Project proposal: parallel minimax for two player games

team_nik
nh2677

In this project I plan to write a solving engine for two player games using a minimax algorithm. I plan to use my algorithm as a chess engine, and have the outcome of my project be a general purpose chess engine that can find a strong move from any position using minimax. My engine will not be a complete chess game, but a program that reads in a string representing a chess board, some info about the game (dead pieces, turn etc), a max search tree depth and output a string representing a single move.

Minimax game solver algorithm description

I plan to parallelize the popular minimax algorithm, which works by constructing a search tree consisting of all possible (legal) moves from a certain starting point and scoring the final outcomes at a certain depth. I will follow the traditional minimax chess approach which seeks to maximize the score for white and minimize for black (or vice versa) and compute the dfs search tree.

Advantages of minimax

The minimax algorithm is fairly straightforward to implement and is able to produce very capable chess engines.

Weaknesses of minimax

Due to the lack of greater strategic reasoning, a minimax algorithm with a too low search tree depth, might not be able to make optimal moves due to a too short lookahead. I plan to experiment with different search tree depths to see how strong the algorithm is at varying depths. Another weakness is that without pruning, the minimax algorithm can waste significant time on exploring clearly incorrect moves.

Plans to parallelize and improve performance

I will first implement a serial minimax game solver. I plan on including pruning to make the serial implementation faster. Next I will parallelize it using the Par monad. I plan to parallelize the top level of the chess engine, meaning that I parallelize across all currently playable figures and then run the algorithm in series for each piece / worker. I expect this to reduce in efficiency as the chess board empties out, but conversely as there are less pieces, the search trees become significantly smaller so I don't expect this to become a bottleneck. I will try and apply the same speedup strategies to both the serial implementation and measure the differences in performance increase.

Speedup strategy

- Optimize search tree depth to balance performance and speed
- use alpha beta pruning.

Getting data

A chess engine doesn't actually need any data. I don't plan on comparing game states, which is costly and may potentially require some IO operations. Instead, I will implement a simple board evaluation strategy that runs some precomputed evaluations on the board state. This could include things like:

- pieces captured
- balance of the board
- pieces 'covering' other pieces
- checkmate (most optimal reward)