A Comprehensive Study of Conflict Resolution Policies in Hardware Transactional Memory

Ege Akpınar, Sasa Tomic, Adrian Cristal, Osman Unsal, Mateo Valero

TRANSACT 2011
Background Eager HTM

• Conflict resolution especially critical in eager-versioning, eager-conflict management HTM implementations since these implementations are typically optimized for commit and therefore assume that conflicts are rare

• Although conflict resolution policy plays a vital role in performance, there is not a commonly accepted policy yet
Objective

- This paper aims to
  - Evaluate existing conflict resolution policies
  - Identify and remedy performance bottlenecks that occur common during transactional executions
  - Propose new policies based on identified performance bottlenecks
  - Carry out a general comparison of existing and proposed conflict resolution policies
Progress – Performance Bottlenecks

• 5 existing performance issues (as mentioned in *Performance Pathologies* paper) and two new bottlenecks are described
  - Livelock
  - Deadlock
  - FriendlyFire
  - StarvingElder
  - FutileStall
  - InactiveStall
  - CascadingStall

• Remedies are proposed
Progress – Performance Bottlenecks

Deadlock

Livelock
Progress – Performance Bottlenecks

(from Pathologies paper)

FriendlyFire

FutileStall

StarvingElder

Tx 1

Tx 2

Tx 1

Tx 2

Tx 1

Tx 2

Tx 3

Tx 4

X

X

X

X

C

C

C

C
Progress – Performance Bottlenecks

(Contributions of this paper)

InactiveStall

CascadingStall
Progress – Perfect waiting and stalling

• **Perfect waiting**: Ideal backoff algorithm where a transaction waits precisely until all conflicting transactions terminate.

• **Perfect stalling**: When a transaction (Tx1) gets aborted (due to Tx2), it restarts execution and starts waiting when it encounters a conflict with the same transaction (Tx2).
Progress - Methodology

- STAMP benchmark suite is used
- Number of ticks spent during parallel sections are calculated (number of ticks spent after the first transaction to enter that section until the last transaction to exit that section)
- Speedup is calculated using number of ticks compared to that of single core executions
Baseline policy

- Very simple, can be achieved by small modifications to cache protocol
- A transaction that fails to retrieve a cache line (likely because it has already been retrieved by another transaction) aborts itself
**Timestamp policy variations**

**Timestamp**: Time when a transaction begins execution

- A transaction’s timestamp is maintained until commit (thus, it doesn’t reset after abort or stall)
- Comparison of timestamps yield which transaction is older/younger (has begun earlier/later)
Timestamp policy variations

• 5 timestamp policies are tested

• Variations tackle
  - Deadlock
  - Livelock
  - StarvingElder
  - InactiveStall
  - CascadingStall
  - FriendlyFire

• Perfect stalling improved results significantly
Size policy variations

Size : Summation of read-set and write-set
• If an element is present in both read-set and write-set, it contributes twice to size
• Size is a good indicator of amount of work done since work typically consists of memory accesses (reads/writes)

Largeness factor: A transaction is deemed larger than another only if its size exceeds the other transaction’s size by a largeness factor
Size policy variations

- 3 categories of policies are tested

- Variations tackle
  - Deadlock
  - Liveloop
  - StarvingElder
  - FutileStall
  - InactiveStall
  - CascadingStall
  - FriendlyFire

- A largeness factor of 1.25 performed the best

<table>
<thead>
<tr>
<th></th>
<th>Tx 1</th>
<th>Tx 2</th>
<th>Tx 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>A</td>
<td>Write X</td>
<td>Read C</td>
</tr>
<tr>
<td>Read</td>
<td>B</td>
<td>Write X</td>
<td>Write D</td>
</tr>
<tr>
<td>Write</td>
<td>A</td>
<td>Write X</td>
<td>Read E</td>
</tr>
<tr>
<td>Read</td>
<td>C</td>
<td>Write C</td>
<td>Write F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Write G</td>
</tr>
<tr>
<td>Size</td>
<td>4</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

BSC - Microsoft Research Centre
Prioritization policy variations

• Prioritization is based on
  - Number of stalled transactions (by a transaction)
  - Number of aborted transactions (by a transaction)

• Primarily aims to avoid bottlenecks FutileStall and CascadingStall
Prioritization policy variations

- 5 prioritization policies are tested
- Variations tackle
  - Deadlock
  - Livelock
  - InactiveStall
  - FutileStall
  - CascadingStall
- Results are highly variable among different applications
Alternating Priorities Policy

**Alternating Priorities**: Transactions alternate priorities in pairs. Eg. Tx1 gets aborted by Tx2. When they conflict again, Tx1 will abort Tx2.

- Designed for fairness, rather than performance
- Measured performance and scalability is good
Results

• Overall, performance increase (from baseline policy) of 5-10% was measured, amounting up to 15%
Conclusion

- Conflict resolution is a vital characteristic for performance.
- Taking into common performance bottlenecks into consideration has an important effect on performance.
- It is difficult to identify a single resolution policy as the globally best performer since performance varies greatly with application characteristics.
- Transactional Memory will be realized only if its performance promises are solid; therefore, conflict resolution is an important research space.
Timestamp policies

- Timestamp1: At conflicts, older transactions abort the younger ones and carry on execution (no stall).
- Timestamp-2: At conflicts, older transactions start waiting on younger transactions (stall). However, when a younger transaction requests a cache line that is owned by an already stalled older transaction, younger transaction aborts itself (in order to avoid deadlock) and begins perfect waiting.
- Timestamp-3: (StarvingElder remedy) For every transaction, a "committed after me" list is maintained. When a transaction commits, that transaction’s thread is added to the "committed after me“ list of all transactions that are active, aborted or stalled. A transaction’s "committed after me" list is reset after its every commit. At conflicts, transactions whose threads are present in "committed after me" list of conflicting transactions are aborted.
- Timestamp-4: Same as Timestamp-3 configuration except for its InactiveStall policy. Instead of aborting the younger transaction, older transaction is aborted at InactiveStall case.
- Timestamp-5: Naïve timestamp and stall policy (Timestamp-2) with perfect stalling enhancement.
Size policies

- Size-1 At conflicts, larger transactions take over and smaller conflicting transactions are aborted. There is no stalling.
- Size-2 At conflicts, larger transactions are favored. However, at conflicts, owner of the conflicting cache line is allowed to resume execution regardless of its size. When a transaction requests a cache line from a larger stalled transaction, it aborts itself and restarts execution. When the aborted transaction again conflicts with the same larger transaction, it is stalled (perfect stalling).
- Size-3 At conflicts, larger transactions are favored. However, at conflicts, owner of the conflicting cache line is allowed to resume execution regardless of its size. When a transaction requests a cache line from a larger stalled transaction, small transaction aborts itself and starts perfect waiting.
Priority policies

- Priority-1 Transactions gain priority when they stall other transactions. When a transaction requests to acquire a cache line that is already acquired by another transaction and if their priorities are equal, then the current owner is allowed to continue execution.

- Priority-2 Transactions gain priority as they stall other transactions. In addition, transactions lose priority as they abort other transactions.

- Priority-3 Similar to Priority-2. However, transactions gain (not lose) priority as they abort other transactions.

- Priority-4 Similar to Priority-1. At conflicts, if a transaction loses, it aborts instead of stalling.

- Priority-5 Similar to Priority-1, transactions gain priority as they stall others. However, priority is a measure calculated using the number of transactions in conflict. For instance, when a transaction gets stalled due to a conflict with n transactions, all n transactions gain 1/n priority.