Software Transactional Memory
C++14 STM

Functional Specification

Zoltan Fuzesi

April 10, 2018

Student ID: C00197361
Supervisor: Joseph Kehoe
Institute of Technology Carlow
Software Engineering

Institiúid Teicneolaíochta Cheatharlach
INSTITUTE of TECHNOLOGY CARLOW
At the Heart of South Leinster

Contents

1 Introduction

Since the modern computers has more the one core embedded into the CPU, the concurrent and parallel programming is the way to keep the CPU throughput as high as possible and create faster applications. The concurrent and parallel programming requires to sharing memory spaces between processes, cores associated with a running application.

The purpose of this project to implement a Software Transactional Memory library in C++14 programming language, to enable other programs to include and use this library as a lock free solution, to use for concurrent programming to protect shared memory spaces, which are used by an application. Because the STM library implementation does not require graphical user interface, database connection and network connectivities, the implementation is limited only for logical C++ code solution, that can be used and tested by an external application. The library itself will be a file with '*.a' or '*.so' extension on Linux , that need to be placed into the file system in the Linux operating (shared *so file) system or placed to the C++ project folder structure (static *.a) to link together with the application. So, as the part of the project, it is require to create a test application to illustrate and test the library usage.

As the mandatory requirements, the project will be fully documented by the Doxygen document generator tool, to provide the description of the available interface functions to other programmers, if they intend to use the library. Create a test application to demonstrate the library usage.

To make it available on different operating systems, the library will be developed and tested on Linux, Windows and MAC OSX to make it available in cross platform versions. As a final result, create benchmarking test, to compare with other Software Transactional Memory solutions or similar systems.

2 What the product can do

The STM library will be developed in C++14 language syntax to provide lock free solution to client applications to protect shared memory within a transactional process. It will support an API interface to the applications to enable library call(s) after a transaction object has created. It will allows the transaction object to pass objects to the library, to store and make changes on them. As the part of the transaction process, the library will create a transaction, make shared local data structure to store objects involved in the transactions, and when finish with the changes, try to end the transaction and only write changes in the memory spaces if the transaction not fail. When the transaction fails, then need to restart the process. Because the STM library provide transactional environment to the client applications, the programmers don't need to worry about to use mutex, semaphore and can avoid of deadlock and livelock situations.

3 Functional description

Software Transactional Memory allows to compose several actions, that all can be execute together and run as a single transaction. The library will provide public API functions, such as register, load, store and commit to use values/pointers to work with, and these functions will working together to achieve the transactional behaviour. When the transaction library receive the data to work with, then will create and start a transaction. At the last step in the transaction try to commit the changes, that need to carry out on the shared data. It is involves in few inner steps such as compare the version numbers and update the changes. If the version number are different between the working object and the global object, then the transaction will restart. The transaction steps are detailed in below.

