On the Performance of Commit-Time-Locking Based Software Transactional Memory

Zhengyu He and Bo Hong
School of Electrical and Computer Engineering
Georgia Institute of Technology
zhengyu.he,bohong@gatech.edu

Abstract

Compared with lock-based synchronization techniques, Software Transactional Memory (STM) can significantly improve the programmability of multithreaded applications. Existing research results have demonstrated through experiments that current STM designs have slower execution speed than the locks. This paper develops a theoretical explanation for the performance difference. In particular, commit-time-locking (CTL) based STMs are analyzed. A queuing theory based statistical model is developed to quantify the performance of lock-based and STM-based schemes. Analytical results obtained from the model are validated by simulations. Our study shows that (1) lock-based synchronization outperforms CTL-based STMs, and (2) when the contention level becomes low, locks and CTL-based STMs exhibit similar performance. Furthermore, we show that the performance of CTL-based STMs is sensitive to the number of threads, transaction issue rate, and the bandwidth of interconnect. Our results are expected to be useful in the early stages of designing parallel programs, especially on the selection of design schemes for STMs.

1. Introduction

Recently, multicore computing has emerged as a necessary path to advance the performance of computer systems. Concurrency control mechanisms, which control accesses to shared resources, are likely to be the key for software to take full advantage of multicore computing.

The most frequently used concurrency control mechanism today is lock, which is typically implemented using hardware support of atomic instructions such as “compare-and-swap”, and “load-linked/store-conditional” [5]. Proper deployment of locks in multi-processor or multi-core systems may require quite complex hardware and/or software support and may lead to substantial synchronization issues. Programming using locks is also known to be extremely error-prone.

Transactional memory is a recently emerging concurrency control mechanism. It is considered to be a promising alternative to lock-based synchronization as it eliminates many of the programmability issues mentioned above.

The idea of providing hardware support for transactions originated in [11] and has since been explored [1], [3]. Software-only transactional memory has recently been the focus of intensive research, and support for practical implementations is growing [14], [9], [10]. Schemes that mix hardware and software have also been explored [15], [12]. STM implementations have evolved from lock-free designs to lock-based designs as the latter performs better. For lock-based STMs, two types of designs have been studied intensively: encounter-time locking (ETL) and commit-time locking (CTL).

Although lock-based STM designs exhibit improved performance over lock-free STM designs, in practice STMs still suffer a performance hit by as much as 50% relative to fine-grained locks [6]. A wide range of STM designs, both ETL and CTL based, have been proposed to reduce or bridge the performance gap between locks and STMs. These studies focus on the reduction of various STM overheads (e.g. rollback, logging, committing, etc.). Performance of these STM designs are measured through either simulations or actual executions, which have been validating the existence of the gap as in [6]. To the best of our knowledge, we are not aware of any theoretical explanations for the performance gap. Furthermore, the following intriguing question has not been addressed by the research community either: suppose STMs can be implemented with zero (or close to zero) overhead, can they outperform locks?

This paper aims to attack the above problems by analyzing STMs theoretically and comparing it against locks. Our studies focuses on CTL-based STMs. Other types of STMs, including ETL-based, will be addressed in our future studies. In this paper, statistical models are established for both lock-based and STM-based synchronization methods. Queuing models are developed to describe the behaviors of these two types of concurrency control methods. The overheads of locks and STM are considered in the model through the feedback on the execution rate at which the critical sections (transactions) are executed. Formal analytical results have been derived to evaluate the mean completion time of critical sections (transactions). The analytical results are validated through extensive experiments using the M5 multi-processor simulator.
Our results show that (1) locks always outperform CTL-based STM, and (2) locks are only marginally better when the contention level is low. The analysis further shows that the performance of CTL-based STMs is affected by the contention level, which in turn is affected by a list of factors including the number of threads, transaction issue rate and the bandwidth of interconnect. We expect the model to be useful in making early design and research decisions in STM systems, particularly in determining the applicability of STMs for a given work load and system platform.

