15-618 Project Proposal: Parallel Path-finding

Daksha Shrivastava, Akash Hegde
{dakshas, akashh}@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA

Project website: Parallel Path-finding

I. Summary

We are going to implement an optimized parallel version of the A* algorithm which exploits shared address space and message passing parallel programming models. We plan to use OpenMP, for intra-node shared memory and MPI, for inter-node communication. We aim to make our implementation scalable with both, problem size and number of processing elements.

II. Background

A path-finding algorithm [2], at its core, searches a graph by starting at one vertex and exploring adjacent nodes until the destination node is reached, generally with the intent of finding the shortest route measured by some metric. This field of research is based heavily on Dijkstra's algorithm for finding a shortest path on a weighted graph. Dijkstra's Algorithm lets us prioritize which paths to explore. Instead of exploring all possible paths equally, it favors shorter paths, as opposed to graph traversal algorithms like Depth First Search, which explore equally in all directions.

A* is a modification of Dijkstra's Algorithm that is optimized for a single destination. Dijkstra's Algorithm can find paths to all locations; A* finds paths to one location, or the closest of several locations. It prioritizes paths that seem to be leading closer to a goal. A* achieves better performance by using heuristics to guide its search. At each iteration of its main loop, A* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. Specifically, A* selects the path that minimizes,

$$\mathbf{f}(\mathrm{n}) = \mathbf{g}(\mathrm{n}) + \mathbf{h}(\mathrm{n})$$

where `n` is the next node on the path, `g(n)` is the cost of the path from the start node to `n`, and `h(n)` is a heuristic function that estimates the cost of the cheapest path from `n` to the goal. A* terminates when the path it chooses to extend is a path from start to goal or if there are no paths eligible to be extended. The heuristic function is problem-specific. If the heuristic function never overestimates the actual cost to get to the goal, A* is guaranteed to return a least-cost path from start to goal.

A Algorithm Pseudo Code*

```
unvistedList = []
vistedList = []
put startNode on unvistedList (leave it's f at zero)
while unvistedList is not empty
    currentNode = Node with least f value
    remove currentNode from unvistedList
    add currentNode to vistedList
    if currentNode is the goal
        Declare victory! Woot! Backtrack to get path
    neighbors = [adjacent nodes of currentNode]
    for each neighbor in neighbors
        if neighbor is in vistedList
            continue to beginning of for loop
        neighbor.g = currentNode.g + distance(currentNode,neighbor)
        neighbor.h = distance from neighbor to end
        neighbor.f = neighbor.g + neighbor.h
        if neighbor.position is in unvistedList's nodes positions
            if neighbor.g is higher than the unvistedList node's g
                continue to beginning of for loop
        add the neighbor to unvistedList
```

Potential aspects of the A* algorithm that can benefit from parallelism are:

- Computation of the heuristic function: In some applications of A* search, computing the heuristic functions is quite expensive and becomes the bottleneck of the whole algorithm.
- Hierarchical division of the graph into clusters [3][7]: The graph can be split into clusters, with entry and exit points, A* can be used to find paths between nodes that connect the clusters in parallel for each cluster. These paths can then be used in conjunction to find the shortest path between source and destination.

- Main Loop: At each iteration of the main loop, A* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal, this can be computed in parallel for each possible neighbor of the node.

Path-finding algorithms are integral to applications such as movement of AI agents in games, analysis of social networks, navigation and exploration.

III. The Challenge

- **Algorithm Challenges** Path-finding is complex and often very high on the list of expensive operations in an application. The computation time required to find the best traversal path increases exponentially with the size of a graph.
  - Parallelizing path-finding can provide great benefits to applications employing it, in terms of performance. However, being irregular and deeply nested A* search algorithm imposes acceleration challenges. It has been shown that parallelizing A* is not trivial and poses significant challenges over parallelizing other path-finding algorithms like the Bellman-Ford algorithm [8]. Therefore parallelizing it presents interesting learning opportunities.
  - For scenarios with pre-computed heuristics, we expect to have low arithmetic intensity which might affect the performance.
  - There is a trade-off to be made between run-time and accuracy of the path found for parallel implementations of A* algorithm.
  - Dividing the graph into clusters [3][7], as explained in `Section II`, requires inserting specific "Start" and "End" positions in each individual cluster. These markings are needed for maintaining continuity of paths while preserving the accuracy of A* search algorithm. However, this process of marking additional positions can be a significant overhead and can bottleneck the entire algorithm.
- **Workload Challenges**: We plan to test our implementation on both grid and non-grid based graphs
  - We expect grid based workloads to have better locality compared to non-grid based graphs because of the inherent structure of grids, however locality of grid based graphs will reduce as the graph size increases and parameters of neighboring nodes overflow a single cache line.
  - We also expect poor locality from non-grid based graphs with vertices with a high degree, as neighboring nodes are more likely to have significantly far apart memory addresses.

