08241 Abstracts Collection
Transactional Memory: From Implementation to Application
— Dagstuhl Seminar —

Christof Fetzer[1], Tim Harris[2], Maurice Herlihy[3] and Nir Shavit[4]

[1] TU Dresden, DE
christof.fetzer@inf.tu-dresden.de
[2] Microsoft Research, Cambridge, GB
tharris@microsoft.com
[3] Brown Univ. - Providence, USA
herlihy@cs.brown.edu
[4] Tel Aviv University, IL
shanir@cs.tau.ac.il

Abstract. From 08.06. to 13.06.2008, the Dagstuhl Seminar 08241 "Transactional Memory: From Implementation to Application" was held in Schloss Dagstuhl – Leibniz Center for Informatics. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Multiprocessors, Multi-core machines, Concurrent Programming, Parallel Programming, Synchronization, Transactional Memory

08241 Summary – Transactional Memory : From Implementation to Application

A goal of current multiprocessor software design is to introduce parallelism into software applications by allowing operations that do not conflict in accessing memory to proceed concurrently. The key tool in designing concurrent data structures has been the use of locks. Unfortunately, course grained locking is easy to program with, but provides very poor performance because of limited parallelism. Fine-grained lock-based concurrent data structures perform exceptionally well, but designing them has long been recognized as a difficult task better left to experts. If concurrent programming is to become ubiquitous, researchers agree that one must develop alternative approaches that simplify code design and verification.

Dagstuhl Seminar Proceedings 08241
Transactional Memory: From Implementation to Application
http://drops.dagstuhl.de/opus/volltexte/2008/1775

Keywords: Multiprocessors, Multi-core machines, Concurrent Programming, Parallel Programming, Synchronization, Transactional Memory

Joint work of: Fetzer, Christof; Harris, Tim; Herlihy, Maurice; Shavit, Nir

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2008/1774

C/C++ Language Extensions for Transactions

Ali-Reza Adl-Tabatabai (Intel - Santa Clara, US)

I will talk about the C++ language extensions for transactions in the latest whatif.intel.com latest release of the Intel TM compiler.

Sequential Analysis For Serializability and Beyond

Hagit Attiya (Technion - Haifa, IL)

A concurrent module provides procedures that may be invoked concurrently by its clients. We consider concurrent modules that utilize locking protocols, such as two-phase locking (2PL) or tree locking (TL), to guarantee serializability. We present analyses for verifying that a given module satisfies TL/2PL. Such analyses are useful as serializability is a desirable property of interest to the end user (programmer).

However, there is another compelling factor that motivates us to pursue these analysis problems: if a concurrent module is verified to be serializable, then we can exploit this fact to perform other analyses of the module, e.g., verifying the absence of memory errors in the module, more efficiently and more precisely, by considering only serial executions of its procedures.

This approach (to using sequential reasoning for subsequent analyses) becomes less attractive if the verification of serializability itself requires reasoning explicitly about all possible interleaved executions of the module's procedures.

One of our main results is that the verification of the locking protocols (TL/2PL) can itself be done using a sequential analysis. We, in fact, prove this result for any locking protocol that satisfies certain reasonable conditions. Another contribution of this paper is that we present a couple of approaches to performing a sequential analysis of a concurrent module, and study the correctness conditions required by these approaches. One interesting outcome of this study is that one natural approach to sequential analysis actually requires a termination analysis for correctness.

Keywords: Serializability, 2 phase locking, tree locking, sequential verification

Joint work of: Attiya, Hagit; Ramalingam, Ganesan; Rinetzky, Noam; Sagiv, Mooly; Yahav, Eran; Bouajjani, Ahmed

STM for a Distributed Java VM in Sensor Networks

Annette Bieniusa (Universität Freiburg, DE)

The AmbiComp project develops a distributed Java virtual machine for embedded sensor and control units. In particular, it creates a single system illusion to simplify software development in settings of many physically distributed interacting controllers.

I would like to discuss how AmbiComp can use STM as a parallel programming paradigm.

