Exploring Parallel MCTS on Chess Game

曾正豪 0716325
CS NYCU
Hsinchu, Taiwan

張宸愷 0710018
EECSHP NYCU
Hsinchu, Taiwan

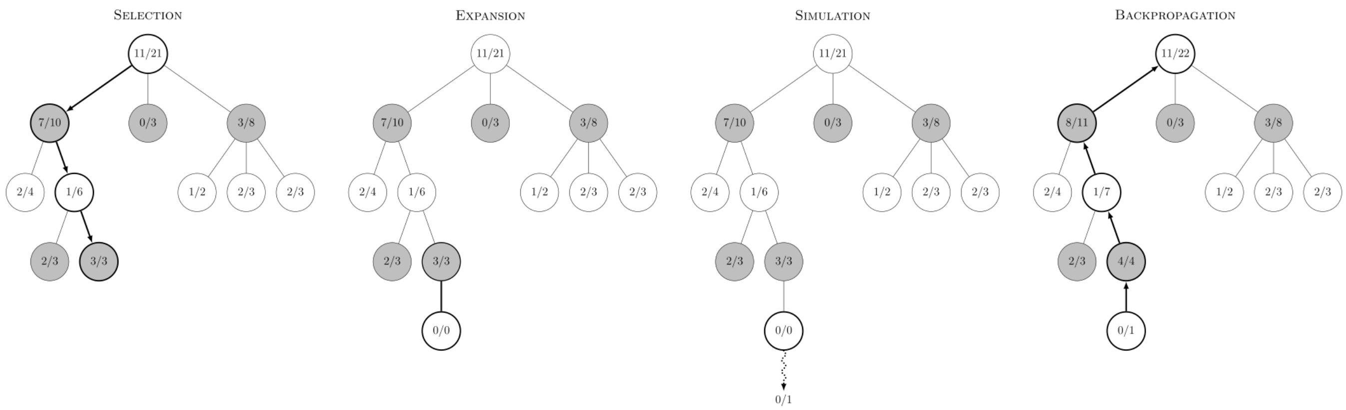

Figure 1: Illustration for a single step of MCTS

ABSTRACT

A project proposal for the course 'Parallel Programming Fall 2021'. We decided to explore the parallelization of Monte Carlo Tree Search using the techniques and knowledge we have learned in this course. We will use quantitative benchmarks to compare different approaches to solve this kind of parallelization.

KEYWORDS

MCTS, parallel programming, Pthreads, CUDA, Chess

ACM Reference Format:
曾正豪 0716325 and 張宸愷 0710018. 2021. Exploring Parallel MCTS on Chess Game. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

'Monte Carlo Tree Search', abbreviated in this text from now on as MCTS, is a probability sampling based tree search method for many applications. One of the most famous application of MCTS is AlphaGO [3]. It uses MCTS with 2 other neural networks to play Go. An early version of AlphaGo was tested on hardware with various numbers of CPUs and GPUs, running in asynchronous or distributed mode. It was tested with search threads from 12 to 64, number of CPUs from 48 to 1920, and number of GPUs from 1 to 280. And in 2016, it changed to use TPUs (tensor processing units) as its computing unit. In recent years, it keeps beating many go players. Overall, MCTS is an algorithm that can be highly parallelized because of the high number of simulations. Hence, we decided to use MCTS as the topic of our final project.

2 STATEMENT OF PROBLEM

One of our teammates had taken the course AI capstone, and during that course he had been doing a final project about playing a 3D version of connect-4 using upper confidence bound MCTS, i.e. UCB-MCTS. He noticed that when he uses root parallelization, the performance of the MCTS decreases dramatically. The multithreaded version with 8 threads didn't even manage to beat the single threaded one. Thus, we want to explore further on why it didn' t perform better and try to improve the multithreaded performance. The teammate's guess is that

