RUBIKS FUNc - A parallel Rubiks cube solver in Haskell

Yash Ashok Agarwal - ya2467
Chandrashekhar Dhulipala - cd3132

Summary:

Rubik's cube was invented by a Hungarian sculptor in 1972. The cube has 6 faces where each face is a grid of colors of size (n x n). Usually, for each face in the rubiks cube we have 3 possible moves 1) rotate by 90 degrees clockwise 2) rotate by 90 degrees anticlockwise 3) rotate by 180 degrees. Since there are 6 faces with 3 moves for each face we have a total of 6*3=18 moves.
Our aim is to create a Rubik's cube solver in Haskell.

Methodology:

Rubiks cube solvers are typically implemented using search strategies. One of the most common ways to implement a cube solver is to use an Iterative-Deepening A* search algorithm(IDA*) on the graph of various cube orientations. Using IDA* we try to find the goal state by iteratively increasing the depth parameter in DFS. In IDA* since the depth for DFs is increased at each step a lot of states are reexplored. This can be avoided by storing the intermediate results or running the IDA* parallelly for different depths. The search space for IDA* can be reduced by keeping the track of the previous move made. Our aim is to explore other pruning techniques and leverage the power of parallelism to speed up the solver.

Scope:

We see the following avenues for parallelism while implementing a Rubiks cube solver :

1. Pre-computation of heuristic function: Any A* - like search algorithm, in particular the IDA* uses a heuristic function for 'estimating' the number of moves needed to reach the goal state from a particular state. This estimate is used in the execution of the algorithm to find an optimal path. For the Rubiks cube, this heuristic function is a lookup table that can be pre-computed and stored for all exhaustive orientations of the cube. Although this is computed once and reused, the computation is very slow and can be parallelized into batches.

2. Parallelizing the search algorithm: There is a considerable amount of literature available on the ways to parallelize state-space search algorithms like IDA*. In particular, one direct way to parallelize this procedure is to delegate disjoint parts of the state-space to different cores when performing the cost-bounded Depth-First search. For example, we can divide our tree of cube orientations into separate subtrees (left, right) and run DFS parallelly on them. Another approach is to make various processors search the tree to different depths in parallel.

3. Load-balanced solving of a batch of cubes : We will be benchmarking our implementation by running it on a batch of ~1000 random cubes. Instead of solving each of them sequentially, we could also parallelize the execution of the procedure on these cubes, similar to Marlow’s technique of parallelizing the Sudoku solver.

References:

1) https://stackoverflow.com/questions/60130124/heuristic-function-for-rubiks-cube-in-a-algorithm-artificial-intelligence
2) http://kociemba.org/cube.htm
3) Rao, V. Nageshwara, Vipin Kumar, and K. Ramesh. *A parallel implementation of iterative-deepening-a*. Artificial Intelligence Laboratory, University of Texas at Austin, 1987