COMS W4995 003
Parallel Functional Programming
Fall 2024

Generalized Tic Tac Toe Solver

Instructor: Prof Stephen A Edwards
Student Name: Milin Saini
Student UNI: mks2249

Problem Definition

- Implement a generalized Tic Tac Toe solver
- Find the best moves for Tic Tac Toe in a N X N grid
- NP-complete problem
- Objective is to use parallelization to improve the performance over sequential implementation

Tic Tac Toe

Game Rules:

Board Configuration: The game is played on an N×N grid, where N ≥ 3.

Players: Two players take turns placing their respective symbols on empty cells.

Win Condition: A player wins by placing K consecutive symbols in a horizontal, vertical, or diagonal line, where K ≤ N.

Draw Condition: If all cells are filled without any player achieving the win condition, the game ends in a draw

Implementation

- Sequential Minimax Algorithm
  * Depth-first game tree search
  * Complete state space exploration
  * No alpha-beta pruning initially, but added later

Implementation

Parallel Implementation

* Haskell's par/rdeepseq constructs
* Concurrent move evaluation
* Parallelized game tree exploration

Implementation

Performance Evaluation

*Added timing functions in the code for performance evaluation

*Built the project to enable profiling by Threadscope

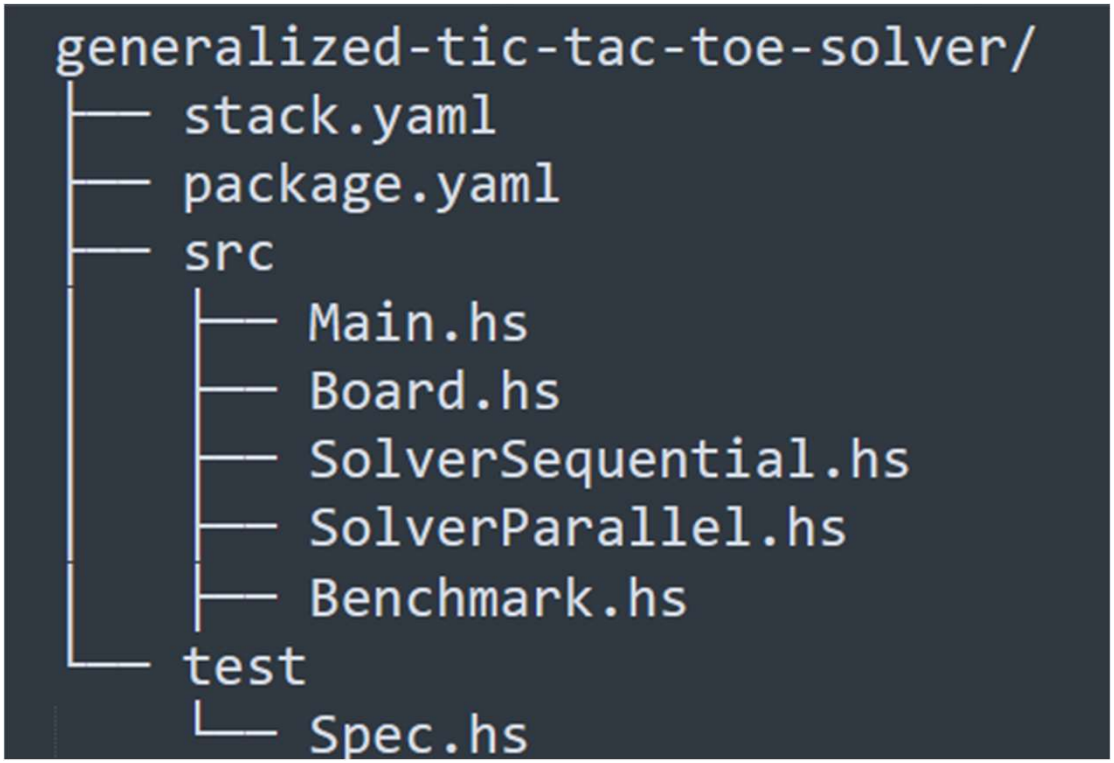
Environment: GHC 6.9.9, Stack 3.1.1

Processor Specs

Brand	Intel
Model	i5-8265U
Cores	4
Hardware Threads	8

https://www.intel.com/content/www/us/en/products/sku/149088/intel-core-i58265u-processor-6m-cache-up-to-3-90-ghz/specifications.html

Code Structure



Code Structure

`stack.yaml`: Stack configuration file with necessary resolver and extra-deps.

