J. Parallel Distrib. Comput. 104 (2017) 73–87

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Enhancing scalability in best-effort hardware transactional memory systems

Ricardo Quislant *, Eladio Gutierrez, Emilio L. Zapata, Oscar Plata

Department of Computer Architecture, University of Malaga, Spain

HIGHLIGHTS

- Hardware irrevocability with overflow anticipation and transaction stalling.
- Two-phase abort to allow certain privileged mode code inside transactions.
- Allow asking for irrevocability if privileged code evicts a transactional block.
- Privileged-aware cache replacement policy to favour transactions over privilege code.
- Evaluation of requester-wins and requester-stalls conflicts resolution policies.

ARTICLE INFO

Article history:
Received 10 June 2016
Received in revised form
22 December 2016
Accepted 4 January 2017
Available online 16 January 2017

Keywords:
Hardware transactional memory
Best-effort
Irrevocability
Privileged mode code
Cache replacement policy
Requester-wins
Requester-stalls

ABSTRACT

Current industry proposals for hardware transactional memory focus on best-effort solutions where hardware limits are imposed on transactions. These designs can efficiently execute transactions but they may abort due to different hardware and operating system limitations, with a significant impact on performance. For instance, transactions cannot survive capacity overflows, exceptions, interrupts, operating system events like page faults, migrations, context switches, and so on. To deal with these events, best-effort hardware transactional memory systems usually provide a software fallback path to execute a non-transactional version of the code.

In this paper we propose hardware implementation solutions to make transactions survive some of such limitations, in order to improve the performance and scalability of transactional applications in best-effort systems. First, we propose a hardware irrevocability mechanism that works either when hardware capacity overflows occur or in high contention scenarios. It anticipates capacity overflows and reduces the abort count. This mechanism avoids writing a fallback code, simplifying the programming of the transactional application. Second, we propose a two-phase abort mechanism to support both the execution of privileged mode code inside transactions and the interaction of this code with the irrevocability mechanism. Third, we propose a privileged-aware cache replacement policy to reduce capacity overflows in the presence of privileged code.

We evaluate our proposals with all the benchmarks of the STAMP transactional suite and carry out a performance comparison with a fallback-based hardware transactional memory system, after considering different fallback codes, showing significant performance benefits for requester-wins and requester-stalls conflict resolution policies.

1. Introduction

Transactional Memory (TM) [16] was first presented in 1993 [17] as a non-blocking synchronization mechanism for shared memory chip multiprocessors (CMPs). TM provides the programmer with the *transaction* construct that executes the code enclosed by it atomically and in isolation. The underlying system is in charge of maintaining those transactional properties with dedicated hardware and changes to the coherence protocol (hardware TM, or HTM).

Since Herlihy's seminal approach there have been several proposals exploring different designs of an HTM system [4,9,15,26], and many others that have gained insight into the virtualization of the transactional hardware [4,6,31,37] to support transactions of any size and duration.

* Corresponding author.
E-mail addresses: quislant@uma.es (R. Quislant), eladio@uma.es (E. Gutierrez), zapata@uma.es (E.L. Zapata), oplata@uma.es (O. Plata).

http://dx.doi.org/10.1016/j.jpdc.2017.01.002
0743-7315/

However, it is not until recently that some processor manufacturers have included HTM support in their commercial off-the-shelf CMPs [7,19,35,38]. Current industry proposals focus on best-effort solutions (BE-HTM) where hardware limits are imposed on transactions. These designs can efficiently execute transactions but they may abort due to different hardware and operating system (OS) limitations, with a significant impact on performance. For instance, transactions cannot survive capacity overflows, exceptions, interrupts, OS events like page faults, migrations, context switches, and so on. To deal with these events, BE-HTM systems usually provide a software fallback path to execute a non-transactional version of the code, often comprising a global lock.

In this paper we propose three hardware mechanisms to help transactions running in BE-HTM systems survive some hardware and OS limitations and to enhance their performance. Our proposals are the following:

- A hardware irrevocability mechanism that operates whenever a transaction aborts a given number of times, either because of conflicts with other concurrent transactions or due to hardware capacity overflows (a BE-HTM system relies on caches to store transactional information and a transactional cache block replacement implies losing such information). Irrevocability [6,36] is a transactional execution mode that ensures transaction forward progress. We show how this mechanism can be easily implemented by tailoring the coherence protocol, and discuss its benefits over a software fallback path in terms of performance and ease of use. We study and evaluate irrevocability in the context of eager–eager requester-wins/stalls BE-HTM systems. Our irrevocability mechanism is able to anticipate a cache block replacement and to stall other concurrent transactions instead of aborting them.
- We propose a two-phase abort mechanism to support the execution of privileged mode code within transactions, such as that of exceptions or interrupts. Under this mechanism, the release isolation part of the abort (first phase) is separated from the part of restoring the transaction's checkpoint (second phase). The first phase is executed in case the privileged mode code replaces a transactional block, although the irrevocability mechanism is first tried to prevent the transaction from aborting. The second phase is executed when the privileged mode code execution comes to an end. We rely on a transaction-aware OS to enforce certain invariants that ensures correctness.
- Since privileged mode code may interfere with ongoing transactions by replacing transactional cache blocks that might abort them, we propose a privileged-aware (PA) cache replacement policy that reduces the number of privileged-mode-accessible blocks, maximizing the number of blocks devoted to transactions. It proves to work well for the benchmarks that exhibit more aborts due to privileged mode code cache replacements.

