Software Transactional Memory : a Survey

Meenu
Associate Professor
Computer Science and Engineering Department,
Madan Mohan Malaviya University of Technology,
Gorakhpur.(Uttar Pradesh)

ABSTRACT

Software transactional memory (STM) is a programming abstract for shared variable concurrency. There are very few updates available in Haskell's STM implementation since its introduction in 2005 by Tim Harris et al. In this paper we present a concurrency model, based on Software transactional memory.

CCS Concepts: • Information systems → Database Management; Database transaction processing

KEYWORDS Software transactional memory (STM), Non-nested TVar, NestedTVar

1. Introduction

Today we are living in an era of multicore and multiprocessor systems . Parallel programming [27] is utilizing the full power of multicore processors. Parallel programming is more difficult than sequential programming. Parallel programming has many challenges to parallel software developers, one of them is synchronizing concurrent accesses to shared memory by multiple threads. Composing scalable parallel software using lock-based techniques is difficult and full of drawbacks i.e. full of errors if lock is fine-grained or not scalable if lock is coarse-grained and have problems like deadlocks, convoying, priority inversion and inefficient fault tolerance. When we incorporate transactions [40] into the parallel programming model it builds a new concurrency control [14] paradigm for multicore systems named Transactional Memory (TM) [4,13]. A TM system executes codes atomically which permits application threads to operate on shared memory through transactions. TM eases the development of parallel applications.Three types of Transactional Memory are, Hardware Transactional Memory (HTM) [13] , Software Transactional Memory (STM) [11] and Hybrid Transactional Memory (HyTM) [7] .Hardware Transactional Memory (HTM) systems track memory accesses in hardware. It was first coined by Herlihy and Moss in 1993.Transaction metadata is maintained in hardware. It is dependent of hardware structures such as caches. Software transactional memory (STM) systems track memory accesses in software. STM was first coined by Shavit and Touitou in 1997. It is independent of hardware structures such as caches. STM is a software system that implements nondurable (no durability) transactions along with ACI (failure atomicity, consistency, and isolation) properties for threads accessing shared data. Durability is only needed in

databases because memory transactions become obsolete as soon as the system shuts down. But it has overhead cost associated with log memory accesses, resolve access conflicts, and maintain other transactions metadata.Hybrid Transactional Memory (HyTM) was first coined by Damron in 2006.In Hybrid Transactional Memory (HyTM), software and hardware schemes can combine into a hybrid system in support of transactional programs, permitting use of low-cost HTM when it works, but reverting to STM when it doesn't.Nested transactions allow splitting of a large transaction into smaller subtransactions, each of which can perform some portion of work, and possibly fail without aborting work already done by the parent [15]. Nested transactional memory (NTM) was first coined by Moss in 2006 [8]. Nested transactional memory (NTM) provides software composition by allowing one module invoke another without either knowing whether the other uses transaction.Three types of nesting in Transactional Memory (TM) are, Linear nesting, Closed nesting and Open nesting.

2.STM DESIGN

Design differences in STM system affects a system's programming model and performance. [4,31,32]

2.1 Transaction granularity

The **basic unit of storage** over which an STM system **detects conflicts** is called **transaction granularity**.**Word-based and object-based** are two classes of transaction granularity in STMs. Detecting conflicts at **word level gives the highest accuracy** but it also has **higher communication and bookkeeping cost**.In **object-based STMs**, resources are managed at the object level granularity. This means that the STM uses an object-oriented language approach which is **more understandable, easy to implement, and less expensive.STM Haskell is Object based.**

2.2 Update Policy

A transaction **updates an object and modifies its contents**. When a transaction completes its execution successfully it updates the object's original values with updated values. Direct update and deferred update are two classes of update policy in STMs.**Direct update:** In direct update, a transaction **directly updates the value of an object**. Direct update requires a technique to record the original value of an updated object, so that it can be reversed in case if a transaction aborts.**Deferred update:** In deferred update, a **transaction updates the value of an object in a location that is private to the transaction.** The transaction ensures that it has read the updated value from this location. The value of this object is updated when a transaction commits. The transaction is updated by copying values from the private copy. In case the transaction aborts, the private copy is discarded.**STM Haskell has Deferred update strategy.**

