Performance Comparison of Various STM Concurrency Control Protocols Using Synchrobench

Ajay Singh    Dr. Sathya Peri    Anila Kumari    Monika G.

February 24, 2017
Sequential computing -
Sequential computing - manycore/multicore computing
Sequential computing - manycore/multicore computing - Moore’s Law.

Challenge: harnessing full potential of multiple cores
Sequential computing - manycore/multicore computing - Moore’s Law.

Challenge: harnessing full potential of multiple cores

Solution:
- parallel programming paradigms: shared memory programming
- multiple threads, mutual exclusion, concurrency control
- semaphores, monitors
- fine grained locks, coarse grained locks
The Big But

Composability, deadlocks, priority inversion, implementation woes, debugging woes, managing code, scalability, cannot focus on parallel logic.
The Big But

Composability, deadlocks, priority inversion, implementation woes, debugging woes, managing code, scalability, cannot focus on parallel logic.

God Help! Make parallel programming easier.
Software Transactional Memory: The Saviour

Don’t Worry!!
Software Transactional Memory: The Saviour

Don't Worry!!

Easier to code, More focus on logic, composable
Outline of the Presentation

1. Introduction to STM
   - Prelim Definitions
   - STM Model

2. STM Concurrency Control Protocols
   - Basic Timestamp Ordering (BTO) Algorithm
   - Serialization Graph Testing (SGT) Algorithm
   - Multiversion Time Stamp Ordering Protocol

3. Test Application - Set
   - Overview
   - Pseudocode
   - Performance Evaluation Against Synchrobench

4. Conclusion

5. Summary
Abstract parallel programming model to lift the burden of synchronization from programmer.

- Uses concept of transactions borrowed from database -
  - begin()
  - read()
  - write()
  - commit()
Software Transactional Memory

- Abstract parallel programming model to lift the burden of synchronization from programmer.
- Uses concept of transactions borrowed from database - Atomicity, Consistency, Isolation property
- Wrap your shared code or critical section into **transactions** using the methods exposed by STM.
  - t_begin(),
  - t_read(), t_write()
  - t_commit()
## Simple Programming Example

### Coarse Grained lock

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
<th>Transaction Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock(B1)</td>
<td>1</td>
<td>t_begin()</td>
</tr>
<tr>
<td>lock(B2)</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>read(B1)</td>
<td>3</td>
<td>t_read(B1)</td>
</tr>
<tr>
<td>read(B2)</td>
<td>4</td>
<td>t_read(B2)</td>
</tr>
<tr>
<td>B1 = B1 + B2</td>
<td>5</td>
<td>B1 = B1 + B2</td>
</tr>
<tr>
<td>write(B1)</td>
<td>6</td>
<td>t_write(B1)</td>
</tr>
<tr>
<td>unlock(B1)</td>
<td>7</td>
<td>tryC()</td>
</tr>
<tr>
<td>unlock(B2)</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Execution under STMs
Outline

1 Introduction to STM
   • Prelim Definitions
   • STM Model

2 STM Concurrency Control Protocols
   • Basic Timestamp Ordering (BTO) Algorithm
   • Serialization Graph Testing (SGT) Algorithm
   • Multiversion Time Stamp Ordering Protocol

3 Test Application - Set
   • Overview
   • Pseudocode
   • Performance Evaluation Against Synchrobench

4 Conclusion

5 Summary
Definitions

Transaction
A transaction is a sequence of instructions that needs to be executed together in memory.

Schedule
A set of interleaved transactions.

Correctness
Holy grail is that if any schedule is equivalent to some sequential execution your schedule is correct. -
Definitions

Transaction
A transaction is a sequence of instructions that needs to be executed together in memory.

Schedule
A set of interleaved transactions.

