Implementation of a Distributed Chess Playing Software System Using Principal Variation Splitting

Khondker Shajadul Hasan¹, Alok Chowdhury², Asif Mahbub², Ahammed Hossain² and Abul L. Haque²

¹Computer Science Department, University of Oklahoma, Norman, Oklahoma, USA
²Department of Electrical Engineering & Computer Science, North South University, Dhaka, Bangladesh

Abstract - Many chess playing artificial intelligence software take considerable time in calculating the optimal move that can be made by a computer system. Most computer systems that make such calculations use some version of the minimax search with alpha beta pruning. Minimax with alpha beta pruning can be made distributed with principal variation splitting. The system can thus be used to reduce the time necessary to calculate the optimal move.

Keywords - Adversarial Search, Artificial Intelligence, Minimax Algorithm, Alpha Beta Pruning, Distributed Systems, Principal Variation Splitting.

1 Introduction

The game of chess has been in existence for a very long time. During much of that time it has been played between human players. Only in recent times (relative to the time chess has been played) have computers started to play chess. The idea of a non human playing chess came to realization with the advent of digital computers. Even with the advent of digital computers, the computer could not play chess well. This was mainly due older computers not having the necessary computational power to play chess.

With the improvement in computational power, along with improving algorithms, came better chess playing from computers. However, this alone was not enough to challenge the best human players. So, computer scientists turned to parallel processing to change this. Parallel processing allowed a massive increase in computational power needed to challenge top human players. In 1997, IBM’s Deep Blue supercomputer created a sensation by defeating then world champion Gary Kasparov. This caused the world to take computer chess seriously. Nowadays, computer chess has become very common. Chess software is available on nearly any kind of computer.

However, when playing chess on personal computers, except in easy mode, it takes a lot of time for the computer to calculate its move. If the time needed for the computer to move could be cut, then the game against a computer could be more enjoyable. This is our objective in this project. To reduce the time need by a computer to move.

To do this, distributed computing was used. The processing power of two or more computers connected over a LAN was used to reduce time needed to calculate a move. The part of the program that performs the actual computations will run in the background of the various computers being used. This will allow the computers to be used for other, non cpu intensive purposes, while the computation is being done in the background.

To make chess distributed, it was necessary to divide the task of finding an optimal move for the computer. To do this, Principal Variation Splitting, or PV-Splitting was used. PV-Splitting is an algorithm that was originally designed for shared memory multiprocessor systems. PV-Splitting had to be changed slightly to make it run in a distributed system.

2 Model

The model of the distributed system consists of two types of nodes: a single main node and two or more worker node. The main node divides the job of calculating the optimal move for the computer to make into smaller jobs. These jobs are then assigned to the worker nodes to compute. These jobs are assigned using messages passed from the main node to the worker nodes using a local area network. The worker nodes are implemented as a Microsoft
Windows service. This allows the worker node to run in the background.

The jobs are divided using principal variation splitting. Each job consists of searching the sub tree of a particular node as its root. Different jobs do not have any intersection. This means the sub trees that are searched do not form a directed acyclic graph. Two different jobs with different root nodes do not have anything in common. Thus there is no communication necessary between workers searching different sub trees.

3 Principal Variation Splitting

Principal Variation Splitting [3, 4] or PV Splitting was originally designed for multi-processor systems. In this case, it was adapted to work on a multi computer distributed system. PV Splitting is ideal for our model of the distributed system, as PV Splitting allows the job to be divided into smaller jobs from one main node and then have the job assigned to different workers.

The PV Splitting searches the same tree as the minimax algorithm. It does the search in a parallel fashion, or in this case using multiple computers in a distributed system. PV Splitting is designed for the case when alpha beta pruning is used along with minimax search.

In PV Splitting, starting with the child nodes of the root node, the leftmost child node’s sub tree is searched first and then the remaining child nodes are then searched. This is done recursively. That is when searching the leftmost sub tree, the leftmost child node is expanded into its children first. Among its children, the leftmost child’s sub tree is again searched first before searching the remaining children in parallel.

Each sub tree is searched using the minimax[1] algorithm with alpha beta[2] pruning. When a job is assigned, the alpha beta value of the sub tree’s root node’s parent node is used. Once a job is finished, the min or max value of the sub tree’s root node is returned as a message. The alpha beta values of the parent are then updated.

Since there is only one main node in our distributed system, it will reside in one computer. The many worker nodes may reside in more than one computer. There may be a main node and one or more worker nodes on the same computer. There may also be more than one worker node per computer. This can be used for performance gains for computers with multiple CPUs or multiple cores.

Also, all the worker nodes can reside on a single computer. The system will not be distributed, but will be able to take advantages of multi-cpu or multi-core systems. It will also not suffer from any network lag.

Figure 1: Distributed System Model

![Distributed System Model](image)

Figure 2: Principal Variation Splitting Procedure

This is illustrated in Figure 2. The root node s0 is expanded into its children s11, s12, s13, s14, s15, s16. The left sub tree, s11 and all its descendants are searched first before the other children of s0 are searched in parallel. The node s11 is expanded into s21, s22, s23, s24, s25, s26. Again, the left sub tree is expanded first. That is, s21 and its descendants are searched before the remaining children are searched in parallel. The s21 node is then expanded into its
children s31, s32, s33, s34, s35, s36. The node s31, being the leftmost child node, is searched first before all other nodes. The remaining children, s32, s33, s34, s35, s36 are searched after the search of s31 is complete.

