Space-Constrained Structures for HTM

Nick Armstrong
University of Sydney
narm6003@uni.sydney.edu.au

Vincent Gramoli
University of Sydney
vincent.gramoli@sydney.edu.au

Pascal Felber
University of Neuchâtel
pascal.felber@unine.ch

Abstract

Up to now, most of the research efforts to improve performance of programs based on hardware transactions were devoted to designing new hybrid transactional memories and transactional lock elision algorithms to speedup software fallback paths. Unfortunately hardware transactions remain insufficiently exploited, limited especially on microarchitectures where the access set is particularly limited, like the IBM POWER8.

By contrast, we propose a novel class of concurrent data structures, called *space-constrained data structures*, especially designed to boost programs based on hardware transactions. To illustrate our idea we propose a concurrent sorted tree with insertions and deletions of time complexity $O(\log \log m)$ where m is the size of the key range. Preliminary experiments on the Synchrobench benchmark suite show that our space-constrained tree leads to a 5-fold speedup over a traditional red-black tree on a 10-core IBM POWER8.

1 Introduction

Transactional memory is now supported in hardware in modern processors, ranging from Intel Haswell and Skylake to IBM POWER8 microarchitectures. Hardware support means unprecedented performance compared to the traditional software transactional memory libraries. Since hardware transactions are inherently limited by physical capacities, a software fallback mechanism is necessary. The solution is to use either *hybrid transactional memory* [5] with best-effort hardware transaction and software transaction or *transactional lock elision* [16] that consists of trying to speculatively execute a critical section in a fast path hardware transaction before reverting potentially to a fallback path that handles the synchronization in software.

Unfortunately, the cost of the software fallback is prohibitive, and sometimes annihilates the benefit of the hardware transaction. A naive implementation of the software fallback path consists of acquiring a global lock that serializes the critical section similar to an irrevocable transaction. The hardware transaction simply has to start by reading (or *subscribing to*) the lock to make sure conflicts with the software path will be detected during the course of its execution. Unfortunately, this naive approach suffers from contention issues, like the lemming effect in which a lock acquirement, caused by an abort of a hardware transaction, forces all other transactions to abort and to try to acquire the lock in turn [7].

Numerous improvements were proposed in the past years to cope with the cost of the software path [1, 4, 6, 8, 10]. Lazy subscription [4] lets the hardware transaction postpone its subscription to the lock, however, this may lead to inconsistencies [6]. Read-write lock-elision exploits the suspend/resume features of the POWER8 to avoid using hardware transactions and to guarantee progress of read-side critical sections [10]. Amalgamated lock elision exploits fine-grained locks in the fallback path to detect conflicts with the fast path [1]. Allowing only one software thread to hold the lock [8] simplifies the software path to offer a middle-ground between hybrid transactional memory and transactional lock elision. Most of these solutions have shown promising performance improvements on binary search trees, either used as micro-benchmarks [17] or as database tables of more elaborate applications [14].

In contrast with this body of work, we propose to improve performance by devising *space-constrained data structures* accessed by hardware transactions. The challenge lies in constraining the data structure size to maximize the success of hardware transactions while avoiding the need to revert to the slow software fallback path. In particular, we suggest the exploration of data structures whose universe is constrained appropriately to fit within the hardware boundaries [18]. Besides the perfect fit for hardware transactions, these data structures offer asymptotically better time complexity in $O(\log \log m)$ than the classically used $O(\log n)$ data structures with $n \leq m$ the number of elements taken from the range of size m. Note that the previous attempts to limit the size of software transactions in data structures [3, 9] did not reduce the step complexity of accesses but rather split accesses in multiple transactions.

To illustrate our proposal, we present a space-constrained tree resulting from a concurrent variant of a van Emde Boas tree [18] that leverages the POWER8 hardware transactions. To this end, we started adding hardware transactional memory to Synchrobench [11], a benchmark-suite to evaluate synchronization techniques. Until now, Synchrobench has compared algorithms synchronized with lock-based techniques, lock-free techniques, read-copy-update and copy-on-write, or software-only transactional memories. Our preliminary performance evaluation shows that our space-constrained tree can achieve up to 5× higher performance than a classic red-black tree.

2 Constraints of Hardware Transactions

In this section, we illustrate the importance of minimizing the size of hardware transactions to maximize their chance of committing.