Correctness
Holy grail is that if any schedule is equivalent to some sequential execution your schedule is correct. -opacity, linearizability
1 Introduction to STM
   • Prelim Definitions
   • STM Model

2 STM Concurrency Control Protocols
   • Basic Timestamp Ordering (BTO) Algorithm
   • Serialization Graph Testing (SGT) Algorithm
   • Multiversion Time Stamp Ordering Protocol

3 Test Application - Set
   • Overview
   • Pseudocode
   • Performance Evaluation Against Synchrobench

4 Conclusion

5 Summary
STM Model

STM vs Synchrobench

IIT Hyderabad February 24, 2017 11/49
STM Model

Ensure No conflicting operations occur concurrently.

Generates local data

Reads shared memory

Writes to shared memory

Sequential code

- \texttt{t\_begin}
- \texttt{t\_read}
- \texttt{t\_write}
- \texttt{t\_commit}
STM Model

Feed transactions

- t_begin
- t_read
- t_write
- t-commit

Generates local data
Reads shared memory

Writes to shared memory

validate
commit

Ensure No conflicting operations occur concurrently.

Sequential code
Outline

1 Introduction to STM
   • Prelim Definitions
   • STM Model

2 STM Concurrency Control Protocols
   • Basic Timestamp Ordering (BTO) Algorithm
   • Serialization Graph Testing (SGT) Algorithm
   • Multiversion Time Stamp Ordering Protocol

3 Test Application - Set
   • Overview
   • Pseudocode
   • Performance Evaluation Against Synchrobench

4 Conclusion

5 Summary
Basic Timestamp Ordering (BTO) Algorithm

**Timestamp Ordering (TO) Rule**

- Each transaction \( t_i \) is assigned a unique timestamp \( ts(t_i) \)
- If \( p_i(x) \) and \( q_j(x) \) are in conflict, then the following must hold:
  \[ p_i(x) <_s q_j(x) \text{ iff } ts(t_i) < ts(t_j) \text{ for every schedule } s. \]

**Data Structures Maintained:**

For each data item \( x \), maintain:

- \( \text{max-}r(x) = \max\{ ts(t_j) \mid r_j(x) \text{ has been scheduled} \} \)
- \( \text{max-}w(x) = \max\{ ts(t_j) \mid w_j(x) \text{ has been scheduled} \} \)
BTO Algorithm

- **Read rule** - When a read operation $r_i(x)$ arrives,
  - Compare $ts(t_i)$ with $max-w(x)$.
  - If $ts(t_i) < max-w(x)$ then transaction $t_i$ is aborted.

- **Write rule** - When a write operation $w_i(x)$ arrives,
  - It is added to the local buffer of the transaction.
  - Validation is done during the commit phase (deferred write approach).

- **Commit rule** - When a commit request arrives,
  For each data object $x$ in the write buffer,
  - If $ts(t_i) < max-w(x)$ or $ts(t_i) < max-r(x)$ then the transaction is aborted.
  - Otherwise it is allowed to commit.
BTO Example:
\[ s = r_1(x) \ w_2(x) \ r_3(y) \ w_2(y) \ c_2 \ w_3(z) \ c_3 \ r_1(z) \ c_1 \]
BTO Example:

\[ s = r_1(x) \, w_2(x) \, r_3(y) \, w_2(y) \, c_2 \, w_3(z) \, c_3 \, r_1(z) \, c_1 \]

\[ s = r_1(x) \, w_2(x) \]
BTO Example:

\[ s = r_1(x) \, w_2(x) \, r_3(y) \, w_2(y) \, c_2 \, w_3(z) \, c_3 \, r_1(z) \, c_1 \]
BTO Example:

\[ s = r_1(x) \ w_2(x) \ r_3(y) \ w_2(y) \ c_2 \ w_3(z) \ c_3 \ r_1(z) \ c_1 \]
BTO Example:
\[ s = r_1(x) \ w_2(x) \ r_3(y) \ w_2(y) \ c_2 \ w_3(z) \ c_3 \ r_1(z) \ c_1 \]

\[ s = r_1(x) \ w_2(x) \ r_3(y) \ w_2(y) \ c_2 \]
BTO Example:

\[ s = r_1(x) \, w_2(x) \, r_3(y) \, w_2(y) \, c_2 \, w_3(z) \, c_3 \, r_1(z) \, c_1 \]

\[ s = r_1(x) \, w_2(x) \, r_3(y) \, w_2(y) \, a_2 \]

