2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies

Exploring the Emerging Applications for Transactional Memory

Jiaqi Zhang
Department of Computer Science and Technology, Tsinghua University
Zhang-jq06@mails.tsinghua.edu.cn

Wenguang Chen
Department of Computer Science and Technology, Tsinghua University
cwg@tsinghua.edu.cn

Xinmin Tian
Intel Cooperation
xinmin.tian@intel.com

Weimin Zheng
Department of Computer Science and Technology, Tsinghua University
zwm-dcs@tsinghua.edu.cn

ABSTRACT

With the dominance of multicore processors, parallel programming has become more important. Transactional Memory is a promising solution to synchronation issues that are hurting parallel programmers. While there are a lot of researches on the implementation tradeoffs of TM, there is rare study on the applications that may utilize the techniques, which is essential to both providing feedbacks to the TM designers and to helping potential users.

This paper makes the first step of this work by presenting our identification of emerging applications for the comprehensive study of TM. The selection is based on application-domains including popular server/client softwares, multimedia applications, bioinformatics applications, data mining applications, and other scientific applications, which cover most of the dwarfs. A preliminary experiment is also provided to illustrate what we can get from this work.

Keywords

Transactional Memory, Application, Category

1. INTRODUCTION

As the mainstream vendors are all contributing themselves to the development of multi-core processors, the era of monoprocessors is gone together with the free lunch of improving the performance by merely increasing the clock rate. Concurrent programming and parallelism are destined to be introduced to emerging software designs. A traditional and typical way of programming multithreads is to use locks to perform the synchronization. However, controlling the locks could be tricky. While coarse-grained locks introduce performance issues, fine-grained locks are error-prone and greatly complicate the programs. In the meanwhile, both of them are not composable and hard to maintain.

Transactional Memory (TM), implemented in both hardware and software, is proposed as an alternative way to break this situation. The target of TM is to shift the burden of managing synchronization issues from the programmers to the underlying runtime systems without severely hurting the performance. Previous researches on the implementation of both STM and HTM show that there are good opportunities to achieve this goal [1-7]. However, most TM research projects are based on the trivial or small workloads and data structures, which are not sufficient to justify the effectiveness of TM technology and its stack that can enable a wide range of applications for the state-of-art multicore processor. As a new programming paradigm, there are many open issues to be addressed. For instance, are existing STM language extensions sufficient to deploy STM technology in real large multithreaded application? What are the key aspects to bring the application performance and scalability using TM to the level of performance and scalability using fine-grained locks in application? What are general methodology of debugging and tuning the TM applications? Are there enough composable components that could be utilized by the users? These issues, while certainly essential, are not effectively solved due to lack of handy runtime support to TM programming.

To seek the answers to these questions, we have identified a suite of emerging and complex software from different application domains. With the recently released prototype version of Intel C++ STM compiler, we are planning to reimplement, at least, the synchronization parts of these softwares and give rich feedback to the TM designers, as well as experiences and possible help to incoming TM programmers. Our goals are:

- Perform a comprehensive study of a wide application domain to identify a set of real applications plus several benchmarks and develop TM version of these real applications using Intel® C++ STM Compiler Prototype Edition [31]
- Conduct a comprehensive study of performance and programmability issues based on our TM workloads to compare with using coarse-grained locks and fine-grained locks
- Perform an application domain-specific analysis on the transactional behavior (such as nesting properties, I/O operations, read/write set sizes, and transaction length, etc.) of potential TM applications
- Develop new methods to identify and tune TM application performance issues.

This paper presents the first step of our work: the identification of applications that are suitable for the study.

978-0-7695-3443-5/08 $25.00 © 2008 IEEE
DOI 10.1109/PDCAT.2008.77

2. RELATED WORK

Regarding the design of TM, [8] has done a comprehensive survey, and this part will not be the focus of this paper. As applications are necessities to evaluate any TM implementation and to provide feedback to TM developers to make further improvements, there are several works that try to provide or to analysis transactional implementation of programs and benchmarks. According to the complexity of the software these works focus, they can be divided into 3 categories: simple data structures, single application, and complicated application suites.

