Parallel depth-first search algorithms

Andrej Sekáč
Software Engineering
371 79 Karlskrona
ansa14@student.bth.se

Vincent Lemordant
Software Engineering
371 79 Karlskrona
ansa14@student.bth.se

WORK DISTRIBUTION

The group members participated in the idea creation and in the report writing with the following amount of involvement :

Group Members	Idea creation	Writing report
Andrej Sekáč	55%	55%
Vincent Lemordant	45%	45%

We planned several meeting to discuss the topic, the research questions and the searching strategy. We have performed the literature searching and then separately evaluated found articles and applied necessary criteria, then we have compared resulting articles and created the final list. Afterwards we arranged further meetings to discuss article reviewing strategy and finalise goals of the research. Then we discussed the found information, formulated our conclusions and started with report writing.

ABSTRACT

With the huge developments in hardware today, the number of processors in the same computer is increasing every year. This enables great computational power to tasks handled by parallel algorithms. The Depth-First Search (DFS) algorithm is well known and widely used, but it is designed for a sequential run. In this paper, we summarise problems, which are solved by utilising powers of parallel depth-first search and we summarise parallelisation techniques used in implementation of particular algorithms. We conclude that parallel DFS is extensively studied and its applications bring significant improvements of performance and runtime.

Keywords : Multiprocessing system - Depth-First Search - Parallel algorithm - Parallelisation

INTRODUCTION

Computers today often include multiple core processors either on CPU or GPU. These allow them to perform parallel computations, possibly leading to faster problem solutions. However, a lot of known algorithms are designed for sequential run only and in many cases making them run in parallel is not a trivial task. Despite that many researchers turn their attention to this problem and are struggling to find the best ways to parallelise algorithm so they can utilise the computational power available.

One example of this is a traditional depth-first search (DFS) algorithm. It is inherently sequential and its parallelisation includes solving a number of issues [18]. One of those is the decomposition, the way how and when is the work divided to multiple workers. Another one is the load balancing, where the goal is to try to distribute the work to workers so that the computation time is used as efficiently as possible.

We believe that DFS is for these reasons a good subject of study as there are many things that can be improved. We conducted a systematic literature review to find out what are the problems in which the DFS algorithm is recently used. In relation with these problems then we studied what are the used techniques to overcome the most important limitations of DFS when used in a parallel fashion.

We selected 16 articles for deep review and conducted an analysis of reviewed research. We found two mainly used approaches to the decomposition issue. Both of them were used in the same number of different problems and therefore we cannot conclude any preference or advantage of any of the approaches. Problem of load balancing was only sparsely discussed in the reviewed articles and so we have no findings that could help us to understand solutions of this issue further.

We found 11 unique problems that are solved using DFS algorithm. And one of them was extensively studied in 6 of the reviewed research articles. Within these articles the decomposition technique did not differ. Except for that, we did not find any relation between the problem and used parallelisation techniques, not even between the state space representation and parallelisation techniques. In addition we found that all of the studies reached successful results and most of them improved the runtime by using parallel version of DFS algorithm. In some cases this improvement was significant as in the case of Baldwin [3].

From the gathered data we concluded that despite the number of problems that need to be solved to successfully implement parallel DFS algorithm, improvements in performance are reached. These leads us to believe that further research on parallelisation of DFS can help better utilise available multi-core computational power.

Stage	**Applied criteria**	**Studies left**
Searching using the basic searching string	no inclusion/exclusion criteria	296
Modifying the search	• search in Title/Abstract/Subject fields	272
Narrowing the search	• English only • Studies peer reviewed and published • Studies available in full text • Studies published in 2010 and later	48
Reading abstracts and briefly articles	• Studies addressing parallelization • Studies answering the research questions	16

Table 1: Studies inclusion process in stages

Finally, we found missing a deep analysis of two different solutions for LTL model checking problem. One proposed by Laarman based on Nested DFS [10] and the other by Xuan-Linh Ha based on Strong Connected Components together with Tarjan's algorithm [9]. We believe that further research of these two approaches and their comparison can bring light on possibilities when dealing with LTL model checking problem. For this we propose a possible research question: "What is the most time efficient algorithm for LTL model checking problem?". An rigorous experimental comparison of available algorithms can point a way to following research.