\[
\begin{align*}
\text{t}_1 & : r_1(x) \\
\text{t}_2 & : w_2(x) \quad w_2(y) \quad \text{abort} \\
\text{t}_3 & : r_3(y)
\end{align*}
\]
BTO Example:

\[ s = r_1(x) \, w_2(x) \, r_3(y) \, w_2(y) \, c_2 \, w_3(z) \, c_3 \, r_1(z) \, c_1 \]
BTO Example:

\[ s = r_1(x) \ \textcolor{red}{w_2(x)} \ r_3(y) \ \textcolor{red}{w_2(y)} \ c_2 \ \textcolor{red}{w_3(z)} \ c_3 \ r_1(z) \ c_1 \]

\[ s = r_1(x) \ w_2(x) \ r_3(y) \ w_2(y) \ a_2 \ w_3(z) \ c_3 \]

Diagram:
- \( t_1 \): \( r_1(x) \)
- \( t_2 \): \( w_2(x) \) to \( w_2(y) \) with \( \text{abort} \)
- \( t_3 \): \( r_3(y) \) to \( w_3(z) \) with \( \text{try\_commit} \)
BTO Example:

\[ s = r_1(x) \ w_2(x) \ r_3(y) \ w_2(y) \ a_2 \ w_3(z) \ c_3 \]

\[ s = r_1(x) \ w_2(x) \ r_3(y) \ w_2(y) \ a_2 \ w_3(z) \ c_3 \]
BTO Example:

\[ s = r_1(x) \, w_2(x) \, r_3(y) \, w_2(y) \, c_2 \, w_3(z) \, c_3 \, r_1(z) \, c_1 \]
\[ s = r_1(x) \ w_2(x) \ r_3(y) \ w_2(y) \ c_2 \ w_3(z) \ c_3 \ r_1(z) \ c_1 \]
BTO Example:

\[ s = r_1(x) \ w_2(x) \ r_3(y) \ w_2(y) \ c_2 \ w_3(z) \ c_3 \ r_1(z) \ c_1 \]
Outline

1. Introduction to STM
   - Prelim Definitions
   - STM Model

2. STM Concurrency Control Protocols
   - Basic Timestamp Ordering (BTO) Algorithm
   - Serialization Graph Testing (SGT) Algorithm
   - Multiversion Time Stamp Ordering Protocol

3. Test Application - Set
   - Overview
   - Pseudocode
   - Performance Evaluation Against Synchrobench

4. Conclusion

5. Summary
- SGT maintains a conflict graph.
- Inserts edges for each read write operation.
- Checks if graph is still acyclic. If not take actions to ensure acyclicity.
When a operation $p_i(X)$ arrives:

1. If its first operation of transaction create node $T_i$.
   - Add real time edges from all committed Tx to current Tx nodes.

2. If operation is:
   - read: do validation step.
   - write: buffer locally defer untill tryCommit operation.

3. Validation: insert edge $T_j \rightarrow T_i$ for each conflicting operation scheduled before $T_i$, $i \neq j$
   - graph becomes cyclic: Abort $T_i$
   - graph is acyclic: All is well 😊
SGT Example:

\[ s = r_1(y) r_2(y) w_1(y) w_1(x) c_1 w_2(x) c_2 w_3(x) c_3 \]

\[ r_1(y) \]
SGT Example:

\[ s = r_1(y)r_2(y)w_1(y)w_1(x)c_1 w_2(x)c_2 w_3(x)c_3 \]
SGT Example:

\[ s = r_1(y)r_2(y)w_1(y)w_1(x)c_1w_2(x)c_2w_3(x)c_3 \]

\[ r_1(y)r_2(y)w_1(y) \]

