IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-6, Issue-7, July 2018 E-ISSN: 2347-2693

Flexible Programming Approach using STM

Ryan Saptarshi Ray [1*], Parama Bhaumik [2], Utpal Kumar Ray [3]

[1,2,3]Dept. of Information Technology, Jadavpur University, Kolkata, India

[*]*Corresponding Author: ryan.ray@rediffmail.com, Tel.: 9831520613*

Available online at: www.ijcseonline.org

Accepted: 19/July/2018, Published: 31/July/2018

***Abstract*—** Software Transactional Memory (STM) is a promising new approach to programming shared-memory parallel processors which does not suffer from the drawbacks of locks. However STM also has some limitations. One of the limitations of STM is that while programming with STM users have to identify the critical sections explicitly and enclose them in transactions using appropriate STM calls to ensure synchronization. This approach is similar to using locks in parallel programs. This paper introduces a new flexible approach for programming using STM in which users do not need to identify critical sections explicitly. In this approach whenever users need to perform read or write operations they can do so using appropriate STM calls and STM will ensure synchronization by its internal constructs. Thus users can concentrate only on the algorithm of the parallel problem without thinking about synchronization. Thus this approach is very user-friendly. Time taken will also be less than lock programming as users do not have to identify critical sections explicitly.

***Keywords*—** Multiprocessing, Parallel Processing, Locks, Software Transactional Memory, Flexible Programming Approach

I. INTRODUCTION

Ensuring synchronization is a very important aspect of parallel programming. Currently locks are used to ensure synchronization. But locks suffer from some drawbacks. Software Transactional Memory (STM) is a promising new approach to programming shared-memory parallel processors which does not suffer from the drawbacks of locks. But STM also has some limitations. One of the limitations of STM is that while programming with STM users have to identify the critical sections explicitly and enclose them in transactions using appropriate STM calls to ensure synchronization. This approach is similar to using locks in parallel programs where also users have to identify critical sections explicitly and enclose them using appropriate lock calls to ensure synchronization. This paper introduces a new flexible approach for programming using STM in which users do not need to identify critical sections explicitly. In this approach whenever users need to perform read or write operations they can do so using appropriate STM calls and STM will ensure synchronization by its internal constructs. Thus users can concentrate only on the algorithm of the parallel problem without thinking about synchronization. Thus this approach is very user-friendly. Time taken will also be less than lock programming as users do not have to identify critical sections explicitly.

Section III describes the Flexible Programming Approach. Section IV solves the Readers-Writers Problem using the Flexible Programming Approach. Section V shows the experimental results for solving the Readers-Writers Problem using Flexible Programming Approach.

II. RELATED WORK

Different approaches have been proposed to improve the performance of STM. These are discussed below.

In 2007 Yang Ni, Vijay Menon, Richard L. Hudson, Ali-Reza Adl-Tabatabai, J. Eliot, B. Moss, Bratin Saha, Antony L. Hosking, Tatiana Shpeisman published a paper entitled "Open Nesting in Software Transactional Memory". **[1]** This paper presented an implementation of open nested transactions in a Java-based software transactional memory (STM) system. It described new language constructs to support open nesting in Java and also discussed new abstract locking mechanisms that a programmer could use to prevent logical conflicts. It demonstrated how these constructs could be mapped efficiently to existing STM data structures. Finally, it evaluated the system on a set of Java applications and data structures, demonstrating how open nesting could enhance application scalability. In 2009 Zhengyu He and Bo Hong published a paper entitled "Impact of Early Abort Mechanisms on Lock-Based Software Transactional Memory". **[2]** This paper presented a theoretical analysis characterizing the properties of early abort and its impact on the performance of lock-based STMs. Queuing theory was adopted to model the behaviors of transactional execution. Analytical results were obtained for STMs with and without early abort. The analysis was validated through extensive