Flexible Decoupled Transactional Memory Support

Sandhya Dwarkadas (University of Rochester, US)

A high-concurrency transactional memory (TM) implementation needs to track concurrent accesses, buffer speculative updates, and manage conflicts, requiring support for data isolation and memory monitoring (DIMM).

In this talk, I will present a system we call FlexTM (FLEXible Transactional Memory), that utilizes four decoupled hardware mechanisms that provide DIMM support: read and write signatures, which summarize per-thread access sets; per-thread conflict summary tables (CSTs), which identify the threads with which conflicts have occurred; Programmable Data Isolation, which maintains speculative updates in the local cache and employs a thread-private buffer (in virtual memory) in the rare event of overflow; and Alert-On-Update, which selectively notifies threads about coherence events.

All mechanisms are software-accessible, to enable virtualization across context switches, overflow, and page swaps.

FlexTM (1) decouples conflict detection from conflict management and allows software to control management time (i.e., eager or lazy); (2) tracks conflicts on a thread-by-thread basis rather than a location-by-location basis and enables software to dictate policy without the overhead of separate metadata; and (3) permits TM components to be used for non-transactional purposes. To the best of our knowledge, it is the first hardware TM to admit a distributed commit protocol with no global arbitration.

If time permits, I will present results to demonstrate the efficiency of the system, as well as the utility of its flexible policies.

I will also demonstrate how the DIMM mechanisms may be used for other non-transactional purposes such as program debugging.

Joint work of: Shriraman, Arrvindh; Dwarkadas, Sandhya; Scott, Michael L.

Obstruction-Free Algorithms can be Practically Wait-Free

Faith Ellen (University of Toronto, CA)

We present a transformation that converts any obstruction-free algorithm for an asynchronous shared memory system into a wait-free algorithm when the system is semi-synchronous, even if the bound between the relative speed of processes is unknown. Like pragmatic contention managers, our transformation has negligible overhead when there is no contention, but, unlike them, it also guarantees progress.

Joint work of: Ellen, Faith; Luchangco, Victor; Moir, Mark; Shavit, Nir

STM for speculative out-of-order event processing

Pascal Felber (Université de Neuchâtel, CH)

In event stream applications, events flow through a network of components that perform various types of operations, e.g., filtering, aggregation, transformation. When the operation only depends on the input events, one can trivially parallelize its processing by replicating the associated components. This is not possible, however, with stateful components or when there exist dependencies between the events. Parallel versions of a number of simple stream mining operators have been designed, but, in general, complex and user-defined operators are limited by single thread performance. In this paper, we propose leveraging the processing capabilities of multi-core processors to improve the efficiency of stateful components using optimistic parallelization techniques (as provided by transactional memory). We show that, even though some speculative event executions might need to be disregarded, the overall throughput increases noticeably in the general case and latency can be reduced by pre-processing out-of-order events. Moreover, we show how simple conflict predictors can boost the parallelism even more and reduce the amount of resources used for a given level of parallelism.

Keywords: Transactional memory, event processing

Joint work of: Felber, Pascal; Brito, Andrey; Fetzer, Christof; Sturzrehm, Heiko

Full Paper:
http://doi.acm.org/10.1145/1385989.1386023

See also: In Proceedings of the International Conference on Distributed Event-Based Systems (DEBS'08), Rome, Italy, July 2008.

My programming-languages view of TM: Research and Conjectures

Daniel Grossman (University of Washington, US)

I will give a very brief overview/advertisement of the TM research my group has been conducting over the last three years. This research approaches TM from a programming-languages perspective. We have focused on issues of motivation (why are transactions actually better), semantics (particularly strong vs. weak isolation), language design (how do transactions interact with other features), and language implementation (what compiler/runtime optimizations are helpful).

I will finish with some conjectures about the promise of TM technology that may help start informal discussions to help guide the TM community.

Transactional Memory Input Acceptance

Rachid Guerraoui (EPFL - Lausanne, CH)

We present a new metric to characterize Transactional Memories (TMs): the input acceptance. This metric represents, for a given TM, its ability to commit transactions depending on the interleaving (i.e., schedule) of their actions. Up to now, the main evaluation metric was the number of committed transactions by time unit (a.k.a. throughput), however, throughput does not capture the likeliness for a TM to commit a transaction.

