Hardware Support For Relaxed Concurrency Control In Transactional Memory

Utku Aydonat
uaydonat@eecg.toronto.edu

Edward S. Rogers Department of Electrical and Computer Engineering
University of Toronto

MICRO-43
2010
Overview

- Transactional Memory (TM) eases writing parallel programs by eliminating user-locks.

- TM performance is not satisfactory for applications with long-running transactions and high data sharing.
  - The two-phase locking (2PL) algorithm is too conservative, i.e., it causes unnecessary aborts and delays.

- We describe a more relaxed algorithm.
  - It can efficiently be implemented in hardware using simple components and it does not require changes to the cache coherence protocols.
  - It reduces aborts which leads to performance improvements.
Outline

- Transactional Memory
  - Two-Phase Locking

- Relaxed Concurrency Control
  - The SON Algorithm

- Hardware Support

- Experimental Evaluation

- Conclusions
Transactional Memory

- Traditionally, locks are used to protect shared variables in critical sections.

\[
\text{shared } A, B; \\
\ldots \\
C = A + B; \\
A = C + 1; \\
B = C + 2; \\
\ldots
\]

- Difficult to program and is error-prone

\[
\text{Lock}(A); \text{Lock}(B); \\
C = A + B; \\
A = C + 1; \\
B = C + 2; \\
\text{Unlock}(B); \text{Unlock}(A);
\]

- TM system guarantees consistent reads and writes

\[
\text{transaction} \{ \\
C = A + B; \\
A = C + 1; \\
B = C + 2; \\
\}
\]
Transactional Memory

- The TM system keeps track of the shared data accesses and makes sure that reads and writes are consistent.
  - If inconsistency is plausible, it can abort or delay transactions.

- Inconsistencies are caused by conflicts.
  - A conflict occurs when two transactions access the same datum and at least one of them writes to it.

- The traditional TM model: every single conflict causes aborts or delays.
No matter which design policies are used, this code results in delays and/or aborts in typical TM systems.
Traditional Conflict Handling

- This mechanism is referred to as **Two-Phase Locking**.
- Conflicting accesses are not allowed between active transactions.
- If there are no conflicts, the execution is equivalent to a serial execution.
Two-Phase Locking is Conservative

- Even if there are conflicts, the execution can still be equivalent to a serial execution.
- The property is called **Conflict-Serializability (CS)**.
Conflict-Serializability

- CS provides better concurrency than Two-Phase Locking.
  - Some conflicting transactions can commit successfully.

- CS is harder to implement efficiently.
  - Straightforward implementations can cause overheads in TM systems.

- We propose the use of an algorithm which can be efficiently implemented in TM systems.
Each transaction is assigned a number (SON) based on its conflicting actions.

- For each conflict, the transaction that performs the action earlier gets a smaller SON than the other conflicting transaction.
Serializability Order Numbers (SONs)

SON(TX1) < SON(TX3)
SON(TX2) < SON(TX3)
SON(TX3) < SON(TX4)
SON(TX3) = [0:∞)
SON(TX3) = (SON(TX1):∞)
SON(TX3) = (SON(TX2):∞)
SON(TX3) = (SON(TX2):SON(TX4))
The SON Algorithm

- Each transaction keeps a lower bound and an upper bound of SON.
- At commit time, a transaction selects an SON within its bounds.
- Conflicts are recorded as they occur, but transactions are not aborted.
- Conflicts cause lower and upper bound updates when one of the conflicting transactions commit.
- The transaction aborts if the bounds become invalid, i.e. lower bound > upper bound
Problem: There can be active transactions serialized before committed transactions.

How to serialize transactions according to the conflicts with already committed transactions?
Solution: Store the SONs with shared addresses using *write-numbers* and *read-numbers*.
We implement the SON algorithm on top of a standard base TM system with *eager conflict-detection* and *lazy versioning*. 
### SONTM Hardware Components

#### Main Memory
- Write-Numbers Table

#### Shared L2
- L1I/D
- Log

- L1I/D
- Log

#### Processor#0
- Register File
- Checkpoint Register File
- PC
- BACKOFF
- Read Set
- Write Set
- lb, ub, son
- dst-sas, dst-sal, dst-las
- Read History Table

#### Processor#1
- Register File
- Checkpoint Register File
- PC
- BACKOFF
- Read Set
- Write Set
- lb, ub, son
- dst-sas, dst-sal, dst-las
- Read History Table
**SONTM Hardware Components**

**Registers: lb, ub, son:**
- They keep the lower bound, the upper bound and the SON of a transaction.
SONTM Hardware Components

