Parallel Game Tree Search Using GPU

Ľubomír Lackovič*
Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 3, 842 16 Bratislava, Slovakia
lubomir.lackovic@gmail.com

Abstract

Parallel performance of graphics cards in desktop computers generally outreaches performance of conventional processors. The purpose of this paper is to identify possibilities of tasks parallelization when searching and evaluating game trees and to propose algorithms that would perform better on SIMD processors of graphics cards than on regular desktop processors. On proposed algorithms' basis are created two prototypes that implement game tree search of game Czech draughts. Performance of these prototypes is compared in different levels of game tree on various compute devices. Implemented algorithm achieves approximately two to three times better performance using graphics card against conventional quad-core processor.

Categories and Subject Descriptors

I.2.8 [**Artificial intelligence**]: Problem Solving, Control Methods, and Search—*graph and tree search strategies*; I.3.1 [**Computer graphics**]: Hardware architecture—*graphics processors, parallel processing*

Keywords

Game tree search, evaluation, graphics card, GPU, OpenCL, draughts, parallelization

1. Introduction

Searching in complete game trees by conventional means, such are processors, is mostly impossible task due to time complexity. Common partial solution to this problem is to search only up to some predefined maximal depth following the rule - the more nodes we explore, the better solution we get.

With advance in GPGPU (General-purpose computing on graphics processing units) one of the options of exploring higher amount of nodes appears to be the use of massive parallel performance of graphics cards. Graphics cards achieve in some specific tasks higher performance than conventional processors. These tasks are primarily specific by need of decomposition of main task into higher number of smaller subtasks that can be processed concurrently by stream processors of graphics card. The processing speed of such decomposed subtasks is, however, farther dependent on architecture of graphics cards consisting of several SIMD (Single instruction multiple data) processors. The algorithms designed for SIMD processors differ from other parallel algorithms mainly by closer connection with underlying hardware and the need of its correct understanding. It is therefore the reason to pay higher attention to designing of the algorithm. Incorrectly designed algorithm for this architecture need not have any performance contribution, or even can perform worse. On the other hand well designed algorithm can bring high performance boost.

In the following sections we have identified tasks that can be processed concurrently when searching in game tree. Consequently we have designed game tree search algorithms for game Czech draughts optimized for execution on SIMD processors of graphics cards. Lastly we have compared performance of designed algorithms when executed on graphics cards against conventional desktop processor.

2. Parallelization in game trees

The most common goal of game tree search is finding of player's move that maximizes his chance of winning. To achieve this goal it is necessary to create game tree consisting of as much nodes as possible generated by legal moves of both players. Consequently the game tree is evaluated and it is chosen the move that leads to node with highest price. Commonly the game tree is created by sequential generation of game nodes (e.g. breadth-first search, depth-first search, iterative deepening depth-first search). Evaluating of game nodes is also realized sequentially, the most common algorithms involve algorithm minimax and its modifications [5].

*Master degree study programme in field Information Systems. Supervisor: Dr. Peter Lacko, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies, STU in Bratislava.
Work described in this paper was presented at the 7th Student Research Conference in Informatics and Information Technologies IIT.SRC 2011 and awarded by IEEE Czechoslovakia Section Award.

Lackovič, Ľ. Parallel Game Tree Search Using GPU. Information Sciences and Technologies Bulletin of the ACM Slovakia, Special Section on Student Research in Informatics and Information Technologies, Vol. 3, No. 2 (2011) 79-82

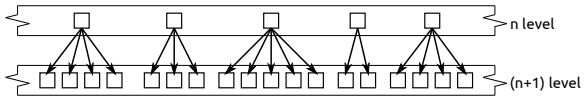

Figure 1: Generation of game nodes - every thread is processing one node and generates several children nodes in deeper level of the game tree.

