Efficiency of Software Transactional Memory

Erik Helin & Henry Rodrick

Supervisor: Mads Dam

KTH, 26th of January 2010

1 Background

The question of how to handle concurrency in computer programs have lately become more relevant than ever because of the advent of multicore processors. However, the de facto mechanism for concurrency control on shared memory data structures is still mutual exclusion locks.

Although such locking mechanisms appear conceptually simple, programs using them are known to be notoriously hard to get right. Dead locks, the event of two or more threads waiting for each other to finish, and priority inversion, the scenario where a low priority thread blocks a higher prioritized one, are two classical examples of the issues that one can run into. [6]

One approach to concurrency control that has received a lot of attention lately is Software Transactional Memory (STM). This method borrows heavily from the transaction based methods commonly used in database systems to handle simultaneous queries.

The basic idea of such transactions is that a query is processed as if its the only thing accessing the database. After the query is processed, the result is committed back to the database. Given there are no conflicts with other concurrent transactions, the commit succeeds and the result is written back to the database. If the commit fails, the transaction is rolled back and run again.

The idea of using similar transactions for concurrency control has been around since at least 1977, but a practical implementation of transactional memory first appeared in 1993 when Herlihy and Moss proposed a hardware-supported solution. [1] Lately, software based transactional memory systems have been developed. A notable contribution is the STM system in Concurrent Haskell, described in 2005 by Tim Harris, Simon Peyton Jones et al. [3]

Although it might seem like STM, just as its database counterpart, would be an elegant solution to many shared-memory concurrency issues, critics have highlighted several possible problems such as false conflicts and over-instrumentation (unnecessary generation of read/write barriers due to lack of application knowledge). It's believed that such problems would appear in real-world applications based on transactional memory and constitute serious performance bottlenecks. [5]

2 Problem statement

The objective of the paper is to evaluate the effiency of STM. Effiency is a very subjective term, and in this paper it will be defined by a series of tests and evaluations. The effiency of STM will then be determined by the results of these tests and evaluations.

3 Project plan

The paper will first provide the reader with a thorough introduction to STM, which explains the key concepts of STM as well as relates it to other concurrency methods and models. This part of the paper will also discuss the theoretical strenghts and weaknesses of STM.

To determine if STM is an efficient solution for developing concurrent applications we will develop small programs which solves classical concurrency problems found in [2]. For each problem, we will develop the solutions by two different methods, the first one by using semaphors and the second one by using STM.

The programming language which will be used when developing the solutions is Haskell. The reason for choosing Haskell is because it provides us with an opportunity to use both semphors and STM without installing any third-part libraries [3] [4]. The only other major programming language which features this is Clojure [7], which none of us have used.

All the solutions will then be measured by several tests and we will compare the results. The tests will consist of:

- **Number of operations** - we will compare the number of operations which the Glasgow Haskell Compiler (GHC)[9] generates for the different solutions
- **Execution time** - we will compare the execution of time of the different solutions by using the GHC profiler and also DTrace [10]. Simpler tools such as the UNIX `time` program will probably also be used
- **Scalability** - we will compare how well the different solutions scales on a CPU with 1,2 and 4 cores by executing the programs on different machines
- **Memory usage** - we will compare the memory used by the two different solutions with the help of the GHC heap profiler and Valgrind [8].
- **Number of lines** - we will compare the number of lines of code used for each solution to examine if the usage of STM generates shorter programs

The last test is more subjective than the others, but to provide an answer if STM is efficient or not, we also want to consider how effective it is to use it.

4 Time plan

- **14/2** The part of paper which explains STM and other concurrency models shall be finished
- **7/3** At this time, all the solutions shall be solved, although the quality of the code can be of prototypal nature
- **4/4** All the tests and measurements shall now be done, and all the data we need for analysis should have been collected
- **25/4** The paper shall be finished, and the remaning 3 days will be used for proofreading and correcting smaller errors.

References

[1] J Larus and R Rajware, *Transactional Memory*, Morgan & Claycool Publishers, First Edition, 2007

[2] A B Downey, *The little book of semaphors*. Green Tea Press, Version 2.1.5, 2008

[3] T Harris et al, *Composable Memory Transactions*, PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallell programming, 2005

[4] SL Peyton Jones, A Gordon and S Finne, *Concurrent Haskell*, 23rd ACM Symposium on Principles of Programming Languages, 1996

[5] R M Yoo et al, *Kicking the Tires of Software Transactional Memory: Why the Going Gets Tough*, SPAA '08: Proceedings of the twentieth annual symposium on Parallellism in algorithm and architectures, 2008

[6] K Fraser and T Harris, *Concurrent Programming Without Locks*, ACM Transactions on Computer Systems, 2007

[7] `http://clojure.org/concurrent_programming`, January, 2010

[8] `http://valgrind.org/`, January, 2010

[9] `http://www.haskell.org/ghc/`, January, 2010

[10] `http://en.wikipedia.org/wiki/DTrace`, January, 2010

[11] `http://en.wikipedia.org/wiki/Software_transactional_memory`, Janurary, 2010