Towards Verified Construction of Correct and Optimised GPU Software

Marieke Huisman
m.huisman@utwente.nl
University of Twente
Enschede, The Netherlands

Anton Wijs
A.J.Wijs@tue.nl
Eindhoven University of Technology
Eindhoven, The Netherlands

Abstract

Techniques are required that support developers to produce GPU software that is both functionally correct and high-performing. We envision an integration of push-button formal verification techniques into a Model Driven Engineering workflow. In this paper, we present our vision on this topic, and how we plan to make steps in that direction in the coming five years.

CCS Concepts: • **Software and its engineering → Software verification and validation**; **Software functional properties**; **System description languages**.

Keywords: GPU software, formal verification, model transformation, code generation

ACM Reference Format:
Marieke Huisman and Anton Wijs. 2020. Towards Verified Construction of Correct and Optimised GPU Software. In *Proceedings of the 22th ACM SIGPLAN International Workshop on Formal Techniques for Java-Like Programs (FTfJP '20), July 23, 2020, Virtual, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3427761.3428344

1 Introduction

The use of *graphics processors* (GPUs) for general purpose computing has greatly impacted the computational capabilities regarding linear algebra (e.g., matrix-vector multiplication [44, 88]), computational biology (e.g., genomics [85] and genetic network reconstruction [16]), statistics [63], physics (e.g., fluid dynamics [9]), image processing [64], formal verification [6–8, 33, 87, 89, 92, 93], and machine learning (deep learning [57]). However, to effectively use GPUs, expert knowledge is required about the hardware characteristics, and even then, software development can be time-consuming and frustrating. Proper techniques to make the development and maintenance of GPU software more insightful and less prone to introduce bugs, while helping the developer to introduce performance optimisations, are lacking [43, 50, 73]. The existence of such tools would make GPU computing a far more attractive option for most software developers. In the current paper, we outline our vision to integrate *formal verification techniques* [5, 12, 26, 38, 70] into a *Model Driven Engineering* (MDE) [17, 75] workflow, to provide suitable GPU software development tools. It is crucial that those techniques do not require expert knowledge on formal verification, to make them usable for the average software developer.

In MDE, one reasons about the system under development in terms of a model written in a Domain-Specific Language (DSL) [41, 65]. *Model transformations* are applied on models, for instance, to add more information, to rewrite the model in a different DSL, or to generate source code.

MDE enables a very structured way of software development, and improves flexibility: the model can be updated, and code can be regenerated and optimised at any time. However, MDE currently provides no guarantees that the resulting software will be correct and efficient, i.e. a) that it does what it is supposed to do, and b) that it does this while realising the full potential of the hardware it is running on.

2 Our Envisioned MDE Workflow

In the coming five years, in the ChEOPS project[1], we plan to use formal verification techniques to establish in every step of an MDE workflow that the produced GPU system will be functionally correct and preserve the semantics of the model. First, a model is constructed that describes the desired functionality using an appropriate DSL. This model is formally verified to determine whether it correctly addresses the intended functionality. Next, a GPU implementation of the system is generated. Verification of such an implementation with a code verification tool, such as VerCors [15] and VeriFast [78], can be very time-consuming and typically requires a formal verification expert. Therefore, the approach we plan to follow is to automatically annotate the code with the semantics of the model, such that those tools can be used with the push of a button. Recently, we have applied this

---

[1] http://cheops.win.tue.nl.

FTfJP '20, July 23, 2020, Virtual, USA

ACM ISBN 978-1-4503-8186-4/20/07…$15.00
https://doi.org/10.1145/3427761.3428344

approach to achieve verified generation of multi-threaded software [86]. Finally, in a number of steps, the produced source code can be transformed to optimise it for specific GPU hardware. In these steps, annotations will be refined along with the code and code verification will be reapplied.

To achieve this, a number of topics need to be addressed.