A hardware transaction accesses a series of memory locations, whose values get recorded in what is called the read set or the write set of the transaction. During the execution of the hardware transaction, these read and write sets are progressively stored in a hardware dedicated structure that has a limited storage capacity. At runtime, if a new access cannot be recorded in the data structure, then the corresponding hardware transaction aborts.

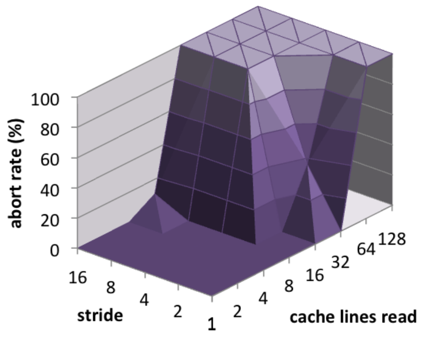


Figure 1: Abort rate of read-only hardware transactions on IBM POWER8

The IBM POWER8 is particularly sensitive to the length of the hardware transaction as it offers a data structure relatively small in comparison to the size of the caches used by the Intel Haswell to store the read and write sets. To measure empirically the probability for a hardware transaction to abort depending on its size, we reused the capacity benchmark of Hasenplaugh, Nguyen and Shavit [12] and plotted the frequencies of abort while we increased the transaction size in cache lines and cache line strides. A stride of k indicates that only the first of k consecutive cache lines are accessed. Depending on the associativity of the structure, a stride of $k > 2$ increases the chances of abort.

Figure 1 depicts the capacity of the POWER8 simply under read-only workloads and confirms that the read set of hardware transactions cannot span 64 cache lines, regardless of the placement of cache lines as 100% of the hardware transactions would abort. In addition, we observe that hardware transactions start aborting when they use 8 cache lines depending on the accessed cache lines, illustrating the effects of associativity.

3 A Space-Constrained Tree

The space-constrained data structure we explore is a van Emde Boas tree [18] (vEB tree), a recursive tree structure capable of holding elements in the range $[0, m-1]$ where $m = 2^k$ for integral k. A vEB tree of size m consists of $\sqrt{m}$ child vEB trees, each of size $\sqrt{m}$. Additionally, each tree maintains an auxiliary vEB tree of size $\sqrt{m}$ to keep track of nonempty children. The minimum and maximum values stored in a tree are only stored in the root of the tree and not in any of its children. These properties are the key to the $O(\log\log m)$ time complexity of the search, insert and delete operations.

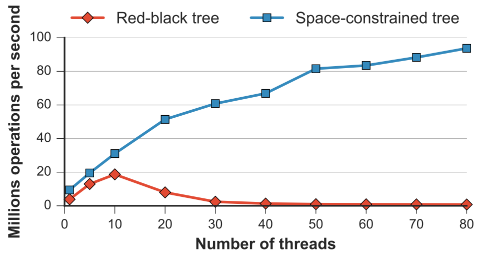
Naively, this structure will take $O(m)$ space as memory must be allocated for every possible value in our universe of m elements. We improve on this by only allocating memory to child and auxiliary trees when they are needed, so in practice the memory usage is reduced for trees holding few elements.

To evaluate the performance of the vEB tree with HTM, we implement a sequential version of the tree and simply wrap the sequential operations in hardware transactions. This allows us to use the HTM capabilities of the Power8 with minimal extra programming effort.

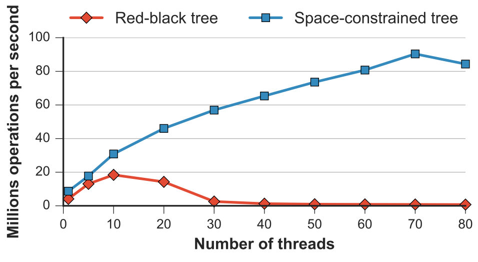
4 Evaluation

We added our space-constrained tree to Synchrobench [11] to compare its performance against the red-black tree on the same ground. We also implemented support for HTM in Synchrobench as this benchmark suite was originally designed to compare synchronization techniques including software transactions but not hardware transactions. The code used for the hardware transaction and the software fallback is relatively simple and similar to [15]. Increasing retries from 5 to 15 before acquiring the fallback lock was found to improve the performance of the HTM implementation [2].