2.3 Read Policy

There are two types of read policies **invisible and visible reads [33]** . In **invisible reads** multiple transactions can **read the same shared resources without any conflict,** so most STM systems make shared resources invisible. In invisible reads each **transaction validates its read set before commit**. **In visible reads**, **STM systems acquire locks or offer a list of readers for each read set on shared objects**. In this policy when a transaction wants to modify a shared resource, it checks if there are any readers on that shared resource. If other reader is found, then it must wait until the resources get free.**STM Haskell has visible reads.**

2.4 Acquire Policy

Accessing the shared memory exclusively is called acquiring it. There are two classes of acquiring shared memory i.e. Eager and Lazy acquire [33] .**Eager acquire:** If **a transaction acquires shared resources and modifies them as well** then, it is called eager acquire. Eager acquire transactions has advantages, because these transactions will come to know that the shared resource is being accessed by some other transaction. In case of long transactions, Eager acquire has drawbacks because the current long transaction will not permit other transactions to access the shared resources until it completes its work. **Lazy acquire:** The lazy acquire policy works well with buffered writes because **the memory is modified only at the commit time.** Using this method ensures that as all the computations are completed, all the changes can be written back to shared memory without any interruption.

2.5 Write Policy

Whenever a transaction is executed it can make some changes in shared resources. Atomically **the transaction either modifies all or nothing**. **Committing or aborting a transaction is not always successful**. The two techniques that are used to handle this problem are **Write-through or undo and Buffered write [33] .Write-through or undo:** In this technique **changes are directly written to the shared memory**. The write-through approach is **fast as changes are made directly to the shared memory**. But **aborting a transaction is expensive** as all the changes made need to be undone. **Buffered write:** In this technique, **writes are buffered, and changes are only made upon successful commit to the shared memory**. In buffered write technique, **aborting a transaction is simple** as no changes are made to the shared memory. To **commit a transaction values are copied from the buffer to the shared memory.**

2.6 Conflict Detection

A conflict occurs when **two or more transactions try to acquire and operate on the same object**. Most of the STMs use single-write multiple-read strategy. It also distinguish between RW (Read-Write) and WW (Write-Write) conflicts. The Conflict can be detected at different phases of a running transaction.**Detecting conflict before commit** is **early conflict detection** which reduces the amount of computation by the aborted transaction.**Detecting conflict on commit** is **late conflict detection** which maximizes the amount of computation discarded when a transaction aborts.**STM Haskell has lazy conflict detection.** In STM Haskell **conflicts are detected at a TVar level**, e.g, **two different transactions writing to the same TVar.**

2.7 Concurrency Control

An STM system that **executes more than one transaction concurrently requires synchronization** among the transactions for simultaneous accesses to an object.The **three events** (**conflict occurrence, detection, and resolution**) **can occur at different times** but not in different order. In general, there are two alternative techniques to concurrency control.**Pessimistic concurrency control:** In pessimistic concurrency control, **all three events happen at the same time in execution**. As a transaction tries to access a location, the system detects conflict and resolves it. In this type of concurrency control, a system claims **exclusive access to a shared resource** and prevents other transactions from accessing it.**Optimistic concurrency control:** In optimistic concurrency control, **detection and resolution of conflicts can happen after conflicts occur**. In this type of concurrency control multiple transactions can access an object concurrently. It **detects and resolves conflicts before a transaction commits.Concurrency control forward progress is guaranteed** with two techniques **blocking synchronization (lock-based) and non-blocking synchronization (wait-free, lock-free, and obstruction-free).Lock-based:** Lock based STM **does not provide any guarantee of progress** because **locks are used for mutual and exclusive access to the shared resources [23] .Wait-free:** A wait-free STM **guarantees greater progress**. In this STM, **all the threads require to workout with each other in order to make sure that every thread is making progress**.**Lock-free:** The difference between the lock-free and wait-free is that a STM which guarantees lock-free only makes sure that the **progress is made by at least one** thread in a finite number of steps keeping an entire system in context.**STM Haskell is Lock free (Optimistic). Lock free methodology based on Haskell's implementation** is composable software transactional memory by Tim.**Obstruction-free:** An obstruction-free STM guarantees **even further less progress**. Obstruction-free **ensures the progress of at least one thread in a finite number of steps in the absence of disputation.**