2.1 Specifying and Verifying GPU Program Models

In the literature, many DSLs have been proposed to specify parallel systems, some of which targeting GPU systems, e.g. [2, 31, 34, 36, 39, 47, 53, 59, 81]. The ability to produce high-performance GPU systems is an obvious criterion, but much less attention has been given to the verifiability of the produced models. We believe that DSLs are needed that allow reasoning at an abstraction level that is not too high, to be able to still address the relevant behaviour, but also not too low, since that would make verification infeasible. Developers also need to be supported in specifying desired functional properties. Various proposals have been made for the specification of functional properties of parallel systems, for instance to use some form of message sequence charts [18, 25, 54, 55, 61]. However, it is yet to be determined whether such approaches are suitable for GPU systems.

To verify models, we plan to use model checking [5, 26, 89, 93], because of its push-button nature. Previously, GPU programs have been model checked w.r.t. specific properties such as data races and pointer safety [66], but no experience has been reported on checking whether models of GPU software satisfy user-defined functional properties.

2.2 Verified Generation of Code by Means of Model-to-Code Transformations

As GPU code will be produced by using model-to-code transformations, it is crucial to establish that those transformations preserve the semantics of the source model.

Existing approaches, such as [1, 13, 24, 29, 35, 71, 76, 79, 80, 82–84], are not general enough for our purpose; they do not directly support the development of GPU software (or other types of parallel software). Moreover, they do not achieve a push-button approach. For model-to-model transformations, we have developed such techniques [32, 90, 91], but model-to-code transformations pose extra challenges related to the target hardware and programming language.

By generating annotated code, existing tools such as OpenJML [27], KeY [3] and Dafny [58] can be applied. Originally aimed at sequential code, these techniques are currently extended to support concurrent and parallel software. On the one hand, there is a line of very active research defining highly advanced program logics for fine-grained concurrency such as CAP [37], TaDa [28], and Iris [52, 56]. On the other hand, there are tools that support the verification of concurrent and parallel programs, such as VerCors [4, 14, 15, 30, 74] (using Viper [67]), and VeriFast [78]. Techniques specifically designed for the analysis of GPU software, such as GPUverify [10, 11] and PUG [60], address crucial correctness issues, such as thread divergence and data races, but they do not support checking user-defined functional requirements.

Regarding the generation of program annotations, we observe that many auxiliary annotations can be generated automatically. In particular, there exists a large body of literature on invariant generation, see e.g., [42, 46, 51, 72, 77]. However, most of those techniques are not applied in program verification, as they do not always match directly with what is needed. For instance, for program verification, one often needs invariants that express properties about arrays or other kinds of data types, but hardly any techniques to automatically generate those have been developed. We believe that this is an important topic to address.

2.3 Correct Transformation of Source Code to Achieve High-Performance Software

When focussing on correctness, performance should not be ignored, especially when developing software for parallel hardware architectures. The main purpose of DSLs is to allow developers to reason at a comfortably high level of abstraction about the intended functionality of the system. Performance-related aspects necessarily require a more technical, low-level view, since optimisations involve closely mapping the software to the hardware. This discrepancy means that an implementation generated from a DSL model is typically not optimised for performance, and additional transformations at the program level are necessary.

In the literature, a large collection of program transformations to improve performance of GPU applications is available, see e.g. [22, 48, 94]. Only in a few cases, formal correctness of these is addressed, for instance in [68].

We will work on producing GPU programs that come with detailed annotations describing which parts of the memory are accessed and changed by which parts of the code (see [49] for preliminary ideas). Besides making a program verifiable, these annotations will also provide valuable information to determine which transformations can be applied to optimise the program. Work on algorithmic classification, which classifies a given program and based on that suggests to apply certain optimisations, is relevant here [19–21, 23, 40, 45, 69]. Moreover, we observe that the information in the annotations can be used to speed up the auto-tuning process, i.e., the process to identify which program configuration achieves the best performance. Optimal tuning can be a timely process, and the use of information to reduce the state space for tuning can have substantial impact [62]. Because of the very precise information available from our program annotations, we believe that we can substantially reduce the overhead of this process. Using detailed program annotations for both verification and optimisation is very new, and we strongly believe that it can be used effectively to produce high-quality GPU software.

