Performance Analysis of Software Transaction Memory Techniques

*Varun Jasuja, **Amit Sandhu, ***Kiran
*Assistant Professor, Deptt. of Computer Science & Engineering
Punjab College Of Engg And Tech.,Lalru Mandi(Punjab), varunjasuja@yahoo.co.in
**Assistant Professor, Deptt. of Computer Science & Engineering
Arni School of Technology, Arni University, Kathgarh(HP), sandhuamit80@yahoo.com
***Student at IGNOU, New Delhi

Abstract: Software Transactional Memory is generic non blocking synchronization construct that enables automatic conversion of sequential objects into correct concurrent objects. Because it is nonblocking, STM avoids traditional performance and correctness problems due to thread failures, preemption, page faults and priority inversion. The advent of multicore processors has put the performance of traditional parallel programming techniques in question. The traditional lock-based parallel programming techniques are error prone and suffer from various problems such as deadlock, live-locks, priority inversions etc. In This paper we paper we present some of the Software Transactional Memory technique.
Keywords: Multiprocessor, Concurrency, Synchron- - ization, Transactional Memory

I INTRODUCTION

Computer system with a single processor has almost reached their limits. In the last decade, we saw incredible improvement in CPU design and performance. In the beginning of nineties, there were CPUs with on million transistors(e.g. Intel 336) and in 2003.there were CPUs with one billion transistors(e.g. Intel Itanium),which is an improvement of 1000x.CPUs with mode transistors have mode hardware units and much mode efficient, but now they are running into problem which will prevent their future improvement. for example, more transistors spend much more electricity and they heat the die more. Today's CPUs dissipate the same amount of energy on a square millimeter as the burner while cooking coffee. Other problem is that if we increase the clock frequency two time, CPU speed will increase 1.3 times and dissipation will be increased 4 times. The locking mechanism and mutual exclusion have been used to achieve this synchronization. However, it may create performance bottle-necks, and it is more time consuming and vulnerable to the errors. Conventional parallel programming synchronization mechanism , such as locks, monitors and semaphores are exceptionally difficult to program correctly Transactional Memory allows data sharing without using locking mechanisms. The Transactional Memory can be implemented in software as well as in hardware. It commits the data in atomic code sequences called Transactions. Before committing a transaction it checks whether the data

which it read was not outdated or changed by another transaction. When there is a read/ write conflict, it aborts the transaction and rolls back and process the transaction again until it has no conflict. So in this procedure, Transactional Memory maintains a log for each transaction so that it would go back to its previous state. Software Transactional Memory has a flexible framework for executing parallel operations with contention manager for resolving the conflicts and load balancing. Transactions have been used a lot in databases since long ago and it do not suffer from resource starvation and deadlocks.

II SOFTWARE TRANSACTIONAL MEMORY CONCEPTS

A memory transaction is basically a finite sequence of instructions, preserving the serializability and atomicity properties.

Transactions in Database Programming

Transactions have long been a part of database programming while their importance is now realized in parallel programming. Database Systems allow multiple queries to run in parallel and it maintains concurrency with consistency. In other words, if a concurrent transaction is left in an illegal state, then it is aborted and rolled back.

Database Transaction Properties

A transaction is a set of instructions which are basically one unit. It has proper start and end with consistent results. A particular database transaction has four basic properties: Atomicity, Consistency, Isolation and Durability also known as ACID, to ensure that transactions take place with correctness.

1. Atomicity: Atomicity means that a transactions either successfully completes or it fails and roll back. On successful completion it commits its results and on failure it aborts.
2. Consistency: Consistency means that modification will not alter the structure being modified in an inconsistent way;that is , all modifications preserve the underlying structure of object being modified.
3. Isolation: Isolation is a property which makes sure that each transaction can execute in parallel independently and its internal execution and data should be isolated and hidden from other transactions and failure of one transaction may not affect the result of other transactions.

4. Durability: Durability implies once committed the changes persist , or that once aborted no residue of partial transitional will remain.

III HARDWARE TRANSACTIONAL MEMORY CONCEPTS

The operation sets a memory location to a new value, but only if currently contains a specified executed value.The basic algorithm CompareAndSwap implements a shown in figure.