2.1 Simple Data Structure

The use of single data structures to test STM implementations is widely applied even from the occurrence of STM systems. And most of the relatively early uses of these micro benchmarks are covered in the survey of [8]. And it is also used in a lot of recent researches that are not covered in the survey [1, 2].

2.2 Single Application

[9] targets on the workload of SPECjbb2000, which is a benchmark for JAVA middleware. By exploring the parallelism within a single warehouse, the authors revealed the advantages in performance of transactional memory over coarse grained locks. Although the parallelism mainly focuses on the access of a B-tree structure which presents all the stuffs in a warehouse, the 3-tier architecture of SPECjbb2000 makes this application significantly more complicated than single data structures.

[10] provides one of the earliest suggestions on how to think of general purpose applications in a TM view. The author found while Lee's algorithm on routing can be hardly parallelized by locks, it could be solved in a transactional manner. Similarly, [11] also presents how to apply the idea of transactional memory to real programs, and provides feedback to the implementations of STM in the view of a user.

[12, 13] presents a variant of Linux implemented using several self-defined HTM primitives. The authors implemented a set of HTM semantics, introduced a way to cooperate between locks and transactions, and integrated transactions with the OS scheduler. This work provides how a legacy OS could utilize TM and its performance. Although it basically focuses on HTM, it could bring valuable experiences in the development of STM.

2.3 Complicated Application Suite

[14] studies the support for TM runtime from different aspects of the whole software stack including data structures, OS, language, and so on. The paper focuses on the description of major problems in each field (such as IO in OS, language features), but doesn't provide experiences from real applications.

[1] has implemented 5 benchmarks including *genome*, *vacation*, *k-means*, *lybrinth*, and *bayes*, together with several data structures to evaluate their implementation of a hybrid TM. However, the applications they used are relatively small and is hardly representative.

[15] presents a benchmark derived from OO7. It by itself is not a real world application of any kind. It tries to produce a benchmark that could simulate different access patterns of TM applications. The idea is to create a complex shared data structure and perform different operations to access it concurrently and in different patterns. STMBench7 has generalized four categories with a total of 45 operations to constitute a dedicated full testcase. However, while the benchmark provides an alternative way to test the general performance of an STM implementation, it is difficult to simulate varied application domains since it relies on the users to generalize the access pattern of their programs, which may costs a lot of human resource and is error-prone as well. Besides, the data structure it uses is regular and thus could not be representative to certain workloads.

[16] performes analysis on the transactional behavior of up to 35 multithreaded applications by lexically replacing all the lock primitives to corresponding TM marks. The analysis focuses on the transactional length, read-set and write-set sizes, and the frequency of nesting and I/O operations. The work groups the applications by the programming models the programs adopt (JAVA, POSIX, OpenMP, ANL Macros). However, as the work suggests, it focuses on the code analysis and transactions are not executed in reality. Thus it cannot analyze certian transaction behaviors such as retries, which are identical in the performance analysis.

[35] presents the work of identifications of bugs in popular concurrent software. The work analyzes these bugs and describes how TM could help avoid them, and at the same time proposes suggestions on how future TM implementations should be designed to help programmers. However, this work focuses only on the concurrency bugs instead of the whole design of applications.

[17] presents our closest relative comprehensive study on TM practice. The paper implements a Java based software transactional memory runtime and a suite of corresponding Java benchmarks including SPECjbb2000, DaCapo hsqldb, STMbench7, and a set of micros. However, they are all implemented in JAVA, which can be significantly different from C/C++ in both performance and programming model. Also, mainstream databases and servers are mostly implemented in C/C++.