The rest of the paper is organized as follows: Section 2 summarizes the background and related work. Section 3 describes the two models for locks and transactional memory. Section 4 presents the numerical analysis of the model and experimental results. Section 5 concludes the paper with discussions and suggestions for future research directions.

2. Background and Related Work

A lock consists of three components: acquisition, waiting algorithm, and release [5]. A variety of lock designs have been proposed, including Test&Set, Test&Set with back off, Load-Linked Store-Conditional, Ticket Lock, Array-based Lock, etc. Locks ensure mutual exclusion of the critical section with the following usages:

```
lock(L);
critical section();
unlock(L);
```

Only one thread can acquire the lock and consequently be active in the critical section. Other threads attempting to enter the critical section will be blocked at lock(L) executing the waiting algorithm, and will be un-blocked upon the release of the lock.

STM is a recently emerging concurrency control mechanism. In STM, a transaction completes modifications to shared memory regardless of what other threads might be doing. Reads and writes inside transactions should logically occur at a single instant in time. No intermediate states can be observed or interfered by peers. STM is typically utilized in the following form:

```
while (true) {
    STM_begin()
    try {
        critical section()
    } catch STM_invalid { continue }
    STM_commit()
    break
}
```

STM designs have evolved from lock-free to obstruction-free, and to lock-based. Ennals [7] is observed that non-blocking transactions are unnecessary for practical uses and demonstrated the superiority of lock-based STM through experiments. Ennals’ STM is also the first encounter-time-locking (ETL) based STM. Saha et al. [13] implemented their McRT-STM in a runtime system. Similar to [7], the scheme in [13] requests locks at encounter time and uses pessimistic reads and writes. A transaction always attempts to acquire exclusive ownership (via locks) before it reads a variable, failure to do so will cause it to abort and roll back. Write operations in [13] are handled similarly.

Dice et al. [6] implemented Transactional Locking II (TL2), the first commit-time locking (CTL) based STM scheme. TL2 then evolves to TL2C [2] with the adoption of a distributed clock. In both TL2 and TL2C, locks are acquired only during commit time and early abort is therefore not supported. Read and write operations in a transaction are recorded in a read/write-set. During commit time, all read/write locations are locked, and version numbers of the memory locations are checked to determine whether the transaction should commit or abort.

Recently, ETL is revisited by Felber[8] et al. A write-back strategy is introduced where the memory location is locked at encounter time but the updates are stored in a write-set and validated only during commit time. This is a hybrid ETL/CTL scheme that employs eager conflict detection to lower the cost of aborting. Nevertheless, experiments demonstrated similar performance between the write-through and write-back schemes.

On going research are still debating between ETL and CTL, though experiments have shown that CTL is able to withstand higher contention level than ETL[6], [8]. In this paper, we focus on CTL. ETL is reserved for our future studies.

3. Modeling of Locks and CTL-based STMs

In this section, we establish models for lock-based and CTL STM-based (CTL in short) synchronization methods. For notational convenience, we use ‘critical sections’ and ‘transactions’ interchangeably in our discussion. The semantics of the two terms are different, but in terms of completion time, both critical sections and transactions can be abstracted as an atomic entity and the notational abuse will not cause any confusions.

The execution of critical sections protected by locks is modeled as shown in Fig. 1(a). Without loss of generality, we assume that threads T1, T2, T3 and T4 enter their critical sections in the illustrated order. When T1 is executing its critical section, the other three threads attempt to acquire the lock, though none of them will succeed until T1 completes the critical section. Rate $l$ represents the lock-induced overhead traffic on the interconnect and its value is dependent on the quality of the lock implementation.

CTL-based STM protects critical sections in a different fashion as illustrated in Fig. 1 (b). When T1 commits its
transaction, T2 will not know immediately and will continue execute its current transaction. When T2’s transaction is about to commit, T2 realizes that T1 has modified the variables it recorded, and therefore rolls back its transaction immediately and re-executes it. Next, T2 can commit the transaction successfully but T3 has to rollback once more and so does T4.