Unlike throughput, the input acceptance of a TM indicates the quantity of given schedules for which the TM commits its transactions. The difficulty in designing a correct TM comes more from ensuring that some serializable transactions commit than from ensuring that non-serializable transactions abort.

We identify few TM designs shared by several existing TMs and we compare them along with this new metric. Our theoretical results, confirmed by experimental results, totally order the input acceptance of these designs.

Keywords: Input acceptance, Transactional memory, Theory, Performance

Joint work of: Gramoli, Vincent; Harmanci, Derin; Felber, Pascal

Pay-to-use strong atomicity

Tim Harris (Microsoft Research UK - Cambridge, GB)

I'll introduce a new way to provide "strong atomicity" in an implementation of atomic blocks using transactional memory.

Strong atomicity lets us offer clear semantics to programs, even if they access the same locations inside and outside atomic blocks.

It also avoids differences between hardware-implemented transactions and software-implemented ones. Our new idea is to use off-the-shelf page-level memory protection hardware to detect conflicts between normal memory accesses and transactional ones. The page-level system ensures correctness but gives poor performance because of the costs of manipulating memory protection hardware from user-mode and the costs of synchronizing protection settings between processors or cores. However, in practice, we show how a combination of careful object placement and dynamic code update allow us to eliminate almost all of the protection changes. Existing implementations of strong atomicity in software rely on detecting conflicts by conservatively treating some non-transacted accesses as short transactions. In contrast, our page-level technique provides a foundation that lets us be less conservative about how nontransacted accesses are treated; we avoid changes to non-transacted code until a possible conflict is detected dynamically, and we can respond to phase changes where a given instruction sometimes generates conflicts and sometimes does not. We evaluate our implementation with C# versions of many of the STAMP benchmarks.

Our implementation requires no changes to the operating system.

CAR-STM: Scheduling-Based Collision Avoidance and Resolution for Software Transactional Memory

Danny Hendler (Ben Gurion University, IL)

Transactional memory (TM) is a key concurrent programming abstraction.

Several software-based transactional memory (STM) implementations have been developed in recent years. All STM implementations must guarantee transaction atomicity but different STM implementations may provide different progress guarantees. In order to ensure progress, an STM implementation must resolve transaction conflicts. This is done either by the implementation itself (if it is *lock-free*) or by delegating conflict resolution to a separate *contention manager* module that tries to resolve transaction collisions once they are detected.

In this talk, we describe a novel approach for increasing STM efficiency: rather than handle collisions post factum, we propose proactive collision reduction by pre-assigning transactions that are more likely to collide to the same core.

We present CAR-STM, a scheduling-based mechanism for STM collision avoidance and resolution, that can be incorporated into existing STM implementations. In addition to proactive collision avoidance that is based on application-specific hints, CAR-STM's transaction scheduling supports novel and highly efficient contention managers that resolves conflicts by serializing the execution of colliding transactions.

We have incorporated CAR-STM into the University of Rochester's STM (RSTM) and compared the performance of the new implementation with that

of the original RSTM by using STMBench7. Our results show that the new implementation provides orders-of-magnitude reduction of execution times and improved throughput for almost all concurrency levels.

Additionally, since CAR-STM greatly reduces the unpredictable influence of operating-system scheduling on STM performance, the new implementation provides much more *stable performance*. In contrast, the performance of the original RSTM implementation on STMBench7 workloads exhibits extremely high variance. Though our current work focuses on software transactional memory, we believe the ideas introduced by CAR-STM may prove useful also for hybrid implementations of transactional memory.

Joint work of: Dolev, Shlomi; Hendler, Danny; Suissa, Adi

Are transactions concurrent enough?

Maurice Herlihy (Brown Univ. - Providence, US)

Most TM implementations synchronize on the basis of read/write sets. There is reason to believe that read/write synchronization unnecessarily restricts concurrency for data structurs such as hashmaps, queues, etc. I'd like to ask whether this is really a problem, and if so, what can be done about it.