Diagram: Two circles labeled T1 and T2 connected by an arrow labeled (Y).
SGT Example:

\[ s = r_1(y)r_2(y)w_1(y)w_1(x)c_1 w_2(x)c_2 w_3(x)c_3 \]

\[ r_1(y)r_2(y)w_1(y)w_1(x)c_1 \]
SGT Example:

\[ s = r_1(y)r_2(y)w_1(y)w_1(x)c_1w_2(x)c_2w_3(x)c_3 \]

\[ r_1(y)r_2(y)w_1(y)w_1(x)c_1w_2(x) \]
SGT Example:

\[ s = r_1(y)r_2(y)w_1(y)w_1(x)c_1w_2(x)c_2w_3(x)c_3 \]
SGT Example:

\[ s = r_1(y)r_2(y)w_1(y)w_1(x)c_1w_2(x)c_2w_3(x)c_3 \]
Outline

1 Introduction to STM
   - Prelim Definitions
   - STM Model

2 STM Concurrency Control Protocols
   - Basic Timestamp Ordering (BTO) Algorithm
   - Serialization Graph Testing (SGT) Algorithm
   - Multiversion Time Stamp Ordering Protocol

3 Test Application - Set
   - Overview
   - Pseudocode
   - Performance Evaluation Against Synchrobench

4 Conclusion

5 Summary
MVTO Algorithm

- **Read rule** - When a read operation \( r_i(x) \) arrives,
  
  - If \( p_i(X) \) is \( \text{tm\_read} \): If the value of \( X \) is in the local log, \( p_i(X) \) directly return this value. Else \( p_i(X) \) reads a value \( X_k(K_{th} \text{ version of } X) \) created by \( T_k \) such that, \( TS(X_k) \) is the largest timestamp \(< TS(T_i) \). And \( X_k \) is also locally logged.

- **Write rule** - When a write operation \( w_i(x) \) arrives,
  
  - If \( p_i(X) \) is \( \text{tm\_write} \): It creates a new version of \( X \) and its value is locally logged.

- **Commit rule** - When a commit request arrives,
  
  For each data object \( x \) in the write buffer,
  
  - If \( r_j(X_k) \) has already been scheduled such that \( TS(T_k) < TS(T_i) < TS(T_j) \); this implies the version created by \( T_i \) is obsolete and it needs to be aborted.
  
  Otherwise, add version \( X_i \) to shared memory and it is made visible to other transactions.
STM Middleware Design

Shared Memory

transaction Objects

MiddleWare

BTO, SGT, MVTO

commit
Abort

user

T1
tm_write

T2
tm_read

T3
tm_commit
Outline

1 Introduction to STM
   • Prelim Definitions
   • STM Model

2 STM Concurrency Control Protocols
   • Basic Timestamp Ordering (BTO) Algorithm
   • Serialization Graph Testing (SGT) Algorithm
   • Multiversion Time Stamp Ordering Protocol

3 Test Application - Set
   • Overview
   • Pseudocode
   • Performance Evaluation Against Synchrobench

4 Conclusion

5 Summary
Application: Set using Linked-list

- **Node structure:**
  1. ID - unique int, val - complex datastructure.
  
- **List Structure:**
  1. 2 sentinel nodes at head and tail.
  2. Nodes are sorted in order of their unique ID.
Application: Linked List

- Insert()
Application: Linked List

- Insert()
- Delete()
Application: Linked List

- Insert()
- Delete()
- Lookup()
Application: Linked List

- Insert()
- Delete()
- Lookup()
- Framework made it easy do them all within transactions
Application: Linked List

- Insert()
- Delete()
- Lookup()

Framework made it easy to do them all within transactions.

For sure you will make every node as a shared variable why?
Application: Linked List

- Insert()
- Delete()
- Lookup()

Framework made it easy do them all within transactions

For sure you will make every node as a shared variable why?
Outline

1. Introduction to STM
   - Prelim Definitions
   - STM Model

2. STM Concurrency Control Protocols
   - Basic Timestamp Ordering (BTO) Algorithm
   - Serialization Graph Testing (SGT) Algorithm
   - Multiversion Time Stamp Ordering Protocol

