論文の内容の要旨

Large Scale Tree Search on GPU
(GPU による大規模木探索)

氏名　　ロツキ　カミル　マレック

Monte Carlo Tree Search (MCTS) is a method for making optimal decisions in artificial intelligence (AI) problems, typically move planning in combinatorial games. It combines the generality of random simulation with the precision of tree search. Research interest in MCTS has risen sharply due to its spectacular success with computer Go and potential application to a number of other difficult problems. Its application extends beyond games, and MCTS can theoretically be applied to any domain that can be described in terms of state, action pairs and simulation used to forecast outcomes such as decision support, control, delayed reward problems or complex optimization. The main advantages of the MCTS algorithm are that it does not require any strategic or tactical knowledge about the given domain to make reasonable decisions and algorithm can be halted at any time to return the current best estimate. So far, current research has shown that the algorithm can be parallelized on multiple CPUs.

The motivation behind this work was caused by the emerging GPU- based systems and their high computational potential combined with relatively low power usage compared to CPUs. As a problem to be solved I chose developing an AI GPU(Graphics Processing Unit)-based agent in the game of Reversi (Othello) and SameGame puzzle which provide sufficiently complex problems for tree searching with non-uniform structure. The importance of this research is that if the MCTS algorithm can be efficiently parallelized on GPU(s) it can also be applied to other similar problems on modern multi-CPU/GPU systems such as the TSUBAME 2.0 supercomputer. Tree searching algorithms are hard to parallelize, especially when GPU is considered. Finding an algorithm which is suitable for GPUs is crucial if tree search has to be performed on recent supercomputers. Conventional ones do not provide good performance, because of the limitations of the GPUs' architecture and the programming scheme, threads' communication boundaries. One of the problems is the SIMD execution scheme within GPU for a group of threads. It means that a standard CPU parallel implementation such as root-parallelism fail. The other problem is a difficulty in generating pseudo-random numbers on GPU which is important in Monte Carlo methods. Available methods are usually very time consuming. Third of all, no current research work discusses scalability of the algorithm for millions of threads (when multiple GPUs are considered), so it is important to estimate to what

extent the parallelism can be increased.In this thesis I present an efficient parallel GPU MCTS implementa- tion based on the introduced 'block-parallelism' scheme which combines GPU SIMD thread groups and performs independent searches without any need of intra-GPU or inter-GPU communication. I compare it with a simple leaf parallel scheme which implies certain performance limitations. The obtained results show that using my GPU MCTS implementation on the TSUBAME 2.0 system one GPU's performance can be compared to 50-100 CPU threads depending on factors such as the search time and other MCTS parameters. The block-parallel algorithm provides better results than the naive leaf-parallel scheme which fail to scale well beyond 1000 threads on a single GPU. The block-parallel algorithm is approximately 4 times more efficient in terms of the number of CPU threads' results comparable with the GPU implementation. In order not to generate random numbers on GPU I introduce an algorithm, where the numbers are transferred from the CPU for each GPU block accessible as a look-up table. This approach makes the time needed for random-sequence generation insignificantly small. In this thesis for the first time I discuss scalability of the algorithm for millions of threads. The program is designed in the way that it can be run on many nodes using Message Passing Interface (MPI) standard. As a method of evaluating my results I compared the results of multiple CPU cores and GPUs playing against the standard sequential CPU implementation. Therefore the algorithm's scalability is analyzed for multiple CPUs and GPUs. My results show that this algorithm implies almost no inter-node communication overhead and it scales linearly in terms of the number of simulation performed in a given time period. However, beyond a certain number of running threads, a lack of performance improvement was observed. I concluded that this limit is affected by the algorithm's implementation and it can be improved to some extent by tuning the parameters or adjusting the algorithm itself. The improvements I propose and analyze are variance-based error estimation and simultaneous CPU/GPU execution. Using these two methods modifying the MCTS algorithm the overall effectiveness can be increased by 10-15% further, compared to the basis block-parallel implementation. Also, another factor considered is the criteria of estimating the performance is the overall score of the game (win percentage or the score). Not all the parameters in the MCTS algorithm are analyzed thoroughly in regarding the GPU's implementation and their importance considering scalability. This is caused by the certain limitations of the proposed evaluation method. As it is based on the average score, multiple games have to be played to get accurate results and time needed to aquire them is relatively long.