Parallel A* Search Proposal

Lucas Wu, Katrina Hu

URL (Project Website)

https://katrina0406.github.io/parallel_astar_search/

Summary

We propose to parallelize A* search in this project. There have been various algorithms that parallelize A* and we choose to implement one or both of HDA* and GA*, depending on time availability. We will implement HDA* on the MPI platform and GA* on the CUDA platform.

Background

A* algorithm is the backbone of many artificial intelligence algorithms and has wide applications, such as robot planning and protein design. A* maintains a priority queue of nodes that is sorted by the cost of nodes plus a heuristics estimating the cost to the target node. At each iteration, A* expands the node at the top of the priority queue. Given a consistent heuristic, A* is guaranteed to find the optimal path to the target node.

We aim to speed up the algorithm by utilizing the parallel power of modern CPU or GPU.

Hash Distributed A* [1] statically divides the nodes to threads by a hashing function. Each thread maintains its own priority queues. When a node is generated, it's hashed to its owner thread and sent with non-blocking communication.

GA* [2] is a massively parallel algorithm that utilizes thousands of cores on GPU. Each thread maintains its own priority queues. Through synchronization, GA* detects and removes duplicate nodes.

The challenge

- Work overhead. A* guarantees that with a consistent heuristic, each node is expanded at most once. This condition is inherently sequential and does not hold in parallel. The parallel version could expand many times more nodes than sequential A* does, and we need to mitigate the work overhead.

- Communication overhead. Both method requires significant communication between threads. We need to utilize locality between nodes to reduce communications.
- Choice of hash function. For HDA*, the hash function is used to divide the work. It’s essential to load balancing. There is also a tradeoff between communication overhead and load balancing, as a perfectly random (balance) hash function usually implies poor locality. For GA*, a hash table is required for detecting duplicate nodes. Implementing a hash table with high efficiency on GPU is non-trivial. We need to figure out how to utilize shared memory and avoid intra-block communication.

Resources

We plan to use the PSC cluster for MPI implementation and GPUs on GHC machines for CUDA implementation.

We will start from a sequential A* codebase and write everything parallel from scratch. The code base is from the Search-based planning lab at RI.

Goals and Deliverables

Evaluation

There are various benchmarks for testing A* performance. Since one of the most important applications of A* is pathfinding, we choose to run our implementations on this commonly used benchmark [3] for grid pathfinding problem. We will mainly test the speedup of our implementations over the sequential A*. We will measure and show communication and work overhead to understand how effective is parallelism. We will also run ablations on parameters and design choices in our algorithm, such as choice of hash function, etc.

Goals

- Baseline: The sequential A*.
- Expected achievement: Finish implementing one of the HDA* or GA*. Achieve a comparable speedup with the original paper.
- Extra goal: Implement the other one.

Deliverables

- Speedup graphs compared to sequential version of A* search
- Plots on the communication overhead and work overhead
- Potentially some ablations.
- Analysis report for how the algorithm is parallelized and what can be improved

Platform Choice

Choice 1: MPI（HDA*)

- Given that data consistency is essential in A* search, using a message passing model is more efficient and easier to guarantee correctness.
- MPI not only provides easy way for processes to communicate with each other but is also highly scalable, which is suitable for searching on large graphs.

Choice 2: CUDA (GA*)

- Given that A* search leverages integer heuristics functions and performs massive access to global memory, it can benefit a lot of from acceleration provided by a GPU.

Schedule

Nov 13 - Nov 19

- Finialize project idea and plan
- Organize A* search resource and set up sequential version for baseline performance

Nov 20 - Nov 26

- Implement advanced version of parallel A* search using one of the two choices above
- Analyze speedup and possible improvement

Nov 27 - Dec 3

- Implement advanced version of parallel A* search using one of the two choices above
- Finish Project Milestone Report

Dec 4 - Dec 10

- Finish implementing advanced version of parallel A* search using one of the two choices above
- Fun experiments and ablations
- Write Final Project Report

Dec 11 - Dec 14

- Wrap up Final Project Report
- If still have time, implement other versions of parallel A* search / propose other solutions

Reference

[1] Kishimoto, A., Fukunaga, A., & Botea, A. (2009). Scalable, Parallel Best-First Search for Optimal Sequential Planning. Proceedings of the International Conference on Automated Planning and Scheduling, 19(1), 201-208. https://doi.org/10.1609/icaps.v19i1.13350

[2] Yichao Zhou and Jianyang Zeng. 2015. Massively parallel a* search on a GPU. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI'15). AAAI Press, 1248–1254.

[3] N. R. Sturtevant, "Benchmarks for Grid-Based Pathfinding," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no. 2, pp. 144-148, June 2012, doi: 10.1109/TCIAIG.2012.2197681.