Performance Modelling of Hardware Transactional Memory

Daniel Filipe Salvador de Castro

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: **Paolo Romano**

Examination Committee

Chairperson:	Prof. José Carlos Martins Delgado
Supervisor:	Prof. Paolo Romano
Member of the Committee:	Prof. Aleksandar Ilic

October 2016

Abstract

Transactional Memory (TM) is a recent alternative to traditional lock based synchronization mechanisms for parallel programming. Our analysis of existing literature in these areas highlights the existence of a relevant gap, which we aim to fill with this dissertation: the lack of performance models for hardware-based implementations of TM, also known as Hardware Transactional Memory (HTM). In order to monetize all the available transistors in a modern processor, HTM is usually built on top of the existing cache coherency protocols, whose dynamics we capture in the presented simulative and analytical models. Moreover, the simulation model is tested regarding Intel's implementation of HTM and predicts, with little discrepancies, probability of abort and throughput. Subsequently, the analytical model is validated against this simulation, with an average error of 1.69% and 4.07% regarding probability of abort and throughput, respectively.

Keywords: transactional memory, hardware, performance modeling, concurrency control

1 Introduction

One of the main sources of complexity in parallel programming is related to the need of properly synchronizing accesses to shared memory regions. In fact, the traditional, lock-based approach to synchronize concurrent memory accesses is well-known to be error prone even for experienced programmers: on the one hand, using coarse-grained locking, e.g., synchronizing any concurrent access via a single lock, can severely limit parallelism; on the other hand, while the usage of fine-grained locks can enable larger degrees of parallelism, it also opens the possibility of deadlocks and hinders a key property at the basis of modern software engineering, composability [1].

TM has emerged as a simpler, and hence more attractive, alternative to lock-based synchronization: unlike mutual exclusion, TM only requires the identification of the atomic code blocks. How atomicity is achieved no longer concerns the programmer.

Over the last 20 years, several implementations of TM have been proposed, using either software, hardware, or a combination of both. Yet, at current date, performance of TM remains a controversial issue [2, 3].

Hardware implementations of TM (HTM) avoid the instrumentation costs incurred by Software Transactional Memory (STM), but their nature is inherently restricted and best-effort. In fact, commercially available HTM implementations (such as the ones provided by Intel Haswell or IBM P8 processors) rely on cache coherency protocols to keep track of the memory regions accessed by transactions. As such, they suffer of spurious aborts whenever relevant transactional metadata has to be evicted by the cache (e.g., due to cache capacity limitations).

These widely available HTMs, launched with recent processors, thus provide performance that varies significantly with the workload. However, the internals of their implementations are mostly undisclosed, for which reason it is hard to predict a priori how an HTM will perform for a given new application.

This work aims to develop tools for predicting the performance of HTM-based applications. This objective will be achieved in two steps: i) first a discrete event based simulator is developed. This simulator will focus on modeling the performance dynamics of the cache coherency protocol used by Intel Haswell processors, which, as is discussed in more detail in Section 2, represents the backbone used to implement the HTM provided by Intel; ii) next an analytical model of Haswell's HTM is developed. The analytical model will rely on queuing theory and probabilistic techniques (extending prior literature in the area of performance modeling of concurrency control algorithms in STMs and Database Management Systems (DBMSs) literature [4, 5, 6, 7, 8]) and will be validated by means of the event based simulator.

The remainder of this document is structured as follows. In Section 2 we present our experimentation on a Haswell machine, relevant assumptions needed to develop the models are also presented in that section. Section 4 explains the simulation model. The analytical model is presented in Section 5. Section 6 presents the validation of the simulation and of the analytical model. In Section 7 the related work is discussed, with emphasis on the different implementations of TM both in software and hardware, as well as on works that aim to model STMs and DBMSs. Finally, in Section 8, conclusions for this dissertations are presented.

2 Study on Intel TSX

We gathered some information also from some studies on HTM in public literature [9, 10, 11]. We also conducted further experiments by our own on Intel Transactional Synchronization Extensions (TSX).

All the tests are executed in an Intel Xeon E3-1275 v3 @ 3.50GHz. Intel TSX allows the usage of two interfaces: i) lock elision; and ii) Restricted Transactional Memory (RTM). All the following tests use RTM. In RTM, contrarily from lock elision, the programmer must explicit call begin and commit operations.

According to the processor specifications, L1 cache has 64 sets 8-way set associative, L2 has 512 sets, also 8-way. The capacity of L3 is 8 MB. The cache line size in all levels of cache is 64 B. The tested processor is a quad-core with Hyper-Threading.

Firstly, analyzing the available literature, and conducting experiments, we discovered that Intel detects conflicts eagerly. Then, we mounted a experiment to stress the conflict detection in TSX. This experiment consists in start concurrently two transactions, T_1 and T_2, both access the same granule. The trick is to schedule the access such that T_1 does the access first, then T_2 accesses. We discovered that who accesses the granule first aborts. This can be explained with the MESI protocol, as explained in Section 2.1

Secondly, we studied the transactional capacity of TSX. In this front we did three experiments: i) we stressed the fully associativity of Intel's cache; ii) we experimented various contiguous and strided access; iii) we analyzed the maximum capacity for random access patterns.

In the experiment i), we are confident that for strided write only accesses multiple of 64, the maximum capacity is 8, i.e., the granule is always mapped to the same set in L1 cache. For strided write only accesses less than 64, i.e., experiment ii), the theoretical capacity is also closely achieved, however, we are confident that some "extra" granules are kept in L1, either transactional metadate or transparent memory accesses injected by the compiler.

