PARALLELIZED MINIMAX AND ALPHA-BETA PRUNING FOR CHESS

Zhifang Zeng, Chenxi Yang

Department of Electrical Engineering and Computer Science
University of California, Irvine

ABSTRACT

In this paper we develop 4 parallelized Artificial Intelligence (AI) models based on the minimax and alpha-beta pruning AI. Model1 is nested thread extension, model2 is static grouped searching, model3 is dynamic task assigning, and model4 is unbalanced searching. Our 4 AI models show better performance in comparison to serial AI, but different models have different speedup.

1. INTRODUCTION

Minimax is a very popular AI algorithm used in many games and is still popular today in many small size chess games. Alpha-Beta pruning is a reinvented version of minimax that aims to decrease the size of the amount of computing and processing, but the core of the two algorithms are the same, both are based on tree searching. In order to combine practice and knowledge from the class, we decided to research on a very classic game that interests both of us, the chess game.

2. BACKGROUND

2.1. Minimax

Minmax is a decision rule used in chess AI.[1] For each board situation, we have an evaluation function which gives a score of the board, the higher the score that a board has, the more advantage the current player has. The minmax develop a decision tree which first lists all boards that contain the possible next move of the current situation. Then the next level lists all possible moves from the opponent. The strategy is that we always want to minimum opponents score and maximize our own score. When the tree reached to the bottom, we calculate the leaves and find the minimum or maximum value then trace back to select the best move.

2.2. Alpha-Beta Pruning

Alpha-beta pruning is an optimization method to reduce the size of the tree. In figure 1, the value of the first level is -50, the left mode is -80. in this case, any third level in the left branch that is greater than or equal to 50 can be pruned, because a wise opponent AI will always try to decrease your score.

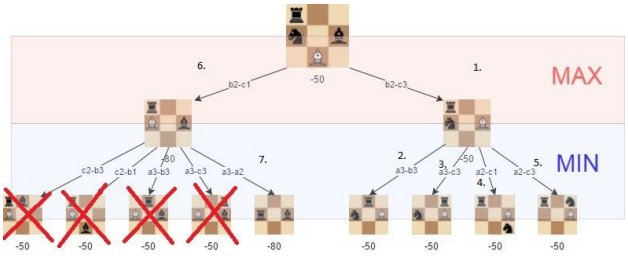

Fig.1. alpha-beta pruning in chess game[1]

3. PROPOSED METHOD: 4 MODELS

3.1. Model1: Nested Thread Extension

The first model is to extend threads for different layer in the search tree. As figure 2 shows, at different steps, the program will create a couple of new threads to do the simulation until it reaches the preset max step thresh.

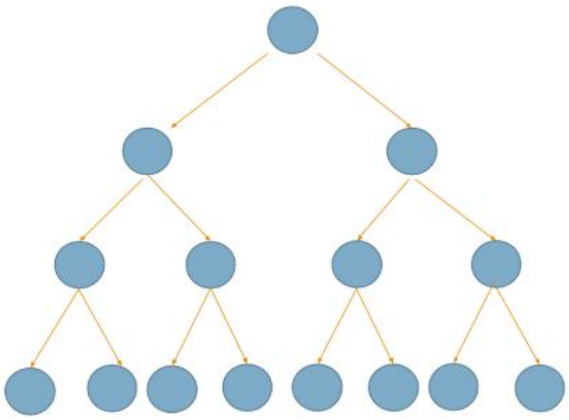

Fig.2. Assign a thread to each node.

3.2. Model2: Static Grouped Searching

In model2, each thread will be responsible for a tree node and its subtree only. And the thread assigned to the certain subtree will do the searching for the whole subtree, no matter what the size of the subtree will be, see figure 3.

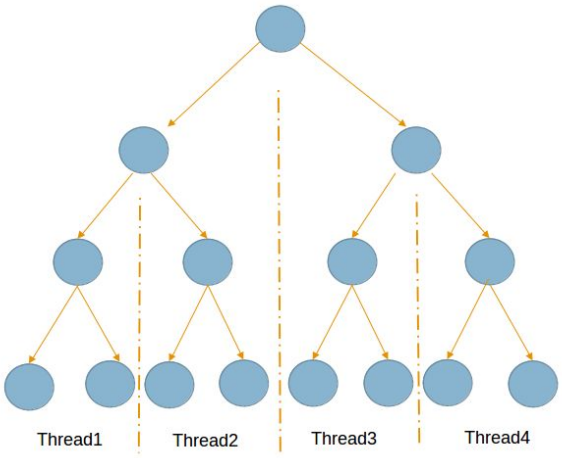

Fig.3. Let each thread search for a specific tree.

3.3. Model3: Dynamic Task Assigning

In model3, all the nodes to process will be viewed as a task, and a queue will be set up to store these tasks. Whenever a thread is free, a task will be dequeued into this thread and start processing, see figure 4.

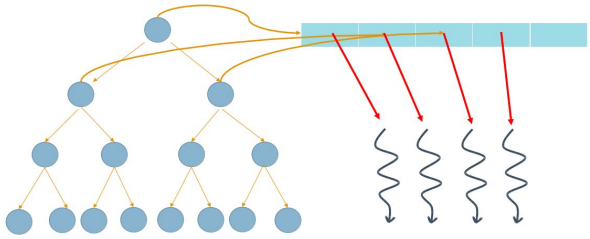

Fig.4. Suppose there is a queue containing the branches to explore, whenever a thread finishes processing, dequeue a branch from the queue and search on this branch.

