Faculty of Informatics
Masaryk University
Czech Republic

Platform-Dependent Verification

Habilitation Thesis
(Collection of Articles)

Jiří Barnat

October 2010

Abstract

The computer industry is undergoing a paradigm shift. Chip manufacturers are shifting development resources away from single-core chips to a new generation of multi-core or even many-core chips. Huge clusters of multi-core workstations are easily accessible everywhere, external memory devices, such as hard disks or solid state disks, are getting more powerful both in terms of capacity and access speed. This fundamental technological shift in core computing architectures requires a fundamental change in how we ensure the quality of software. The key issue is that the verification techniques need to undergo a similarly deep technological transition to catch up with the complexity of software designed for the new hardware. It is, therefore, inevitable to come up with new techniques that allow full exploitation of the power offered by the new computer hardware to make the automated verification techniques capable of handling next-generation computer systems. In particular, this thesis demonstrates how the automated formal verification procedures, such as explicit LTL model checking or decomposition of a directed graph into strongly connected components, can be adapted to employ the computational power of clusters, multi-cored workstations, disks or graphics processing units.

The thesis is conceived as a collection of articles. The collection contains thirteen technical papers published in journals or conference proceedings, and six tool papers describing software tools released under the supervision of the author of this thesis. The author contributed to the collection mainly by formulating the ideas of results published in the articles of the collection, but also by performing numerous analyses and interpretations of experimental measurements, by writing down significant parts of texts, and by implementing parts of released software tools.

Abstrakt

Počítačový průmysl prochází výraznou změnou výpočetního paradigmatu. Výrobci čipů se nadále nezaměřují na výrobu jednojaderných čipů, ale na výrobu vícejaderných nebo dokonce mnohojaderných čipů. I díky tomu jsou dnes běžně dostupné obrovské výpočetní klastry vícejaderných uzlů. Rostou také výkonostní parametry jako kapacita, nebo přístupová doba, všech externích paměťových médií. Tento fundamentální technologický posun v kvalitě výpočetních architektur sebou nese také posun ve způsobu, jakým je třeba zajišťovat kvalitu produkovaných výpočetních systémů. Klíčovým aspektem je zejména to, aby verifikační techniky podstoupily podobný technologický posun, a byly tak schopny zachytit komplexnost soudobých systémů. Je nezbytné vyvinout nové techniky, které umožní plně využít sílu soudobých a nadcházejících výpočetních systémů. V této habilitační práci je konkrétně demonstrováno, jakým způsobem je možné adaptovat techniky automatizované formální verifikace, jmenovitě proces ověřování modelu pro logiky lineárního času a proces dekompozice orientovaného grafu na silně souvislé komponenty tak, aby tyto techniky využily výpočetní síly klastrů, vícejaderných pracovních statnic, disků, nebo grafických karet.

Tato habilitační práce je koncipována jako soubor uveřejněných vědeckých prací (§72 odst. 3 písmena b zákona o vysokých školách). Soubor obsahuje třináct článků publikovaných v časopisech nebo konferenčních sbornících a šest článků popisujících programové nástroje, které vznikly pod supervizí autora této habilitační práce. Příspěvek autora do souboru uveřejněných prací tkví zejména ve formulaci publikovaných myšlenek, ale také v provádění nesčetných analýz a interpretací experimentálních měření, v psaní textu samotných článků a v implementaci částí zveřejněných softvérových nástrojů.

Acknowledgments

First of all, I would like to thank to Luboš Brim for being my supervisor. I would never be where I am without his guidance, support and the courage to start the parallel model checking topic. I appreciate all the fruitful discussions, tiring squash matches, and even rare quarrels we had.

I also wish to thank all my coauthors and acknowledge all the work they did. Especially, I thank all the students participating in the development of DiVinE tool and all its spin-offs and branches.

Many thanks should also go to my wife and our daughters for their endless patience and moral support.

Contents

I Commentary **1**

1 Introduction **3**
1.1 Motivation . . . 3
1.2 Focus of the Thesis . . . 4
1.3 Preliminaries . . . 5

2 State of the Art **7**
2.1 Parallel Model Checking . . . 7
2.1.1 State Space Generation . . . 7
2.1.2 Beyond State Space Generation . . . 7
2.1.3 Shared-Memory Architectures . . . 8
2.1.4 GPU Computing . . . 8
2.2 Parallel Symbolic Model Checking . . . 9
2.3 Embarrassingly Parallel Model Checking . . . 10
2.4 SCC Decomposition . . . 10
2.5 Model Checking with Disks . . . 11

3 Thesis Contribution **13**
3.1 Parallel and Distributed-Memory LTL Model Checking . . . 13
3.2 I/O Efficient Verification . . . 14
3.3 SCC Decomposition . . . 16
3.4 Verification of Probabilistic Systems . . . 16
3.5 Tools and Tool Papers . . . 17

4 Bibliography **19**

II Collection of Articles **31**

5 Journal and Conference Papers **33**

6 Tool Papers **35**

Part I

Commentary

Chapter 1

Introduction

1.1 Motivation

The computing power of computers has increased by a factor of a million over the past couple of decades. As a matter of fact, the development effort, both in industry and in academia, has gone into developing bigger, more powerful and more complex applications. Due to various factors such as continuing miniaturization, parallel and distributed computing, etc., we may still expect a similar rate of growth in the next few decades. With the increase in complexity of computer systems, it became even more important to develop formal methods for ensuring their quality and reliability. Various techniques for automated and semi-automated analysis and verification have been designed and successfully applied to small real-life systems. However, many of these techniques are computationally demanding and memory-intensive in general and their applicability to large and complex systems routinely seen in practice these days is limited. The major hampering factor is the *state space explosion problem* due to which large industrial models cannot be efficiently handled unless more sophisticated and scalable methods are used.

A lot of attention has been paid to the development of approaches to fight the state space explosion problem [58] in the field of automated formal verification [125]. Many techniques, such as a state compaction [75], compression [94], state space reduction [124, 56, 68], symbolic state space representation [45], etc., are used to reduce the memory requirements needed to handle the verification problem with a standard sequential software tool. Employing these techniques allows user to process larger systems with the same computing power. A complementary approach suggests to employ more computational power. To that end, various verification and analysis techniques that can efficiently utilize the power of combined hardware resources have been studied. Some of the techniques are general and applicable across a broad range of computing platforms, some of them are tailored to the specific capabilities of a particular hardware architectures. Examples include techniques to fight the memory limits with an efficient utilization of external memory devices [134], techniques that introduce cluster-based algorithms to employ the aggregate power of network-interconnected computers [133, 115, 73, 7], techniques to speed-up the verification process on multi-core processors [96, 14, 113], etc. An inevitable aspect of employing combined hardware resources is parallel processing. Unfortunately, it is not the

case that all the sequential solutions that are used for serial processing can be easily applied in parallel setting. On the contrary, many sequential solutions and algorithms are practically ineffective when used to utilize combined hardware resources. As a result, different solutions must have been and must be devised in order to facilitate parallel processing.

The idea of using combined resources to increase the computational power is far from being new. Attempts to use hard drives or parallel computers for verification of large systems have appeared in the very early years of the automated formal verification era. However, the inaccessibility of cheap parallel computers with sufficiently fast external memory devices together with the negative theoretical complexity results excluded these approaches from the main stream in formal verification. Moreover, thanks to the Moore's law, the performance of software tools kept improving continuously for years as the power of a single cored CPU grew. The situation changed dramatically with oncoming of multi core CPU chips. The progress in computer design over the past decades had measured several orders of magnitude with respect to various physical parameters such as power consumption, efficiency, physical size or cost. As a result, it became more efficient for chip producers to introduce multiple CPU cores on a single chip rather than to increase the speed of a single core. As the speed of a single core virtually stopped growing, every piece of software that was built upon a serial algorithm could not take the advantage of technological progress anymore. The focus of parallel and distributed-memory computing community shifted away from unique massively parallel systems competing for world records towards smaller and more cost effective systems built up from small and cheap personal computer parts. Suddenly, the need for parallel processing become rather general and wide spread in all science fields relying on complex computation operations, automated formal verification being not an exception.

Besides the parallel processing, the interest of formal verification community in specific hardware platforms has widen to graphics processing units and NVIDIA's CUDA technology, but also to contemporary external memory devices, such as solid state disks. As a matter of fact, the interest in the platform-dependent formal verification has been revived.

1.2 Focus of the Thesis

One particularly successful approach to automated formal verification is model checking [57, 5]. It builds upon an automated procedure that takes a model of a system and decides whether the model satisfies a given property or not. This thesis focuses, in particular, on platform dependent techniques and algorithms for model checking of formulas of *Linear Temporal Logic* (LTL) [127].

Articles included in the thesis describe results that allow implementation of scalable parallel LTL model checking algorithms. Upon the theoretical results presented in the thesis, software tools that are capable of efficient usage of aggregate computation resources of shared-memory and distributed-memory parallel architectures are presented. Thesis also describes new, the so called I/O efficient, algorithms for LTL model checking with external memory devices. Besides the LTL model checking prob-

lem, parallel algorithms for decomposing a directed graph into strongly connected components (SCCs) are described. SCC decomposition problem is inherently present in the core of many automated formal verification procedures. Finally, the platform-dependent verification of discrete nondeterministic systems is carried on to discrete probabilistic systems and systems with degradation.

