**Parallelizing Alpha-Beta pruning algorithm**

**Objective:** In this project we plan to parallelize Alpha-Beta pruning algorithms which are widely used in various two-person zero sum normal-form games like chess, checkers, tic-tac-toe and Go. We intend to first execute the sequential code of these algorithms and then move on to parallelizing them, so that we can compare them on their performance. Each member of our team will be doing this for one game among chess, checkers and tic-tac-toe.

A brief description of various search techniques/algorithms has been given below.

**Alpha-Beta pruning algorithm**

Alpha-beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. Alpha-beta pruning is a way of finding the optimal minimax solution while avoiding searching subtrees of moves which won’t be selected. In this way, Alpha-beta pruning is an improvement over the minimax algorithm. The problem with minimax is that the number of game states it has to examine is exponential in the number of moves resulting in a lot of time cost. Borrowing the idea of pruning, the algorithm allows us to discard large parts of the tree from consideration. When applied to minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

**Search and Minimax algorithms**

Search algorithms tend to utilize a cause-and-effect concept—the search considers each possible action available to it at a given moment; it then considers its subsequent moves from each of those states, and so on, in an attempt to find terminal states which satisfy the goal conditions it was given. Upon finding a goal state, it then follows the steps it knows are necessary to achieve that state.

For two player games, the minimax algorithm is such a tactic, which uses the fact that the two players are working towards opposite goals to make predictions about which future states will be reached as the game progresses, and then proceeds accordingly to optimize its chance of victory. The theory behind minimax is that algorithm’s opponent will be trying to minimize whatever value the algorithm is trying to maximize. Thus, the computer should make the move which leaves its opponent capable of doing the least damage.

**Parallelizing Chess program**

Fundamental to chess and any other turn-based perfect information game is the game tree. The tree represents all possible paths through the game’s state space. Each node in the tree is a game state, and each arc represents a legal move for the player on-move at that state. In many games, including chess, the game tree in reality is a directed acyclic graph, where different sequences of moves can transpose into the same board position. In order to choose its move, a game-playing program searches for a path through the tree that terminates with the most favorable result. Assuming that each player will attempt to respectively minimize or maximize its score at every turn leads to the recursive algorithm af minimax search.

In order to parallelize the search, it is necessary to assign processors simultaneously search multiple subtrees of the game tree. The technique which we’ll be using is principal variation splitting where at each node, it first recursively searches the left-most branch to find an alpha-bound for the remaining branches which can then be searched with parallel invocations of AlphaBeta.

Shailja- write about you tic tac toe game and need to parallelize algorithms.

Kumari- write about checkers and make additions if you want.

We’ll complie everything tomorrow.