3. APPLICATIONS OVERVIEW

The categories and their related programs are listed in Table 1. It presents the short application description together with their lines of code (LOC) and possible related dwarfs[29]. In our study, we generalized 5 categories of emerging applications. That is popular softwares, multimedia applications, bioinformatics supplications, data mining programs, and other scientific programs. Except for the popular softwares, the other categories are selected basically because they are becoming increasingly important and essential to our lives, and they are also consuming more and more computing power. We are also trying to cover as many as possible the dwarfs described in [29]. As a result, we find the applications relate to approximately 11 out of the 13 dwarfs. Note however, the estimated mapping of the applications to their related dwarfs is based on the lexical search of the application properties on Figure6 of [29], and the real mapping may depend on the application implementation. For example, the SVM may relate to Dense Linear Algebra or Sparse Linear Algebra depending on its implementation. Thus an exact mapping can only be performed after solid understanding and implementation of the application. Also, it is very possible that the applications we marked N/A in the dwarf column turn out to be related to certain dwarfs. The applications also cover the three key aspects of the RMS applications

Table 1: The identified 23 applications that will be used in our study

Application	Source	LOC	Description	Related Dwarf(s)
Popular Software and Tools				
STL	GNU GCC	70K	The C++ Standard Template Library	N/A
MCSTL	GNU GCC	15K	The multi-core ready C++ Standard Template Library	N/A
Apache HTTP Server	Apache	270K	An Apache implementation of HTTP web server	Finite State Machine
FastDB	FastDB	31K	A Main Memory Relational Database Management System	Combinational Logic/ Dynamic Programming
FileZilla Server	FileZilla	50K	A popular FTP server provided by Mozilla	Finite State Machine
Multi-core MapReduce	Stanford Phoenix	1.5K	The implementation of MapReduce on CMP platform provided by Stanford	Monte Carlo
Multimedia Applications				
Sphinx 3	ALPbench	30K	An application for speech recognition	N/A
RayTrace	Tachyon	10K	Tachyon ray tracer that renders 3-D scene	Monte Carlo
MPEG-2 Encoder/Decoder	ALPbench	28K	Standard video Encoder/Decoder	Dense Linear Algebra/Structured Grids/Finite State Machine
Bioinfomatics				
ClustalW_smp	BioPerf	26.7K	Multiple sequence alignment for nucleotides acids	N/A
Hmmer	BioPerf	4K	Hidden Markov Models for aligning multiple sequences	Graphical Models/ Sparse Linear Algebra
FASTA	BioPerf	48K	Local similarity search programs	N/A
GRAPPA	BioPerf	21K	Genome Rearrangements Analysis under Parsimony and other Phylogenetic Algorithms	N/A
MST	GaTech	2.9K	Minimium Spanning Forest	Combinational Logic
Data Mining				
ScalParC	MineBench	2K	Decision tree classification	Graph Traversal
k-Means	MineBench	1.3K	Mean-based data portioning method	Dense Linear Algebra
SVM-RFE	MineBench	4.5K	Support Vector Machines - Recursive Feature Elimination	Dense/Sparse Linear Algebra
SEMPHY	MineBench	20K	Structure learning algorithm based on phylogenetic trees	N/A
Scientific programs				
FFT	FFTW	6K	3D Fast Fourier Transformation	FFT
Art	SPEComp	2K	Image Recognition/Neutral Network	N/A
Equake	SPEComp	1.6K	Earthquake Modeling	N/A
Barnes-Hut	SPLASH-2	3K	Simulates the interactions in a system of N bodies	N-Body
SSCAv2	GaTech	1.7K	DARPA graph theory benchmark	Graph Traversal

proposed by Intel [36]. For example, Sphinx in the multimedia applications and SVM-RFE in the data mining applications are related to Recognition, most of the data mining applications are related to Mining, and the popular softwares, Barnes-Hut, and Equake are related to Synthesis [29].

We have also considered the implementation complexity. As will be suggested in Section 4, we assume it takes 1 developer 3 hours to convert 1K lines of code. As there are totally about 635K lines, ideally, it require effort from 3 developers over the course of about 80 days. And the time cost is acceptable.

When dealing with applications of specific field, we tend to rely on the latest emerging benchmarks in that field, because we believe that the experts know more about their own area then anyone else. The detailed and exact explanations of each application can be found on their corresponding benchmarks. This section will

only make a brief introduction to them and include our reasons of selection.