We consider the following scenario in our analysis:

1) The system consists of \( N \) processor cores sharing the main memory, where each core executes a single thread. Every thread repeatedly issues critical sections/transactions with a Poisson distribution.

2) All the critical sections/transactions protect the code that access the same memory locations. We here focus on scenarios with only one type of critical section/transaction. Interleaved or nested transactions, or transactions with semi-overlapped read/write sets have complicated behaviors and we leave them for our future studies.

3) For each critical section/transaction, the time required to complete it is also a random variable with exponential distribution.

### 3.1. Modeling of Locks

![Figure 2. State transition diagram of traditional locking queuing system](image)

Lock-based system can be modeled as a finite source queuing system where the interconnect functions as the single server answering requests from the processors. The transition diagram is shown in Fig. 2, where state \( i \) means currently there are \( i \) threads attempting to execute the critical section.

The cumulative arrival rate of all critical sections \( \lambda_i \) depends on the number of threads in the queue:

\[
\lambda_i = \frac{N-i}{N} \lambda_0, \quad (i = 1, 2, 3, \ldots)
\]

where \( \lambda_0 \) is \( N \) times the arrival rate of an individual thread. The arrival rate of an individual thread depends on a variety of factors, including the workload in the transactions, the workload between the transactions, and the frequency of the processors. These factors are abstracted with the parameter \( \lambda_0 \). We also assume in this paper that memory latency, which affects the turn-around time for the processors to get one data element from the memory, is abstracted in \( \lambda_0 \). We will explicitly model the impact of memory latencies in our future study.

The interconnect, shared among the processors, is a major constraint affecting scalability. We characterize the interconnect with its bandwidth \( B \). The service rate also depends on the number of critical sections currently waiting in the queue, because they will keep trying to acquire the lock, which will cause overhead traffic on the interconnect and consumes bandwidth. The service rate \( \mu_i \) is therefore:

\[
\mu_i = \begin{cases} 
\mu_0 & \text{if } c + (i-1)l < B \\
\frac{\mu_0 B}{c + (i-1)l} & \text{otherwise.}
\end{cases}
\]

where \( l \) denotes the traffic generated by lock acquisitions, \( c \) denotes the bandwidth consuming rate of a single critical section/transaction, and \( \mu_0 \) is the effective service rate when the interconnect can fulfill the bandwidth requirements of all the critical sections (i.e. when \( c + (i-1)l < B \)). When the total requested bandwidth exceeds the capability of the bus (i.e. when \( c + (i-1)l > B \)), we assume the total bandwidth of \( B \) will be equally shared among the \( i \) threads, thus the thread currently active in the critical section will only receive its portion of the bandwidth as stated in Eq. 2.

With \( \lambda_i \) and \( \mu_i \) defined, we obtain the following set of equations for the steady state:

\[
\begin{align*}
P_0 \lambda_0 &= P_1 \mu_0 \\
P_1 (\lambda_1 + \mu_0) &= P_0 \lambda_0 + P_2 \mu_1 \\
\vdots \\
P_N \mu_{N-1} &= P_{N-1} \lambda_{N-1} \\
P_0 + P_1 + \cdots + P_{N-1} + P_N &= 1
\end{align*}
\]

where \( P_i \) is the probability that the system is in state \( i \) in steady state.

Solving these equations gives:

\[
\begin{align*}
P_0 &= \frac{1}{1 + \sum_{j=0}^{N} \prod_{k=0}^{j-1} \frac{\lambda_k}{\mu_k}} \\
P_i &= \frac{\lambda_{i-1}}{\mu_{i-1}} P_{i-1}
\end{align*}
\]

The expected number of critical section instances queued in the system can be calculated as follows:
can finish its transaction before anyone else commits, it will commit successfully (the $3' \to 4$ transition). Therefore, the next state will be $(i + 1)$ and the rate is denoted by $\omega'_i$. In case (b), the arriving thread fails to commit its transaction because another thread completes its transaction earlier (the $3' \to 4'$ transition). In this case, the next state will be $i'$ and $\omega_i$ is used to denote the rate of these arrivals.

