International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 15 (2017) pp. 4854-4857

A Survey on Issues at the GPGPU Acceleration of the Monte Carlo Tree Search

SeongKi Kim[*, 1]

Sangmyung University, 20, Hongjimun 2-gil, Jongno-gu, Seoul, Republic of Korea.
*(*Corresponding author)*

[1]*Orcid: 0000-0002-2664-3632*

Abstract

The Monte Carlo Tree Search (MCTS) has been widely used in the game, physics and mathematics fields, and the quality or winning ratio of suggested decision by the algorithm heavily depends on the numbers of executed simulations and iterations. More simulations or iterations make the MCTS evaluate various nodes and make the evaluations correct. To increase the numbers of them, we can use a high-performance CPU or run the algorithm for a long time. But, both require additional resources such as the cost and the time, and the GPGPU can significantly increase the number of executed simulations while minimizing resource usage, thus increase the quality within limited resources. But, it is difficult to parallelize the MCTS through the GPGPU due to some issues. This paper classifies the suggested algorithms for the GPGPU acceleration to the MCTS and describes their issues for the future researches.

Keywords: MCTS, GPGPU, Parallelization, UCT, UCB

INTRODUCTION

The goal of MCTS (Monte Carlo Tree Search) is to find an optimal solution. For this purpose, it searches the tree through the best-first method, expands the search tree according to the latest results, simulates a state up to an ending state, then updates the results up to the root node. This algorithm can be used in the following area: The first area is that the deterministic methods are impossible, like areas of uncertainty in the inputs. In the game scenario, the artificial intelligence can determine the best action of a monster only after the character's action, and cannot determine it before it. The second area is that the search tree is used for finding solutions. As an example in the game scenario, the possible actions from a state can be built to a search tree, and the tree can be used for choosing the best action. The third area is that the state space is too large to explore all states. In the Game Go, the state-space size is approximately equal to $3.7 * 10^{79}$ for the 13x13 board size [1]. The fourth area is that finding a good heuristic, evaluation function, to guide the search is not applicable.

Game and mathematics include this area, and the MCTS is widely used for them [2]. Especially, board games such as Hex [3], Havannah [4], Real-time video games such as Ms Pac-Man [5], as well as Go such as AlphaGo [6] use this algorithm. Besides the games, mathematics uses this algorithm for a multivariate polynomial problem [7].

However, the MCTS has the characteristic that the quality of a decision is heavily dependent on the numbers of random simulations and iterations, and the GPGPU can significantly increase the numbers within a given time and memory constraints. With more simulations, the evaluation result can become more correct, and more actions can be checked with more iterations. Because of this characteristic, many researchers have tried to use the GPGPU to increase the quality of a decision [8][9][10]. But, when parallelizing the MCTS through the GPGPU, the researchers will meet many problems and issues that this paper describes in more detail.

For the future researches and developments, this paper describes the details of the algorithm, problems and issues in the GPGPU parallelization. The remainder of this paper is organized as follows. Section 2 describes the MCTS. Section 3 describes the issues when accelerating it through the GPGPU, and our conclusion is provided in section 4.

MONTE CARLO TREE SEARCH (MCTS)

A search tree (state tree) is used at the MCTS. At the search tree, each node represents a state that can be chosen, and each edge represents an action that can be chosen. Fig. 1 shows an example of the search tree.

