Threaded Thinker : Chess Engine Optimizations

Prof. Nisha Auti[1], Gaurav Lonsane[2], Mitesh Patil[3], Ritesh Dimble[4], Chintan Randhe[5],Poonam Lakhe[6]
*[*1]Professor Department of Computer Engineering, JSPM Narhe Technical Campus, Pune, Maharashtra, India*

[,2,3,4,5,6]Student, Department of Computer Engineering, JSPM Narhe Technical Campus, Pune, Maharashtra, India*

ABSTRACT
Chess engine was always been a source of fascination and will going to be, as there is still room for lot of improvements, because of inherited complexity of chess and its variants so we proposed novel methods for optimizing the performance of chess engines, with a focus on improving search depth by leveraging hardware resources and enhancement done to the algorithms to improve calculation and accuracy that is to reduce the number of position evaluated, increase depth and improve the overall speed of decision making.

Keywords - Chess, Engine, Parallel Search, Game Cache.

I. INTRODUCTION

Chess engines are complex systems designed to evaluate and generate chess moves. The objective is to create an engine that can beat human players or at least provide a significant challenge. In the field of artificial intelligence, chess engines have been a benchmark for computational power and strategic analysis. Modern chess engines heavily rely on optimizations that involves improving their efficiency, speed and the quality of their decision that is to refine its two major key components, the first is search function and second is evaluation function. Evaluation function is the "brain" of the chess engine. It assigns a numerical value to a given position, which represents the perceived advantage for one player over the other. The function is often designed with knowledge of basic chess principles such as material balance, piece activity, king safety, pawn structure and control of the center. The optimization of the evaluation function can involve tweaking these parameters or incorporating more complex chess concept. Chess engine often leverage hardware resources to speed up evaluations of multiple position or nodes at the same time.

Second key component that can be refined is search function that is "eye" of the chess engine. It explores possible future positions, branching out from current position like a tree. Search function uses the evaluation function to assess the positions it explores. The most common search algorithm, often used in chess engines is the min-max algorithm, often optimized with techniques such as Alpha-Beta Pruning. The optimization of the search function involves making it faster and more selective, so it can explore deeper into the game tree in a given amount of time.

This paper describes optimization techniques used by strong chess engines that can compute millions of possible positions guided by search algorithms and evaluated dynamically using neural network, simultaneously by leveraging hardware resources that supports vectorization.

II. LITRATURE SURVEY

Paper Name: Accelerating Monte Carlo Tree Search Using Speculative Tree Traversal

Author: Juhwan Kim, Byeongmin Kang, Hyugmin cho

Abstract : Monte Carlo Tree Search (MCTS) algorithms show outstanding strengths in decision-making problems such as the game of Go. However, MCTS requires significant computing loads to evaluate many nodes in the decision tree to make a good decision. Parallelizing MCTS node evaluations is challenging because MCTS is a sequential process that each round of tree traversal depends on the previous node evaluations. In this work, we present SpecMCTS, a new approach for accelerating MCTS by speculatively traversing the search tree. Many MCTS applications, such as AlphaGo Zero, use a deep neural network (DNN) model to evaluate the tree nodes during the search. SpecMCTS uses a pair of DNN models, the speculation model and the main model.

2.**Paper Name: A Parallel Algorithm for Game Tree Search Using GPGPU**

Author: Liang Li, Hong Liu, Wei Li and Hao Wang

Abstract : Game tree search is a classical problem in the field of game theory and artificial intelligence. Fast game tree search algorithm is critical for computer games asking for real-time responses. In this paper, we focus on how to leverage massive parallelism capabilities of GPU to accelerate the speed of game tree search algorithms and propose a concise and general parallel game tree search algorithm on GPU. The performance model of our algorithm is presented and analyzed theoretically.

