Design for Hardware Memory Model Verification

Yao Hsiao, Yasas Seneviratne*, Tommy Tracy*, Kevin Skadron*, Caroline Trippel
Stanford University, *University of Virginia
{yaohsiao,trippel}@stanford.edu,{yasas,tjt7a,skadron}@virginia.edu

1 Introduction

Memory consistency models (MCMs) [26] specify the values that shared memory loads are allowed to return in a parallel program. A large body of work has produced formally specified MCMs for a variety of ISAs [58, 49, 6, 53, 47, 64, 4, 34] and high-level languages (HLLs) [44, 14, 50, 13, 11, 46]. These efforts have in turn enabled verified compilers [13, 12, 54, 33, 56, 50, 55, 61, 62] which correctly translate between HLL and ISA MCM primitives and automated tools which find [27, 2, 3, 1, 28, 29, 21, 30, 32, 52] and repair [5, 63, 19, 48] MCM bugs in programs.

The Hardware MCM Verification Challenge. MCMs and the analyses they support are *useless* if we cannot guarantee *microarchitectural compliance*. The prevailing approach for hardware MCM verification given a SystemVerilog implementation of a processor, the *design under verification (DUV)*, is as follows. First, teams of industry engineers spend significant time [20] manually encoding instruction-level *update* properties (e.g., "store instructions update memory") and *ordering* properties (e.g., "stores update memory in an order consistent with their program order") as *linear temporal logic (LTL)* assertions [51, 43] involving low-level signals in the DUV. Second, commercial *model checkers* (e.g., Cadence's JasperGold [15]) are deployed to prove/refute microarchitectural compliance. This process requires significant manual decomposition of both the properties and the DUV to scale verification. Ultimately, industrial hardware MCM verification efforts culminate in *bounded* (e.g., "proven up to *n* cycles") correctness proofs. This incompleteness manifests as hardware bugs in real products [7, 8, 60, 42, 33, 22].

2 Bottom-Up Hardware MCM Verification

Our recent work takes a radically different *bottom-up* approach to the hardware MCM verification challenge by synthesizing axiomatic models of MCM implementations, called *μspec models*, directly from SystemVerilog processor descriptions [25]. These μspec models can be input to existing formal MCM verification flows that evaluate ISA MCM compliance [35, 41, 36, 60, 39, 38, 45]. Such approaches achieve efficiency and scalability by operating on hardware descriptions (μspec models) that eschew irrelevant design details (combinational logic). Moreover, our bottom-up verification approach, called RTL2μspec,[1] out-performs top-down alternatives, since abstract μspec models can be incrementally constructed by evaluating the DUV's adherence to simple (quick to check) and generic (easy to instantiate automatically) low-level LTL properties.

Our proof-of-concept RTL2μspec design synthesizes a complete,and proven correct, μspec model from the open-source four-core, three-stage RISC-V multi-V-scale processor [37, 40] (written in SystemVerilog) in 6.84 minutes. Verifying ISA MCM compliance of the synthesized μspec model takes seconds [36]. Prior work (based on the *top-down approach*) timed out after 11 hours of runtime when attempting to verify ISA MCM compliance of the same microarchitecture [39].

Our goal is to conduct formal MCM verification of *advanced* processor designs which include standard commercial features—e.g., pipelines, including super-scalar and diversified, out-of-order execution, speculation, caches, prefetchers, multi-core. We believe this goal requires: (1) scaling RTL2μspec itself to accommodate advanced processor implementations in both size and complexity, and (2) developing novel *design-for-verification (DFV)* approaches based on formally verified runtime monitors to compensate for model-checker limitations. Our talk will discuss both of these directions, which we outline in the remainder of this abstract.

3 RTL2μspec for Advanced Processors

μhb Analysis. MCM verification approaches that operate on μspec models rely on *microarchitectural happens-before (μhb) analysis*, which models hardware-specific program executions as directed *μhb graphs* (Fig. 1). Nodes are *hardware events*, namely program instructions (graph column labels) updating *sets* of hardware state elements (graph row labels); and directed edges denote *happens-before* relations.

μspec Models. To support μhb analysis, a microarchitecture is encoded as a *μspec model*—a collection of first-order logic axioms (i.e., rules) which describe how to construct μhb graph representations of encode hardware-specific program executions on the design. The axioms describe (1) how instructions flow through the microarchitecture (column-wise nodes and edges), and (2) how instructions interact with each other (edges between graph columns). To verify ISA MCM compliance of a μspec model, one can use formal tools (e.g., *satisfiability (SAT)* [18] or *SAT Modulo Theory (SMT)* [10] solvers) to search the space of all *possible* hardware-specific program executions encoded by the model for those which exhibit MCM violations. Each microarchitectural execution *possibility* is denoted by a distinct *acyclic* μhb graph that satisfies all μspec model axioms.