2.8 Memory Management

Memory management is allocation and de-allocation of memory [33] . If a transaction **allocates memory and is not successful**, it should be possible to **free the allocated memory; otherwise it can result into memory leakage**.If **a transaction de-allocates memory and is not successful or it aborts**, then this memory should still be available to **restore into previous state**

2.9 Contention Management

STM needs a contention manager [33] . C**ontention manager resolves conflicts**. There is an **attacker and a victim** during a conflict among different transactions. Upon a conflict between two transactions, the contention manager **can abort the victim, or abort the attacker, or force the attacker to retry after some period**. The contention manager can use different approaches to avoid conflicts.

Following are different techniques **for conflict resolution**:

- **Timid [34] :** Simplest of all, always **aborts a transaction whenever a conflict occurs.**
- **Polka [35] :** Backs off for different intervals equal to the difference in priorities between the transaction and its enemy.

- **Greedy [36] :** The greedy contention manager guarantees that **each transaction commits within a finite or bounded time.**
- **Serializer [37] :** The serializer works like greedy contention manager with an exception that **on aborting a transaction, each transaction gets a new priority.**

Contention manager must select one of the following **options whenever a conflict occurs**:

- **Wait:** A simple way of resolving a conflict is to **wait for some time until the resolution of an issue on its own**.
- **Abort self:** In some cases, it is not possible for a transaction to carry on its work due to the fact that another transaction may be holding the shared resource which is required by this transaction. A way to resolve this situation is that this **transaction aborts itself and restarts again.Abort other:** One of the options is to **abort the transaction which is holding the lock of required shared resources by the in-progress transaction**. A transaction having high priority aborts those transactions having low priority.

2.10 Isolation

STM Haskell has weak isolation. It does not ensure isolation at all: while one thread is accessing an IORef inside an atomic block (holding the Global Lock), there is nothing to stop another thread writing the same IORef directly (i.e. outside atomically, without holding the Global Lock), thereby destroying the isolation guarantee.

2.11 Nesting Model

Nested transactions support **software composability** [31,41].

2.11.1 Nesting by flattening

Nesting of transactions is needed to support software composability . The simple way to use **nesting is by flattening transactions into the outermost level**. For example, **DSTM and RSTM use nesting by flattening transactions.**

- **DSTM**

The **Dynamic Software Transactional Memory (DSTM**) [10] overcame the drawbacks of earlier STM systems where the transaction size and memory requirements were statically defined in advance. DSTM has C++ and Java APIs for programming **dynamic data structures**, such as lists and trees, for synchronizing applications without using locks.

- **RSTM**

The **Rochester Software Transactional Memory (RSTM**) [29] is another example of a STM that provides flattened transactions to support nesting.

2.11.2 Linear Nesting

In linear nesting model a transaction may have **only one nested transaction active at a given time**. According to J. Moss and A. Hosking [8] , there are two types of nested transactions, **closed and open**.

The linear nested transactions can be represented in a **tree structure**. Each **transaction** is a **node.** Parenthood relations are depicted by **directed edges from the child to the parent** transaction and **root is a top-level transaction**. In linear nesting, only **one of the branches of the tree may be active at a given time.**

2.11.2.1 Closed nested transactions

In closed nested transactions (**CNTs**) **commit causes union of its read- and write-sets with its parent's** [40] **.** For example, **McRT-STM, NOrec, Nested LogTM and Haskell STM have closed nested transactions**.