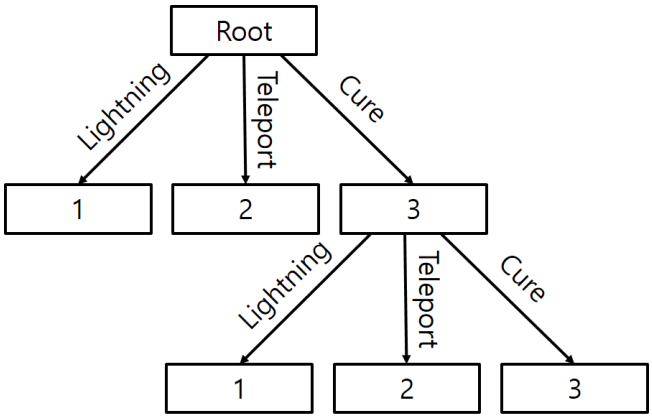

Figure 1. Example of the search tree

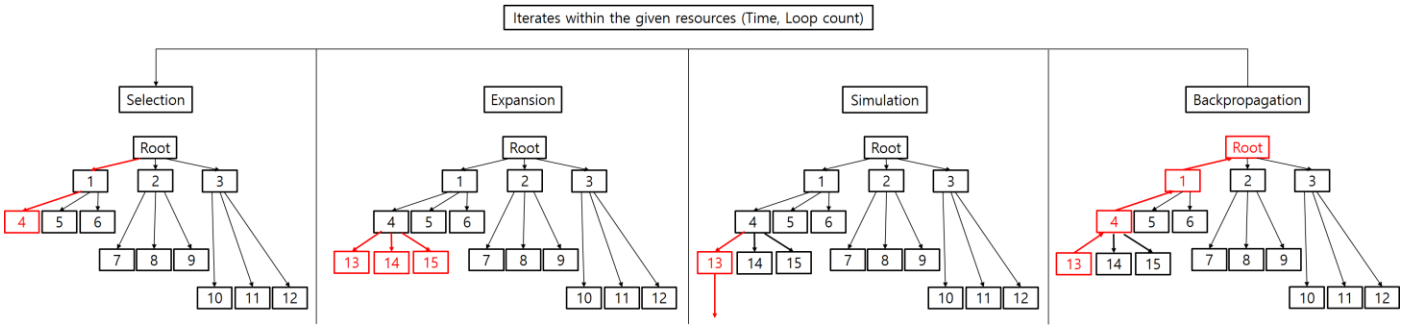

Figure 2: Overview of MCTS

When a wizard can spell one of the magic such as lightning, teleport, and cure, its search tree is created in Fig. 1. At the root state, the wizard can spell one of these magic, then the state is changed into state 1, 2 or 3 depending on the game. At the state 3, the wizard can also spell another magic, and the state is changed again.

Based on this search tree, the MCTS searches the best action at the current state. Fig. 2 illustrates the overview of MCTS, which consists of 4 different processes that are described as follows:

- **Selection**: While the state is found in the tree, the next action is chosen according to the statistics stored. When selecting a node, the following Eq. 1 is usually used.

$$UCB(i) = \frac{w_i}{n_i} + c\sqrt{\frac{\ln s}{n_i}} \quad (1)$$

  In Eq. 1, w_i, n_i, s and c are the number of wins computed by playouts at child i, the number of visits to child i, the number of visits to a parent of child i, and constant, respectively. The first term is an exploitation of the most promising node, and the second term is an exploration of the unexplored node. When firstly choosing a node without any stored data (w_i, n_i, and s), a node is usually selected randomly.

  Eq. 1 is widely used, but it can be replaced with another equation. The MCTS with Eq. 1 is called UCT (Upper Confidence Bounds for Trees).

- **Expansion**: MCTS follows the best child until a terminal node, and the next possible children are added into the state tree. If all of the possible children are added, the best child is selected again.

- **Simulation**: MCTS evaluates the chosen child until the end of a game. User's and computer's choices are randomly determined.

- **Backpropagation**: After reaching the end of the simulated game, the numbers of node's wins and visits is updated up to the root node.

The details of algorithm can be found in [2]. Because the results of the algorithm is closely related with the number of iterations, many researchers have tried to increase it through a parallelization. Next subsections categorize, and describe them.

Leaf parallelization

The easiest GPGPU parallelization of MCTS is the leaf parallelization that a node is evaluated by many GPU threads. Fig. 3 illustrates the overview of leaf parallelization.

