Report from Dagstuhl Seminar 21462

Foundations of Persistent Programming

Edited by

Hans-J. Boehm[1], Ori Lahav[2], and Azalea Raad[3]

1 **Google – Mountain View, US,** boehm@acm.org
2 **Tel Aviv University, IL,** orilahav@tau.ac.il
3 **Imperial College London, GB,** azalea.raad@imperial.ac.uk

Abstract

Although early electronic computers commonly had persistent core memory that retained its contents with power off, modern computers generally do not. DRAM loses its contents when power is lost. However, DRAM has been difficult to scale to smaller feature sizes and larger capacities, making it costly to build balanced systems with sufficient amounts of directly accessible memory. Commonly proposed replacements, including Intel's Optane product, are once again persistent. It is however unclear, and probably unlikely, that the fastest levels of the memory hierarchy will be able to adopt such technology. No such non-volatile (NVM) technology has yet taken over, but there remains a strong economic incentive to move hardware in this direction, and it would be disappointing if we continued to be constrained by the current DRAM scaling.

Since current computer systems often invest great effort, in the form of software complexity, power, and computation time, to "persist" data from DRAM by rearranging and copying it to persistent storage, like magnetic disks or flash memory, it is natural and important to ask whether we can leverage persistence of part of primary memory to avoid this overhead. Such efforts are complicated by the fact that real systems are likely to remain only partially persistent; some memory components, like processor caches and device registers. may remain volatile.

This seminar focused on various aspects of programming for such persistent memory systems, ranging from programming models for reasoning about and formally verifying programs that leverage persistence, to techniques for converting existing multithreaded programs (particularly, lock-free ones) to corresponding programs that also directly persist their state in NVM. We explored relationships between this problem and prior work on concurrent programming models.

Seminar November 14–17, 2021 – http://www.dagstuhl.de/21462

2012 ACM Subject Classification Theory of computation → Concurrency; Hardware → Non-volatile memory; Theory of computation → Program semantics

Keywords and phrases concurrency; non-volatile-memory; persistency; semantics; weak memory models

Digital Object Identifier 10.4230/DagRep.11.10.94

Edited in cooperation with Michalis Kokologiannakis

Foundations of Persistent Programming, *Dagstuhl Reports*, Vol. 11, Issue 10, pp. 94–110
Editors: Hans-J. Boehm, Ori Lahav, and Azalea Raad

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Executive Summary

Hans-J. Boehm (Google – Mountain View, US, boehm@acm.org)
Ori Lahav (Tel Aviv University, IL, orilahav@tau.ac.il)
Azalea Raad (Imperial College London, GB, azalea.raad@imperial.ac.uk)

We brought together 15 in-person attendees at Schloss Dagstuhl, with a roughly equal number of remote attendees. Remote attendance was challenging, particularly for attendees from very different time-zones. It nonetheless provided the opportunity for us to hear from a wider selection of participants.

We decided up-front not to try to cater the schedule to remote participation. Given the number of time zones covered by the participants, we continue to believe that, although it clearly had adverse impacts, it was the right decision.

We had a number of remote presentations that included interactions with the speakers. Otherwise the discussion tended to happen mostly among the in-person participants, in spite of the excellent AV systems at Dagstuhl. Many remote participants were limited in attendance due to time-zone issues.

Our area is perhaps unique, in that it includes deep theoretical work, but is also very dependent on technological developments. Accordingly, the participants of the seminar were from a spectrum of topics ranging from theory of distributed systems to hardware specification and design. The seminar gave many of us the opportunity to catch up on both theory and practice, including input from some participants with more direct insights into industrial developments.

We began the seminar by reviewing some of the underlying assumptions that were made by prior work in this field, often without certainty about their correctness. We were actually able to get much more shared clarity on a few of these as a result of audience discussion during the seminar. Some of us learned that non-volatile caches are being publicly discussed by Intel, and that there also is similar agreement that memory encryption, to restore volatility when needed, is desirable. We also learned that writes to the same cache line are not just believed by software researchers to reach memory in the correct order, but at least one hardware vendor also agrees. Though this last fact is rather obscure, it is important for some NVM algorithms, and not normally reflected in hardware manuals.