experiments. Also in 2009 Yossi Lev, Victor Luchangco, Virendra J. Marathe, Mark Moir, Dan Nussbaum and Marek Olszewski published a paper entitled "Anatomy of a Scalable Software Transactional Memory". **[3]** This paper described novel techniques to eliminate bottlenecks from existing STM mechanisms and presented SkySTM. SkySTM was the first STM that supported privatization and scaled on modern multicore multiprocessors with hundreds of hardware threads on multiple chips. A central theme in this work was avoiding frequent updates to centralized metadata, especially for multi-chip systems, in which the cost of accessing centralized metadata increased dramatically. A key mechanism used to do so was a scalable nonzero indicator (SNZI), which was designed for this purpose. A secondary contribution of the paper was a new and simplified SNZI algorithm. The scalable privatization mechanism imposed only about 4% overhead in low-contention experiments; when contention was higher, the overhead still reached only 35% with over 250 threads. In contrast, prior approaches had been reported as imposing over 100% overhead in some cases, even with only 8 threads. In 2010 Justin E. Gottschlich, Manish Vachharajani, Jeremy G. Siek published a paper entitled "An Efficient Software Transactional Memory Using Commit-Time Invalidation". This paper presented an efficient implementation of committime invalidation, a strategy where transactions resolved their conflicts with in-flight (uncommitted) transactions before they commited. **[4]** Commit-time invalidation supplied the contention manager (CM) with data that was unavailable through commit-time validation, allowing the CM to make decisions that increased transaction throughput. Commit-time invalidation also required notably fewer operations than commit-time validation for memory-intensive transactions, used zero commit-time operations for dynamically detected read-only transactions, and guaranteed full opacity for any transaction in O(N) time, an improvement over incremental validation's $O(N^2)$ time. The experimental results showed that for contending workloads, the efficient commit-time invalidating software TM (STM) was up to 3 times faster than TL2, a state-of-the-art validating STM. In 2011 Sandhya S.Mannarswamy and Ramaswamy Govindarajan published a paper entitled "Variable Granularity Access Tracking Scheme for Improving the Performance of Software Transactional Memory". **[5]** In order to mitigate the disadvantages associated with Uniform Granularity Access Tracking (UGAT) scheme, this paper proposed a Variable Granularity Access Tracking (VGAT) scheme. It proposed a compiler based approach wherein the compiler used inter-procedural whole program static analysis to select the access tracking granularity for different shared data structures of the application based on the application's data access pattern. It described the prototype VGAT scheme, using TL2 as the STM implementation. The experimental results revealed that VGAT-STM scheme could improve the application performance of STAMP benchmarks from 1.87% to up to 21.2%.

In our work we have used the flexible approach to improve the performance of STM which has not been used earlier.

III. FLEXIBLE PROGRAMMING APPROACH

In the flexible approach with STM programmers do not need to identify the critical sections of the code explicitly. For performing read operation on a shared element stm_unit_load( ) function is used. Similarly for performing write operation on a shared element stm_unit_store( ) function is used. Then STM by its internal constructs ensures synchronization in the program. stm_unit_load( ) reads the specified memory location outside of the context of any transaction and returns its value. The operation behaves as if executed in the context of a dedicated transaction (i.e., it executes atomically and in isolation) that never aborts, but may get delayed. Similarly stm_unit_store( ) writes a value to the specified memory location outside of the context of any transaction. It also behaves as if executed in the context of a dedicated transaction (i.e., it executes atomically and in isolation) that never aborts, but may get delayed.

STM follows the principle of optimistic execution. The benefit of this optimistic approach is increased concurrency: no thread needs to wait for access to a resource, and different threads can safely and simultaneously modify disjoint parts of a data structure that would normally be protected under the same lock. This means that all the transactions can execute simultaneously without being concerned about the execution of other transactions. When all transactions finish their execution the variables whose values were changed are checked to ensure that the values are consistent. If not then all the transactions are aborted and again started, otherwise the transactions are committed. So there is no situation of deadlock like in the case of locks.

The execution speed of the codes when this approach is used is same as when only the critical section is enclosed with STM calls or lock calls. However since in this approach users do not have to identify the critical section explicitly the total time taken is less. Also the user does not need to bother about synchronization at all and can concentrate entirely on the algorithm of the problem. So it can also be said that this approach proves that STM is more user-friendly than locks.

IV. READERS-WRITERS PROBLEM USING FLEXIBLE PROGRAMMING APPROACH

In the Readers/Writers Problem an object (a buffer which can be a global array) is shared among many threads which can perform either read or write operation on the object. Readers (Reader threads) read data, but never modify the data. However Writers (Writer threads) can both read data and modify them. So multiple reads may be performed simultaneously. But only one write operation can be

performed at a time to ensure consistency. In the Readers/Writers Problem there are multiple readers and writers accessing the elements in the same buffer at the same time. The buffer is of fixed size. We have taken the buffer size as 100000000. The problem is to synchronize these accesses properly so that when a write operation is occurring it should not be affected by any other read or write operation. [6]

In the parallel program using threads and STM using Flexible Programming approach which solves the Readers-Writers problem there are two processes, reader and writer whose functions are respectively, as the names suggest. The array is divided into several parts depending on the value of NUM_THREADS and the reader/writer pair accesses the corresponding part of the array. The reader thread is invoked using the thread ID which is passed to it as the parameter num_ptr. Based on this parameter, each thread accesses the corresponding part of the array. The writer process works in a similar manner. The read operation in the reader thread and the write operation in the writer thread are the critical sections. But in the flexible approach there is no need to identify the critical sections. Read and write operations are performed using appropriate STM calls (stm_unit_load( ) for read and stm_unit_store( ) for write) and then STM by its internal constructs ensures synchronization. In the program reader and writer are the two thread processes for reading and writing elements from the buffer respectively. The array arr is the buffer. The global array rcount keeps track of the position of elements in the buffer.

