



Universiteit
Leiden
The Netherlands

Structured parallel programming for Monte Carlo Tree Search

Mirsoleimani, S.A.

Citation

Mirsoleimani, S. A. (2020, June 17). *Structured parallel programming for Monte Carlo Tree Search*. *SIKS Dissertation Series*. Retrieved from <https://hdl.handle.net/1887/119358>

Version: Publisher's Version

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/119358>

Note: To cite this publication please use the final published version (if applicable).

Cover Page



Universiteit Leiden



The handle <http://hdl.handle.net/1887/119358> holds various files of this Leiden University dissertation.

Author: Mirsoleimani, S.A.

Title: Structured parallel programming for Monte Carlo tree search

Issue Date: 2020-06-17

Structured Parallel Programming
FOR
Monte Carlo Tree Search

S. Ali Mirsoleimani

Structured Parallel Programming

FOR

Monte Carlo Tree Search

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof. mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 17 juni 2020
klokke 11.15 uur

door

Sayyed Ali Mirsoleimani
geboren te Abadeh, Iran
in 1986

Promotoren:

Prof. dr. H. J. van den Herik

Prof. dr. A. Plaat

Copromotor:

Dr. J. A. M. Vermaseren

Nikhef

Promotiecommissie:

Prof. dr. P. J. G. Mulders

Vrije Universiteit Amsterdam

Prof. dr. F. J. Verbeek

Prof. dr. H. A. G. Wijshoff

Dr. F. Khunjush

Shiraz University, Shiraz, Iran

Dr. W. A. Kusters

Dr. ir. A. L. Varbanescu

Universiteit van Amsterdam



HEPGAME, ERC Advanced Grant No. 320651

The research reported in this thesis has been additionally funded by Nikhef, the Nationaal instituut voor subatomaire fysica.



In the first year, the research reported in this thesis has been performed at Tilburg center for Cognition and Communication (TiCC) at Tilburg University, the Netherlands.



The research reported in this thesis has been completed at

Leiden Centre of Data Science (LCDS) hosted by Leiden Institute of Advanced Computer Science (LIACS) at the Faculty of Science, Universiteit Leiden, the Netherlands.



SIKS Dissertation Series No. 2020-08

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Copyright © 2020 by S.A. Mirsoleimani

An electronic version of this dissertation is available at

<http://openaccess.leidenuniv.nl/>.

I would like to dedicate this thesis to my wife Elahe and to my parents,
for all of their love and support.

In loving memory of my grandfathers,
Bahram and Abolghasem

Preface

The thesis is part of a bigger project, the HEPGAME (High Energy Physics Game). The project started in 2011 when Jos Vermaseren developed the first ideas on improving FORM at Nikhef, Amsterdam. In 2012 he submitted an ERC advanced research grant together with Tilburg University. It was accepted on 12/12/2012. Half a year later in July 2013, the program started. The main objective for HEPGAME was the utilization of AI solutions, particularly by using MCTS for simplification of HEP calculations. One of the issues is solving mathematical expressions of interest with millions of terms. Up to 2011, these calculations were executed with the FORM program, which is software for symbolic manipulation. These calculations are computationally intensive and take a large amount of time. Hence, the FORM program was parallelized to solve large equations in a reasonable amount of time. Therefore, any new algorithm, for instance, the ones based on MCTS, should also be parallelized. Here our research comes in. It is dedicated to parallelization of MCTS on multi-core and many-core processors. The research was ambitious and challenging. Therefore, we divided the research area into three main parts: (1) the evaluation of current methods for parallelization of MCTS, (2) addressing the shortcomings in these methods, and (3) providing new ways of parallelization for MCTS. In the first part, we investigated the current methods and evaluated them in terms of performance and scalability on both multi-core and manycore processors. In the second part, we examined how we can solve the actual shortcomings in the existing parallelization methods for MCTS. The third part was dedicated to finding new ideas, methods, and ways beyond the existing ones to parallelize MCTS.