ComapreAndSwap (a: WordAddress, old: Word, new: Word) :Bool

1 if *a=old

2 then *a← new

3 return True

4 else return False

CompareAndSwap

ComapreANDSwap is executed in one machine instruction,operating on the specified address automically.This seems to be simple,and not very useful instruction and is expressive enough to create arbitrarily complicated non-blocking algorithms.Also,

The other important primitive commonly available on modern processors is part of instructions LoadLinked and StoreConditional .LoadLinked loads a value from memory and “locks” it,such that a following StoreConditional instructions to the same memory location will only succeed if memory location read in LoadLinked step has not been modified by some other memory operation.

LoadLinked (r:Register ,a: WordAddress)

1 r ← *a

2 Linked(a) ← True

StoreConditional (r:register ,a: WordAddress)

1 if Linked(a) =True

2 then *a ← r

3 r ← 1

4 else r ← 0

LoadLinked and StoreConditional

Conventional Methods

Achieving Synchronization in Parallel Applications

Parallel applications share data and the traditional mechanism to achieve synchronization has been a locking mechanism. Locking uses mutexes, semaphores etc. to ensure mutual exclusion in resource sharing. Figure shows a code in which the variable counter is accessed exclusively using a locking mechanism.

```
Lock ()
{
// shared variable counter
counter++;
}
Unlock ()
```

Fig. A Shared variable with Lock.

Locking ensures mutual access to the shared data but it creates a bottleneck for other threads or parallel processes. Other processes have to wait until the thread which is holding the lock completes its execution. Blocking a process can lead to the following problems .

Priority Inversion: Priority inversion takes place when a lower priority process is holding a resource which is required by a higher priority process, which makes the higher priority process wait until resource is released.

Convoying: Convoying takes place when a process holding a lock is re-scheduled due to the different reasons, such as, if the process has consumed its processing quantum of time and yet not completed its execution, may be due to the page fault or due to some other interference. Meanwhile other threads waiting in queue to acquire the lock will not be able to progress ahead until this thread release the lock. Even if the lock is released, it will take some time to re-set the queue, which as a result will slow down the processing.

Deadlock and Livelock: A deadlock is a situation where one or more processes are waiting for each other to release a resource and this situation lead to a circular chain of wait with no progress taking place on part of each process.

On the other hand, livelock does not wait for anything but keeps on processing based on the erroneous input. A good example of the livelock can be endless loop. It is analogous to the deadlock that no real progress is made ahead yet differs in a sense that no process is blocked or waiting for any resource.

Wait-Freedom:This property of the non-blocking system allows each process to progress without taking the contention into the context. Wait-freedom infact ensures that there would not be any starvation. However practically its not possible to develop efficient wait-free algorithms in parallel applications as the memory cost increases linearly with the number of processes. Therefore not much attention has been paid in this regard.

Lock-Freedom: The Lock-freedom ensures that multiple processes run at the same time but only one process goes ahead and completes its execution within finite number of execution time. The rest of the processes have to wait. The Lock-freedom ensures deadlock prevention but suffers from starvation. In lock-freedom, every process try to complete its execution but when it identify that original values have been changed by another process then it rolls-back and starts its processing again based on new values.

Obstruction-Freedom:An algorithm is obstruction-free if it allows completing a process only if it is not obstructed by another process. This is a very weak property of a non-blocking algorithm as it is hardly possible that another process will not contend the currently executing process. Furthermore, Obstruction-free algorithm introduces the problem of livelock.Therefore to avoid livelock and deadlock, roll-back is used. Moreover, a contention manager can be used to decide which processes have higher priority and based on the priority level higher priority process is allowed to execute while lower priority processes are obstructed.

Software Transactional Memory Systems	**Synchronization**
STM (Shavit, Touitou)	Lock-free
WSTM (Fraser, Harris)	Lock-free
OSTM (Fraser)	Lock-free
DSTM (Herlihy et al)	Obstruction-free
RSTM (Marathe)	Obstruction-free
Time based STM (Riegel)	Obstruction-free
DSTM 2 (Herlihy OOPSLA)	Obstruction-free
McRT-STM (Saha et al)	Lock-based
TL2 (Dave Dice, Ori Shalev, Nir Shavit)	Lock-based
HybridTM (Kumar)	Hybrid TM
PhTM (Lev)	Hybrid TM
SigTM (Chi Cao Minh)	Hybrid TM

IV SOFTWARE TRANSACTIONAL MEMORY (STM)

The STM system proposed by Shavit and Touitou identifies and tries to get access of all the memory locations which it would need for a particular transaction. The basic unit of memory, on which this system is based on, is word. In other words, the transaction granularity is at word level. The basic design features of Shavit and Touitou's STM are shown in table When a transaction holds the control of memory word, it becomes the owner of that memory word.

