Platform Dependent Verification: On Engineering Verification Tools for 21st Century[*]

Luboš Brim
Faculty of Informatics
Masaryk University
Brno, Czech Republic
brim@fi.muni.cz

Jiří Barnat
Faculty of Informatics
Masaryk University
Brno, Czech Republic
barnat@fi.muni.cz

1 Fighting the Space Explosion

With the increase in complexity and degree of parallelism of computer systems, it became even more important to develop formal methods for ensuring their quality. Correctness and reliability became a must have flavor for business success, and therefore, various techniques for automated and semi-automated formal verification and analysis have been designed and successfully applied. Formal verification and analysis bring many benefits such as early integration in design process, more effective detection of logic errors, etc. Even though introduction of formal analysis is rather costly, it pays off after all as it results in significant reduction in verification time as well as development costs and time-to-market. Attempts are being made to integrate formal verification techniques and tools with other design approaches to support engineering of complex industrial systems. The iFEST Artemisia project [59] is an example of a promising tools integration in embedded systems domain.

Model checking is a distinguished technique of formal verification of complex hardware and software designs. Founders of the technique Edmund M. Clarke jr. (CMU, USA), Allen E. Emerson (Texas at Austin, USA), and Joseph Sifakis (IMAG Grenoble, France) were awarded ACM Turing Award in 2007 *for their roles in developing model checking into a highly effective verification technology, widely adopted in the hardware and software industries*. Unfortunately, model checking procedure is computationally demanding and memory-intensive in general, hence, its applicability to large and complex systems routinely seen in practice these days is still 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 [37] in the field of automated formal verification [77]. Many techniques, such as state compaction [47], compression [56], state space reduction [76, 36, 44], symbolic state space representation [29], etc., were introduced to reduce the memory requirements needed to handle the verification problem with a standard sequential software tool. These techniques allowed system developers to verify larger systems without the need of increased computing power.

To verify even larger systems, however, no option was left out than to employ combined computing power of multiple computing devices. Unfortunately, some verification techniques cannot preserve their efficiency if adapted to non-sequential models of computation, and therefore an urgent need for new and quite different verification procedures emerged. Many new techniques have been introduced. Some of

[*]This research has been supported by the Czech Grant Agency grant No. GAP202/11/0312 and by Artemisia iFEST project grant No. 100203.

them are 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 [80], techniques that introduce cluster-based algorithms to employ the aggregate power of network-interconnected computers [79, 73, 46, 4], techniques to speed-up the verification process on multi-core processors [58, 11, 71], etc.

However, back at the beginning of the 21st century, many of these techniques waited to be yet discovered. Even at that time the idea of using combined resources to increase the computational power was far from being new in formal verification. 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. As a matter of fact, the interest in the *platform-dependent* formal verification has been revived.

2 The DiVinE Story!

DiVinE [15, 18] is a tool for LTL model checking and reachability analysis of discrete distributed systems. The tool is able to efficiently exploit the aggregate computing power of multiple network-interconnected multi-cored workstations in order to deal with extremely large verification tasks. As such it allows to analyse systems of which size is far beyond the size of systems that can be handled with regular sequential tools. DiVinE tool follows the explicit-state automata-based approach to LTL model checking. Due to Vardi and Wolper [81], the LTL model checking problem reduces to the problem of emptiness of Büchi automata, hence to the problem of the accepting cycle detection in the underlying directed graph of a Büchi automaton.

2.1 Parallel Algorithms in LTL 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 distributed the work among participating processors in order to take advantage of aggregate memory and parallel processing at the same time.

Based on a parallel algorithm for state space generation [30] a static partitioning scheme relying on a hash function was introduced [34]. 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. There were approaches requiring the user of the tool to specify the concrete parts of the state descriptor to be used for partitioning [34, 73], other approaches employed automated or semi-automated techniques to do it [35]. DiVinE implicitly uses a hash-based partitioning over the full state descriptor. Parts of the descriptor used for partitioning might be statically redefined prior compilation. Techniques for load balancing the set of visited states, also known as re-partitioning techniques, have been suggested [2, 74, 70] as well as state space generation schemes employing probability aspects [67, 66]. Nevertheless, none of them have been implemented in DiVinE.

When DiVinE model checker development started, several previous tools had existed. The first known public implementation of a distributed memory tool for verification of communication protocols was the parallel implementation of the Murφ tool [39, 79]. Murφ parallel work-flow relied on the standard MPI-like approach to messaging, nevertheless, active messages were later introduced into Murφ to improve its efficiency [43]. The successful story of the Murφ was followed by other verification tools: SPIN [73, 74], CADP [46], UPPALL [22], etc. Distributed-memory state space generation as a technique of automated formal verification also appeared in the context of Petri Nets [34, 55] and Markov chains [54, 53].