  - We also expect higher communication to computation ratios for benchmarks where pre-computed heuristics are considered.
  - The benchmarks identified have widely different formats for representing the graphs, hence reading the datasets and initializing the required data structures will be a challenge too.

- **System Constraints**: We plan to test the performance of our project on the Latedays cluster [12]
  - Based on the specifications of the cluster, we expect performance to first drop when we scale to more than 6 processing elements, because of the NUMA memory architecture
  - And then again when we scale to more than 12 processing elements due to the communication overheads between different nodes.
  - We do not expect the performance to increase beyond 12 processing elements as message passing over ethernet connections will then be involved which will have extremely high latencies.
  - However, we can still show scalability of our implementation as far as correctness is concerned using multiple nodes of the latedays cluster.

IV. Resources

- All the code will be written from scratch in C/C++, and OpenMP and MPI libraries will used to perform parallelism.
- We plan to deploy and test our program on the latedays cluster.
- We will primarily focus on implementing strategies taught in class [1] to parallelize A* algorithm.
- The Hierarchical Breakdown A* (HBA) approach, which was identified as a potential strategy for parallelism in `Section II`, will be implemented as described in [3],[7].
- We will be gauging the performance of our work using one or more of the following benchmarks:
  - 2D Grid World Benchmarks: Simplified representations of maps used in games made by the company BioWare [9].
  - Open Flights: This database contains graphs representing over 10,000 airports, train stations and ferry terminals spanning the globe [10].
  - SNAP System: General purpose graph dataset used for high performance system for analysis. [11].

V. Goals and Deliverables

- Primary Goals
  - Implement a correct serial version of the A* algorithm
  - Implement a correct parallel version of the A* algorithm exploiting intra-node shared memory configuration of a single latedays node, using OpenMP.
  - Extend parallel version to make use of inter-node communication capabilities of the latedays cluster, using MPI.
  - Optimize/fine-tune the parallel version using the strategies described in previous sections while maintaining scalability in terms of problem size and processor nodes
  - Perform detailed performance analysis of the parallel implementation, to characterize performance and scalabilty with graph size, graph structure and system configuration.
  - Package parallelized version as an easy to use, general purpose library
- Extra Goals
  - We hope to extend the capabilities of our implementation to support multiple simultaneous start and end points [5] (multi-agent)
  - Extend implementation to support GPU hardware [4]
  - Add visualization capabilities to our implementation, to better represent the advantages of our parallel implementation over the serial one
- Final Report and Final Poster/Demo:
  - For the final poster presentation and demo, we hope to show a significant speed-up between our parallel and serial implementations of the A* algorithm for a benchmark suite
  - We also aim to characterize the speed-up obtained by the parallel implementation with respect to problem size, number of processing elements, cache size and configuration, etc. and hope to be able to attribute the trends observed to the system configuration, inherent properties of the benchmark and structure of the code
  - If we achieve our extra goals, we also hope to give a live visual demonstration of our implementation's speed-up with respect to the serial implementation

VI. Platform Choice

- Given the ubiquitous nature of path-finding algorithms and extremely large problem sizes, it is necessary that the platforms performing these computations are well equipped to handle the scale of the problem.
- Based on this, we think that using libraries like OpenMP and MPI are ideal for our project as these libraries are commonly used for compute intensive workloads running on large compute clusters and supercomputers.
- We also choose to deploy/test our project on the latedays cluster for the same reason.
- Using OpenMP and MPI will also make our project theoretically scalable to any large compute cluster or supercomputer since it does not have to be modified for different number of processing elements or different intra-platform network configurations.

VII. Schedule

Date	***Milestone***
03/28 - 04/30	Potential Topic Exploration/ Topic selection
03/31 - 04/02	Literature Survey
04/03 - 04/03	Project Proposal Checkpoint
04/04 - 04/08	Identifying Datasets, Platforms, Libraries, Resources and Benchmarks
04/09 - 04/10	Project Proposal
04/11 - 04/17	Serial Implementation and Baseline Parallel Implementation
04/18 - 04/19	Project Checkpoint I
04/20 - 04/24	Optimization and Characterization of Parallel Implementation
04/25 - 04/26	Project Checkpoint II
04/27 - 05/01	Extra Goals
05/02 - 05/06	Poster and Final Project Report

VIII. References

1. 15-418/15-618: Parallel Computer Architecture and Programming, Spring 2019
2. Amit Patel, Introduction to the A* Algorithm
3. Thornton Haag-Wolf, A* Path-Finding with MPI
4. Avi Bleiweiss, NVIDIA Corporation, GPU Accelerated Pathfinding
5. Raz Nissim, et. al, Multi-Agent A* for Parallel and Distributed Systems
6. Marc Lanctot, et. al, Path-finding for large scale multiplayer computer games
7. Vahid Rahmani, et. al, Improvements to Hierarchical Pathfinding for Navigation Meshes
8. Implementation of Parallel Path Finding in a Shared Memory Architecture
9. Path-finding Benchmarks
10. Airport Database
11. SNAP
12. Latedays Cluster (latedays.andrew.cmu.edu)