3. Test Application - Set
   - Overview
   - Pseudocode
     - Performance Evaluation Against Synchrobench

4. Conclusion

5. Summary
Algorithm 1 set_add algorithm

1: procedure set_add(val)
2: \[ T \leftarrow \text{begin}() \] \quad \triangleright \text{transaction begin}
3: \[ \text{set}_\text{obj}_p \leftarrow \text{new common}_\text{tOB} \] \quad \triangleright \text{initialize prev and next pointers of list}
4: \[ \text{tm_read}(T, \text{set}_\text{obj}_p) \]
5: \[ \text{set}_\text{obj}_n \leftarrow \text{next}(\text{set}_\text{obj}_p) \]
6: \[ \text{tm_read}(T, \text{set}_\text{obj}_n) \]
7: \[ \textbf{while} \ \text{value}(\text{set}_\text{obj}_n) \prec \text{val} \ \textbf{do} \]
8: \[ \quad \text{set}_\text{obj}_p \leftarrow \text{set}_\text{obj}_n \]
9: \[ \quad \text{set}_\text{obj}_n \leftarrow \text{next}(\text{set}_\text{obj}_n) \]
10: \[ \quad \text{tm_read}(T, \text{set}_\text{obj}_n) \]
11: \[ \textbf{if} \ \text{value}(\text{set}_\text{obj}_n) \neq \text{val} \ \textbf{then} \]
12: \[ \quad \text{set}_\text{obj} \leftarrow \text{newshared}_\text{obj}(\text{val}) \]
13: \[ \quad \text{next}(\text{set}_\text{obj}_p) \leftarrow \text{set}_\text{obj} \]
14: \[ \quad \text{next}(\text{set}_\text{obj}) \leftarrow \text{set}_\text{obj}_n \]
15: \[ \quad \text{tm_write}(T, \text{set}_\text{obj}) \]
16: \[ \quad \text{tm_write}(T, \text{set}_\text{obj}_p) \]
17: \[ \quad \text{tm_write}(T, \text{set}_\text{obj}_n) \]
18: \[ \text{tm_commit}(T, \text{error}_\text{id}) \] \quad \triangleright \text{commit transaction if write succeeds}
Algorithm 2 set_remove algorithm

1: procedure set_remove(val)
2:    \( T \leftarrow \text{begin}() \) ▶ transaction begin
3:    set_obj_p \leftarrow \text{new common_tOB} ▶ initialize prev and next pointers of list
4:    tm_read(T, set_obj_p)
5:    set_obj_n \leftarrow \text{next(set_obj_p)}
6:    tm_read(T, set_obj_n)
7:    while true do
8:       \( v = \text{value(next(set_obj_p))} \)
9:       if \( v \geq val \) then
10:          break
11:     set_obj_p \leftarrow set_obj_n
12:    set_obj_n \leftarrow \text{next(set_obj_n)}
13:    tm_read(T, set_obj_n)
14:    if \( v == val \) then
15:       \text{next(set_obj_p)} \leftarrow \text{next(set_obj_n)}
16:    tm_write(T, set_obj_p)
17:    tm_write(T, set_obj_n)
18:    tm_commit(T, error_id) ▶ commit transaction if write succeeds
Algorithm 3 set_contains algorithm

1: procedure set_contains(val)
2:     T ← begin() ▷ transaction begin
3:     set_obj_p ← new common_tOB
4:     tm_read(T, set_obj_p)
5:     set_obj_n ← next(set_obj_p)
6:     tm_read(T, set_obj_n)

7:     while true do
8:         v = value(next(set_obj_p))
9:         if v >= val then
10:             break
11:             set_obj_p ← set_obj_n
12:         set_obj_n ← next(set_obj_n)
13:         tm_read(T, set_obj_n)
14:     if v == val then
15:         Found
16:     else
17:         NotFound
18:     tm_commit(T, error_id) ▷ commit transaction
Performance Analysis with Synchrobench Benchmark

System Setup

Intel(R) Core(TM) i3 CPU with 2 cores, 3.20GHz and 3GB main memory. The system uses Ubuntu version 16.04 for 64bit systems, Glibc version 2.23 and g++ 5.4.0-6.