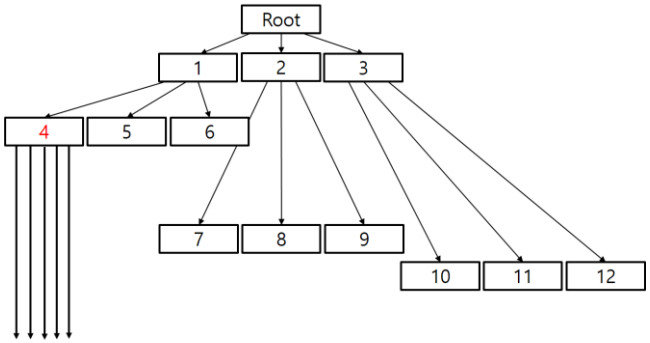

Figure 3. Overview of leaf parallelization

At the leaf parallelization in Fig. 3, many threads simulate the node 4. Because of the multiple simulations, the results are more correct. The leaf parallelization places emphasis on the exploitation.

Root parallelization

At the root parallelization, each GPU thread makes its tree and expands it. Fig. 4 presents the overview of root parallelization.

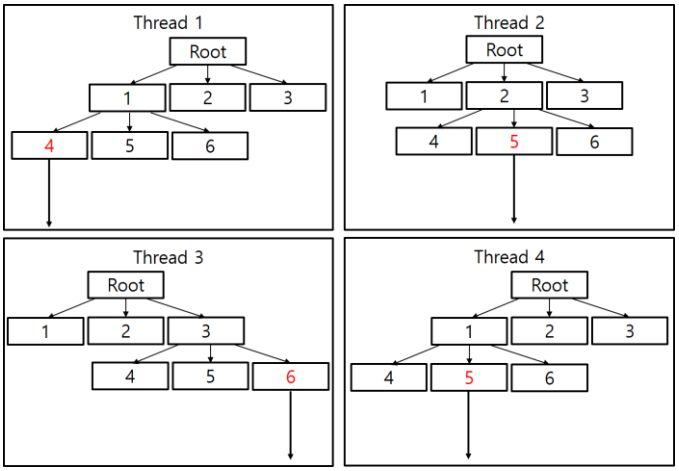

Figure 4. Overview of root parallelization

At the root parallelization in Fig. 4, each thread selects, expands, simulates, and backpropagates the simulation result up to the root node. The root parallelization places emphasis on the exploration.

Tree parallelization

At the tree parallelization, many GPU threads update a single tree at different nodes. Fig. 5 shows the overview of tree parallelization.

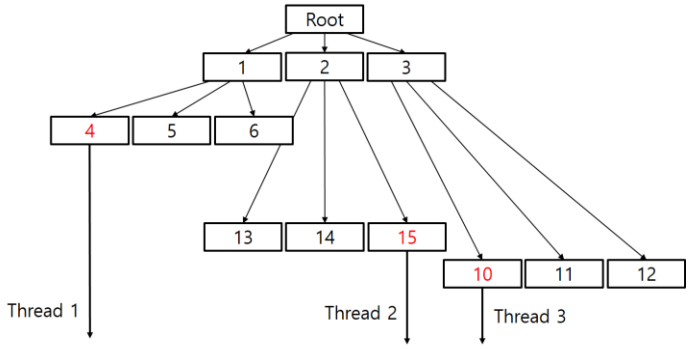

Figure 5. Overview of tree parallelization

At the tree parallelization in Fig. 5, many threads select, expand, simulate and backpropagate a single tree.

ISSUES AT THE GPGPU ACCELERATION

This Section describes the issues when implementing the MCTS with the GPGPU acceleration.

Sequential algorithm

UCB calculation at the selection process requires the numbers of visits and wins that will be updated only after the previous backpropagation. So, the next iteration can start only after the previous iteration. Thus each iteration cannot simultaneously run. The dependency makes the algorithm sequential, which makes the parallelization more difficult.