The last experiment iii), regarding a simulated L1 capacity, shown that closely to 30 granules are not possible to use, i.e., are polluted.

For the read accesses, we are developing a model where we theorize that reads can be model as they occupy some space c in L1, such that $0 < c < 1$, and this c is a function of the provided workload.

2.1 The MESI protocol

As we previously stated: the last transaction to acquire a granule, in an incompatible access, aborts the remaining transaction. This can be explained by the invalidation process in Modified, Exclusive, Shared, Invalid (MESI).

In this cache coherence protocol a line may be in one of the following states: i) (M)odified; ii) (E)xclusive, ; iii) (S)hared; or, iv) (I)nvalid.

For example, assume two caches, c_1 and c_2, and a granule g, such that in the start the granule is not present in any of the caches, i.e., $s_g : [I, I]$. If c_1 reads g, a copy is stored in c_1: $s_g : [E, I]$. If now c_2 stores a modification of g, the copy in c_1 must be invalidated, i.e., $s_g : [I, M]$.

Using this protocol, the following accesses produce the granule invalidation in $T1$: T_1 writes g than T_2 reads or writes; T_1 reads, then T_2 writes.

On granule invalidation, or eviction, the respective transaction is aborted. We did not studied the eviction policy in L1, but we assume to be Least Recently Used (LRU), hence, the transactional accesses have priority over previous non-transactional ones, allowing the usage of almost all L1.

2.2 The TSX algorithm

We are modeling the case of HTM with a fall-back to global lock solution. Each transactions starts with some amount of budget that is updated upon an abort. When the budget exhaust, the transaction runs serially by means of the global lock alternative. This is the approach presented in Algorithm 1.

In order to start a transaction, an explicit call to `BeginXact` (Line 4) is needed. This routine handles the budget update and transparently interacts with the TSX RTM interface.

The fall-back lock is checked twice, once immediately before starting the transaction (Line 6), and again after it begins (Line 11)). This avoid a phenomenon where all transactions in the system deplete their budgets when checking for the lock, also called the *lemming effect*.

Algorithm 1: TSX RTM with lock fall-back.

```
1  int budget = INIT_BUDGET
2  int roaPolicy = {LINEAR, HALF, ZERO}
3
4  void BeginXact() {
5    while(1) {
6      while(sgl == 0) {
7        asm("pause;");
8      }
9      int status = _xbegin();
10     if(status == XBEGIN_STARTED) {
11       if(IS_LOCKED(sgl)) { _xabort(30); }
12       break;
13       if(status == XABORT_CAPACITY) {
14         budget = SPEND_BUDGET(budget);
15       } else { tries--; }
16       if(tries <= 0) {
17         while(CAS(sgl, 0, 1) == 1) { asm("pause;"); }
18         break;
19       }
20     }
21   }
22 }
23
24 void EndXact() {
25   if{budget > 0) { _xend(); }
26   else { sgl= 0; }
27 }
28
29 int SPEND_BUDGET(int b) {
30   if(roaPolicy == LINEAR) { return b - 1; }
31   else if(roaPolicy == HALF) { return b/2; }
32   else { return 0; }
33 }
```

The lock is seen as it is a granule in memory. When someone writes the lock, the lock granule generates a conflict in all active transactions, which have to abort (Line 17), this conflict detection behavior is further analyzed in the next section.

When transactions return to the begin point (Line 10), due to an abort, they restart non-transactionally. The abort cause in variable `status` (Line 9) may be the following: i) `XABORT_CAPACITY`; ii) `XABORT_CONFLICT`; or iii) 0, for non-specified cause.

If the cause is a conflict either: the lock is taken; or there is a conflicts on some other granule (Line 15).

If the cause is a capacity exception (Line 14), the transaction might be too large to fit into cache. In this situation, three different policies (Line 2) have been studied in literature [39]: i) `LINEAR` (Line 30); ii) `HALF` (Line 31); and iii) `ZERO` (Line 32).

The transaction restarts until: i) it commits; or, ii) the budget B exhausts. If i) occurs, then in `EndXact` (Line 24) the respective `_xend` routine is called and commits the hardware transaction. If ii) occurs, then the previously acquired lock (Line 17) is released (Line 26).

3 Notation

The HTM system is composed by a number of θ threads. Those may execute Transactional Code Block (TCB) or Non-Transactional Code Block (NTCB). A TCB is the code executed inside a transaction. A NTCB, by opposition, is the code executed outside a transaction. Given the speculative nature of transactions, a thread may restart a TCB several times.

At some instant, an HTM system has θ_t threads running TCBs, θ_f threads running serialized TCBs and θ_n threads running NTCB, such that $\theta_t + \theta_n + \theta_f = \theta$. A thread may start a transaction with probability p_t ($1 - p_t$ for non-transactions).

Assume when some transaction enters the serialized path, only one transaction executes in the system at the same time. The remaining TCBs and serialized TCBs are blocked.

Each transaction T starts with initial budget B, i.e., the number of allowed restarts before falling back into a serialized TCB. The remaining budget is decremented by 1 after an abort (i.e., linear policy).

We named transactions with remaining budget equal to 1 *dangerous transactions*, because they may cascading abort all other transactions if they abort. If the remaining budget is greater than 1, then by opposition, they are named *non-dangerous transactions*.

