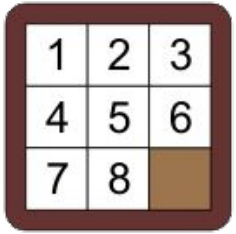
Solving N-Puzzle problem using AI Algorithms on GPU

Team Members:-
Anubhav Jaiswal
Arshdeep Singh
Kaustav Vats

Summary

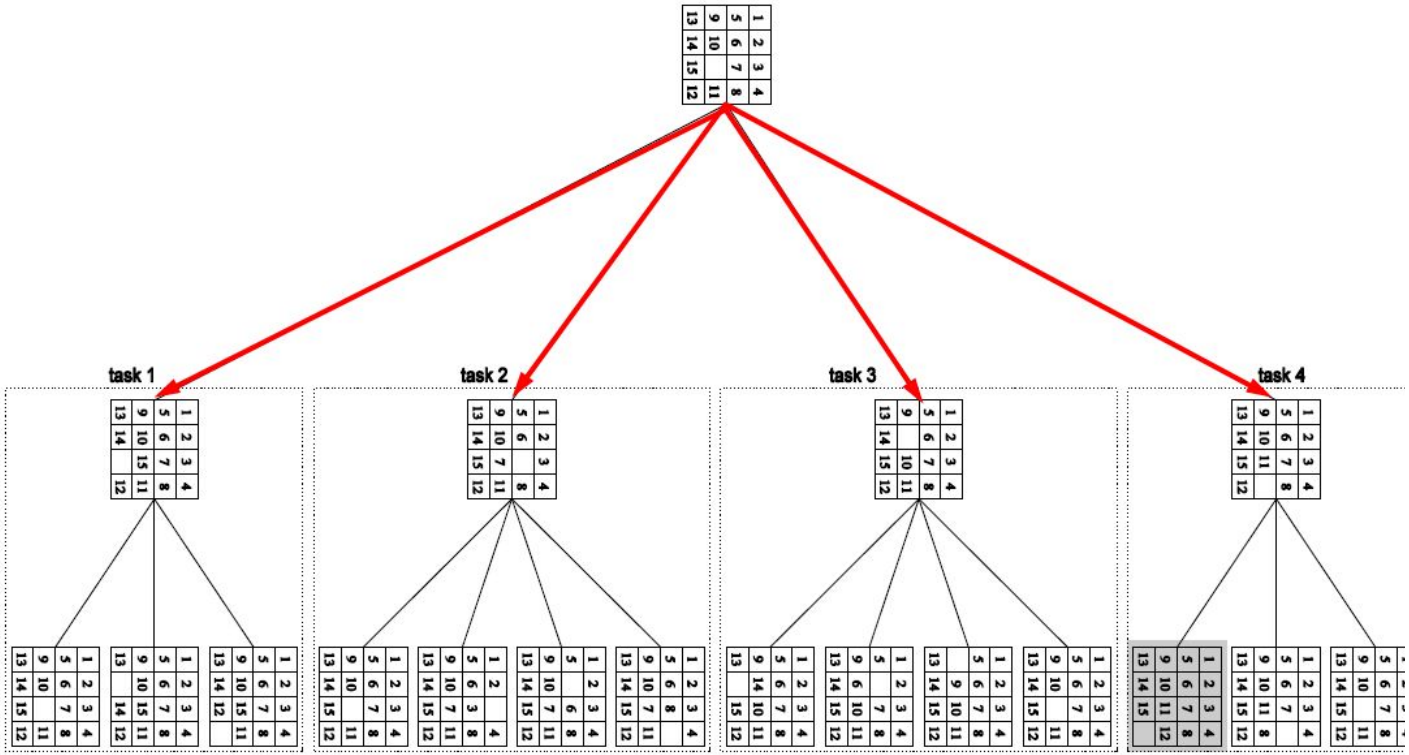


The 8-Puzzle(n=8) problem consist of 3x3 square tiles numbered from 1 to 8(n=8) with one tile missing. The objective of the problem is to arrange tile in numerical order.