- **Haskell STM**

Haskell STM [1,2,3,5,6,9,12] uses **type system** for safety of the operations executing within a transaction, ruling out operations with side-effects.
The possibility of **blocking** between threads for producer-consumer patterns, is supported with the "**retry"** construct. The construct "**orElse" is used to allow a failed transaction to recover** in another way: if the first atomic action aborts, the second one is attempted. This is repeated until nested orElse blocks have all been attempted. Each **orElse block** corresponds to a new **closed nested transaction.**

- **McRT-STM**

The **Multicore Runtime STM** [24] is implemented in both **C++ and Java**. It **uses closed nested transactions.**

- **NOrec**

The **No Ownership Records STM** (NORec) [25] is developed to get the least overhead possible. It **uses closed nested transactions**.

- **Nested LogTM**

LogTM is an **HTM s**ystem. LogTM has **both open and closed nested transactions** [20,26] .

2.11.2.2 Open nested transactions

In open nested transactions **(ONTs) a committing inner transaction can release isolation immediately:** The commit is partially performed as if it was a top-level transaction [39,42] . For example, **ATOMOS, Safe open-nested transactions have open nested transactions**.

- **ATOMOS**

Atomos [21] is an **extension of Java with implicit transactions and strong atomicity**. Atomos **uses open-nested transactions with open keyword**.

2.11.3 Parallel Nesting

Parallel nested transactions can have an **arbitrary number of branches in the nesting tree with active transactions**.For example, **NeSTM, HParSTM, NePalTM, CWSTM, PNSTM and SSTM have parallel nesting.**

- **NeSTM**

The Nested STM (NeSTM) [17] is based on McRT-STM.

- **HParSTM**

The Hierarchy-based Parallel STM (HParSTM) [18] is based on Damien Imbs' STM and has opacity and progressiveness.

- **NePalTM**

The Nested Parallelism for Transactional Memory (NePalTM) [19] is based on OpenMP API and Intel's STM to combine parallel and atomic blocks.

- **CWSTM**

CWSTM [16] is based on Cilk language. Cilk is a C-based runtime system for multithreaded parallel programming and has spawn construct to create new threads with assigned tasks.

- **PNSTM**

The Parallel Nesting STM (PNSTM) [28] is based on CWSTM. PNSTM has simpler work-stealing approach.

- **SSTM**

Sibling STM (SSTM) [22] is based on .NET Common Language Runtime (CLR). It has coordinated siblings transactions and uses xfork API.

3. STM Performance Metrics

For analyzing, the transactional behaviors of TM applications, following metrics are commonly used [32] :

3.1 Commit Phase Time and Abort Phase Time
3.2 Commit Reads and Abort Reads
3.3 Commit Writes and Abort Writes
3.4 Execution time
3.5 Aborts per Commit
3.6 Transaction Retry Rate

3.1 Execution time

The execution time displays the transactional effectiveness of application scale with respect to the increasing number of threads.

3.2 Aborts per Commit

This is ratio of aborted transactions to committed transactions. This metric indicates the efficiency with which computing resources have been utilized as amount of work committed is assessed by the workload inputs

3.3 Transaction Retry Rate

This metric is used to exploit the inherent concurrency of the underlying STM implementation.A transaction self aborts explicitly and in the process, it also reschedules itself after detecting its precondition for the operation, which may not hold any longer.

3.4 Work and commit times

Time used to **execute the transaction is work time** and **time to commit it is commit time.T**he **commit phase overhead** is obtained by dividing the **commit phase time** by the **total time**.