3.**Paper Name: Metamorphic Testing of an Artifically Intelligent Chess Game**

Author: Aisha Lia, Muddassar Azam Sindhu and Ghanzanfar Farooq Siddiqui

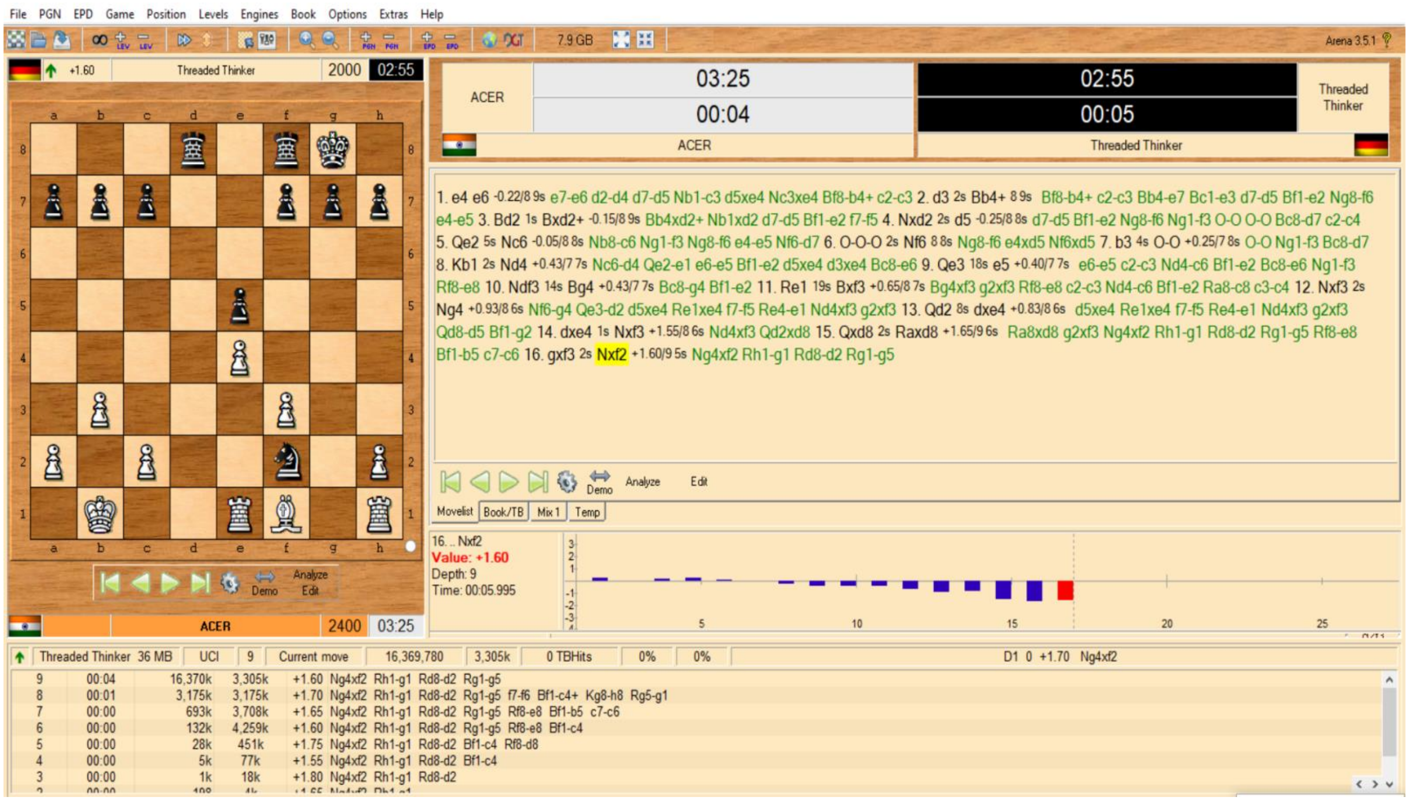
Abstract : Artificially intelligent (AI) game software incorporates different algorithms to generate intelligent human-like responses to the users playing them. Testing AI game software poses great difficulty because of the complex possibilities that can result at a given point and analysis of said possibilities is a tedious task. Also during software development there are resource constraints due to which testing targets specific parts of the software. An AI game of Chess takes into consideration a large amount of possible outcomes at any given point before deciding a move. Therefore, testing it in its entirety is impractical. In this paper we propose a metamorphic testing approach for testing an AI Chess i.e. a Chess engine's algorithm of determining and pruning out possible outcomes and ultimately deciding on a final outcome. For validating our approach, we have done error seeding on an open source Chess engine and tested it through our approach. The results for our proposed approach for testing an AI Chess game through metamorphic relations show that it is successful in revealing 71% of the total seeded faults.