`package.yaml`: Package configuration listing dependencies, executables, test suites, etc.

`src/Main.hs`: Entry point of the program, sets up benchmarking scenarios, runs sequential and parallel solvers.

`src/Board.hs`: Defines board types, players, moves, and related functions.

`src/SolverSequential.hs`: Implements the sequential minimax solver.

`src/SolverParallel.hs`: Implements the parallel minimax solver using Haskell's `parallel` strategies.

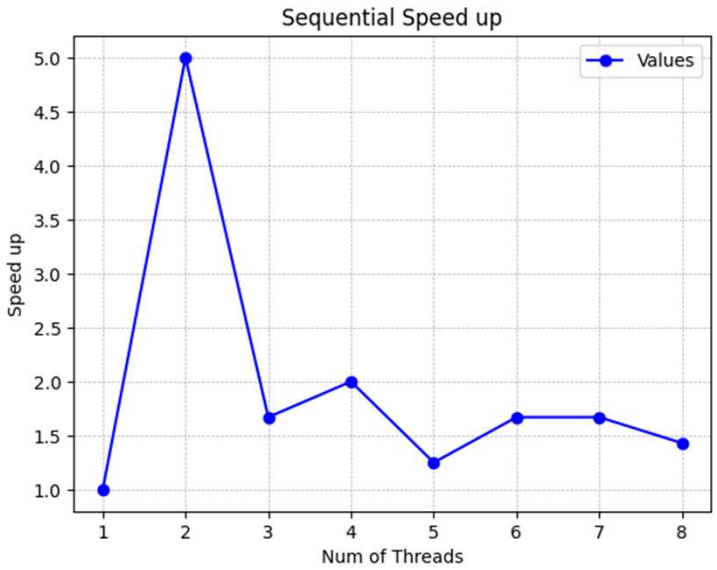
`test/Spec.hs`: Basic test cases for correctness on small boards.

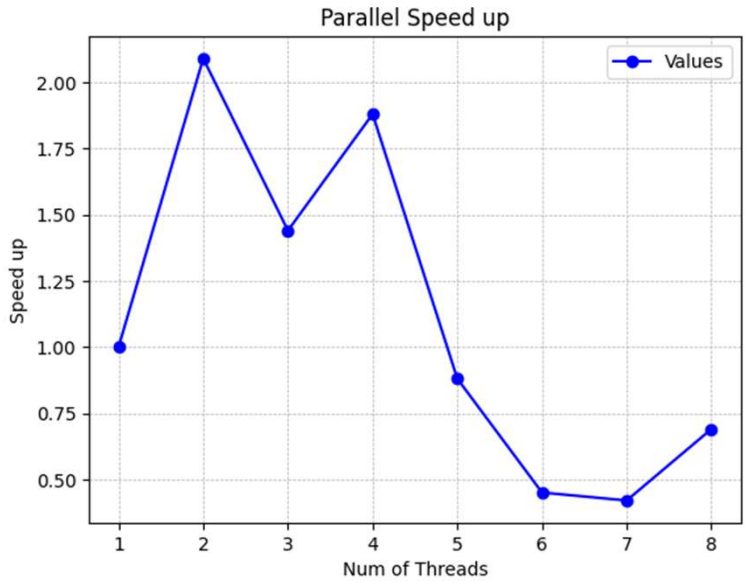
`src/Benchmark.hs`: Code for running timing tests and reporting performance metrics.