The rest of the seminar consisted of talks and group discussions. Three talks were longer overview talks on different aspects (Michael Scott on buffered persistency, Parosh Aziz Abdulla on verification, and Erez Petrank on persistent lock-free data structures). We did not feel the need for smaller break-out sessions, since the in-person group was quite small, largely due to our timing with respect to Covid waves. Much of the benefit here appears to have been in listening to discussions that often were either significantly more theoretical or significantly more practical than our own research.

NVM programming is both complicated by, and often synergistic with concurrent programming. Much of our focus was on the interaction between the two. Due to these close interactions, we asked several speakers to talk about concurrency issues that seemed particularly relevant (e.g., Peter Sewell on Armv8-A virtual memory model, Mark Batty on novel solution to the "thin air problem", and Paul McKenney on weak memory schemes used in the Linux kernel).

Several talks raised questions on the foundations of the field, such as what are the hardware-supplied programming models, or what it means for a persistent program to be

correct. It is entirely possible that most future NVM programmers will be more concerned with something like the persistent transactions discussed in Michael Bond's talk. However, given the relative immaturity of the area and foundational uncertainty, the emphasis seemed appropriate.

2 Table of Contents

3 Overview of Talks

3.1 Recoverable Self-Implementations of Primitive Operations

Hagit Attiya (Technion – Haifa, IL)

Joint work of Liad Nahum, Hagit Attiya, Ohad Ben-Baruch, Denny Hendler
Main reference Liad Nahum, Hagit Attiya, Ohad Ben-Baruch, Danny Hendler: "Recoverable and Detectable Fetch&Add". OPODIS 2021, LIPIcs Vol. 217
URL https://doi.org/10.4230/LIPIcs.OPODIS.2021.29

An attractive way to derive recoverable programs is to substitute every invocation of a primitive operation with a recoverable *self-implementation* of the same primitive.

We explore the properties that should be satisfied by such implementations for this approach to work. We also present some positive (implementations) and negative (impossibility) results, for primitives such as `test&set`, `fetch&add`, and `compare&swap`.

3.2 Consistency and Persistency: Challenges and Opportunities in Program Verification

Parosh Aziz Abdulla (Uppsala University, SE)

Joint work of Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, Prakash Saivasan
Main reference Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, Prakash Saivasan: "Deciding reachability under persistent x86-TSO", Proc. ACM Program. Lang., Vol. 5(POPL), pp. 1–32, 2021.
URL http://dx.doi.org/10.1145/3434337

Nowadays, most application platforms offer more relaxed semantics than the classical Sequential Consistency (SC) semantics. There are two primary sources of relaxation, namely weak consistency and weak persistence. Weakly consistent platforms are present at all level of the system design: at the hardware level, e.g., multiprocessors such as x86-TSO, SPARC, IBM POWER, and ARM; at the language level, e.g., C11 or Java; and at the application level, e.g., distributed databases, and geo-replicated systems. All these platforms sacrifice SC to provide stronger efficiency guarantees.

Weak persistence means that the order in which data persist over system crashes is inconsistent with the order in which the data is generated by the application. Weakly persistent systems arise in intermittent computing, file systems, and (more recently) architectures that employ Nonvolatile memories (NVRAMs).

Concurrent programs exhibit entirely new behaviors compared to SC when running on platforms with relaxed semantics. Even textbook programs such as small mutual exclusion protocols or concurrent data structures that are provably correct under SC can now show counter-intuitive behaviors. Hence, the verification community is currently facing new exciting, complicated, and practically motivated challenges.

To make the ideas concrete, I will present the semantics of concurrent programs that run on the Persistent Intel x86 architecture (Px86), implemented in Intel's Optane memory chip. We investigate the state reachability problem and show how to prove its decidability for finite-state programs. To achieve that, we provide a new formal model that is equivalent to Px86, and that has the feature of being a well-structured system. Deriving this new model results from a deep investigation of the properties of Px86 and the interplay of its component.

