Parallel MCTS-minimax hybrids

Peter de Groot and Sai Bhargav Yalamanchi

URL: http://ysaibhargav.github.io/parallel-mcts-minimax-hybrids

SUMMARY: We are going to implement a hybrid MCTS-minimax algorithm to solve chess problems. The algorithm will take advantage of various sources of parallelism in the algorithm with CUDA and OpenMP and the times to execute will be compared against speedup over sequential.

BACKGROUND: Minimax with alpha-beta pruning has been the state of the art algorithm used by many chess engines (until the advent of AlphaZero). A drawback of using bare bones minimax is that the search space becomes combinatorially large and unwieldy to perform brute force enumerations in order to determine the best move. Monte Carlo Tree Search is a best-first tree search algorithm that focuses on the most promising lines of play. However it often succumbs to traps in games like chess either due to missing a critical move or underestimation of utilities of certain terminal states. A hybrid approach of MCTS-minimax can potentially result in an algorithm that is fast (from avoiding brute force enumerations where possible due to MCTS) and can avoid traps (from exhaustive minimax searches in near terminal states).

CHALLENGE: This is an interesting problem from the parallelism standpoint as the game tree grows exponentially quickly - with naive minimax, one could parallelize the minimax action evaluation process by adding nodes to a work queue that threads/cores could access and pull work out of when they are done executing their current tasks, this is the approach described in [3]. [2] describes several methods to parallelize naive MCTS. The key challenge in our proposed idea is to obtain a speedup in a hybrid of MCTS and minimax, for this we plan on using ideas from both [2] and [3]. There are two axes of parallelism in such hybrids - MCTS tree level and minimax level. Too large a workload size may result in very deep minimax searches which may not be tractable, the right workload size would need to present sufficiently large game trees we could parallelize over while ensuring feasibility of minimax searches. To determine this size we plan on gradually increasing the workload (varying the n in ‘mate in n’) until this balance is achieved.

RESOURCES: CUDA capable GPU will be required in addition to a system that can run OpenMP. Thus the GHC and latedays machines should suffice.

GOALS AND DELIVERABLES: Our focus is on the following hybrid-parallel models

1. Minimax-rollout leaf parallel model
2. Minimax-selection/evaluation root parallel model
3. Minimax-selection/evaluation tree parallel model

Minimax-rollout is a hybrid method in which the MCTS node value estimation is done in a more informed way through minimax searches starting from the node instead of performing random actions until a terminal state is reached. Minimax-selection/evaluation on the other hand triggers minimax searches at certain nodes based on a heuristic (often visit count based) to validate existing estimates of the node values. This is helpful in growing the tree quickly as it avoids unnecessary simulations at nodes for which we already have reliable information. For more details, see [1].

In minimax-rollout leaf parallel, the idea is to parallelize minimax searches at the leaf nodes. Only the simulation step is performed in parallel. In minimax-selection/evaluation root parallel, each thread has an independent copy of the MCTS tree. The minimax search would be triggered locally based on the visit count heuristic. The trees are updated independently of one another; in order to make an action in the game, the trees are combined together as described in [2]. In minimax-selection/evaluation tree parallel, there is only one tree shared by the threads/cores. Each thread/core handles a randomly sampled leaf node and triggers minimax search in the evaluation step based on the heuristic as described earlier. For updates to the tree we plan on exploring global/local mutexes as described in [2].

What we plan to achieve is to get a CUDA implementation of the leaf node parallelization of the MCTS-minimax hybrid and an OpenMP implementation of root parallelization for comparison. Additional goals include getting a CUDA or OpenMP implementation of tree parallelism and comparing that to the other implementation and writing a domain specific language for the tree searches. We aim to determine whether OpenMP or CUDA would be the most effective method of parallelizing our MCTS-minimax hybrid algorithm.

PLATFORM CHOICE: For our leaf parallelism, the GPU with the CUDA cores makes the most sense since the individual minimax search paths will spawn multiple simple paths that can be parallelized over when we have the cores for the task. With root parallelization, we plan to parallelize the search across nodes for each node in the tree, leading to a simpler OMP implementation since each node will use different data from the tree. We plan to use C++ to write the sequential algorithm.

SCHEDULE:

Week 1	Sequential code (combining chess code + MCTS + minimax, implementing and getting minimax-rollout hybrid to work)
Week 2	Sequential (implementing and getting minimax-selection hybrid to work) + CUDA leaf parallelism (minimax-rollout)
Week 3	CUDA leaf parallelism (minimax-rollout) + OpenMP root parallelism (minimax-selection)
Week 4	OpenMP root parallelism (minimax-selection)
Week 5	Tree parallelism (CUDA/OpenMP) (minimax-selection)
Week 6	Report and finalization

REFERENCES:
[1] https://dke.maastrichtuniversity.nl/m.winands/documents/mcts-minimax_hybrids_final.pdf
[2] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.159.4373&rep=rep1&type=pdf
[3] http://olab.is.s.u-tokyo.ac.jp/~kamil.rocki/rocki_ppam09.pdf

RESOURCES:
Reference code base:
Chess - https://github.com/tobijk/simple-chess
MCTS github repo - https://github.com/memo/ofxMSAmcts
Minimax github repo - https://github.com/ryancesiel/tictactoe-minimax

Chess puzzle PGNs:
https://chess.stackexchange.com/questions/19633/chess-puzzle-database-with-pgn-or-fen
http://gorgonian.weebly.com/pgn.html Auerswald collection