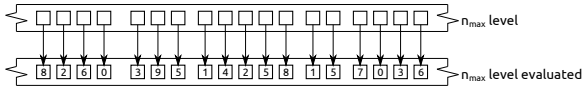

Figure 2: Evaluation of end nodes - every thread is processing one node and computes its price.

First step when adopting parallelization at game tree searching is therefore choosing appropriate algorithm for searching and identifying parts that can be executed concurrently. After analysing several sequential game tree searching algorithms the most natural way appears to be the use of breadth-first search algorithm and minimax algorithm for game nodes evaluation. When we take a closer look at them, there are three parts at which we can apply parallelization. The identified parts are generation of game nodes, evaluation of end nodes and backwards evaluation of game tree levels.

2.1 Generation of game nodes

When generating game nodes the input of the algorithm consists always of game nodes that have the same depth in game tree. The goal of the algorithm is to generate new game nodes of the next deeper level of the game tree (see Figure 1). New game nodes are created from parent node by playing all possible moves.

Parallelization is applied on input nodes where each of these nodes generates usually different amount of children nodes that are in the next deeper level in game tree. In one execution of this parallel algorithm we can process only game nodes that have the same level in game tree. Algorithm can be by sequel executed on nodes in deeper level of game tree while the input of the algorithm is always composed of the output nodes from previous execution of the algorithm. Therefore by sequel execution of this algorithm we can theoretically create complete game tree.

2.2 Evaluation of end nodes

The next identified possible use of parallelization in game trees is parallel evaluation of end nodes of game tree. The input of the algorithm consists of non-evaluated end nodes. The goal of the algorithm is to determine price for each of these nodes (how beneficial is getting into given node for the player). The output of algorithm consists of evaluated end nodes (see Figure 2).

2.3 Backwards evaluation of game tree levels

The last identified possible use of parallelization in game trees is backwards evaluation of game tree after end nodes were evaluated. The input of the algorithm consists always of game nodes in lowest not yet evaluated level in game tree. For each node algorithm determines price as the maximum or minimum (depending on actual game tree level) of the prices of node's children (algorithm minimax). The output of the algorithm consists of evaluated

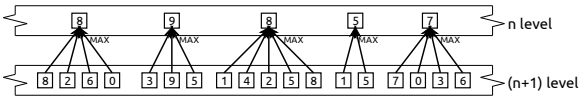

Figure 3: Backwards evaluation of game tree levels - every thread is processing one node and determines its price as the maximum (minimum) of its children nodes.

input nodes (see Figure 3). By sequel execution of algorithm on nodes from deeper game tree levels up to root node we can evaluate the whole game tree and therefore find the best possible move in the game for the player.

3. Parallel generation of game nodes in draughts

From identified possible uses of parallelization in game trees described in previous section we focus on generation of game nodes. Generation of nodes is the most time consuming operation from the whole process of game tree search.

For exploration of the next detailed possibilities of parallel generation of game nodes we use game tree of the game Czech draughts[1]. Czech draughts are played only on black cells of game board of size 8x8. Each of the players has on the game opening eight stones on his side of the game board. Stones can move only about one cell forward diagonally. After reaching the farthest opposite row the stone turns into king that can move diagonally all directions about arbitrary amount of cells. The goal of the game is by jumping over the opponent's stones to eliminate all his stones on the game board.

Game tree of the draughts consist of nodes that represent game boards. Game board in deeper level of the game tree is created from parent board by moving with exactly one stone in it. Branching factor of draughts' game tree has at the game opening value of 7, but during the game it has tendency to increase markedly. In early game it is caused by revealing of stones on border row and thus allowing them to move, in the later game it is due to higher amount of possible moves of kings. Total number of nodes in complete draughts' game tree is at about 10^{31} nodes [1].

As we implied in section 2.1 the simplest way of parallel generation of game nodes is when we process in parallel always only one whole level of game tree at the time that results in generation of nodes in the next deeper level of game tree. When processing game tree of draughts we can go even farther and to process in parallel even cells or stones. On the basis of how much work the single thread will do we can recognize three different approaches - parallel processing of game boards, parallel processing of cells in game boards and parallel processing of stones in game boards.

