PUNO: Predictive Unicast and Notification towards Efficient Transactional Memory Execution

Lihang Zhao[1], Lizhong Chen[2], Jeffrey Draper[1]
[1]Information Sciences Institute, [2]Ming Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, CA
{lihangzh, lizhongc}@usc.edu, draper@isi.edu

ABSTRACT

Hardware Transactional Memory (HTM) schemes usually piggyback onto the cache coherence protocol to detect data access conflicts between transactions. We identify an intrinsic mismatch between the coherence scheme and transaction execution which causes a sizable amount of unnecessary transaction aborts. This pathological behavior is called false aborting and increases the amount of wasted computation and on-chip communication, manifesting itself as a performance and energy pitfall in HTM designs. For the TM applications we examined, 41% of the transactional write requests incur false aborting. We propose *Predictive Unicast and Notification* (PUNO), a novel hardware mechanism to combat false aborting. PUNO reduces transaction aborts by 61% and network traffic by 32% in workloads representative of future TM applications. The improvement is achieved with a meager 0.41% area overhead.

1. INTRODUCTION

Chip multiprocessor architectures are ubiquitous in today's high performance computing systems. 84.6% of the Top500 supercomputers use processors with six or more cores [1]. To increase the programmability of chip multiprocessors, Transactional Memory (TM) shifts the burden of synchronizing shared memory accesses from the programmer to either the software runtime or hardware. The Hardware Transactional Memory (HTM) approach implements hardware support for accelerated transaction execution. Extensive research in the past decade has paved the way for HTM to be implemented into commodity microprocessors [3, 2]. As of June 2013, HTM-enabled microprocessors have been deployed in four of the Top10 supercomputers [1]. HTM typically implements contention management to detect and resolve data access conflicts. The majority of HTM designs [6, 5], including commercial implementations (e.g., IBM System z [4]), piggyback onto the coherence protocol (typically directory-based) for conflict detection. However, there is an intrinsic difference between the cache coherence scheme and transaction execution. The participating entities of cache coherence are processors with equal priority. In contrast, the participating entities of transaction execution in a HTM are transactions with different priorities. This difference results in a mismatch between the coherence scheme and conflict detection. In the coherence scheme, a GETX (request for exclusive access) is always forwarded exhaustively (multicast) to the entire set of sharer nodes so that all the sharers will invalidate their private data copy. However, in conflict detection, not all sharers need to receive the request if a high priority sharer can detect and resolve the conflict properly. As the HTM piggybacks onto the cache coherence protocol to forward the GETX from a requester transaction to all the sharer transactions, the sharers with higher priority than the requester will nack the request while other sharers with lower priority will acknowledge the request and abort themselves to avoid conflicts. However, if the request is eventually nacked (i.e., the conflicts do not materialize), any aborted transactions on low-priority sharers could have continued their execution. In other words, the aborting is unnecessary as no conflicts involving the aborted sharers actually materialized. This pathological aborting behavior is identified as *false aborting*, which wastes energy and degrades performance because 1) valid transaction computation is discarded needlessly and 2) the multicast of transactional write requests to all the sharers generates superfluous inter-transaction communication. According to our study, 92% of the transaction aborts are caused by the transactional GETX requests and 41% of these requests incur false aborting.

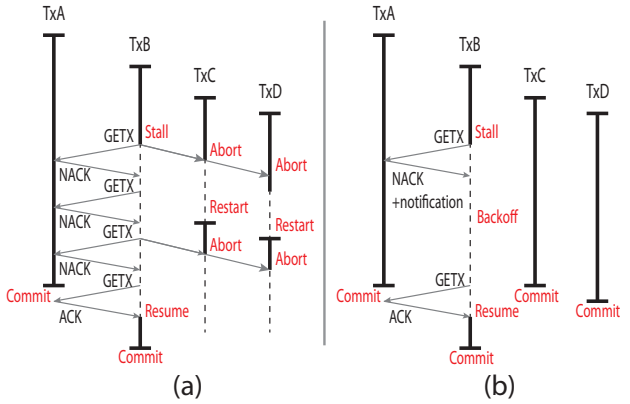

Figure 1: Comparison of transaction executions in the conventional scheme and PUNO.

False aborting is caused by the protocol's exhaustive multicast of transactional GETX requests to the entire set of sharer transactions. Unfortunately, it is impractical to tackle false aborting by re-engineering a HTM-specific cache protocol due to the exorbitant cost. In this paper, we introduce *Predictive Unicast and Notification* (PUNO), a novel hardware mechanism to mitigate false aborting. Our evaluations using full system simulation show that PUNO reduces transaction aborting by 61% on average (up to 89%) in a set of

high contention benchmarks that are representative of future TM workloads. Consequently, the on-chip network traffic is reduced by 32% on average (up to 67%) while the execution time is reduced by 12%. These improvements are achieved with a meager 0.41% area overhead.