RESEARCH QUESTIONS

To understand current trends in parallel programming of depth-first searching algorithm we formulated following research questions.

RQ.1: *What are possible decomposition and load balancing techniques in parallelization of the depth-first search algorithm?*

The depth-first searching algorithm is well known and widely used. Its parallel version can help achieving better performance for extensive searching tasks. However, the strategies of implementing a parallel depth-first algorithm vary. For this reason in our research we want to focus on different ways of parallelisation. First key part of parallelisation of depth-first algorithm is decomposition, which means how and when is the computation divided to multiple computation nodes [18]. Second is the load balancing, a way how to balance the computation load on nodes so, that their computation power and time is efficiently used [18].

RQ.2: *What are the recently studied problems where parallelised depth-first algorithm is used and how was their solution performance influenced by the parallelisation?*

There exist a number of different approaches to parallelization of the depth-first algorithm. Their application depends on the problem they are used for. To find possible links between problems and parallelisation techniques used we want to summarise the recent studies, which use depth-first search algorithm and create an overview of problems studied. Furthermore we want to see if and how parallelisation improved runtimes.

RESEARCH METHODOLOGY

The research is conducted in a form of systematic literature review. Furthermore the research complies to guidelines presented by Kitchenham [17]. This study tries to present an overview of parallelization methods used in recent research.

Search strategy and refining search strings

For developing a searching strategy we followed guidelines by Kitchenham [17]. We chose the Inspec as our searching engine as it is a renowned engine in the field of Computer Science and offers a variety of available articles and an easy to use searching interface. Based on the research questions we selected following keywords: parallelization, search algorithm, depth first.

Then we started our searching by creating the preliminary search strings. We iteratively used and modified preliminary searching strings to first estimate the volume of available literature and then to narrow our search enough. To help ourselves with modifying of keywords we have randomly chosen some articles and took inspiration from keywords used in the articles. After several iterations we finalized our search with the following searching string:

((parallelization) AND (search OR searching) AND (algorithm) AND (depth first)

To further narrow our search results we applied the searching string on the title, abstract and subject fields.

Include/exclude criteria

With defined searching strategy we developed several criteria used for determining whether to include an article or not. These criteria were then applied on the results obtained by following our searching strategy. For application of some criteria the researchers had to read the articles. The following list is the final list of criteria used in this study.

- Articles written in English
- Articles peer reviewed and published
- Articles available in full text
- Search string applied to Title/Abstract/Subject fields
- Articles addressing field of parallelization
- Articles answering the research questions
- Articles published in 2010 and later

For the researchers to be able to analyze an article we were interested only in studies written in English and available in full text. To focus on quality research only we included only articles peer reviewed and published in a journal or on a conference. To address only the recently used methods we included only articles, which were published in the year of 2010 and later.

Quality assessment criteria

When assessing the quality of selected papers we read through the articles and evaluated them in several aspects.

QA.1: Is the used algorithm clearly described?

QA.2: Is the correctness of used algorithm proved?

QA.3: Is the algorithm experimentally tested?

QA.4: Are other algorithms for the same task mentioned?

For the purposes of our research we were mostly interested in the algorithm description and its correctness. That is why we considered these two factors important for determining quality of articles. From our perspective an experimental testing of algorithm and comparison with other algorithms improved quality of particular research as well.

Data extraction process

After the selection of articles the data extraction process was initiated. For the purposes of extracting data from articles in a systematic and efficient way we created several data extraction questions, which helped the reviewers to focus on information important for answering the stated research questions. This also allowed to compare articles in a systematic and clear manner thus helping to get an overview on the studied area. The final data extraction questions were the following.

DQ.1: What is the used decomposition technique?

DQ.2: What is the used load balancing technique?

DQ.3: What is the task to be solved?

DQ.4: What is the representation of state space? (Tree/Graph)

DQ.5: What is the base sequential algorithm used?