Locking problem

In the tree parallelization, the threads should update the same variable of a node during the backpropagation stage (E.g. root node). So, there can be a locking problem. As one of the solutions to this problem, minimum lock approaches [10] are suggested, but it will ignore some evaluations. As another solution, local-lock approach [10] is available, but it still needs a lock.

Load Balancing

The GPU threads should finish all of their works before returning their controls to the CPU, and some threads can have more works to do, and other threads can have fewer works. In this case, the threads with fewer jobs should wait for the threads with more works. Particularly in the leaf parallelization, some threads can finish their evaluation earlier, but they should wait until the other threads. To reduce this unbalanced loads, the similar amounts of works should be given to each thread so that all threads can finish their works at the similar times.

Node Simulation

Only a single thread evaluates the node in the root parallelization. So, the evaluation can be incorrect. As time goes by, the single thread will be evaluated more and more. Thus the evaluation will be more correct. But, it needs more times.

Unnecessary Computations

If 16 threads are evaluating a node, and 8 (faster) finished games are all losses, it will be highly probable that most games will lead to a loss. But, the GPU has no ability to terminate the working threads. Thus eight lost threads should still wait for eight remaining threads.

Local Minimum

In the leaf parallelization, a single node is evaluated by many threads. If the evaluated node is a wrong node, the leaf parallelization will waste the GPU's computational power that can be utilized for the other promising nodes.

CONCLUSIONS

The MCTS can be used for many fields such as games, mathematics, and physics. But, it has the characteristic that the quality heavily depends on the resources such as the cost and the time. To reduce the required resources while maintaining a high-quality, the GPGPU can be used. However, the GPGPU acceleration of the algorithm has a lot of difficulties and issues. This paper describes the MCTS for the beginners and classifies the studies on the parallelizations of MCTS. This paper also introduces the difficulties and issues at the GPGPU acceleration of the MCTS for the future researches.

ACKNOWLEDGEMENT

This research was supported by the NRF in Korea (2015R1C1A1A01051839).

REFERENCES

[1] Rocki K., Suda R.: Massively Parallel Monte Carlo Tree Search, Proceedings of the 9th International Meeting High Performance Computing for Computational Science, 2010.

[2] C. B. Browne et al., A Survey of Monte Carlo Tree Search Methods, in IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no. 1, pp. 1-43, March 2012.

[3] Broderick Arneson, Ryan B. Hayward, Philip Henderson, "Monte Carlo Tree Search in Hex," IEEE Trans. Comp. Intell. AI Games, vol. 2, no. 4, pp. 251–258, 2010.

[4] F. Teytaud and O. Teytaud, "Creating an Upper-Confidence-Tree program for Havannah," in Proc. Adv. Comput. Games, LNCS 6048, Pamplona, Spain, 2010, pp. 65–74.

[5] B. K.-B. Tong, C. M. Ma, and C. W. Sung, "A Monte-Carlo Approach for the Endgame of Ms. Pac-Man," in Proc. IEEE Conf. Comput. Intell. Games, Seoul, South Korea, 2011, pp. 9–15.

[6] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.

[7] H. Jaap van den Herik, Jan Kuipers, Jos A.M. Vermaseren and Aske Plaat: Investigations with Monte Carlo Tree Search for finding better multivariate Horner schemes, Communications in Computer and Information Science CCIS 2014.

[8] Barriga, N. A., Stanescu, M., & Buro, M. (2014, August). Parallel UCT search on GPUs. In Computational Intelligence and Games (CIG), 2014 IEEE Conference on (pp. 1-7). IEEE.

[9] Rocki, Kamil, and Reiji Suda. "Massively parallel monte carlo tree search." Proceedings of the 9th International Meeting High Performance Computing for Computational Science. 2010.

[10] Kamil Rocki, Reiji Suda, Parallel Monte Carlo Tree Search on GPU, Proceedings of SCAI, 2011