References

[1] L. Ab. Rahim and J. Whittle. 2010. Verifying Semantic Conformance of State Machine-to-Java Code Generators. In *Proceedings of MODELS 2010 (Lecture Notes in Computer Science, Vol. 6394)*. Springer, 166–180.

[2] P. Adamczyk. 2003. The anthology of the finite state machine design patterns. In *Proceedings of PLoP 2003*.

[3] W. Ahrendt, B. Beckert, R. Bubel, R. Hähnle, P. Schmitt, and M. Ulbrich. 2016. *Deductive Software Verification - The KeY Book: From Theory to Practice*. Lecture Notes in Computer Science, Vol. 10001. Springer.

[4] A. Amighi, C. Haack, M. Huisman, and C. Hurlin. 2015. Permission-based separation logic for multithreaded Java programs. *Logical Methods in Computer Science* 11, 1 (2015). https://doi.org/10.2168/LMCS-11(1:2)2015 arXiv:1411.0851v1

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

[6] J. Barnat, P. Bauch, L. Brim, and M. Češka. 2012. Designing fast LTL model checking algorithms for many-core GPUs. *J. Parallel and Distrib. Comput.* 72, 9 (2012), 1083–1097. https://doi.org/10.1016/j.jpdc.2011.10.015

[7] E. Bartocci, R. DeFrancisco, and S. A. Smolka. 2014. Towards a GPGPU-parallel SPIN Model Checker. In *Proceedings of SPIN 2014*. ACM Press, 87–96.

[8] S. Berkovich, B. Bonakdarpour, and S. Fischmeister. 2013. GPU-based runtime verification. In *Proceedings of IPDPS 2013*. IEEE Computer Society Press, 1025–1036. https://doi.org/10.1109/IPDPS.2013.105

[9] C. Bertolli, A. Betts, G. Mudalige, M. Giles, and P. Kelly. 2012. Design and performance of the OP2 library for unstructured mesh applications. In *Proceedings of CGWS 2011 (Lecture Notes in Computer Science, Vol. 7155)*. Springer, 191–200. https://doi.org/10.1007/978-3-642-29737-3-22

[10] A. Betts, N. Chong, and A. F. Donaldson. 2012. GPUVerify: a verifier for GPU kernels. In *Proceedings of OOPSLA 2012*. ACM Press, 113–132. https://doi.org/10.1145/2398857.2384625

[11] A. Betts, N. Chong, A. F. Donaldson, J. Ketema, S. Qadeer, P. Thomson, and J. Wickerson. 2015. The Design and Implementation of a Verification Technique for GPU Kernels. *ACM Transactions on Programming Languages and Systems* 37, 3 (2015), 1–49. https://doi.org/10.1145/2743017

[12] P. Bjesse. 2005. What is Formal Verification? *ACM SIGDA Newsletter* 35, 24 (2005).

[13] J.O. Blech, S. Glesner, and J. Leitner. 2005. Formal Verification of Java Code Generation from UML Models. In *3rd International Fujaba Days 2005 - MDD In Practice*. 49–56.

[14] S. Blom, M. Huisman, and M. Mihelčić. 2014. Specification and Verification of GPGPU programs. *Science of Computer Programming* 95 (2014), 376–388. Issue 3.

[15] S. C. C. Blom, S. Darabi, M. Huisman, and W. Oortwijn. 2017. The VerCors tool set: Verification of parallel and concurrent software. In *Proceedings of iFM 2017 (Lecture Notes in Computer Science, Vol. 10510)*. Springer, 102–110. https://doi.org/10.1007/978-3-319-66845-1_7

[16] D. Bošnački, M. R. Odenbrett, A. J. Wijs, W. P. A. Ligtenberg, and P. A. J. Hilbers. 2012. Efficient Reconstruction of Biological Networks via Transitive Reduction on General Purpose Graphics Processors. *BMC Bioinformatics* 13 (2012), 281.

[17] M. Brambilla, J. Cabot, and M. Wimmer. 2012. *Model Driven Software Engineering in Practice*. Morgan & Claypool Publishers.