DQ.6: What is the best case scenario speed-up?

DQ.7: What is the worst case scenario speed-up?

Selected articles were deeply studied and answers to data extraction questions were written down.

INCLUDING AND EXCLUDED STUDIES

After defining the searching strategy we initiated the searching process. In the first stage we used the final searching string and collected 296 articles. Then we applied the inclusion and exclusion criteria sequentially in a few stages. Table 1 describes this process in more detail. Afterwards the researchers started reading the selected articles in more detail and assessed corresponding quality using quality assessment criteria described in the previous section. The quality assessment process was conducted by both researchers on all papers. All the articles were deemed of a good quality for our research.

RESULTS

With prepared list of studies to be reviewed we started the data extraction process as described in the research methodology section. We summarized the results of systematic data extraction in Table 2 and Table 3.

Table 2 shows data for DQ.3, DQ.4 and DQ.5. One of the first visible facts is that most of the problems solved in reviewed articles operate in a state space defined as a graph so in a more general structure than a tree.

Studies	**DQ.3: Problem**	**DQ.4: state space**	**DQ.5: base algorithm**
Laarman, A.; Langerak, R. 2011 [10]	LTL model checking	Graph	Nested Depth-First Search
Evangelista, S.; Petrucci, L. 2011 [14]	LTL model checking	Graph	Nested Depth-First Search
Evangelista, S; Laarman, A. 2012 [6]	LTL model checking	Graph	Nested Depth-First Search
Uelschen, M. 2013 [12]	The exact set cover problem	Tree	Knuth dancing links sequential algorithm
Yang Yuan 2011 [2]	Treewidth computation	Graph	Depth-first search algorithm
Xuan-Linh Ha; Thanh-Tho Quan 2013 [9]	LTL model checking	Graph	Tarjan's Algorithm
Azad, A. ; Halappanavar, M. 2012 [11]	matching of maximum cardinality in a bipartite graph	Graph	Depth-first Search algorithm
Kai Li Lim ; Kah Phooi Seng 2015 [16]	Uninformed pathfinding	Graph	Dijkstra's algorithm and IDDFS
Jian Cao; Sijie Caib; Yanhai Zhaoc 2010 [1]	Distributed constraint optimisation problem	Graph	Dynamic Programming Optimization (DPOP)
Igbal, S.M.Z.; Grahn, H.; Krasemann, J.T. 2012 [4]	Rescheduling the train schedule	Tree	Greedy DFS algorithm
Garnaud, E.; Hanusse, N. 2014 [13]	extracting functional dependencies of databases	Tree	SeqKey algorithm based on DFS
Fei Wang ; Jianqiang Dong ; Bo Yuan 2013 [5]	substructure pattern algorithms	Graph	gSpan algorithm
Baldwin, A. ; Asaithambi, A. 2011 [3]	finding a global minimum of a differentiable function	Tree	AZM algorithm
Laarman, A.; Olesen, M.C. 2013 [8]	Timed Automata LTL checking	Graph	nested depth-first search algorithm
Li Zeng; Jiefeng Cheng; Jintao Meng 2013 [7]	De Bruijn graph processing for genome assembly	Graph	traversal Depth-first search
Laarman, A.; Wijs, A. 2014 [15]	LTL model checking	Graph	Partial Order Reduction and NDFS-based search algorithms.

Table 2: Data extraction (part 1)