Any code block is assumed to complete in a given C average time, exponentially distributed. Each TCB acquires exactly L granules uniformly distributed over the total D available granules. After each interval $W = C/L$, a granule is acquired. Assume NTCB to not conflict with TCB, for example, in the tested workloads they access different regions in memory.

The total number of possible acquirable granules is given by the parameter D. The larger the D, the larger the pool from where T may choose a granule. The workload is further characterized with the probability of write, i.e., P_W.

The probability of abort is noted p_a, and the throughput X, i.e., the number of completed transactions per unit of time.

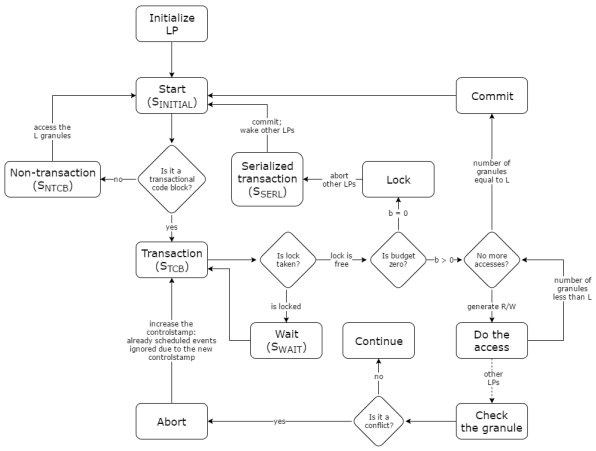


Figure 1: Simulation diagram.

4 Simulation model

Our simulation is built on top of ROme OpTimistic Simulator (ROOT-SIM) [12]. This discrete event simulator requires the programmer to define a number of Logical Processes (LPs). In our simulation, each LP simulates a thread τ_i.

Each τ is on an initial state S_{INITAL}, where it decides with probability p_t to start a transaction, i.e., enter state S_{TCB}.

After W time units, the first granule g_i is acquired and the acquisition of the next granule g_j is scheduled. All the remaining τ_i receive an event with the acquired g_i. A LP aborts if it contains g_i.

After L accesses τ commits. After commit, τ enters again the S_{INITAL} state, and repeats the process until a given number of sampled transactions is obtained or a timeout occurs.

Other simulated behavior includes the acquisition of the fall-back lock: in the simulation the fall-back lock is acquired by consensus, if no one in the serialized path and no one intendeds to enter it. Other aspect is the unscheduling of future granule acquisitions on transactional abort. Since ROOT-SIM does not provide an unschedule mechanism, we introduced a sequence number, if the sequence number in the event does not match the LP state, the event is ignored.

To simulate capacity, the simulation must be provided with the number of sets S and the n-way set associativity N of the cache. Then, the cache is abstracted as a group of bins, all with the same capacity. The mapping function for a granule g in a bin is the modulus function, i.e., $g \mod S$. Other functions may be used. A capacity exception is detected if a granule is mapped to a full bin.

5 Analytical model

As in any white box model, knowledge of the system internals is needed. Therefore, we will assume the empirically observed behavior presented in Section 2.

The ultimate goal for this analytical model is to predict the system probability of abort, i.e., p_a, and its throughput, i.e., X.

The description of the model follows a top down approach. First, we describe how we model the evolution of the system depending on the transactional events that take place (e.g., aborts and commits). The probabilities of, and the rate at, which these events occur are then analytically derived.

At some instant t, this system has θ threads running TCBs, serialized TCBs or NTCBs. Among the TCBs, those threads may have different remaining budget i, such that, $0 < i \leq B$, assume $i = 0$ to be the serialized TCBs.

Each instant t is characterized by a generic state s in the form: $[\theta_B, .., \theta_i, .., \theta_1, \theta_f, \theta_n]$. All $s[i]$ positions must sums θ. Threads may change the state when they complete a code block, or abort. The possible transitions for a generic thread τ are the following (rates presented in Table 1):

i) τ completes a NTCB and starts other NTCB:
$[\theta_B, .., \theta_f, \theta_n] \Rightarrow [\theta_B, .., \theta_f, \theta_n]$

ii) τ completes a NTCB and starts a TCB:
$[\theta_B, .., \theta_f, \theta_n] \Rightarrow [\theta_B + 1, .., \theta_f, \theta_n - 1]$

iii) τ commits a TCB and starts other TCB:
$[\theta_B, .., \theta_i, .., \theta_1, 0, n] \Rightarrow [\theta_B + 1, .., \theta_i - 1, .., \theta_1, 0, \theta_n]$

iv) τ commits a TCB starts a NTCB:
$[\theta_B, .., \theta_i, .., t_1, 0, n] \Rightarrow [\theta_B, .., \theta_i - 1, .., \theta_1, 0, \theta_n + 1]$

v) τ aborts a non-dangerous TCB:
$[\theta_B, .., \theta_i, .., \theta_1, 0, n] \Rightarrow [\theta_B, .., \theta_i - 1, \theta_{i-1} + 1, .., \theta_1, 0, \theta_n]$

vi) τ aborts a dangerous TCB:
$[\theta_B, .., \theta_i, .., \theta_1, 0, \theta_n] \Rightarrow [0, \theta_B, .., \theta_{i+1}, .., \theta_2, \theta_1, \theta_n]$

vii) τ completes a serialized TCB and starts a TCB:
$[\theta_B, .., \theta_1, \theta_f, n] \Rightarrow [\theta_B + 1, .., \theta_1, \theta_f - 1, \theta_n]$

viii) τ completes a serialized TCB and starts a NTCB:
$[\theta_B, .., \theta_1, \theta_f, \theta_n] \Rightarrow [\theta_B, .., \theta_1, \theta_f - 1, \theta_n + 1]$

The solution vector $\vec{\pi}$, gives, for some state $s \in S$, the probability of being in that state $\vec{\pi}_s$.