Keywords: Concurrency

Locality in Concurrent Data Structures

Eshcar Hillel (Technion - Haifa, IL)

To reap the performance benefits of multi-core and multiprocessing systems, algorithms and data structures should accommodate concurrent access, without effectively sequentializing all operations.

We explain the concept of locality and describe a method for multi-word synchronization that increases concurrency and throughput.

Note on scheduling: I am arriving on Monday afternoon, and leaving on Friday (very early in the) morning.

Challenges and Directions for TM Research

Christos Kozyrakis (Stanford University, US)

The Transactional Memory (TM) research community has made great progress within the past five years.

We now have a reasonable understanding of how to implement TM with hardware or software, how to optimize TM code, how to argue about and prove TM semantics, how to manage contention, and how to reason about issues such as strong atomicity or composability. More important, the TM community has created excitement and attracted participation from multiple research domains including architecture, compilers, programming languages, operating systems, and distributed algorithms.

There are several low-level implementation issues for TM researchers to debate and investigate. Nevertheless, the goal of this talk is to initiate a discussion on the urgent high-level challenges for the TM community. The following five issues are often raised by colleagues outside of this community and are particular important for the long-term success of TM research. Some suggestions on how to address these issues based on past and current work at the TCC group at Stanford are also included:

1) How does TM fit with parallel programming environments?

Since TM cannot address on its own all the challenges of concurrency, it is important to consider how it fits within complete parallel programming environments. We have placed significant effort on integrating transactions with existing parallel idioms (C++ threads, Java threads, OpenMP, etc). It is now time to explore how transactions fit with other innovative ideas for parallel programming. One such idea is domain-specific languages that hide the complexities of concurrency using high-level, domain-specific abstractions.

In such an environment, transactions may simply be an implementation tool, hidden from the end programmer. The advantage of this approach is that we can limit the type and scope of transactions used in practice, avoiding the difficult cases of nesting, inter-transaction communication, etc.

2) How does TM fit in the stack of a modern computer system?

Related to (1), modern computing environments do not consist of just processors and memory. They also include I/O, networking, interprocess communication, distributed environment over cluster substrates, etc.

How does TM technology fit in such a stack? Can we provide atomicity and isolation as user code interacts with multiple system components? If yes, what are the semantics and what are the restrictions? So far, we have either ignored these issues or "stretched" TM to cover parts of the system functionality. An alternative approach is to consider system-scale transactions, where TM is just one of the many transactional components in the system. Similar to IBM's QuickSilver system, a transactional manager would coordinate the execution of user-level transactions across transactional components such as TM, log-based file systems, DBMS, and network queues.

(3) Does TM technology scale?

The only concurrency that matters is concurrency that scales. For TM to remain relevant, its language abstractions and implementations must scale from tens to hundred of thousands of threads. Virtually all TM implementations currently rely on coherent, shared memory, a technology that we are still not certain how to scale. On the other hand, transactions may be the abstraction that makes

inter-thread communication sufficiently coarse-grained in space and time so that coherent shared memory can scale to large numbers of threads.

(4) Can TM help with system challenges beyond concurrency?

Application developers are facing several challenges in addition to exploiting concurrency. Security, reliability and robustness, debugging and testing are a few of the many. The basic mechanisms of a TM system (data versioning, conflict detection, serializability enforcement) can potentially help simplify or improve solutions towards these challenges.

The opportunity for the TM community is that such uses may be the points that convince system vendors to deploy TM and application developers to actually use it. The challenge is to explore how such uses interact with transactions for concurrency control.

(5) How much easier does TM make parallel programming after all?

Last but definitely not least, for all our work on TM, we still have no quantitative data to support the main claim for TM research. Measuring ease of programming is an extremely difficult task, but it is also a task that we must undertake. We need to consider what are the user studies or deployments that we should put together to quantify some aspects of programmability. Apart from convincing critics, this will help us understand how programmers will actually use transactions, what are the common cases, what are the pitfalls, how useful transactions are beyond concurrency etc. Such work is much more important at this point than yet another optimization of some implementation aspect.