3.3 Isolating the Thin Air Problem: Semantic Dependency for Optimised Concurrency

Mark Batty (University of Kent – Canterbury, GB)

Joint work of Marco Paviotti, Simon Cooksey, Anouk Paradis, Daniel Wright, Scott Owens, Brijesh Dongol, Alan Jeffrey, James Riely, Ilya Kaysin, Anton Podkopaev, Mark Batty
Main reference Alan Jeffrey, James Riely, Mark Batty, Simon Cooksey, Ilya Kaysin, Anton Podkopaev: "The leaky semicolon: compositional semantic dependencies for relaxed-memory concurrency", Proc. ACM Program. Lang., Vol. 6(POPL), pp. 1–30, 2022.
URL http://dx.doi.org/10.1145/3498716

Languages like C/C++ (and Java) include both aggressive sequential optimisation and unmediated concurrent access to memory. These features collude to produce the out-of-thin-air problem – the language definition allows the conjuring of erroneous values in concurrent executions that can never be seen in practice, wrecking our ability to reason about concurrent code. Previous solutions involve disallowing optimisations at a cost to performance, or rebasing the language semantics on a very different model – an unlikely route for a language specification like C/C++. Here we present a less intrusive fix: an oracle takes the program and calculates semantic dependency, a record of which thread-local dependencies are preserved, and this dependency relation is used in the standard concurrency model of C/C++. We highlight recent work presenting two options for the calculation of semantic dependency, and a prospective operational model and program logic built upon semantic dependency. We believe this approach is reusable in the context of persistent memory.

3.4 Persistent Transactions: Desirable Semantics and Efficient Designs

Michael D. Bond (Ohio State University – Columbus, US)

Joint work of Kann Genç, Guoqing Harry Xu, Michael D. Bond

I'll argue that persistent transactions should provide failure atomicity and strict durability and respect inter-thread dependencies, under an assumption of data race freedom. Is this behavior reasonable and sufficient? And how do we implement such persistent transactions efficiently? I'll suggest that a combination of shadow memory, redo logs, and reference-counting-based dependence tracking yields a simple design that is likely more efficient than prior designs, although the performance story is complicated by tradeoffs of using shadow memory. Looking forward to your feedback and discussion.

3.5 Model Checking Persistent Memory Programs

Brian Demsky (University of California – Irvine, US)

Joint work of Hamed Gorjiara, Guoqing Harry Xu, Brian Demsky
Main reference Hamed Gorjiara, Guoqing Harry Xu, Brian Demsky: "Jaaru: efficiently model checking persistent memory programs", in Proc. of the ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021, pp. 415–428, ACM, 2021.
URL http://dx.doi.org/10.1145/3445814.3446735

Persistent memory (PM) technologies combine near DRAM performance with persistency and open the possibility of using one copy of a data structure as both a working copy and a persistent store of the data. Ensuring that these persistent data structures are crash consistent is a major challenge. Stores to persistent memory are not immediately made persistent — they initially reside in processor cache and are only written to PM when a flush occurs due to space constraints or explicit flush instructions. It is more challenging to test crash consistency for PM than for disks given the PM's byte-addressability that leads to significantly more states.

I present Jaaru, a fully-automated and efficient model checker for PM programs. Key to Jaaru's efficiency is a new technique based on constraint refinement that can reduce the number of executions that must be explored by many orders of magnitude. This exploration technique effectively leverages commit stores, a common coding pattern, to reduce the model checking complexity from exponential in the length of program executions to quadratic. We have evaluated Jaaru with PMDK and RECIPE, and found 25 persistency bugs, 18 of which are new.

3.6 View-Based Owicki–Gries Reasoning for Persistent x86-TSO

Brijesh Dongol (University of Surrey – Guildford, GB)

Joint work of Eleni Bila, Brijesh Dongol, Ori Lahav, Azalea Raad, John Wickerson
Main reference Eleni Bila, BBrijesh Dongol, Ori Lahav, Azalea Raad, John Wickerson: "View-Based Owicki–Gries Reasoning for Persistent x86-TSO". ESOP 2022 (To appear)

This work develops a program logic for reasoning about persistent x86 code that uses low-level operations such as memory accesses, fences, and flushes. Our logic, called Pierogi, benefits from an underlying operational semantics by Cho et al that is based on views. Pierogi is able to handle optimised flush operations that previous program logics for persistency could not. Pierogi is mechanised in the Isabelle/HOL proof assistant, which serves as a semi-automated verification tool. This talk will discuss the basics of Pierogi, its use in program verification, and its encoding in Isabelle/HOL.