Enhanced RTL2μspec. The core limitation of RTL2μspec is its inability to discover *multiple* distinct execution paths

[1] RTL2μspec is open source at https://github.com/yaohsiaopid/rtl2uspec

(μhb columns) for the same instruction on the DUV. E.g., an instruction with more than one path is a load/store that can experience a cache hit *or* a cache miss.

To conduct path discovery for some *instruction under verification (IUV)*, our insight is that a subset of finite state machines (FSMs) in the DUV's control-path, called *micro-op FMS (μFSMs)*, orchestrate the execution of instructions from the time they are fetched to the time they project their final state updates onto the DUV. An instruction *claims* a set of μFSMs during its execution and progresses through a series of control states in each. Each state represents an execution phase which summarizes a set of state updates that the instruction projects onto the DUV in a given cycle [24, 23]. Thus, our enhanced RTL2μSPEC discovers an IUV's paths by exploring all realizable partial orders on visited μFSM states.

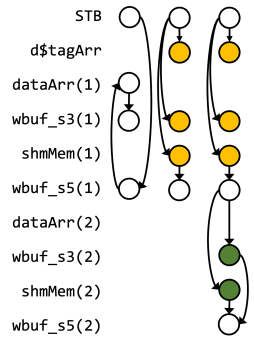

Figure 1. μhb graphs for SW on CVA6 write-through cache.

We used the path discovery procedure outlined above to explore instruction execution behavior on two processors: the RISC-V multi-V-scale (described above) and CVA6 [66]. The latter is a single-issue RISC-V core featuring speculation, limited out-of-order write-back, diversified function units, and a cache. Fig. 1 shows some execution paths discovered for stores within cache hierarchy: cache hit path (left), miss paths with varying address alignment (middle, right). Like original RTL2μSPEC, our path discovery approach relies on a combination of modest design metadata, static netlist analysis, linear temporal logic (LTL) property generation [51, 43], and model checking [9, 16]. Most properties are checked in the order of seconds to minutes.

Challenges. Our talk will discuss several challenges that remain towards scaling RTL2μSPEC to target advanced processors. First, RTL2μSPEC operates on monolithic processor designs; we plan to modularize its implementation to further scale performance. Second, our path discovery procedure encodes paths at the coarse granularity of μFSM control states, rather than at the fine granularity of updated datapath registers; we plan to devise techniques based on RTL-level taint tracking to convert coarse-grained paths into fine-grained ones. Third, RTL2μSPEC cannot yet derive cross-core μSPEC axioms that summarize update and ordering guarantees provided by some DUV's coherence protocol implementation.

RTL2μSPEC will inevitably be limited by undetermined model checker outcomes. Some undetermined outcomes can be proven/refuted on a scaled-down design (e.g., by reducing queue sizes) [17] and generalized to a scaled-up design using induction [31, 65, 57]. Others are more fundamental and arise due to complex dependency chains in the DUV. To address this latter issue, we will explore the use of *verifiable* hardware *monitors* that check for the difficult-to-prove (but required for MCM compliance) LTL properties at runtime.

4 Verified Runtime Monitors

Tracy et al. demonstrate an open-source LTLtoAutomata framework [59] to translate LTL_f (*LTL on finite traces*) formulas into finite state automata and then hardware.

They demonstrated processing 1000s of automata in parallel on an FPGA.

Our talk will propose leveraging this approach to generate explicit NFA hardware monitors from undetermined LTL properties. These hardware monitors will be directly integrated in to the final DUV. To support complete MCM verification with RTL2μSPEC, we will deploy model checkers to verify hardware monitor correctness, a far simpler task which trades off with additional circuitry in the DUV.

We will also explore research challenges associated with integrating the aforementioned hardware monitors into the DUV. For example, efficiently supporting/verifying multiple LTL properties will likely require breaking hardware monitors into multiple modules. This will allow us to reduce routing and I/O complexity between the DUV and the monitor modules. There are several approaches to clustering the LTL properties which we will investigate, including separating properties into clusters with like input signals or into clusters that have signals spatially close to each other in the DUV. Furthermore, we will explore various options for responding to runtime errors detected by monitors (e.g., roll-back or recover mechanisms).