3.1 Parallel processing of game boards

When considering this approach of generation of game nodes every thread processes exactly one game board and creates its children (see Figure 4). The thread in sequel checks black cells whether the player's stone is present on its position. When the thread finds player's stone on the

[1] http://www.damweb.cz/pravidla/cdfull.html

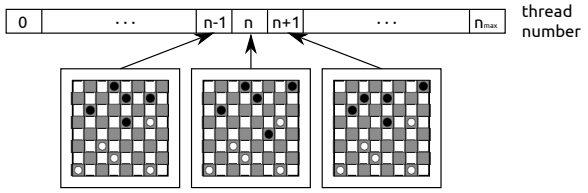

Figure 4: Parallel processing of game boards - every thread processes all player's stones on exactly one assigned game board.

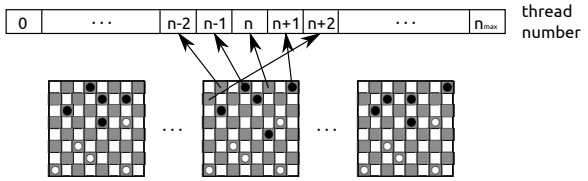

Figure 5: Parallel processing of cells in game boards - every thread processes one black cell of game board (game boards can be processed also in parallel), if the cell does not contain any stone the threads ends immediately.

cell it tries to move by this stone with all possible legal moves and thereby generates new children game boards.

Disadvantage of this approach results from SIMD architecture of graphics cards. Considering that the work items of SIMD processor always process the cell on the same position but on different board at the time, often happens that if one thread finds stone on this position, other threads that do not find have to wait until threads which found stones process those stones. Therefore parallel performance of graphics card is not fully utilized.

This approach can be markedly improved if threads firstly create lists of cells with stones to process. Consequently the threads cycle over items (stones on cells) in their lists and process them. Efficiency of SIMD processors' usage is in this case dependent on how much differ the numbers of stones in processed game boards.

3.2 Parallel processing of cells in game boards

When considering approach of parallel processing cells in game boards every thread has assigned one cell that the thread processes (see Figure 5). The advantage of this approach lies in that there is no need of complicated finding out what to process because it is implicitly defined. We can simply determine concrete processing cell from thread number. On the other hand huge disadvantage is inefficient usage of SIMD processor, because of the most of the threads end already on the start of their program when they do not find any stones on assigned cells. Such threads can continue processing another cell only after all work items in SIMD processors process their cells and that can take a lot of time, especially if other work items find stones on assigned cells.

3.3 Parallel processing of stones in game boards

The goal of parallel processing of stones in game boards is to eliminate inefficiency of previous approach when threads try to process cells with no stones. Difficult thing about this approach is the way of determination of cells with stones that the threads will process.

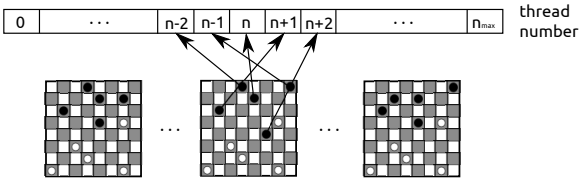

Figure 6: Parallel processing of stones in game boards - every thread firstly finds cell with stone on basis of its thread index and than processes it (game boards can be processed also in parallel).

The simplest solution for this problem is to assign to each game board the number of threads that is equal to maximum number of stones that the player can have on the game board (in draughts eight). Consequently every thread has to find on its start on the basis of its index order of the stone in the game board that it will process (see Figure 6).

Disadvantage of this approach is overhead needed to find correct cell with stone on it that the thread will process and worse usage of SIMD processor if there is not maximum number of stones on the board (especially in the late game). Despite of that is the usage of the SIMD processor much better than in previous approach.