3.5 Readset and writeset, reads and writes

Readset and writeset are expressing the **size of the readset and the writeset respectively**. The **readset** consists of **transactional variables that are strictly for only read.** All of the programs can be categorized using the **number of times they perform a transactional read (marked "reads") on committed transactions.** W**riteset consists of the number of writes performed** because the transactions only persist their updates by committing.**Reads/writes ratio** of the program is used to determine whether the program is **read-dominated or write-dominated.**

4. Advantages and Disadvantages of STM

In addition to their performance benefits, STM greatly simplifies conceptual understanding of multithreaded programs and helps make programs more maintainable by working in harmony with existing high-level abstractions such as objects and modules. Lock-based programming has a number of well-known problems .In contrast, the concept of a memory transaction is much simpler, because each transaction can be viewed in isolation as a single-threaded computation. Deadlock and livelock are either prevented entirely or handled by an external transaction manager; the programmer need hardly worry about it. Priority inversion can still be an issue, but high-priority transactions can abort conflicting lower priority transactions that have not already committed.On the other hand, the need to abort failed transactions also places limitations on the behaviour of transactions: they cannot perform any operation that cannot be undone, including most I/O. Such limitations are typically overcome in practice by creating buffers that queue up the irreversible operations and perform them at a later time outside of any transaction. In Haskell, this limitation is enforced at compile time by the type system.

5. Research Issues for STM

The design of a TM system that supports nested parallel transactions is challenging [32,38].

5.1 Transformation of transactional code

5.2 Conflict detection scheme

5.3 Memory overhead

5.4 Single level of parallelism

5.1 Transformation of transactional code

It is also a challenge in STM. In databases **non transactional code runs inherently as a transaction**. In STM this is done by either separating transactional and non transactional code or dynamically categorizing their access to shared objects.

5.2 Conflict detection scheme

It must be able to correctly track dependencies in **a hierarchical manner** instead of a flat way. Nested parallel transactions may conflict and restart without necessarily aborting their parent transaction.

5.3 **Memory overhead** necessary for tracking the state of nested transactions **should** be **small.**

5.4 Single level of parallelism

Some applications may not use nested parallelism, we must ensure that its **overhead is reasonable** when only a **single level of parallelism is used.**

6 CONCLUSION and FUTURE RESEARCH DIRECTIONS

Software Transactional memory provides a flexible and easy mechanism for parallel programming in multicore processors. It is a promising alternative to lock-based mutual exclusion strategies .This study presented design parameters for Software transactional memory systems. The environment where the STM is to be deployed and the constraints imposed by it have to be analyzed critically during the STM design.This paper presents performance comparisons based on nested model which is a design parameter for Software transactional memory. This paper has begun the process of comparing STM implementations based on performance. This is because performance is important and in fact, it is the motivation for creating STMs in the first place. The evaluation of performance is an important task. We hope that this work will set the stage for more detailed performance comparisons to come. Phase.From the analysis above, we get a research trend that the future STM systems will be more likely on how to trade off between performance and semantics correctness

REFERENCES

[1] Ghosh A., Chaki R. (2016) Implementing Software Transactional Memory Using STM Haskell. Advanced Computing and Systems for Security. Advances in Intelligent Systems and Computing, vol 396. Springer, New Delhi.

[2] Matthew Le, Ryan Yates, and Matthew Fluet. 2016.Revisiting software transactional memory in Haskell in Proceedings of the 9th International Symposium on Haskell .ACM, New York, NY, USA,

[3] Du Bois A.R. (2011) An Implementation of Composable Memory Transactions in Haskell. In: Apel S., Jackson E.(eds) Software Composition. SC 2011. Lecture Notes in Computer Science, vol 6708. Springer, Berlin, Heidelberg.

[4] T. Harris, J. R. Larus, and R. Rajwar. Transactional Memory, 2nd edition. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2010.