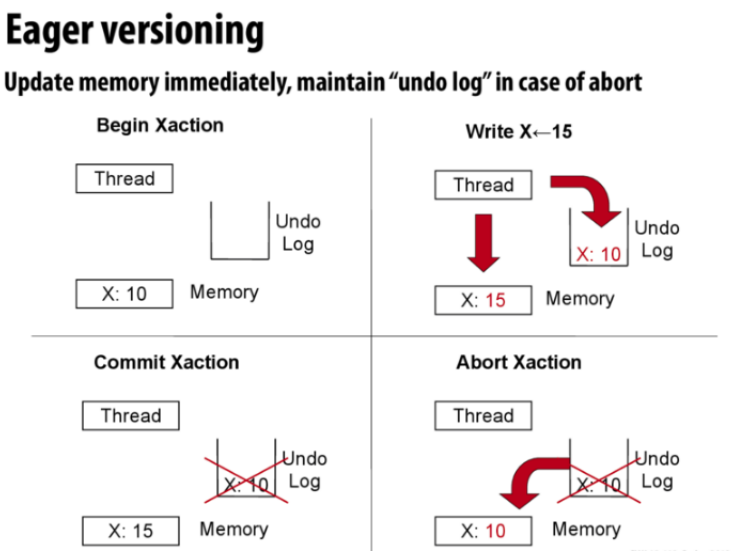


Figure 1: *A simple example of eager version. [3].*

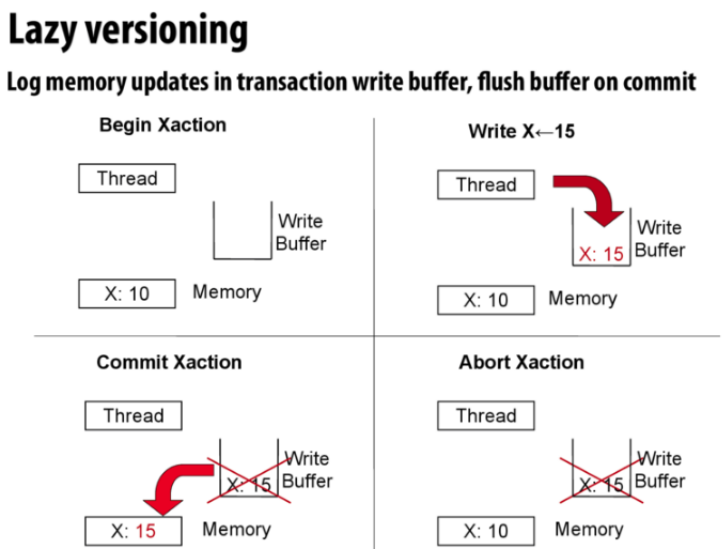


Figure 2: *A simple example of lazy version. [3].*

3.1 Create transaction

When the external program creates an object of the transaction library, it's functions available to the transaction object. The transaction object able to pass the required values to the transaction library, to make changes on them. When the object calls the function and passing the values/object to the transactional memory library, the transaction has created and the transaction life cycle begin.*Figure 3*

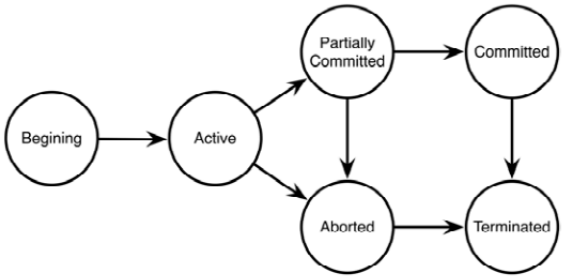


Figure 3: *Transaction basic life cycle. [4].*

The values/object received by the transactional memory library, it creates a copy of the value/object to the local shared data set to work with. It is an important step, since the library not making changes on the original value until the end of the transaction life cycle.*Figure 4*

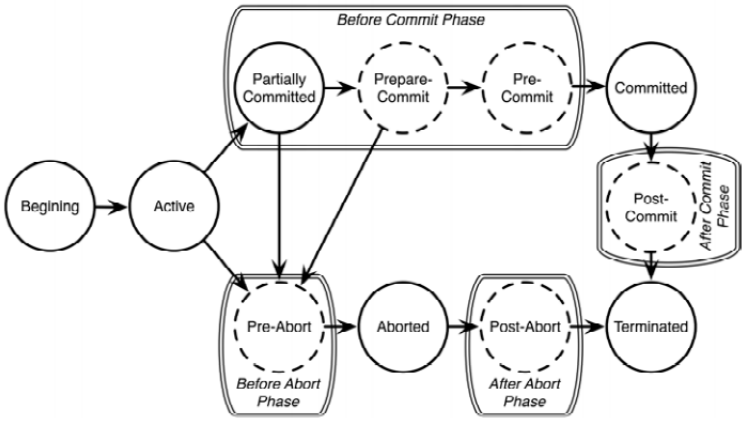


Figure 4: *Transaction life cycle states. [4].*

A copy created and associating a version number with it. This version number

will be used to determine the changes on the value/object by the other transactions within the context.

3.2 Start transaction

When the transaction start, the library check the version number associated with the data/object, to compare against the transaction own copy. If the version numbers are not the same, then the transaction aborting and restart with the new copy of the origanl object. If the version numbers are same, then the transaction locks the global data set copy of the value/object and make the changes. Before the transaction commit, make permanent changes on the value/object used in the transaction it needs to compare the version number again to determine the conflicts.

3.3 Commit transaction

This phase of the transaction, the library locks the used shared memory spaces in an atomic block, and hide from the other processes to write changes into the local data set. After the changes has made on the local data set, before finalize the write process to the destination shared memory spaces, must to determine if any other process had accessed the same data set, and made changes on it. It is happening with compare the transaction object version number with the shared memory space version number. The transaction commits the changes if no conflict detected between version numbers. When the changes has made on the shared memory space, then release the lock on the data/object and the transaction has finished.

3.3.1 Version checking

The version number representing the changes associated with the object accessed by the transactions. The transaction can detect the conflict early when first time accessing the value or when the transaction attempts to commit. When a transaction commit the changes, write the new values to the shared memory space, and the same time increasing the global version number to indicate to the other transactions, that some changes happened with the object. At commit time the transaction compare the local object version numbers against the global object version number. The version number in the actual transaction and the global

version number associated with the object must match. If the values are not the same the transaction will be cancelled and rolling back the changes and restart the transaction.

