Towards White-Box Modeling of Hardware Transactional Memory Systems

Daniel Castro¹, Diego Didona², Paolo Romano¹

¹Lisbon University & INESC-ID, Portugal
²EPFL, Switzerland
Roadmap

• Motivation: Paolo’s HTM Hype Cycle
• Goals, Approach, Challenges
• Related work
• Reverse engineering Intel’s TSX
• A white-box model of TSX:
  – concurrency control
  – capacity
• Validation
Paolo’s HTM Hype Cycle

HTM will fix STM’s performance

Boom of TM: focus on STM

Multicore revolution ~2002

HTM turns mainstream: TSX, P8

HTM can shine!
- DBMS indexes
- auto-tuning
- security
- exploiting advanced TM features (e.g., P8 [EuroSys16])

Conception [ISCA93]

Innovation Trigger

Peak of Inflated Expectations

Trough of Disillusionment

Slope of Enlightenment

Plateau of Productivity

MAKE HTM’s PERFORMANCE MORE PREDICTABLE [this work]
Roadmap

• Motivation: Paolo’s HTM Hype Cycle
• Goals, Approach, Challenges
• Related work
• Reverse engineering Intel’s TSX
• A white-box model of TSX:
  – concurrency control
  – capacity
• Validation
Goals

• Improve our ability to:
  1. understand, and
  2. predict

performance of HTM implementations
  – both current and future ones

• Current work focuses on **TSX**
  – Work in progress on IBM’s POWER8
Approach

• **White-box** analytical model of HTM performance:
  – focus on performance dynamics due to:
    • capacity limitations
    • conflicts between transactions
    • impact of fallback path acquisition (global lock)

• Previous works focused on **black-box** models:
  – poor/no human interpretability
    • do not really contribute to deepen our understanding
  – limited extrapolation power
    • e.g., what if: capacity doubled? CPU had 1000 cores?
Challenges (1/2)

- White-box models require knowledge on internals of target system:
  - internal implementation of TSX is undisclosed
  - some preliminary studies do exist and help [PACT14, IPDPS14, MIT15]
  - but some relevant details are still unclear:
    - effective capacity for transactions that issue different mixes of loads/stores
    - resolution policy for transactional conflicts
Challenges (2/2)

• Tame the complexity of HTM implementations
• Models are an inherent approximation of reality
• The art of white-box performance modelling:
  – pick the “right” approximations
  – make the model simple enough to be:
    1. treatable and computable efficiently
    2. accurate enough to be useful in practice
Roadmap

• Motivation: Paolo’s HTM Hype Cycle
• Goals, Approach, Challenges
• Related work
• Reverse engineering Intel’s TSX
• A white-box model of TSX:
  – concurrency control
  – capacity
• Validation
Related work

• Ample literature on modeling of DBMS’s concurrency control performance:
  – both white- and black-box approaches
  – relevant differences:
    • no capacity limitations
    • no fallback path
• Several white-box models targeted STM, but:
  – different concurrency control scheme
  – no capacity limitations
  – no fallback path
Roadmap

• Motivation: Paolo’s HTM Hype Cycle
• Goals, Approach, Challenges
• Related work
• Reverse engineering Intel’s TSX
• A white-box model of TSX:
  – concurrency control
  – capacity
• Validation
What we know about TSX

Concurrency control
• conflicts are eagerly detected
  – non-transactional operations cause immediate abort of conflict transactions:
  – e.g., fallback path

Capacity
• stores maintained in L1
  – transactions that only issue sequential stores can achieve ~90% of max L1 capacity

• loads are not
  – transactions that only issue sequential load achieve ~50% of L3 capacity
  • way more than L1 & L2’s capacity
What we’d like to know about TSX

Concurrency control
• conflicts are eagerly detected
  – non-transactional operations cause immediate abort of conflict transactions:
  – e.g., fallback path
• Upon a conflict between two or more transactions:
  – which one is aborted?

