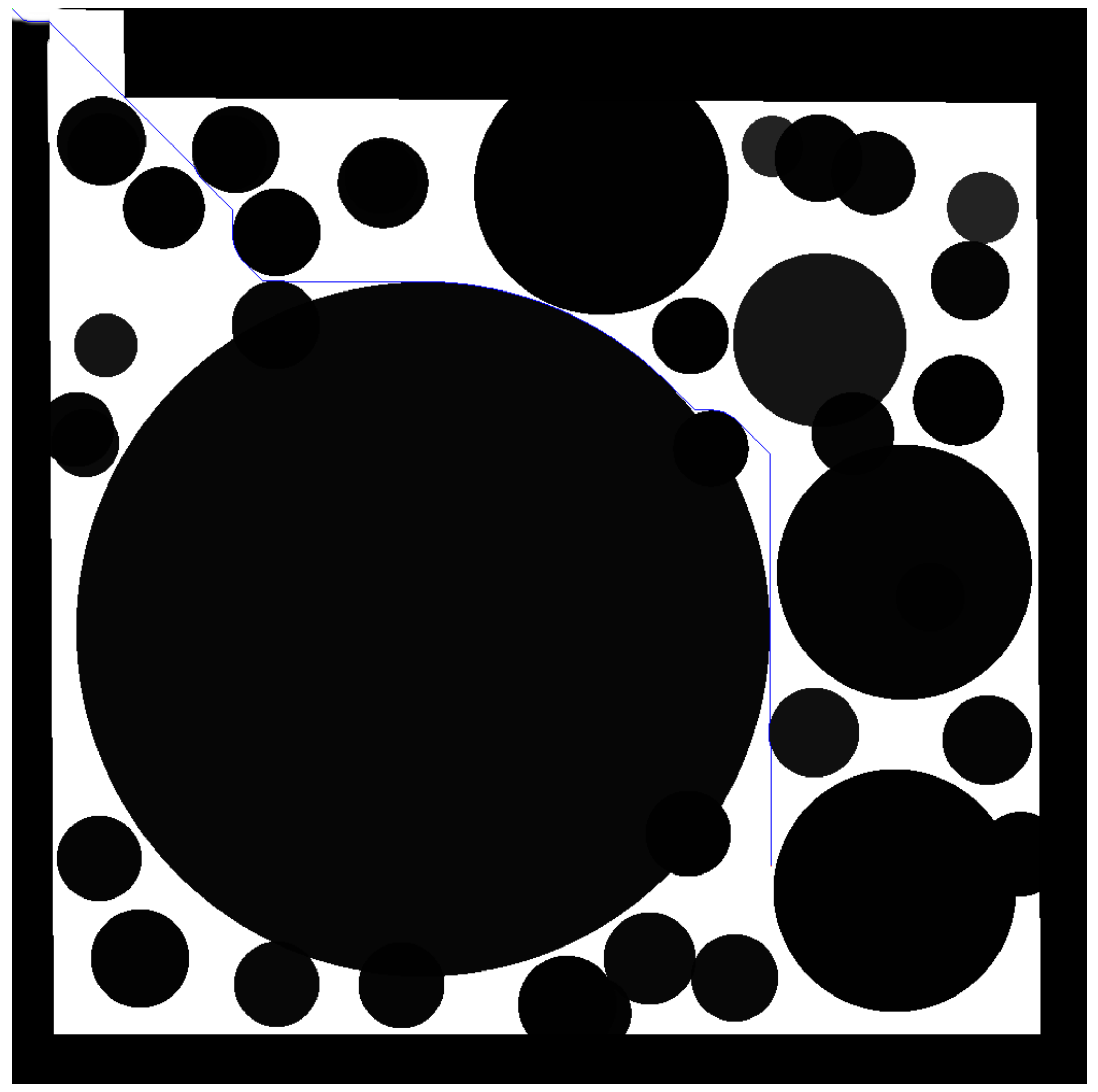

Figure 2: Sample path output from A* algorithm

Results

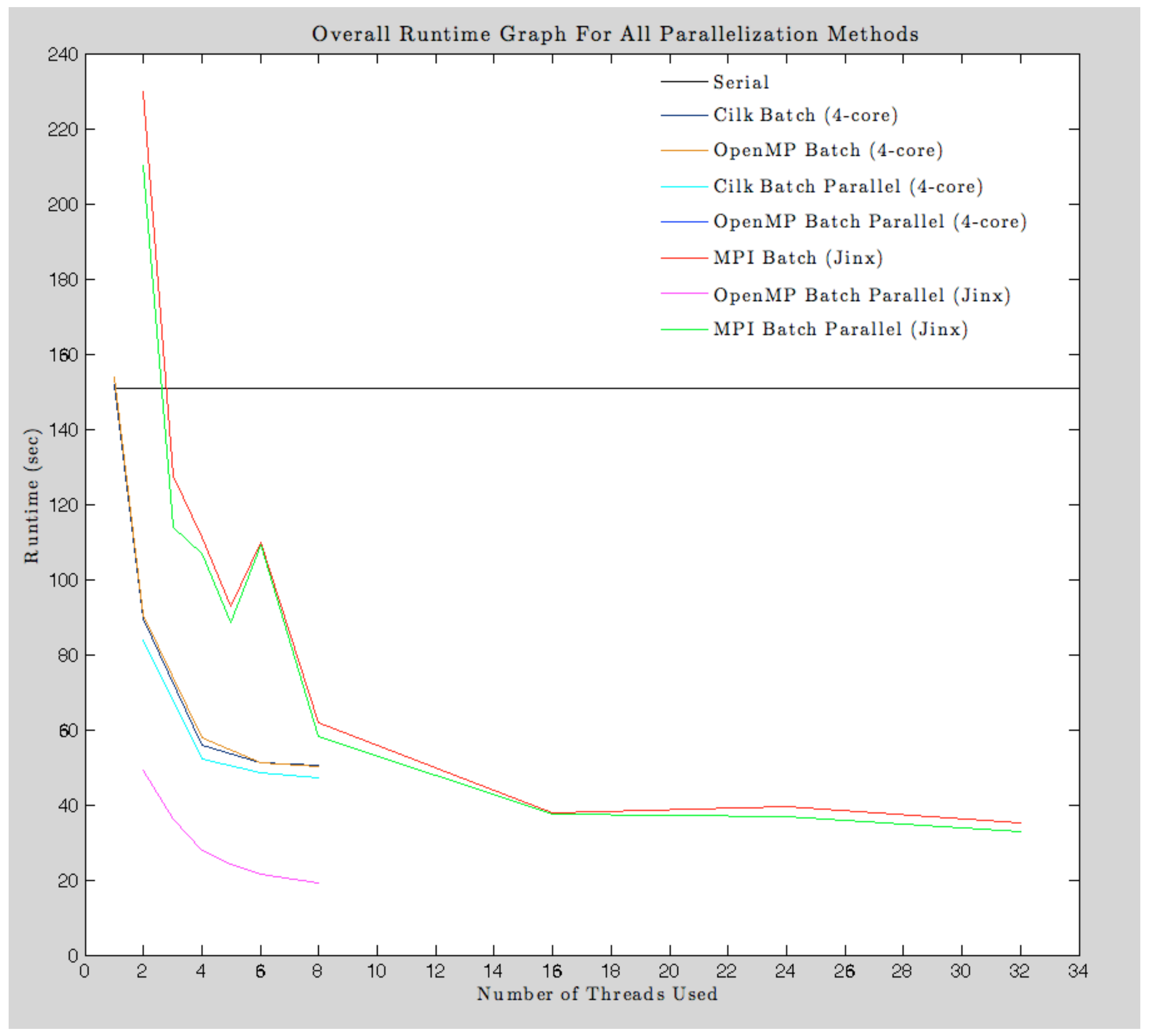

Figure 3: Final results of parallelization methods

The overall results were:

- The OpenMP code running in batch parallel was able to achieve a 8x speedup on a 6 core Jinx cluster
- MPI configurations were slower than expected due to the large number of small messages required for scenario setup and result collection
- Cilk and OpenMP achieved equivalent speedups
- OpenMP is able to integrate with other parallelization methods such as MPI, unlike Cilk
- Paths created by the Fringe Search algorithm were up to 10% longer than A* algorithms. This corresponds to longer flight times for the UAVs

OpenMP A* algorithm in a batch parallel configuration was the fastest method to generate 1000 paths.

Conclusion

A linear speedup of the runtimes was achieved under restrained conditions and a large number of complex paths were able to be calculated in a short timeframe. While it is somewhat difficult to drastically parallelize this code while still maintaining optimality, sufficient parallelization options were available to achieve large speedups. The effects of MPI message size, quantity and type were investigated, as were the effects of combining multiple parallelization paradigms. Ultimately, the economic limits and physical constraints surrounding the use of UAVs as a package delivery system dictated the largest constraints on the project. Had the path generation runtimes been longer than the added flight time of a UAV on a suboptimal path, other path-finding algorithms could have been utilized and parallelized. Thus, a parallelized A* algorithm will be the ideal method for Amazon Prime Air to generate the most efficient UAV routes.

References

[1] "Amazon Prime Air". Retrieved December , 2014

[2] S. Brand, and R. Bidarra, Multi-core scalable and efficient pathfinding with Parallel Ripple Search. Comp. Anim. Virtual Worlds, 23: 73Ð85. doi: 10.1002/cav.1427, 2012

[3] A. Botea, "Near optimal hierarchical path-finding", Journal of Game Development , vol 1 , no , p.7 - 28, 2004.

[4] D. Cohen and M. Dallas, "Implementation of parallel path finding in a shared memory architecture", Dept. CS, Rensselaer Polytech Inst, Troy, NY, 2010.

Contact Information

- Email: sboettcher3@gatech.edu

Georgia Tech College of Computing