3.4. Model4: Unbalanced Searching

In model4, after calculating the scores for different nodes in the layer, the nodes of the layer will get different task_size in searching the subtree. For example, for the i th node in the layer, it will get the task_size by get_task_size(Score[i]), where Score[i] is the score of the current state. After getting the task_size of the current state, then task_size of subtrees to be explored from the current state will be set as a new task, so that different computed nods of the current layer will have different efficiency in searching the subtree. For example, if a node of the current layer has the task_size of 4, and there are 16 subtrees to be explored from the current state, then every 4 subtrees will be set as a task. Instead, if the task_size is 1, that means every subtree to be explored will be set as a task, so that different nodes will have different efficiency in searching the subtree, see figure 5.

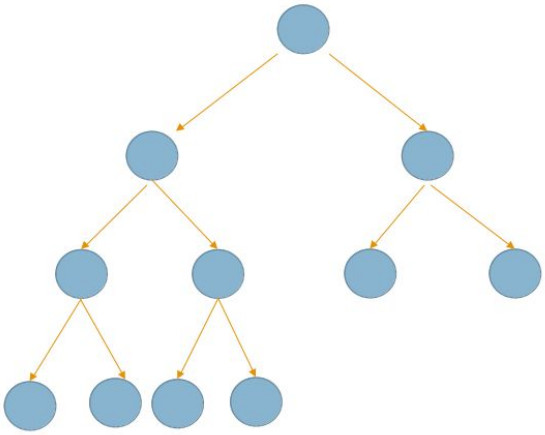

Fig.5. When different thread search on different branches, do a communication after the simulation, and then assigned different number of threads to explore.

Currently we compute the task_size linearly:

$$\textbf{task\_size}= \frac{Score[i]-MinScore}{-1*(MaxScore-MinScore)} \textbf{+MAX\_TASK\_SIZE}$$

So that the task_szie of different nodes will be resized to an integer between 1 and MAX_TASK_SIZE.

After the total number of nodes explored reach a thresh, or the search depth reaches a max depth thresh, the search will be terminated. The value of nodes thresh is attained by playing multiple games between 2 serial AI and record its max explored nodes, which by experiment is 4000000.

4. EXPERIMENT RESULTS

4.1. Experiment Setup:

Processor: Intel i5-8350U @ 1.70GHz ~ 1.90GHz
RAM: 16.0 GB
System: Ubuntu16.04.5 x64
Compiler: gcc 5.6.0
flags: -O3 -Wall

In our experiment we let Different model of AI play with the serial version of AI for 50 times and record the average time that different AI takes to make a move and how much they win, also when the total rounds exceed 300 we consider it a draw.

4.2. Experiment Result:

The result of the average time a model takes to make a move is presented in **Fig.6**, where Y-axis is the average time a model takes to make a move and X-axis is the max step of the minimax tree.

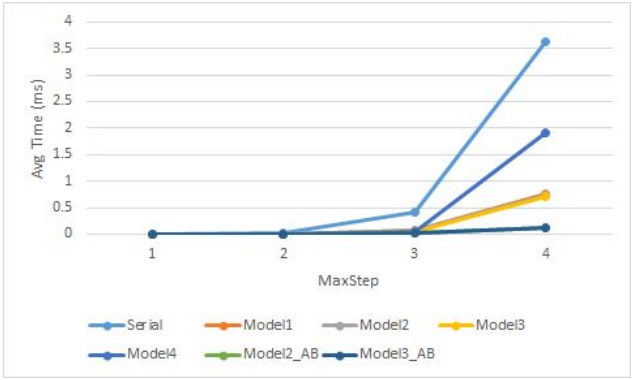

Fig.6. Experiment result of average time of making a move to max step of the minimax tree. (model name followed by AB means it is alpha-beta pruning, else it is just minimax)

From **Fig.6** we can tell that the speedup of the parallelized models grow bigger as the max step of the minimax tree grows. Also, the gap between different models grows bigger when the max step of minimax tree grows bigger as well.

Model	Wins	Draws
1	13	26
2	12	24
3	11	27
4	15	23

Table.1. Experiment result of wins and draws against serial model

To test the improvement of AI in model4, we set the max step of model4 to be 5 and restrict the max number of nodes that model4 can explore. From **Table.1.** we can tell that model4 does has some advantage in the game, but is not clear enough.

5.CONCLUSIONS

In this paper we develop several parallel models of minimax AI and test its performance. Also, we proposed a model that is able to search the minimax tree with different efficiency in different subtrees. From the

experiment result we can derive that 1. When we try to open new threads for different layers in the minimax tree, the speedup will not grow as fast as other models since opening too many new threads will be too costly. 2. Model3 obviously beats Model2, and this is because for different branches, the size of the subtree is also different, thus if different subtrees are assigned fixed threads to process then the CPU resource will not be fully exploited. 3. As the depth of the minimax tree grows bigger, the speedup of different models also grows bigger. 4. Although Model4 helps the AI to win, but it will also decrease the speedup as extra work is necessary to adjust the task size.

11. REFERENCES

[1]Lauri Hartikka (2017, March 30). A step-by-step guide to building a simple chess AI. *FreeCodeCamp*. Retrieved June 12, 2019, from https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/