**Conflict Flags:**
- When a conflicting access is issued, the transaction records the conflict but does not abort.
  - The processor that issues the request receives a NACK message.
- Three bit-vectors are used.
  - \textit{dst-sas} (store-after-store)
  - \textit{dst-las} (load-after-store)
  - \textit{dst-sal} (store-after-load).
**SONTM Hardware Components**

**Write-Number Table:**
- It stores write-numbers in cacheable memory.
- Write-numbers are fetched to validate each load and store instruction.
- Write-numbers are updated at commit time with the son of the committing transaction.
**SONTM Hardware Components**

**Read History Table:**

- It is not efficient to keep a global read-numbers table in memory.
  - It requires iterating over the read-set at commit time to update the read-numbers.
  - This would introduce significant overheads in SONTM.

- Read-numbers are distributed over processors.
Instead of updating a global read-numbers table, the read-set of the committed transaction is saved at each processor.
SON Broadcast:

- It enables serialization of conflicting transactions.

- Committing transaction broadcasts its SON to all the other processors.

- Conflicting processors update their $lb$ or $ub$ based on the type of the conflict.
Transactional Load

(1) write-number request.
(2) write-number in the cache.
(3) $lb$ update with the write-number.
(4) read-set update.
(5) NACK message due to a conflict.
(6) conflict recorded in dst-las.
Commit

(1) write-number request.
(2) read-number request.
(3) write-number in the cache.
(4) read-number received.
(5) lb update with the write number and the read-number.
(6) SON is selected.
(7) broadcast of SON.
(8) update write-numbers.
(9) commit to memory.
Experimental Evaluation

- We used the Simics full-system simulator with Gems toolset to model memory.
  - 8-core CMP with in-order SPARC processors.
  - 32 KB Private L1 Cache, 8 MB Shared L2 Cache.
- We built on top of the EL HTM implementation of Jayaram Bobba et. al, ISCA ’07
- Benchmarks:
  - They consists of a linked-list and STAMP benchmarks.
  - The size of transactions and their contention vary.
- Measurements:
  - Total execution cycles.
  - Abort rate (# of aborted / # of committed).
Simulated HTM Systems

- **2pl-EL**: The base system that uses *Two-Phase Locking* with *eager* conflict detection and *lazy* versioning.

- **2pl-LL**: A version of the base system with *lazy* conflict detection and *lazy* versioning.

- **ideal**: The ideal implementation of SONTM.
  - No overheads for accessing write-numbers and read-numbers, and broadcast messages.
Simulated HTM Systems

- **CO**: an ideal implementation of commit ordering algorithm.
  - It is implemented by DATM that requires a new cache coherence protocol.

- **sontm**: The SONTM with all its features and their associated overheads.
  - The size of the Write-Number Table is 64KB.
  - The read history table consists of 4 perfect read-sets with no false positives at each core.
  - Loading and updating the write-numbers cause contention on the bus and increase the cache misses.
  - Commits take longer due to broadcast messages, write-number updates and read-number requests.
Normalized Total Execution Cycles

2pl-EL  2pl-LL
34%  32%  66%  57%  93%

list  bayes  vacation  labyrinth  yada  intruder  genome  kmeans  ssca2

Legend:
- 2pl-EL
- 2pl-LL
- co
- ideal
- sotnm
Related Work

- **Conflict-Serializability in Transactional Memory:**
  - Our TSTM is the first TM system that uses relaxed concurrency algorithm to reduce the abort rates [AyAb08, AyAb09].
  - TSTM was later followed by DATM [Ram08] and DASTM [Ram09] that use the commit ordering algorithm.
  - DATM requires changes to the cache coherence protocol.

- **Relaxing Serializability in Transactional Memory:**
  - Lazy snapshot isolation [Rie06] relaxes serializability with multi-versioning.
  - It does not provide serializability.
Conclusions

- Traditional Two-Phase Locking algorithm aborts or delays transactions at every conflict.

- We implemented a relaxed algorithm in hardware using only simple components, such as filters, flags, and broadcast.

- This simple implementation introduces overheads, but the performance is improved by 29% on average over nine benchmarks.
  - Benchmarks with long-running transactions and high abort rates benefit the most.
  - Overheads impact the performance for benchmarks with short-running transactions and low abort rates.
Future Work

- Evaluating adaptive concurrency control.
  - adjusts the algorithm according to the application characteristics separately at each processor.

- The impact of more processors on performance.

- The impact of read history table types and their sizes on performance.
  - We expect the impact not to be significant.
  - Can we remove the read history tables all together?
Thank you.