4.**Paper Name: Comparsion Training for Computer Chineese Chess**

Author: Jr-Chang Chen

Abstract: This paper describes the application of a modified comparison training for automatic feature weight tuning. The final objective is to improve the evaluation functions used in Chinese chess programs. First, we apply n-tuple networks to extract features. N-tuple networks require very little expert knowledge through its large numbers of features, while simultaneously allowing easy access. Second, we propose a modified comparison training into which tapered eval is incorporated. Experiments show that with the same features and the same Chinese chess program, the automatically tuned feature weights achieved a win rate of 86.58% against the hand-tuned features.

III. SYSTEM ARCHITECTURE



1. e4 e6 -0.22/8 9s e7-e6 d2-d4 d7-d5 Nb1-c3 d5xe4 Nc3xe4 Bf8-b4+ c2-c3 2. d3 2s Bb4+ 8 9s Bf8-b4+ c2-c3 Bb4-e7 Bc1-e3 d7-d5 Bf1-e2 Ng8-f6 e4-e5 3. Bd2 1s Bxd2+ -0.15/8 9s Bb4xd2+ Nb1xd2 d7-d5 Bf1-e2 f7-f5 4. Nxd2 2s d5 -0.25/8 8s d7-d5 Bf1-e2 Ng8-f6 Ng1-f3 O-O O-O Bc8-d7 c2-c4 5. Qe2 5s Nc6 -0.05/8 8s Nb8-c6 Ng1-f3 Ng8-f6 e4-e5 Nf6-d7 6. O-O-O 2s Nf6 8 8s Ng8-f6 e4xd5 Nf6xd5 7. b3 4s O-O +0.25/7 8s O-O Ng1-f3 Bc8-d7 8. Kb1 2s Nd4 +0.43/7 7s Nc6-d4 Qe2-e1 e6-e5 Bf1-e2 d5xe4 d3xe4 Bc8-e6 9. Qe3 18s e5 +0.40/7 7s e6-e5 c2-c3 Nd4-c6 Bf1-e2 Bc8-e6 Ng1-f3 Rf8-e8 10. Ndf3 14s Bg4 +0.43/7 7s Bc8-g4 Bf1-e2 11. Re1 19s Bxf3 +0.65/8 7s Bg4xf3 g2xf3 Rf8-e8 c2-c3 Nd4-c6 Bf1-e2 Ra8-c8 c3-c4 12. Nxf3 2s Ng4 +0.93/8 6s Nf6-g4 Qe3-d2 d5xe4 Re1xe4 f7-f5 Re4-e1 Nd4xf3 g2xf3 13. Qd2 8s dxe4 +0.83/8 6s d5xe4 Re1xe4 f7-f5 Re4-e1 Nd4xf3 g2xf3 Qd8-d5 Bf1-g2 14. dxe4 1s Nxf3 +1.55/8 6s Nd4xf3 Qd2xd8 15. Qxd8 2s Raxd8 +1.65/9 6s Ra8xd8 g2xf3 Ng4xf2 Rh1-g1 Rd8-d2 Rg1-g5 Rf8-e8 Bf1-b5 c7-c6 16. gxf3 2s Nxf2 +1.60/9 5s Ng4xf2 Rh1-g1 Rd8-d2 Rg1-g5