[18] P. Brosch, U. Egly, S. Gabmeyer, G. Kappel, M. Seidl, H. Tompits, M. Widl, and M. Wimmer. 2012. Towards scenario-based testing of UML diagrams. In *Proceedings of TAP 2012 (Lecture Notes in Computer Science, Vol. 7305)*. Springer, 149–155. https://doi.org/10.1007/978-3-642-30473-6_12

[19] W. Caarls. 2008. *Automated Design of Application-Specific Smart Camera Architectures*. Ph.D. Dissertation. Delft University of Technology.

[20] W. Caarls, P. P. Jonker, and H. Corporaal. 2006. Algorithmic skeletons for stream programming in embedded heterogeneous parallel image processing applications. In *Proceedings of IPDPS 2006*. IEEE Computer Society Press. https://doi.org/10.1109/IPDPS.2006.1639351

[21] D. K. G. Campbell. 1996. *Towards the Classification of Algorithmic Skeletons*. Technical Report. University of York. 1–20 pages. http://www.cs.uiuc.edu/{~}snir/PPP/skeleton/classification.pdf

[22] P. Cantiello and B. Di Martino. 2012. Automatic source code transformation for GPUs based on program comprehension. In *Proceedings of Euro-Par 2011 (Lecture Notes in Computer Science, Vol. 7156)*. Springer, 188–197. https://doi.org/10.1007/978-3-642-29740-3-22

[23] L. Carrington, M. Tikir, C. Olschanowsky, M. Laurenzano, J. Peraza, A. Snavely, and S. Poole. 2011. An Idiom-finding Tool for Increasing Productivity of Accelerators. In *Proceedings of ICS-25*. ACM, 202–212.

[24] S. Chaki, E. M. Clarke, A. Groce, S. Jha, and H. Veith. 2004. Modular verification of software components in C. *IEEE Transactions on Software Engineering* 30, 6 (2004), 388–402. https://doi.org/10.1109/TSE.2004.22

[25] A. Cimatti, S. Mover, and S. Tonetta. 2011. Proving and explaining the unfeasibility of message sequence charts for hybrid systems. In *Proceedings of FMCAD 2011*. IEEE Computer Society Press, 54–62. http://www.scopus.com/inward/record.url?eid=2-s2.0-84857756993{&}partnerID=40{&}md5=25e8cfbd3b30de6b6275ce8d7e66253f

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

[27] David R. Cok. 2014. OpenJML: Software verification for Java 7 using JML, OpenJDK, and Eclipse. In *Proceedings of F-IDE 2014 (Electronic Notes in Theoretical Computer Science, Vol. 149)*. Elsevier, 79–92. https://doi.org/10.4204/EPTCS.149.8 arXiv:1404.6608

[28] P. Da Rocha Pinto, T. Dinsdale-Young, and P. Gardner. 2014. TaDA: A logic for time and data abstraction. In *Proceedings of ECOOP 2014 (Lecture Notes in Computer Science, Vol. 8586)*. Springer, 207–231. https://doi.org/10.1007/978-3-662-44202-9_9

[29] M. Dalvandi, M. Butler, and A. Rezazadeh. 2015. From Event-B models to Dafny code contracts. In *Proceedings of FSEN 2015 (Lecture Notes in Computer Science, Vol. 9392)*. Springer, 308–315. https://doi.org/10.1007/978-3-319-24644-4_21

[30] S. Darabi, S.C.C. Blom, and M. Huisman. 2017. A Verification Technique for Deterministic Parallel Programs. In *NASA Formal Methods (NFM) (LNCS, Vol. 10227)*, C. Barrett, M. Davies, and T. Kahsai (Eds.). 247 – 264.

[31] S.M.J. de Putter, A.J. Wijs, and D. Zhang. 2018. The SLCO Framework for Verified, Model-driven Construction of Component Software. In *Proceedings of FACS 2018 (Lecture Notes in Computer Science, Vol. 11222)*. Springer, 288–296.