[5] Discolo A., Harris T., Marlow S., Jones S.P., Singh S. Lock Free Data Structures Using STM in Haskell. In: Hagiya M., Wadler P. (eds) Functional and Logic Programming. FLOPS 2006. Lecture Notes in Computer Science, vol 3945. Springer, Berlin, Heidelberg

[6] M. Herlihy, V. Luchangco and M. Moir, "A flexible framework for implementing software transactional memory, "Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Languages, Systems, and Applications, pp. 253-262, 2006.

[7] P. Damron et al., ''Hybrid Transactional Memory,'' Proc. 12th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS 06), ACM Press, 2006, pp. 336-346.

[8] J.E.B. Moss, A.L. Hosking, Nested transactional memory: Model and preliminary architecture sketches, in: the OOPSLA 2005 Workshop on Synchronization and Concurrency in Object-Oriented Languages, SCOOL (no formal proceedings), October 2005.

[9] T. Harris, S. Marlow, S. Peyton Jones, and M.Herlihy. Composable memory transactions. In Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '05, pages 48– 60, Chicago, IL, USA, 2005. ACM.

[10] M. Herlihy, V. Luchangco, M. Moir and W. N. Scherer III, (DSTM)"Software transactional memory for dynamic-sized data structures," Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing, pp. 92-101, 2003

[11] Nir Shavit and Dan Touitou. Software transactional memory. Distributed Computing. Volume 10, Number 2. February 1997.

[12] S. Peyton Jones, A. Gordon, and S. Finne. Concurrent Haskell. In 23rd ACM Symposium on Principles of Programming Languages (POPL'96), pp. 295–308.

[13] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: architectural support for lock-free data structures. Proceedings of the 20th annual international symposium on Computer architecture (ISCA '93). Volume 21, Issue 2, May 1993.

[14] P.A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," ACM Computing Surveys, Vol. 13(2), June 1981, pp. 186 – 221.

[15] J.E.B. Moss, Nested Transactions: An Approach to Reliable Distributed Computing, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, Apr. 1981; also published as MIT Laboratory for Computer Science Technical Report 260.

[16] K. Agrawal, J. T. Fineman and J. Sukha. (CWSTM)Nested parallelism in transactional memory. In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '08), 163{174, 2008.

[17] W. Baek and C. Kozyrakis. NesTM: Implementing and Evaluating Nested Parallelism in Software Transactional Memory. In Proceedings of the 9th International Conference on Parallel Architectures and Compilation Techniques (PACT '10), 253{262, 2010.

[18] R. Kumar and K. Vidyasankar. HParSTM: A Hierarchy-based STM Protocol for Supporting Nested Parallelism. In the 6th ACM SIGPLAN Workshop on Transactional Computing (TRANSACT '11),2011.

[19] H. Volos, A. Welc, A.-R. Adl-Tabatabai, T. Shpeisman, X. Tian and R. Narayanaswamy. NePaLTM: Design and Implementation of Nested Parallelism for Transactional Memory Systems.In Proceedings of the 23rd European Conference on Object-Oriented Programming (ECOOP '09),123{14, 2009.

[20] M. J. Moravan, J. Bobba, K. E. Moore, L. Yen, M. D. Hill, B. Liblit, M. M. Swift and D. A.Wood. Supporting Nested Transactional Memory in LogTM. 12th International Conference on Architectural Support for Programming Languages and Operating Systems In SIGPLAN Notices (Proceedings of the 2006 ASPLOS Conference), 41(11):359{370, 2006

[21] B. Carlstrom, A. Mcdonald, H. Cha_, J. Chung, C. Minh, C. Kozyrakis and K. Olukotun. The ATOMOS Transactional Programming Language. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'06), 1{13, 2006.

[22] H. Ramadan and E. Witchel. The xfork (SSTM)in the road to coordinated sibling transactions. In the 4th ACM SIGPLAN Workshop on Transactional Computing (TRANSACT '09), 2009.

[23] D. Imbs and M. Raynal. A Lock-Based STM Protocol That Satisfies Opacity and Progressiveness.In Proceedings of the 12th International Conference on Principles of Distributed Systems (OPODIS'08), 226{245, 2008.