Once the search of the sub tree of s31 has been finished, then sub trees of its sibling nodes s32, s33, s34, s35, s36 are searched in parallel. Once they have been searched, s21 has been searched completely. The siblings of s21: s22, s23, s24, s25 and s26 are then searched in parallel. Once they have been searched, their parent node s11 has been searched completely. Once s11, has been searched, the sub trees of its siblings: s12, s13, s14, s15, s16 are then searched in parallel. Once these searches are complete, their parent s0 has been searched completely. Thus the entire tree has been searched. All searches of a sub tree are done with minimax and alpha beta pruning.

Here is the proposed PV-Splitting[5] algorithm adapted for distributed systems:

```java
function PVSplit(start_state)
begin
    r := PVSplit_Max(start_state, -infinity, +infinity)
    return r
end

function PVSplit_MAX(state, alpha, beta)
begin
    if (state is a leaf node)
        return evaluation_function(state)
    state[0] = first child of state
    if(current depth < max PV-Split Depth)
        r = PVSplit_MAX(state, alpha, beta)
    else
        r = MAX(state, alpha, beta)
        alpha = r
        max = r
        state[1..N] = children of state, not including state[0]
        for i = 1 to n in parallel
            (Wait for a worker)
            r = MIN(s[i], alpha, beta) //MIN
            //being minimax algorithms min //executed on a free worker
            if r < min
                min = r
            if r <= alpha
                prune tree
            if r < beta
                beta = r
        return max
end
```

Next function, PVSplit_MIN algorithm for distributed system is given below:

```java
function PVSplit_MIN(state, alpha, beta)
begin
    if (state is a leaf node)
        return evaluation_function(state)
    state[0] = first child of state
    if(current depth < max PV-Split Depth)
        r = PVSplit_MAX(state, alpha, beta)
    else
        r = MAX(state, alpha, beta)
        beta = r
        min = r
        state[1..N] = children of state, not including state[0]
        for i = 1 to n in parallel
            (Wait for a worker)
            r = MAX(s[i], alpha, beta) //MAX
            //being minimax algorithms min //executed on a free worker
            if r < min
                min = r
            if r <= alpha
                prune tree
            if r < beta
                beta = r
        return min
end
```

Each sibling in a parallel search [3] is searched using a worker. Each worker has the ability to search a sub tree using minimax and alpha beta pruning. When the parallel search is started, each sibling node attempts to acquire the free worker to make sure that sibling nodes do not attempt acquire more workers than exists in the system, a semaphore is used.

Once a node acquires a worker, it instructs the worker (through message passing) to search the sub tree of its node. Once the search is done, the worker returns the min/max value for the node as well as other search statistics using message passing. Using the min/max value of the sibling node, the min/max value of the parents is updated as well the alpha beta values.

One of the shortcomings of PV Splitting occurs when the number of worker nodes is greater than the number of sibling nodes that are to be searched. Some of the workers will remain idle if this were the case. The average branching factor for chess is over 30. So on average, for over 30 workers, some of the worker would remain idle.
4 Analysis of Results

Since the objective of the project was to create a distributed program that would reduce the time necessary to calculate moves, the system was evaluated for the amount of time taken by the program in two different scenarios. In the first scenario, one computer was used to calculate the moves. In the second scenario, three computers were used to calculate the same move.

First a set of states from which to perform the evaluations were created. Each state is a certain configuration of pieces on the board. For each of the states, move was initiated by the human player and then the time taken for the system to respond with its move in both scenarios was determined. For the first scenario a move was made and then the time taken system to make its move was observed. In the second scenario, the same move was made from the same state. The time required to make the move as well as the number of nodes searched by each computer was measured.

In the first scenario, a single pc was used and performed moves for certain states. These states are labeled 1, 9, 15, 24. In the second scenario, the same states 1, 9, 15, 24 were used again. In all states a remarkable decrease in the times required to calculate a move was seen. In state 1, the time decreased from 35.48 seconds to 12.5 seconds. In state 9, the time decreased from 45.07 seconds to 20.29 seconds. In state 15, time decreased from 69.87 seconds to 26 seconds. And in state 24, the time decreased from 40.06 seconds to 16.25 seconds. The results are shown as a bar graph in Figure 3.

The rate at which nodes wear search was also calculated. The average search speed on one PC in the first scenario was 421946 nodes/sec. In the second scenario, involving three computers, the search speed was always over 1200000 nodes/sec. Thus the search speed was increased over three times by using three computers instead of one. (It should be noted that one of the computers was a dual core and both cores were used). The results are shown as a bar graph in Figure 4.

In the interest of load balancing, the faster computers got more nodes to search. It can be seen from the table below that the dual core PC#3 searched the most nodes because it was the fastest (summing nodes searched over the two cores).

5 Conclusion

The PV-Splitting algorithm adapted for distributed systems was implemented and was tested. The results of the test were quite impressive. The
moves made by the computer in response to human player moves took far less time than on a single computer.

The tests show that the system significantly reduces time when there were many computers connected instead of just one. The system was tested in a case of one computer vs. three computers. The results show a large increase in speed, where speed was calculated as the number of nodes searched per second. Thus the amount of time taken to calculate a move was decreased, which was the main objective for the system.

The system still has some deficiencies in that it cannot take advantage of more computers than the number of branches of a node. The average branching factor is 35. This is a flaw of the PV-Splitting algorithm. The minimax implementation in the worker nodes can also be improved to provide faster performance.

6 Reference