[32] S. M. J. de Putter and A. J. Wijs. 2018. A formal verification technique for behavioural model-to-model transformations. *Formal Aspects of Computing* 30, 1 (2018), 3–43. https://doi.org/10.1007/s00165-017-0437-z

[33] R. DeFrancisco, S. Cho, M. Ferdman, and S. A. Smolka. 2019. Swarm Model Checking on the GPU. In *Proceedings of SPIN 2019 (Lecture Notes in Computer Science, Vol. 11636)*. Springer, 94–113. https://doi.org/10.1007/978-3-030-30923-7_6

[34] P. Deligiannis, A. F. Donaldson, J. Ketema, A. Lal, and P. Thomson. 2015. Asynchronous Programming, Analysis and Testing with State Machines. In *Proceedings of PLDI 2015 (ACM SIGPLAN Notices, Vol. 50)*. ACM Press, 154–164.

[35] E. Denney, B. Fischer, J. Schumann, and J. Richardson. 2005. Automatic certification of kalman filters for reliable code generation. In *IEEE Aerospace Conference Proceedings*. IEEE Computer Society Press, 1–10. https://doi.org/10.1109/AERO.2005.1559605

[36] A. Desai, V. Gupta, E. K. Jackson, S. Qadeer, S. K. Rajamani, and D. Zufferey. 2013. P: Safe Asynchronous Event-Driven Programming. In *Proceedings of PLDI 2013 (ACM SIGPLAN Notices, Vol. 48)*. ACM Press,

321–332.

[37] T. Dinsdale-Young, M. Dodds, P. Gardner, M. J. Parkinson, and V. Vafeiadis. 2010. Concurrent abstract predicates. In *Proceedings of ECOOP 2010 (Lecture Notes in Computer Science, Vol. 6183)*. Springer, 504–528. https://doi.org/10.1007/978-3-642-14107-2_24

[38] D. A. Duffy. 1991. *Principles of Automated Theorem Proving*. John Wiley & Sons Ltd.

[39] L. J. P. Engelen. 2012. *From Napkin Sketches To Reliable Software*. Ph.D. Dissertation. Eindhoven University of Technology.

[40] J. Enmyren and C. W. Kessler. 2010. SkePU: a multi-backend skeleton programming library for multi-GPU systems. In *Proceedings of HLPP-4*. ACM, 5–14. https://doi.org/10.1145/1863482.1863487

[41] M. Fowler and R. Parsons. 2010. *Domain-Specific Languages*. Addison-Wesley.

[42] J. P. Galeotti, C. A. Furia, E. May, G. Fraser, and A. Zeller. 2015. Inferring loop invariants by mutation, dynamic analysis, and static checking. *IEEE Transactions on Software Engineering* 41, 10 (2015), 1019–1037. https://doi.org/10.1109/TSE.2015.2431688 arXiv:arXiv:1407.5286v2

[43] S. Garfinkel. 2008. History's Worst Software Bugs. http://www.wired.com/software/coolapps/news/2005/11/69355?currentPage=all.

[44] D. Grewe and A. Lokhmotov. 2011. Automatically generating and tuning GPU code for sparse matrix-vector multiplication from a high-level representation. In *Proceedings of GPGPU-4*. 1. https://doi.org/10.1145/1964179.1964196

[45] P. Hijma, C. J. H. Jacobs, R. V. van Nieuwpoort, and H. E. Bal. 2015. Cashmere: Heterogeneous Many-Core Computing. In *Proceedings of IPDPS 2015*. IEEE Computer Society Press, 135–145.

[46] K. Hoder, L. Kovács, and A. Voronkov. 2011. Invariant generation in vampire. In *Proceedings of TACAS 2011 (Lecture Notes in Computer Science, Vol. 6605)*. Springer, 60–64. https://doi.org/10.1007/978-3-642-19835-9_7

[47] S. Hong, S. Salihoglu, J. Widom, and K. Olukotun. 2014. Simplifying Scalable Graph Processing with a Domain-Specific Language. In *Proceedings of CGO 2014*. ACM Press, 208–218. https://doi.org/10.1145/2581122.2544162

[48] D. Huang, M. Wen, C. Xun, D. Chen, X. Cai, Y. Qiao, N. Wu, and C. Zhang. 2014. Automated transformation of GPU-specific OpenCL kernels targeting performance portability on multi-core/many-core CPUs. In *Proceedings of Euro-Par 2014 (Lecture Notes in Computer Science, Vol. 8632)*. Springer, 210–221. https://doi.org/10.1007/978-3-319-09873-9_18

[49] M. Huisman, S. Blom, S. Darabi, and M. Safari. 2018. Program Correctness by Transformation. In *8th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA) (LNCS, Vol. 11244)*. Springer.