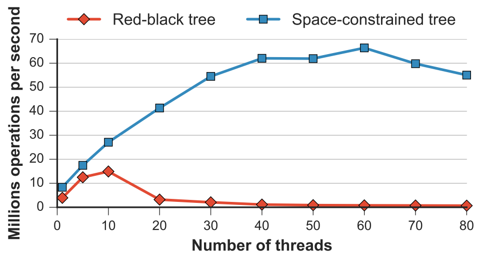
The IBM Power8 consists of 10 cores running at 3.4 GHz up to 8 simultaneous threads each for a total of 80 hardware threads, 32 GB of memory and gcc v4.9.2. A hardware transaction aborts if (i) it issues a load or store to a cache line that is in another transaction's store footprint, (ii) another thread issues a store to a cache line that is in its load footprint, (iii) a non-transaction load is issued on a cache line that is in its load footprint, or (iv) the nesting level exceeds 62 [13].



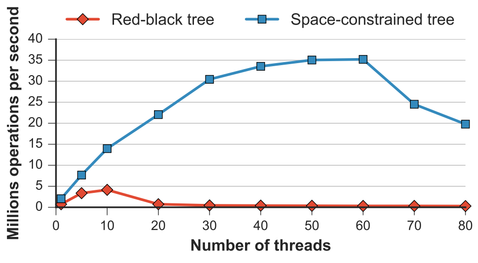
(a) 2^{15} elements and 0% update



(b) 2^{15} elements and 1% update



(c) 2^{15} elements and 5% update



(d) 2^{23} elements and 0% update

Figure 2: Throughput of the space-constrained tree and the red-black tree

Figure 2 depicts the throughput as the number of operations per second averaged over four runs of Synchrobench when executing the red-black tree and our space-constrained concurrent tree with parameters `-i{`2^{15}`, `2^{23}`}-r{`2^{16}`, `2^{24}`}-u{0,1,5}-t{1,5,10,20,30,40,50,60,70,80}`. The universe of

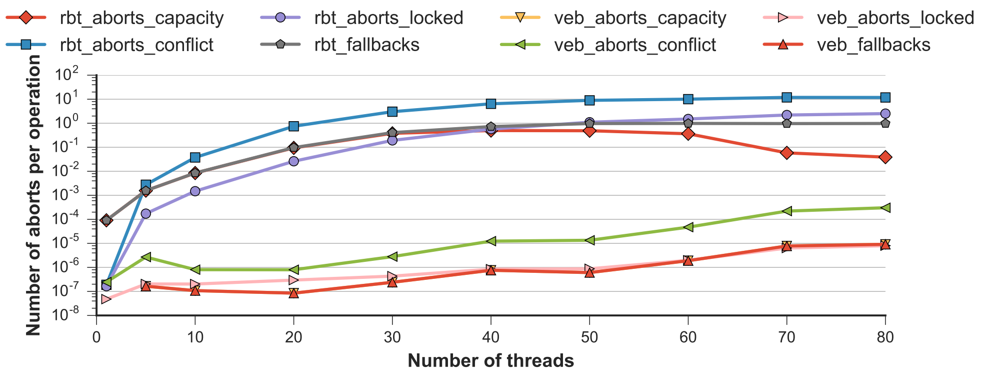


Figure 3: Cause of aborts for trees with 2^{15} elements and 0% update

the space-constrained tree is set to the range of each experiment, so that the actual number of elements is the half of the range in expectation [11]. We observe that the performance of the space-constrained concurrent tree scales to higher number of threads than the red-black tree in all scenarios. As expected, the larger the data structure as in Figure 2(d) or a higher update rate, as in Figure 2(c) translates into lower performance. Finally, on a read-only workload and 2^{15} elements, the peak performance of our tree is 5.01× higher than the peak performance of the red-black tree (cf. Figure 2(a)).

Figure 3 shows a breakdown of the causes of abort in a read-only workload with 2^{15} elements. Also shown is the number of times the process had to fallback to the software lock to complete a transaction. We look at three abort reasons: (i) conflict, where two transactions have conflicting read/write sets; (ii) capacity, where the transaction exceeds the size of the transactional cache structure; (iii) locked, where the fallback lock has already been locked so the HTM fast path cannot be utilised. There are significantly fewer aborts in the vEB tree tests at all thread counts compared to the red-black tree.

We observe a drop in the number of capacity aborts at high thread counts for the red-black tree. This appears related to the increase in conflict aborts which is expected with increasing thread count. We also see for the red-black tree that the number of conflict aborts increases above one per operation and above around 40 threads we see the fallback lock acquired for every operation, severely limiting the performance as in Figure 2(a).