Capacity
• stores maintained in L1
  – transactions that only issue sequential stores can achieve ~90% of max L1 capacity
  – L1 is private, so:
    • why not its whole capacity?
• loads are not
  – transactions that only issue sequential load achieve ~50% of L3 capacity
    • way more than L1 & L2’s capacity
    • why not whole L3 capacity?
  – what if transactions execute mixes of loads and stores?
Conflict resolution policy in TSX

• Simple experiment:
  – run two concurrently
  – inject properly tuned delays to cause:
    • read after write
    • write after write conflicts
    • write after write

• Conclusion:
  – “Last requester wins” policy
  – Spoiler: not the same for POWER8!
What we’d like to know about TSX

Concurrency control

• conflicts are eagerly detected
  – non-transactional operations cause immediate abort of conflict transactions:
  – e.g., fallback path

• Upon a conflict between two or more transactions:
  – which one is aborted?
    • “Last requester wins”

Capacity

• stores maintained in L1
  – transactions that only issue sequential stores can achieve ~90% of max L1 capacity
    – L1 is private, so:
      • why not its whole capacity?

• loads are not
  – transactions that only issue sequential load achieve ~50% of L3 capacity
    • way more than L1 & L2’s capacity
    • why not whole L3 capacity?
  – what if transactions execute mixes of loads and stores?
Actual store capacity

• Nguyen [MIT15] hypothesized the presence of some transactional metadata in L1:
  – how large is this metadata? 10% of the L1 cache?

• We seek an answer by:
  – simulating an L1 with the same geometry as our platform:
    • 512 cache lines, 8-way associativity (Haswell Xeon V3, 4 cores)
  – placing metadata in X random cache lines
  – emulating a tx that issues sequential stores starting from a random address
  – reporting a capacity if we evict a metadata or a cache line written by the transaction
Actual store capacity

- Nguyen [MIT15] hypothesized the presence of some transactional metadata in L1:
  - how large is this metadata? 10% of the L1 cache?

~3 lines are occupied by transactional meta-data
What we’d like to know about TSX

Concurrency control
• conflicts are eagerly detected
  – non-transactional operations cause immediate abort of conflict transactions:
    – e.g., fallback path
• Upon a conflict between two or more transactions:
  – which one is aborted?
    • “Last requester wins”

Capacity
• stores maintained in L1
  – transactions that only issue sequential stores can achieve ~90% of max L1 capacity
  – Why not its whole capacity?
    • Transactional metadata
• loads are not
  – transactions that only issue sequential load achieve ~50% of L3 capacity
    • way more than L1 & L2’s capacity
    • why not whole L3 capacity?
  – what if transactions execute mixes of loads and stores?
Capacity with load/store mixes

- **Experiment:**
  - Tx accesses (cache aligned) addresses selected unif. at rand.
  - each access is a store (resp. load) with prob. $P_W$ (resp. $1-P_W$)
Capacity with load/store mixes

- Key observation:
  - when 50% of accesses are loads, capacity does not double
  - only ~10% increase on average
Capacity with load/store mixes

• **Key observation:**
  – when 50% of accesses are loads, capacity does not double
    • only ~10% increase on average

• **Hypothesis:**
  1. loads can trigger evictions in L1:
     • L1 eviction of written cache lines:
       ➔ capacity abort
     • L1 eviction of read cache lines:
       ➔ safe, metadata stored elsewhere
  2. for non-minimal values of $P_w$ the effective capacity is largely determined by evictions in L1:
     • intuition: L3 is 256x larger than L1
Capacity with load/store mixes

- We use, again, simulation to validate our hypothesis & approximation.

- They hold for write prob. as small as 0.1%!

- Consequences:
  - for modelling purposes we **do not care** where/how TSX stores metadata of loaded cache lines
  - models considering only L1 will have good accuracy
What we’d like to know about TSX

Concurrency control
• conflicts are eagerly detected
  – non-transactional operations cause immediate abort of conflict transactions:
    – e.g., fallback path
• Upon a conflict between two or more transactions:
  – which one is aborted?
    • “Last requester wins”

Capacity
• stores maintained in L1
  – transactions that only issue sequential stores can achieve ~90% of max L1 capacity
  – Why not its whole capacity?
    • Transactional metadata
• loads are not
  – transactions that only issue sequential load achieve ~50% of L3 capacity
    • way more than L1 & L2’s capacity
    • why not whole L3 capacity?
  – transactions executing mixes of loads and stores are constrained by L1
Roadmap