1.3 Preliminaries

Given a model of a system, the model checking problem is to decide whether the model meets a given specification or not. For model checking purposes, the specification needs to be formalized by means of temporal logic, LTL in our case. An efficient automated procedure to decide LTL model checking problem has been introduced [138]. It employs the theory of automata over infinite words, in particular, it exploits the fact that every set of executions expressible by an LTL formula is an ω-regular set and as such can be described by a *Büchi automaton*. The approach suggests to express all the system executions by a *system automaton* and all the executions violating the given LTL formula by a *property* or *negative claim automaton*. These automata are combined into their synchronous product in order to check for the existence of system executions that violate the property. The language recognized by the *product automaton* is empty if and only if no system execution is invalid.

The language emptiness problem for Büchi automata can be expressed as an *accepting cycle detection problem* in a graph. Each Büchi automaton can be naturally identified with an *automaton graph* which is a directed graph $G = (V, E, s, A)$ where V is the set of vertices ($n = |V|$), E is a set of edges ($m = |E|$), s is an initial vertex, and $A \subseteq V$ is a set of accepting vertices. We say that a cycle in G is accepting if it contains an accepting vertex. Let $\mathcal{A}$ be a Büchi automaton and $G_{\mathcal{A}}$ the corresponding automaton graph. Then $\mathcal{A}$ recognizes a nonempty language iff $G_{\mathcal{A}}$ contains an accepting cycle reachable from s. The LTL model-checking problem is thus reduced to the accepting cycle detection problem in the automaton graph.

Optimal sequential algorithms for accepting cycle detection use depth-first search strategy. The individual algorithms differ in their space requirements, length of the counterexample produced, and other aspects [137]. The typical algorithm used is the *Nested DFS* algorithm [61]. The idea of the algorithm is to use two interleaved depth-first searches, where the first one discovers accepting states reachable from the initial state, while the second one – the nested, checks for a self-reachability of all accepting states revealed by the first (outer) search. Several modifications of the algorithm have been suggested to remedy some of its disadvantages [76]. The well known model checker built on the Nested DFS algorithm is model checker SPIN [94, 93, 132].

The optimality of the *Nested DFS* algorithm is achieved due to the particular order in which the graph is processed. The order guarantees that no vertices of the graph are visited more than twice. In fact, all the best-known algorithms rely on the same exploring principle, namely the *postorder* as computed by the depth-first search [60]. Unfortunately, deciding the postorder is P-complete problem [130] and as such it is inherently sequential, which means that any algorithmic solution relying on the depth-first search postorder will have difficulties to efficiently employ contemporary parallel

hardware architectures. A work-optimal scalable parallel algorithm for accepting cycle detection problem is unknown and, due to Reif [130], it is unlikely to exist.

An inseparable task of the model checking procedure is the so called *state space generation problem*. When specifying the system to be verified, the system is typically given by an initial configuration (initial state) and a function describing how the system evolves from one configuration into one or more succeeding ones. The is carried out by the so called next-state function. Such a definition of a system is referred to as an implicit definition. The state space generation problem is then a problem of enumerating all states (configurations) reachable from the initial state (initial configuration) using the next-state function. Performing the state space generation basically amounts to performing a graph traversal procedure. To guarantee termination for cyclic graphs, a graph traversal procedure keeps track of vertices (states) that have been traversed (generated). Due to the huge number of states (configurations) a system can reach, the state space generation procedure is time and memory demanding. The number of states a system can reach tends to grow exponentially with the size of the next-state function description. This is the so called *state space explosion* problem. Due to the state space explosion the amount of memory needed to store all reachable states for a real-life system typically exceeds the memory available to the algorithm, in which case the particular model checking procedure terminates incomplete. Verification approaches that are capable of detecting a violation of the verified property prior the full state space is generated are generally referred to as *on-the-fly* approaches.

Chapter 2

State of the Art

2.1 Parallel Model Checking

The need of parallel processing in automated formal verification stemmed from the desire to fight the state space explosion problem by employing aggregate memory of multiple network interconnected workstations. The crucial aspect studied at first was how to partition the state space (the set of visited states) among individual parts of the distributed-memory platform in order to take advantage of aggregate memory and parallel processing at the same time.

2.1.1 State Space Generation

Based on a parallel algorithm for state space generation [46] a static partitioning scheme relying on a hash function was suggested [52]. As observed by multiple researchers, the hash-based partitioning yields better space locality if only parts of the state descriptor are used as the input to the partitioning function. While there were approaches requiring the user of the tool to specify the concrete parts of the state descriptor to be used for partitioning [52, 115], other approaches employed automated or semi-automated techniques to do it [121, 122]. Techniques to load balance the set of visited states, also known as repartitioning techniques, have been suggested as well [2, 116, 111]. State space generation schemes employing probability aspects were also introduced [107, 106].

The first known public implementation of a distributed memory tool for verification of communication protocols was the parallel implementation of the Murφ tool [63, 133]. Active messages were used later on to improve the efficiency of the distributed-memory parallel processing with Murφ [141]. After the successful story of the Murφ tool, the distributed-memory state space generation appeared in many other verification tools, such as SPIN [115, 116], CADP [73], UPPALL [31], etc. Distributed-memory state space generation as a technique of automated formal verification also appeared in the context of Petri Nets [52, 88] and Markov chains [87, 86].

2.1.2 Beyond State Space Generation

The explicit model checking procedure is typically bound to linear time logic. Due to Vardi and Wolper [139], the LTL model checking problem reduces to the problem

of emptiness of Büchi automata, hence to the problem of accepting cycle detection in a directed graph. Several parallel and distributed-memory algorithms for accepting cycle detection were introduced. The first implementation [17] employed the so called dependency structure to record the reachability relation among accepting states of a distributed graph and applied the topological sort algorithm [105] to detect the presence of a self-reachable accepting state. Other parallel algorithms are built upon various ideas: negative cycle detection [43, 41], property automaton decomposition [8], symbolic SCC hull detection [47], value propagation [42], or back-level edges as produced by a breadth-first search procedure [9, 10]. According to experimental evaluations, practically the best algorithm to be used for parallel accepting cycle detection combines the ideas of symbolic SCC hull detection and value propagation [15].

Besides the LTL model checking, parallel and distributed-memory algorithms for other formal verification procedures were designed. Explicit parallel and distributed-memory algorithms for verification of μ-calculus [37, 38, 91] or alternation-free boolean equation systems [103] are known. Parallel explicit CTL model checking have been introduced as well [44, 40]. Techniques of state space reduction have been studied in the context of parallel processing as well. Approaches to reduce the state space modulo strong bisimulation were designed [34, 35] as well as a distributed-memory tool `LTSmin` to perform signature-based bisimulation reduction for strong and branching bisimulation [36]. Grid-enabled version of probabilistic model checker PRISM [112] has been reported too [143].

2.1.3 Shared-Memory Architectures

Most techniques and results known from the distributed-memory setting are straightforwardly applicable also to shared-memory architectures. However, scalability of distributed-memory solutions is often limited in shared-memory setting [12]. Therefore, shared-memory specific techniques have been developed to improve the efficiency and scalability of many parallel solutions leading in some cases almost to an optimal scalability [113]. The shared-memory specific techniques include, for example, shared communication data structures [98, 13], specific termination detection techniques [13], dual-core algorithms [96] or quite unique partitioning schemes [95].

2.1.4 GPU Computing

After NVIDIA's CUDA technology [62] was introduced, a lot of computational demanding task have been accelerated by GPU-aware algorithms. Examples of GPU accelerated procedures include, but are not limited to sorting procedure [77], reduce operation [85], or numerous biological and physical simulations, such as protein folding [101]. As for graph theory, successful adaptation of graph traversal algorithms were reported [83, 84] demonstrating the computational power of the CUDA device. Nevertheless, to achieve overall speedup in processing the graph to be traversed with a CUDA accelerated algorithm has to be stored in suitable data format, adjacency matrix for example.

The CUDA technology as a computing platform attracted also researches in the field of automated formal verification. The key challenge for which no satisfactory so-

lution is known yet is how to CUDA accelerate the generation of the state space graph from the implicit definition. Preliminary attempts to do so relate to explicit model checking approach. They suggest to employ massively parallel check for enabledness of transitions emanating from the states on the frontier of the search and massively parallel execution of all the enabled transitions [66, 67].

Once the state space is generated and represented in appropriate sparse matrix like structure, many verification tasks could be accelerated using CUDA technology. This has been successfully demonstrated, for example, for explicit LTL model checking [23, 22], or verification of probabilistic systems [39].