3.1 Popular Software and Tools

Popular software and tools here refer to the applications that are targeted on popular purposes and are publicly available. They are usually complex and in large scales. The purpose of selecting large software is in three fold:

- To provide available transactional components and software for public use;
- To find the difficulties and problems in using transactional memory as a synchronization mechanism to implement large scale software;
- To test the performance of STM implementations in real world.

C++ Standard Template Library is a widely used component in numerous softwares developed in C++. However, its implementation is not compatible with current STM solutions. Although Intel C++ STM compiler has provided some kind of mechanisms to automatically apply transactional semantics to legacy codes, it is necessary to provide a hand tuned tm-ready version of STL given its importance. Also it is not recommended to rely on the compiler to perform the analysis in such complicated and crucial environment. A tm-ready version of STL should readily be used with guaranteed thread-safety and transactional semantics in any parallel programs that utilize transactional memory or not.

GNU GCC 4.3 has integrated Multi-Core STL[18], which is a partly parallelized version of STL. It aims to accelerate the performance of programs even without the programmers' knowledge on parallel programming. Since it can be seamlessly integrated to legacy codes that utilize STL, there are various opportunities to compare the transactional memory with locking mechanism on performance issues.

Apache HTTP Server[19] and FileZilla FTP Server[20] are two popular web/FTP servers that are servicing millions of people every day. FastDB[21] is a well-known in-memory database management system, which is the origin of transactional memory.

MapReduce[22] is first proposed and implemented by Google on clusters, and latterly implemented on CMP platforms as Phoenix[23] by Stanford. It provides an alternative and promising abstract of concurrent programming style and is effectively applied to a variety of applications.

Except C++STL, all the other selected software are heavily threaded and thus may easily expose synchronization problems and provide potential to exhibit the advantages and shortcomings of each TM implementation.

3.2 Multimedia Applications

With the increasing computing power of processors and the demanding from both entertainment and industry, multimedia applications become an important workload. A significant feature of current multimedia applications is their ever increasing complexity. In addition, more and more applications, such as 3-D games and video playbacks, require real-time processing of multimedia data. At the same time, most multimedia applications show significant parallelism, such as processing every word in a sentence in parallel or rendering different key frames simultaneously.

ALPBench[24] provides a suite of emerging complex multimedia applications that are parallelized in 3 levels (thread-, data-, and instruction-level). In our study, however, we only make use of the thread level parallelism.

CMU Sphinx 3 is a speech recognizer, which identifies each word in a speech and converts them into texts. Speech recognizer becomes more and more widely used particularly in the field of Human Computer Interface and security, and most emerging OSes are utilizing them as fundamental functionality (e.g. Windows Vista, Symbian).

A ray tracer is used to render a scene given a scene description. The input, which is a scene description, usually contains the information of the objects, the viewers, and light such as location and shapes. And the output is the rendered scene. This technology is widely used in the area of 3-D games, virtual reality and modeling.

MPEG-2 Encoder and Decoder are the applications to convert video frames into compressed MPEG-2 bit-stream and to convert it back. The technology is essential to almost all the video related applications including their recording, editing and playback. Currently there are several new video encodec standard such as MPEG-4 and H.264. However, their algorithms are essentially similar to that of MPEG-2.

Note that the application of FaceRec, which recognize faces from pictures, is abandoned in our study because it exposes rare inferences between threads and thus is not suitable to exhibit the features of TM.

3.3 Bioinformatics Workload

Bioinformatics technology has become an important research field as genetic data is growing exponentially. It has shown its necessities in numerous critical industries such as medicine, agriculture, and environment. Bioinformatics technology could help researchers identify interesting patterns or extract useful information from long sequence of genome. As these applications have become one of the largest computing power consumers, it is necessary to observe their parallelism.

There are several benchmarks available on Bioinformatics, BioPerf[25], BioBench[26], and BioInfoMark[27], to name a few. BioPerf is a most recent one among them, and has parallelized 4 out of 10 of the applications. To demonstrate the features of TM, we have adopted the 4 parallelized applications.

ClustalW_smp is a parallelized version of ClustalW, which takes multiple DNA and protein sequences as input, align them based on their ancestral relationships, and output the results.