2. THE PUNO APPROACH

The basic idea of PUNO is based on the following two important observations regarding false aborting. First, the exhaustive multicast of transactional GETX request to the sharers is needless as long as the conflict caused by the request can be resolved by a sharer with higher priority than the requester. Second, the nacked requester transaction cannot proceed until the nacker sharer transaction finishes executing, as immediate retry of the request will still be rejected by the nacker. PUNO takes advantage of the two observations by 1) replacing the multicast with predictive unicast to a high priority sharer and 2) performing proactive notification to the nacked requester with regard to when to poll the sharers again. When the directory receives a GETX from a transaction, it unicasts the request to the sharer transaction (i.e., the unicast destination) that is predicted with high probability to nack the request. Upon an accurate prediction, the transaction at the unicast destination resolves the conflict by nacking the request. In addition, the nacker transaction proactively notifies the requester with the remaining time the nacker is expected to run. When the requester receives the NACK and the attached notification, it refrains from retrying the request until the time when the nacker transaction is expected to finish.

Figure 1 compares PUNO with the conventional scheme. In the example, a cacheline is read-shared among three transactions (i.e., TxA, TxC and TxD). TxB wishes to write to the cacheline. TxB has a higher priority than TxC and TxD, but has a lower priority than TxA. In the conventional scheme (see Figure 1(a)), The GETX from TxB is forwarded by the directory to all the three sharers. The request is nacked by TxA. However, it causes false aborting as TxC and TxD are aborted unnecessarily. TxB keeps polling the sharers and succeeds with the request when TxA finishes. The polling exacerbates false aborting as TxC and TxD are aborted several times. In contrast, in Figure 1(b), PUNO directs the directory to unicast the GETX request to TxA which is predicted to nack the request. TxA nacks the request and notifies TxB with an estimation of its remaining running time. Consequently, TxB enters backoff and does not retry the request until TxA commits. PUNO reduces inter-transaction communication, and increases transaction throughput by allowing TxC and TxD to commit with TxA.

3. EVALUATION

We conducted cycle-accurate full system simulation using SIMICS and the GEMS tool set to assess the impact of PUNO. Garnet is used as the on-chip network timing model. We present results for all eight workloads from the STAMP benchmark suite that is representative of future TM workloads.

One of the main design objectives for PUNO is to mitigate unnecessary transaction aborting. Figure 2 shows the impact of PUNO on transaction aborting. It is observed that, on average, PUNO reduces transaction aborts by 43%

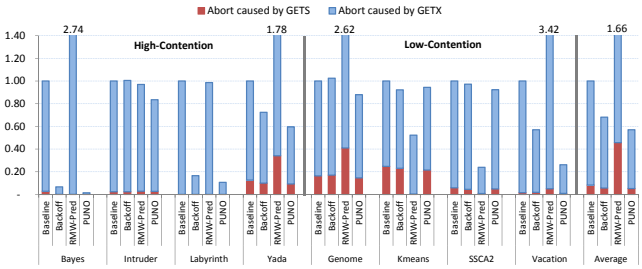

Figure 2: Normalized transaction abort count.

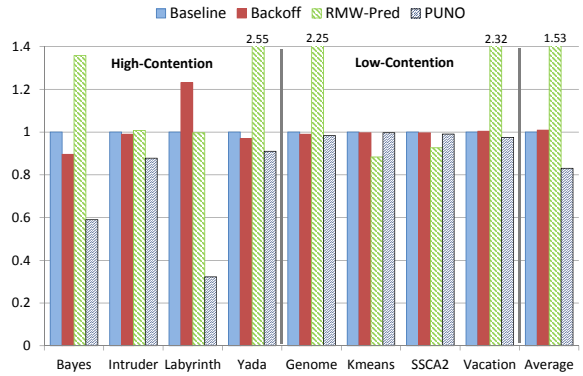

Figure 3: Normalized on-chip network traffic.

(up to 98%) compared with the baseline. In particular, PUNO is effective in reducing aborts caused by transactional GETX requests, which are the main causes of most transaction aborts in the workloads. PUNO achieves significant abort reduction in the high contention benchmarks (61% less aborts). This result is expected as workloads with high contention usually incur more false aborting due to frequent transaction writes and extensive read-read sharing.

Figure 3 shows the normalized on-chip network traffic measured in router traversals by all the network flits. As can be observed, PUNO eliminates 33% (up to 68%) of the traffic in high-contention benchmarks compared with the baseline scheme. Across all the workloads, the network traffic is reduced by an average of 17%. The traffic reduction is due to three facts. First, PUNO replaces the wasteful multicast of GETX requests with unicast when possible. Second, the notification mechanism suppresses unnecessary transaction polling. Third, the reduction in transaction abort translates to less futile traffic from aborted transactions.

4. REFERENCES

[1] http://www.top500.org/.

[2] I. Corp. Intel Architecture Instruction Set Extensions Programming Reference, February 2012.

[3] R. Haring, et al. The IBM Blue Gene/Q Compute Chip. *Micro, IEEE*, 2011.

[4] C. Jacobi, et al. Transactional Memory Architecture and Implementation for IBM System z. In *Procs. of the 45th Int'l Symp. on Microarchitecture*, 2012.

[5] M. Lupon, et al. A Dynamically Adaptable Hardware Transactional Memory. In *Procs. of the 43rd Int'l. Symp. on Microarchitecture*, 2010.

[6] K. E. Moore, et al. LogTM: Log-based Transactional Memory. In *Procs. of the 12th Int'l. Symp. on High Performance Computer Architecture*, 2006.