5 Putting it all Together

In integrating RTL2μSPEC with runtime monitors, we will leverage *counterexample guided abstraction refinement (CEGAR)* to minimize added hardware. In particular, RTL2μSPEC will be used to synthesize a partial μSPEC model; high-level MCM verification will be conducted on the incomplete model to evaluate ISA MCM compliance; if verification fails, a counterexample will be used to direct rtl2uspec towards filling in more of the model. Whenever undetermined results contribute to failed μSPEC model verification, runtime monitors will be inserted in the DUV. In this way, CEGAR can guide minimal monitor insertion. This procedure will continue until μSPEC model synthesis terminates with a verified result or a real MCM bug is found.

6 Acknowledgement

We acknowledge support from the Semiconductor Research Corporation and the Intel Scalable Assurance Program.

References

[1] Parosh Aziz Abdulla, Jatin Arora, Mohamed Faouzi Atig, and Shankaranarayanan Krishna. 2019. Verification of programs under the release-acquire semantics. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*.

[2] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bengt Jonsson, and Carl Leonardsson. 2016. Stateless model checking for power. In *Computer Aided Verification*. Swarat Chaudhuri and Azadeh Farzan, (Eds.)

[3] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bengt Jonsson, and Tuan Phong Ngo. 2018. Optimal stateless model checking under the release-acquire semantics. *Proc. ACM Program. Lang.*, OOPSLA.

[4] Jade Alglave, Will Deacon, Richard Grisenthwaite, Antoine Hacquard, and Luc Maranget. 2021. Armed cats: formal concurrency modelling at arm. *ACM Trans. Program. Lang. Syst.*

[5] Jade Alglave, Daniel Kroening, Vincent Nimal, and Daniel Poetzl. 2017. Don't sit on the fence: a static analysis approach to automatic fence insertion.

[6] Jade Alglave, Luc Maranget, and Michael Tautschnig. 2014. Herding cats: modelling, simulation, testing, and data mining for weak memory. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 36, 2, 7:1–7:74.

[7] AMD. 2012. Revision guide for AMD family 10h processors. https://www.amd.com/system/files/TechDocs/41322_10h_Rev_Gd.pdf. (2012).

[8] Arm. 2011. Cortex-A9 MPCore, programmer advice notice, read-after-read hazards, Arm reference 761319. (2011).

[9] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press. ISBN: 026202649X.

[10] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. 2009. Satisfiability modulo theories. In *Handbook of Satisfiability*, 825–885.

[11] Mark Batty, Alastair F. Donaldson, and John Wickerson. 2016. Overhauling SC atomics in C11 and OpenCL. *43rd Symposium on Principles of Programming Languages (POPL)*.

[12] Mark Batty, Kayvan Memarian, Scott Owens, Susmit Sarkar, and Peter Sewell. 2012. Clarifying and compiling C/C++ concurrency: from C++11 to POWER. *39th Symposium on Principles of Programming Languages (POPL)*.

[13] Mark Batty, Scott Owens, Susmit Sarkar, Peter Sewell, and Tjark Weber. 2011. Mathematizing C++ concurrency. *38th Symposium on Principles of Programming Languages (POPL)*.

[14] Hans-J. Boehm and Sarita V. Adve. 2008. Foundations of the C++ concurrency memory model. *29th Conference on Programming Language Design and Implementation (PLDI)*.

[15] Cadence Design Systems, Inc. [n. d.] Cadence JasperGold formal verification platform. Accessed 12th April 2021. (). https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform.html.

[16] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. 2000. *Model Checking*. MIT Press. ISBN: 0262032708.

[17] David L Dill, Andreas J Drexler, Alan J Hu, and C Han Yang. 1992. Protocol verification as a hardware design aid. In *ICCD*. Vol. 92. Citeseer, 522–525.

[18] Niklas Eén and Niklas Sörensson. 2004. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing*. Enrico Giunchiglia and Armando Tacchella, (Eds.)

[19] Xing Fang, Jaejin Lee, and Samuel P. Midkiff. 2003. Automatic fence insertion for shared memory multiprocessing. In *Proceedings of the 17th Annual International Conference on Supercomputing*.

[20] Harry D. Foster. 2015. Trends in functional verification: a 2014 industry study. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*.

[21] Natalia Gavrilenko, Hernán Ponce-de-León, Florian Furbach, Keijo Heljanko, and Roland Meyer. 2019. Bmc for weak memory models: relation analysis for compact smt encodings. In *Computer Aided Verification*. Isil Dillig and Serdar Tasiran, (Eds.)

