Parallel Functional Programming Final Project Proposal

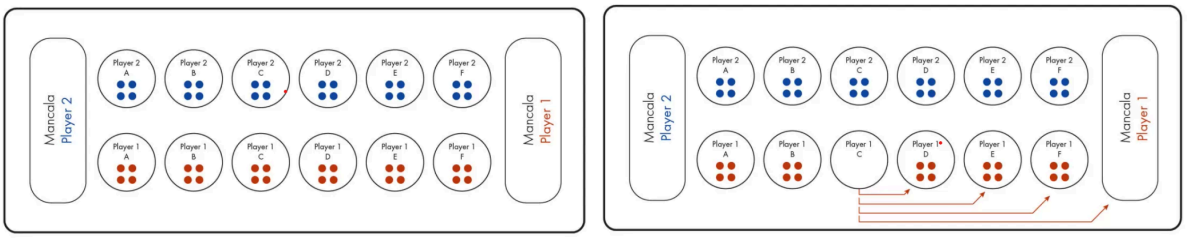
Mancala Parallelization Using Minimax

Daniel Manjarrez (dam2274), Caiwu Chen (cc4786), Sindhu Krishnamurthy (sk4699)

Nov 17, 2024

1. Introduction to Mancala

Mancala is played on a board consisting of 12 small pits and 2 larger pits called "stores" or "mancalas." Each player controls 6 small pits on their side and has one store to collect captured stones. During gameplay, players take turns picking up all the stones from one of their pits and "sowing" them into subsequent pits one by one, including their own store but skipping the opponent's store. If the last stone lands in the player's store, they get another turn; otherwise, control passes to the opponent. If the last stone lands in an empty pit on the player's side, they capture all stones from the opponent's directly opposite pit. The game ends when all pits on one player's side are empty, and any remaining stones are collected into the opponent's store. The player with the most stones in their store wins.



2. Algorithm

- *Minimax*

We will use the Minimax algorithm to make strategic decisions by evaluating potential moves and anticipating the opponent's responses. During gameplay, the algorithm recursively explores possible game states, alternating between the player and opponent, and evaluates each state using a heuristic function that considers factors such as the number of stones in the store, potential captures, and extra turns. The scores from the leaf nodes of the tree are propagated back up, with MAX nodes selecting the highest score and MIN nodes selecting the lowest. The optimal move is chosen based on the highest score at the root node. To improve performance and

reduce computation time, we will apply alpha-beta pruning to eliminate unnecessary branches from the search tree.

- *Alpha-Beta Pruning*

Alpha-beta pruning introduces two parameters: alpha, the maximum score that the maximizing player is guaranteed to achieve so far, and beta, the minimum score that the minimizing player is guaranteed to achieve so far. These parameters track the best possible scores for both the MAX and MIN players. During the search, alpha and beta are updated at each node based on the current best scores for the players. If the current node is a MAX node and its value exceeds beta, it means the minimizing player would not choose this move, so the branch is pruned. Similarly, if the current node is a MIN node and its value is less than alpha, then the maximizing player would not choose this move, so the branch is pruned.

3. Parallelization

- *Node Exploration Parallelization*

  For each move made by an AI or player, the algorithm generates a new state. We will parallelize the process of generating each node to reduce computation time. This part can be implemented using Strategies.

- *Heuristic Evaluations Parallelization*

  Since each heuristic operates independently, we will compute them in parallel at each node of the search tree to increase efficiency. This part can also be implemented using REPA.

- *Alpha-Beta Pruning Parallelization*

  We will explore methods to parallelize Alpha-beta pruning in combination with minimax parallelization. In this approach, each thread will independently maintain its branch and share information if needed.

4. Performance Evaluation

We will evaluate the performance of parallelizing heuristic evaluations in a Minimax Mancala implementation from these aspects: Speedup, CPU Usage & Thread Efficiency, Comparison with Sequential Version, etc. Ideally, we should see a significant reduction in time spent evaluating game states, with minimal increase in memory and overhead costs, leading to a more efficient decision-making process.

Reference

[1] M. Blake, *"Understanding the Rules of Mancala: Capture the Win,"* LoveToKnow, Jul. 9, 2019. [Online]. Available: https://www.lovetoknow.com/life/lifestyle/mancala-rules. Accessed: Nov. 17, 2024.