Hmmer aligns multiple sequences by the means of Hidden Markov Model. It is constituted of several small applications. Among these applications, we are only adopting the threaded Hmmpfam, which search the transcriptional regulatory protein.

As Blast, FASTA also does pairwise local alignment. However, their algorithms are essentially different. It is constituted of a set of small applications, which perform similarity search for sequence databases.

GRAPPA stands for Genome Rearrangements Analysis under Parsimony and other Phylogenetic Algorithms, and is used to reconstruct phylogeny. It provides the first linear-time implementation of inversion distances improving upon the original polynomial time approach.

MST is a combinational problem widely used in various fields, including, but not limited to, medical imaging, proteomics and cancer detection. The GaTech parallel implementation[34] of MST achieves good speedup over a wide range of input graphs.

3.4 Data Mining Applications

Data mining is the technique to help researchers to make discoveries of potentially useful information in a large amount of data. It is originally only limited to the fields of scientific research and medicine. However, with the exponentially growing size of information especially on the Internet, it also becomes essential to increasingly more fields such as AI, web developing, and even HPC. Since the sizes of the target data sets are usually extremely large, data mining process often costs large amount of computing power. These features make data mining workloads suitable applications to illustrate the computing power of current multicore platforms.

As an emerging research domain, there are not many alternative applications to select. And the only benchmark suite by far is MineBench[28]. It broadly classifies the applications of data mining into several categories and selects 5 top categories based on how commonly they are and will be used in the industry. This benchmark is also suitable for our study because all the applications are full-fledged implementations and parallelized as well. We have randomly selected 4 applications, each from a distinct category.

ScalParC comes from the category of classification. It is an efficient and scalable variation of decision tree classification, which is faster than other classification method on high-performance data mining. The general process is to recursively splitting the training dataset based on an optimality criterion until all records belonging to each of the partitions bear the same class label.

K-means belongs to the category of clustering. It is a well-known algorithm that is used to converge all the objects to k (user-specified) centers by iteratively applying the similarity function to each object and the k cluster centers generated in the last iteration.

SVM-RFE stands for Support Vector Machines – Recursive Feature Elimination, which is a feature selection method. It achieves the selection by recursive feature elimination process. The application is widely used in gene expression. Note that the scalability of SVM-RFE is one of the worst one among other applications in the benchmark due to locking of memory structures, it is ideal to show the performance changes TM could bring.

SEMPHY is a structure learning algorithm that is based on phylogenetic trees, which represent the genetic relationship of species. It finds the best tree topology and branch lengths representing the distance between two neighbors by means of probability estimation algorithm.

3.5 Other Scientific Applications

Except the workloads in specific domains been identified, we are trying to seek more suitable applications in other scientific area. The general guideline is to cover more dwarfs that are present in [29]. For example, FFT from FFTW[32] is identified as spectral method, and Barns-Hut is a representative program of the n-body method. SSCAv2[33] is a graph algorithm benchmark suite that contains 4 kernels (graph generation, large sets classification, subgraphs extraction, and graph clustering). Also, it is expected to include some SPEC benchmarks such as those from SPEComp[30], which are ideal applications to present the effectiveness of parallelism.

4. PRELIMINARY EXPERIMENTS

Currently, we have only observed and converted Phoenix. The experiment analysis is based on both programmability issue and performance.

4.1 TM Interfaces of Intel C++ STM Compiler

Intel C++ STM compiler assists the development of TM programs by providing simple language extensions that present TM semantics. The extensions basically serve two purposes: mark the atomic block, and mark functions that are called inside atomic blocks.

To mark a code block as an atomic block (e.g. a transaction), the extended keyword *__tm_atomic* is used. All the instructions (currently except irrevocable code such as I/O operations) inside the *__tm_atomic* marked block are executed in transaction. However, to apply the transactional semantics to the functions that are called inside a *__tm_atomic* block, one should mark either *tm_callable* or *tm_pure* on the function definition. The former one declares that there are accesses to shared variables in the functions, and thus the compiler generates transactional accesses to them automatically. On the other hand, the compiler does not apply any transactional semantics to a function marked with *tm_pure*, and it should be guaranteed by the programmers that these functions do not access shared variables.