3.7 General Constructions for Non-Volatile Memory

Michal Friedman (Technion – Haifa, IL)

Joint work of Michal Friedman, Erez Petrank, Guy E. Blelloch, Pedro Ramalhete, Naama Ben David, Yuanhao Wei
Main reference Michal Friedman, Erez Petrank, Pedro Ramalhete: "Mirror: making lock-free data structures persistent", in Proc. of the PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021, pp. 1218–1232, ACM, 2021.
URL http://dx.doi.org/10.1145/3453483.3454105

With the recent launch of the Intel Optane memory platform, non-volatile main memory in the form of fast, dense, byte-addressable non-volatile memory has now become available. Nevertheless, designing crash-resilient data structures is complex and error-prone, especially when caches and machine registers are still volatile and the data residing in memory after a crash might not reflect a consistent view of the program state. This talk will focus on NVTraverse and Mirror, which are two different general transformations that adds durability in an automatic manner to lock-free data structures, with a low performance overhead.

3.8 Formal Foundations for Intermittent Computing

Limin Jia (Carnegie Mellon University – Pittsburgh, US)

Joint work of Milijana Surbatovich, Brandon Lucia, Limin Jia
Main reference Milijana Surbatovich, Brandon Lucia, Limin Jia: "Towards a formal foundation of intermittent computing", Proc. ACM Program. Lang., Vol. 4(OOPSLA), pp. 163:1–163:31, 2020.
URL http://dx.doi.org/10.1145/3428231

Intermittently powered devices enable new applications in harsh or remote environments, e.g., space or in-body implants, but also introduce problems in programmability and correctness. A variety of intermittent systems to save and restore program state have been proposed to ensure complete program execution despite power failures. For such systems to execute programs correctly, non-volatile memory locations, whose writes may cause intermittent executions to diverge from continuously powered executions, need to be handled carefully. In this talk, I will present our work on formalizing and proving the correctness properties of intermittent systems and discuss our ongoing work targeting Rust.

3.9 Revamping Hardware Persistency Models: View-Based and Axiomatic Persistency Models for Intel-X86 and Armv8

Jeehoon Kang (KAIST – Daejeon, KR)

Joint work of CKyeongmin Cho, Sung Hwan Lee, Azalea Raad, Jeehoon Kang
Main reference Kyeongmin Cho, Sung Hwan Lee, Azalea Raad, Jeehoon Kang: "Revamping hardware persistency models: view-based and axiomatic persistency models for Intel-x86 and Armv8", in Proc. of the PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021, pp. 16–31, ACM, 2021.
URL http://dx.doi.org/10.1145/3453483.3454027

Non-volatile memory (NVM) is a cutting-edge storage technology that promises the performance of DRAM with the durability of SSD. Recent work has proposed several persistency models for mainstream architectures such as Intel-x86 and Armv8, describing the order in which writes are propagated to NVM. However, these models have several limitations; most notably, they either lack operational models or do not support persistent synchronization patterns.

We close this gap by revamping the existing persistency models. First, inspired by the recent work on promising semantics, we propose a unified operational style for describing persistency using views, and develop view-based operational persistency models for Intel-x86 and Armv8, thus presenting the first operational model for Armv8 persistency. Next, we propose a unified axiomatic style for describing hardware persistency, allowing us to recast and repair the existing axiomatic models of Intel-x86 and Armv8 persistency. We prove that our axiomatic models are equivalent to the authoritative semantics reviewed by Intel and Arm engineers. We further prove that each axiomatic hardware persistency model is equivalent to its operational counterpart. Finally, we develop a persistent model checking algorithm and tool, and use it to verify several representative examples.

This talk is based on a conference paper [1].

References

1 Kyeongmin Cho, Sung-Hwan Lee, Azalea Raad, and Jeehoon Kang. *Revamping hardware persistency models: view-based and axiomatic persistency models for Intel-x86 and Armv8*. PLDI 2021). DOI: `https://doi.org/10.1145/3453483.3454027`

3.10 Abstraction for Crash-Resilient Objects

Artem Khyzha (Arm – Cambridge, GB)

Joint work of Artem Khyzha, Ori Lahav
Main reference Artem Khyzha, Ori Lahav: "Abstraction for Crash-Resilient Objects (Extended Version)", CoRR, Vol. abs/2111.03881, 2021.
URL https://arxiv.org/abs/2111.03881

In this talk we discuss formal compositional reasoning under non-volatile memory. We develop a library correctness criterion that is sound for ensuring contextual refinement in this setting, thus allowing clients to reason about library behaviors in terms of their abstract specifications, and library developers to verify their implementations against the specifications abstracting away from particular client programs.

