Parallelized Mini-Max Search and Alpha-Beta Pruning

Wei-Cheng Wu
NCTU EECS
spencerwu85@gmail.com

Tsung-en Hsiao
NCTU EECS
tsn@cs.nctu.edu.tw

Yi-Fan Wu
NCTU CS
jameswu1212@gmail.com

Po-Yi Chou
NCTU EECS
bob2006tw@gmail.com

ABSTRACT

Parrellize the alpha-beta search algorithm. Test and compare the performance of different method by Chess.

KEYWORDS

Parallel, Alpha-beta Search, Mini-max Search, Game

ACM Reference format:
Wei-Cheng Wu, Tsung-en Hsiao, Yi-Fan Wu, and Po-Yi Chou. 2016. Parallelized Mini-Max Search and Alpha-Beta Pruning. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 3 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 PARTICIPANTS

- Wei-Cheng Wu
  - Chessboard and Chesspiece data structures implementation.
  - Minimax search and Alpha-beta search algorithm implementation.
- Tsung-en Hsiao
  - Minimax search and Alpha-beta search algorithm implementation.
  - Parallelized algorithm implementation.
- Yi-Fan Wu
  - Chess game score evaluation.
- Po-Yi Chou
  - Chess game score evaluation.

2 INTRODUCTION

Nowadays, people play games with computer AI. Chess is one of them, and how to make the AI more intelligent is a subject which people have spent a great time working on. To understand how chess AI works, we decided to make our own one. However, the more complicated algorithm we apply to our AI, the more time it takes for every step to make decision. Consequently, to speed up the performance, we parallelize it. Our works use the evaluation policy to grade every kind of chessboard based on other's long-term work and use Alpha-Beta Pruning to search every possible situation after current chessboard to decide which step is the best way to win the game. To speed up the performance as much as possible, we parallelize our search algorithm with glib thread pool.

3 PROPOSED SOLUTION

3.1 Parallel Mini-max Search Algorithms

There are three major parallel algorithms.

(1) PVSplit (Principle Variation Split): easier to implement, less scalability, poor load balancing.
(2) YBWC (Yonng Brothers Wait Concept): little bit better scalability
(3) DTS (Dynamic Tree Splitting):
   - good scalability, good load balancing
   - most of chess engines use this
   - hard to implement

3.2 Introduction to PVSplit Algorithm

The most simple way to parallize alpha-beta search algorithm is just open thread in the root of some subtree. But this kind of algorithm will discard the benefits brought by alpha and beta values. PVSplit is to run the first branch first in order to get the bounds (alpha and beta values) returned by the first branch, then we run the remaining branches in parallel.

3.3 Implementation

We choose to implement PVSplit algorithm. We planned to use OpenMP in our proposal, but when we implement the algorithm, we found that pthread and its flexibility make it better fit this project.

3.3.1 *First Pthread Implementation.*

- Open a thread for each branch of some subtree.
- Run the search recursively.
- Wait then to end and proceed to other subtree.

We found that this implementation is mostly slower than the serial implementation with alpha-beta pruning. We found the following reasons:

- Too many threads: try to invoke N threads at once to avoid this, but still slow.
- Bad load balance: wait other threads to complete. This is inevitable for PVSplit.
- The alpha and beta values are not updated when parallized to threads. We think this is the most serious reason.

3.3.2 *Final Implementation Using Glib thread pool.*

The final solution for the issues is to use Glib thread pool. Glib is a famous open source C library from GNOME community. It provides a lot of convenience tools like the STL in C++. Thread pool solves the problems by:

- Too many threads: we can limit a max number of thread in thread pool.
- Bad load balance: we can lower a little waiting time cause by the threads of same branch, since we don't run them N by N.
- Update global alpha and beta values of some subtreee when a thread is finished. Carefully implemented using mutex library provided by Glib.

4 EXPERIMENTAL METHODOLOGY

4.1 Test Environment

- OS: ArchLinux
- CPU: intel i7-6700, 4 core with hyperthreading on
- RAM: 16GB

4.2 Chessboard Implementation

Our chessboard is built with bitmap. We enum integer 1-15 as different kind of piece, and put these integers into a 64-integer-array, which represented our board. Different integer(peice) would be mapped to its own defined method that help to determine whether a move is avaliable.

```
enum EPieceCode
{
    epc_empty   = ept_pnil,
    epc_wpawn   = ept_wpawn,
    // may be used as off the board blocker in mailbox
    epc_woff    = ept_bpawn,
    epc_wknight = ept_knight,
    epc_wbishop = ept_bishop,
    epc_wrook   = ept_rook,
    epc_wqueen  = ept_queen,
    epc_wking   = ept_king,

   // color code, may used as off the board blocker in mailbox
    epc_blacky  = 8,
    epc_boff    = ept_wpawn  + epc_blacky,
    epc_bpawn   = ept_bpawn  + epc_blacky,
    epc_bknight = ept_knight + epc_blacky,
    epc_bbishop = ept_bishop + epc_blacky,
    epc_brook   = ept_rook   + epc_blacky,
    epc_bqueen  = ept_queen  + epc_blacky,
    epc_bking   = ept_king   + epc_blacky,
};

class ChessBoard {

    public:
        int boardMap[64];
        map<int,Piece*> pieceMap;
}
```

For the evaluation for the algorithm to search, we have different scoreboards for different pieces. We add up the score*weight of each piece on the board.