We evaluate our proposals in a simulation environment with all the benchmarks of the STAMP transactional suite and carry out a performance comparison with a fallback-based HTM system, after considering different fallback codes, showing significant performance benefits for certain benchmarks.

The remainder of the paper is organized as follows. Section 2 describes the baseline architecture design of the BE-HTM system that we have used to implement our proposals. Section 3 presents the details of the hardware irrevocability mechanism and its implementation. In Section 4 we discuss how privileged mode code can be allowed within transactions. Section 5 describes the privileged-aware replacement policy to decrease the abort count due to cache block replacements when running privileged mode code. In Section 6 we discuss the experimental results obtained when evaluating our proposals using the Simics/GEMS simulator and the STAMP benchmark suite. Finally, Section 7 reviews the related work and Section 8 draws the conclusions.

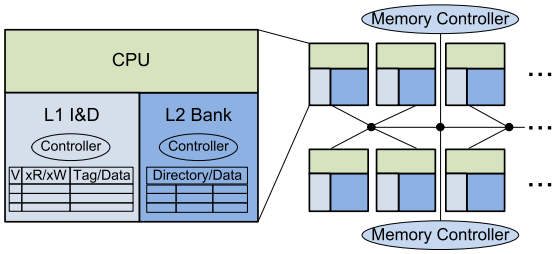

Fig. 1. Baseline architecture of the BE-HTM system.

2. Baseline BE-HTM architecture

The solutions proposed in this work to reduce the impact on performance of the limitations inherent to a BE-HTM system were designed on a baseline architecture similar to that provided by the state-of-the-art processor manufacturers, such as Intel Haswell [38] and IBM Power8 [1].

Fig. 1 shows the baseline architecture. The system relies on the L1 caches to store new transactional values of memory blocks, while old values are kept into the L2 cache. A pair of read and write transactional bits per L1 cache block marks whether the block was read or written within a transaction. Such bits can be flash-cleared on transaction commit and abort. In case of abort, the blocks whose transactional write bit is set are also invalidated. The cache coherence protocol maintains strong isolation [22] and implements an eager conflict detection policy. As to the conflict resolution policy, we consider two of them: (i) Requester-wins, where the requesting transaction wins the conflict and the requested one is aborted. This is the usually implemented policy in BE-HTM systems because of its simplicity; (ii) Requester-stalls, where the requesting transaction is nacked and stalled until the conflict vanishes. The stalled transaction aborts if it can cause a deadlock cycle because of other transactions stalling on it, similar to LogTM [26].

Causes of transaction aborts due to hardware/OS limitations are: a replacement of a transactional block in an L1 cache; an eviction of an L2 cache block that is marked as transactional in the L1 cache, due to the inclusion property of the cache coherence protocol; a change from user mode to privileged mode which can be caused by a hardware interrupt, a migration, a context change, a paging event, a system call, …mainly, OS events. To ensure forward progress of concurrent transactions, the user can provide a fallback code that executes whenever a transaction aborts a given number of times. We propose an alternative hardware irrevocability mechanism in Section 3 that entails certain benefits over the user fallback code.

The cache coherence protocol is a MESI directory protocol that holds a full bit vector of sharers. An L1 cache miss can cost up to four hops, although the last one is not in the critical path: a GET message from the requester to the directory, a FWD (forward) message from the directory to the L1 cache that owns the block, a message with the data between L1 caches (from forwarded to requester), and a message from the requester to unblock the directory, which stays in an intermediate state to forbid the access to the block while the miss is served [33]. The baseline protocol needs some modifications to support the execution of transactions:

- *Backup on first transactional store*: If an L1 cache block is in M state and its write transactional bit is not set, the L1 cache has to send the data to the L2 cache before a transactional store is performed. This way the L2 cache holds the last old value for the block.
- *Abort on evictions*: The replacement of a transactional block in an L1 cache implies losing track of transactional loads and stores, which jeopardizes transaction isolation; therefore, transactions must be aborted on these types of evictions. Besides, L2 cache block replacements may abort a transaction because of the inclusion property.