We employ a recent NVM model, called Persistent Sequential Consistency, as a semantic foundation, and extend its language and operational semantics with useful specification constructs. The proposed correctness criterion accounts for NVM-related interactions between client and library code due to explicit persist instructions, and for calling policies enforced by libraries.

3.11 Taming x86-TSO Persistency

Artem Khyzha (Arm – Cambridge, GB)

Joint work of Artem Khyzha, Ori Lahav
Main reference Artem Khyzha, Ori Lahav: "Taming x86-TSO persistency", Proc. ACM Program. Lang., Vol. 5(POPL), pp. 1–29, 2021.
URL http://dx.doi.org/10.1145/3434328

We study the formal semantics of non-volatile memory in the x86-TSO architecture. We show that while the explicit persist operations in the recent model of Raad et al. from POPL'20 only enforce order between writes to the non-volatile memory, it is equivalent, in terms of reachable states, to a model whose explicit persist operations mandate that prior writes are actually written to the non-volatile memory. The latter provides a novel model that is much closer to common developers' understanding of persistency semantics. We further introduce a simpler and stronger sequentially consistent persistency model, develop a sound mapping from this model to x86, and establish a data-race-freedom guarantee providing programmers with a safe programming discipline. Our operational models are accompanied with equivalent declarative formulations, which facilitate our formal arguments, and may prove useful for program verification under x86 persistency.

3.12 PerSeVerE: Persistency Semantics for Verification under Ext4

Michalis Kokologiannakis (MPI-SWS – Kaiserslautern, DE)

Joint work of Michalis Kokologiannakis, Ilya Kaysin, Azalea Raad, Viktor Vafeiadis
Main reference Michalis Kokologiannakis, Ilya Kaysin, Azalea Raad, Viktor Vafeiadis: "PerSeVerE: persistency semantics for verification under ext4", Proc. ACM Program. Lang., Vol. 5(POPL), pp. 1–29, 2021.
URL http://dx.doi.org/10.1145/3434324

Although ubiquitous, modern filesystems have rather complex behaviours that are hardly understood by programmers and lead to severe software bugs such as data corruption.

As a first step to ensure correctness of software performing file I/O, we have formalized the semantics of the Linux ext4 filesystem, which we have integrated with the weak memory consistency semantics of C/C++. In addition, we have developed an effective model checking approach for verifying programs that use the filesystem. While doing so, we discovered bugs in commonly-used text editors such as vim, emacs and nano.

My talk gives an overview of ext4's persistency semantics and our formalization, as well as of some editor bugs we found.

3.13 Weak Memory Schemes Used in the Linux Kernel

Paul McKenney (Facebook – Beaverton, US)

This talk gives background on persistent-memory mechanisms provided by the Linux kernel. The kernel generally does not rely on persistent memory internally because this would make it difficult to run the kernel on systems lacking persistent memory.

This talk then explains why many researchers have encountered significant performance penalties associated with strong ordering and persistent memory, showing how this is due to the finite speed of light (especially in solids), the non-zero size of atoms, protocol overheads (for example, due to cache coherence protocols), overheads from electrical fundamentals, and, in some cases, chemistry. These penalties are of special interest to organizations having large numbers of systems, where even a 1% increase in overhead can be quite expensive.

Further, the finite speed of light can make it impossible to determine the order in which external events occurred, in which case strong ordering might not be particularly helpful.

The talk concludes with a survey of a few of the weakly ordered concurrent algorithms used in the Linux kernel.

3.14 Hazard Pointer Synchronous Reclamation

Maged M. Michael (Facebook – New York, US)

Deferred reclamation techniques, such as hazard pointers, do not guarantee the timing of reclamation of reclaimable objects by default. This talk describes the problem of synchronous deferred reclamation, where users require guarantees for the timing of reclamation. This talk also reviews the semantics of various synchronous reclamation guarantees, and outlines cohort-based reclamation, a novel scalable algorithm for hazard pointer synchronous reclamation.

3.15 Persistent Lock-Free Data Structures for Non-Volatile Memory

Erez Petrank (Technion – Haifa, IL)

In this talk I will discuss the design of lock-free (concurrent) data structures adequate for non-volatile RAM. I will shortly review constructions of persistent queues and sets, mention general transformation and discuss the basic techniques behind all.