2.2 Parallel Symbolic Model Checking

Symbolic approach to model checking [104] is definitely one of the most important milestones achieved in automated formal verification. The key idea of the approach is to replace the space demanding explicit enumeration of the set of states by significantly more succinct representation, and at the same time, allow for traversing of multiple edges in the state space graph at once rather than handling them one by one as done in the explicit/enumerative approach. Both goals could be achieved if the set of visited states and the next-state functions are encoded using Binary Decision Diagrams (BDDs), see e.g. [57]. The model checking procedure than reduces to manipulation of BDD structures. Unlike the explicit approach, the size of a BDD does not necessarily grow with the number of states stored in the set represented with the BDD, but rather with the irregularity of the set. For regular set of states, as produced e.g. by synchronous systems, the symbolic approach is unbeatable, but for irregular state spaces as produced typically by asynchronous systems BDDs are not that efficient.

Symbolic model checking can be adapted to parallel processing in various ways. The first option is to run a serial model checking algorithm that calls to parallel BDD manipulation routines. Such parallel BDD manipulation approaches were successfully applied to accelerate operations over large BDDs [119, 129, 135].

The second approach to adapt the symbolic model checking procedure to parallel processing mimics the state space partitioning as known from the explicit approach. To that end BDD slicing was introduced [90, 32, 89]. The set of states is a priory partitioned according to the value of BDD control variables (BDD internal nodes) and the BDD is sliced into multiple BDDs that are maintained by individual computation participating workstations. The static partitioning was found inefficient because of the network communication overhead rendered necessarily even for small verification instances. Therefore, dynamic adaptive BDD slicing were introduced later on [80]. Still the model checking process did not exhibited the expected speed-up which was, as identified later, due to the synchronous execution of individual BDD operations. This has been overcome by introducing virtually asynchronous processing over distributed BDD slices [79] that lead to up to ten-fold speedup compared to the synchronous version.

A different approach to symbolic state space generation and model checking is saturation [53, 54]. The idea of it is to avoid encoding of the transition function with a decision diagram, and thus, avoid slightly unpredictable operations over the two

decision diagrams. Instead, the set of states reachable from a given set of states encoded by a BDD or an MDD (multi-valued decision diagram) is computed by direct manipulation of the internal nodes of the decision diagram representing the set of states reached so far. Unfortunately, the order in which the internal nodes of BDD or MDD are manipulated, is strictly given. The order resembles depth-first search postorder, hence, satisfactory scalable parallel technique to saturate a given BDD or MDD has not been found yet [55], some researchers even suggest to optimize the sequential algorithms rather than to parallelize them [71]. Nevertheless, horizontal partitioning [129] was employed for building up a parallel saturation procedure [49] that was improved later on with static [51] and dynamic pattern [50] for speculative execution of system transitions.

Beyond the state space generation, symbolic parallel approach to handle the verification of μ-calculus formulas has been introduced as well [78].

2.3 Embarrassingly Parallel Model Checking

The model checking task can be viewed as one big and computation demanding procedure that is a natural candidate for being solved by means of parallel processing. The parallel solutions mentioned so far introduce multiple parallel agents that process the input data and communicate intensively to achieve the desired goal. However, this is not the only option. The whole model checking procedure can be viewed also as a bunch of many independent tasks that can be executed solely in parallel, i.e. without any communication. Such a parallel solution is generally referred to as an embarrassingly parallel approach. The difference can be nicely demonstrated on the LTL model checking problem. While the classical parallel approaches suggest to employ multiple communicating agents to detect the presence of an accepting cycle in the directed graph, the embarrassingly parallel approach suggests to take individual system executions and check every single one for its conformance with the verified property. The number of executions of a system may, however, be infinite, which renders the embarrassingly parallel approach incomplete. Therefore, the embarrassingly parallel solutions could rather be viewed as fast bug finding techniques. Examples of embarrassingly parallel approaches include parallel randomized state space search [64] or parallel guided counter-example generation [131].

Regarding the LTL model checking procedure, the order in which the vertices of the product automaton graph are explored plays significant role provided the graph contains an error state or accepting cycle to be discovered. With good traverse order the discovery of an error is a matter of relatively small number of steps of the underlying algorithm. An embarrassingly parallel approach to LTL model checking that instanciates multiple standard sequential procedures in parallel each with a randomly modified order of exploration has been introduced [97].

2.4 SCC Decomposition

The problem of decomposition of a directed graph into its strongly connected components is a fundamental graph problem inherently present in many scientific and com-

mercial applications. The problem is defined as follows. Let G be a directed graph, i.e. G is a pair (V, E), where V is a set of vertices, and $E \subseteq V \times V$ is a set of edges. Let E^* be a transitive and reflexive closure of E and $s, t \in V$ two vertices. We say that vertex t *is reachable* from vertex s if $(s, t) \in E^*$. A set of vertices $C \subseteq V$ is *strongly connected*, if for any vertices $u, v \in C$, we have that v is reachable from u. A *strongly connected component* (SCC) is a *maximal* strongly connected $C \subseteq V$, i.e. such that no C' with $C \subsetneq C' \subseteq V$ is strongly connected. The problem of SCC decomposition is the problem of identification of all strongly connected components for a given graph.

As for the automated formal verification, the SCC decomposition problem is used as a subroutine in many algorithmic solutions. For example, the SCC decomposition algorithm is employed for verification of probabilistic systems, state space reduction by τ-confluence, verification of systems with fairness constraints, or verification of liner time properties given by other than Büchi automata. SCC-based algorithms can also be used directly for LTL model checking. While Nested DFS is more space efficient, SCC-based algorithms produce shorter counterexamples in general [69].

An efficient algorithmic solution to this problem is due to Tarjan [136], who showed that, given a graph with n vertices and m edges, it is possible to identify and list all strongly connected components of the graph in $O(n+m)$ time and $O(n)$ space. Unfortunately, the Tarjan's solution builds upon the depth-first search postorder and as such it is limited to sequential computing paradigms, hence inappropriate for contemporary parallel computing platforms. The existence of an work-optimal scalable parallel algorithm for SCC decomposition is an open problem. All the so far known parallel solutions to the problem exhibit unoptimal time complexity.

Different approaches suitable for parallel processing have been considered. See e.g. [74, 59, 3] for algorithm that works in $O(log^2 n)$ time, but requires $O(n^{2.376})$ parallel processors, or [142] for randomized parallel algorithm for the problem. Another parallel algorithm for SCC decomposition exploits the fact that it is possible to efficiently compute in parallel the set of vertices reachable from a certain vertex or set of vertices [72]. The general idea of the algorithm is to repeatedly pick a vertex of the graph and identify the component to which it belongs, by using the forward and a backward parallel reachability procedures. The algorithm proved to be efficient enough in practice, which resulted in several theoretical improvements of it [123, 117]. The worst time complexity of the algorithm is $O(n \cdot (n + m))$. Nevertheless, the algorithm exhibits $O(m \cdot log\, n)$ expected time [72]. A completely different strategy to detect SCC in parallel was introduced in [123]. The algorithm employs value forward value propagation to partition the graph into subgraphs respecting the SCCs. Each subgraph as computed by the algorithm is rooted, hence subsequent backward reachability identifies exactly the leading component of the subgraph. The algorithm performs well for graphs with many small components, however, for graphs with large components it is easily outperformed by other parallel algorithms.

2.5 Model Checking with Disks

Efficient usage of memory hierarchies is an established research topic [118]. Specialized algorithms were devised to efficiently utilize external-memory block devices. The

efficiency is of such the algorithms is typically measured using the so called I/O (input/output) complexity [1]. First of all, general graph traversal algorithms (state space generation, in the context of formal verification) were adapted to become I/O efficient. To that end the delayed duplicate detection was introduced [48] and further improved [144, 6, 81] or specialized for undirected graphs [108, 109].

Employing disk to fight the state explosion problem in formal verification has started by the disk extension of the verification tool Murφ [134, 126]. The external devices were also used to reconstruct the counterexample when applying the sweep-line heuristics search [110].

As for problems beyond the state space generation. First results published employ a generic reduction of model checking problem to the reachability problem [33]. Unfortunately, such a reduction resulted in a quadratic grow in the space demands, which effectively eliminated the possibility of complete search. There were heuristics used instead trying to prove the existence of a counterexample. We have seen random walks strategy [102], or iterative deepening and A^* algorithms to be used [99, 100]. Another incomplete model checking approach suggested builds on the fact that new transitions tend to lead to new states or to a states in recent breadth-first search levels [114].

The quadratic space overhead in the I/O efficient LTL model checking was avoided later on [24] and further improved by introducing the so called merge omissions [26] that allowed for more efficient delayed duplicate detection in the later stages of the computation. Various formulas for actual omissions were introduced [70]. A completely different technique for trading time for space has been suggested and is now referred to as the semi-external approach to LTL model checking problem [65].

A problem related to I/O efficient verification, delayed duplicate detection in particular, exists and is known as the streaming state space problem [92].

Chapter 3

Thesis Contribution

This habilitation thesis is conceived as a collection of articles. Summary of results achieved is given in four sections. Each section groups together results with a common research topic and lists the concrete percentage of contribution by the author of this thesis for each relevant article in the collection. An extra section is then devoted to the related software tools that were solely supervised and partly developed by the author of this thesis.