We know the sum of $\omega_i$ and $\omega'_i$ is the cumulative arrival rate:

$$\omega_i + \omega'_i = \lambda_i$$

3.2. The Model of CTL-based STMs

CTL does not enforce exclusive execution of the transactions. To model the rollback behavior of CTL-based STMs, we differentiate two scenarios. We say the system is at state $i$ if there exist $i$ threads, and exactly one of them (denoted as the working thread) will commit its transaction and the others (denoted as the interfering threads) will rollback. We say the system is at state $i'$ if there exist $i$ threads and all of them will rollback their transaction at the commit point. The steady-state probability that the system is at state $i$ (or $i'$) is represented by $P_i (P'_i)$. The state diagram is shown in Fig. 3.

3.2.1. Arrival Rate. The cumulative arrival rate is decided by the number of threads currently in the system which is similar to Eq. 1.

If a thread issues a new transaction, the system will transit to state $(i + 1)$ because either the current working thread will still be able to commit or the new thread will become a working thread.

Next we derive the probabilities for both cases. The probability of case (a) is equal to the probability that the arriving thread (T1 in Fig. 4) can commit its newly issued transaction. Due to the memoryless property of Poisson distribution, it is safe to assume that all other threads (T2, T3 and T4 in Fig. 4) start the transaction at the same time $t_0$. In Fig. 4, $t_{jk}$ represents the time of the thread Tj’s $k$th event. T1 is able to commit its transaction only when $t_{11}$ is smaller than $t_{22}$, $t_{32}$ and $t_{42}$, the probability of which is expressed as Eq. 8.

Because our assumption is that the arrival and departure of all transactions obey Poisson distribution, the intervals $\Delta t_i$ ($t_{01}$ to $t_{11}$) and $\Delta t'_i$ ($t_{11}$ to $t_{12}$) of thread $i$ are independent identically-distributed (I.I.D.) and obey exponential distribution. Therefor, $Z_i = \Delta t_i + \Delta t'_i$ obeys Erlang-2 distribution with rate $\frac{\lambda_i}{2}$ and also I.I.D. We can extend Eq. 8 to Eq. 9.

$$Pr\{(i-1)' \to i\}$$

$$= \int_0^{\infty} Pr\{Z_2 > x, Z_3 > x, ..., Z_i > x|\Delta t_1 = x\}dx$$

$$= \int_0^{\infty} Pr\{Z_2 > x|\Delta t_1 = x\}Pr\{Z_3 > x|\Delta t_1 = x\}\cdots Pr\{Z_i > x|\Delta t_1 = x\}dx$$

$$= \int_0^{\infty} Pr\{Z_2 > x|\Delta t_1 = x\}^{i-1}dx$$

$$= \int_0^{\infty} (1 - F_Z(x))^{i-1}f_{\Delta t_1}(x)dx$$

where $F_Z(z)$ is CDF of Erlang-2 distribution with rate $\frac{\lambda_i}{2}$ and $f_{\Delta t_1}(\Delta t_1)$ is the PDF of exponential distribution with rate $\frac{\lambda_i}{2}$. After plugging them into the Eq. 9, we get

Figure 3. State transition diagram of CTL queuing system

$$L = \sum_{i=0}^{N} iP_i$$

(5)

We know the expected arrival rate $E(\lambda) = \sum_{i=0}^{N} \lambda_i P_i$. According to Little’s Law, the mean completion time $W$ of each critical section is therefore:

$$W = \frac{L}{E(\lambda)} = \frac{\sum_{i=0}^{N} iP_i}{\sum_{i=0}^{N} \lambda_i P_i}$$

(6)
where different service rates need to be considered for the following servicing event occurs in this queuing system. Consequently, the system may stay at the same state even though some bandwidth has consumed certain bandwidth. In other words, the overall serving capability \( \mu \) mines when a transaction is completed.  

\[ \omega = \frac{1}{i} \sum_{i=1}^{l-1} \prod_{m=0}^{i-1} \left( \frac{i-m}{i} \right) \lambda_i \]