Sayyed Ali Mirsoleimani, Leiden, July 2019

Contents

Preface	vii
Contents	ix
List of Definitions	xv
List of Figures	xvii
List of Tables	xxi
List of Listings	xxiii
List of Abbreviations	xxv
1 Introduction	1
1.1 HEPGAME	2
1.2 Monte Carlo Tree Search	2
1.3 Parallelism and Parallelization	3
1.3.1 Thread-level Parallelization	4
1.3.2 Task-level Parallelization	4
1.4 General Obstacles for Parallelization of MCTS	5
1.4.1 Irregular Parallelism Causes Load Balancing Overhead	6
1.4.2 Shared Data Structure Causes Synchronization Overhead	6
1.4.3 Ignoring Data Dependencies Causes Search Overhead	7
1.4.4 Complex Interactions Leading to Deployment Overhead	8

1.5	Performance and Scalability Studies	8
1.6	Scope and Research Goals	9
1.7	Problem Statement and Research Questions	10
1.8	Research Methodology	12
1.9	Structure of the thesis	12
1.10	Contributions	13
2	Background	15
2.1	Upper Confidence Bound (UCB)	16
2.2	Upper Confidence Bounds for Trees (UCT)	16
2.2.1	UCT Formula	17
2.2.2	UCT Algorithm	17
2.3	Parallelization Methods for MCTS	17
2.3.1	Parallel Methods with a Shared Data Structure	17
2.3.2	Parallel Methods with More than one Data Structure	18
2.4	Case Studies	19
2.4.1	Case 1: The Game of Hex	19
2.4.2	Case 2: Horner Schemes	20
2.5	Performance Metrics	20
2.5.1	Playout Speedup	21
2.5.2	Playing Strength	21
2.6	Our ParallelUCT Package	22
2.6.1	Framework of multiple benchmark problems	22
2.6.2	Framework of multiple parallelization methods	23
2.6.3	Framework of multiple programming models	23
3	Thread-level Parallelization for MCTS	25
3.1	Micro-benchmark Code Performance	27
3.1.1	Xeon Phi Micro-architecture	27
3.1.2	Experimental Setup	28
3.1.3	Experimental Design	30
3.1.4	Experimental Results	30
3.1.5	Section Conclusion	33
3.2	FUEGO Performance and Scalability	33
3.2.1	Experimental Setup	34
3.2.2	Experimental Design	34
3.2.3	Experimental Results	35
3.2.4	Section Conclusion	39
3.2.5	Answer to RQ1a for FUEGO	39
3.3	ParallelUCT Performance and Scalability	40

3.3.1	Experimental Setup	40
3.3.2	Experimental Design	41
3.3.3	Experimental Results	41
3.3.4	Section Conclusions	46
3.3.5	Answer to RQ1a for ParallelUCT	47
3.4	Related Work	48
3.5	Answer to RQ1	48
4	Task-level Parallelization for MCTS	51
4.1	Irregular Parallelism Challenge	52
4.2	Achieving Task-level Parallelization	52
4.2.1	Decomposition of Iterations into Tasks	53
4.2.2	Ignoring Data Dependencies among Iterations	53
4.3	Threading Libraries	53
4.3.1	Cilk Plus	54
4.3.2	Threading Building Blocks	54
4.4	Grain Size Controlled Parallel MCTS	54
4.5	Implementation Considerations	56
4.5.1	Shared Search Tree Using Locks	56
4.5.2	Random Number Generator	57
4.6	Performance and Scalability Study	57
4.7	Experimental Setup	58
4.8	Experimental Design	58
4.9	Experimental Results	59
4.10	Discussion and Analysis	60
4.11	Related Work	64
4.12	Answer to RQ2	64
5	A Lock-free Algorithm for Parallel MCTS	67
5.1	Shared Data Structure Challenge	68
5.1.1	Parallelization with a Single Shared Tree	69
5.1.2	The Race Conditions	69
5.1.3	Protecting Shared Data Structure	70
5.2	Related Work	71
5.2.1	Lock-based Methods	71
5.2.2	Lock-free Methods	72
5.3	A New Lock-free Tree Data Structure and Algorithm	73
5.4	Implementation Considerations	77
5.5	Experimental Setup	77
5.5.1	The Game of Hex	78