[22] R. Guanciale, H. Nemati, C. Baumann, and M. Dam. 2016. Cache storage channels: alias-driven attacks and verified countermeasures. *2016 IEEE Symposium on Security and Privacy (S&P)*.

[23] Chian-Min Richard Ho. 1997. *Validation Tools for Complex Digital Designs*. Ph.D. Dissertation.

[24] R.C. Ho and M.A. Horowitz. 1996. Validation coverage analysis for complex digital designs. In *Proceedings of International Conference on Computer Aided Design*.

[25] Yao Hsiao, Dominic P. Mulligan, Nikos Nikoleris, Gustavo Petri, and Caroline Trippel. 2021. Rtl2uspec. https://github.com/yaohsiaopid/rtl2uspec. (2021).

[26] IBM. 1983. *IBM System/370 Principles of Operation*.

[27] Michalis Kokologiannakis, Ori Lahav, Konstantinos Sagonas, and Viktor Vafeiadis. 2017. Effective stateless model checking for c/c++ concurrency. *Proc. ACM Program. Lang.*, POPL.

[28] Michalis Kokologiannakis, Azalea Raad, and Viktor Vafeiadis. 2019. Model checking for weakly consistent libraries. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*.

[29] Michalis Kokologiannakis and Viktor Vafeiadis. 2021. Genmc: a model checker for weak memory models. In *Computer Aided Verification*.

[30] Michalis Kokologiannakis and Viktor Vafeiadis. 2020. HMC: model checking for hardware memory models. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[31] Robert P Kurshan and Ken McMillan. 1989. A structural induction theorem for processes. In *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, 239–247.

[32] Ori Lahav and Roy Margalit. 2019. Robustness against release/acquire semantics. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*.

[33] Ori Lahav, Viktor Vafeiadis, Jeehoon Kang, Chung-Kil Hur, and Derek Dreyer. 2017. Repairing sequential consistency in C/C++11. *38th Conference on Programming Language Design and Implementation (PLDI)*.

[34] Daniel Lustig, Simon Cooksey, and Olivier Giroux. 2022. Mixed-proxy extensions for the nvidia ptx memory consistency model: industrial product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*.

[35] Daniel Lustig, Michael Pellauer, and Margaret Martonosi. 2014. PipeCheck: specifying and verifying microarchitectural enforcement of memory consistency models. *Proceedings of the* 47th *International Symposium on Microarchitecture (MICRO)*.

[36] Daniel Lustig, Geet Sethi, Margaret Martonosi, and Abhishek Bhattacharjee. 2016. COATCheck: verifying memory ordering at the hardware-OS interface. *Proceedings of the* 21st *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[37] Albert Magyar. 2016. A Verilog implementation of the RISC-V Z-scale microprocessor. https://github.com/ucb-bar/vscale. (2016).

[38] Yatin A. Manerkar, Daniel Lustig, Margaret Martonosi, and Aarti Gupta. 2018. PipeProof: automated memory consistency proofs for microarchitectural specifications. *Proceedings of the* 51st *International Symposium on Microarchitecture (MICRO)*.

[39] Yatin A. Manerkar, Daniel Lustig, Margaret Martonosi, and Michael Pellauer. 2017. RTLCheck: verifying the memory consistency of RTL designs. *Proceedings of the* 50th *International Symposium on Microarchitecture (MICRO)*.

[40] Yatin A. Manerkar, Daniel Lustig, Margaret Martonosi, and Michael Pellauer. 2017. RTLCheck: verifying the memory consistency of RTL designs. https://github.com/ymanerka/rtlcheck. (2017).

[41] Yatin A. Manerkar, Daniel Lustig, Michael Pellauer, and Margaret Martonosi. 2015. CCICheck: using μhb graphs to verify the coherence-consistency interface. *Proceedings of the* 48th *International Symposium on Microarchitecture (MICRO)*.

[42] Yatin A. Manerkar, Caroline Trippel, Daniel Lustig, Michael Pellauer, and Margaret Martonosi. 2016. Counterexamples and proof loophole for the C/C++ to POWER and Armv7 trailing-sync compiler mappings. *CoRR*, abs/1611.01507. http://arxiv.org/abs/1611.01507. http://arxiv.org/abs/1611.01507.

[43] Zohar Manna and Amir Pnueli. 1995. *Temporal Verification of Reactive Systems*. Springer. ISBN: 978-1-4612-8701-8.