(10)

as the probability of case (a). The probability of case (b) is 
\[ P_{r\{i-1\}' \rightarrow i\} = 1 - P_{r\{i-1\}' \rightarrow i\} \]. Thus, we have

\[ P_{r\{i-1\}' \rightarrow i\} = \int_0^\infty \frac{1}{i} \prod_{i=1}^{l-1} \left( \frac{i-m}{i} \right) \lambda_i \]

(11)

\( \omega_i \) and \( \omega'_i \) are obtained as follows:

\[
\begin{align*}
\omega_i &= \frac{1}{i} \sum_{i=1}^{l-1} \prod_{m=0}^{i-1} \left( \frac{i-m+1}{i} \right) \lambda_i \\
\omega'_i &= \frac{1}{i} \sum_{i=1}^{l-1} \prod_{m=0}^{i-1} \left( \frac{i-m}{i+1} \right) \lambda_i
\end{align*}
\]

(12)

3.2.2. Service Rate. In a queuing system, service rate determines the rate at which customers leave the system. Similarly in our transactional system, the service rate determines when a transaction is completed.

The overall serving capability \( \mu \) at state \( i \) and \( i' \) (both having \( i \) active transactions) can be derived as:

\[
\mu_i = \begin{cases} 
\mu_0 & \text{if } i < B/c \\
B & \text{otherwise}
\end{cases} \quad (i = 1, 2, 3, \ldots, N)
\]

(13)

where \( \mu_0 \) denotes the bandwidth requirement of a single transaction and again, \( c \) denotes the bandwidth consuming rate of transactions.

Due to the fact that a transaction may be rolling back, the transaction may remain in the system even though it has consumed certain bandwidth. In other words, the overall serving capability is different from the actual serving rate, and system may stay at the same state even though some servicing event occurs in this queuing system. Consequently, different service rates need to be considered for the following four state transitions:

1) \( i \rightarrow i' \): When an interfering thread finishes its transaction and needs to rollback, the system will remain at state \( i \). Its service rate is denoted by \( \mu_{i-1}' \).

2) \( i \rightarrow (i-1)' \): When a working thread finishes its transaction and the transaction commits successfully, the system will transit from state \( i \) to \( (i-1)' \) because all other interfering threads need to rollback. Its service rate is denoted as \( \mu_{i-1}' \).

3) \( i' \rightarrow i' \): When an interfering thread rolls back and starts its re-execution, another interfering thread may interfere it and let it abort. As a result, the system will remain in state \( i' \). The service rate is denoted as \( \gamma_i \).

4) \( i' \rightarrow i \): When an interfering thread rolls back and if the re-execution will commit successfully, the system will transit from state \( i' \) to \( i \). The service rate is denoted as \( \gamma_i' \).

We observe that the cumulative service rate for each pair (Cases 1&2, Cases 3&4) is fixed which is similar to Eq.7:

\[
\mu_i' + \mu_i'' = \mu_i (\text{for } i = 1, 2, 3, \ldots, N)
\]

(14)

\[
\gamma_i + \gamma_i' = \mu_i (\text{for } i = 2, 3, \ldots, N)
\]

(15)

At state \( i \), the probability of Case 2 is straightforward \( \frac{1}{i} \). Correspondingly, the probability of Case 1 is \( \frac{i-1}{i} \). Then

\[
\mu_i' = \frac{i-1}{i} \mu_i
\]

(16)

For Cases 3 and 4, similar to the derivation of Eq. 10, we obtain

\[
\begin{align*}
\gamma_i &= \frac{1}{i} \sum_{i=1}^{l-1} \prod_{m=0}^{i-1} \left( \frac{i-m}{i} \right) \mu_i \\
\gamma_i' &= \frac{1}{i} \sum_{i=1}^{l-1} \prod_{m=0}^{i-1} \left( \frac{i-m}{i} \right) \mu_i
\end{align*}
\]

(17)

3.2.3. Steady State Transaction Completion Time.

Let \( \Pi \) denote the steady state probability vector \([P_0, P_1, P_2, \ldots, P_{N-1}, P_N', P_{N-1}', P_N', P_N']\). The intensity matrix (Eq. 20) can be obtained by following the state diagram in Fig. 3.