Equation 1 presents the probability of reaching the i-granule. To compute it, we relied on the exponential assumptions of conflict arrival rate. The arrival rate of conflicts is given by $H(i) = P_I P_{\text{conflict at i}} \lambda$ and depends on: i) the number of acquired granules, $P_{\text{conflict at i}} = i/D$; ii) the compatibility of the access type, $P_I = 1 - (1 - P_W)^2$; iii) the number of concurrent transactions, $\lambda = (\theta_t - 1)/W$.

$$P_R(i) = \begin{cases} 1 & \text{, if } i = 0 \\ P_R(i-1)e^{-H(i-1)W} & \text{, if } i > 0 \end{cases} \quad (1)$$

Once defined the probability of abort per state, $p_{a,s}$, the global probability of abort is computed as in Equation 2.

$$p_a = \sum_{s(i,j=0,k) \in S} \vec{\pi}_s p_{a,s} \quad (2)$$

The global throughput is computed in Equation 3, and follows a similar approach. $\mu_{n,s}$ and $\mu_{f,s}$ are equal to $1/C$. $\mu_{c,s}$ is computed as the probability of commit multiplied by the rate of complete TCBs.

$$\begin{aligned} X = & \sum_{s(\theta_i, \theta_f = 0, \theta_n) \in S} \vec{\pi}_s (\theta_n \mu_{n,s} + \theta_i \mu_{c,s}) + \\ & + \sum_{s(\theta_i, \theta_f > 1, \theta_n) \in S} \vec{\pi}_s (\theta_n \mu_{n,s} + \mu_{f,s}) \end{aligned} \quad (3)$$

Our model assumes $p_{a,s}$ equal to $1 - P_R(L)$ and the average response of a TCB R as in Equation 4. $\mu_{c,s}$ is the inverse of R, i.e., $1/R$.

$$\begin{aligned} R = & P_R(L)C + \sum_{i=1}^{L-1} P_R(i) \Big(iW(1 - e^{-H(i)W}) + \\ & + \frac{1}{H(i)} - e^{-H(i)W} \left(W + \frac{1}{H(i)} \right) \Big) \end{aligned} \quad (4)$$

Note that $H(i)$ does not account the aborts due to dangerous transactions acquiring the global lock. After computing the solution vector $\vec{\pi}$, $p'_{a,s}$ and $\mu'_{c,s}$ are calculated to have an approximated valued, and then are used in the p_a and X.

$p'_{a,s}$ and $\mu'_{c,s}$ are the weighted average of the cases where τ is dangerous (using $H_d(i) = H(i) + (\theta_1 - 1)\mu_{c,s} p_{a,s}$) and non-dangerous (using $H_n(i) = H(i) + \theta_1 \mu_{c,s} p_{a,s}$).

Transition	rate
i)	$\theta_n \mu_n (1 - p_t)$
ii)	$\theta_n \mu_n p_t$
iii)	$\theta_i \mu_t (1 - p_a) p_t$
iv)	$\theta_i \mu_t (1 - p_a)(1 - p_t)$
v)	$\theta_i \mu_t p_a$
vi)	$\theta_i \mu_t p_a$
vii)	$\mu_f p_t$
viii)	$\mu_f (1 - p_t)$

Table 1: State transition rates.

5.1 Capacity exceptions

The previously presented model assume the cache to have unlimited capacity, and only capture conflicts.

An easy approach to introduce capacity exceptions, is to update $P_R(L)$, which the model extensively uses, to encompass the probability of capacity exception at the i-th granule. This approach treats capacity exceptions as conflict exceptions in terms of budget update.

$$N_{\text{B,C,I}} = \sum_{x=\min_c}^{\max_c} \binom{B}{x} \prod_{y=0}^{\text{x-1}} \binom{\text{I-yC}}{C} \times N_{\text{B-x,C-1,I-xC}} \quad (5)$$

Mapping this problem to the problem: "how many combinations one can put I distinguishable balls, in B bins of capacity C". The total number of combinations is given by Equation 5, where $\min_c$ and $\max_c$ are, respectively, the minimum and maximum number of full bins with I balls. Assume that $N_{0,C,I}$ and $N_{B,C,0}$ are 1.

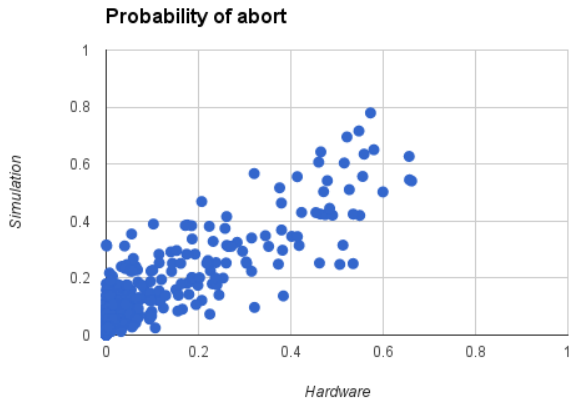
The probability of capacity at the i-th ball is $1 - N_{B,C,I}/B^I$, where B^I is the number of combinations without the capacity constraint. This formula is slow to compute for large number of B, C and I. In the future an approximation must be used.

