Optimizing Halma

Parallel Minimax and Alpha-Beta Pruning

Catherine Lyu and Luana Liao

Table of contents

01 Introduction

02 Project Overview

03 Algorithm

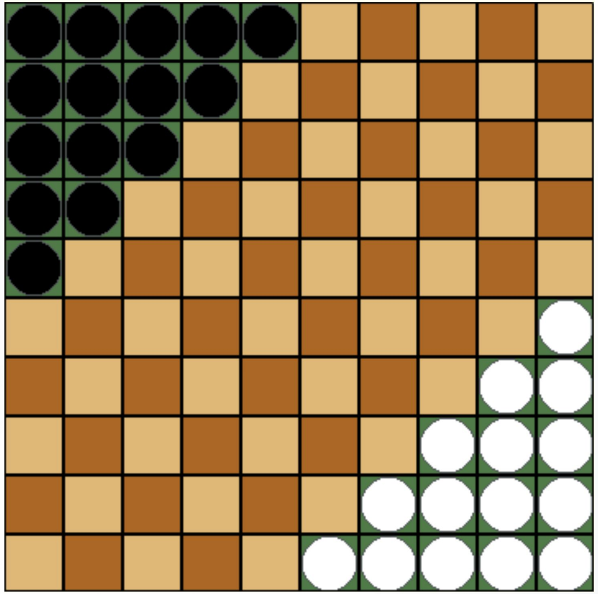
04 Performance & Evaluation

05 Next Steps

01

Introduction

What is Halma?



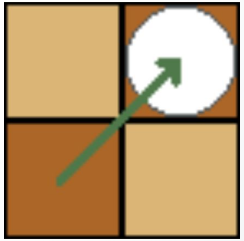
Objective: Move all pieces from your corner to the opponent’s.

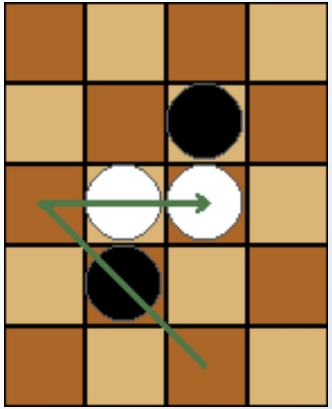
Game Mechanics

Types of Moves

- Single Move: Move one square to an adjacent empty space.
- Jump: Leap over a piece (own or opponent’s) to a blank square.

Pieces are never captured; jumping is optional.





02

Project Overview

Project Plan

MiniMax

Implement a sequential version of Minimax to serve as a baseline.

Alpha-Beta Pruning

Enhance the algorithm with alpha-beta pruning to cut off unproductive branches of the search tree.

Parallelization

Parallelization of the algorithm to distribute computational workload across multiple cores, reducing runtime.

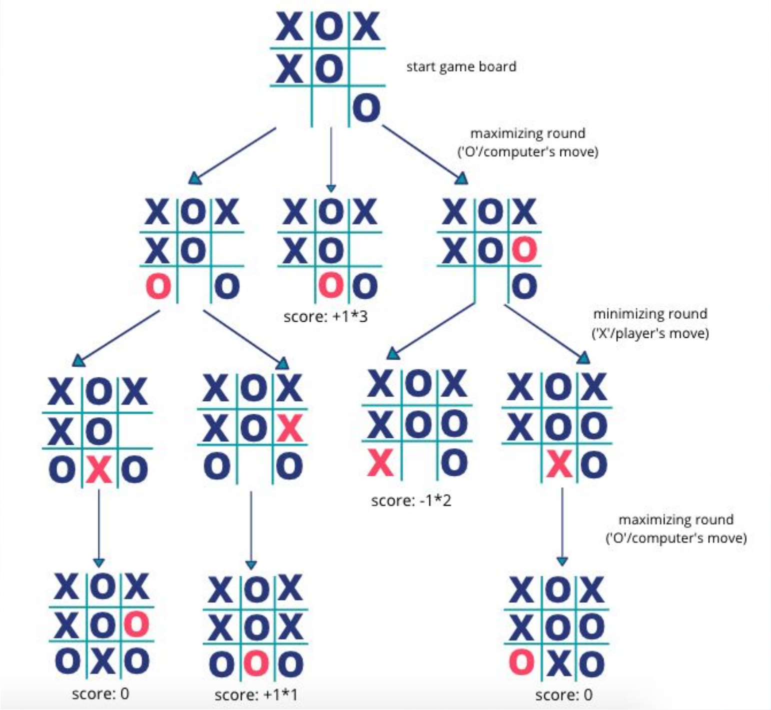
Evaluation

Evaluate the effectiveness of these approaches by measuring runtime improvements and decision quality