3.1 Parallel and Distributed-Memory LTL Model Checking

Achieved results

Distributed-Memory LTL Model Checking Parallel LTL model checker DiVinE [18] has been successfully adapted to various contemporary hardware platforms. Initially the tool was intended to aggregate computational power and system memory of multiple network interconnected workstations (clusters) in order to facilitate the verification of large model checking instances [17, 7]. We demonstrated that the tool succeeded the mission in terms of both the speedup achieved due to parallel processing and the ability of processing large model checking problem instances [140].

Shared-Memory LTL Model Checking In the light of technological shift towards shared-memory systems, we described relative advantages and disadvantages of shared versus private hash tables [29]. These were evaluated, both theoretically and practically, in a prototype implementation [14]. Later we have further improved the scalability of the tool and were able to demonstrate that the parallel processing even with an unoptimal algorithm outperforms highly efficient work-optimal sequential model checker SPIN [12].

On-the-fly Parallel Algorithm for LTL Model Checking Though, the optimality of the algorithm employed for parallel processing is an issue. There is an important subclass of LTL for which optimal scalable parallel algorithm exists [47]. However, this algorithm suffers from not being an on-the-fly algorithm. Since the on-the-fly verification is an important practical aspect, we have devised a modification of this algorithm that allows for on-the-fly verification in most verification instances [15].

CUDA Accelerated LTL Model Checking Finally, recent technological advancements in GPU computing made available a new rather specific computing platform – the NVIDIA's CUDA technology [62]. It allows for acceleration of computation intensive applications with GPU hardware. We have succeeded to adapt algorithms for accepting cycle detection to CUDA framework and demonstrated significant speedup of the LTL model checking process with CUDA technology [23].

Articles in Collection

[29] **J. Barnat** and P. Ročkai. Shared Hash Tables in Parallel Model Checking. *ENTCS*, 198(1):79–91, 2008.

Author's contribution: 50%, significant part of the writing, main idea.

[12] **J. Barnat**, L. Brim, and P. Ročkai. Scalable shared memory LTL model checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(2):139–153, 2010.

Author's contribution: 33%, significant part of the writing, analysis of experimental results and formulation of conclusions.

[140] K. Verstoep, H. Bal, **J. Barnat**, and L. Brim. Efficient Large-Scale Model Checking. In *23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS 2009)*. IEEE, 2009.

Author's contribution: 25%, DiVinE architecture consultant, marginal part of writing.

[15] **J. Barnat**, L. Brim, and P. Ročkai. A Time-Optimal On-the-Fly Parallel Algorithm for Model Checking of Weak LTL Properties. In *Formal Methods and Software Engineering (ICFEM 2009)*, volume 5885 of *LNCS*, pages 407–425. Springer, 2009.

Author's contribution: 70%, most of the writing, main idea, implementation, and experimental validation.

[23] **J. Barnat**, L. Brim, M. Češka, and T. Lamr. CUDA accelerated LTL Model Checking. In *15th International Conference on Parallel and Distributed Systems (ICPADS 2009)*, pages 34–41. IEEE Computer Society, 2009.

Author's contribution: 25%, most of the writing, main ideas.

3.2 I/O Efficient Verification

Achieved results

I/O Efficient LTL Model Checking Due to the state space explosion problem, the graph to be searched for the presence of an accepting cycle tends to be extremely large. For that reason the LTL model checking verification procedure suffers from limited applicability w.r.t the size of model checking instance if performed on a single workstation. Reduction techniques [57, 5] are simply not strong enough to solve

the problem. To move the frontier of still tractable systems a little bit further external memory devices (disks) are an option. We were first to show that the LTL model checking process can be done I/O efficiently with the same space complexity as the standard pure in memory solution [24].

Improved Delayed Duplicate Detection Technique The idea of LTL model checking with external memory devices is to keep the track of vertices that have been explored by the algorithm on the external memory. Unfortunately, in order to access the external memory efficiently, the standard work-flow of a graph traversal algorithm has to be modified. This modification is referred to as the *delayed duplicate detection* [108, 109, 120, 134]. According to our experimental measurements, the standard delayed duplicate detection technique becomes rather ineffective once the graph traversal procedure is about to complete the search. We have, therefore, defined an improved version of the work-flow and demonstrated its positive impact on I/O efficient verification [26]. Unfortunately, not all the parallel graph traversal algorithms that are suitable for in memory computing are compatible with our new work-flow modification. Hence, we have also defined a criterion for deciding the compliance of a graph traversal algorithm with our modification – the so called *revisiting resistance*.

Parallel I/O Efficient Model Checking We have also investigated how parallel disks can be combined to further improve the I/O efficient LTL model checking procedure [25] and whether the recent introduction of flash memory disks have some implications on the field of I/O efficient processing [11].

Articles in Collection

[24] **J. Barnat**, L. Brim, and P. Šimeček. I/O Efficient Accepting Cycle Detection. In *Computer Aided Verification*, volume 4590 of *LNCS*, pages 281–293. Springer, 2007.

Author's contribution: 33%, analyses of experimental results, significant part of writing.

[11] **J. Barnat**, L. Brim, S. Edelkamp, D. Sulewski, and P. Šimeček. Can Flash Memory Help in Model Checking. In *Formal Methods for Industrial Critical Systems (FMICS 2008)*, volume 5596 of *LNCS*, pages 150–165. Springer-Verlag, 2008.

Author's contribution: 25%, analyses of experimental results, formulation of conclusions, significant part of writing.

[26] **J. Barnat**, L. Brim, P. Šimeček, and M. Weber. Revisiting Resistance Speeds Up I/O-Efficient LTL Model Checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS).*, volume 4963 of *LNCS*, pages 48–62. Springer, 2008.

Author's contribution: 25%, revisiting resistant work-flow identification, analyses of experimental results, formulation of conclusions, significant part of writing.

[25] **J. Barnat**, L. Brim, and P. Šimeček. Cluster-Based I/O Efficient LTL Model Checking. In *24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)*, pages 635–639. IEEE Computer Society, 2009.

Author's contribution: 33%, analyses of experimental results, formulation of conclusions, significant part of writing.

3.3 SCC Decomposition

Achieved results

OBF Graph Decomposition Procedure We have developed a new parallel technique to partition a directed graph into multiple SCC respecting parts – the so called *OBF* technique [28]. The technique is unique as it can partition the graph into a number of subgraphs in linear time. A such it combines the good properties of the forward-backward strategy [72] that works in linear time but produces only a constant number of subgraphs, and the value propagation approach [123] that identifies a number of subgraphs, but requires quadratic time.

Recursive OBF Algorithm for Parallel SCC Decomposition The OBF technique has been further improved and used recursively to build a new standalone parallel algorithm for SCC decomposition – *Recursive OBF* [27]. According to our experimental evaluation over various types of directed graphs, the new algorithm outperforms all the known parallel SCC decomposition algorithms known so far.

Articles in Collection

[28] **Jiří Barnat** and Pavel Moravec. Parallel Algorithms for Finding SCCs in Implicitly Given Graphs. In *Formal Methods: Applications and Technology*, volume 4346 of *LNCS*, pages 316–330. Springer, 2006.

Author's contribution: 60%, OBF technique, complete writing.

[27] **J. Barnat**, J. Chaloupka, and J. Van De Pol. Distributed Algorithms for SCC Decomposition. *Journal of Logic and Computation Advance Access*, 2010.

Author's contribution: 50%, Recursive OBF algorithm idea.

3.4 Verification of Probabilistic Systems

Achieved results

Parallel Quantitative LTL Model Checking Quantitative analysis of probabilistic systems has been studied mainly from the global model checking point of view. In the global model checking problem, the goal of the verification is to decide the probability of satisfaction of a given property for all reachable states in the state space of the system under investigation. On the other hand, in the local model checking approach the probability of satisfaction is computed only for the set of initial states. We devised

parallel local model checking procedure and demonstrated that with the platform dependent local model checking procedure we were able to reduce the runtime needed for verification from days to minutes [20].

Degradation Concept The quantitative model checking procedure was extended also to the systems with *degradation* [30]. Under some condition, systems with degradation may be viewed as the standard probabilistic systems – Markov Decision Processes (MDP's) [128]. Rather theoretical result we obtained was that the degradation properties can distinguish probabilistic systems (MDP's) that are indistinguishable by means of the standard probabilistic logics such as LTL, PCTL [82] or PCTL* [4].

Articles in Collection

[20] **J. Barnat**, L. Brim, I. Černá, M. Češka, and J. Tůmová. Local Quantitative LTL Model Checking. In *Formal Methods for Industrial Critical Systems (FMICS 2008)*, volume 5596 of *LNCS*, pages 53–68. Springer-Verlag, 2008.

Author's contribution: 20%, analyses of experimental results, algorithmics, formulation of conclusions, significant part of writing.

[30] **J. Barnat**, I. Černá, and J. Tůmová. Quantitative Model Checking of Systems with Degradation. In *Proceeding of the Sixth International Conference on Quantitative Evaluation of Systems (QEST 2009)*, pages 21–30. IEEE, 2009.

Author's contribution: 33%, concept of degradation, relation to probabilistic systems, writing.

3.5 Tools and Tool Papers

In this section we describe software tools that were solely supervised and partly developed by the author of this thesis.