Prior the DiVinE model checker, the existing distributed-memory parallel tools were focused on state-space generation and reachability analyses only. The reason was simple: the lack of parallel algorithms for accepting cycle detection in distributed-memory setting. Nested Depth First Search algorithm (Nested DFS) and other algorithms relying on dept-first search stack cannot be used in distributed-memory setting as the distributed and parallel maintenance of the depth-first search stack is inefficient [4]. Therefore, new parallel and distributed-memory algorithms for accepting cycle detection had to be introduced with the development of DiVinE tool. The first implementation of DiVinE employed the so called dependency structure [14] to record the reachability relation among accepting states of a distributed graph and applied the topological sort algorithm [65] to detect the presence of a self-reachable accepting state. Other parallel algorithms appeared with the time building upon various ideas: detection of negative cycles (NEGC Algorithm) [28, 26], explicit-state implementation of symbolic SCC hull detection (OWCTY Algorithm) [31], value propagation (MAP Algorithm) [27], or algorithm based on back-level edges as produced by a breadth-first search procedure (BLEDGE Algorithm) [8, 9]. These new algorithms differed in theoretic complexity as well as practical efficiency. After large experimental evaluation, some of the algorithms were discontinued in DiVinE. The latest version of DiVinE employs a combination of MAP and OWCTY algorithms by defualt [12]. Table in Figure 1 gives complexities and on-the-fly abilities of newly introduced parallel LTL model checking algorithms.

Distributed-memory processing cannot fight the state space explosion problem alone and must be combined with other techniques. One of the most successful technique to fight the state space explosion in explicit-state model checking is Partial Order Reduction [76]. DiVinE is able to perform this reduction, however, new topological sort proviso had to be developed in order to maintain efficiency of parallel and distributed-memory processing [13]. Another important algorithmic improvement relates to classification of LTL formulas [32]. For some classes of LTL formulas (weak LTL) the parallel algorithms may by significantly improved [7]. With this observation the OWCTY algorithm can be improved so that its complexity even meets the complexity of the optimal sequential Nested DFS algorithm, see Table in Figure 1. 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 [12].

While DiVinE focuses on “complete” verification, parallel distributed-memory “incomplete” verification due to lossy state compaction has been introduced by PReach tool [24].

	Complexity	Optimality	On-The-Fly
Nested DFS	O(N+M)	Yes	Yes
OWCTY Algorithm			
general LTL properties	O(N.(N+M))	No	No
weak LTL properties	O(N+M)	Yes	No
MAP Algorithm	O(N.N.(N+M))	No	Yes
MAP-OWCTY Algorithm			
general LTL properties	O(N.(N+M))	No	Yes
weak LTL properties	O(N+M)	Yes	Yes
BLEDGE Algorithm	O(N.N.(N+M))	No	Yes
NEGC Algorithm	O(N.N.(N+M))	No	Yes

Figure 1: Overview of complexities and on-the-fly processing ability of Nested DFS and parallel algorithms for accepting cycle detection.

2.2 Algorithm Engineering for Parallel LTL Model Checking

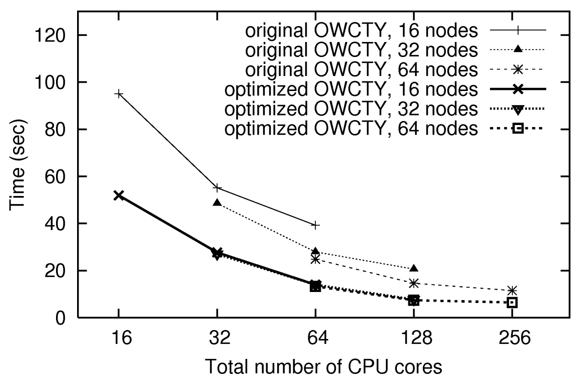
There is no doubt that without an appropriate parallel algorithm the LTL model checking procedure cannot be successfully adapted to contemporary parallel computing platforms. Nevertheless, the algorithm is not the only ingredient required. Even the best algorithms in theory may not outperform good-but-not-optimal algorithms that are equipped with platform-aware heuristics. This observation is even more applicable to parallel processing where the scalability and absolute runtime reduction are typically more valued achievements than theoretical optimality. To that end there is another ingredient behind the development of parallel and distributed-memory tool DiVinE – *Algorithm Engineering*.

> *Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice.*
>
> [Aho et al. [1997], Emerging Opportunities for Theoretical Computer Science]

In other words, characteristics of individual computing platforms must be taken into account in order to obtain efficient implementations on these platforms. In order to take the advantage of the processing power various platforms provide, algorithm and data structure implementations need to be platform-dependent and platform-aware.