5 Conclusion

We explored the use of HTM with a van Emde Boas tree, a type of *space-constrained data structure*, to implement an integer set and evaluated its concurrent performance compared to an existing red-black tree. Our experiments show that we can obtain a 5.01-fold speedup in peak throughput (operations per second) when run using the HTM implementation of an IBM POWER8 processor. Not only does the peak performance increase, but this peak comes at a much higher thread count when compared to the red-black tree. Similar space-constrained data structures should be able to be tuned for particular applications.

Acknowledgments

This research was supported under Australian Research Council's Discovery Projects funding scheme (project number 160104801) entitled "Data Structures for Multi-Core". Vincent Gramoli is the recipient of the Australian Research Council Discovery International Award.

References

[1] Yehuda Afek, Alexander Matveev, Oscar R. Moll, and Nir Shavit. Amalgamated lock-elision. In *Distributed Computing - 29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*, pages 309–324, 2015.

[2] Trevor Brown, Alex Kogan, Yossi Lev, and Victor Luchangco. Investigating the performance of hardware transactions on a multi-socket machine. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '16, 2016.

[3] Tyler Crain, Vincent Gramoli, and Michel Raynal. A speculation-friendly binary search tree. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP 2012, pages 161–170, 2012.

[4] Luke Dalessandro, François Carouge, Sean White, Yossi Lev, Mark Moir, Michael L. Scott, and Michael F. Spear. Hybrid norec: A case study in the effectiveness of best effort hardware transactional memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 39–52, 2011.

[5] Peter Damron, Alexandra Fedorova, Yossi Lev, Victor Luchangco, Mark Moir, and Daniel Nussbaum. Hybrid transactional memory. In *Proceedings of the Thirteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, 2006.

[6] Dave Dice, Timothy L. Harris, Alex Kogan, Yossi Lev, and Mark Moir. Pitfalls of lazy subcription. In *6th Workshop on the Theory of Transactional Memory (WTTM)*, 2014.

[7] Dave Dice, Maurice Herlihy, Doug Lea, Yossi Lev, Victor Luchangco, Wayne Mesard, Mark Moir, Kevin Moore, and Dan Nussbaum. Applications of the adaptive transactional memory test platform. In *3rd ACM SIGPLAN Workshop on Transactional Computing (Transact)*, 2008.

[8] Dave Dice, Alex Kogan, and Yossi Lev. Refined transactional lock elision. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP 2016, pages 19:1–19:12, 2016.

[9] Aleksandar Dragojević and Tim Harris. Stm in the small: Trading generality for performance in software transactional memory. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 1–14, 2012.

[10] Pascal Felber, Shady Issa, Alexander Matveev, and Paolo Romano. Hardware read-write lock elision. In *Proceedings of the Eleventh European Conference on Computer Systems, EuroSys 2016, London, United Kingdom, April 18-21, 2016*, page 34, 2016.

[11] Vincent Gramoli. More than you ever wanted to know about synchronization: Synchrobench, measuring the impact of the synchronization on concurrent algorithms. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP 2015, pages 1–10, 2015.

[12] William Hasenplaugh, Andrew Nguyen, and Nir Shavit. Quantifying the capacity limitations of hardware transactional memory. In *7th Workshop on the Theory of Transactional Memory (WTTM)*, 2015.

[13] H. Q. Le, G. L. Guthrie, D. E. Williams, M. M. Michael, B. G. Frey, W. J. Starke, C. May, R. Odaira, and T. Nakaike. Transactional memory support in the IBM POWER8 processor. *IBM Journal of Research and Development*, 59(1), 2015.

[14] Chi Cao Minh, JaeWoong Chung, Christos Kozyrakis, and Kunle Olukotun. STAMP: Stanford transactional applications for multi-processing. In *IISWC*, pages 35–46. IEEE, 2008.

[15] Takuya Nakaike, Rei Odaira, Matthew Gaudet, Maged M. Michael, and Hisanobu Tomari. Quantitative comparison of hardware transactional memory for Blue Gene/Q, zEnterprise EC12, Intel Core, and POWER8. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 144–157, New York, NY, USA, 2015. ACM.

[16] Ravi Rajwar and James R. Goodman. Speculative lock elision: Enabling highly concurrent multithreaded execution. In *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 34, pages 294–305, 2001.

[17] Dimitrios Siakavaras, Konstantinos Nikas, Georgios Goumas, and Nectarios Koziris. Performance analysis of concurrent red-black trees on HTM platforms. In *10th ACM SIGPLAN Workshop on Transactional Computing (Transact)*, 2015.

[18] P. van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, FOCS '75, pages 75–84, 1975.