More detailed descriptions of the compiler can be found in the manual on [31].

4.2 Implementation Method

The general converting process of Phoenix turns out to be fairly easy and straightforward given the language extensions provided by the compiler. The basic process is simply converting locking statements to TM marks (e.g. __tm_atomic), and annotating certain routines to be one of the TM attributes (e.g. __tm_callable). However, the memory manipulation in the critical regions is a problem. While we can implement transactional *memset*, implementing transactional memory *allocation/free* is not trivial, and cannot be easily solved by the compiler of version 1.0. Users have to deploy their own memory allocation policy in order to utilize the TM semantics. Luckily the prototype 2.0 has provided transactional version of memory *allocation/free* operations. In general, except for reimplementation of *memset*, only 10 out of 1500 lines of code are modified. Also there are slight modifications in some of the user programs.

The performance test of TM_Phoenix is carried out on a 2-way 4-core Xeon platform. The transactional behavior of each program is collected by setting ITM_STATISTICS to "*verbose*".

4.3 Experimental Results

Excluding the memory operation issues, the converting process takes 1 developer 2 hours. That is, about 750 lines of code per hour. Since there may be unforeseeable problems in the future development, and we are not familiar with applications from other domains, we make a conservative estimate that it will cost 3 hours per developer to process 1K lines. This estimate also agrees with the result in [17].

From the implementation process discussed above, we've learned two issues related to the interfaces that should be provided by a TM solution targeted on industry use:

- Scalable memory allocation and deallocation operations should be provided by the runtime.
- Certain libraries (such as *memset*) are expected to be provided to ease the programming and make use of the composability of TM.

Table2: speedup of WordCount

procs	P	TP
serial	1	1
2	1.979	1.953
4	4.073	4.009
8	5.242	5.281

Table3: speedup of Histogram

procs	P	TP
serial	1	1
2	1.19	1.1914
4	1.7957	1.793
8	1.913	1.9262

Table4: speedup of ReverseIndex

procs	P	TP
serial	1	1
2	5.049	4.323
4	12.187	10.433
8	21.589	19.125

Table5: speedup of Kmeans

procs	P	TP
serial	1	1
2	2.727	2.457
4	4.691	4.716
8	7.816	7.871

Table2, 3, 4, and 5 show the speedup of the converted Phoenix (TM_Phoenix). The programs and datasets are all provided by the Phoenix package. From the speedup, we find that the TM_Phoenix shows the same scalability as the original one. While the TM versions tend to invoke some overhead, there is no significant difference. This is due to the relatively short critical regions in both the system and the user programs, which can also be demonstrated by the following statistics.

Table 6, 7, 8, and 9 present the transactional behavior of each program. It includes the number of transactions (Txns), the number of retries (Retries), and how many bytes do they read/write in transaction. From the statistics, we could easily find that the number of retries grow significantly with the number of threads. For ReverseIndex, when we are running 8 threads, there are 24 times

Table6: transactional behavior of WordCount

procs	Txns	Retries	BytesRead		BytesWritten	
			Mean	Total	Mean	Total
2	687	0	49.94	34311	18.83	12936
4	695	4	49.7	34539	18.63	12948
8	711	8	49.14	34939	18.22	12956

Table7: transactional behavior of Histogram

procs	Txns	Retries	BytesRead		BytesWritten	
			Mean	Total	Mean	Total
2	21718	29	59.61	1294636	19.86	431304
4	21722	271	59.93	1301908	19.9	432272
8	21730	491	60.23	1308744	19.93	433148

Table8: transactional behavior of ReverseIndex

procs	Txns	Retries	BytesRead		BytesWritten	
			Mean	Total	Mean	Total
2	6792	41	170.3	1156680	104.17	707556
4	6796	177	170.6	1159660	104.16	707868
8	6804	974	172.9	1176400	104.2	708992

Table9: transactional behavior of Kmeans

procs	Txns	Retries	BytesRead		BytesWritten	
			Mean	Total	Mean	Total
2	12432	24	43.87	545432	21.2	263520
4	12544	79	43.76	548908	21.04	263884
8	12768	197	43.49	555332	20.7	264292

of the number of retries than that of 2 threads. This result indicates two concerns that relate to the scalability of TM programs:

- For the TM runtime designers, they should pay attention to the behavior of the runtime on transaction aborts. Although the percentage of retries is not so significant now (a maximum of 0.143%), it may grow very fast when we are running on more and more cores using numerous threads.
- For the programmers, they should also realize that the possibility of retries is increasing rapidly when there are more threads, and they should put efforts in minimizing the overhead of abort, that is, decreasing the memory accesses in the transactions. The Phoenix programs we observe serve good examples. Despite the increasing number of retries, they are scaling well because they invoke rare memory accesses (see the "mean" column of the BytesRead/BytesWrite, which indicate how many bytes does a transaction access on average).

Note however, although reasonable, the results here are still not representative to all the applications from various domains. A comprehensive study can only be done with all the applications rewritten in TM.

5. CONCLUSIONS AND FUTURE WORK

This paper presents our identification of emerging applications for transactional memory. The applications are collected from various different application domains including popular softwares, multimedia applications, bioinformatics applications, data mining applications, and several other scientific applications.

Preliminary results show that with user-friendly TM interfaces, programming in a TM model can be convenient. And the performance is not significantly hurt. However, there are still a lot of works to do to ease the programming, for example, the library support and memory allocation. And there are also issues that we should pay attention to in order to guarantee the scalability.

Our future work includes implementing all the identified applications in TM, and identifying potential issues and making suggestions on the further TM runtime for industry use. We will also perform a comprehensive study on the various transactional behavior of these representative applications based on our implementation.

REFERENCES

[1] C. Minh, M. Trautmann, J. Chung, A. McDonald, N. Bronson, J. Casper, C. Kozyrakis, and K. Olukotun. An effective hybrid transactional memory system with strong isolation guarantees. In *Proceedings of the 34th annual International Symosium on Computer Architecture (ISCA)*, pages 69-80, 2007

[2] Y. Ni, V. Menon, A. Adl-Tabatabai, A. Hosking, R. Hudson, J. Moss, B. Saha, and T. Shpeisman. Open nesting in software transactional memory. In *Proceedings of the 12th ACM SIGPLAN symposium on Principles and Practics of Parallel Programming (PPoPP)*, pages 68-78, 2007.

[3] D. Dice, O. Shalev, and N. Shavit. Transactional Locking II. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC)*, 2006

[4] K. Fraser, Practical lock freedom. Cambridge, UK: University of Cambridge Computer Laboratory, 2004, p. 116 (http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-579.pdf)

[5] W. Scherer, M. Scott, Advanced contention management for dynamic software transactional memory. In *Proceedings of the 24th annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 240-248, 2005.

[6] A. McDonald, J. Chung, B. Carlstrom, C. Minh, H. Chafi, C. Kozyrakis, and K. Olukotun. Architectural semantics for practical transactional memory. *ACM SIGARCH Computer Architecture News*, pages 53-65, Volume 34, issue 2, 2006

[7] K. Moore, J. Bobba, M. Moravan, M. Hill, and D. Wood. LogTM: log-based transactional memory. In *Proceedings of the 20th international symposium on High-Performance Computer Architecture (HPCA)*, pages 254-265, Feb, 2006

[8] Ravi Rajwar and James Larus. Transactional Memory – Morgan Claypool, 2006.

[9] J. Chung, C. Minh, B. Carlstrom, and C. Kozyrakis. Parallelizing SPECjbb2000 with transactional memory. "Workshop on Transactional Memory Workloads", 2006

[10] I. Watson, C. Kirkham, and M. Lujan. A study of a transactional parallel routing algorithm. In *Proceedings of the 16th international conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 388-398, 2007

[11] M. Kulkarni, L. Chew, and K. Pingali. Using transactions in delaunay mesh generation. "Workshop on Transactional Memory Workloads", 2006.

[12] C. Rossbach, O. Hofmann, D. Porter, H. Ramadan, B. Aditya, and E. Witchel. TxLinux: using and managing hardware transactional memory in an operating system. In *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 87-102, 2007.