3.16 TSOPER: Efficient Coherence-Based Strict Persistency

Konstantinos Sagonas (Uppsala University, SE)

Joint work of Per Ekemark, Yuan Yao, Alberto Ros, Konstantinos Sagonas, Stefanos Kaxiras
Main reference Per Ekemark, Yuan Yao, Alberto Ros, Konstantinos Sagonas, Stefanos Kaxiras: "TSOPER: Efficient Coherence-Based Strict Persistency", in Proc. of the IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021, Seoul, South Korea, February 27 – March 3, 2021, pp. 125–138, IEEE, 2021.
URL http://dx.doi.org/10.1109/HPCA51647.2021.00021

We propose a novel approach for hardware-based strict TSO persistency, called TSOPER. We allow a TSO persistency model to freely coalesce values in the caches, by forming atomic groups of cachelines to be persisted. A group persist is initiated for an atomic group if any of its newly written values are exposed to the outside world. A key difference with prior work is that our architecture is based on the concept of a TSO persist buffer, that sits in parallel to the shared LLC, and persists atomic groups directly from private caches to NVM, bypassing the coherence serialization of the LLC. To impose dependencies among atomic groups that are persisted from the private caches to the TSO persist buffer, we introduce a sharing-list coherence protocol that naturally captures the order of coherence operations in its sharing lists, and thus can reconstruct the dependencies among different atomic groups entirely at the private cache level without involving the shared LLC. The combination of the sharing-list coherence and the TSO persist buffer allows persist operations and writes to non-volatile memory to happen in the background and trail the coherence operations. Coherence runs ahead at full speed; persistency follows belatedly. Our evaluation shows that TSOPER provides the same level of reordering as a program-driven relaxed model, hence, approximately the same level of performance, albeit without needing the programmer or compiler to be concerned about false sharing, data-race-free semantics, etc., and guaranteeing all software that can run on top of TSO, automatically persists in TSO.

3.17 The Case for Buffered Persistence

Michael Scott (University of Rochester, US)

Joint work of Wentao Cai, Haosen Wen, Vladimir Maksimovski, Mingzhe Du, Rafaello Sanna, Shreif Abdallah, Michael L. Scott, Benjamin Valpey, Louis Jenkins, H. Alan Beadle, Mohammad Hedayati, Joseph Izraelevitz, Hammurabi Mendes
Main reference Wentao Cai, Haosen Wen, Vladimir Maksimovski, Mingzhe Du, Rafaello Sanna, Shreif Abdallah, Michael L. Scott: "Fast Nonblocking Persistence for Concurrent Data Structures", in Proc. of the 35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference), LIPIcs, Vol. 209, pp. 14:1–14:20, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
URL http://dx.doi.org/10.4230/LIPIcs.DISC.2021.14

For machines with nonvolatile memory (NVM) but volatile caches, most work on persistent data structures has assumed the need for strict durable linearizability – for operations whose effects are guaranteed to survive any crash that occurs after their return to the caller. Most programmers, however, aren't interested in persisting existing transient structures: they're interested in avoiding serialization and deserialization of structures currently kept long-term in block-structured files and databases. For such structures, programmers are comfortable with the familiar concept of *buffering*, which separates persistence from atomicity and consistency, allowing data to persist some time in the (not-too-distant) future or in response to an explicit `sync` operation.

Given the high latency of fence instructions, buffered persistence has the potential to significantly shorten the critical path of the application. To evaluate this potential, we have created a general-purpose persistence system, Montage, that divides time into multi-millisecond *epochs*. Each epoch boundary is guaranteed to represent a consistent cut across the happens-before graph of operations. On a crash, recovery restores state as of the second-to-last epoch boundary. Experiments with both micro and macro benchmarks confirm that Montage dramatically outperforms existing general-purpose persistence systems, rivals or exceeds the performance of special-purpose persistent structures, and indeed approaches the performance of *nonpersistent* structures placed in NVM.

3.18 A Taste of Armv8-A Relaxed Virtual Memory

Peter Sewell (University of Cambridge, GB)

Joint work of Ben Simner, Alasdair Armstrong, Jean Pichon-Pharabod, Christopher Pulte, Peter Sewell

Virtual memory is an essential mechanism for enforcing security boundaries, but its relaxed-memory concurrency semantics has not previously been investigated in detail. The concurrent systems code managing virtual memory has been left on an entirely informal basis, and OS and hypervisor verification has had to make major simplifying assumptions.