[50] ISTAG. 2012. Software Technologies - The Missing Key Enabling Technology: Toward a Strategic Agenda for Software Technologies in Europe.

[51] M. Janota. 2007. Assertion-based loop invariant generation. *RISC-Linz* 03 (2007). http://www.risc.uni-linz.ac.at/publications/download/risc{_}3128/proceedings.pdf{#}page=23

[52] R. Jung, D. Swasey, F. Sieczkowski, K. Svendsen, A. Turon, L. Birkedal, and D. Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of POPL 2015*, Vol. 50. ACM Press, 637–650. https://doi.org/10.1145/2775051.2676980

[53] N. Kapre and S. Bayliss. 2016. Survey of domain-specific languages for FPGA computing. In *Proceedings of FPL 2016*. IEEE Computer Society Press, 1–12. https://doi.org/10.1109/FPL.2016.7577380

[54] A. Knapp, S. Merz, and C. Rauh. 2002. Model Checking Timed UML State Machines and Collaborations. In *Proceedings of FTRTFT 2002 (Lecture Notes in Computer Science, Vol. 2469)*. Springer, 395–416. https://doi.org/10.1007/3-540-45739-9_23

[55] A. Knapp and J. Wuttke. 2006. Model Checking of UML 2.0 Interactions. In *Proceedings of ELS 2006 (Lecture Notes in Computer Science, Vol. 4364)*. Springer, 42–51. https://doi.org/10.1007/978-3-540-69489-2_6

[56] R. Krebbers, R. Jung, A. Bizjak, J. H. Jourdan, D. Dreyer, and L. Birkedal. 2017. The essence of higher-order concurrent separation logic. In *Proceedings of ESOP 2017 (Lecture Notes in Computer Science, Vol. 10201)*. Springer, 696–723. https://doi.org/10.1007/978-3-662-54434-1_26

[57] Q. V. Le, A. Coates, B. Prochnow, and A. Y Ng. 2011. On Optimization Methods for Deep Learning. In *Proceedings of ICML 2011*. Omnipress, 265–272. https://doi.org/10.1.1.220.8705

[58] K. R. M. Leino. 2010. Dafny: An automatic program verifier for functional correctness. In *Proceedings of LPAR-16 (Lecture Notes in Computer Science, Vol. 6355)*. Springer, 348–370. https://doi.org/10.1007/978-3-642-17511-4_20

[59] R. Leißa, K. Boesche, S. Hack, R. Membarth, and P. Slusallek. 2015. Shallow embedding of DSLs via online partial evaluation. In *Proceedings of GPCE 2015*. ACM Press, 11–20. https://doi.org/10.1145/2814204.2814208

[60] G. Li and G. Gopalakrishnan. 2012. Parameterized verification of GPU kernel programs. In *Proceedings of IPDPSW 2012*. IEEE Computer Society Press, 2450–2459. https://doi.org/10.1109/IPDPSW.2012.302

[61] X. Li, J. Hu, L. Bu, J. Zhao, and G. Zheng. 2005. Consistency checking of concurrent models for scenario-based specifications. In *Proceedings of SDL 2005 (Lecture Notes in Computer Science, Vol. 3530)*. Springer, 298–312.

[62] R. Lim, B. Norris, and A. Malony. 2017. Autotuning GPU Kernels via Static and Predictive Analysis. In *Proceedings of ICPP 2017*. IEEE Computer Society Press, 523–532. https://doi.org/10.1109/ICPP.2017.61 arXiv:1701.08547

[63] X. Liu, S. Tan, and H. Wang. 2012. Parallel Statistical Analysis of Analog Circuits by {GPU}-accelerated Graph-based Approach. In *Proceedings of DATE 2012*. IEEE Computer Society Press, 852–857. https://doi.org/10.1109/DATE.2012.6176615

[64] T. G. Mattson, B. A. Sanders, and B. L. Massingill. 2004. *Patterns for Parallel Programming*. Addison-Wesley.