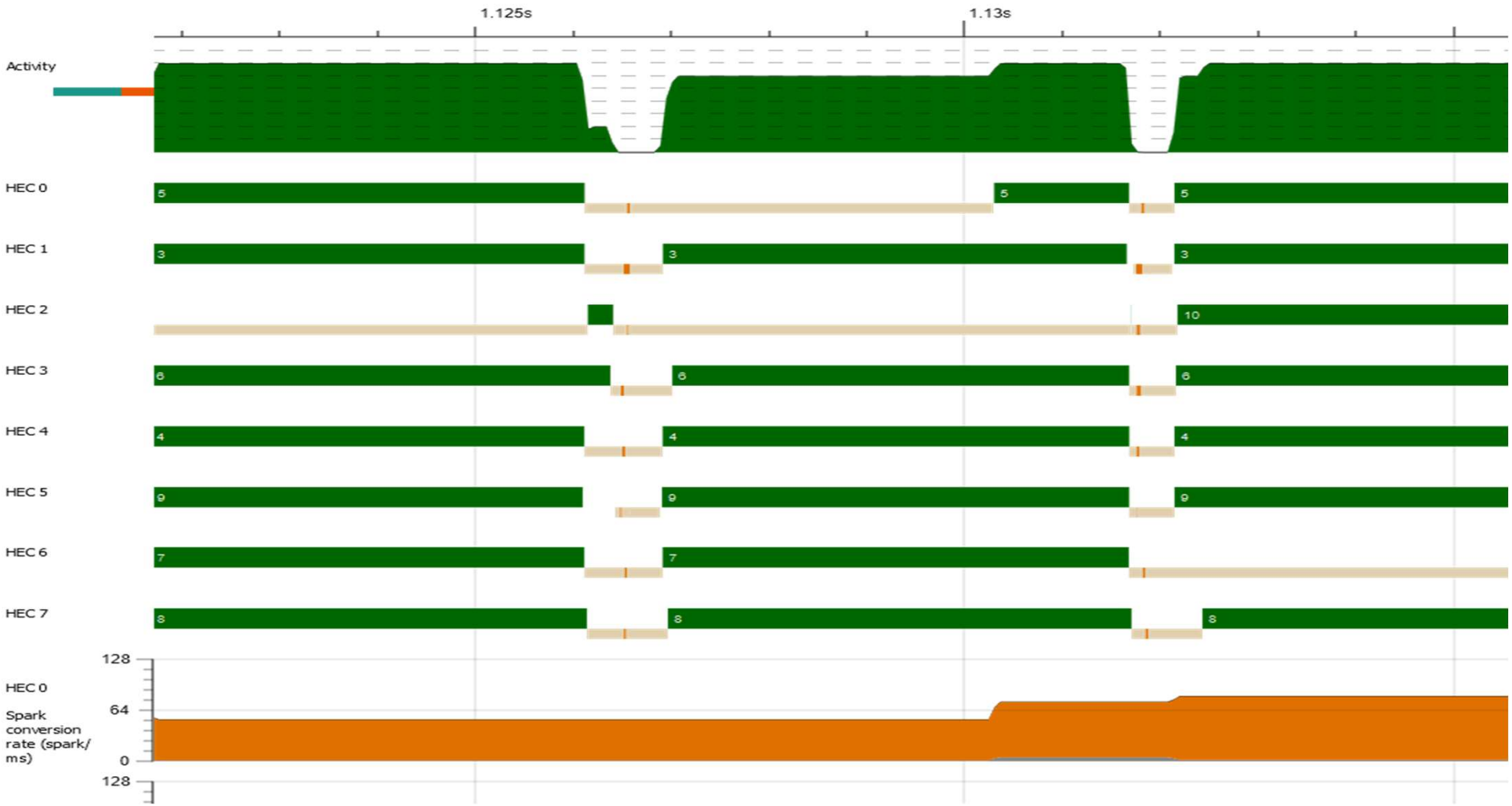
Results - Execution times (3x3 Grid)