4. Prototypes of parallel game tree search

On the basis of last two identified approaches we have created two prototypes that create game tree of draughts with respect to legal moves of ordinary stones. Prototypes are implemented using OpenCL framework that allows to execute prototypes besides on graphics cards also on conventional processors.

4.1 First prototype

First prototype uses approach of parallel processing of cells described in section 3.2. This prototypes has been primarily developed for graphics card ATI Radeon 4890 that is characterized by only elementary support for OpenCL. It supports minimal number of OpenCL extensions and missing LDS (Local Data Storage) memory is emulated by slower global memory. That lead to many compromises (involving used data structures and overall work with memory) needed to be able to execute prototype on this graphics card. Prototype is able to generate nodes from maximum depth of six due to memory problems.

4.2 Second prototype

Second prototype uses approach of parallel processing of stones described in section 3.3. This prototype has been primarily developed for newer graphics card Nvidia GTX460 with decent OpenCL support and language extensions. The usage of these extensions for advanced memory addressing and atomic instructions allowed implementation of more efficient algorithm but at the cost of incompatibility with older graphics cards. Prototype is able to generate nodes from maximum depth of eight (for deeper levels algorithm modification is necessary).

4.3 Performance comparison

In Table 1 can be found comparison of implemented algorithms between graphics cards ATI Radeon 4890 and Nvidia GTX460 and quad-core processor Intel i5 750.

Table 1: Performance comparison.

Depth	First Prototype [ms]			Second Prototype [ms]	
	Radeon 4890	GTX460	i5 750	GTX460	i5 750
2	318,21	0,62	1,63	1,07	0,89
3	327,04	0,77	1,74	0,69	1,03
4	333,05	1,04	7,94	0,68	1,41
5	403,96	5,21	11,84	0,99	3,53
6	859,17	38,86	52,75	3,89	20,63
7	-	-	-	33,2	75,32
8	-	-	-	222,46	612,16

Performance of the prototypes was measured on operating system Ubuntu 10.10.

As we can see from comparison first prototype running on Radeon 4890 performs much worse than processor Intel i5 750 although the prototype is optimized for this card. Nvidia GTX460 with better GPGPU support performs a little better than processor in deeper levels of game tree, but performance gain is not very impressive.

After implementing second prototype using algorithm of parallel processing of stones performance when running on processor did not change a lot, but there was a decent performance boost when running on graphics card Nvidia GTX460. This was mainly caused by better utilization of SIMD processors due to used algorithm and memory optimizations.

5. Conclusion

In this paper we have identified three different parts of common game tree search algorithms that can be parallelized - generation of game nodes, evaluation of end nodes and backwards evaluation of game tree levels. Since generation of game nodes is usually the most time consuming operation we have focused at finding algorithms that would decrease this time using graphics card.

We have examined different approaches of game nodes generation for game Czech draughts and demonstrated that not all of these approaches are suitable for efficient processing by graphics cards. As the best solution it has proved to be parallel processing of stones although small overhead is needed to find positions of stones on the game board at the algorithm start.

As a result of this study we have achieved approximately two to three times higher speed of the nodes generation using graphics card Nvidia GTX460 compared to quad-core processor Intel i5 750. This proves that graphics card can be efficiently used to accelerate game tree searching.

Acknowledgement. This work was partially supported by the Scientific Grant Agency of Slovak Republic under grants VG1/0141/10 and VG1/0508/09.

References

[1] V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, University of Limburg, Maastricht, 1994.

[2] AMD. *ATI Stream Computing Programming Guide: OpenCL*. Sunnyvale, 2010.

[3] R. Feldmann. *Game Tree Search on Massively Parallel Systems*. PhD thesis, University of Paderborn, Paderborn, 1993.

[4] T. A. Marsland and F. Popowich. Parallel game tree search. Technical report, University of Alberta, Edmonton, 1984.

[5] P. Návrat et al. *Umelá inteligencia*. STU Press, Bratislava, 2007.