STM	
Synchronization	Non-blocking (Lock-freedom)
Concurrency Control	Pessimistic
Conflict Detection level (Granularity)	Word
Update Strategy	Direct Update
Conflict Detection	Early
Conflict Management Strategy	Helping
Nested Transaction Type	N/A

The Basic Design features of STM.

The ownership record either has a valid address of the owner or it has a Null value which indicates that no transaction owns the data. Although each transaction can access shared data, but one memory block can be owned by only one transaction at a time If a transaction fails to acquire ownership of a memory object then it aborts and releases the memory locations which it already has acquired. If a transaction manages to take all the desired memory locations then it completes its execution and updates the results without the risk of rollback. This implies that system uses the Direct Update approach.

Design Limitations

One of the major draw back which Shavit and Touitou's system have is its helping mechanism for conflict resolution. This conflict resolution technique is based on the concept that if a transaction can not proceed ahead due to some conflict then it should help other transaction in their completion. In this case two threads are executing the same transaction. This means that if transaction X found that it is conflicting with transaction Y then transaction X makes update on behalf of transaction Y.

In recursive helping a transaction being helped may be helping another transaction. Moreover, consistent helping can deteriorate the performance by unnecessary

conflicts.However, in Shavit's STM, helping is restricted to a specific level only; even then it leads to a great level of complexity, as it exposes data to one or more threads. One of the limitations this system, includes the advance declaration of the memory locations that a transaction will acquire. This restricts the transaction from acquiring a memory location dynamically.However, recent versions of the STM can acquire the memory locations dynamically e.g. Hash table based STM developed by Harris and Fraser .

V SIGNATURE ACCELERATED TRANSACTIONAL MEMORY (SIGTM)

Cao Minh et al. presented Signature Accelerated Transactional Memory, SigTM in 2007. The SigTM is a Hybrid Transactional Memory system. The SigTM uses hardware signatures to keep track of read-set and write-set and help in conflict detection. The signatures are basically data structures that can store the data access information of the transactions. A SigTM signature is shown in table.However, data versioning information is stored in software part of the system. The signature data-structure in SigTM does not require any modification to the hardware caches which reduce the hardware cost. The SigTM supports nested transactions and strong isolation [48]. It utilizes the strong isolation and performance characteristics of HTM with low cost and flexibility of the STM.

SigTM	
Synchronization	Non-blocking
Concurrency Control	Optimistic
Conflict Detection level (Granularity)	Cache line/ Word
Update Strategy	Deferred Update
Conflict Detection Strategy	Late
Conflict Management	Aborting
Nested Transaction Type	Supported
Isolation	Strong

The Basic Design features of SigTM.

Design Details

The SigTM uses hardware signatures for conflict detection and strong isolation by looking up coherence requests. While other functionality, i.e., transactional versioning, commit and rollback are dealt in software part. The SigTM uses TL2 as STM part of Hybrid TM system.The TL2 is locked based STM using optimistic concurrency control with granularity at word level and works fine for range of contention scenarios. The TL2 uses global version clock to generate time stamps for the data versioning. The STM transactions are slower than the HTM transactions due to the versioning and conflict detection overhead. Every word read, must be re-validated for its time stamp while committing a transaction. However, the SigTM eliminates the global version clock and locking mechanism in the base STM. Moreover, it eliminates the need for software read-set. However software write-set is still required to store the transactional updates, until the transaction commits. The SigTM does not require to switch between STM and HTM

Instruction	Instruction Description
rsSigReset wsSigReset	Reset all bits in read-set or write-set signature
rsSigInsert r1 wsSigInsert r1	Insert the address in register r1 in the read-set or write-set signature
rsSigMember r1,r2 wsSigMember r1,r2	Set register r2 to 1 if the address in register r1 hits in the read-set or write-set signature
rsSigSave r1 wsSigSave r1	Save a portion of the read-set or write-set signature into register r1
rsSigRestore r1 wsSigRestore r1	Restore a portion of the read-set or write-set signature from register r1
fetchEx r1	Pre-fetch address in register r1 in exclusive state; if address in cache, upgrade to exclusive state if needed

The Signature in SigTM

The contention management in SigTM is analogous to the base STM, i.e., TL2. A conflicting transaction is backed-off and retried after a delay. And when a transaction is repeatedly backed-off then it is eventually aborted. The SigTM implements lazy data versioning and

transactional updates are buffered until the transaction commit. Moreover, the operating system may also suspend a SigTM transaction.