2.2.1 Parallelism in Distributed-Memory

Distributed-memory parallel platform was the first platform that the DiVinE tool was adapted to. The intention was to aggregate computational power and distributed system memory of multiple network interconnected workstations (clusters) in order to facilitate the verification of large model checking instances [14, 4]. The general idea of employing the distributed-memory platform for execution of a parallel graph algorithm was, and still is, as follows. The set of vertices of the graph to be processed is partitioned among participating computation nodes using a static partitioning function. When a computation node processes a vertex it enumerates all its immediate successors and checks them for their



Cores	Runtime (sec)	Efficiency
1	631.7	100%
64	13.3	74%
128	7.4	67%
256	5.0	49%

Figure 2: DiVinE performance – optimized ond original implementaion of OWCTY algorithm.

ownership. If a newly generated vertex is local according to the partitioning function, it is pushed to the local queue where it waits for further processing. In the other case a network message is created containing the vertex and sent to the queue of the owning computation node. With this work-flow there is a message generated with every edge connecting vertices from different partitions of the graph. This is where the theory is done, however, when it comes to the implementation there are still numerous design choices to be made. Some of them are detailed for individual computing platforms in the following subsections.

Message aggregation and buffering are the standard techniques in parallel computing to alleviate the burden of network communication overhead. Therefore, DiVinE tool maintains buffers of messages to be sent to individual network-attached computing nodes. In the first implementation of DiVinE, a buffer was flushed (messages were sent to network) upon one of the following situations: 1) buffer was explicitly flushed by the executed graph algorithm, 2) maximal number of messages to fit the buffer has been reached, 3) the local computing node was (otherwise) idle, and 4) messages in the buffer were too old. Deep experimental evaluation, however, showed that the fourth condition is completely ineffective in terms of network flow, while its checking is quite expensive. After dropping the fourth rule for flushing of buffers DiVinE significantly gained in performance.

There were other distributed-memory performance bugs in earlier versions of DiVinE. For exmaple, uncontrolled polling of incoming network messages, massive flushing of all buffers at the same time, or insufficient separation of initialization and computation phases. For more details see [82]. Cumulative effect of elimination of these bugs from DiVinE is shown in Figure 2.

2.2.2 Parallelism in Shared-Memory

Most techniques and results known from the distributed-memory setting are straightforwardly applicable to shared-memory architectures. DiVinE architecture follows this observation, which means that if DiVinE is executed on a multi-core machine with shared-memory, it mimics distributed-memory behavior. In particular, the graph to be processed is partitioned among individual parallel shared-memory threads in the same way as it would be in the distributed-memory setting. Each individual thread maintains its own hash table and its own pool of vertices to be processed. Vertices belonging to different threads are pushed to their local pools by means of lock-free shared-memory queues [10]. Relative advantages and disadvantages of shared versus private hash tables within the context of thread-private pools of vertices to be processed have been discussed in [21]. These approaches were evaluated, both theoretically and practically, in a prototype implementation [11].

Nevertheless, the scalability of parallel distributed-memory solutions to shared-memory is often limited. Therefore, shared-memory specific techniques are needed to improve the efficiency and scalability of existing parallel distributed-memory solutions on shared-memory architectures. Examples of successful shared-memory specific techniques include, e.g., shared communication data structures [60, 10], specific termination detection techniques [10], dual-core algorithms [58], or quite a unique partitioning scheme [57]. As for DiVinE, it seems that the design choice of having thread-private pools of vertices to be processed was not the best one [71]. However, an experimental confirmation still waits to be done.

2.2.3 Employing External Memory

Efficient algorithmic usage of computing devices with memory hierarchies is an established research topic [75]. Numerous algorithms were devised to efficiently utilize external-memory block devices, such as disks. The efficiency of such an algorithm is typically measured in the number of I/O (input/output) operations. To that end, the I/O efficient complexity has been defined [1] and the standard breadth-first graph traversal algorithms adapted to the I/O setting. The crucial technique used to do so is the so called delayed duplicate detection [33] that has been further improved in [83, 3, 49] and specialized for undirected graphs [68, 69]. Regarding formal verification, the graph traversal algorithms are used for state space generation and verification of safety properties, see e.g. disk extension of the verification tool Murφ [80, 78].

As for problems beyond the state space generation, breadth-first search graph traversal algorithms are unsuitable. Therefore, the first approach to LTL model checking with external memory device employed a generic reduction of the LTL model checking problem to the reachability problem [23]. Unfortunately, such a reduction results in a quadratic grow in the memory demands, which effectively eliminates its application to large scale industry cases. Therefore, "incomplete" verification approaches dominated the research field for some time. We have seen random walks strategies implemented [64], iterative deepening and A^* algorithms [61, 62], or breadth-first search based approaches with limited amount of stored information [72] to be used.

The I/O branch of DiVinE was started with the invention of a new I/O algorithmic technique that efficiently avoids the quadratic space overhead [19]. The new approach was further improved by introduction of the so called merge omissions [20] that allowed for more efficient delayed duplicate detection in the later stages of the computation. Various formulas for control of what should be omitted were introduced [45], however, they were not implemented within the I/O branch of DiVinE. A completely different technique for trading time for space employing perfect hashing has been implemented in the I/O branch of DiVinE. This technique is referred to as the semi-external approach to LTL model checking problem [40].