DiVinE, DiVinE Cluster LTL model checker built over the MPI standard allowing efficient utilization of computational resources of a cluster of workstations (*Obsolete*).

DiVinE-MC Clone of DiVinE dedicated for usage solely on multi-cored CPUs with shared memory architecture (*Obsolete*).

DiVinE-CUDA Clone of DiVinE dedicated for usage with NVIDIA's CUDA technology on workstations with appropriate graphics processing units.

DiVinE 2.x New implementation of parallel LTL model checker with the combined capabilities of previous DiVinE versions. With the release of DiVinE 2.x tool DiVinE and DiVinE-MC became obsolete.

ProbDiVinE Tool for qualitative model checking of probabilistic systems capable of employing aggregate computational power of a cluster of workstations.

ProbDiVinE-MC Tool for quantitative model checking of probabilistic systems capable of efficient utilization of multiple cores on a shared-memory platform.

Articles in Collection

[18] **J. Barnat**, L. Brim, I. Černá, P. Moravec, P. Ročkai, and P. Šimeček. DiVinE – A Tool for Distributed Verification (Tool Paper). In *Computer Aided Verification*, volume 4144/2006 of *LNCS*, pages 278–281. Springer Berlin / Heidelberg, 2006.

Author.'s contribution: 40%, Overall concept of the tool, implementation of multiple parallel accepting cycle detection algorithms.

[14] J. Barnat, L. Brim, and P. Ročkai. DiVinE Multi-Core – A Parallel LTL Model-Checker. In *Automated Technology for Verification and Analysis (ATVA 2008)*, volume 5311 of *LNCS*, pages 234–239. Springer, 2008.

Author's contribution: 33%, LTL model checking algorithmics.

[22] J. Barnat, L. Brim, and M. Češka. DiVinE-CUDA: A Tool for GPU Accelerated LTL Model Checking. *Electronic Proceedings in Theoretical Computer Science (PDMC 2009)*, 14:107–111, 2009.

Author's contribution: 40%, algorithmics, algorithm engineering.

[16] Jiří Barnat, Luboš Brim, and Petr Ročkai. DiVinE 2.0: High-Performance Model Checking. In *2009 International Workshop on High Performance Computational Systems Biology (HiBi 2009)*, pages 31–32. IEEE Computer Society Press, 2009.

Author's contribution: 33%, tool road-map, processing of precompiled models.

[19] J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. ProbDiVinE: A Parallel Qualitative LTL Model Checker. In *Fourth International Conference on the Quantitative Evaluation of Systems (QEST'07)*, pages 215–216. IEEE Computer Society, 2007.

Author's contribution: 20%, algorithmics, tool distribution.

[21] J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. Probdivine-mc: Multi-core ltl model checker for probabilistic systems. In *QEST '08: Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems*, pages 77–78, Washington, DC, USA, 2008. IEEE Computer Society.

Author's contribution: 20%, algorithmics.

Bibliography

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[2] S. Allmaier, S. Dalibor, and D. Kreische. Parallel Graph Generation Algorithms for Shared and Distributed Memory Machines. In G. Bilardi, A. G. Ferreira, R. Lüling, and J. D. P. Rolim, editors, *Proceeding of the Parallel Computing Conference PARCO'97 (Bonn, Germany)*, volume 1253 of *LNCS*, pages 207–218. Springer, 1997.

[3] N. Amato. Improved Processor Bounds for Parallel Algorithms for Weighted Directed Graphs. *Information Processing Letters*, 45(3):147–152, 1993.

[4] Adnan Aziz, Vigyan Singhal, Felice Balarin, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *Proceedings of the 7th International Conference on Computer Aided Verification*, pages 155–165, London, UK, 1995. Springer-Verlag.

[5] Ch. Baier and J. P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[6] Tonglaga Bao and Michael Jones. Time-efficient model checking with magnetic disk. In *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, pages 526–540. Springer, 2005.

[7] J. Barnat. *Distributed Memory LTL Model Checking*. PhD thesis, Masaryk University Brno, Faculty of Informatics, 2004.

[8] J. Barnat, L. Brim, and I. Černá. Property driven distribution of Nested DFS. In *Proc. Workshop on Verification and Computational Logic*, number DSSE-TR-2002-5 in DSSE Technical Report, pages 1–10. University of Southampton, UK, 2002.

[9] J. Barnat, L. Brim, and J. Chaloupka. Parallel Breadth-First Search LTL Model-Checking. In *18th IEEE International Conference on Automated Software Engineering (ASE'03)*, pages 106–115. IEEE Computer Society, Oct. 2003.

[10] J. Barnat, L. Brim, and J. Chaloupka. From Distributed Memory Cycle Detection to Parallel LTL Model Checking. *Electronic Notes in Theoretical Computer Science*, 133(1):21–39, 2005.

[11] J. Barnat, L. Brim, S. Edelkamp, D. Sulewski, and P. Šimeček. Can Flash Memory Help in Model Checking. In *Formal Methods for Industrial Critical Systems (FMICS 2008)*, volume 5596 of *LNCS*, pages 150–165. Springer-Verlag, 2008.

[12] J. Barnat, L. Brim, and P. Ročkai. Scalable shared memory LTL model checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(2):139–153, 2010.

[13] J. Barnat, L. Brim, and P. Ročkai. Scalable multi-core ltl model-checking. In *Model Checking Software*, volume 4595 of *LNCS*, pages 187–203. Springer, 2007.

[14] J. Barnat, L. Brim, and P. Ročkai. DiVinE Multi-Core – A Parallel LTL Model-Checker. In *Automated Technology for Verification and Analysis (ATVA 2008)*, volume 5311 of *LNCS*, pages 234–239. Springer, 2008.

[15] J. Barnat, L. Brim, and P. Ročkai. A Time-Optimal On-the-Fly Parallel Algorithm for Model Checking of Weak LTL Properties. In *Formal Methods and Software Engineering (ICFEM 2009)*, volume 5885 of *LNCS*, pages 407–425. Springer, 2009.

[16] J. Barnat, L. Brim, and P. Ročkai. DiVinE 2.0: High-Performance Model Checking. In *2009 International Workshop on High Performance Computational Systems Biology (HiBi 2009)*, pages 31–32. IEEE Computer Society Press, 2009.

[17] J. Barnat, L. Brim, and J. Stříbrná. Distributed LTL Model-Checking in SPIN. In *Proc. SPIN Workshop on Model Checking of Software*, volume 2057 of *LNCS*, pages 200–216. Springer, 2001.

[18] J. Barnat, L. Brim, I. Černá, P. Moravec, P. Ročkai, and P. Šimeček. DiVinE – A Tool for Distributed Verification (Tool Paper). In *Computer Aided Verification*, volume 4144/2006 of *LNCS*, pages 278–281. Springer Berlin / Heidelberg, 2006.

[19] J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. ProbDiVinE: A Parallel Qualitative LTL Model Checker. In *Fourth International Conference on the Quantitative Evaluation of Systems (QEST'07)*, pages 215–216. IEEE Computer Society, 2007.

[20] J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. Local Quantitative LTL Model Checking. In *Formal Methods for Industrial Critical Systems (FMICS 2008)*, volume 5596 of *LNCS*, pages 53–68. Springer-Verlag, 2008.

[21] J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. ProbDiVinE-MC: Multi-core LTL Model Checker for Probabilistic Systems. In *QEST '08: Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems*, pages 77–78, Washington, DC, USA, 2008. IEEE Computer Society.

[22] J. Barnat, L. Brim, and M. Češka. DiVinE-CUDA: A Tool for GPU Accelerated LTL Model Checking. *Electronic Proceedings in Theoretical Computer Science (PDMC 2009)*, 14:107–111, 2009.

[23] J. Barnat, L. Brim, M. Češka, and T. Lamr. CUDA accelerated LTL Model Checking. In *15th International Conference on Parallel and Distributed Systems (ICPADS 2009)*, pages 34–41. IEEE Computer Society, 2009.

[24] J. Barnat, L. Brim, and P. Šimeček. I/O Efficient Accepting Cycle Detection. In *Computer Aided Verification*, volume 4590 of *LNCS*, pages 281–293. Springer, 2007.

[25] J. Barnat, L. Brim, and P. Šimeček. Cluster-Based I/O Efficient LTL Model Checking. In *24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)*, pages 635–639. IEEE Computer Society, 2009.

[26] J. Barnat, L. Brim, P. Šimeček, and M. Weber. Revisiting Resistance Speeds Up I/O-Efficient LTL Model Checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS).*, volume 4963 of *LNCS*, pages 48–62. Springer, 2008.

[27] J. Barnat, J. Chaloupka, and J. Van De Pol. Distributed Algorithms for SCC Decomposition. *To appear in Journal of Logic and Computation*, 2010.

[28] J. Barnat and P. Moravec. Parallel Algorithms for Finding SCCs in Implicitly Given Graphs. In *Formal Methods: Applications and Technology*, volume 4346 of *LNCS*, pages 316–330. Springer, 2006.

[29] J. Barnat and P. Ročkai. Shared Hash Tables in Parallel Model Checking. *ENTCS*, 198(1):79–91, 2008.