Eagar data Versioning

Because of its TL2 heritage, SigTm implements lazy data versioning. Trasactional are buffered in the write-set until transaction commit. Alternatively we could start with the eagar versioning. Writes within transaction update memory location in place, logging the original value in an undo log in case of abort. Since SigTM implements data versioning in software, no hardware changes to support eagar-based scheme.

Design Limitations

The SigTM used hardware signatures to track the read-set and write-set while reducing the overhead of Software transactions. Moreover, signature data structure makes the implementation of nested transactions easy. One of the major performance challenges faced by SigTM is the in-exact nature of the signatures. Therefore, it is hard to find that what operations are taking place in read-sets and write-sets. This in-exact nature of signatures, lead to false conflict detection. Using,such off-the-shelf hardware for data access tracking, in place of signatures can over-come the deficiencies introduced by signatures.

The Algorthm For Basic Function in the Signature Software Trasactional Memory Technique is given presented below

1. SIGTMTXSTART
   checkpoint()
   enableRSlookup(exclusive)
2. SIGTMWRITERBARRIER(addr,data)
   wsSigInsert(addr)
   writeSet.insert(addr,data)

3. SIGTMREADBARRIER(addr)
   If wsSigMember(addr) and writeset.member(addr)
   then
   Return writeSet.lookup(addr)
   [End of if structure]
   rsSigInsert(addr)

Return Memory[addr]

4. SIGTMTXCOMMIT

enableWSlookup(exclsive,shared)

Repeat step 5 while addr!=null in writeSet

5. fetchEx(addr)

enableWSnack(exclusive,shared)

rsSigReset()

disableRSlookup()

[End of Step 4 loop]

6. Repeat step 7 while addr!=null in writeSet

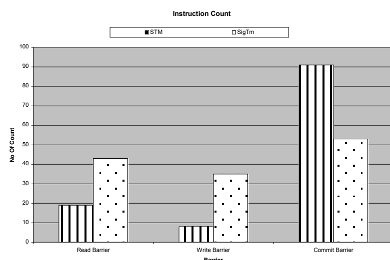
7. Memory[addr]=writeSetLookup(addr)

wsSigReset()

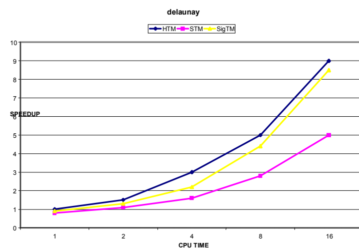
disableWSnack()

[End of Step 6 loop]

8. Exit



Graph for the performance showing no. of barrier used by execution of same instruction set by STM and SigTM



Graph for showing cputime taken by STM, HTM,SigTM in delaunay algorithm

V Conclusion

Although a substantial amount of research has been carried-out in last one and half decade regarding Transactional Memory systems, there are still many overheads and problems that need to be further researched and investigated. This survey identifies the following areas Achieving Strong Isolation, Nested Transactions, Integration Overheads for future research with regards to the implementation of Software Transactional Memory systems.

Reference

[1] Abadi, M., Harris, T., and Mehrara, M. Transactional memory with strong atomicity using off-the-shelf memory protection hardware. In Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Raleigh, NC, USA. PPoPP '09. ACM, New York, NY, February 2009. DOI= http://doi.acm.org/10.1145/1504176.1504203.

[2] Ennals, R. Software transactional memory should not be obstruction-free. Technical Report Nr. IRC-TR-06-052. Intel Research Cambridge Tech Report, 2006.

[3] Marathe, V. J. and Scott, M. L. A Qualitative Survey of Modern Software Transactional Memory Systems. Technical Report Nr. TR 839, University of Rochester Computer Science Dept., 2004.

[4] Larus, J. R. and Rajwar, R. Transactional Memory.

[5] Minh, C. C., Trautmann, M., Chung, J., McDonald, A., Bronson, N., Casper, J., Kozyrakis, C., and Olukotun, K.An effective hybrid transactional memory system with strong isolation guarantees. SIGARCH Comput. Archit. News, June 2007. DOI= http://doi.acm.org/10.1145/1273440.1250673