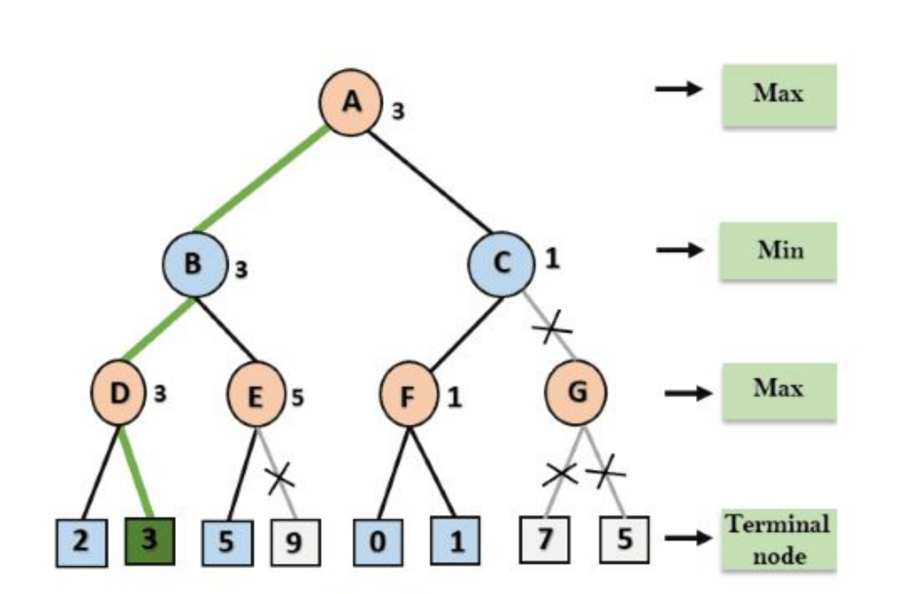
03

Algorithm

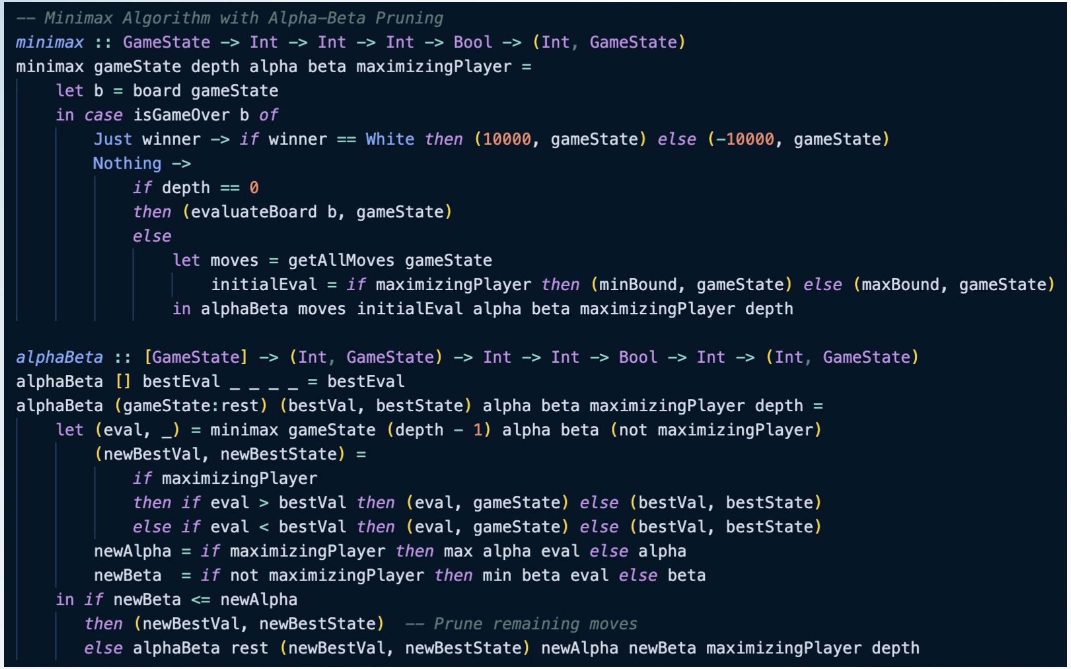
Minimax


start game board
maximizing round
('O'/computer's move)
score: +1*3
minimizing round
('X'/player's move)
score: -1*2
maximizing round
('O'/computer's move)
score: 0
score: +1*1
score: 0