[30] J. Barnat, I. Černá, and J. Tůmová. Quantitative Model Checking of Systems with Degradation. In *Proceeding of the Sixth International Conference on Quantitative Evaluation of Systems (QEST 2009)*, pages 21–30. IEEE, 2009.

[31] G. Behrmann, T. S. Hune, and F. W. Vaandrager. Distributed timed model checking — how the search order matters. In *Proc. 12th Conference on Computer-Aided Verification CAV00*, volume 1855 of *LNCS*, pages 216–231. Springer, 2000.

[32] S. Ben-David, T. Heyman, O. Grumberg, and A. Schuster. Scalable Distributed On-the-fly Symbolic Model Checking. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *Proc. 3rd International Conference on Formal Methods in Computer-Aided Design (FMCAD'00), Austin, Texas*, volume 1954 of *LNCS*, pages 390–404, 2000.

[33] Armin Biere, Cyrille Artho, and Viktor Schuppan. Liveness checking as safety checking. *Electronic Notes in Computer Science*, 66(2), 2002.

[34] Stefan Blom and Simona Orzan. Distributed State Space Minimization. *Electr. Notes Theor. Comput. Sci.*, 80, 2003.

[35] Stefan Blom and Simona Orzan. A distributed algorithm for strong bisimulation reduction of state spaces. *STTT*, 7(1):74–86, 2005.

[36] Stefan Blom, Jaco van de Pol, and Michael Weber. LTSmin: Distributed and Symbolic Reachability. In *Computer Aided Verification*, volume 6174 of *LNCS*, pages 354–359. Springer, 2010.

[37] B. Bollig, M. Leucker, and M Weber. Parallel model checking for the alternation free mu-calculus. In T. Margaria and W. Yi, editors, *Proc. TACAS 2001*, volume 2031 of *LNCS*, pages 543–558. Springer, 2001.

[38] Benedikt Bollig, Martin Leucker, and Michael Weber. Local parallel model checking for the alternation-free mu-calculus. In *Proceedings of the 9th International SPIN Workshop on Model checking of Software (SPIN '02)*. Springer-Verlag Inc., 2002.

[39] D. Bosnacki, S. Edelkamp, and D. Sulewski. Efficient Probabilistic Model Checking on General Purpose Graphics Processors. In *Model Checking Software (SPIN 2009)*, volume 5578 of *LNCS*, pages 32–49. Springer, 2009.

[40] M. Bourahla. Distributed CTL model checking. *IEE Proceedings - Software*, 152(6):297–308, 2005.

[41] L. Brim, I. Černá, and L. Hejtmánek. Parallel Algorithms for Detection of Negative Cycles. Technical Report FIMU-RS-2003-04, Faculty of Informatics, Masaryk University Brno, 2003.

[42] L. Brim, I. Černá, P. Moravec, and J. Šimša. Accepting predecessors are better than back edges in distributed LTL model-checking. In *Formal Methods in Computer Aided Design (FMCAD)*, volume 4144 of *LNCS*, pages 352–366. Springer, 2004.

[43] L. Brim, I. Černá, P. Krčál, and R. Pelánek. Distributed LTL model checking based on negative cycle detection. In *Proc. of Foundations of Software Technology and Theoretical Computer Science (FST TCS 2001)*, volume 2245 of *LNCS*, pages 96–107. Springer, 2001.

[44] Lubos Brim and Jitka Žídková. Using Assumptions to Distribute Alternation Free [mu]-Calculus Model Checking. *ENTCS*, 89(1):17 – 32, 2003.

[45] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[46] S. Caselli, G. Conte, and P. Marenzoni. Parallel state space exploration for GSPN models. In G. de Michelis and M. Diaz, editors, *Applications and Theory of Petri Nets 1995*, volume 935 of *LNCS*, pages 181–200. Springer Verlag, 1995.

[47] I. Černá and R. Pelánek. Distributed explicit fair cycle detection. In Thomas Ball and Sriram K. Rajamani, editors, *Model Checking Software, 10th International SPIN Workshop*, volume 2648 of *LNCS*, pages 49–73. Springer-Verlag, 2003.

[48] Yi-Jen Chiang, Michael T. Goodrich, Edward F. Grove, Roberto Tamassia, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory graph algorithms. In *Symposium on Discrete Algorithms (SODA)*, pages 139–149. Society for Industrial and Applied Mathematics, 1995.

[49] Ming-Ying Chung and Gianfranco Ciardo. Saturation NOW. In *1st International Conference on Quantitative Evaluation of Systems (QEST 2004)*, pages 272–281. IEEE Computer Society, 2004.

[50] Ming-Ying Chung and Gianfranco Ciardo. A dynamic firing speculation to speedup distributed symbolic state-space generation. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*. IEEE, 2006.

[51] Ming-Ying Chung and Gianfranco Ciardo. A Pattern Recognition Approach for Speculative Firing Prediction in Distributed Saturation State-Space Generation. *Electr. Notes Theor. Comput. Sci.*, 135(2):65–80, 2006.

[52] G. Ciardo, J. Gluckman, and D.M. Nicol. Distributed State Space Generation of Discrete-State Stochastic Models. *INFORMS Journal on Computing*, 10(1):82–93, 1998.

[53] Gianfranco Ciardo, Gerald Lüttgen, and Radu Siminiceanu. Saturation: An Efficient Iteration Strategy for Symbolic State-Space Generation. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, volume 2031 of *LNCS*, pages 328–342. Springer, 2001.

[54] Gianfranco Ciardo and Andy Jinqing Yu. Saturation-Based Symbolic Reachability Analysis Using Conjunctive and Disjunctive Partitioning. In *Correct Hardware Design and Verification Methods (CHARME 2005)*, volume 3725 of *LNCS*, pages 146–161. Springer, 2005.

[55] Gianfranco Ciardo, Yang Zhao, and Xiaoqing Jin. Parallel symbolic state-space exploration is difficult, but what is the alternative? *CoRR*, abs/0912.2785, 2009.

[56] E. M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Form. Methods Syst. Des.*, 9(1-2):77–104, 1996.

[57] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[58] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Progress on the State Explosion Problem in Model Checking. In R. Wilhelm, editor, *Informatics - 10 Years Back. 10 Years Ahead*, volume 2000 of *LNCS*, pages 176–194. Springer, 2001.

[59] R. Cole and U. Vishkin. Faster Optimal Parallel Prefix Sums and List Ranking. *Information and Computation*, 81(3):334–352, 1989.

[60] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. McGraw-Hill Science/Engineering/Math, July 2001.

[61] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory-Efficient Algorithms for the Verification of Temporal Properties. *Formal Methods in System Design*, 1:275–288, 1992.

[62] NVIDIA CUDA Compute Unified Device Architecture - Programming Guide Version 2.0,, 2009. `http://www.nvidia.com/object/cuda_develop.html`, June 2009.

[63] David L. Dill. The murφ verification system. In *Conference on Computer-Aided Verification (CAV '96)*, Lecture Notes in Computer Science, pages 390–393. Springer-Verlag, July 1996.

[64] Matthew B. Dwyer, Sebastian G. Elbaum, Suzette Person, and Rahul Purandare. Parallel Randomized State-Space Search. In *International Conference on Software Engineering (ICSE 2007)*, pages 3–12. IEEE Computer Society, 2007.

[65] Stefan Edelkamp, Peter Sanders, and Pavel Šimeček. Semi-external LTL model checking. In *CAV '08: Proc. of the 20th international conference on Computer Aided Verification*, pages 530–542, Berlin, Heidelberg, 2008. Springer.

[66] Stefan Edelkamp and Damian Sulewski. Model Checking via Delayed Duplicate Detection on the GPU. Technical Report Technical Report 821, TU Dortmund, 2008. Presented on the 22nd Workshop on Planning, Scheduling, and Design PUK 2008.

[67] Stefan Edelkamp and Damian Sulewski. Parallel State Space Search on the GPU. In *International Symposium on Combinatorial Search (SoCS 2009)*, 2009.

[68] E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Form. Methods Syst. Des.*, 9(1-2):105–131, 1996.

[69] J. Esparza and S. Schwoon. A note on on-the-fly verification algorithms. In *TACAS'05*, volume 3440 of *LNCS*, pages 174–190. Springer, 2005.

[70] Sami Evangelista. Dynamic delayed duplicate detection for external memory model checking. In *SPIN '08: Proc. of the 15th international workshop on Model Checking Software*, pages 77–94, Berlin, Heidelberg, 2008. Springer.

[71] Jonathan Ezekiel, Gerald Luttgen, and Radu Siminiceanu. To Parallelize or to Optimize? *Advance access of Journal of Logic and Computation*, page exp006, 2009.

[72] L. K. Fleischer, B. Hendrickson, and A. Pinar. On Identifying Strongly Connected Components in Parallel. In *Parallel and Distributed Processing*, volume 1800 of *LNCS*, pages 505–511. Springer, 2000.

[73] H. Garavel, R. Mateescu, and I.M Smarandache. Parallel State Space Construction for Model-Checking. In Matthew B. Dwyer, editor, *Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN'2001)*, volume 2057 of *LNCS*, pages 216–234, Toronto, Canada, 2001. Springer-Verlag.