6 Experimental results

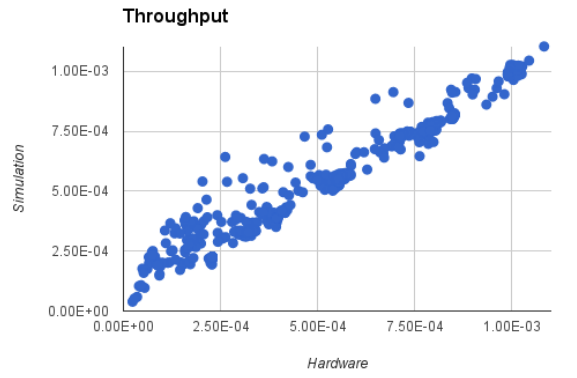
The accuracy of the proposed models will be evaluated using 3 main metrics:

- Mean Absolute Error (MAE), defined by the formula: $\frac{1}{n} \sum_{i=1}^{n} |x_i - y_i|$

- Mean Absolute Percentage Error (MAPE), defined as $\frac{1}{n}\sum_{i=1}^{n}\frac{|x_i - y_i|}{y_i}$
- The Pearson correlation factor r, defined as $\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$.



(a) Probability of abort.
MAE = 2.75%, $r = 0.8781$



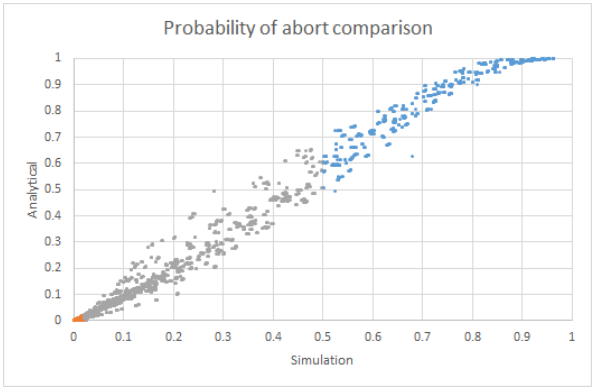
(b) Throughput.
MAPE = 14.98%, $r = 0.9971$

Figure 2: Intel TSX compared with the simulation. Input: $\theta \in \{2,3,4\}$, $L \in \{1,2,5,10,15,20,30\}$, $D \in \{1024, 2048, 4096, 8192, 16384, 32768\}$, $P_W \in \{0.5, 1\}$ and $B \in \{2,4,6,8,10\}$

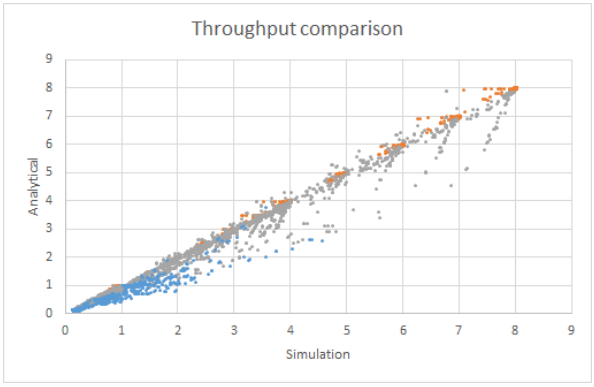
To test if the simulation is correctly capturing the behavior from a real HTM system. The workload is executed in the Intel machine presented in Section 2.

In order to the comparison of the throughput be possible. The simulation is provided with the measured C and the delay time between two consecutive transactions. For this experiment assume $p_t = 1$.

The Intel Time Stamp Counter (TSC) instruction is used to measure the latency between the start and finish points of a transaction. It was also used to measure the time of the commit instruction, i.e., `_xend()`, and the non-transactional time between two TCB.



(a) Probability of abort.
MAE = 1.69%, $r = 0.9949$



(b) Throughput.
MAPE = 4.07%, $r = 0.9963$

Figure 3: Simulation compared with the analytical model. The workload setup is the following: $\theta \in \{2,3,4,5,6,7,8\}$, $L \in \{2,5,10,20,50\}$, $B \in \{1,2,3,4,5,6\}$, $C \in \{1,2,8\}$, $P_W \in \{0.1, 0.5, 0.9, 1\}$, and $D \in \{2000, 5000, 10000, 50000, 100000\}$

The obtained data in Figure 2 shows strong correlation between the simulation and the TSX. However, the measured probability of abort and throughput in TSX have huge variations between runs. In a near future we should fix this issue in order to obtain more stable measurements.

We also tested for $B = 1$, but the results are not aligned with the simulation. In this scenario,

the MAE, regarding probability of abort, is 17.1%, and the MAPE, regarding throughput, is 71.5%. Though, using low values of budget like $B = 1$ is not recommended, due to the best effort nature of HTM. This can be caused to a data race condition on the fall-back lock that may reduce the available budget of non-fall-back transactions. In both models we assume all serialized transactions execute sequentially before non-serialized ones starting its execution again.

The validation data for our analytical model is presented in Figure 3. Over 10000 workloads were tested, stressing many aspects as the impact of different values of budget, concurrent threads, number of accessed granules and probability of writing those granules.

Figure 3 shows the validation data for the first anaytical model.