Benchmark Used

- Synchrobench set implemented as linked list module.
- **Protocols**: lazy-list, lockfree-list, ESTM, lock-coupling list.
Performance Analysis with Synchrobench Benchmark

Performance Metrics

- **Clock measures**: linux man page specification
  1. CLOCK_MONOTONIC_RAW: Clock that cannot be set and represents monotonic time since some unspecified starting point.
  2. CLOCK_PROCESS_CPUTIME_ID: Per-process CPU-time clock (measures CPU time consumed by all threads in the process).
  3. CLOCK_THREAD_CPUTIME_ID: Thread-specific CPU-time clock.

- **Attributes**: average time taken in milli-seconds Vs number of threads.

- **Operation ratio**: 70% operations are updates and 30% are read only operations
Performance Analysis with Synchrobench - per thread time

estm Vs (SGT, BTO and MVTO)

lazylist Vs (SGT, BTO and MVTO)
Performance Analysis with Synchrobench - per thread time

lock coupling Vs (SGT, BTO and MVTO)

lockfree Vs (SGT, BTO and MVTO)
thread-cputime, process-cputime & monotonic-raw for all protocols
Software Transactional Memory is one of most promising alternative to bring parallel programming easier.

Our analysis shows that for a number of threads greater than 60 and update rate 70%:

- BTO takes (17% to 29%) and (6% to 24%) less CPU time per thread when compared against lazy-list and lock-coupling list respectively.
- MVTO takes (13% to 24%) and (3% to 24%) less CPU time per thread when compared against lazy-list and lock-coupling list respectively.
- BTO and MVTO have similar per thread CPU time. BTO and MVTO outperform SGT by 9% to 36%.
Implementation of read/writes model based software transactional memory using state of art concurrency control protocols; BTO, SGT and MVTO.

Transactional implementation of the set as linked list with add, delete and contains methods, using our STM middleware.

Detailed Performance comparison of BTO, SGT and MVTO of the STM middleware against the set implementation using lazy-list, lock-coupling list, lock-free list and ESTM concurrency control mechanisms of synchrobench benchmark.

For exhaustive evaluation, we used three clock measures: per thread CPU time, total CPU time and real time taken by the application.
Summary

- Implementation of read/writes model based software transactional memory using state of art concurrency control protocols; BTO, SGT and MVTO.
- Transactional implementation of the set as linked list with add, delete and contains methods, using our STM middleware.
Implementation of read/writes model based software transactional memory using state of art concurrency control protocols; BTO, SGT and MVTO.

Transactional implementation of the set as linked list with add, delete and contains methods, using our STM middleware.

Detailed Performance comparison of BTO, SGT and MVTO of the STM middleware against the set implementation using lazy-list, lock-coupling list, lock-free list and ESTM concurrency control mechanisms of synchrobench benchmark.
Summary

- Implementation of read/writes model based software transactional memory using state of art concurrency control protocols; BTO, SGT and MVTO.
- Transactional implementation of the set as linked list with add, delete and contains methods, using our STM middleware.
- Detailed Performance comparison of BTO, SGT and MVTO of the STM middleware against the set implementation using lazy-list, lock-coupling list, lock-free list and ESTM concurrency control mechanisms of **synchrobench benchmark**.
- For exhaustive evaluation, we used three clock measures: per thread CPU time, total CPU time and real time taken by the application.
Acknowledgement

Our special Thanks to ANURAG labs, DRDO for supporting the work.

code website

www.iith.ac.in/~sathya_p/lib_files/Research-Libraries.html
For Further Reading...

Gerhard Weikum and Gottfried Vossen.  
Morgan Kaufmann, 2002.

Vincent Gramoli.  
More than you ever wanted to know about synchronization: Synchrobench, measuring the impact of the synchronization on concurrent algorithms.  

Rachid Guerraoui and Michal Kapalka.  
*Principles of Transactional Memory.*  

Tim Harris, James R. Larus, and Ravi Rajwar.  
*Transactional Memory, 2nd edition.*  

Petr Kuznetsov and Sathya Peri.  
Non-interference and local correctness in transactional memory.  

Nir Shavit and Dan Touitou.  
Software transactional memory.  
Thank You
Different Time Measure Comparision

BTO Vs (different clock measures)