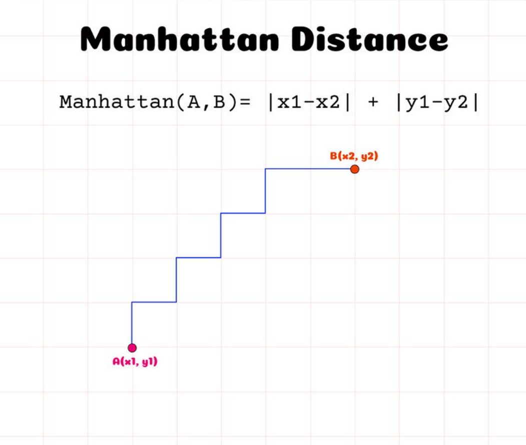
Alpha-Beta Pruning



Our Algorithm



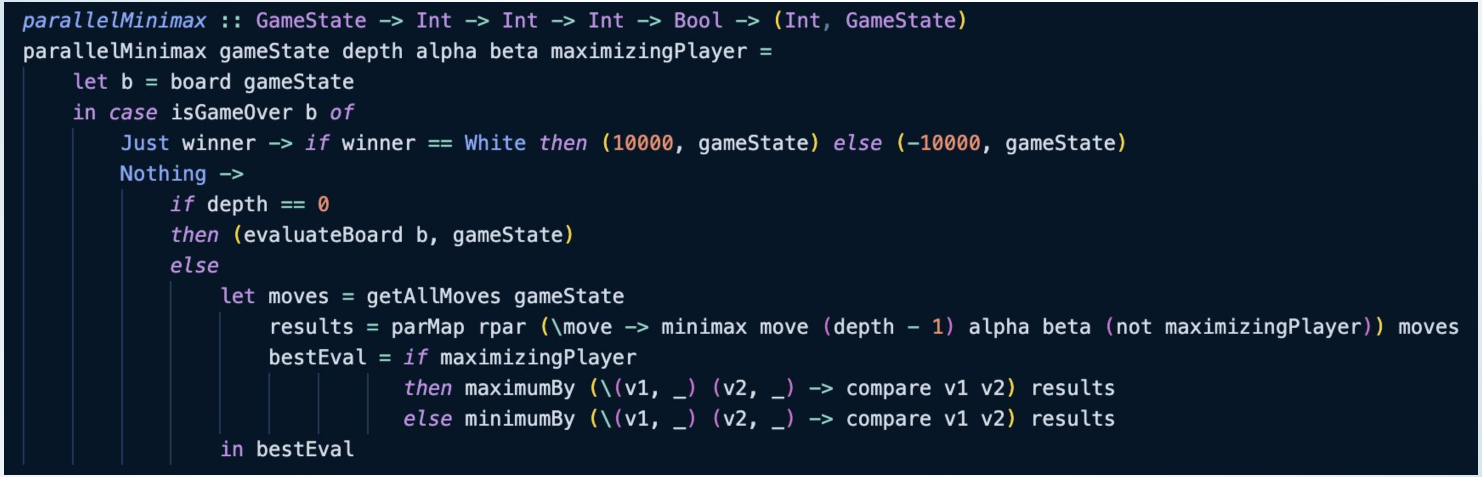
```haskell
-- Minimax Algorithm with Alpha-Beta Pruning
minimax :: GameState -> Int -> Int -> Int -> Bool -> (Int, GameState)
minimax gameState depth alpha beta maximizingPlayer =
    let b = board gameState
    in case isGameOver b of
        Just winner -> if winner == White then (10000, gameState) else (-10000, gameState)
        Nothing ->
            if depth == 0
            then (evaluateBoard b, gameState)
            else
                let moves = getAllMoves gameState
                    initialEval = if maximizingPlayer then (minBound, gameState) else (maxBound, gameState)
                in alphaBeta moves initialEval alpha beta maximizingPlayer depth

alphaBeta :: [GameState] -> (Int, GameState) -> Int -> Int -> Bool -> Int -> (Int, GameState)
alphaBeta [] bestEval _ _ _ _ = bestEval
alphaBeta (gameState:rest) (bestVal, bestState) alpha beta maximizingPlayer depth =
    let (eval, _) = minimax gameState (depth - 1) alpha beta (not maximizingPlayer)
        (newBestVal, newBestState) =
            if maximizingPlayer
            then if eval > bestVal then (eval, gameState) else (bestVal, bestState)
            else if eval < bestVal then (eval, gameState) else (bestVal, bestState)
        newAlpha = if maximizingPlayer then max alpha eval else alpha
        newBeta  = if not maximizingPlayer then min beta eval else beta
    in if newBeta <= newAlpha
       then (newBestVal, newBestState)  -- Prune remaining moves
       else alphaBeta rest (newBestVal, newBestState) newAlpha newBeta maximizingPlayer depth
```



