Othello Parallelization Proposal

Siying Ding (sd3609) Carly Kiang (tk2990)

1. Introduction to Game

Othello, also known as Reversi, is a classic strategy board game played on an 8×8 grid between two players, traditionally represented by black and white discs. The game starts with four discs arranged in a square at the center of the board, two of each color. Players then take turns placing a disc of their color on the board with the goal of capturing their opponent's discs.

The position a new disk is placed must abide by two rules; we will refer to them as the adjacency and outflanking rules. For the adjacency rule, a disc can only be placed on an empty square that is adjacent—horizontally, vertically, or diagonally—to at least one of the opponent's discs. In addition, to follow the outflanking rule, the placement of the new disk must create at least one straight line in which the newly placed disc flanks one or more of the opponent's discs between itself and another disc of the player's color. A disc flanks one or more of the opponent's discs if it is placed on the board so that there is a straight line (horizontal, vertical, or diagonal) of one or more opponent discs between the new disc and another disc of the player's color. All the opponent's discs in that line are considered flanked and are flipped to the player's color.

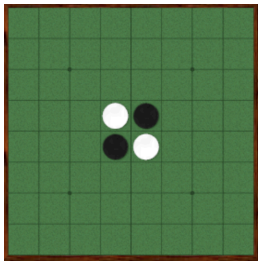

Figure 1. Othello Start Board Configuration

Figure 2. Before (on the left) and After (on the right) Player Holding White Disc Picks a Move

The strategic depth of Othello lies in balancing short-term gains with long-term positioning. Early moves often focus on controlling the center, while later stages require careful attention to corners and edges, which provide stable positions that cannot be flipped. The game ends when the board is full or no valid moves remain for either player, and the winner is the one with the majority of discs in their color. The combination of clear rules and deep strategic possibilities also makes Othello an ideal subject for

computational research, including the development and parallelization of algorithms such as the minimax algorithm.

2. Project Overview

A common decision making strategy for optimal moves in two-player games like Othello is the minimax algorithm. Given an input of the current game board, the minimax algorithm constructs a tree of potential boards after next moves, and selects an optimal move for the current player based on a heuristic score. However, since the Minimax algorithm is tree-recursive, there is an exponential number of next potential game boards that can be explored. This makes the minimax algorithm very computationally expensive. The goal of our project is to parallelize the Minimax algorithm to lead to more efficient gameplay experience.

3. Algorithm and Parallelism Approach

Parallelism of Minimax Algorithm

The minimax algorithm can be parallelized by distributing the computation of child nodes in the game tree across different threads. This parallelization can happen directly at the root, or at a hard-coded depth level of the game tree. In the context of Othello, the number of child nodes, or branches of the tree, will depend on the current state of the game board. As described above in section 1, the player moves are constrained in 8 directions, and must abide by the flanking and adjacency rules. To achieve this parallelism in Haskell, we will use the Haskell Par monad and Parallel evaluation strategies.

Alpha-Beta Pruning with Principal Variation

To help minimize the search space of the minimax algorithm, we plan to use alpha-beta pruning, which is an optimization technique that eliminates certain branches in the minimax search tree. More specifically, alpha-beta pruning removes tree branches that do not influence the final decision. Since alpha-beta pruning is primarily a sequential algorithm, it could be difficult to parallelize effectively.

Principal Variation Splitting (Stretch goal)

One possible method for parallelization of alpha-beta pruning is Principal variation splitting. The leftmost child is first run sequentially to get the initial bounds for the alpha and beta values, and then the remaining children are run in parallel. Similarly to the minimax algorithm, we plan to use rpar and rseek to implement principal variation splitting. One caveat of principal variation splitting is that its effectiveness is based on the assumption that the leftmost child is a relatively good next move. For this project, we will explore whether or not it is an effective parallelization strategy for Othello. It could be interesting to discuss possible trade-offs between time efficiency versus effective minimax results.

4. Game Setup

1. The 8x8 Othello board will be represented as a 2D list of integers, where 0 represents an empty cell, 1 represents a white disk (player 1), and 2 represents a black disk (player 2)
    a. The initial input data to the minimax algorithm will be the initial board setup, as shown in Figure 1.
2. Writing minimax algorithm will likely require helper methods
    a. possible_moves(board, player): Given an input board and current player, return all available next moves for the player

b. heuristic(board): Given an input board, calculate the heuristic score for the board. It is also possible to try out different heuristic algorithms for Othello, but these typically are not computationally expensive, and hence do not need to be parallelized

Sources

https://hackage-content.haskell.org/package/parallel-3.3.0.0/docs/Control-Parallel-Strategies.html

https://stackoverflow.com/questions/13314288/need-heuristic-function-for-reversiothello-ideas

https://www.chessprogramming.org/Principal_Variation_Search