Studies	DQ.1: decomposition	DQ.4: state space	DQ.5: base algorithm
Laarman, A.; Langerak, R. 2011 [10]	on start with defined number of nodes	Graph	Nested Depth-First Search
Evangelista, S.; Petrucci, L. 2011 [14]	on start with defined number of nodes	Graph	Nested Depth-First Search
Evangelista, S; Laarman, A. 2012 [6]	on start with defined number of nodes	Graph	Nested Depth-First Search
Uelschen, M. 2013 [12]	to number of children on each step	Tree	Knuth dancing links sequential algorithm
Yang Yuan 2011 [2]	to number of children on each step	Graph	Depth-first search algorithm
Xuan-Linh Ha; Thanh-Tho Quan 2013 [9]	on start with defined number of nodes	Graph	Tarjan's Algorithm
Azad, A. ; Halappanavar, M. 2012 [11]	to number of children on each step	Graph	Depth-first Search algorithm
Kai Li Lim ; Kah Phooi Seng 2015 [16]	to number of children on each step	Graph	Dijkstra's algorithm and IDDFS
Jian Cao; Sijie Caib; Yanhai Zhaoc 2010 [1]	on start with defined number of nodes	Graph	Dynamic Programming Optimization (DPOP)
Igbal, S.M.Z.; Grahn, H.; Krasemann, J.T. 2012 [4]	fixed number of threads on a fixed stage	Tree	Greedy DFS algorithm
Garnaud, E.; Hanusse, N. 2014 [13]	on start with defined number of nodes	Tree	SeqKey algorithm based on DFS
Fei Wang ; Jianqiang Dong ; Bo Yuan 2013 [5]	to number of children on each step	Graph	gSpan algorithm
Baldwin, A. ; Asaithambi, A. 2011 [3]	on start with defined number of nodes	Tree	AZM algorithm
Laarman, A.; Olesen, M.C. 2013 [8]	on start with defined number of nodes	Graph	nested depth-first search algorithm
Li Zeng; Jiefeng Cheng; Jintao Meng 2013 [7]	on start with defined number of nodes	Graph	traversal Depth-first search
Laarman, A.; Wijs, A. 2014 [15]	on start with defined number of nodes	Graph	Partial Order Reduction and NDFS-based search algorithms.

Table 3: Data extraction (part 2)

Furthermore we can see that parallelized depth-first search algorithm is used in a number of articles for Linear Temporal Logic (LTL) model checking problem [10, 14, 6, 9, 8, 15]. In most of the studies some kind of depth-first search algorithm is used as a base for the parallel version. Typically it is a modification of depth-first search algorithm for purpose of the problem. As for example is often seen the

Nested Depth-First Search (NDFS) algorithm for LTL model checking problem or Iterative Deepening Depth-First Search (IDDFS) for the pathfinding problem [16]. However, some studies base their solution on a different kind of algorithm and extend it with a depth-first search principle in their parallel version [12].

Table 3 shows relation between decomposition methods, type of problem and achieved speed-up, data for DQ.1, DQ.3 and DQ.6. Results show that decomposition approaches are limited almost only to two solutions with one exception. In majority of studies the decomposition is made at the start of algorithm run and the number of computation nodes is usually set in advance and is the same throughout the whole runtime. Other commonly used option is to use available resources as much as possible by forking the computation to multiple nodes in runtime to the number of successors of currently explored graph node. In research by Igbal we can see a completely different approach, where the decomposition takes place on a previously defined depth of the tree and to a defined number of threads [4]. These parameters can be then optimized to the particular instance of problem. Except for that we can see values of speed-up for different algorithms, which in some cases reach relatively high numbers. However, these values were sometimes not present and they lack similar experimental conditions.

The values for data extraction question DQ.2 and DQ.7 were not included in any of the tables, because in most of the reviewed articles were these issues not discussed. From only a few values it would be hard to draw objective comparisons and final conclusions could be therefore misleading.

DISCUSSION

From the results of the review process we try to answer our research questions.

Regarding the RQ.1 recent research shows that the decomposition takes place either at the start of the algorithm or in exploration steps depending on available resources. Each of the approaches was used in 5 unique types of problems and so we can conclude that their usage is dependent only on the requirements of the problem. Only one study used a different decomposition technique that defines decomposition in strict step and on a strict number of nodes [4]. However, the later part of RQ.1 was not possible to answer from the gathered data as the information was incomplete.