- Motivation: Paolo’s HTM Hype Cycle
- Goals, Approach, Challenges
- Related work
- Reverse engineering Intel’s TSX
- A white-box model of TSX:
  - overview
  - concurrency control
  - capacity
- Validation
Key parameters and assumptions

• θ threads running concurrently on different physical cores:
  – no Hyper-threading

• After B aborts, the fallback path (global lock) is activated

• Key workload characteristics:
  – Interleaving of transactional/non-transactional code
  – Transaction length (# accesses) and duration
  – Transaction data access patterns

• Target KPIs:
  – avg. throughput/execution time (including aborted retries)
  – abort probability: contention, fallbacks, capacity
Key parameters and assumptions:
Tx/non-tx code blocks

- Threads execute either transactional or non-transactional code block with probability $P_t$

Data race freedom:
- Transactional and non-transactional code access disjoint data
- 1 notable exception: the fallback lock
Key parameters and assumptions: Transaction length and duration

- Transactions perform, on average, L accesses, each mapping to a different cache line
  - if multiple accesses map to the same cache line, only the first is accounted for
- Duration of transactions is exponentially distributed with mean value C
  - C/L: avg. time between two memory accesses
  - 1/C << hardware timer interrupt frequency
Transaction data access patterns

• Tx accesses are uniformly distributed across D granules:
  – each granule has the size of a cache line
  – non-uniform access patterns can be approximated via an “equivalent” (smaller) uniform one [TAS14]

• Each access is a store, resp. load, with probability $P_w$, resp. $1-P_w$
Modelling execution of threads (1/3)

• At any point in time, the state of the system can be described as follows:
  – \( tx^i \): #threads running transactions with \( i \in [1,B] \) retries left
  – \( nt \): #threads executing non-transactional code
  – \( fb \): #threads in the fallback path

where \( \sum_{i=1..B} tx^i + nt + fb = \theta \) // tot thread count

• We encode the system’s state via tuples of the form:
  \(<tx^B, ..., tx^i, ..., tx^1, nt, fb>\)

• Each state:
  – has a different abort probability
  – produces a different throughput
Modelling execution of threads (2/3)

• ...and model execution via a Markov Chain:

• whose transition rates depend on source states’:
  • throughput
  • abort probability
  • fallback path activation probability
Modelling execution of threads (3/3)

• The use of a Markov Chain (MC) allows for simplifying modelling:
  – target KPIs can be computed on a state-by-state basis:
    • thus focusing on a simpler case
  – next, the stationary distribution, $\vec{\pi}_s$, of the MC can be computed:
    • probability to be in each state of the MC
  – finally, the KPIs for the whole system are obtained as the average of the KPIs in each state weighted by the probability of each state, e.g.:
    \[
    p_a = \sum_{s \in S} \vec{\pi}_s p_{a,s}
    \]
Roadmap

• Motivation: Paolo’s HTM Hype Cycle
• Goals, Approach, Challenges
• Related work
• Reverse engineering Intel’s TSX
• A white-box model of TSX:
  – overview
  – concurrency control
  – capacity
• Validation
Modelling transaction conflicts

- Avg. frequency of conflicts for a tx at its i-th operation:

\[ H(i) = \frac{(\theta^t - 1)}{W} \frac{i}{D} (1 - (1 - P_W)^2) \]

Every \( W = C/L \) time units, the remaining \( \theta^t - 1 \) transactional threads access an item

This access must target one of the \( i \) items already accessed by the transaction

To generate a conflict, at least one of the two accesses must be a write

- Probability of reaching operation i, \( P_R(i) \), is computed recursively:

\[ P_R(i) = P_R(i - 1)(1 - e^{-H(i-1)C/L}) \]
Modelling aborts due to fallbacks

- When a transactions with 1 retry left aborts due to a conflict, it will cause the abort of any other concurrent transaction
- We model this by:
  - first computing abort probability w/o fallbacks
  - correcting the frequency of conflicts:
    \[ H^n(i) = H(i) + d\mu_t\alpha. \]
    \[ H^d(i) = H(i) - d\mu_t\alpha. \]
  - computing again the abort probability