2.2.4 Many-Core Parallelism

After NVIDIA's CUDA technology [38] was introduced, a lot of computational demanding tasks have been accelerated by GPU-aware algorithms. Examples of GPU accelerated procedures include, but are not limited to, sorting [48], reduce operations [52], or numerous biological and physical simulations, such as protein folding [63]. As for the graph theory, successful adaptation of general graph traversal algorithms have been reported too [50, 51] demonstrating the tremendous computational power of the CUDA device. On the other hand, graphs to be explored efficiently with a CUDA accelerated algorithm must be encoded explicitly in a compact way.

The CUDA technology as a computing platform attracted also researches in the field of automated formal verification. The key challenge for which no satisfactory solution is known yet is how to accelerate the generation of explicitly encoded state space graph from the implicit definition. Preliminary attempts to do so relate to explicit model checking. They suggest to employ massively parallel check for enabled transitions emanating from the vertices on the frontier of the search and their massive parallel execution [41, 42].

Once the state space is generated and explicitly represented in an appropriate sparse matrix like structure, many verification tasks can be accelerated using CUDA technology. This has been successfully demonstrated, e.g., on verification of probabilistic systems [25] or LTL model checking [17]. Latest developments in DiVinE CUDA tool [16] allow for efficient utilization of multiple CUDA devices [5] and acceleration of detection of strongly connected components [6].

3 Summary

Platform dependent verification is an alternative approach how to make automated formal verification attractive for industry. Despite significant progress in the development of various specific techniques and tools on the algorithmic level, mainly for parallel architectures, there is still a gap between pseudo-code and implementation. Implementations must be tuned for specific platforms, e.g. memory access patterns seem to play crucial role. In platform depended verification we should learn to appreciate engineering solutions.

References

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

[2] S. Allmaier, S. Dalibor & D. Kreische (1997): *Parallel Graph Generation Algorithms for Shared and Distributed Memory Machines*. In G. Bilardi, A. G. Ferreira, R. Lüling & J. D. P. Rolim, editors: *Proceeding of the Parallel Computing Conference PARCO'97 (Bonn, Germany)*, *LNCS* 1253, Springer, pp. 207–218. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.2006&rep=rep1&type=pdf`.

[3] Tonglaga Bao & Michael Jones (2005): *Time-Efficient Model Checking with Magnetic Disk*. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005)*, *LNCS* 3440, Springer, pp. 526–540, doi:10.1007/978-3-540-31980-1_34.

[4] J. Barnat (2004): *Distributed Memory LTL Model Checking*. Ph.D. thesis, Masaryk University Brno, Faculty of Informatics. Available at `http://anna.fi.muni.cz/papers/src/public/74925f5d8351a5a41120c11f26f3a21b.pdf`.

[5] J. Barnat, P. Bauch, L. Brim & M. Češka (2010): *Employing Multiple CUDA Devices to Accelerate LTL Model Checking*. In: *16th International Conference on Parallel and Distributed Systems (ICPADS 2010)*, IEEE Computer Society, pp. 259–266, doi:10.1109/ICPADS.2010.82.

[6] J. Barnat, P. Bauch, L. Brim & M. Češka (2011): *Computing Strongly Connected Components in Parallel on CUDA*. In: *International Parallel & Distributed Processing Symposium (IPDPS'11)*, IEEE Computer Society, pp. 541–552, doi:10.1109/IPDPS.2011.59.

[7] J. Barnat, L. Brim & I. Černá (2002): *Property driven distribution of Nested DFS*. In: *Proc. Workshop on Verification and Computational Logic*, *DSSE Technical Report* DSSE-TR-2002-5, University of Southampton, UK, pp. 1–10. Available at `http://anna.fi.muni.cz/papers/src/public/5f773c95a13c7b85f303123b36985210.pdf`.

[8] J. Barnat, L. Brim & J. Chaloupka (2003): *Parallel Breadth-First Search LTL Model-Checking*. In: *18th IEEE International Conference on Automated Software Engineering (ASE'03)*, IEEE Computer Society, pp. 106–115, doi:10.1109/ASE.2003.1240299.

[9] J. Barnat, L. Brim & J. Chaloupka (2005): *From Distributed Memory Cycle Detection to Parallel LTL Model Checking*. *Electronic Notes in Theoretical Computer Science* 133(1), pp. 21–39, doi:10.1016/j.entcs.2004.08.056.

[10] J. Barnat, L. Brim & P. Ročkai (2007): *Scalable Multi-core LTL Model-Checking*. In: *Model Checking Software*, *LNCS* 4595, Springer, pp. 187–203, doi:10.1007/978-3-540-73370-6_13.

[11] J. Barnat, L. Brim & P. Ročkai (2008): *DiVinE Multi-Core – A Parallel LTL Model-Checker*. In: *Automated Technology for Verification and Analysis (ATVA 2008)*, *LNCS* 5311, Springer, pp. 234–239, doi:10.1007/978-3-540-88387-6_20.