3 PROPOSED APPROACHES

3.1 Overall design

Figure 2 is our system design Chess game module: Deal with the game state of Chess, once it receives a legal move, it will change its game state and return it. arbiter and round-robin tournament maker: It will coordinate the game player and the game module. It checks if one move is legal and whether the game ends. 3 MCTS versions: To calculate the best move next. It will receive a game state and return its best move. The difference between these 3 versions is their parallel approach. Additional: Maybe we will compete with other Chess engines to see the performance if we have enough time. The performance of each method will depend on the games won, and the total amount of expanded nodes.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/1122445.1122456

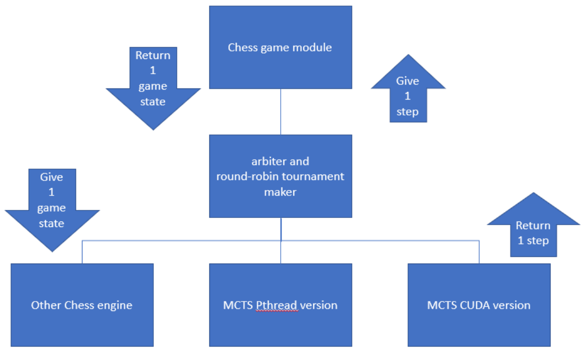

Figure 2: Our system diagram

3.2 parallelization details

There are four ways to parallelize traditional UCB-MCTS mentioned in [1], and illustrated in figure 3. Leaf parallelization, root parallelization, tree parallelization with global mutex, and tree parallelization with local mutex.

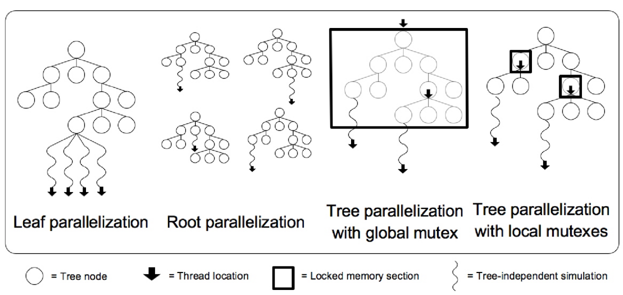

Figure 3: Ways of parallelizing MCTS

4 LANGUAGE SELECTION

We will use C++ as our main programming language, together with Pthreads and CUDA, since we want to compare the performance of CPU and GPU on parallel programs.

5 RELATED WORK

A pretty detailed work [2]. Different methods for enhancing the capability of MCTS on board games were explored in [4]. [1][4] also outlined some of the parallelization techniques for MCTS, root parallelization, leaf parallelization, and tree parallelization, etc.

6 EXPECTED RESULTS

When using tree parallelization, there will be two methods. One is we use a global mutex to protect all the nodes in the tree. The other is to give each node a mutex. The former will be much slower because many threads are contending for a single lock, while the latter will use significantly more memory because each node has a lock, but the performance will be better.
The

7 TIMETABLE

Below is the rough due date for the individual components of our project.

- 10/27 proposal
- 11/05 chess engine wrapper, and game module
- 11/20 Pthread version
- 12/10 CUDA version
- 12/17 Performance analysis
- 12/31 Finalize report and source code

8 APPENDICES

REFERENCES

[1] Guillaume M.J-B. Chaslot, Mark H.M. Winands, and H. Jaap van den Herik. 2008. Parallel Monte-Carlo Tree Search.
[2] Anji Liu, Yitao Liang, Ji Liu, Guy Van den Broeck, and Jianshu Chen. 2020. On Effective Parallelization of Monte Carlo Tree Search. arXiv:2006.08785 [cs.LG]
[3] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529 (01 2016), 484–489. https://doi.org/10.1038/nature16961
[4] Martin Weigel. 2017. Monte Carlo methods for massively parallel computers. arXiv:1709.04394 [physics.comp-ph]

A ONLINE RESOURCES

(1) Monte Carlo tree search (https://en.wikipedia.org/wiki/Monte_Carlo_tree_search)