This talk will give a taste of work in progress on relaxed virtual memory semantics for the Armv8-A architecture, to support future system-software verification. We identify many design questions, in discussion with Arm; develop a test suite, including use cases from the pKVM production hypervisor under development by Google; delimit the design space with axiomatic-style concurrency models; prove that under simple stable configurations our architectural model collapses to previous "user" models; develop tooling to compute allowed behaviours in the model integrated with the full Armv8-A ISA semantics; and develop a hardware test harness.

This lays out some of the main issues in relaxed virtual memory, bringing these security-critical systems phenomena into the domain of programming-language semantics and verification, with foundational architecture semantics, for the first time.

3.19 The ISA Semantics / Concurrency Model Interface

Peter Sewell (University of Cambridge, GB)

Joint work of Peter Sewell, Jean Pichon-Pharabod, Alasdair Armstrong, Ben Simner, Brian Campbell, Christopher Pulte, Thomas Bauereiss, Shaked Flur

We want to establish a standard interface between instruction-set architecture (ISA) semantics and concurrency models, to support the many things one would like to do above them – especially as one moves to concurrency models that handle more systems features, and to complete ISA definitions rather than idealised fragments. In this talk I'll spell out some of the desiderata and our current sketch design; hopefully it will stimulate discussion with potential users.

The interface shouldn't be large or complex, but it does have to harmonise several different usages and cope with all the many phenomena we care about, and it may underlie a lot of future work, so it's worth polishing it as much as we reasonably can up-front (though it will surely also have to change as it's used). It builds on our existing ISA/concurrency integrations in the rmem and isla-axiomatic tools, for full Armv8-A and RISC-V ISAs in Sail, and for user, ifetch, and virtual-memory concurrency, and on our Sail-generated Isabelle and Coq ISA semantics. We want to clean these up and generalise them, both for tools and for theorem-prover reasoning.

3.20 Non-Temporal Stores and their Semantics

Viktor Vafeiadis (MPI-SWS – Kaiserslautern, DE)

Joint work of Azalea Raad, Luc Maranget, Viktor Vafeiadis
Main reference Azalea Raad, Luc Maranget, Viktor Vafeiadis: "Extending Intel-x86 consistency and persistency: formalising the semantics of Intel-x86 memory types and non-temporal stores", Proc. ACM Program. Lang., Vol. 6(POPL), pp. 1–31, 2022.
URL http://dx.doi.org/10.1145/3498683

In the research community, there are several formalisations of the sequential semantics of various fragments of the Intel-x86 architecture and a few that also describe their weakly consistent concurrency semantics and/or their persistency semantics. As far as the concurrency semantics is concerned, Intel-x86 is widely believed to follow the x86-TSO model of Sewell et al. [1]. The persistency semantics is similarly believed to follow the Px86 model of Raad et al. [2].

Nevertheless, this is not the full story. The x86-TSO and Px86 models cover only a small fragment of its available features that are relevant for the consistency semantics of multi-threaded programs and the persistency semantics of programs interfacing with non-volatile memory. In particular, besides normal store instructions, Intel-x86 supports non-temporal stores, which provide higher performance and are used to ensure that updates are flushed to memory. Furthermore, Intel-x86 allows declaring regions of memory with different types—uncacheable, write-back, write-combined, write-protected, and write-through—and existing models cover only the semantics of the (default) write-back memory regions.

Since both non-temporal stores and memory regions with different types are widely used in systems code, Azalea Raad, Luc Maranget, and I have extended the existing x86-TSO and Px86 formalizations to cover these features and the interaction between them. Our formal model is published at POPL'22 [3]. My talk at the seminar presented an overview of these additional features: how they can be used and what semantics they have.

References

1 Peter Sewell, Susmit Sarkar, Scott Owens, Francesco Zappa Nardelli, and Magnus O. Myreen. *x86-TSO: a rigorous and usable programmer's model for x86 multiprocessors*. Commun. ACM 53(7), 2010. DOI: `https://doi.org/10.1145/1785414.1785443`

2 Azalea Raad, John Wickerson, Gil Neiger, and Viktor Vafeiadis. *Persistency semantics of the Intel-x86 architecture*. Proc. ACM Program. Lang. 4(POPL), 2020. DOI: `https://doi.org/10.1145/3371079`

3 Azalea Raad, Luc Maranget, and Viktor Vafeiadis. *Extending Intel-x86 consistency and persistency: formalising the semantics of Intel-x86 memory types and non-temporal stores*. Proc. ACM Program. Lang. 6(POPL), 2022. DOI: `https://doi.org/10.1145/3498683`

3.21 Architectural Support for Persistent Programming

William Wang (Arm – Cambridge, GB)

The talk opens with the use cases of NVM, including "more" memory and persistent memory, leveraging its density and persistence. Then the talk focuses on the persistent use, which requires software changes that bring programming challenges.