For the steady state probability, we also have

\[
\begin{align*}
\Pi Q &= 0 \\
\sum_{i=0}^{N} P_i + \sum_{j=1}^{N} P_j' &= 1
\end{align*}
\]

(18)

(19)

where \( I \) is the identity matrix and \( P_i (P_j') \) is the probability that the system is at state \( ii' \) in steady state. Substituting the results from Eq. 1, 12, 16 and 17 into intensity matrix and solving Eq. 21, we can derive the steady-state probabilities for all states. Note that although it is difficult to obtain a closed form solution for Eq. 21, a numerical solution can
be obtained easily. The expected number of threads in the system is therefore

\[ L = \sum_{i=0}^{N} iP_i + \sum_{j=1}^{N} jP_j' \]  

and because the mean arrival rate is \( E(\lambda) = \sum_{i=0}^{N} \lambda_i P_i + \sum_{j=1}^{N} \lambda_j P_j' \), by Little’s Law, we derive the mean transaction completion time:

\[ W = \frac{L}{E(\lambda)} = \frac{\sum_{i=0}^{N} iP_i + \sum_{j=1}^{N} jP_j'}{\sum_{i=0}^{N} \lambda_i P_i + \sum_{j=1}^{N} \lambda_j P_j'} \]  

which can be calculated numerically.

4. Experimental Results

In this section, we present the numerical analysis based on the above models and the relevant experimental results. In order to validate our model more accurately, we should minimize the overheads that were abstracted away in the models (initialization, logging, locking and committing, etc.).

Thus, we choose the M5 multi-processor simulator [4] to conduct extensive simulation-based experiments. The other advantage of simulation-based experiments is that we can easily adjust the system bandwidth to prove the bandwidth is one of constraints for the performance. Note that the results generated by the simulator are quite close to those we collected from the experiments of comparison between TL2 and locks on real systems.

In the experiments, a critical section/transaction contains a series of operations on an array of 10k 32-bit words. Two types of operations are used in the experiments. The first type performs pair-wise swap of the array elements. This type of critical sections are designed to stress the interconnect. To offset caching effect, each pair of elements are selected from different cache blocks. Such transactions are called “full transactions”. The second type only reads the variables to a local cache and does not perform any write operations. This type of critical sections are designed to put the least amount of pressure on the interconnect. We denote this type of transactions as “empty transactions”.

Each thread will execute multiple (20) transactions separated by random time intervals that obey exponential distribution. The reported mean transaction completion time is averaged across all the threads. The processors are configured to operate at 1GHz and one instruction per cycle. L1 cache is 16KB with 1ns latency, and L2 cache is 128KB with 10ns latency. The CPUs connect to a 256MB shared memory through a shared bus.

We compared the performance of locks and STM’s by investigating the impacts of the number of threads, transaction issue rate, the bandwidth of interconnect, the bandwidth consuming rate of transactions, and lock quality. The first four parameters were directly controlled in the simulations. Lock quality was controlled through the adjustable waiting algorithms described below:

```c
while (!trylock(L)) {
   waiting_algorithm();
}

critical_section();
unlock(L);
```

By adjusting the waiting period between acquisition attempts, we were able to adjust the lock quality (parameter \( l \)) in terms of the amount of traffic that is sent onto the bus. STM is implemented similarly to the one described in [6], which has demonstrated competitive performance among state-of-the-art STM designs. Because we know in advance the operations involved in each transaction, we therefore statically implemented locking and abort functionalities in the transactions. Consequently, the overheads associated with initialization, logging, locking and committing are greatly reduced. The reduced overhead is particularly helpful for the validation of our theoretical model which has neglected these overheads.