5.5.2	Performance Metrics	78
5.5.3	Hardware	78
5.6	Experimental Design	78
5.7	Experimental Results	79
5.7.1	Scalability and C_p parameters	79
5.7.2	GSCPM vs. Root Parallelization	82
5.8	Answer to RQ3	83
6	Pipeline Pattern for Parallel MCTS	85
6.1	Data Dependencies Challenges	86
6.1.1	Loop Independent Data Dependency	86
6.1.2	Loop Carried Data Dependency	87
6.1.3	Why a Pipeline Pattern?	87
6.2	Design of 3PMCTS	88
6.2.1	A Pipeline Pattern for MCTS	88
6.2.2	Pipeline Construction	91
6.3	Implementation Considerations	92
6.4	Experimental Setup	92
6.4.1	Horner Scheme	92
6.4.2	Performance Metrics	93
6.4.3	Hardware	93
6.5	Experimental Design	93
6.6	Experimental Results	94
6.6.1	Performance and Scalability of 3PMCTS	94
6.6.2	Flexibility of Task Decomposition in 3PMCTS	96
6.7	Answer to RQ4	97
7	Ensemble UCT Needs High Exploitation	99
7.1	Ensemble UCT	100
7.2	Related Work	101
7.3	Experimental Setup	102
7.3.1	The Game of Hex	102
7.3.2	Hardware	102
7.4	Experimental Design	103
7.5	Experimental Results	103
7.6	Answer to the First Part of RQ5	106

8	An Analysis of Virtual Loss in Parallel MCTS	109
8.1	Virtual Loss	110
8.2	Related Work	112
8.3	Experimental Setup	112
8.4	Experimental Design	112
8.5	Experimental Results	113
8.6	Answer to the Second Part of RQ5	115
8.7	A Complete answer to RQ5	115
9	Conclusions and Future Research	117
9.1	Answers to the RQs	117
9.1.1	Answer to RQ1	117
9.1.2	Answer to RQ2	118
9.1.3	Answer to RQ3	118
9.1.4	Answer to RQ4	119
9.1.5	Answer to RQ5	119
9.2	Answer to the PS	120
9.3	Limitations	120
9.3.1	Maximizing Hardware Usage	120
9.3.2	Using More Case Studies	121
9.4	Future Research	121
	Bibliography	123
	Appendices	131
A	Micro-benchmark Programs	133
B	Statistical Analysis of Self-play Experiments	135
C	Implementation of GSCPM	137
C.1	TBB	137
C.2	Cilk Plus	137
C.3	TPFIFO	138
D	Implementation of 3PMCTS	139
D.1	Definition of Token Data Type (TDT)	139
D.2	TBB Implementation Using TDD	141
	Summary	143

Samenvatting	145
Acknowledgment	147
Curriculum Vitae	149
Publications	151
SIKS Dissertation Series	153

List of Definitions

1.1	Parallelization	3
1.2	Thread	4
1.3	Multi-core Processor	4
1.4	Task	4
1.5	Many-core Processor	5
1.6	Parallel Pattern	5
1.7	Irregular Parallelism	6
1.8	Load Balancing	6
1.9	Shared Data Structure	7
1.10	Synchronization	7
1.11	Loop Carried Data Dependency	7
1.12	Loop Independent Data Dependency	7
1.13	Search Overhead	7
1.14	Complex Interactions	8
1.15	Deployment Overhead	8
1.16	Performance Study	8
1.17	Payout Speedup	8
1.18	Playing Strength	8
1.19	Scalability Study	9
1.20	Memory Bandwidth	9
1.21	Uniform Memory Access	10
1.22	Many Integrated Core	10
1.23	Non Uniform Memory Access	10

2.1	Exploitation	16
2.2	Exploration	16
2.3	Tree Parallelization	18
2.4	Root Parallelization	19
2.5	Strong Scalability	21
3.1	Thread Affinity Policy	28
3.2	Double-Precision Floating-Point Format	29
3.3	Integer Format	30
4.1	Iteration Pattern	53
4.2	Fork-join Pattern	53
4.3	Iteration-level Task	55
4.4	Iteration-level Parallelism	55
4.5	Fork-join Parallelism	55
5.1	Race Condition	69
5.2	Lock-based	70
5.3	Lock-free	71
6.1	Operation-Level Task	86
6.2	Operation-Level Parallelism	86
6.3	Sequence Pattern	87
6.4	Iteration Pattern	87
6.5	Pipeline Pattern	88
8.1	Virtual Loss	110