16. .. Nxf2
Value: +1.60
Depth: 9
Time: 00:05.995

```
Threaded Thinker 36 MB | UCI | 9 | Current move | 16,369,780 | 3,305k | 0 TBHits | 0% | 0% | D1 0 +1.70 Ng4xf2
 9   00:04   16,370k  3,305k   +1.60 Ng4xf2 Rh1-g1 Rd8-d2 Rg1-g5
 8   00:01    3,175k  3,175k   +1.70 Ng4xf2 Rh1-g1 Rd8-d2 Rg1-g5 f7-f6 Bf1-c4+ Kg8-h8 Rg5-g1
 7   00:00      693k  3,708k   +1.65 Ng4xf2 Rh1-g1 Rd8-d2 Rg1-g5 Rf8-e8 Bf1-b5 c7-c6
 6   00:00      132k  4,259k   +1.60 Ng4xf2 Rh1-g1 Rd8-d2 Rg1-g5 Rf8-e8 Bf1-c4
 5   00:00       28k    451k   +1.75 Ng4xf2 Rh1-g1 Rd8-d2 Bf1-c4 Rf8-d8
 4   00:00        5k     77k   +1.55 Ng4xf2 Rh1-g1 Rd8-d2 Bf1-c4
 3   00:00        1k     18k   +1.80 Ng4xf2 Rh1-g1 Rd8-d2
```

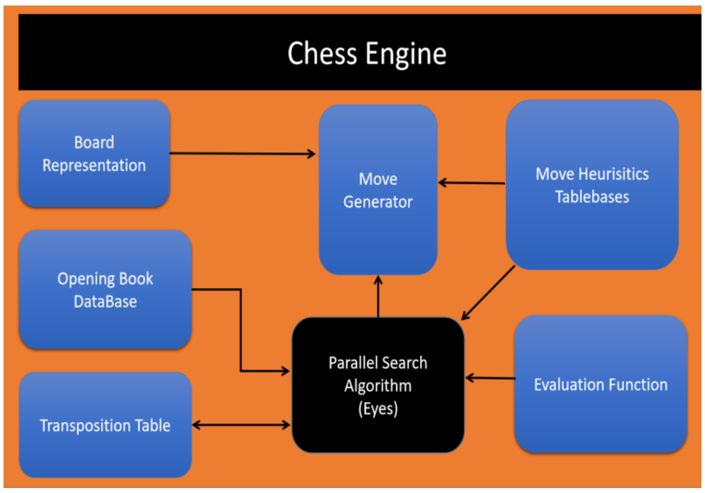
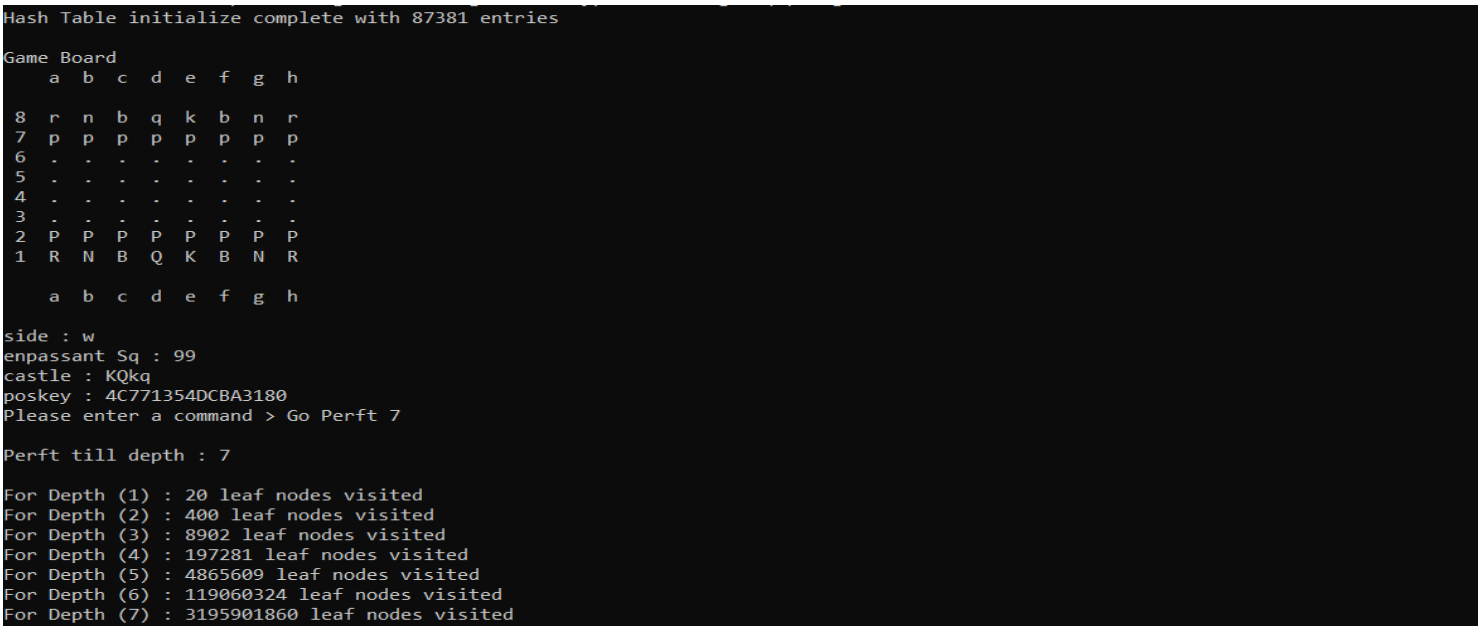