[74] H. Gazit and G. L. Miller. An Improved Parallel Algorithm That Computes the BFS Numbering of a Directed Graph. *Information Processing Letters*, 28(2):61–65, 1988.

[75] Jaco Geldenhuys and P. J. A. de Villiers. Runtime efficient state compaction in SPIN. In *SPIN*, pages 12–21, 1999.

[76] Jaco Geldenhuys and Antti Valmari. Tarjan's algorithm makes on-the-fly LTL verification more efficient. In *TACAS'04*, volume 2988 of *LNCS*, pages 205–219. Springer, 2004.

[77] Naga K. Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. GPUTeraSort: high performance graphics co-processor sorting for large database management. In *International Conference on Management of Data (SIGMOD 06)*, pages 325–336. ACM, 2006.

[78] O. Grumberg, T. Heyman, and A. Schuster. Distributed Model Checking for μ-calculus. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proc. 13th Conference on Computer-Aided Verification CAV01*, volume 2102 of *LNCS*, pages 350–362. Springer, 2001.

[79] Orna Grumberg, Tamir Heyman, Nili Ifergan, and Assaf Schuster. Achieving speedups in distributed symbolic reachability analysis through asynchronous computation. In *Correct Hardware Design and Verification Methods (CHARME 2005)*, volume 3725 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2005.

[80] Orna Grumberg, Tamir Heyman, and Assaf Schuster. A work-efficient distributed algorithm for reachability analysis. *Formal Methods in System Design*, 29(2):157–175, 2006.

[81] Moritz Hammer and Michael Weber. "To Store or Not To Store" Reloaded: Reclaiming Memory on Demand. In Luboš Brim, Boudewijn R. Haverkort, Martin Leucker, and Jaco van de Pol, editors, *FMICS/PDMC*, volume 4346 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2006.

[82] Hans Hansson and Bengt Jonsson. A Framework for Reasoning about Time and Reliability. In *IEEE Real-Time Systems Symposium*, pages 102–111, 1989.

[83] P. Harish and P. J. Narayanan. Accelerating Large Graph Algorithms on the GPU Using CUDA. In *HiPC*, volume 4873 of *LNCS*, pages 197–208. Springer, 2007.

[84] P. Harish, V. Vineet, and P. J. Narayanan. Large Graph Algorithms for Massively Multithreaded Architectures. Technical Report IIIT/TR/2009/74, Center for Visual Information Technology, International Institute of Information Technology Hyderabad, INDIA, 2009.

[85] M. Harris. Optimizing Parallel Reduction in CUDA,. `http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf`, March 2010.

[86] B. R. Haverkort, A. Bell, and H. C. Bohnenkamp. On the efficient sequential and distributed generation of very large markov chains from stochastic petri nets. In *Proc. 8th Int. Workshop on Petri Net and Performance Models (PNPM'99), 8-10 October 1999, Zaragoza, Spain*, pages 12–21. IEEE Computer Society Press, 1999.

[87] Boudewijn R. Haverkort, Henrik Bohnenkamp, and Alexander Bell. Efficiency improvements in the evaluation of large stochastic petri nets. In Desel, J., Kemper, P., Kindler, E., and Oberweis, A., editors, *Forschungsbericht: 5. Workshop Algorithmen und Werkzeuge für Petrinetze*, pages 55–61. Universität Dortmund,

Fachbereich Informatik, 1998. Published as Forschungsbericht: 5. Workshop Algorithmen und Werkzeuge für Petrinetze, number 694.

[88] Keijo Heljanko, Victor Khomenko, and Maciej Koutny. Parallelisation of the Petri Net Unfolding Algorithm. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, volume 2280 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 2002.

[89] Tamir Heyman, Daniel Geist, Orna Grumberg, and Assaf Schuster. A Scalable Parallel Algorithm for Reachability Analysis of Very Large Circuits. *Formal Methods in System Design*, 21(3):317–338, 2002.

[90] Tamir Heyman, Danny Geist, Orna Grumberg, and Assaf Schuster. Achieving scalability in parallel reachability analysis of very large circuits. In O. Grumberg, editor, *Computer Aided Verification, 12th International Conference*, volume 1855 of *LNCS*, pages 20–35. Springer, 2000.

[91] Fredrik Holmén, Martin Leucker, and Marcus Lindström. UppDMC – a distributed model checker for fragments of the μ-calculus. In Lubos Brim and Martin Leucker, editors, *Proc. of the 3rd Workshop on Parallel and Distributed Methods for Verification*, volume 128/3 of *Electronic Notes in Computer Science*. Elsevier Science Publishers, 2004.

[92] Viliam Holub and Petr Tůma. Streaming state space: A method of distributed model verification. In *1st Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering*, pages 356–368. IEEE Computer Society, 2007.

[93] Gerard J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997. Special issue on Formal Methods in Software Practice.

[94] Gerard J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

[95] Gerard J. Holzmann. A Stack-Slicing Algorithm for Multi-Core Model Checking. *ENTCS*, 198(1):3–16, 2008.

[96] Gerard J. Holzmann and Dragan Bosnacki. The design of a multicore extension of the spin model checker. *IEEE Trans. Software Eng.*, 33(10):659–674, 2007.

[97] Gerard J. Holzmann, Rajeev Joshi, and Alex Groce. Swarm Verification. In *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008)*, page 6 pages. IEEE, 2008.

[98] Cornelia P. Inggs and Howard Barringer. CTL* model checking on a shared-memory architecture. *Electronic Notes in Computer Science*, 128(3):107–123, 2005.

[99] Shahid Jabbar and Stefan Edelkamp. I/O efficient directed model checking. In *Proc. of 6th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2005)*, volume 3385 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 2005.

[100] Shahid Jabbar and Stefan Edelkamp. Parallel external directed model checking with linear I/O. In *Proc. of 7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2006) Charleston*, volume 3855 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2006.

[101] G. Jayachandran, V. Vishal, and V. S. Pande. Using massively parallel simulations and Markovian models to study protein folding: Examining the Villin head-piece. *Journal of Chemical Physics*, 124(6):903–914, 2006.

[102] Michael Jones and Eric Mercer. Explicit state model checking with Hopper. In *SPIN*, volume 2989 of *Lecture Notes in Computer Science*, pages 146–150. Springer, 2004.

[103] Christophe Joubert and Radu Mateescu. Distributed On-the-Fly Model Checking and Test Case Generation. In *Model Checking Software*, volume 3925 of *LNCS*, pages 126–145. Springer, 2006.

[104] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proc. of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.

[105] A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

[106] W. Knottenbelt, P.G Harrison, M. Mestern, and P.S. Kritzinger. A Probabilistic Dynamic Technique for the Distributed Generation of Very Large State Spaces. *Performance Evaluation*, 35(1–4):127–148, Feb 2000.

[107] W. Knottenbelt, M. Mestern, P.G Harrison, , and P.S. Kritzinger. Probability, parallelism and the state space exploration problem. In R. Puigjaner, editor, *Tools'98*, volume 1469 of *LNCS*, pages 165–179. Springer Verlag, 1998.

[108] R. Korf. Best-First Frontier Search with Delayed Duplicate Detection. In *AAAI'04*, pages 650–657. AAAI Press / The MIT Press, 2004.

[109] R. Korf and P. Schultze. Large-Scale Parallel Breadth-First Search. In *AAAI'05*, pages 1380–1385. AAAI Press / The MIT Press, 2005.

[110] Lars Michael Kristensen and Thomas Mailund. Efficient path finding with the sweep-line method using external storage. In *ICFEM*, volume 2885 of *Lecture Notes in Computer Science*, pages 319–337. Springer, 2003.

[111] Rahul Kumar and Eric G. Mercer. Load balancing parallel explicit state model checking. *Electronic Notes in Computer Science*, 128(3):19–34, 2005.

[112] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In *Proc. TACAS'02*, 2002.

[113] Alfons Laarman, Jaco van de Pol, and Michael Weber. Boosting Multi-Core Reachability Performance with Shared Hash Tables. In *Formal Methods in Computer Aided Design (FMCAD 2010). To Appear*, 2010.

[114] Peter Lamborn and Eric A. Hansen. Layered duplicate detection in external-memory model checking. In *SPIN '08: Proc. of the 15th international workshop on Model Checking Software*, pages 160–175, Berlin, Heidelberg, 2008. Springer.

[115] Flavio Lerda and Riccardo Sisto. Distributed-memory Model Checking with SPIN. In *Proc. of the 5th International SPIN Workshop*, volume 1680 of *LNCS*. Springer-Verlag, 1999.

[116] Flavio Lerda and Willem Visser. Addressing Dynamic Issues of Program Model Checking. In *Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN'2001)*, volume 2057 of *LNCS*, pages 80–102, Toronto, Canada, 2001. Springer-Verlag.

[117] W. McLendon III, B. Hendrickson, S. J. Plimpton, and L. Rauchwerger. Finding Strongly Connected Components in Distributed Graphs. *Journal of Parallel and Distributed Computing*, 65(8):901–910, 2005.

[118] Ulrich Meyer, Peter Sanders, and Jop Sibeyn, editors. *Algorithms for Memory Hierarchies*. Springer, 2003.