[44] Jeremy Manson, William Pugh, and Sarita Adve. 2005. The Java memory model. *32nd Symposium on Principles of Programming Languages (POPL)*.

[45] Martonosi Research Group. 2017. Check: research tools and papers. http://check.cs.princeton.edu. (2017).

[46] Kyndylan Nienhuis, Kayvan Memarian, and Peter Sewell. 2016. An operational semantics for C/C++11 concurrency. *31st International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*.

[47] NVIDIA. 2017. Parallel thread execution ISA version 6.0. http://docs.nvidia.com/cuda/parallel-thread-execution/index.html. (2017).

[48] Jonas Oberhauser et al. 2021. Vsync: push-button verification and optimization for synchronization primitives on weak memory models. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*.

[49] Scott Owens, Susmit Sarkar, and Peter Sewell. 2009. A better x86 memory model: x86-TSO. *Proceedings of the* 22nd *International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*.

[50] Gustavo Petri, Jan Vitek, and Suresh Jagannathan. 2015. Cooking the books: formalizing JMM implementation recipes. *29th European Conference on Object-Oriented Programming (ECOOP)*.

[51] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*.

[52] Hernán Ponce-de-León, Florian Furbach, Keijo Heljanko, and Roland Meyer. 2017. Portability analysis for weak memory models porthos: one tool for all models. In *Static Analysis*. Francesco Ranzato, (Ed.)

[53] Christopher Pulte, Shaked Flur, Will Deacon, Jon French, Susmit Sarkar, and Peter Sewell. 2017. Simplifying Arm concurrency: multicopy-atomic axiomatic and operational models for Armv8. *ACM Programming Languages*.

[54] Susmit Sarkar, Kayvan Memarian, Scott Owens, Mark Batty, Peter Sewell, Luc Maranget, Jade Alglave, and Derek Williams. 2012. Synchronising C/C++ and POWER. *33rd Conference on Programming Language Design and Implementation (PLDI)*.

[55] Jaroslav Ševčık and David Aspinall. 2008. On validity of program transformations in the Java memory model. *Proceedings of the 22Nd European Conference on Object-Oriented Programming*. ECOOP '08.

[56] Peter Sewell. 2016. C/C++11 mappings to processors. https://www.cl.cam.ac.uk/~pes20/cpp/cpp0xmappings.html.

[57] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. 2000. Checking safety properties using induction and a sat-solver. In *Formal Methods in Computer-Aided Design: Third International Conference, FMCAD 2000 Austin, TX, USA, November 1–3, 2000 Proceedings 3*. Springer, 127–144.

[58] Daniel Sorin, Mark Hill, and David Wood. 2011. *A Primer on Memory Consistency and Cache Coherence*. Mark Hill, (Ed.) *Synthesis Lectures on Computer Architecture*. Morgan and Claypool Publishers.

[59] Tommy Tracy, Lucas M. Tabajara, Moshe Vardi, and Kevin Skadron. 2020. Runtime verification on fpgas with ltlf specifications. In *2020 Formal Methods in Computer Aided Design (FMCAD)*, 36–46. DOI: 10.34727/2020/isbn.978-3-85448-042-6_10.

[60] Caroline Trippel, Yatin A. Manerkar, Daniel Lustig, Michael Pellauer, and Margaret Martonosi. 2017. TriCheck: memory model verification at the trisection of software, hardware, and ISA. *Proceedings of the* 22nd *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[61] Viktor Vafeiadis, Thibaut Balabonski, Soham Chakraborty, Robin Morisset, and Francesco Zappa Nardelli. 2015. Common compiler optimisations are invalid in the C11 memory model and what we can do about it. *42nd Symposium on Principles of Programming Languages (POPL)*.

[62] Viktor Vafeiadis and Chinmay Narayan. 2013. Relaxed separation logic: a program logic for C11 concurrency. *28th International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*.

[63] Viktor Vafeiadis and Francesco Zappa Nardelli. 2011. Verifying fence elimination optimisations. In *Static Analysis*.

[64] Andrew Waterman and Krste Asanović. 2019. The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA Document, Version 20190608-Base-Ratified. Tech. rep. SiFive Inc. and CS Division, EECS Department, University of California, Berkeley, (June 2019). https://riscv.org/specifications/.

[65] Pierre Wolper and Vinciane Lovinfosse. 1990. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems: International Workshop, Grenoble, France June 12–14, 1989 Proceedings 1*. Springer, 68–80.

[66] Florian Zaruba and Luca Benini. 2019. CVA6 RISC-V CPU. https://github.com/openhwgroup/cva6. (2019).