On the Energy and Performance of Commodity Hardware Transactional Memory

Nuno Diegues
INESC-ID / Instituto Superior Técnico, Univ. of Lisbon
nmld@tecnico.ulisboa.pt

Paolo Romano
INESC-ID / Instituto Superior Técnico, Univ. of Lisbon
romano@inesc-id.pt

Luís Rodrigues
INESC-ID / Instituto Superior Técnico, Univ. of Lisbon
ler@tecnico.ulisboa.pt

ABSTRACT

The advent of multi-core architectures has brought concurrent programming to the forefront of software development. In this context, Transactional Memory (TM) has gained increasing popularity as a simpler, attractive alternative to traditional lock-based synchronization. The recent integration of Hardware TM (HTM) in the last generation of Intel commodity processors turned TM into a mainstream technology, raising a number of questions on its future and that of concurrent programming.

To evaluate the potential impact of Intel's HTM, we conducted the largest study on TM to date, comparing different locking techniques, hardware and software TMs, as well as different combinations of these mechanisms, from the dual perspective of performance and power consumption. As a result we perform a workload characterization, to help programmers better exploit the currently available TM facilities, and identify important research directions.

Categories and Subject Descriptors

D.1.3 [**Software**]: Programming Techniques - Concurrent Programming

Keywords

Transactional Memory; Performance Evaluation; Energy

1. INTRODUCTION

For many years, locking has represented the *de-facto* standard approach to synchronization in concurrent applications. However, its inherent complexity and error-proneness motivated the research for alternatives. Transactional Memory (TM) is an appealing alternative in which programmers are required only to identify which code blocks should run atomically, and not how concurrent accesses to shared state should be synchronized to enforce isolation (as with locks). The TM is then responsible for guaranteeing correctness, by aborting transactions that would generate unsafe histories.

SIGMETRICS'14, June 16–20, 2014, Austin, Texas, USA.
ACM 978-1-4503-2789-3/14/06.
http://dx.doi.org/10.1145/2591971.2592030 .

Over the last decade a large body of TM research focused on software-based implementations (STM) due to their ease of prototyping [1]. Unlike hardware-based implementations, however, STM needs to instrument reads and writes, to trace conflicts at run-time between concurrent transactions. These instrumentation costs can, in certain scenarios, introduce large overheads and hinder performance. HTM support, which has recently been commercialized in commodity processors [3], avoids that problem by relying on the cache coherence protocol of the processor. On the downside, commodity HTMs have architectural restrictions that may limit their practical ability to efficiently deal with some workloads.

The advent of commodity HTM raises a number of questions concerning the future of TM and concurrent programming: how competitive are available HTMs when compared with state of the art STMs? Will the achievable performance be sufficiently alluring to make TM mainstream? What role will STM play now that HTMs are widely available? How limiting are the architectural restrictions of existing HTMs?

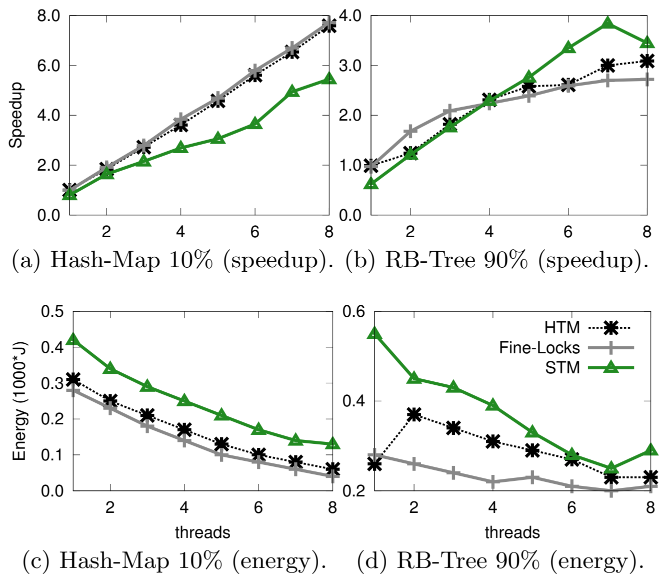
2. COMPARATIVE STUDY

In this work we contribute to answer the previous questions by conducting the largest study to date on TM-based synchronization. We compare, from the twofold perspective of performance and energy-efficiency, a range of synchronization mechanisms: 6 lock approaches with different granularities; 4 STMs; HTM (using Intel TSX); and 2 Hybrid TMs (HyTM) that use STM and HTM mechanisms in synergy. In the following we summarize some of our results; details can be found in an extended version of the paper [2].