[65] M. Mernik, J. Heering, and A. M. Sloane. 2005. When and How to Develop Domain-Specific Languages. *Comput. Surveys* 37, 4 (2005), 316–344.

[66] F.R. Monteiro, E.H. Alves, I. Silva, H.I. Ismail, L.C. Cordeiro, and E.B. de Lima Filho. 2018. ESBMC-GPU: A Context-Bounded Model Checking Tool to Verify CUDA Programs. *Science of Computer Programming* 152 (2018), 63–69.

[67] P. Müller, M. Schwerhoff, and A. J. Summers. 2016. Viper: A verification infrastructure for permission-based reasoning. In *Proceedings of VMCAI 2016 (Lecture Notes in Computer Science, Vol. 9583)*. Springer, 41–62. https://doi.org/10.1007/978-3-662-49122-5_2

[68] K. Namjoshi and N. Singhania. 2016. Loopy: Programmable and Formally Verified Loop Transformations. In *Proceedings of SAS 2016 (Lecture Notes in Computer Science, Vol. 9837)*. Springer, 383–402.

[69] C. Nugteren. 2014. *Improving the Programmability of GPU Architectures*. Ph.D. Dissertation. Eindhoven University of Technology.

[70] S. Owre, J. M. Rushby, and N. Shankar. 1992. PVS: a Prototype Verification System. In *Proceedings of CADE 1992 (Lecture Notes in Computer Science, Vol. 607)*. Springer, 748–752.

[71] A. Pnueli, O. Shtrichman, and M. Siegel. 1998. The Code Validation Tool CVT: Automatic Verification of a Compilation Process. *Software Tools for Technology Transfer* 2 (1998), 192–201.

[72] E. Rodríguez-Carbonell and D. Kapur. 2007. Automatic generation of polynomial invariants of bounded degree using abstract interpretation. *Science of Computer Programming* 64, 1 SPEC. ISS. (2007), 54–75. https://doi.org/10.1016/j.scico.2006.03.003

[73] D. Rosenberg. 2009. Avoiding the software 'fail whale'. http://news.cnet.com/8301-13846{_}3-10349034-62.html.

[74] M. Safari, W. Oortwijn, S.J.C. Joosten, and M. Huisman. 2020. Formal Verification of Parallel Prefix Sum. In *NASA Formal Methods (NFM) (LNCS)*, R. Lee, S. Jha, and A. Mavridou (Eds.). Springer.

[75] D. C. Schmidt. 2006. Model-Driven Engineering. *IEEE Computer* 39, 2 (2006), 25–31.

[76] J. Schumann, B. Fischer, M. Whalen, and J. Whittle. 2003. Certification support for automatically generated programs. In *Proceedings of HICSS 2003*. IEEE Computer Society Press, 1–10. https://doi.org/10.1109/HICSS.2003.1174914

[77] R. Sharma, I. Dillig, T. Dillig, and A. Aiken. 2011. Simplifying loop invariant generation using splitter predicates. In *Proceedings of CAV 2011 (Lecture Notes in Computer Science, Vol. 6806)*. Springer, 703–719. https://doi.org/10.1007/978-3-642-22110-1_57

[78] J. Smans, B. Jacobs, and F. Piessens. 2012. Verifying Java Programs with VeriFast. *Aliasing in Object-oriented Programming* 2 (2012), 1–18. http://www.csc.kth.se/utbildning/kth/kurser/DD2460/Tools/VeriFast/verifast-java.pdf

[79] M. Staats and M. Heimdahl. 2008. Partial Translation Verification for Untrusted Code-generators. In *Proceedings of ICFEM 2008 (Lecture Notes in Computer Science, Vol. 5256)*. Springer, 226–237.