• The execution time of a TCB, i.e., the cost of executing a TCB, is computed as the weighted average that corresponds to the abort case.
Roadmap

• Motivation: Paolo’s HTM Hype Cycle
• Goals, Approach, Challenges
• Related work
• Reverse engineering Intel’s TSX
• A white-box model of TSX:
  – overview
  – concurrency control
  – capacity
• Validation
Modelling Capacity Aborts
Write-only workloads ($P_W=1$)

• Modelled as a ball into bins problem:
  – $C$-associative cache with $B$ sets $\Rightarrow B$ bins, each with capacity $C$

• Compute probability that at least a bin is full after $i$ balls.

• Different sequences of $I$ ball throws w/o causing any bin overflows (up to $C$ ball in each bin):

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

- $\left\lfloor \frac{I}{C} \right\rfloor$ all balls hit the same bin, until this is full
- no. ways in which we can choose $x$ bins out of $B$
- we are left with:
  - $I-xC$ balls
  - $B-x$ bins, not filled
- no. ways in which we can throw $Cx$ balls filling $x$ bins
Modelling Capacity Aborts
Write-only workloads ($P_W=1$)

- Cast to a ball into bins problems:
  - $C$-associative cache with $B$ sets $\rightarrow B$ bins, each with capacity $C$
- Compute probability that at least a bin is full after $i$ balls.
- Different sequences of $I$ ball throws w/o causing any bin overflows (up to $C$ ball in each bin):

$$N_{B,C,I} = \sum_{x=\text{min}_c}^{\text{max}_c} \binom{B}{x} \prod_{y=0}^{x-1} \binom{I-yC}{C} \times N_{B-x,C-1,I-xC}$$

- Probability that at least one bin overflows after $I$ balls:

$$P(c \leq I) = 1 - \frac{N_{B,C,I}}{B^I}$$
Modelling Capacity Aborts Mixed read/write workloads

• As already discussed, models can focus only on L1 dynamics for $P_w > 0.1$
• But the exact computation is more complex with read/write “balls”:
  – both mathematically and computationally
  – we propose an approximate approach

Details in the paper
Roadmap

• Motivation: Paolo’s HTM Hype Cycle
• Goals, Approach, Challenges
• Related work
• Reverse engineering Intel’s TSX
• A white-box model of TSX:
  – overview
  – concurrency control
  – capacity
• Validation
Validation

• Based on Xeon E3-1275 v3 running at 3.5GHz (Haswell), 4 physical cores

• Capacity:
  – conflict free workload, single threaded

• Conflicts:
  – short transactions, not to cause capacity except.

• In both tests we generate uniformly distributed accesses over data sets of size D
This section reports the results of a validation study that compares the probabilities of having a capacity abort exactly at operation completion. The plot shows that the model is able to predict well the probability of a capacity abort as a function of the number of transaction reads and writes. The model in predicting both the throughput and abort probability of the input to the model. In order to stress its prediction capabilities, we have introduced in Section 4.4 to obtain a closed form solution for the probability of a capacity abort as a function of the number of transactions and can be computed recursively setting
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
\]
where \( P_{W} \) is the probability that a transaction issues a write operation with probability \( P_{W} \), and can be computed by expressing it
\[
P_C^{I} = \sum_{c=1}^{C} P_{W}^{c} \quad \text{for } I = 1, \ldots, \infty,
Conflicts (and fallback)

MAPE = 8.12%, R = 0.9989.

- No. threads = \{1,2,3,4\}
- Retry budget = \{2,4,6\}
- # accesses in a tx = \{2,5,10,20\}
- Data set size = \{512, 2048, 8192, 32768\}
- Prob. that an access is a write = \{0.5, 1.0\}
Conclusions and future work

• First analytical model of HTM
  – focus on capacity, conflicts and fallback
  – based on empirical validation of hypothesized system behavior

• Work ahead:
  – Validation with complex benchmarks (STAMP) and larger parallel machines
  – Approximate/more scalable analytical model of contention
  – Modelling IBM’s POWER8
    • scalability analysis up to 1000 cores!
Q&A