Fig 1. System Architecture

IV. Result and Outcomes



```
Hash Table initialize complete with 87381 entries

Game Board
    a  b  c  d  e  f  g  h

 8  r  n  b  q  k  b  n  r
 7  p  p  p  p  p  p  p  p
 6  .  .  .  .  .  .  .  .
 5  .  .  .  .  .  .  .  .
 4  .  .  .  .  .  .  .  .
 3  .  .  .  .  .  .  .  .
 2  P  P  P  P  P  P  P  P
 1  R  N  B  Q  K  B  N  R

    a  b  c  d  e  f  g  h

side : w
enpassant Sq : 99
castle : KQkq
poskey : 4C771354DCBA3180
Please enter a command > Go Perft 7

Perft till depth : 7

For Depth (1) : 20 leaf nodes visited
For Depth (2) : 400 leaf nodes visited
For Depth (3) : 8902 leaf nodes visited
For Depth (4) : 197281 leaf nodes visited
For Depth (5) : 4865609 leaf nodes visited
For Depth (6) : 119060324 leaf nodes visited
For Depth (7) : 3195901860 leaf nodes visited
```

Fig 2. Move Generator

***** The perft (performance test) function is often used in chess engine to debug move generation. It calculates the total number of legal moves in a position for a given depth.

Outcome : Search algorithm predicted best move with the help of other components such transposition table, move heuristic tablebase, opening and endgame table.

Results : The Move Generator generates correct number of total legal moves to given depth as per mentioned on wikipedia.

The alpha beta pruning can able to prune nodes and irrelevant branches based on information gather from previous search that contributes towards wining.

V. MOTIVATION

The motivation for this project came while watching our chess engine playing with other strong chess engines that already uses some of the optimization techniques to have better decision making process and therefore to incorporate the same with some other techniques in this project, we will able to leverage the performance too.

VI. OBJECTIVE OF THE SYSTEM

* To enable the student understand chess

* To enable the student to improve their problem solving skill

* To enable the student to improve analytic thinking

* To spark interest of student in other field that are interconnected to system

VII. SYSTEM REQUIRMENT

A. Software Requirement

1.	Operating system :	Windows 8
2.	Coding Language :	C/C++
3.	IDE :	Visual Studio

B. Hardware Requirement

4.	System:	Intel Core i9 (8 cores)
5.	SSD :	526 GB.
6.	Ram :	8GB

VIII. METHODOLOGY

Alpha-beta Search

The alpha-beta Pruning algorithm is a search algorithm used in computer chess for reducing the number of nodes that are evaluated by the min-max algorithm. It's an optimization technique that is used in conjunction with the min-max algorithm to eliminate portions of the game tree that are irrelevant to the best move selection. The idea is that if you have seen enough of a subtree to know its final value is worse than a previously examined subtree, you "prune" it and don't bother looking at its other nodes.