[24] B. Saha, A.-R. Adl-Tabatabai, R.L. Hudson, C. Cao Minh and B. Hertzberg. McRT-STM: a high performance Software Transactional Memory system for a multi-core runtime. In Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'06), 187{197, 2006.

[25] L. Dalessandro, M. Spear and M. Scott. NOrec: Streamlining STM by abolishing ownership records. In Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '10), 67{78, 2010

[26] K. Moore, J. Bobba, M. Moravan, M. Hill and D. Wood. LogTM: log-based transactional memory.In Proceedings of the 12th High-Performance Computer Architecture International Symposium (HPCA '06), 254{265, 2006

[27] R. Blumofe, C. Joerg, B. Kuszmaul, C. Leiserson, K. Randall and Y. Zhou. Cilk: An efficient multithreaded runtime system. In Journal of Parallel and Distributed Computing, 37(1), 55{69,August 1996.

[28] J. Barreto, A. Dragojevic, P. Ferreira, R. Guerraoui and M. Kapalka.(PNSTM) Leveraging parallel nesting in transactional memory. In Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '10), 91{100, 2010.

[29] V. Marathe, M. Spear, C. Heriot, A. Acharya, D. Eisenstat, W. Scherer III and M. Scott. (RSTM)Lowering the overhead of Software Transactional Memory. In the 1st ACM SIGPLAN Workshop on Transactional Computing (TRANSACT '06), 2006.

[30] Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco (1992) .

[31] Diegues, N., Cachopo, J.: Review of nesting in transactional memory. Tech. rep., Technical Report RT/1/2012, Instituto Superior Técnico/INESC-ID (2012).

[32] Gulfam Abbas, Naveed Asif, " Performance Tradeoffs in Software Transactional Memory", Master Thesis Computer Science, School of Computing Blekinge Institute of Technology Sweden ,No:MCS-2010-28, May 2010.

[33] S. Classen, "LibSTM: A fast and flexible STM Library," Master's Thesis, Laboratory for Software Technology, Swiss Federal Institute of Technology, ETH Zurich, Feb, 2008.

[34] W. N. S. III and M. L. Scott, (TIMID)"Contention Management in Dynamic Software Transactional Memory," in *Proceedings of the ACM PODC Workshop on Concurrency and Synchronization in Java Programs*, St. John's, NL, Canada, July, 2004.

[35] I. William N. Scherer and M. L. Scott,(POLKA) "Advanced contention management for dynamic software transactional memory," in *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, Las Vegas, NV, USA, 2005, pp. 240-248.

[36] R. Guerraoui, *et al.*,(GREEDY) "Toward a theory of transactional contention managers," in *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, Las Vegas, NV, USA, 2005, pp. 258-264.

[37] M. L. Scott. (SERIALIZER) Applications Included with RSTM WebPage:

http://www.cs.rochester.edu/research/synchronization/rstm/applications.shtml.

[38] W. Baek, N. Bronson, C. Kozyrakis, and K. Olukotun. Implementing and evaluating nested parallel transactions in software transactional memory. In Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '10, pages 253–262, Thira, Santorini, Greece, 2010. ACM.

[39] Alexandru Turcu and Binoy Ravindran. On open nesting in distributed transactional memory. In SYSTOR '12,2012

[40] Alexandru Turcu, Binoy Ravindran, and Mohamed M. Saad. On closed nesting in distributed transactional memory. In TRANSACT '12,2011

[41] T. Harris and S. Stipic. Abstract nested transactions. In Second ACM SIGPLANWorkshop on Transactional Computing, 2007.

[42] Y. Ni, V. S. Menon, A.-R. Adl-Tabatabai, A. L. Hosking, R. L. Hudson, J. E. B. Moss, B. Saha, and T. Shpeisman. Open nesting in software transactional memory. In PPoPP '07: Proceedings of the 12th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, pages 68–78, New York, NY, USA, 2007. ACM Press.