The transaction executing rate \( \mu_0 \) is normalized to 1. The transaction issue rate is the rate at which an individual processor starts new transactions and issue them onto the interconnect and it is denoted as \( \lambda_0 \). It can be adjusted by changing the length of the intervals between transactions. \( B \) determines the number of transactions the bus can support under zero interference. The bandwidth consuming rate \( c \) is relative to the traffic that a critical section/transaction involves. Lock overhead rate \( l \) represents the lock quality in lock model: when \( l = 1 \) (or \( l = 0.1 \)), the waiting algorithm of this lock will generate the same (or 10%) amount of traffic as the critical section itself. Unless otherwise stated, the experiments are configured with 16 threads, \( \lambda_0 = 0.1 \), \( B = 640Mbps \), \( c = 10Mbps \), and \( l = 0.01 \).

We first conducted experiments with different bandwidth consuming rates of transactions. We tested “empty transaction” and “full transaction” as described above. Fig. 5 shows...
the results. Compare to the “empty transaction” which has no traffic on the bus, “full” transaction (maximum bandwidth consuming rate) performs worse, but the difference is small enough to be negligible. When the transactions have a bandwidth consumption rate in between the “empty transactions” and the “full transactions”, the performance is expected to be close, too. The results suggest that the bandwidth consumption rate of the individual transactions does not have a major impact on the performance of STMs. In the following experiments, we will only present results on “full transactions” as the results on “empty transactions” are very close.

The next set of experiments demonstrate the impact of locking overheads. \( B \) is set to 160Mbps. We tested lock overhead rates of 0.01, 0.08 and 0.1 respectively and the results are illustrated in Fig. 6. Practical lock implementations (such as the ticket lock) have optimized the traffic on the interconnect, and \( l = 0.1 \) represents scenarios for unrealistically poor locks. Even for such a large value of \( l \), locks still outperform STM, especially when the number of threads is large. This is because when the contention level increases, locks can reduce the amount of traffic on the interconnect, but STMs will cause excessive traffic due to the large number of transactions that need to be rolled back. In this situation, STM effectively reduces the amount of available bandwidth.

Fig. 7 shows the performance of both locks and STM when the number of threads changes. Locks outperform CTL-based STM in numerical analysis as well as in the simulations. Further more, the performance gap increases as the number of threads increases, from 23% (2 threads) to 77% (64 threads).

Fig. 8 demonstrates the impact of the transaction issue rate. This set of experiments were configured with 16 threads. We controlled the transaction issue rate by changing the length of time intervals between transactions. Given a linear growth of transaction issue rate, our analytical results demonstrated the same trend as the simulations. For locks and STMs, the transaction completion time increases when transaction issue rate increases. However, locks leads to shorter completion time and the performance gap increases (from 40% to 62%) as the transaction issue rate increases.

Bus bandwidth is another factor impacting the contention level and hence the performance of Locks and STMs. We tested its influence when \( B \) is set to 160Mbps, 640Mbps (Fig. 7) and 3.2Gbps. The result is illustrated in Fig. 9. When the bus bandwidth is high enough or when the number of threads is relatively small, both locks and STM has almost the same performance. However, when the number of threads exceeds 32, the completion time of each transaction grows considerably (by as much as 4 times) when the bus has a lower bandwidth of 160Mbps. The results show that the performance of STM is more sensitive to the drop of bus bandwidth.

5. Conclusion and Discussions

In this paper, we develop statistical models for locks and CTL-based STMs. The models, as validated by the simulation results, show that fine-grained locks outperform CTL-based STMs. The performance gap is marginal when
the contention level is low, but increases considerably as the contention level increases. Simulations show that the contention level is affected by the number of threads, transaction issue rate, and bandwidth of the interconnect. When the amount of resources are limited in a system, lock-based method will be able to achieve a lower transaction completion time. In other words, lock-based method can accommodate more transactions/critical sections. Our results suggest that on-going studies of CTL need to investigate techniques to reduced the amount of interfering traffic on the bus.

Please note that the study in this paper did not model the overheads of maintaining the local transaction log, rolling back, and committing, etc. As part of our future research, we will embed these overheads into our model to describe more realistic scenarios. When such overheads are incorporated into the model, we expect to observe further performance penalties for CTL-based STMs.

References