[12] J. Barnat, L. Brim & P. Ročkai (2009): *A Time-Optimal On-the-Fly Parallel Algorithm for Model Checking of Weak LTL Properties*. In: *Formal Methods and Software Engineering (ICFEM 2009)*, *LNCS* 5885, Springer, pp. 407–425, doi:10.1007/978-3-642-10373-5_21.

[13] J. Barnat, L. Brim & P. Ročkai (2010): *Parallel Partial Order Reduction with Topological Sort Proviso*. In: *Software Engineering and Formal Methods (SEFM 2010)*, IEEE Computer Society Press, pp. 222–231, doi:10.1109/SEFM.2010.35.

[14] J. Barnat, L. Brim & J. Stříbrná (2001): *Distributed LTL Model-Checking in SPIN*. In: *Proc. SPIN Workshop on Model Checking of Software*, *LNCS* 2057, Springer, pp. 200–216, doi:10.1007/3-540-45139-0_13.

[15] J. Barnat, L. Brim, I. Černá, P. Moravec, P. Ročkai & P. Šimeček (2006): *DiVinE – A Tool for Distributed Verification (Tool Paper)*. In: *Computer Aided Verification*, *LNCS* 4144/2006, Springer Berlin / Heidelberg, pp. 278–281, doi:10.1007/11817963_26.

[16] J. Barnat, L. Brim & M. Češka (2009): *DiVinE-CUDA: A Tool for GPU Accelerated LTL Model Checking*. *Electronic Proceedings in Theoretical Computer Science (PDMC 2009)* 14, pp. 107–111, doi:10.4204/EPTCS.14.8.

[17] J. Barnat, L. Brim, M. Češka & T. Lamr (2009): *CUDA accelerated LTL Model Checking*. In: *15th International Conference on Parallel and Distributed Systems (ICPADS 2009)*, IEEE Computer Society, pp. 34–41, doi:10.1109/ICPADS.2009.50.

[18] J. Barnat, L. Brim, M. Češka & P. Ročkai (2010): *DiVinE: Parallel Distributed Model Checker (Tool paper)*. In: *Parallel and Distributed Methods in Verification and High Performance Computational Systems Biology (HiBi/PDMC 2010)*, IEEE, pp. 4–7, doi:10.1109/PDMC-HiBi.2010.9.

[19] J. Barnat, L. Brim & P. Šimeček (2007): *I/O Efficient Accepting Cycle Detection*. In: *Computer Aided Verification*, *LNCS* 4590, Springer, pp. 281–293, doi:10.1007/978-3-540-73368-3_32.

[20] J. Barnat, L. Brim, P. Šimeček & M. Weber (2008): *Revisiting Resistance Speeds Up I/O-Efficient LTL Model Checking*. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS).*, *LNCS* 4963, Springer, pp. 48–62, doi:10.1007/978-3-540-78800-3_5.

[21] J. Barnat & P. Ročkai (2008): *Shared Hash Tables in Parallel Model Checking*. *ENTCS* 198(1), pp. 79–91, doi:10.1016/j.entcs.2007.10.021.

[22] G. Behrmann, T. S. Hune & F. W. Vaandrager (2000): *Distributed Timed Model Checking — How the Search Order Matters*. In: *Proc. 12th Conference on Computer-Aided Verification CAV00*, *LNCS* 1855, Springer, pp. 216–231, doi:10.1007/10722167_19.

[23] Armin Biere, Cyrille Artho & Viktor Schuppan (2002): *Liveness Checking as Safety Checking*. *Electronic Notes in Theoretical Computer Science* 66(2), pp. 160 – 177, doi:10.1016/S1571-0661(04)80410-9.

[24] Brad Bingham, Jesse Bingham, Flavio M. de Paula, John Erickson, Gaurav Singh & Mark Reitblatt (2010): *Industrial Strength Distributed Explicit State Model Checking*. In: *Parallel and Distributed Methods in Verification, 2010 Ninth International Workshop on, and High Performance Computational Systems Biology, Second International Workshop on*, IEEE Computer Society, pp. 28–36, doi:10.1109/PDMC-HiBi.2010.13.

[25] D. Bosnacki, S. Edelkamp & D. Sulewski (2009): *Efficient Probabilistic Model Checking on General Purpose Graphics Processors*. In: *Model Checking Software (SPIN 2009)*, *LNCS* 5578, Springer, pp. 32–49, doi:10.1007/978-3-642-02652-2_7.

[26] L. Brim, I. Černá & L. Hejtmánek (2003): *Parallel Algorithms for Detection of Negative Cycles*. Technical Report FIMU-RS-2003-04, Faculty of Informatics, Masaryk University Brno. Available at `http://www.fi.muni.cz/informatics/reports/files/2003/FIMU-RS-2003-04.pdf`.