Hopefully, the Dagstuhl workshop can make some progress towards addressing these challenges.

Preparing Debuggers for Transactional Programs

Yossi Lev (Brown Univ. - Providence, US)

With the recent emergence of multiprocessors and multicore computers, Transactional Memory (TM) is becoming the programming API of choice for writing concurrent programs. The transactional programming model promises to simplify the task of writing concurrent, correct and scalable programs. In order to support this new model, debuggers will need to change. As far as we know, little work has been done on this front.

In this talk, we describe the development of libtm_db, the first library to provide debuggers with general debugging support for transactional programs. libtm_db is an open source, external library, designed to assist debuggers for transactional programs by isolating them from the internals of the particular runtime TM in use. The library is not targeted to a specific debugger, and can be extended to support various runtime TM systems. We hope that the library will assist debugger writers in the required shift to supporting transactional programs' debugging, and provide developers of new runtime TM systems with a well-defined interface for transactional debugging support.

Keywords: Debugging, Transactional Memory

What do we really want from transactional memory?

Victor Luchangco (Sun Microsystems Laboratories - Burlington, US)

There's been a lot of excellent work in improving transactional memory implementations, and even some in beginning to precisely define the semantics of transactional memory.

That's great, and we need to keep plugging at that. But we should also take a step back and figure out what we actually hope to achieve with transactional memory, and what properties transactional memory (and its implementations) must have to achieve our goals. For example (ignoring for now the lack of precise definitions for the following terms), can we efficiently implement strongly atomic linearizable unbounded transactional memory? If not, what are we willing to give up? Efficiency? Strong atomicity? Linearizability? Unboundedness? Some combination of these? And how should we decide? I think we should decide based on our goals for transactional memory, which we must therefore attempt to ennunciate clearly.

I'd like to begin this attempt and present some preliminary observations and directions that these suggest to me.

Profile of the Elusive TM Killer App

Maged Michael (IBM TJ Watson Research Center, US)

One of the main obstacles to the wide adoption of transactional memory is the interdependence between the justification for the cost of effective architectural support for TM and the performance impact of TM on important applications. Without showing clear positive impact of TM on the performance of important applications, it is difficult to justify architectural support of TM that goes beyond small transactions. On the other hand, software-only TM implementations have not delivered robust performance that can motivate application developers to adopt the TM programming model. After years of wide interest in TM, applications that can be clearly identified as TM killers apps remain elusive. In this talk, we explore the characteristics that would make an application a candidate for being a TM killer app, and what pitfalls reduce the value of TM to some applications. Three main application characteristics are identified as critical to identification as a TM killer app: (1) importance of performance, (2) inherent high concurrency, and (3) irregular multi-object transactional span.

A key purpose of TM is enabling concurrency. Therefore, the performance of a killer app must be important enough to warrant aggressive parallelization. TM enables concurrency but does not create it. Therefore, a killer app must have inherent concurrency–irrespective of concurrency limitations related to the use of specific synchronization such as lock contention. The importance of application performance combined with the high overheads of TM may justify investment in low-overhead fine-grain synchronization by expert programmers. Therefore,

a TM killer app must be very difficult to parallelize effectively and efficiently without TM. Transactions that span multiple data objects with irregular access patterns present a nearly impossible challenge to expert programmers to develop implementations that are both efficient and maintainable. The importance of code maintainability offered by TM due to its composability trump the performance advantage of fine-grain synchronization.

Towards Pragmatic Semantics for Transactional Memory

Mark Moir (Sun Microsystems Laboratories - Burlington, US)

I want to talk about pragmatic approaches to defining transactional memory semantics in the short to medium term, specifically for unsafe languages such as C and C++. Specific topics will include how supposedly-simple specifications such as Single Lock Atomicity address semantics of features not already included in the underlying language (such as explicit abort), progress properties, memory models, exceptions, etc.