Manhattan Distance

Manhattan(A,B)= |x1-x2| + |y1-y2|

Parallelized Minimax

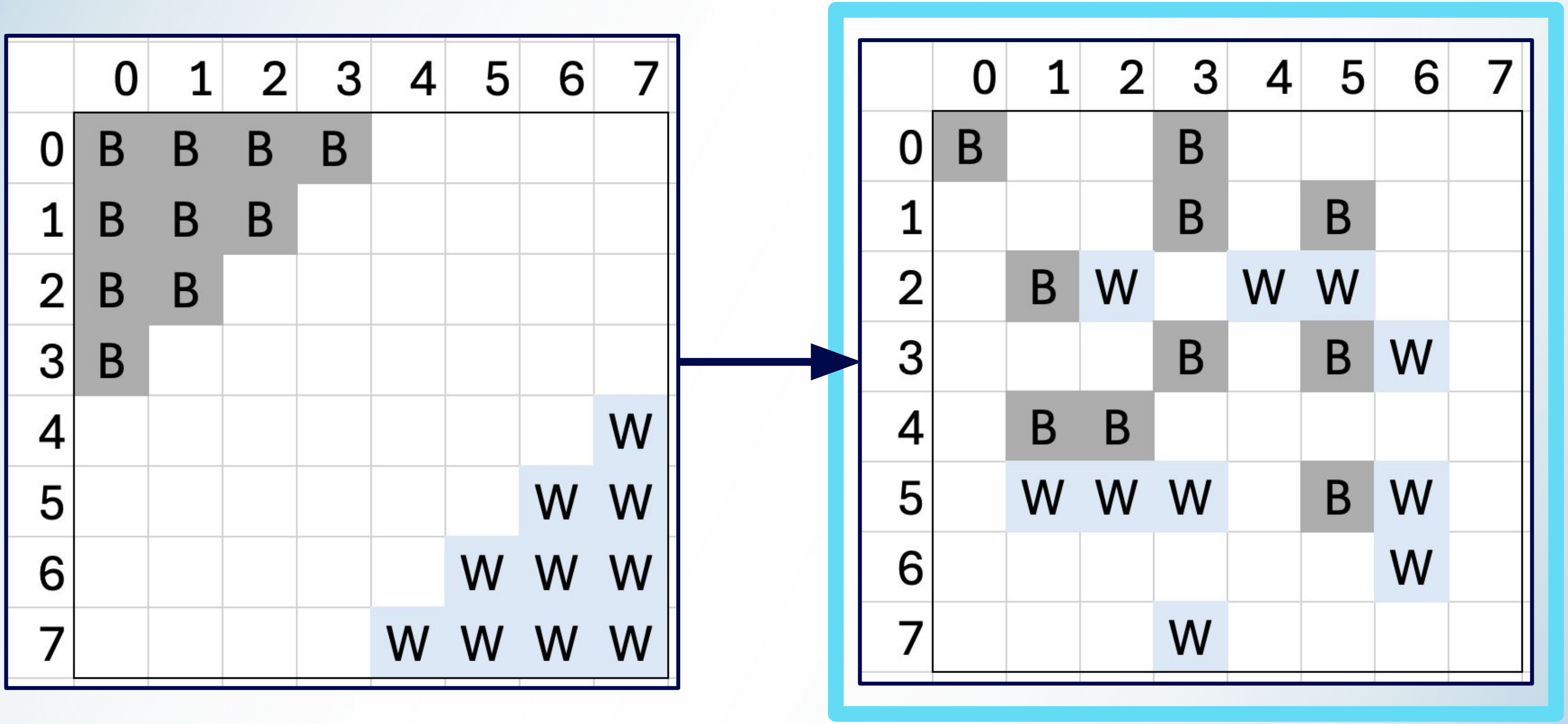


```haskell
parallelMinimax :: GameState -> Int -> Int -> Int -> Bool -> (Int, GameState)
parallelMinimax gameState depth alpha beta maximizingPlayer =
    let b = board gameState
    in case isGameOver b of
        Just winner -> if winner == White then (10000, gameState) else (-10000, gameState)
        Nothing ->
            if depth == 0
            then (evaluateBoard b, gameState)
            else
                let moves = getAllMoves gameState
                    results = parMap rpar (\move -> minimax move (depth - 1) alpha beta (not maximizingPlayer)) moves
                    bestEval = if maximizingPlayer
                               then maximumBy (\(v1, _) (v2, _) -> compare v1 v2) results
                               else minimumBy (\(v1, _) (v2, _) -> compare v1 v2) results
                in bestEval
```