To reduce the barrier of entry for persistent memory and simplify persistent programming, we ask whether the Arm architecture has sufficient support for programming persistent memory. In searching for an answer, two problems related to ensuring persistent ordering across threads and within a thread are uncovered, and two solutions are outlined. Specifically, the persistent transitive stores at the instruction set architectural level and the battery-backed buffers at the microarchitectural level.

To wrap up, the talk briefly mentions about other persistent programming challenges, including failure atomicity, persistent addressing that can also benefit from architectural support.

3.22 Modularizing Verification of Durable Opacity

Heike Wehrheim (Universität Oldenburg, DE)

Joint work of Eleni Bila, John Derrick, Simon Doherty, Brijesh Dongol, Gerhard Schellhorn, Heike Wehrheim
Main reference Eleni Bila, John Derrick, Simon Doherty, Brijesh Dongol, Gerhard Schellhorn, Heike Wehrheim: "Modularising Verification Of Durable Opacity", CoRR, Vol. abs/2011.15013, 2020.
URL https://arxiv.org/abs/2011.15013

Opacity is the standard correctness condition for Software Transactional Memory Algorithms. One way of proving opacity is by showing a refinement relationship to hold between an abstract sequential specification and an implementation. Durable opacity is the analogue of opacity for settings with non-volatile memory. It can similarly be shown by refinement, now using a durable abstract specification. In the talk, I will present work towards (a) reusing existing refinement proofs of opacity for proving durable opacity, and (b) modularizing such proofs by capsulating all accesses to shared state in a library, and only showing this library to be durable linearizable.

Participants

- Mark Batty
University of Kent – Canterbury, GB
- Hans-J. Boehm
Google – Mountain View, US
- Michael D. Bond
Ohio State University – Columbus, US
- Brijesh Dongol
University of Surrey – Guildford, GB
- Michal Friedman
Technion – Haifa, IL
- Michalis Kokologiannakis
MPI-SWS – Kaiserslautern, DE
- Ori Lahav
Tel Aviv University, IL
- Maged M. Michael
Facebook – New York, US
- Erez Petrank
Technion – Haifa, IL
- Azalea Raad
Imperial College London, GB
- Konstantinos Sagonas
Uppsala University, SE
- Michael Scott
University of Rochester, US
- Viktor Vafeiadis
MPI-SWS – Kaiserslautern, DE
- William Wang
Arm – Cambridge, GB
- Heike Wehrheim
Universität Oldenburg, DE

Remote Participants

- Hagit Attiya
Technion – Haifa, IL
- Parosh Aziz Abdulla
Uppsala University, SE
- Piotr Balcer
Intel Technology – Gdansk, PL
- Eleni Bila
University of Surrey – Guildford, GB
- Dhruva Chakrabarti
AMD – Santa Clara, US
- Soham Chakraborty
TU Delft, NL
- Ricardo Ciríaco da Graca
MPI-SWS – Kaiserslautern, DE
- Brian Demsky
University of California – Irvine, US
- Derek Dreyer
MPI-SWS – Saarbrücken, DE
- João F. Ferreira
INESC-ID – Lisboa, PT
- Maurice Herlihy
Brown University – Providence, US
- Joseph Izraelevitz
University of Colorado – Boulder, US
- Limin Jia
Carnegie Mellon University – Pittsburgh, US
- Jeehoon Kang
KAIST – Daejeon, KR
- Artem Khyzha
Arm – Cambridge, GB
- Umang Mathur
National University of Singapore, SG
- Paul McKenney
Facebook – Beaverton, US
- Nikos Nikoleris
Arm – Cambridge, GB
- Andy Rudoff
Intel – Boulder, US
- Peter Sewell
University of Cambridge, GB
- John Wickerson
Imperial College London, GB