List of Figures

1.1	An example of the search tree.	3
1.2	The main loop of MCTS.	3
1.3	One iteration of MCTS.	4
2.1	A sample board for the game of Hex	19
3.1	Intel Xeon Phi Architecture.	29
3.2	Performance and scalability of double-precision operations for different numbers of iterations.	31
3.3	Memory bandwidth of double-precision operations on the Xeon Phi for increasing numbers of threads. Each interval contains 27 points.	32
3.4	Performance and scalability of integer operations of the Xeon Phi for different numbers of threads.	33
3.5	Performance and scalability of FUEGO in terms of PPS when it makes the second move. Average of 100 games for each data point. The board size is 9×9	35
3.6	Scalability of FUEGO in terms of PW with N threads against FUEGO with $N/2$ threads. The board size is 9×9	38
3.7	Performance and scalability of ParallelUCT in terms of PPS for both Tree and Root Parallelization.	43
3.8	Scalability of ParallelUCT in terms of PW for Tree Parallelization.	44
3.9	Scalability of ParallelUCT in terms of PW for Root Parallelization.	46

4.1	The scalability profile produced by Cilkview for the GSCPM algorithm. The number of tasks is shown. Higher is more fine-grained.	57
4.2	Speedup for task-level parallelization utilizing five methods for parallel implementation from four threading libraries. Higher is better. Left: coarse-grained parallelism. Right: fine-grained parallelism.	61
4.3	Comparing Cilkview analysis with TPFIFO speedup on the Xeon Phi. The dots show the number of tasks used for TPFIFO. The lines show the number of tasks used for Cilkview.	63
5.1	(5.1a) The initial search tree. The internal and non-terminal leaf nodes are circles. The terminal leaf nodes are squares. The curly arrows represent threads. (5.1b) Thread 1 and 2 are expanding node v_6 . (5.1c) Thread 1 and 2 are updating node v_3 . (5.1d) Thread 1 is selecting node v_3 while thread 2 is updating this node.	69
5.2	Tree parallelization with coarse-grained lock.	72
5.3	Tree parallelization with fine-grained lock.	72
5.4	The scalability of Tree Parallelization for different parallel programming libraries when $C_p = 1$. (5.4a) Coarse-grained lock. (5.4b) Lock-free.	80
5.5	The scalability of Tree Parallelization for different parallel programming libraries when $C_p = 1$ on the Xeon Phi. (5.5a) Coarse-grained lock. (5.5b) Lock-free.	80
5.6	(5.6a) The scalability of the algorithm for different C_p values. (5.6b) Changes in the depth of tree when the number of tasks are increasing.	81
5.7	The playing results for lock-free Tree Parallelization versus Root Parallelization. The first value for C_p is used for Tree Parallelization and the second value is used for Root Parallelization.	82
6.1	(6.1a) Flowchart of a pipeline with sequential stages for MCTS. (6.1b) Flowchart of a pipeline with parallel stages for MCTS.	88
6.2	Scheduling diagram of a pipeline with sequential stages for MCTS. The computations for stages are equal.	89
6.3	Scheduling diagram of a pipeline with sequential stages for MCTS. The computations for stages are not equal.	89
6.4	Scheduling diagram of a pipeline with parallel stages for MCTS. Using parallel stages create load balancing.	90
6.5	The 3PMCTS algorithm with a pipeline that has three parallel stages (i.e., EXPAND, RANDOMSIMULATION, and EVALUATION).	91