In the thread **reader**, elements are read from the buffer by the following statements.

```
for((k=(((num1*ARRAY_SIZE)/(NUM_THREADS))));k<(((n
um1+1)*ARRAY_SIZE)/(NU
M_THREADS))/2;k++)
  {
     byte_under_stm1=(unsigned char)
UNITLOAD(&rcount);
     printf("The data read is %d\n",arr[rcount[num1]]);
     UNITSTORE(&rcount,byte_under_stm1);
     gettimeofday(&ini_tv,NULL);
}
```

In the thread **writer** elements are written into the buffer by the following statements.

```
for(k=((((num)*ARRAY_SIZE)/(NUM_THREADS)));k<((nu
m+1)*ARRAY_SIZE)/(NUM_
THREADS)/2;k++,ki++)
  {
     byte_under_stm1=(unsigned char)
UNITLOAD(&rcount);
     arr[rcount[num]]=1;
     printf("The data written is %d\n",arr[rcount[num]]);
     rcount[num]++;
     UNITSTORE(&rcount,byte_under_stm1);
}
```

The following statement is used to record the time before the threads are created:

```
gettimeofday(&ini_tv,NULL);
```

The same call is also used to record the time when all threads have just finished their executions.

The total time taken is then calculated and printed using the following statement:

```
printf("Total Time Taken = %ld\n", final_tv.tv_sec -
ini_tv.tv_sec);
```

Some STM calls which have been used in the program are shown below.

stm_unit_load( ) reads the specified memory location outside of the context of any transaction and returns its value. The operation behaves as if executed in the context of a dedicated transaction (i.e., it executes atomically and in isolation) that never aborts, but may get delayed.

stm_unit_store( ) writes a value to the specified memory location outside of the context of any transaction. It also behaves as if executed in the context of a dedicated transaction (i.e., it executes atomically and in isolation) that never aborts, but may get delayed.

V. EXPERIMENTAL RESULTS

The experimental results of the code with threads and STM using flexible approach for solving the Readers-Writers Problem are shown in Table 1.

Table 1. Experimental Results for Readers-Writers Problem using Flexible Programming Approach

Number of Reader-Writer Pairs	Time Taken(seconds)
1	170
2	121
3	73
4	58
5	43
6	36
7	30
8	24
9	23
10	21
11	20
12	15

The execution time is same as that of the parallel programs which were used to solve the Readers-Writers Problem using locks and STM (without the flexible approach). But as programmers do not have to identify the critical section explicitly the total time taken is less.

VI. SYSTEM SPECIFICATIONS

The codes were compiled and executed in a server called PARAM Shavak server which has been developed by CDAC and is currently present in Jadavpur University. **[7]** The specifications of the system are given below:

SYSTEM DESCRIPTION

1. Hardware Configuration

Number of CPU cores: 12
Total Memory Space: 64 GB

2. Operating System

CentOS release 6.9(final)

3. Software Configuration

1) The language used in the programs is C.
2) gcc compiler version 4.4.7

VII. CONCLUSION

In this paper we showed the Flexible Approach to improve the performance of STM by taking the example of solving the Readers-Writers Problem. In this approach there is no need for programmers to identify critical sections explicitly. While performing read or write operations programmers need to use appropriate STM calls. Then STM by its internal constructs ensures synchronization in the program. Thus this approach is more user-friendly as programmers can concentrate on the algorithm of the problem only without thinking about ensuring synchronization. The execution time using this approach is same as that of the programs in which locks or STM (without the flexible approach) are used. But the total time taken is less as programmers do not need to identify the critical sections explicitly.