[27] L. Brim, I. Černá, P. Moravec & J. Šimša (2004): *Accepting Predecessors are Better than Back Edges in Distributed LTL Model-Checking*. In: *Formal Methods in Computer Aided Design (FMCAD)*, *LNCS* 4144, Springer, pp. 352–366, doi:10.1007/978-3-540-30494-4_25.

[28] L. Brim, I. Černá, P. Krčál & R. Pelánek (2001): *Distributed LTL Model Checking Based on Negative Cycle Detection*. In: *Proc. of Foundations of Software Technology and Theoretical Computer Science (FST TCS 2001)*, *LNCS* 2245, Springer, pp. 96–107, doi:10.1007/3-540-45294-X_9.

[29] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill & L. J. Hwang (1992): *Symbolic Model Checking: 10^{20} States and Beyond*. *Information and Computation* 98(2), pp. 142–170. Available at `http://citeseer.nj.nec.com/burch92symbolic.html`.

[30] S. Caselli, G. Conte & P. Marenzoni (1995): *Parallel state space exploration for GSPN models*. In G. de Michelis & M. Diaz, editors: *Applications and Theory of Petri Nets 1995*, *LNCS* 935, Springer Verlag, pp. 181–200, doi:10.1007/3-540-60029-9_40.

[31] I. Černá & R. Pelánek (2003): *Distributed Explicit Fair Cycle Detection*. In: *Model Checking Software, 10th International SPIN Workshop*, *LNCS* 2648, Springer, pp. 49–73, doi:10.1007/3-540-44829-2_4.

[32] I. Černá & R. Pelánek (2003): *Relating Hierarchy of Temporal Properties to Model Checking*. In: *Mathematical Foundations of Computer Science (MFCS)*, *LNCS* 2747, Springer, pp. 318–327, doi:10.1007/978-3-540-45138-9_26.

[33] Yi-Jen Chiang, Michael T. Goodrich, Edward F. Grove, Roberto Tamassia, Darren Erik Vengroff & Jeffrey Scott Vitter (1995): *External-memory graph algorithms*. In: *soda*, Society for Industrial and Applied Mathematics, pp. 139–149, doi:10.1145/313651.313681.

[34] G. Ciardo, J. Gluckman & D.M. Nicol (1998): *Distributed State Space Generation of Discrete-State Stochastic Models*. *INFORMS Journal on Computing* 10(1), pp. 82–93, doi:10.1287/ijoc.10.1.82.

[35] Gianfranco Ciardo (1997): *Automated parallelization of discrete state-space generation*. *J. Parallel Distrib. Comput.* 47, pp. 153–167, doi:10.1006/jpdc.1997.1409.

[36] E. M. Clarke, R. Enders, T. Filkorn & S. Jha (1996): *Exploiting symmetry in temporal logic model checking*. *Form. Methods Syst. Des.* 9(1-2), pp. 77–104, doi:10.1007/BF00625969.

[37] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu & H. Veith (2001): *Progress on the State Explosion Problem in Model Checking*. In R. Wilhelm, editor: *Informatics - 10 Years Back. 10 Years Ahead*, *LNCS* 2000, Springer, pp. 176–194.

[38] (2009): *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide Version 2.0,*. `http://www.nvidia.com/object/cuda_develop.html`, June 2009.

[39] David L. Dill (1996): *The Murφ verification system*. In: *Conference on Computer-Aided Verification (CAV'96)*, LNCS, Springer-Verlag, pp. 390–393. Available at `http://dl.acm.org/citation.cfm?id=647765.735832`.

[40] Stefan Edelkamp, Peter Sanders & Pavel Šimeček (2008): *Semi-external LTL Model Checking*. In: *Computer Aided Verification (CAV 2008)*, Springer, Berlin, Heidelberg, pp. 530–542, doi:10.1007/978-3-540-70545-1_50.

[41] Stefan Edelkamp & Damian Sulewski (2008): *Model Checking via Delayed Duplicate Detection on the GPU*. Technical Report Technical Report 821, TU Dortmund. Available at `http://www.tzi.de/~edelkamp/GPU_Technical.pdf`. Presented on the 22nd Workshop on Planning, Scheduling, and Design PUK 2008.

[42] Stefan Edelkamp & Damian Sulewski (2009): *Parallel State Space Search on the GPU*. Available at `http://www.cs.ualberta.ca/~nathanst/sara/papers/socs09_submission_24.pdf`. International Symposium on Combinatorial Search (SoCS 2009).

[43] T. von Eicken, D. E. Culler, S. C. Goldstein & K. E. Schauser (1992): *Active messages: a mechanism for integrated communication and computation*. In: *19th Annual International Symposium on Computer Architecture*, pp. 256–266, doi:10.1145/285930.286002.

[44] E. Allen Emerson & A. Prasad Sistla (1996): *Symmetry and model checking*. *Form. Methods Syst. Des.* 9(1-2), pp. 105–131, doi:10.1007/BF00625970.