Keywords: Semantics

Provably Correct Abstract Concurrency Control, And More

J. Eliot B. Moss (Univ. of Massachusetts - Amherst, US)

We offer a language for specifying the abstraction (model) that a Java class implements. Given specifications of conditions under which operations conflict, we can prove those specifications correct, and possibly derive them. We can also prove correctness of inverse (compensating) actions at the abstract level.

All this is possible because the model language is carefully designed to be powerful enough to specify interesting abstractions yet restricted enough to allow automated proofs of correctness. We briefly describe Set and OrderedSet abstractions and mention results of automated proofs to date. This technology overcomes the criticism of open nesting that abstract concurrency control is too difficult for programmers to get right.

The Other.pdf document is Trek Palmer's PhD proposal on this work.

Keywords: Open nesting, concurrency control

Joint work of: Moss, J. Eliot B.; Palmer, Trek

Pervasive Parallelism and Real Hardware Prototypes for TM

Kunle Olukotun (Stanford University, US)

We are now at a point in TM research were to move the field forward we must experiment with challenging applications and full-scale hardware prototypes.

In this talk I will describe the work we are doing in the Pervasive Parallelism Lab to develop environments for new challenging parallel applications and flexible hardware prototyping platforms for TM.

Putting the R back in ROI

Ravi Rajwar (Intel Corp. - Hillsboro, US)

Technology success is often based on real-world "Return on Investment".

So far, much TM research has focused on portions of the "I" and have largely ignored the more critical R question.

I will discuss this topic and present thoughts on how TM researchers can help.

Concurrency in Enterprise Systems

Asuman Suenbuel (SAP Research Labs - Palo Alto, US)

Concurrency and parallelism is getting more and more important also for business applications because most of the future gain in runtime efficiency will come from the use of multicore processors. Enterprise systems are usually very complex systems that have organically grown often over decades, and they are designed and optimized to run on single-core processors with no concurrency and parallelism involved.

The question that we would like to address is how concurrency theory can help applications so complex as enterprise systems make use of the full potential provided by the multicore processors without compromising the factors that are important to business applications: robustness, backwards-compatibility, scalability, data-safety, transactional integrity to only mention a few. We believe that enterprise systems due to their complexity in size and structure are a perfect playing field to validate the practicability of novel approaches in concurrency that have so far only be tested in a smaller scale.

TokenTM: Efficient Execution of Large Transactions with Hardware Transactional Memory

David A. Wood (Univ. Wisconsin - Madison, US)

I will talk about our recent work in efficient support for large transactions. I'm leaving early Friday.

Full Paper:
http://www.cs.wisc.edu/multifacet/papers/isca08_tokentm.pdf

Using Hardware Memory Protection to Build a High-Performance, Strongly-Atomic Hybrid TM

Craig Zilles (Univ. of Illinois - Urbana, US)

We demonstrate how hardware fine-grained memory protection can be used in support of transactional memory systems: first showing how a software transactional memory system (STM) can be made strongly atomic by using memory protection on transactionally-held state, then showing how such a strongly-atomic STM can be used with a bounded hardware TM system to build a hybrid TM system in which zero-overhead hardware transactions may safely run concurrently with potentially-conflicting software transactions.

In addition, I quickly survey 5 other topics: 1) that speculative compiler optimization is a second killer application for TM/SLE hardware, 2) the keys for good hybrid performance are: good hardware contention management and hardware feedback on the reason for an abort, 3) how important is concurrency within a transaction (I don't think hardware can help this case), 4) that single-thread performance is important for scalability because how fast each thread commits determines the level of contention where there are conflicts, and 5) a question for the community as to what the desired performance profile (perf. vs. transaction size); is it more important to have maximal performance for small and medium transactions, or is it more important to have smooth performance of transactions (no discontinuities) as memory footprints increase in size?

Keywords: HTM, Memory Protection, Strong Atomicity, Hybrid Transactional Memory, Primitives, STM

Full Paper:
http://www-faculty.cs.uiuc.edu/~zilles/papers/ufo_hybridTM.isca2008.pdf

See also: Lee Baugh, Naveen Neelakantam, and Craig Zilles