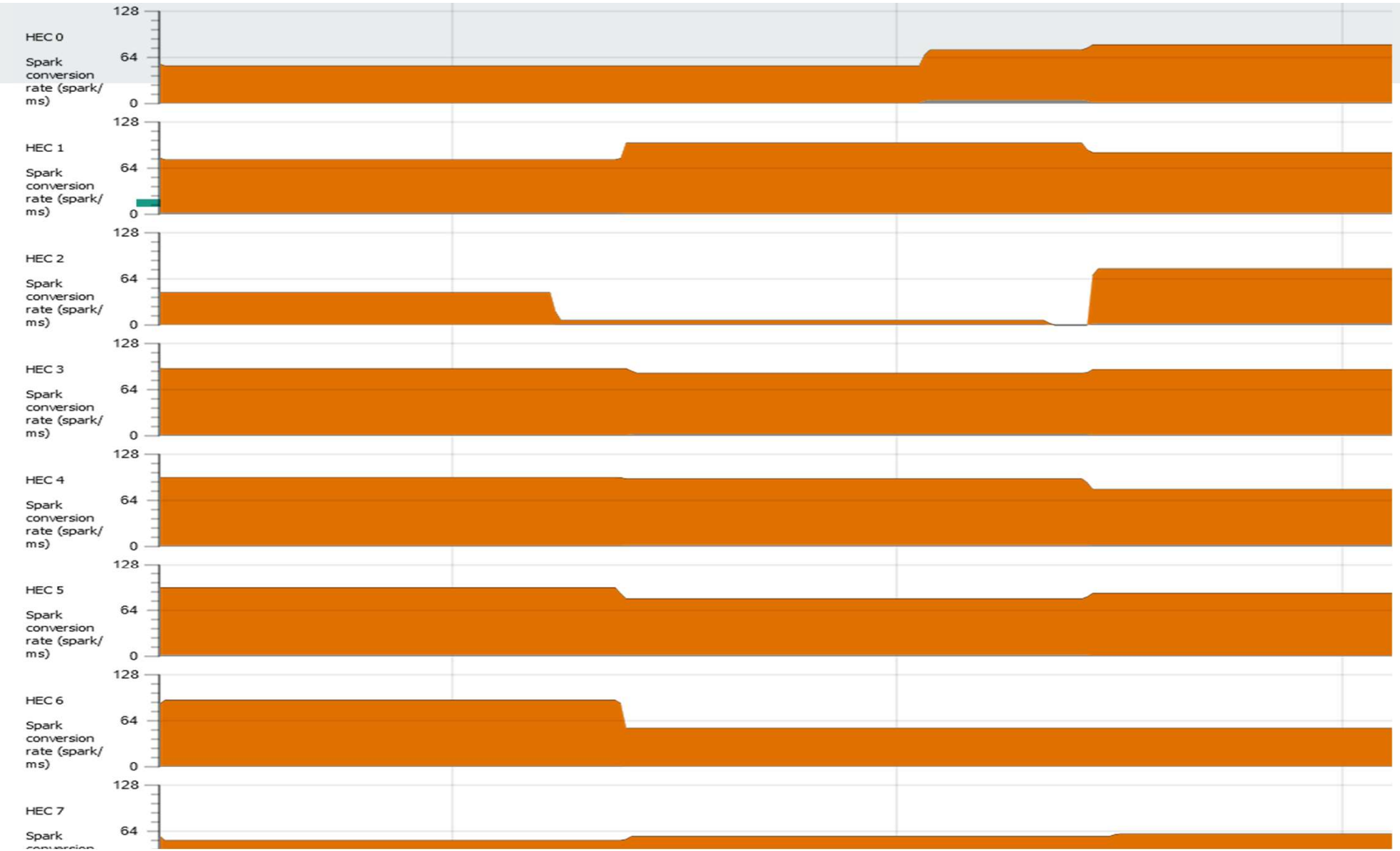
Sequential(ms)	Parallel (ms)
156.25	3140.63
31.25	1500.00
93.75	2187.50
78.13	1671.88
125.00	3562.50
93.75	6906.25
93.75	7421.88
109.38	4562.50

Results – Speed up (3x3 Grid)






1.125s
1.13s
Activity
HEC 0
HEC 1
HEC 2
HEC 3
HEC 4
HEC 5
HEC 6
HEC 7
5
3
10
6
4
9
7
8
128
HEC 0
Spark conversion rate (spark/ms)
64
0
128


HEC 0
Spark conversion rate (spark/ms)
128
64
0
HEC 1
Spark conversion rate (spark/ms)
128
64
0
HEC 2
Spark conversion rate (spark/ms)
128
64
0
HEC 3
Spark conversion rate (spark/ms)
128
64
0
HEC 4
Spark conversion rate (spark/ms)
128
64
0
HEC 5
Spark conversion rate (spark/ms)
128
64
0
HEC 6
Spark conversion rate (spark/ms)
128
64
0
HEC 7
Spark
128
64

Results - Execution Time

- 4x4, 5x5 variants computations in the orders of trillions and more (16! moves)
- Current implementation with just minimax would take several days
- Optimized with Alpha Beta pruning
- But CPU parallel runtime still estimated to be 2-3 days
- Presenting challenge in profiling higher grid size
- Proceed with 3x3 grid for now

Challenges and Areas of Improvement

Scope of improving Parallel Solver performance

- Remove excessive Parallelization In SolverParallel.hs
- First level in bestMoveParallel with parList
- Again in maximizeAB and minimizeAB with nested parList calls
- Evaluation on small grid size
- Overheads in parallelization outweigh the processing gain in small grid size

Thank you

References

- https://wiki.haskell.org/index.php?title=ThreadScope_Tour
- https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning
- https://en.wikipedia.org/wiki/Minimax
- https://www.haskell.org/ghcup/
- https://hackage.haskell.org/