Intuitively solving this problem requires the algorithm to visit all the possible states and select the final solution state. Visiting states requires recursion and backtracking which is easy to implement in a serial code(Non threading implementation), Thus implementing these graph algorithms on the GPU is a challenging task that we wish to accomplish.

There are 4 algorithms in this field that we wil research about - BFS, DFS, A* and IDA* algorithm.

Task Dependency Graph



No Task interaction present in N-Puzzle Problem

Optimisation Strategies

1. Parallelizing the helper functions used by A* algorithms.
    a. Heuristic function
    b. Priority queue
    c. Next state generation
2. We move level by level in the parallel implementation.
    a. Each thread has its own priority queue, thus analysing the nodes parallelly.
    b. Each thread also generates a maximum of 4 neighbours, which can be implemented serially as well as parallelly.
    c. Visited nodes problem had been tackled to an extent using parent creation type.
    d. All recursion have been averted as they presented a problem of variable size stack.

Deliverables

Mid-Term Goals

- Serial implementation of A*, BFS, DFS and IDA* Algorithm
- Parallel implementation of BFS Algorithm
- Literature survey of parallel algorithm of A*

End-Term Goals

- Cuda Implementation of DFS Algorithm.
- Cuda Implementation of A* and IDA* Algorithm
- Comparison of the performance all the implementations.

Deliverables

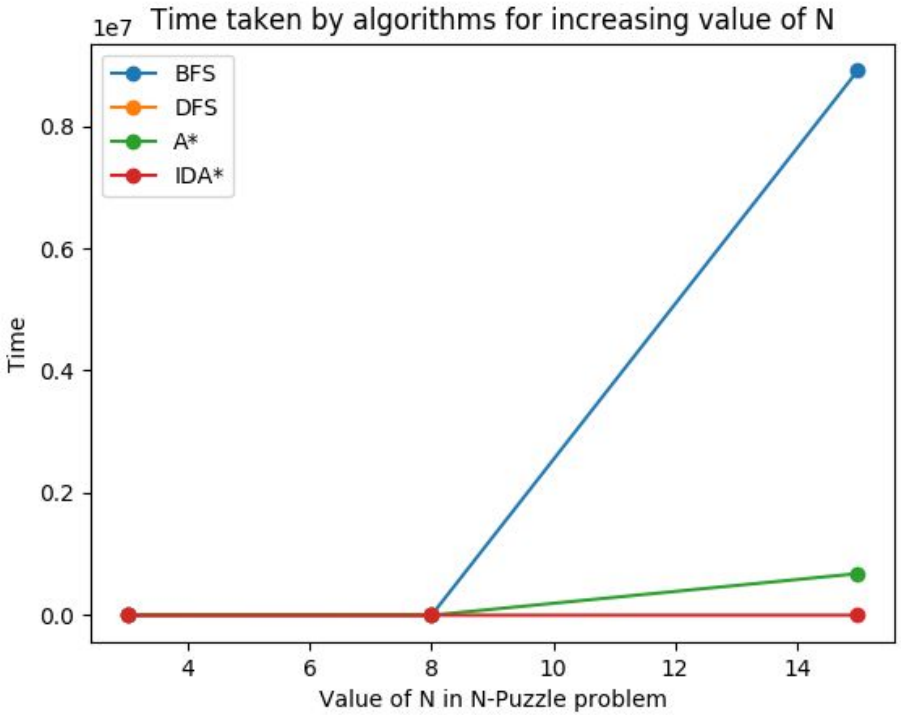
Mid-Term Goals

- Serial implementation of A*, BFS, DFS and IDA* Algorithm
- Parallel implementation of BFS Algorithm
- Literature survey of parallel algorithm of A*

End-Term Goals

- Custom Priority Queue Implementation
- Cuda Implementation of A* Algorithm
- Comparison of the performance all the implementations.
- Analysis of various algorithms