To answer RQ.2 we gathered data about problems and possible speed-up values of parallelised algorithms used to solve these problems. We identified 11 unique problems, that are solved by parallel algorithms based on depth-first search. The LTL model checking problem was greatly discussed in 6 studies [10, 14, 6, 9, 8, 15]. This probably means that it is relatively important topic and there is a great interest in optimising available solutions. No evidence of relation between problems or space state representations and used parallelisation techniques was found. However, we see that all of the studies dealing with LTL model checking problem used the same decomposition technique.

Information about speed-up values are missing for some of the algorithms, but the present values show that speed-up to some degree is usually achieved. In some cases a significant improvement of runtime was achieved. Baldwin managed to measure speed-up of 180 on 80 cores in global minimum finding problem [3]. And for the LTL model checking problem Laarman decreased the runtime of algorithm 43 times by combined approach of parallel depth-first search and Partial-Order Reduction [15].

LIMITATIONS

In the first place, our study was strictly limited by time, as it was conducted as a part of university course, which lasted less than 2 months. This left us with only short time to deeply read and understand articles and the scope of literature review was limited.

As we limited our search to Inspec database only we could have missed some articles, which are not mapped by Inspec. Furthermore the total number of articles reviewed was 16 and it is therefore hard to objectively generalize conclusions about the parallelization techniques usage.

Another validity threat is that research was conducted in university environment by students, who lack deep knowledge in the field of parallelization. Their experience in conducting research was limited and thus interpretation of articles and drawn conclusions may suffer from some novice mistakes.

CONCLUSION

In our literature review we gathered recent data about research incorporating depth-first search algorithm. We summarized used decomposition techniques and found that there is no strictly dominating technique, which would be used on a majority of problems. We did not manage to objectively summarize load balancing techniques as they were only briefly discussed or not described at all.

Furthermore we created an overview on currently studied problems, where depth-first search has an use. This includes a variety of problems such as train schedule replanning [4], processing of genome assembly [7], extracting functional dependencies of databases [13] or treewidth computation [2]. One of the problems was more extensively studied in the last years. 6 articles focused on LTL model checking problem and improvement of existing algorithms, which leads us to believe that it has a great importance in the field of Computer Science. Most of the studies showed successful results and improved the current state-of-art solutions. In some cases the improvement was significant as speed-up 180 in [3] or the linear speed-up in [13].

We believe that our research shows that there is a great potential in parallelisation of algorithms and that it requires further research also in for other problems. Also we showed that despite the inherent sequential nature of depth-first algorithm it is possible to find parallelisation methods and achieve significant increase in performance and reduce the runtime of algorithms.

As a goal of the future research we point to LTL model checking problem. Particularly checking of liveness LTL property, which in automaton means that "eventually something good happens", is a problem hard for parallelisation. In reviewed research we found algorithms based on two different principles. First is based on nested DFS algorithm by Laarman [10]. The second one is based on Strong Connected Components and Tarjan's algorithm by Xuan-Linh Ha [9].

We believe that a rigorous experimental comparison of these two algorithms could bring more light on a way, which the research should follow after. For this research we propose a research question: "What is the most time efficient algorithm for LTL model checking problem?". An experiment run on a number of different settings and instances of problem can determine, which of the concurrently developing approaches deserve more attention by researcher dealing with LTL model checking problem.

REFERENCES

[1] Jian Cao, Sijie Cai, and Yanhai Zhao, "A Distributed Asynchronous Constraint Optimization Algorithm Based on Dynamic Reduction of Constraint Graph," in *2010 IEEE/ACM International Conference on Web Intelligence-Intelligent Agent Technology (WI-IAT 2010), 31 Aug.-3 Sept. 2010*, 2010, vol. vol.2, pp. 365–9.

[2] Yang Yuan, "A Fast Parallel Branch and Bound Algorithm for Treewidth," in *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence (ICTAI 2011), 7-9 Nov. 2011*, 2011, pp. 472–9.

[3] A. Baldwin and A. Asaithambi, "An Efficient Method for Parallel Interval Global Optimization," in *2011 International Conference on High Performance Computing & Simulation, 4-8 July 2011*, 2011, pp. 317–21.