[45] Sami Evangelista (2008): *Dynamic Delayed Duplicate Detection for External Memory Model Checking*. In: *SPIN '08: Proc. of the 15th international workshop on Model Checking Software*, Springer, Berlin, Heidelberg, pp. 77–94, doi:10.1007/978-3-540-85114-1_8.

[46] H. Garavel, R. Mateescu & I.M Smarandache (2001): *Parallel State Space Construction for Model-Checking*. In Matthew B. Dwyer, editor: *Model Checking of Software (SPIN'2001)*, *LNCS* 2057, Springer-Verlag, pp. 216–234, doi:10.1007/3-540-45139-0_14.

[47] Jaco Geldenhuys & P. J. A. de Villiers (1999): *Runtime Efficient State Compaction in SPIN*. In: *SPIN*, pp. 12–21, doi:10.1007/3-540-48234-2_2.

[48] Naga K. Govindaraju, Jim Gray, Ritesh Kumar & Dinesh Manocha (2006): *GPUTeraSort: high performance graphics co-processor sorting for large database management*. In: *International Conference on Management of Data (SIGMOD 06)*, ACM, pp. 325–336, doi:10.1145/1142473.1142511.

[49] Moritz Hammer & Michael Weber (2006): *"To Store or Not To Store" Reloaded: Reclaiming Memory on Demand*. In: *Formal Methods: Applications and Technology*, *LNCS* 4346, Springer, pp. 51–66, doi:10.1007/978-3-540-70952-7_4.

[50] P. Harish & P. J. Narayanan (2007): *Accelerating Large Graph Algorithms on the GPU Using CUDA*. In: *HiPC*, *LNCS* 4873, Springer, pp. 197–208, doi:10.1007/978-3-540-77220-0_21.

[51] P. Harish, V. Vineet & P. J. Narayanan (2009): *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. Available at `http://cvit.iiit.ac.in/papers/pawan09GraphAlgorithms.pdf`.

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

[53] B. R. Haverkort, A. Bell & H. C. Bohnenkamp (1999): *On the efficient sequential and distributed generation of very large Markov chains from stochastic Petri nets*. In: *Petri Net and Performance Models (PNPM'99)*, IEEE Computer Society Press, pp. 12–21, doi:10.1109/PNPM.1999.796528.

[54] Boudewijn R. Haverkort, Henrik Bohnenkamp & Alexander Bell (1998): *Efficiency Improvements in the Evaluation of Large Stochastic Petri Nets*. In: *Forschungsbericht: 5. Workshop Algorithmen und Werkzeuge für Petrinetze*, Universität Dortmund, Fachbereich Informatik, pp. 55–61. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.41.3435&rep=rep1&type=pdf`.

[55] Keijo Heljanko, Victor Khomenko & Maciej Koutny (2002): *Parallelisation of the Petri Net Unfolding Algorithm*. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002)*, *LNCS* 2280, Springer, pp. 371–385, doi:10.1007/3-540-46002-0_26.

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

[57] Gerard J. Holzmann (2008): *A Stack-Slicing Algorithm for Multi-Core Model Checking*. *ENTCS* 198(1), pp. 3–16, doi:10.1016/j.entcs.2007.10.017.

[58] Gerard J. Holzmann & Dragan Bosnacki (2007): *The Design of a Multicore Extension of the SPIN Model Checker*. *IEEE Trans. Software Eng.* 33(10), pp. 659–674, doi:10.1109/TSE.2007.70724.

[59] *iFEST: Integration Framework for Embedded Systems Tools*. Available at `http://www.artemis-ifest.eu`.

[60] Cornelia P. Inggs & Howard Barringer (2005): *CTL* Model Checking on a Shared-Memory Architecture*. *Electronic Notes in Theoretical Computer Science* 128(3), pp. 107–123, doi:10.1016/j.entcs.2004.10.022.

[61] Shahid Jabbar & Stefan Edelkamp (2005): *I/O Efficient Directed Model Checking*. In: *Verification, Model Checking, and Abstract Interpretation (VMCAI 2005)*, *LNCS* 3385, Springer, pp. 313–329, doi:10.1007/978-3-540-30579-8_21.

[62] Shahid Jabbar & Stefan Edelkamp (2006): *Parallel External Directed Model Checking with Linear I/O*. In: *Verification, Model Checking, and Abstract Interpretation (VMCAI 2006)*, *LNCS* 3855, Springer, pp. 237–251, doi:10.1007/11609773_16.

[63] G. Jayachandran, V. Vishal & V. S. Pande (2006): *Using massively parallel simulations and Markovian models to study protein folding: examining the villin head-piece*. *Journal of Chemical Physics* 124(6), p. 903–914, doi:10.1063/1.2186317.

[64] Michael Jones & Eric Mercer (2004): *Explicit State Model Checking with Hopper*. In: *Model Checking Software (SPIN 2004)*, *Lecture Notes in Computer Science* 2989, Springer, pp. 146–150, doi:10.1007/978-3-540-24732-6_10.