Some input combinations are not present due to the complexity of the analytical model. For instance, with $\theta = 8$ and $B = 6$ the analytical model does not produce a solution in useful time. The number of states grows exponentially with the number of threads and budget. In these the transition matrix becomes quite large.

At this point the analytical model do not capture the effect of the capacity exceptions. Thus, capacity behavior is disabled in the simulation.

6.1 Capacity tests

We compared the simulation with TSX in a scenario where conflicts are very unlikely to occur, i.e., 1 thread running. The workload is write only and the accesses are done randomly in the granule pool. We tested 10000 different random access patterns.

As shown in Figure 4, the distribution of probability of capacity (blue line with circles) closely match the simulator assuming 30 granules to be inaccessible in memory (these granules are contiguously stored in the cache).

Then we validated the capacity formulas presented in Section 5.1, which closely match the simulation, as is shown in Figure 5. However the complexity of the formula do not allowed us to test with 64 sets 8-way set associative. Thus, we had to tune down the simulation to only encompasses 32 sets 8-way set associative.

Since the formula $N_{B,C,I}$ vastly uses recursion, we cache the values in a map where the key

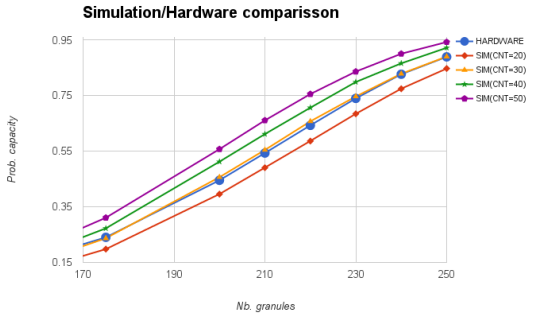


Figure 4: Comparisson of the capacity of Intel TSX with our simulation marking some granules in the cache as metadata (writes only).

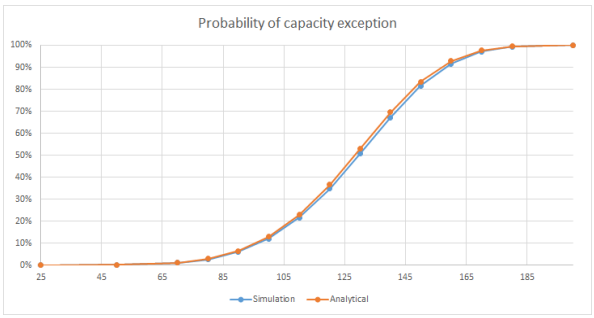


Figure 5: Comparison between simulation and analytical model. Cache geometry is 32 sets and 8-way set associative. $r = 0.9998$.

is the tuple (B, C, I) and the value is $N_{B,C,I}$, i.e., $\langle (B, C, I), N_{B,C,I} \rangle$. Though, the values of $N_{B=64,C=8,I=200}$ is computed as infinite, using double-precision floating-point arithmetic. Thus, higher precision needs to be used to computed this scenario.

7 Related Work

During the past decade there has been intense research in the area of TM, motivated by the current paradigm of multi-core hardware, which creates a great need for concurrency control mechanisms that are simple to use and yet provide scalable performance.

TM provides a simple yet powerful API for concurrency control. The paradigm consist in mark the target critical section using some predefined rou-

tines, e.g., $BeginTX()$ and $EndTX()$. Marking explicitly the reads/writes accesses to the shared resources may also be needed, e.g. $value = Read(granule)$ and $Write(granule, value)$.

Many software of implementations of TM exist [14, 15, 16, 17, 3, 18], i.e., STMs, but only few are available in hardware, as is the case of Intel TSX. However many proposals for the implementations of TM in hardware, i.e., HTM exist [19, 20, 21]. Most hardware implementations rely on the cache coherency protocols to eagerly check when a conflict happens. There are some studies on maintaining the cache coherency in a multi-processor machine [22].

The introduction of HTM in current processors adds extra complexity to the circuit, which may cause bugs in very specific, hard to test, workloads. Thus, manufactures have been very careful when adding extensions with this new future. In the case of Intel, it only implements a best effort HTM where transactions do not survive context switches.

STM may use pessimistic (mutual exclusion) or optimist (timestamps) approaches, or a combination of both. They usually have read and write sets where the granules are temporarily stored. Then there is a validation phase to check any violation of the granule timestamp. However, more pessimistic approaches may detect the conflict eagerly and abort the transaction sooner.

Usually, HTM implementations are best effort and have many limitations. Therefore, there are studies that try to use of both STM and HTM libraries in order to provide the performance of HTM and yet allowing all kind of transactions by falling back to STM, when HTM cannot handle the transaction. This approach is called Hybrid Transactional Memory (HyTM) [23]. However, in some recent studies show that HyTM may perform even worse than pure HTM solutions [24].

TM is still being actively developed. Thus, there are contradictory conclusions concerning the TM performance gains when compared with traditional synchronization techniques [2, 3].

TM must provide that transactions execute isolated and preserve the consistency of the system. The reference correctness criterion for TM is opacity [25]. Opacity is stronger than Serializability, enforcing: i) the real-time order of execution (as in strict serializability [26]); ii) aborted transactions should observe a state producible by executing transactions in some sequential order.

Since many implementations of TM exist, comparing how they perform in the presence of different workloads must be done through well know benchmarks. Benchmarks for TM are, therefore, synthetic applications that use TM to synchronize themselves [27, 28, 29, 30].