[80] K. Stenzel, N. Moebius, and W. Reif. 2011. Formal Verification of QVT Transformations for Code Generation. In *Proceedings of MoDELS 2011 (Lecture Notes in Computer Science, Vol. 6981)*. Springer, 533–547.

[81] A. K. Sujeeth, K. J. Brown, H. Lee, T. Rompf, H. Chafi, M. Odersky, and K. Olukotun. 2014. Delite: A Compiler Architecture for Performance-Oriented Embedded Domain-Specific Languages. *ACM Transactions on Embedded Computing Systems* 13, 4s (2014), 1–25. https://doi.org/10.1145/2584665

[82] M. Sulzmann and A. Zechner. 2012. Model Checking DSL-Generated C Source Code. In *Proceedings of SPIN 2012 (Lecture Notes in Computer Science, Vol. 7385)*. Springer, 241–247.

[83] J. Whittle and B. Gajanovic. 2005. Model Transformations Should Be More Than Just Model Generators. In *Satellite Events at the MoDELS 2005 Conference (Lecture Notes in Computer Science, Vol. 3844)*. Springer, 32–38.

[84] J. Whittle, J. Van Baalen, J. Schumann, P. Robinson, T. Pressburger, J. Penix, P. Oh, M. Lowry, and G. Brat. 2001. Amphion/NAV: Deductive Synthesis of State Estimation Software. In *Proceedings of ASE 2001*. IEEE Computer Society Press, 395–399.

[85] S. Wienke, P. Springer, C. Terboven, and D. An Mey. 2012. OpenACC - First experiences with real-world applications. In *Proceedings of EuroPar 2012 (Lecture Notes in Computer Science, Vol. 7484)*. Springer, 859–870. https://doi.org/10.1007/978-3-642-32820-6_85

[86] A.J. Wijs and M. Wiłkowski. 2019. Modular Indirect Push-button Formal Verification of Code Generators. In *Proceedings of SEFM 2019 (Lecture Notes in Computer Science, Vol. 11724)*. Springer, 410–429.

[87] A. J. Wijs. 2015. GPU Accelerated Strong and Branching Bisimilarity Checking. In *Proceedings of TACAS 2015 (Lecture Notes in Computer Science, Vol. 9035)*. Springer, 368–383.

[88] A. J. Wijs and D. Bošnački. 2012. Improving GPU Sparse Matrix-Vector Multiplication for Probabilistic Model Checking. In *Proceedings of SPIN 2012 (Lecture Notes in Computer Science, Vol. 7385)*. Springer, 98–116.

[89] A. J. Wijs and D. Bošnački. 2014. GPUexplore: Many-Core On-The-Fly State Space Exploration Using GPUs. In *Proceedings of TACAS 2014 (Lecture Notes in Computer Science, Vol. 8413)*. Springer, 233–247.

[90] A. J. Wijs and L. Engelen. 2014. *REFINER: Towards formal verification of model transformations*. Vol. 8430 LNCS. https://doi.org/10.1007/978-3-319-06200-6_21

[91] A. J. Wijs and L. J. P. Engelen. 2013. Efficient Property Preservation Checking of Model Refinements. In *Proceedings of TACAS 2013 (Lecture Notes in Computer Science, Vol. 7795)*. Springer, 565–579.

[92] A. J. Wijs, J. P. Katoen, and D. Bošnački. 2016. Efficient GPU algorithms for parallel decomposition of graphs into strongly connected and maximal end components. *Formal Methods in System Design* 48, 3 (2016), 274–300. https://doi.org/10.1007/s10703-016-0246-7

[93] A. J. Wijs, T. Neele, and D. Bošnački. 2016. GPUexplore 2.0: Unleashing GPU explicit-state model checking. In *Proceedings of FM 2016 (Lecture Notes in Computer Science, Vol. 9995)*. Springer, 694–701. https://doi.org/10.1007/978-3-319-48989-6_42

[94] B. Wu, G. Chen, D. Li, X. Shen, and J. Vetter. 2015. Enabling and Exploiting Flexible Task Assignment on GPU through SM-Centric Program Transformations. In *Proceedings of ICS 2015*. ACM Press, 119–130. https://doi.org/10.1145/2751205.2751213