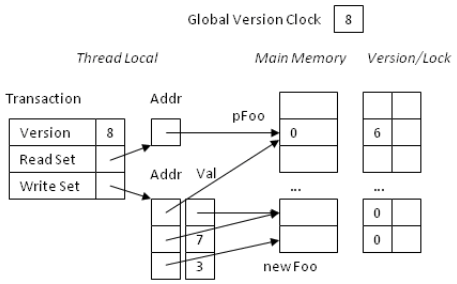


Figure 5: *Check version number. [1].*

3.3.2 Roll-back transaction

The roll-back part of the transaction, simply goes back to the begging of the transaction and restart the process. The library should implement some sort of checking logic, that the roll-back process must be cancelled at some stage, otherwise it can create an infinite or very long looping time in the client application.

3.3.3 Updating

At commit time, if the compared versions are holding same values, then the transaction updating the shared memory space on the target data set and closing the transaction.

The following code snippet is an atomic transaction with compiler support to change nodes in linked list structure:

```
atomic {
  int x = node.head;
  tail = node.tail;
}
```

4 Target users

The potential target users can be any programmer, how planning to use concurrent programming features in their application, but in the same time they wants to avoid of the lock based programming disadvantages. Since, STM provide interface to access the built in transactional functionalities through object accessible method call, it is easy to use because only need to include the library into the client application.

The potential target users can be who:

1. Individual programmers, software companies.
2. Want to use transactional environment.
3. Want to avoid of lock based programming.
4. Want to use concurrent programming features in safe environment.
5. Want to avoid deadlock and livelock.

5 Metrics

The C++14 Software transactional Memory project is successful, when the library working as it is expected in all circumstances. To fulfil these achievement the project must produce the following requirements:

Mandatory requirements:

1. The project have to:
   - Implemented Software Transactional Memory library in C++ language working on Linux platform.
   - Documentation of the library using Doxygen documentation tool.
   - Tutorial showing how to use the library, including test application that use the STM library.

Discretionary requirements:

2. The project have to:
   - Portable : tested across multiple platform (Windows, Linux, OSX).
   - Web site: library backed up and demonstrating it usage.

Exceptional requirements:

3. The project have to:
   - Benchmarked : The project including benchmarking test to show the performance in different circumstances.
   - Comparisons : Compare between other STM solutions and other approaches with usability, performance.

5.0.1 FURPS

As part of the metrics, the library should implement not functional requirements (FURPS)[2]

1. **Functionality** - The product features that represent the domain of the product solution being developed.
2. **Usability** - Capturing and stating the requirements based on functional interface of the product and how easy to use based on the product documentation.
3. **Reliability** - Includes the availability, frequency of failures and recoverability if the product has unexpected failure.
4. **Performance** - Includes throughput, response time with diverse circumstances - example 10, 100 or 1000 threads are using the library with benchmarking test.
5. **Supportability** - Includes all other requirements such as testability, adaptability, maintainability, scalability and portability between platforms.

6 Other STM libraries

The Software Transactional Memory was implemented in C/C++ language many time before. However, this project must produce a fully implemented STM library to support lock free transactions, possibly the library functionalities will be a bit restricted compare with the other C/C++ library implementations available. Why? Because those libraries are available years ago and developed over the time.

The Software Transactional Memory was implemented in many different languages as well, including C, C++, Clojure, Common Lisp, Haskell, Java, Python, Pearl and many more.