REFERENCES

[1] Yang Ni, Vijay Menon, Richard L. Hudson, Ali-Reza Adl-Tabatabai, J. Eliot, B. Moss, Bratin Saha, Antony L. Hosking, Tatiana Shpeisman,"Open Nesting in Software Transactional Memory", In the Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 68-78, 2007

[2] Zhengyu He, Bo Hong, "Impact of Early Abort Mechanisms on Lock-Based Software Transactional Memory", In the Proceedings of International Conference on High Performance Computing (HiPC), 2009

[3] Yossi Lev, Victor Luchangco, Virendra J. Marathe, Mark Moir, Dan Nussbaum, Marek Olszewski, "Anatomy of a Scalable Software Transactional Memory", In the Proceedings of the 4th ACM SIGPLAN Workshop on Transactional Computing , 2009

[4] Justin E. Gottschlich, Manish Vachharajani, Jeremy G. Siek, "An Efficient Software Transactional Memory Using Commit-Time Invalidation", In the Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization , pp. 101-110, 2010

[5] Sandhya S.Mannarswamy, Ramaswamy Govindarajan, "Variable Granularity Access Tracking Scheme for Improving the Performance of Software Transactional Memory", In the Proceedings of International Conference on Parallel Architectures and Compilation Techniques, pp. 232-242, 2011

[6] Anupriya Chakraborty, Sourav Saha, Ryan Saptarshi Ray, Utpal Kumar Ray," Lock-Free Readers/Writers", International Journal of Computer Science Issues (IJCSI), ISSN (PRINT): 1694 – 0814, ISSN (ONLINE): 1694 – 0784, Volume- 10, Issue-4, No-2, pp. 180-186, 2013

[7] Sandeep Agrawal, Shweta Das, Manjunatha Valmiki, Sanjay Wandhekar, Prof. Rajat Moona, "A case for PARAM Shavak: Ready-to-use and affordable supercomputing solution", In the Proceedings of the International Conference on High Performance Computing & Simulation, pp. 396-401, 2017

[8] Ryan Saptarshi Ray, Parama Bhaumik, Utpal Kumar Ray," Hybrid Parallel Programming Using Locks and STM", International Journal of Computer Sciences and Engineering (IJCSE) E-ISSN:2347-2693, Volume- 5, Issue-10, pp. 185-192, 2017

[9] Anjum Mohd Aslam, Mantripatjit kaur," A Review on Energy Efficient techniques in Green cloud: Open Research Challenges and Issues", International Journal of Scientific Research in Computer Sciences and Engineering ISSN: 2320-7639, Volume-6, Issue-3, pp. 44-50, 2018

[10] S. Vimala, P. Uma, S. Senbagam," Adaptive Vector Quantization for Improved Coding Efficiency", International Journal of Scientific Research in Network Security and Communication ISSN: 2321-3256, Volume- 6, Issue-3, pp. 18-22, 2018

Authors Profile

Ryan Saptarshi Ray received the degree of B.E. in I.T. from School of Information Technology, West Bengal University of Technology, India in 2007. He received the degree of M.E. in Software Engineering from Jadavpur University, India in 2012. Currently he is PhD Scholar in the Department of Information Technology, Jadavpur University, India.
He was employed as Programmer Analyst from 2007 to 2009 in Cognizant Technology Solutions. He has published 3 papers in International Conferences, 13 papers in International Journals and also a book titled "Software Transactional Memory: An Alternative to Locks" by LAP LAMBERT ACADEMIC PUBLISHING, GERMANY in 2012 co-authored with Utpal Kumar Ray.

Parama Bhaumik received B.Sc Phy(Hons.), B.Tech and M.Tech in Computer Science & Engineering from Calcutta University, India in 1996,1999 and 2002 respectively. She has done her Ph.D in Engineering from Jadavpur University, India in 2009. Currently she is working as Associate Professor in the Department of Information Technology, Jadavpur University, India. She has more than 32 research publications in Journals of repute, Book chapters and International Conferences.

Utpal Kumar Ray received the degree of B.E. in Electronics and Telecommunication Engineering in 1984 from Jadavpur University, India and the degree of M.Tech in Elecrical Engineering from Indian Institute of Technology, Kanpur in 1986.
He was employed in different capacities in WIPRO INFOTECH LTD., Bangalore, India; WIPRO INFOTECH LTD., Bangalore, India, Client: TANDEM COMPUTERS, Austin, Texas, USA; HCL America, Sunnyvale, California, USA, Clent: HEWLETT PACKARD, Cupertino, California, USA; HCL Consulting, Gurgaon, India; HCL America, Sunnyvale, California, USA; RAVEL SOFTWARE INC., San Jose, California, USA; STRATUS COMPUTERS, San Jose, California, USA; AUSPEX SYSTEMS, Santa Clara, California, USA and Sun Micro System, Menlo Park, California, USA for varying periods of duration from 1986 to 2002. From 2003 he is working as Assistant Professor in the Department of Information Technology, Jadavpur University, India. He has published 23 papers in different conferences and journals. He has also published a book titled "Software Transactional Memory: An Alternative to Locks" by LAP LAMBERT ACADEMIC PUBLISHING, GERMANY in 2012 co-authored with Ryan Saptarshi Ray.