For detail implementation please refer to our original code. Our code can be found on this Github repo: https://github.com/spencerwuwu/PP_project/tree/master/src

4.3 Testing method

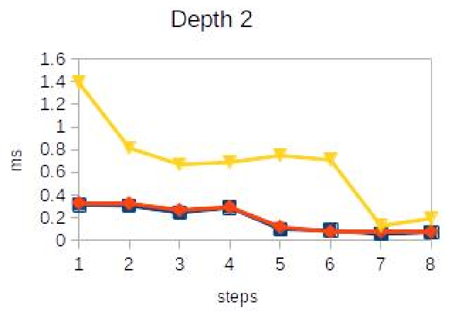
Since that we are can not guarantee to have the same result by searching the same board, which we will explain the reason for it later, and that alpha-beta may cut up in the middle of the search, we compare our data by plotting the line chart of the time of each step for different serach methods and different depths. So we have different graph for different search depth, each graph compares the time to search for each round by different search algorithm.

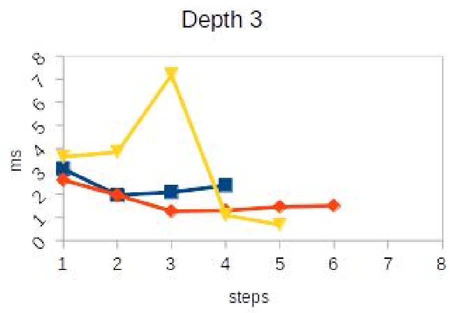
To test the scalability of our algorithm, we compare the performance of different number of thread created in depth 5 and depth 6.

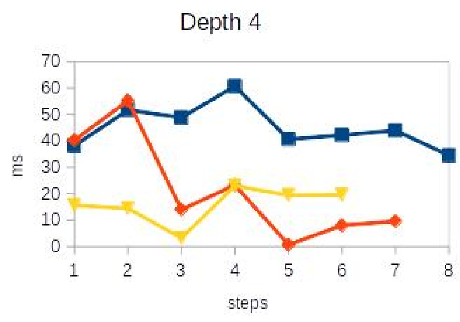
5 EXPERIMENT RESULTS

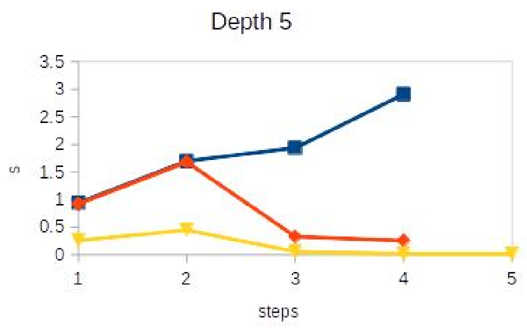
The y-axis is the time spent by each step. The x-axis is the step number. There are 5 graph representing different search depth of tests.

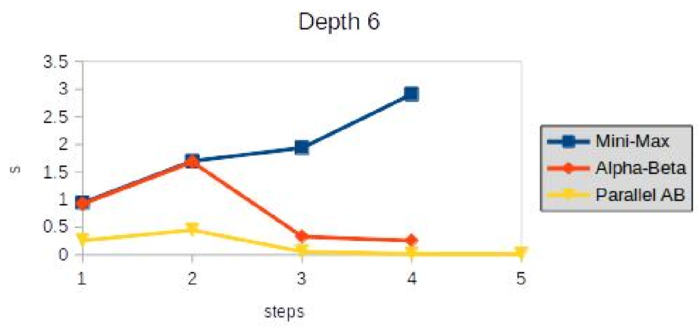
5.1 Graph











5.2 Scalability

In this table, we can see the scalability of our program. The number in the table is add all stop (total time) in our tests.

	2 cores	4 cores	8 cores	16 cores
Depth 5	0.764s	0.572s	0.357s	1.103s
Depth 6	21.784s	12.240s	10.207s	18.643s

6 CONCLUSION

From the results, we can see although we can get speedup from parallization, but the scalability isn't very good. The issue is mainly come from idle cores (load balancing). When the game goes to further steps, the possible moves of some subtree will get less. That is, there will be more idle threads. One possible solution is EPVS (enhanced), that is let idle cores to help busy cores to search some subtrees. But this seems not to bring large improvements. To really solve this issue, we have to implement the DTS algorithm.

Another issue of this algorithm is that when the trees were cut too early, the difference between parallel and sequential version is not so obvious.

7 RELATED WORKS

There are lots of research on Chess programming. We mentioned some of them in the third section in this report. Also, there are some interesting research of GPU-accelerated alpha pruning.

8 FUTURE WORKS

If we want to improve our Chess program, here are some possible improvements:

- Implementing the Dynamic Tree Splitting, which is now the standard.
- Using GPU algorithm: see how GPU can accelerate it.
- Better evaluation. The score evaluation in this project is a preliminary heuristic method, which is not so smart. We can try some method like TD-Learning or other common methods.

9 REFERENCES

- Parallelizing a Simple Chess Program: http://iacoma.cs.uiuc.edu/~greskamp/pdfs/412.pdf
- Parallel Game Tree Search http://www.iis.sinica.edu.tw/~tshsu/tcg/2013/slides/slide11.pdf
- Parallel Minimax Tree Searching on GPU: http://olab.is.s.u-tokyo.ac.jp/~kamil.rocki/rocki_ppam09.pdf
- Parallel Alpha-Beta Search on Shared Memory Multiprocessors: http://www.top-5000.nl/ps/Parallel%20Alpha-Beta%20Search%20on%20Shared%20Memory%20Multiprocessors.pdf
- Chess Programming Wiki: https://chessprogramming.wikispaces.com/