Transactional systems are older than TM. Long before TM there were DBMS with similar concerns. Modeling this systems is vastly covered in many papers [4, 5, 6]. These models are mainly analytical validated using the help of a simulation model. Analytical and simulation models are white-box approaches. One must know how the system works to devise the correspondent equations, or simulate its long term behavior. Simulators should run on the assumptions of the system, and abstract most of the complexity that is not capture in the analytical model.

Models for STM also exist [7, 31, 8, 32, 33, 34, 35, 36, 37, 38]. Some make use of analytical models [7, 31, 8]. Others use black-box techniques [32, 33]. And some more recent approaches try to combine analytical and black-box models to achieve better results [34, 35, 36, 37, 38].

The use of black-box techniques in HTM is exploited in Diegues et al. [39]. The model tunes the HTM parameters while the application is running. This approach provides best performance in long term, assuming that the workload in the system has stable and predictable characteristics. This dissertations tries to fill a gap in literature by developing a white-box model for HTM.

8 Conclusion

This report surveyed the state of the art in TM, by explaining its abstraction and implementations in software, hardware, and combinations of both. Given the recent release of best-effort hardware support in Intel processors, some emphasis is given to this new technology, whose details are undisclosed and of the utmost relevance to understand and tune its performance.

As such, the proposal presented here develops a simulation and an analytical model aimed to capture the performance dynamics of Intel TSX. For this, the main approaches for modeling concurrent systems is presented, ranging from black-box to

white-box models.

The reason for developing both models is that the simulator has a twofold purpose: it allows to assess the impact of future changes in the processor architecture (e.g., larger caches), and it can serve to validate the analytical model.

Out experimental results showed some correlations between the obtained data from Intel machine and the simulation. This means that our assumptions are not completely wrong. Some adjustments in our experiment with Intel system must be carefully revisited, in order to decrease its huge variation regarding the measurement of probability of abort and throughput.

Also, our analytical model seems to best fit the simulation data quite well with a MAE of 1.69% regarding probability of abort, and a MAPE of 4.07% regarding throughput. Some numerical issues while computing the transition matrix may have some influence in the results.

In terms of capacity modeling, we are confident that Intel stores the write set in L1 cache, due to the similarities with the gathered data from the simulator. We also developed an equation to compute the probability of capacity exception, which is highly correlated with the simulation data. Although this formula becomes very expensive in the presence of many sets and granules. An approximation must replace it in the future.

How Intel is storing the read set in cache is still a mystery. Possible they use some complex mapping function that we are not aware of.

Finally, some feature are still missing in our model. For example, the validation of NTCB, i.e., $p_t < 1$, how TSX handles conflicts between TCB and NTCB. More parameters must be added to better characterize the workloads. Rather than the provided value for C, this one should be computed having in account the latency of fetching granules from either the different levels of cache or main memory. Among other feature that were not the main focus for this dissertation.

References

[1] Maurice Herlihy and Nir Shavit. The Art of Multiprocessor Programming. mar 2008.

[2] Calin Cascaval, Colin Blundell, Maged Michael, Harold W. Cain, Peng Wu, Stefanie Chiras, and Siddhartha Chatterjee. Software Transactional Memory: why is it only a research toy? *Queue*, 6(5):46, 2008.

[3] Aleksandar Dragojevic, Pascal Felber, Vincent Gramoli, and Rachid Guerraoui. Why STM can be more than a research toy. *Communications of the ACM*, 54(4):70, 2011.

[4] Y. C. Tay, Nathan Goodman, and R. Suri. Locking performance in centralized databases. *ACM Transactions on Database Systems*, 10(4):415–462, 1985.

[5] Rakesh Agrawal, Michael J. Carey, and Miron Livny. Concurrency control performance modeling: alternatives and implications. *ACM Transactions on Database Systems*, 12(4):609–654, 1987.

[6] Philip S. Yu, Daniel M. Dias, and Stephen S. Lavenberg. On the analytical modeling of database concurrency control. *Journal of the ACM*, 40(4):831–872, 1993.

[7] C Zilles and R Rajwar. Transactional memory and the birthday paradox. *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 303–304, 2007.

[8] Pierangelo Di Sanzo, Bruno Ciciani, Roberto Palmieri, Francesco Quaglia, and Paolo Romano. On the analytical modeling of concurrency control algorithms for Software Transactional Memories: The case of Commit-Time-Locking. *Performance Evaluation*, 69(5):187–205, 2012.

[9] Bhavishya Goel, Ruben Titos-Gil, Anurag Negi, Sally A. McKee, and Per Stenstrom. Performance and Energy Analysis of the Restricted Transactional Memory Implementation on Haswell. In IEEE, editor, *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 615–624, Phoenix, AZ, may 2014. IEEE.

[10] Takuya Nakaike, Matthew Gaudet, and Maged M Michael. Quantitative Comparison of Hardware Transactional Memory for Blue Gene / Q , zEnterprise EC12 ,. *ISCA*, 2015.

[11] Andrew Nguyen William Hasenplaugh and Nir Shavit. *Investigation of Hardware Transactional Memory*. PhD thesis, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2015.

[12] Alessandro Pellegrini and Francesco Quaglia. The ROme OpTimistic Simulator: A Tutorial. In *Proceedings of the 1st Workshop on Parallel and Distributed Agent-Based Simulations*, pages 501–512. 2014.