[13] H. Ramadan, C. Rossbach, D. Porter, O. Hofmann, A. Bhandari, and E. Witchel. Evaluating transactional memory tradeoffs with TxLinux. In *ISCA*, 2007.

[14] B.Carlstrom, J. Chung, C. Kozyrakis, and K. Olukotun. The software stack for transactional memory. First Workshop on Tools for Multicore Systems (STMCS), Mar, 2006.

[15] R. Guerraoui, M. Kapalka, and J. Vitek. STMBench7: a benchmark for software transactional memory. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 315-324, 2007.

[16] J. Chung, H. Chafi, C. Minh, A. McDonald, B. Carlstrom, C. Kozyrakis, and K. Olukotun. The common case transactional behavior of multithreaded programs. In *Proceedings of the 20th international symposium on High-Performance Computer Architecture (HPCA)*, pages 266-277, 2006

[17] E. Brevnov, Y. Dolgov, B. Kuznetsov, D. Yershov, V. Shakin, D. Chen, V. Menon, and S. Srinivas. Practical experience with Java software transactional memory. In Proceedings of the 13th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP), pages 287-288, 2008

[18] J. Singler, P. Sanders, and F. Putz. MCSTL: the multi-cores standard template library. In *Proceedings of the 12th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 144-145, 2007

[19] The Apache HTTP Server Project. http://httpd.apache.org/.

[20] The FileZilla FTP solution. http://filezilla-project.org/.

[21] FastDB: The Main Memory Relational Database Management System. http://www.ispras.ru/~knizhnik/fastdb.html

[22] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on symposium on Operating System Design & Implementation (OSDI)*, pages 10-23, 2004

[23] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating MapReduce for multi-core and multiprocessor systems. In *Proceedings of the 13th international symposium on High-Performance Computer Architecture (HPCA)*, pages 13-24, 2007.

[24] M. Li, R. Sasanka, S. Adve, Y. Chen, and E. Debes. The ALPBench benchmark suite for complex multimedia applications. In *Proceedings of the IEEE international symposium on Workload Characterization (IISWC)*, pages 34-45, 2005.

[25] D. Bader, Y. Li, T. Li, and V. Sachdeva. BioPerf: A benchmark suite to evaluate high-performance computer architecture on bioinformatics applications. In *Proceedings of the IEEE international symposium on Workload Characterization (IISWC)*, 2005

[26] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C. Tseng, and D. Yeung. BioBench: A benchmark suite of bioinformatics applications. In *Proceedings of the 5th International Symposium on Performance Analysisi of Systems and Software (ISPASS)*, Mar. 2005.

[27] Y. Li, T. Li, T. Kahveci, and J. Fortes. Workload characterization of bioinformatics applications. In *Proceedings of the 13th IEEE international symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 15-22, Spet. 2005.

[28] J. Zambreno, B. Özışıkyılmaz, G. Memik, A. Choudhary, and J. Pisharath. Performance Characterization of Data Mining Applications using MineBench. In *9th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-9)*, Austin, TX, February 2006.

[29] Asanovic, et al. The landscape of parallel computing research: a view from Berkeley. Dec. 2006. http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[30] The SPEC OpenMP benchmark Suite. http://www.spec.org/omp/.

[31] Intel C++ STM compiler. http://softwarecommunity.intel.com/articles/eng/1460.htm.

[32] FFTW. http://www.fftw.org

[33] D. Bader and K. Madduri. Design and implementation of the HPCS graph analysis benchmark on symmetric multiprocessors. In *Proceedings of the 12th international conference on High Performance Computing (HiPC)*, 2005

[34] D. Bader and G. Cong, Fast shared memory algorithms for computing the minimum spanning forest of sparse graphs. In *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2004

[35] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes- a comprehensive study on real world concurrency bug characteristics. In *Proceedings of the 13th international conference on architecture support for programming languages and operating system (ASPLOS)*, March, 2008.

[36] P. Dubey. Recognition, Mining and Synthesis Moves computers to the Era of Tera. *Technology@intel Magazine*, Feb. 2005.