Parallelizing the alpha-beta algorithm to take advantage of multi-core processors has been a significant area of research in computer chess.

Parallelization of Alpha-beta Search

1) Principle Variation Splitting :

In this method, the search tree is split at nodes along the principle variation i.e the sequence of moves considered best by engine. This provide a reasonable speedup but can be hard to implement correctly and efficiently. PVS can be a part of parallelization strategies used in game tree searching algorithms like chess. In this approach, one processor starts searching the principle variation while the remaining processors are idle. As soon as the first processor reaches a point where it can split the work("split point"), it sends parts of the search tree to the idle processors. This splitting continues recursively untill all processors are busy or the search tree is fully explored.

2) Young Brothers Wait Concept(YBWC) :

YBWC is a method for parallelizing game tree search algorithms like alpha beta pruning in chess. It balances workload across multiple processors cores and minimizes synchronization needs. It works when a thread reaches a node and begins searching the first child. If there are additional children ("young brothers"), they become available for other threads to search. The original thread waits untill another thread finishes searching a sibling before continuing. YBWC offers good load balancing by keeping all threads busy, requires minimal synchronization and

Maintains the properties of alpha beta pruning by completing the search of first child before considering its siblings.

3) Lazy SMP(Symmetric Multi-Processing) :

It is a simple method that parallelizes game tree searches by running multiple independent searches on the same position using multiple threads.It works when threads performs separate searches without direct communication, except for sharing transposition table (a cache of previously searched positions). Each thread may search at different depths and follow different paths. Lazy SMP is easy to implement and can provide good speedup with minimal overhead. The absence of complex synchronization communication between threads reduces potential bottlenecks and allows for efficient utilization of multi-core processors.

NNUE (Efficiently Updatable Neural Network)

It is a specific type of neural network architecture used for evaluating chess positions. It was introduced to computer chess world by the shogi engine and later adopted by stockfish. The NNUE architecture is designed to be efficient to evaluate and update, making it suitable for use in a game tree search algorithm like alpha-beta pruning. NNUE is designed to run efficiently on CPUs, not requiring a GPU which makes it more accessible for typical users who may not have high-end GPU hardware.

IX. CONCLUSION

Chess engine optimization have revolutionized the game of chess by significantly improving engine's performance and strength. New Advancements in search algorithms, evaluation functions, parallel processing, machine learning and hardware optimizations techniques have collectively pushed the existed boundaries of chess engine capabilities.

X. REFERENCES

[1] X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang, "Deep learning for realtime atari game play using offline monte-carlo tree search planning," in NIPS, 2014.

[2] D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol.

529, no. 7587, pp. 484–489, 2016.

[3] "Mastering the game of Go without human knowledge," Nature, vol. 550, no. 7676, pp. 354–359

[4] Y. Tian et al., "ELF OpenGo: an analysis and open reimplementation of AlphaZero," in ICML, 2019.

[5] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in Euro. Conf. Mach. Learn.

[6] G. M.-B. Chaslot, M. H. Winands, and H. J. v. d. Herik, "Parallel Monte-Carlo Tree Search," in International Conference on Computers and Games, 2008.

[7] C. B. Browne et al., "A survey of Monte-Carlo tree search methods," IEEE Trans. Comput. Intell. AI

in Games, vol. 4, no. 1, pp. 1–43, 2012.

[8] A. Liu, J. Chen, M. Yu, Y. Zhai, X. Zhou, and J. Liu, "Watch the Unobserved: A Simple Approach to Parallelizing Monte Carlo Tree Search ," in ICLR, 2020.