[13] Yujie Liu, Justin Gottschlich, Gilles Pokam, and Michael Spear. TSXProf: Profiling Hardware Transactions. *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, 2016-March:75–86, 2016.

[14] Dave Dice, Ori Shalev, and Nir Shavit. Transactional Locking II. *Distributed Computing*, 4167:194–208, 2006.

[15] Pascal Felber, Christof Fetzer, and Torvald Riegel. Dynamic performance tuning of word-based software transactional memory. *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 237–246, 2008.

[16] Pascal Felber, Christof Fetzer, Patrick Marlier, and Torvald Riegel. Time-Based Software Transactional Memory. *IEEE Transactions on Parallel and Distributed Systems*, 21(12):1793–1807, dec 2010.

[17] Aleksandar Dragojevic, Rachid Guerraoui, and Michal Kapalka. Stretching transactional memory. *ACM SIGPLAN Notices*, 44:155, 2009.

[18] João Cachopo and António Rito-Silva. Versioned boxes as the basis for memory transactions. *Science of Computer Programming*, 63(2):172–185, 2006.

[19] C Scott Ananian, Krste Asanovic, Bradley C Kuszmaul, Charles E Leiserson, and Sean Lie. Unbounded Transactional Memory. In *11th Int'l Symp. on High-Performance Computer Architecture (HPCA'05)*, pages 316–327, 2005.

[20] Kevin E Moore, Jayaram Bobba, Michelle J Moravan, Mark D Hill, and David A Wood. LogTM: Log-based Transactional Memory. pages 1–12, 2006.

[21] Ravi Rajwar, Maurice Herlihy, and Konrad Lai. Virtualizing Transactional Memory. *Proceedings - International Symposium on Computer Architecture*, 00(C):494–505, 2005.

[22] Daniel Molka, Daniel Hackenberg, and Wolfgang E Nagel. Cache Coherence Protocol and Memory Performance of the Intel Haswell-EP Architecture. 2015.

[23] Peter Damron, Alexandra Fedorova, Yossi Lev, Victor Luchangco, Mark Moir, and Daniel Nussbaum. Hybrid Transactional Memory. In *ASPLOS-XII: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[24] Nuno Diegues, Paolo Romano, and Luís Rodrigues. Virtues and Limitations of Commodity Hardware Transactional Memory. *Pact*, pages 3–14, 2014.

[25] Rachid Guerraoui. Opacity : A Correctness Condition for Transactional Memory. *Communication*, 2007.

[26] Christos H. Papadimitriou. The serializability of concurrent database updates. *Journal of the ACM*, 26(4):631–653, 1979.

[27] Anu G. Bourgeois and S. Q. Zheng, editors. *Algorithms and Architectures for Parallel Processing*, volume 5022 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[28] Rachid Guerraoui, Michal Kapalka, and Jan Vitek. STMBench7. *ACM SIGOPS Operating Systems Review*, 41(3):315, jun 2007.

[29] Wenjia Ruan, Trilok Vyas, Yujie Liu, and Michael Spear. Transactionalizing legacy code: an experience report using GCC and Memcached. *ACM SIGARCH Computer Architecture News*, 42(1):399–399–399–412–412–412, apr 2014.

[30] Chi Cao Minh, JaeWoong Chung, Christos Kozyrakis, Kunle Olukotun, Cao Minh Chí, JaeWoong Chung, Christos Kozyrakis, and Kunle Olukotun. STAMP: Stanford Transactional Applications for Multi-Processing. *IISWC '08: Proceedings of The IEEE International Symposium on Workload Characterization*, pages 35–46, 2008.

[31] Armin Heindl and Gilles Pokam. An analytic framework for performance modeling of software transactional memory. *Computer Networks*, 53(8):1202–1214, 2009.

[32] Márcio Castro, Luis Fabrício Wanderley Góes, Christiane Pousa Ribeiro, Murray Cole, Marcelo Cintra, and Jean François Méhaut. A machine learning-based approach for thread mapping on transactional memory applications. *18th International Conference on High Performance Computing, HiPC 2011*, 2011.

[33] Diego Rughetti, Pierangelo Di Sanzo, Bruno Ciciani, and Francesco Quaglia. Machine learning-based self-adjusting concurrency in software transactional memory systems. *Proceedings of the 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2012*, pages 278–285, 2012.

[34] Diego Didona, Paolo Romano, Sebastiano Peluso, and Francesco Quaglia. Transactional Auto Scaler: Elastic Scaling of In-memory Transactional Data Grids. *ACM Transactions on Autonomous and Adaptive Systems*, 9(2):125–134, 2014.

[35] Diego Didona and Paolo Romano. Performance Modelling of Partially Replicated In-Memory Transactional Stores. In *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 265–274. IEEE, sep 2014.

[36] Diego Didona and Paolo Romano. Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning. *ACM/SPEC International Conference on Performance Engineering (ICPE)*, 2015.

[37] Diego Rughetti, Pierangelo Di Sanzo, Bruno Ciciani, and Francesco Quaglia. Analytical/ML mixed approach for concurrency regulation in software transactional memory. *Proceedings - 14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2014*, pages 81–91, 2014.

[38] Diego Didona and Paolo Romano. On Bootstrapping Machine Learning Performance Predictors via Analytical Models. In *ICPADS*, 2015.

[39] Nuno Diegues and Paolo Romano. Self-Tuning Intel Transactional Synchronization Extensions. *ICAC*, pages 1–11, 2014.