04

Performance & Evaluation

Game State



	0	1	2	3	4	5	6	7
0	B	B	B	B				
1	B	B	B					
2	B	B						
3	B							
4								W
5							W	W
6						W	W	W
7					W	W	W	W

	0	1	2	3	4	5	6	7
0	B			B				
1				B		B		
2		B	W		W	W		
3				B		B	W	
4		B	B					
5		W	W	W		B	W	
6							W	
7				W				

Performance

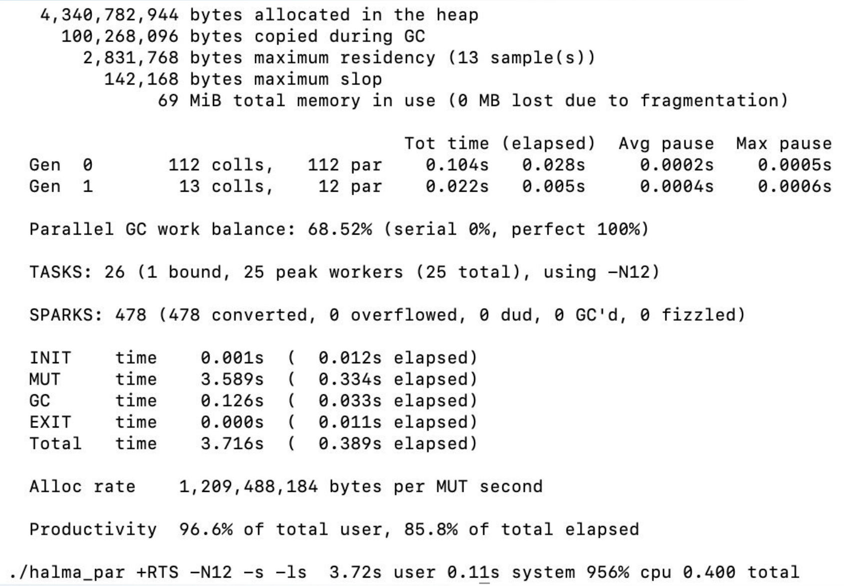
	Avg. Run Time (s)
Sequential Minimax	14.81
Sequential Minimax + Alpha Beta Pruning	0.79

Parallel Minimax (first level)												
Threads	1	2	3	4	5	6	7	8	9	10	11	12
Total Run Time (s)	2.47	1.35	0.96	0.74	0.62	0.54	0.50	0.47	0.44	0.42	0.40	0.39