[4] S. M. Z. Igbal, H. Grahn, and J. T. Krasemann, "A parallel heuristic for fast train dispatching during railway traffic disturbances: early results," in *1st International Conference on Operations Research and Enterprise Systems (ICORES 2012), 4-6 Feb. 2012*, 2012, pp. 405–14.

[5] Fei Wang, Jianqiang Dong, and Bo Yuan, "Graph-based Substructure Pattern Mining Using CUDA Dynamic Parallelism," in *Intelligent Data Engineering and Automated Learning - IDEAL 2013. 14th International Conference, IDEAL 2013, 20-23 Oct. 2013*, 2013, pp. 342–9.

[6] S. Evangelista, A. Laarman, L. Petrucci, and J. van de Pol, "Improved Multi-Core Nested Depth-First Search," in *Automated Technology for Verification and Analysis. 10th International Symposium, ATVA 2012, 3-6 Oct. 2012*, 2012, pp. 269–83.

[7] Li Zeng, Jiefeng Cheng, Jintao Meng, Bingqiang Wang, and Shengzhong Feng, "Improved Parallel Processing of Massive De Bruijn Graph for Genome Assembly," in *Web Technologies and Applications. 15th Asia-Pacific Web Conference, APWeb 2013, 4-6 April 2013*, 2013, pp. 96–107.

[8] A. Laarman, M. C. Olesen, A. E. Dalsgaard, K. G. Larsen, and J. van de Pol, "Multi-core Emptiness Checking of Timed Buchi Automata Using Inclusion Abstraction," in *Computer Aided Verification. 25th International Conference, CAV 2013, 13-19 July 2013,* 2013, pp. 968–83.

[9] Xuan-Linh Ha, Thanh-Tho Quan, Yang Liu, and Jun Sun, "Multi-core Model Checking Algorithms for LTL Verification with Fairness Assumptions," in *2013 20th Asia-Pacific Software Engineering Conference (APSEC), 2-5 Dec. 2013*, 2013, vol. vol.1, pp. 547–52.

[10] A. Laarman, R. Langerak, J. van de Pol, M. Weber, and A. Wijs, "Multi-core Nested Depth-First Search," in *Automated Technology for Verification and Analysis. 9th International Symposium, ATVA 2011, 11-14 Oct. 2011*, 2011, pp. 321–35.

[11] A. Azad, M. Halappanavar, S. Rajamanickam, E. G. Boman, A. Khan, and A. Pothen, "Multithreaded Algorithms for Maximum Matching in Bipartite Graphs," in *2012 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), 21-25 May 2012*, 2012, pp. 860–72.

[12] M. Uelschen, "Parallelization of the dancing links algorithm," in *2013 2nd International Conference on Information Management in the Knowledge Economy, 19-20 Dec. 2013*, 2013, pp. 172–5.

[13] E. Garnaud, N. Hanusse, S. Maabout, and N. Novelli, "Parallel mining of dependencies," in *2014 International Conference on High Performance Computing & Simulation (HPCS), 21-25 July 2014*, 2014, pp. 491–8.

[14] S. Evangelista, L. Petrucci, and S. Youcef, "Parallel nested depth-first searches for LTL model checking," in *Automated Technology for Verification and Analysis. 9th International Symposium, ATVA 2011, 11-14 Oct. 2011*, 2011, pp. 381–96.

[15] A. Laarman and A. Wijs, “Partial-Order Reduction for Multi-core LTL Model Checking,” in *Hardware and Software: Verification and Testing. 10th International Haifa Verification Conference, HVC 2014, 18-20 Nov. 2014*, 2014, pp. 267–83.

[16] Kai Li Lim, Kah Phooi Seng, Lee Seng Yeong, Li-Minn Ang, and Sue Inn Ch’ng, “Uninformed pathfinding: a new approach,” *Expert Systems with Applications*, vol. 42, no. 5, pp. 2722–30, Apr. 2015.

[17] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, “Preliminary guidelines for empirical research in software engineering,” *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721–734, Aug. 2002.

[18] T. Rauber and G. Rünger: “Parallel Programming for Multicore and Cluster Systems, 2nd ed.”