6.6	Playout-speedup as function of the number of tasks (tokens). Each data point is an average of 21 runs for a search budget of 8192 playouts. The constant C_p is 0.5. Here a higher value is better.	94
6.7	Number of operations as function of the number of tasks (tokens). Each data point is an average of 21 runs for a search budget of 8192 playouts. Here a lower value is better.	95
6.8	Percentage of win as function of the number of tasks (tokens). Each data point is the outcome of 100 rounds of playing between the two opponent players. Each player has a search budget of $2^{20} = 1,048,576$ playouts in each round. Here a higher value is better.	97
7.1	The number of visits for root's children in Ensemble UCT and plain UCT. Each child represents an available move on the empty Hex board with size 11×11 . Both Ensemble UCT and plain UCT have 80,000 playouts and $C_p = 0$. In Ensemble UCT, the size of the ensemble is 8. .	104
7.2	The percentage of wins for ensemble UCT is reported. The value of C_p for plain UCT is always 1.0 when playing against Ensemble UCT. To the left few large UCT trees, to the right many small UCT trees.	105
8.1	Search overhead (SO) for Horner (average of 20 instances for each data point). Tree parallelization is the green line which is indicated by circles, and Tree Parallelization with virtual loss is the blue line which is indicated by triangles. Note that the higher SO of Tree Parallelization with virtual loss means lower performance.	113
8.2	Efficiency (Eff) for Horner (average of 20 instances for each data point). Tree parallelization is the green line which is indicated by circles and Tree Parallelization with virtual loss is the blue line which is indicated by triangles. Note that Tree Parallelization with virtual loss has a lower efficiency meaning lower performance.	114

List of Tables

3.1	Thread affinity policies	29
3.2	Performance of FUEGO on the Xeon CPU. Each column shows data for N threads. The board size is 9×9	36
3.3	Performance of FUEGO on the Xeon Phi. Each column shows data for N threads. The board size is 9×9	37
4.1	The conceptual effect of grain size.	56
4.2	Sequential baseline for GSCPM algorithm. Time in seconds.	59
5.1	Sequential execution time in seconds.	81
6.1	Sequential time in seconds when $C_p = 0.5$	94
6.2	Definition of layouts for 3PMCTS.	96
6.3	Details of experiment to show the flexibility of 3PMCTS.	96
7.1	Different possible configurations for Ensemble UCT. Ensemble size is n	101
7.2	The performance of Ensemble UCT vs. plain UCT based on win rate.	103

List of Listings

A.1	Micro-benchmark code for measuring performance of Xeon Phi.	133
A.2	Micro-benchmark code for measuring memory bandwidth of Xeon Phi.	134
C.1	Task parallelism for GSCPM using TBB (<i>task_group</i>).	137
C.2	Task parallelism for GSCPM using Cilk Plus (<i>cilk_spawn</i>).	138
C.3	Task parallelism for GSCPM using Cilk Plus (<i>cilk_for</i>).	138
C.4	Task parallelism for GSCPM, based on TPFIFO.	138
D.1	An implementation of the 3PMCTS algorithm in TBB.	141

List of Abbreviations

3PMCTS	Pipeline Pattern for Parallel MCTS.
FIFO	First In, First Out.
FMA	Fused Multiply Add.
GFLOPS	Giga Floating Point Operations per Second.
GIPS	Giga Integers per Second.
GSCPM	Grain Size Controlled Parallel MCTS.
HEP	High Energy Physics.
HEPGAME	High Energy Physics Game.
ILD	Iteration-Level Dependency.
ILP	Iteration-Level Parallelism.
ILT	Iteration-Level Task.
ISA	Instruction Set Architecture.
MC	Memory Controller.
MCTS	Monte Carlo Tree Search.
MIC	Many Integrated Core.
NUMA	Non Uniform Memory Access.

OLD	Operation-Level Dependency.
OLP	Operation-Level Parallelism.
OLT	Operation-Level Task.
PPS	Playouts per Second.
PS	Problem Statement.
PW	Percentage of Wins.
RNG	Random Number Generation.
RQ	Research Question.
SMT	Simultaneous Multithreading.
TBB	Threading Building Blocks.
TD	Tag Directories.
TPFIFO	Thread Pool with FIFO scheduling.
UCB	Upper Confidence Bound.
UCT	Upper Confidence Bounds for Trees.
UMA	Uniform Memory Access.
VPUs	Vector Processing Units.