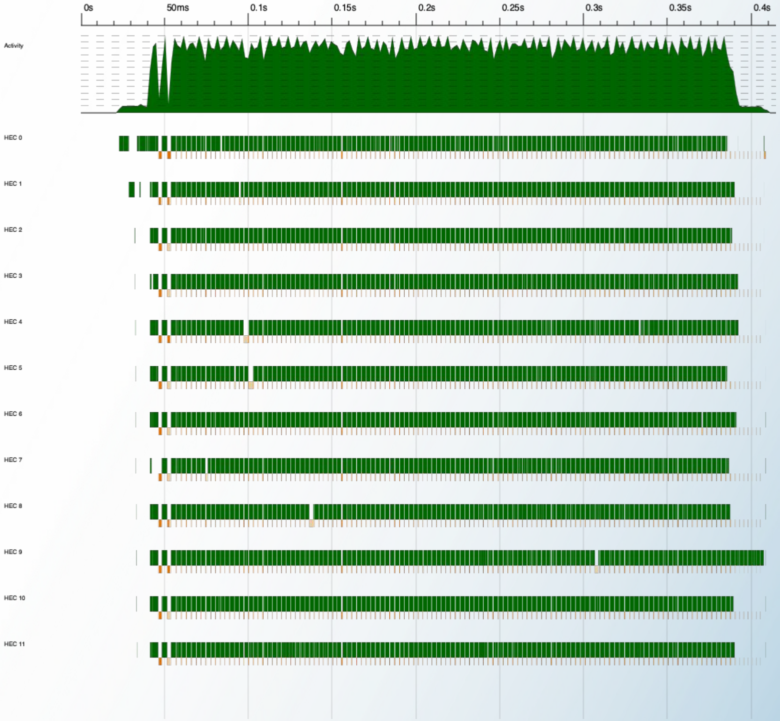
Performance

	Avg. Run Time (s)
Sequential Minimax	14.81
Sequential Minimax + Alpha Beta Pruning	0.79
Top Level Parallelism (12 threads) + Latter Layers Alpha Beta Pruning	0.39
Chunk Parallelism with Global Bounds Updating Alpha Beta Pruning	0.19

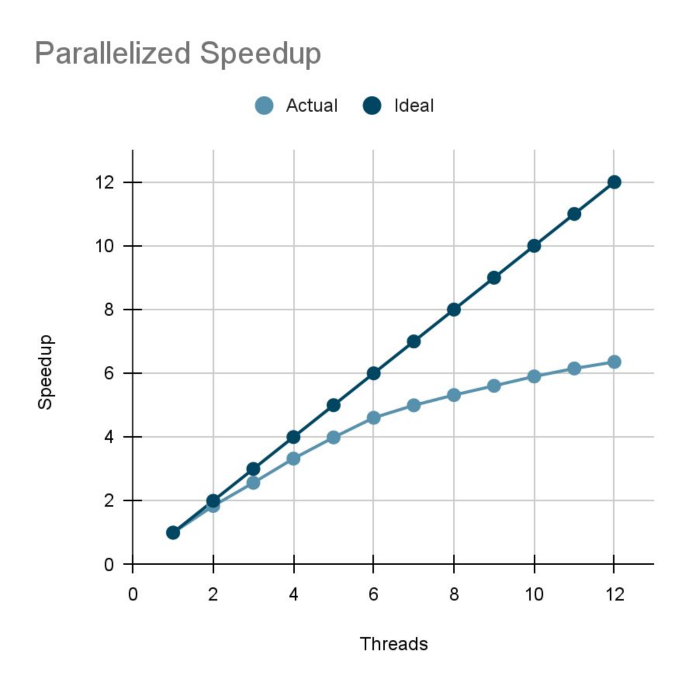
Performance - 12 Threads



```
   4,340,782,944 bytes allocated in the heap
     100,268,096 bytes copied during GC
       2,831,768 bytes maximum residency (13 sample(s))
         142,168 bytes maximum slop
              69 MiB total memory in use (0 MB lost due to fragmentation)

                                     Tot time (elapsed)  Avg pause  Max pause
  Gen  0       112 colls,   112 par    0.104s   0.028s     0.0002s    0.0005s
  Gen  1        13 colls,    12 par    0.022s   0.005s     0.0004s    0.0006s

  Parallel GC work balance: 68.52% (serial 0%, perfect 100%)

  TASKS: 26 (1 bound, 25 peak workers (25 total), using -N12)

  SPARKS: 478 (478 converted, 0 overflowed, 0 dud, 0 GC'd, 0 fizzled)

  INIT    time    0.001s  (  0.012s elapsed)
  MUT     time    3.589s  (  0.334s elapsed)
  GC      time    0.126s  (  0.033s elapsed)
  EXIT    time    0.000s  (  0.011s elapsed)
  Total   time    3.716s  (  0.389s elapsed)

  Alloc rate    1,209,488,184 bytes per MUT second

  Productivity  96.6% of total user, 85.8% of total elapsed

./halma_par +RTS -N12 -s -ls  3.72s user 0.11s system 956% cpu 0.400 total
```



Performance



05

Next Steps

Next Steps: Future Optimizations

- Explore further hybrid approaches between parallel and sequential strategies
- Use best values up to a certain depth to update global alpha-beta bounds, allowing earlier pruning and more efficient searching by feeding back values to the root node for continuous refinement.

Thank you!