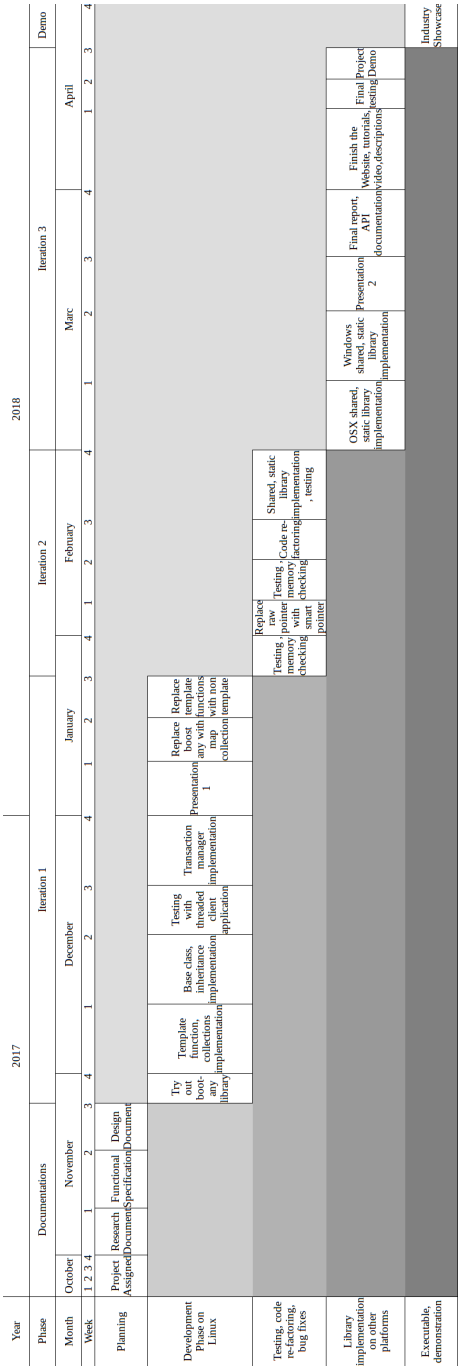
Some C/C++ STM implementation:

1. TinySTM is a lightweight and efficient word-based STM implementation in C++.
2. libCMT based on Composed Memory Transaction written in C.
3. Intel STM Complier prototype Edition implements STM for C/C++ directly in a complier
4. stmmap based on shared memory accessing written in C.
5. TL2 transaction locking implementation of STM thats been most widely accepted
6. CTL based on TL2 and added many extensions and optimization
7. many more

This project will be based on TL2 logics, which are the followings:

1. Sample global version-clock.
2. Run through a speculative execution.
3. Lock the write-set.
4. Increment global version-clock.
5. Validate the read-set.
6. Commit and release the locks.

7 Project Iterations / Updated



7.1 Detailed Iterations description / Updated

2017/2018

1. October
   1.1. Project assigned.
2. November
   2.1. Writing Research Document .
   2.2. Writing Functional Specification.
   2.3. Writing Design Document.
   2.4. **Begin Iteration 1 :** Research to find a C++ collection type to store any type of object. Try out boost-any C++ library.
3. December
   3.1. Implementing template some functions and collections in the transaction class to store objects within the library.
   3.2. Implementing the OSTM base class (blueprint of the class) virtual methods and the required unique identifier generation processes to provide unique identifier to inherited objects the intend to use with the library.
   3.3. First time testing the base functions with a client application, using the library files only.
   3.4. Implementation of the Transaction Manager to control the transactions.
4. January 2018
   4.1. First presentation.
   4.2. Replacing Boost any library with c++ built in std::map.
   4.3. Replacing template functions with non template functions using the OSTM polymorphic objects.
   4.4. Testing the library functionalities with the new polymorphic objects and with the Transaction Manager, memory checking.

5. February
    - 5.1. Replacing RAW pointers with smart pointers, because the RAW pointer can not be deleted from the child class methods.
    - 5.2. Testing library with the smart pointer implementation, memory checking.
    - 5.3. Code re-factoring.
    - 5.4. Implementation of the Shared and the Static libraries in Linux system, testing libraries.
6. March
    - 6.1. MAC OSX Static and Shared libraries implementation, testing.
    - 6.2. Windows Static and Shared libraries implementation, testing using Visual Studio.
    - 6.3. Second presentation.
    - 6.4. API documentation, full library documentation using Doxygen.
7. April
    - 7.1. Website, tutorials, video tutorials, description implementation.
    - 7.2. Final Testing.
    - 7.3. Project Demo.

8 Bibliography

References

[1] m. Bartosz. Beyond Locks: Software Transactional Memory. https://bartoszmilewski.com/2010/09/11/beyond-locks-software-transactional-memory/. 2010 (Accessed : 11/11/2017).

[2] cs.unh.edu. CS 619 Introduction to OO Design and Development . http://www.cs.unh.edu/~cs619/slides/usecase.pdf. 2013 (Accessed : 11/11/2017).

[3] A. X. T. Mayank. Implementing Transactional Memory. http://15418.courses.cs.cmu.edu/spring2013/article/40. 2013 (Accessed : 11/11/2017).

[4] D. Richard, L. Joao, and C. Goncalo. Developing Libraries Using Software Transactional Memory. https://www.google.ie/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwiHvNOJ0L7XAhVqCMAKHeplBUQQFggpMAA&url=http%3A%2F%2Fwww.comsis.org%2Fpdf.php%3Fid%3D038-0612&usg=AOvVaw3J5gJGporTwDe-WY27QHCC. 2016 (Accessed : 11/11/2017).