**ABSTRACT**

Alpha-Beta pruning is one of the most powerful and fundamental MiniMax search improvements. It was designed for sequential two-player zero-sum perfect information games. Here,we will introduce an Alpha-Beta-like sound pruning method for the more general class of “stacked matrix games” that allow for simultaneous moves by both players. This is accomplished by maintaining upper and lower bounds for achievable payoffs in states with simultaneous actions and dominated action pruning based on the feasibility of certain linear programs. Empirical data shows considerable savings in terms of expanded nodes compared to naive depth-first move computation without pruning.

**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.

**INTRODUCTION**

When searching game trees, especially in a competitive setting, significant benefits can be achieved by pruning branches which under no circumstances can affect the decision being made at the root. The best known pruning method is the Alpha-Beta algorithm . It applies to sequential zero-sum two-player games with perfect information such as Chess and Checkers and Tic Tac Toe. Alpha-Beta maintains upper and lower value bounds to decide whether branches can be cut. This type of pruning can lead to considerable search reductions — essentially doubling the search depth over the original MiniMax algorithm when given the same search time.

**Search and Minimax Algorithm**

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.

**NEED FOR PARALLELIZING ALPHA-BETA ALGORITHM**

Serial computers is well-developed and has historically proved its prowess. It is logical, however, to desire to outperform the serial algorithms by using parallel computer architectures. Various implementations of parallel algorithms of alpha-beta are used e.g. i) Even-Split ii)Master/Slave Model. In fact, if great care is not given in parallelizing the algorithm, parallel performance can degrade well below the performance of the serial algorithm . The master/slave model is more than twice as efficient in its use of parallelism. It exhibits a bit of super-linear speedup when going from one to two processors, due to the unnecessary communication overhead present when using only one processor.Hence,while executing sequential algorithms ,there is lot of overhead due to one part of subtree interacting with another subtree ,thus CPU wastes a lot of time waiting idly with unnecessary execution time which increases the time complexity.Here,comes the picture of parallel algorithms where ,in master/slave model ,the master processor assigns tasks to slave processors and after completion of work,each processor sends the result back to master which compares with previously accumulated results and then broadcasts it to everyone.Hence,parallel efficiency is limited by number of processors.Hence,parallel program ensures that all processor work continuously until all the subtrees are pruned ,illustrating that parallel algorithms outperform serial algorithms.

**CHESS**

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.

**TIC TAC TOE**

The tree is partitioned at the top level and each processor investigates a single possible move. However, if alpha-beta cutoffs are to be used, each tree will have to search for its own bounds, and can't make use of better bounds found by other processors.The two methods for game tree partitioning are: i)partitioning at width ii)partitioning at depth.We will go for implementing partitioning at width where if three processors are there ,then 0,3,6 mark is given to processor 1,while 1,4,7 is given to processor 2 and 2,5,8 is given to processor 3 and then parallel computation model based on alpha-beta pruning algorithm is applied with synchronization implications.

**CHECKER**

**N-ARY GAME TREE:**

For finding the optimal move of the checker board that can be made by the machine, N-ary Game tree is generated based on the contemporary state of the checker board by fixing the depth. The N-ary Game tree is generated as follows.

Step 1: The initial node that is the root node consists of the contemporary state of the checker board.

Step 2: And the forthcoming levels of the tree consist of all possible moves that can be made by the machine and the opponent alternatively.

Step 3: Step 2 is repeated until the game tree of depth n gets generated.

Step 4: Heuristic value is calculated for all the leaf nodes of the game tree

**HEURISTIC VALUE:**

For a checker board

The pawns of the machine are given value 1.

The Kings of the machine are given value 2 (since these are favorable for the machine to win)

The pawns of the opponent are given value -1(since these are less favorable for the machine to win)

The kings of the machine are given value -2 (since these are unfavorable for the machine to win)

And the sum of those values present in the checker board is calculated and it is known as the heuristic value of the checker board, more the heuristic value more the favor for the machine to win the game.

**PARALLELIZATION OF GAME TREE USING MPI (MESSAGE PASSING INTERFACE):**

The winning chances of the machine is increased by increasing the depth of the game tree. In order to increase the time efficiency the algorithm is parallelized using MPI.

MPI that runs on shared memory is used to parallelize since there is a use of pointers to generate game tree. Parallelization is done for two cases:

1) Game tree generation

2) Game tree traversal.

For both the cases the tree is parallelized at the depth=2 considering the depth of the root to be 1.

Hence, all nodes till depth 2 are shared by all processors. The following figure represents the parallelization mechanism.