[119] Kim Milvang-Jensen and Alan J. Hu. BDDNOW: A Parallel BDD Package. In *Formal Methods in Computer-Aided Design (FMCAD '98)*, volume 1522 of *Lecture Notes in Computer Science*, pages 501–507. Springer, 1998.

[120] Kameshwar Munagala and Abhiram Ranade. I/O-complexity of graph algorithms. In *SODA '99: Proc. of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 687–694, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[121] D.M. Nicol and G. Ciardo. Automated Parallelization of Discrete State-space Generation. *Journal of Parallel and Distributed Computing*, 47(2):122–131, 1997.

[122] D.M. Nicol and G. Ciardo. Automated Parallelization of Discrete State-space Generation. Technical Report NASA/CR-2000-210082, NASA Langley Research Center, Hampton, USA, 2000.

[123] S. Orzan. *On Distributed Verification and Verified Distribution*. PhD thesis, Free University of Amsterdam, 2004.

[124] Doron Peled. Ten years of partial order reduction. In *Proceedings of the 10th International Conference on Computer Aided Verification*, pages 17–28. Springer-Verlag, 1998.

[125] R. Pelánek. Fighting State Space Explosion: Review and Evaluation. In *Formal Methods for Industrial Critical Systems (FMICS 2008)*, volume 5596 of *LNCS*, pages 37–52. Springer, 2009.

[126] Giuseppe Della Penna, Benedetto Intrigila, Enrico Tronci, and Marisa Venturini Zilli. Exploiting transition locality in the disk based Murφ verifier. In *FMCAD*, pages 202–219, 2002.

[127] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977.

[128] M. L. Puterman. *Markov Decision Processes-Discrete Stochastic Dynamic Programming*. John Wiley &Sons, New York, 1994.

[129] Rajeev K. Ranjan, Jagesh V. Sanghavi, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Binary decision diagrams on network of workstation. In *International Conference on Computer Design (ICCD '96)*, pages 358–364. IEEE Computer Society, 1996.

[130] John H. Reif. Depth-first search is inherently sequential. *Information Processing Letters*, 20(5):229–234, June 1985.

[131] N. Rungta and E. G. Mercer. Generating Counter-Examples Through Randomized Guided Search. In *Model Checking Software (SPIN 2007)*, volume 4595 of *LNCS*, pages 39–57. Springer, 2007.

[132] On-the-fly, LTL model checking with SPIN.
URL: `http://spinroot.com/`.

[133] U. Stern and D. L. Dill. Parallelizing the murφ verifier. In O. Grumberg, editor, *Proceedings of Computer Aided Verification (CAV '97)*, volume 1254 of *LNCS*, pages 256–267. Springer-Verlag, 1997.

[134] U. Stern and D. L. Dill. Using magnetic disk instead of main memory in the Murφ verifier. In *Computer Aided Verification. 10th International Conference*, pages 172–183, 1998.

[135] Tony Stornetta and Forrest Brewer. Implementation of an Efficient Parallel BDD Package. In *Proc. of Design Automation Conference (DAC'96)*, pages 641–644. ACM Press, 1996.

[136] Robert Tarjan. Depth first search and linear graph algorithms. *SIAM journal on computing*, pages 146–160, Januar 1972.

[137] M. Vardi. Automata-Theoretic Model Checking Revisited. In *VMCAI'07*, volume 4349 of *LNCS*, pages 137–150. Springer, 2007.

[138] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings 1st Annual IEEE Symp. on Logic in Computer Science, LICS'86, Cambridge, MA, USA, 16–18 June 1986*, pages 332–344. IEEE Computer Society Press, Washington, DC, 1986.

[139] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. IEEE Symposium on Logic in Computer Science*, pages 322–331. Computer Society Press, 1986.

[140] K. Verstoep, H. Bal, J. Barnat, and L. Brim. Efficient Large-Scale Model Checking. In *23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS 2009)*. IEEE, 2009.

[141] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active messages: a mechanism for integrated communication and computation. In *19th Annual International Symposium on Computer Architecture*, pages 256–266, 1992.

[142] S. Warren. Finding Strongly Connected Components in Parallel Using O(log2n) Reachability Queries. In *SPAA*, pages 146–151. ACM, 2008.

[143] Y. Zhang, D. Parker, and M. Kwiatkowska. Grid-enabled probabilistic model checking with PRISM. In *Proc. 4th All Hands Meeting Workshop (AHM'05)*, 2005.

[144] Rong Zhou and Eric A. Hansen. Structured duplicate detection in external-memory graph search. In *AAAI*, pages 683–689. AAAI Press / The MIT Press, 2004.

Part II

Collection of Articles

Chapter 5

Journal and Conference Papers

1. J. Barnat and P. Ročkai. Shared Hash Tables in Parallel Model Checking. *ENTCS*, 198(1):79–91, 2008.

2. J. Barnat, L. Brim, and P. Ročkai. Scalable shared memory LTL model checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(2):139–153, 2010.

3. K. Verstoep, H. Bal, J. Barnat, and L. Brim. Efficient Large-Scale Model Checking. In *23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS 2009)*. IEEE, 2009.

4. J. Barnat, L. Brim, and P. Ročkai. A Time-Optimal On-the-Fly Parallel Algorithm for Model Checking of Weak LTL Properties. In *Formal Methods and Software Engineering (ICFEM 2009)*, volume 5885 of *LNCS*, pages 407–425. Springer, 2009.

5. J. Barnat, L. Brim, M. Češka, and T. Lamr. CUDA accelerated LTL Model Checking. In *15th International Conference on Parallel and Distributed Systems (ICPADS 2009)*, pages 34–41. IEEE Computer Society, 2009.

6. Jiří Barnat and Pavel Moravec. Parallel Algorithms for Finding SCCs in Implicitly Given Graphs. In *Formal Methods: Applications and Technology*, volume 4346 of *LNCS*, pages 316–330. Springer, 2006.

7. J. Barnat, J. Chaloupka, and J. Van De Pol. Distributed Algorithms for SCC Decomposition. *Journal of Logic and Computation Advance Access*, 2010.

8. J. Barnat, L. Brim, and P. Šimeček. I/O Efficient Accepting Cycle Detection. In *Computer Aided Verification*, volume 4590 of *LNCS*, pages 281–293. Springer, 2007.

9. J. Barnat, L. Brim, S. Edelkamp, D. Sulewski, and P. Šimeček. Can Flash Memory Help in Model Checking. In *Formal Methods for Industrial Critical Systems (FMICS 2008)*, volume 5596 of *LNCS*, pages 150–165. Springer-Verlag, 2008.

10. J. Barnat, L. Brim, P. Šimeček, and M. Weber. Revisiting Resistance Speeds Up I/O-Efficient LTL Model Checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS).*, volume 4963 of *LNCS*, pages 48–62. Springer, 2008.

11. J. Barnat, L. Brim, and P. Šimeček. Cluster-Based I/O Efficient LTL Model Checking. In *24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)*, pages 635–639. IEEE Computer Society, 2009.

12. J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. Local Quantitative LTL Model Checking. In *Formal Methods for Industrial Critical Systems (FMICS 2008)*, volume 5596 of *LNCS*, pages 53–68. Springer-Verlag, 2008.

13. J. Barnat, I. Černá, and J. Tůmová. Quantitative Model Checking of Systems with Degradation. In *Proceeding of the Sixth International Conference on Quantitative Evaluation of Systems (QEST 2009)*, pages 21–30. IEEE, 2009.

Chapter 6

Tool Papers

1. J. Barnat, L. Brim, I. Černá, P. Moravec, P. Ročkai, and P. Šimeček. DiVinE – A Tool for Distributed Verification (Tool Paper). In *Computer Aided Verification*, volume 4144/2006 of *LNCS*, pages 278–281. Springer Berlin / Heidelberg, 2006.

2. J. Barnat, L. Brim, and P. Ročkai. DiVinE Multi-Core – A Parallel LTL Model-Checker. In *Automated Technology for Verification and Analysis (ATVA 2008)*, volume 5311 of *LNCS*, pages 234–239. Springer, 2008.

3. J. Barnat, L. Brim, and M. Češka. DiVinE-CUDA: A Tool for GPU Accelerated LTL Model Checking. *Electronic Proceedings in Theoretical Computer Science (PDMC 2009)*, 14:107–111, 2009.

4. J. Barnat, L. Brim, and P. Ročkai. DiVinE 2.0: High-Performance Model Checking. In *2009 International Workshop on High Performance Computational Systems Biology (HiBi 2009)*, pages 31–32. IEEE Computer Society Press, 2009.

5. J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. ProbDiVinE: A Parallel Qualitative LTL Model Checker. In *Fourth International Conference on the Quantitative Evaluation of Systems (QEST'07)*, pages 215–216. IEEE Computer Society, 2007.

6. J. Barnat, L. Brim, I. Černá, M. Češka, and J. Tůmová. Probdivine-mc: Multi-core ltl model checker for probabilistic systems. In *QEST '08: Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems*, pages 77–78, Washington, DC, USA, 2008. IEEE Computer Society.