Execution time Graph for Serial Implementations

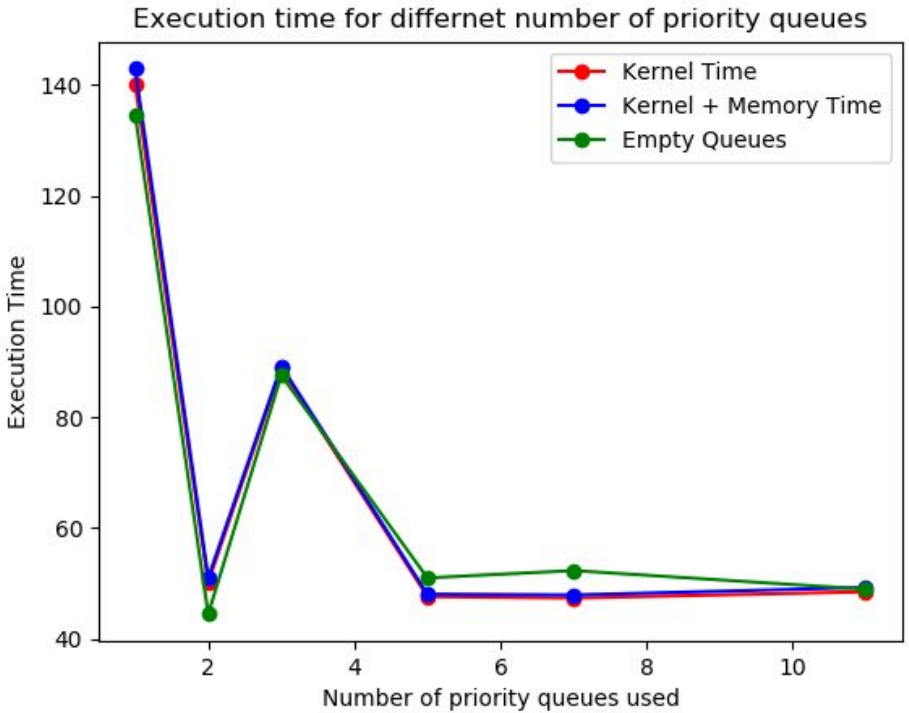


Observations:-

1. A* will always give best result as compared to BFS and DFS algorithms.
2. Depending upon the representations of the Tiles in N-Puzzle Matrix, sometimes IDA* performs much better than the A* algorithm. Similarly for BFS and DFS.
3. Generally BFS is much faster than DFS problem for N-Puzzle.

We couldn't calculate time for much larger values of N. Since it was consuming lot of time and was affecting the server state (Freezing)

Execution time Graph for Cuda Implementations of A*



Observations:-

1. K is the parameter here, which denotes number of priority queues used.
2. We observed that on increasing number of Priority Queues, time decreases for some value of K and later increases because of the overhead.

Final Remarks and challenges faced

- AI algorithms are much faster than other search algorithm
- No cuda parallel algorithm available for DFS and IDA*. We implemented the paper which uses priority queue on GPU and maintains the property of the A* algorithm
- For the BFS algorithm, one of the challenges is the creation of the adjacency list representation that is not known before, hence creating a list and then using it is similar to the serial version of the BFS.
- No Algorithm available to achieve super linear speedup for N-Puzzle problem.
- We tried different variants but were mostly performing worst than the serial implementation of A* algorithm.
- A* uses multiple helper functions, which can be parallelized to achieve the speedup. We observed that for some cases, parallelizing the part makes it slower than before.
- Memory problem for 15-Puzzle and 24-Puzzle problem for parallel A*.

Individual Contributions

1. Anubhav - Single Kernel call implementation of A* algorithm
2. Arshdeep - Multiple Kernel call implementation of A* algorithm
3. Kaustav - Get Neighbours function implemented parallely along with other functions used in A*.
    a. The whole structure of the Algorithms, including all supplementary functions