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

[66] W. Knottenbelt, P.G Harrison, M. Mestern & P.S. Kritzinger (2000): *A Probabilistic Dynamic Technique for the Distributed Generation of Very Large State Spaces*. *Performance Evaluation* 35(1–4), pp. 127–148, doi:10.1016/S0166-5316(99)00061-9.

[67] W. Knottenbelt, M. Mestern, P.G Harrison, & P.S. Kritzinger (1998): *Probability, Parallelism and the State Space Exploration Problem*. In R. Puigjaner, editor: *Tools'98*, *LNCS* 1469, Springer Verlag, pp. 165–179, doi:10.1007/3-540-68061-6_14.

[68] R. Korf (2004): *Best-First Frontier Search with Delayed Duplicate Detection*. In: *AAAI'04*, AAAI Press / The MIT Press, pp. 650–657. Available at `http://dl.acm.org/citation.cfm?id=1597148.1597253`.

[69] R. Korf & P. Schultze (2005): *Large-Scale Parallel Breadth-First Search*. In: *Proceedings of the 20th national conference on Artificial intelligence - Volume 3*, AAAI Press / The MIT Press, pp. 1380–1385. Available at `http://dl.acm.org/citation.cfm?id=1619499.1619555`.

[70] Rahul Kumar & Eric G. Mercer (2005): *Load Balancing Parallel Explicit State Model Checking*. *Electronic Notes in Theoretical Computer Science* 128(3), pp. 19–34, doi:10.1016/j.entcs.2004.10.016.

[71] Alfons Laarman, Jaco van de Pol & Michael Weber (2010): *Boosting Multi-Core Reachability Performance with Shared Hash Tables*. In: *Formal Methods in Computer-Aided Design (FMCAD 2010)*, IEEE, pp. 247–255. Available at `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5770956`.

[72] Peter Lamborn & Eric A. Hansen (2008): *Layered Duplicate Detection in External-Memory Model Checking*. In: *SPIN '08: Proc. of the 15th international workshop on Model Checking Software*, Springer, Berlin, Heidelberg, pp. 160–175, doi:10.1007/978-3-540-85114-1_13.

[73] Flavio Lerda & Riccardo Sisto (1999): *Distributed-memory Model Checking with SPIN*. In: *Proc. of the 5th International SPIN Workshop*, *LNCS* 1680, Springer-Verlag, pp. 22–39, doi:10.1007/3-540-48234-2_3.

[74] Flavio Lerda & Willem Visser (2001): *Addressing Dynamic Issues of Program Model Checking*. In: *International SPIN Workshop on Model Checking of Software (SPIN'2001)*, *LNCS* 2057, Springer, pp. 80–102, doi:10.1007/3-540-45139-0_6.

[75] Ulrich Meyer, Peter Sanders & Jop Sibeyn, editors (2003): *Algorithms for Memory Hierarchies*. Springer.

[76] Doron Peled (1998): *Ten Years of Partial Order Reduction*. In: *Proceedings of the 10th International Conference on Computer Aided Verification*, Springer-Verlag, pp. 17–28.

[77] R. Pelánek (2009): *Fighting State Space Explosion: Review and Evaluation*. In: *Formal Methods for Industrial Critical Systems (FMICS 2008)*, *LNCS* 5596, Springer, pp. 37–52, doi:10.1007/978-3-642-03240-0_7.

[78] Giuseppe Della Penna, Benedetto Intrigila, Enrico Tronci & Marisa Venturini Zilli (2002): *Exploiting Transition Locality in the Disk Based Murφ Verifier*. In: *Formal Methods in Computer-Aided Design (FMCAD 2002)*, pp. 202–219, doi:10.1007/3-540-36126-X_13.

[79] U. Stern & D. L. Dill (1997): *Parallelizing the Murφ Verifier*. In O. Grumberg, editor: *Proceedings of Computer Aided Verification (CAV '97)*, *LNCS* 1254, Springer-Verlag, pp. 256–267, doi:10.1007/BFb0028727.

[80] U. Stern & D. L. Dill (1998): *Using Magnetic Disk Instead of Main Memory in the Murφ Verifier*. In: *Computer Aided Verification. 10th International Conference*, pp. 172–183, doi:10.1007/BFb0028743.

[81] M.Y. Vardi & P. Wolper (1986): *An automata-theoretic approach to automatic program verification*. In: *Proc. IEEE Symposium on Logic in Computer Science*, Computer Society Press, pp. 322–331.

[82] K. Verstoep, H. Bal, J. Barnat & L. Brim (2009): *Efficient Large-Scale Model Checking*. In: *23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS 2009)*, IEEE, pp. 1–12, doi:10.1109/IPDPS.2009.5161000.

[83] Rong Zhou & Eric A. Hansen (2004): *Structured Duplicate Detection in External-Memory Graph Search*. In: *AAAI*, AAAI Press / The MIT Press, pp. 683–689. Available at `http://www.aaai.org/Papers/AAAI/2004/AAAI04-108.pdf`.