Pros and Cons for HTM: We found that in STAMP, a standard benchmark suite for TM, we tested 560 different scenarios and identified three categories of applications in which HTM exhibits different performance (Table 1). Performance of HTM is highly dependent on the memory access patterns on the L1 cache, and long running transactions can lead to frequent cache capacity exceptions and spurious aborts. As such, HTM is the best approach only in Kmeans and SSCA2, which have short transactions interleaved with large non-transactional code blocks. When transaction-intensity is medium, HTM is only the best choice for a limited degree of parallelism, and it is generally better on the energy side than on the performance side. The impact of its hardware limitations is highlighted by several STAMP benchmarks that generate long transactions, in which HTM is outperformed by both locking and STM solutions. By also considering the synchronization of a concurrent Hash-Map and a RedBlack-Tree, HTM clearly unveiled its effectiveness

Table 1: Summary of results according to workload characterization in the STAMP suite, showing the strategies that achieve highest performance, and least energy consumption (note that t = number of threads). Transaction intensity, split in three categories, also shows the percentage of time spent in transactional mode in the application.

	TxIntensity (%)	Speedup	Least Energy
kmeans	low (7)	HTM	HTM
ssca2	low (17)	HTM	HTM
intruder	medium (33)	HTM $\leq 4t$	HTM $\leq 5t$
		STM $\geq 5t$	STM $\geq 6t$
vacation	medium (89)	HTM $\leq 2t$	HTM $\leq 4t$
		STM $\geq 3t$	STM $\geq 5t$
genome	high (97)	STM	STM
yada	high (99)	STM	STM
labyrinth	high (100)	STM	STM



(a) Hash-Map 10% (speedup). (b) RB-Tree 90% (speedup).

(c) Hash-Map 10% (energy). (d) RB-Tree 90% (energy).

Figure 1: Concurrent collections with varying % of updates.

in scenarios with small transactions, in which it is competitive with fine-grained locking (see Fig. 1 for a sample of the workloads). Tests with Memcached (a popular in-memory caching system) confirmed that HTM performed as good as fine locks (4% less performance).

STM is still competitive: Our experiments also show that STM is a robust all-around solution. Among the 4 STMs, there was always one performing best than HTM in 5 out of the 7 STAMP benchmarks. In concurrent data-structures STM is competitive, or even the best, in workloads with many updates (see Fig. 1 for an example). On the other hand, STM has a larger energy consumption. Although STM was initially proposed as a prototyping alternative to actual hardware implementations of TM, its evolution throughout a decade of intense research has resulted in several highly-optimized mechanisms, achieving performance comparable to that of fine-grained locking. This does not mean that STMs embody a perfect solution; instead, these results highlight the current limitations of HTM support, which make STM still a very competitive solution.

3. RESEARCH DIRECTIONS

The results of our study unveil a number of critical issues related with HTM performance and allow for identifying several research problems, whose timely solution could significantly enhance the chances for HTM to turn into a mainstream paradigm for parallel programming:

HyTMs: still a mismatch? The results for HyTM do not live up to the expectation of obtaining the best out of both STMs and HTM. Across our 10 benchmarks and various workloads, HyTMs were never the best solution, and were only competitive in 2 benchmarks. The problem is the mechanisms currently employed for allowing the coexistence of HTM and STM induce overheads in terms of additional spurious aborts. This motivates research in the design of support for non-transactional memory accesses from transactions to allow more efficient synergies of HTM and STM.

Complexity of HTM tuning. HTM performance can be significantly affected by the settings of several parameters and mechanisms. Without proper tuning, Intel TSX yields an average throughput loss of 72% and of 89% in energy consumption. Also, the optimal configuration of these parameters can significantly vary depending on the characteristics of the application's workload. These findings urge for novel approaches capable of removing from programmers the burden of manually tuning HTM parameters, and instead delegating this task to middleware or compiler based solutions.

Relevance of selective instrumentation. Both TSX, as well as current GCC's library for STM, basically trace every memory access performed within a transaction. This causes significant increases of the transaction footprint's size, amplifying the instrumentation overheads in STM, and the chances of incurring in capacity exceptions in HTM. These results motivate research on cross-layer mechanisms, both at compiler and hardware level, to achieve selective instrumentation in a way that is both convenient for the programmer and efficiently implementable in hardware.

4. ACKNOWLEDGMENTS

This work was supported by PEst-OE/EEI/LA0021/2013 from Fundação para a Ciência e Tecnologia, and by the GreenTM project (EXPL/EEI-ESS/0361/2013).

5. REFERENCES

[1] N. Diegues and P. Romano. Time-warp: Lightweight abort minimization in transactional memory. In *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '14, pages 167–178, 2014.

[2] N. Diegues, P. Romano, and L. Rodrigues. The moment of truth: virtues and limitations of commodity Hardware Transactional Memory. Technical Report RT/29/2013, INESC-ID Lisboa, December 2013.

[3] R. M. Yoo, C. J. Hughes, K. Lai, and R. Rajwar. Performance evaluation of intel transactional synchronization extensions for high-performance computing. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 1–19, 2013.