The Potential for Variable-Granularity Access Tracking for Optimistic Parallelism

Mihai Burcea, J. Gregory Steffan, and Cristiana Amza
Department of Electrical and Computer Engineering
University of Toronto
{burceam,steffan,amza}@eecg.toronto.edu

ABSTRACT

Support for optimistic parallelism such as thread-level speculation (TLS) and transactional memory (TM) has been proposed to ease the task of parallelizing software to exploit the new abundance of multicores. A key requirement for such support is the mechanism for tracking memory accesses so that conflicts between speculative threads or transactions can be detected; existing schemes mainly track accesses at a single fixed granularity—i.e., at the word level, cache-line level, or page level. In this paper we demonstrate, for a hardware implementation of TLS and corresponding speculatively-parallelized SpecINT benchmarks, that the coarsest access tracking granularity that does not incur false violations varies significantly across applications, within applications, and across ranges of memory—from word-size to page size. These results motivate a variable-granularity approach to access tracking, and we show that such an approach can reduce the number of memory ranges that must be tracked and compared to detect conflicts can be reduced by an order of magnitude compared to word-level tracking, without increasing false violations. We are currently developing variable-granularity implementations of both a hardware-based TLS system and an STM system.

Categories and Subject Descriptors

D.1.3 [**Concurrent Programming**]: parallel programming

General Terms

Performance

Keywords

Thread-Level Speculation, Transactional Memory, Dependence Tracking, Variable Granularity

1. INTRODUCTION

As most future processors will likely be multicore, the computer systems community is faced with the daunting challenge of parallelizing all software. A great deal of recent research on easing the task of parallelization has focused on *optimistic parallelism*, including support for *thread-level speculation* (TLS) [7, 11, 23] and transactional memory (TM) [6, 17, 18]. Both TLS and TM, whether supported in software, hardware, or a combination, require two key features: (i) the ability to buffer optimistic modifications from the regular memory system; and (ii) the ability to track memory accesses and detect conflicts between speculative threads or transactions.

Focusing on implementations of memory access tracking, there are two conflicting goals: (i) to minimize overheads, including both performance and space overheads; and (ii) to minimize false conflicts. A common approach taken to reduce overhead is to track accesses at coarse granularities. For example, in hardware tracking accesses at a cache-line granularity, or in software tracking accesses at an object or page granularity. Tracking at such coarse granularities saves the prohibitive traffic and latency overheads of tracking at word or sub-word granularities, but has the trade-off of incurring false conflicts. False conflicts can be equally prohibitive to obtaining good performance, as many optimistic parallelization systems are very sensitive to any conflicts. Furthermore, most optimistic parallelization systems track accesses at a single fixed granularity (cache-line, object, page) that may work well on average across different applications, but rarely matches the access patterns of every application or parts of an application. In fact, the portions of applications that have been selected for optimistic parallelization are likely biased to match the granularity of access tracking supported by the underlying system, forgoing optimistic parallelism requiring other granularities that may be available elsewhere in the application.

1.1 Variable-Granularity Access Tracking

We propose that systems for optimistic parallelism should consider supporting *variable-granularity access tracking.* There are two ways that the access tracking granularity can vary. The simplest is to vary the tracking granularity uniformly for the entire system, perhaps per application or part of an application: for example, all accesses are tracked at either word, cache-line, or page granularity. A more aggressive approach would be to vary the tracking granularity across memory: for example, some ranges of memory are tracked at a word granularity while others are tracked at a page granularity. Ideally such a scheme will have less space and latency overhead than a fine-grain approach, since accesses are tracked at a fine granularity only when necessary, and at a very coarse granularity otherwise—reducing the storage to track accesses, as well as the number of items that must be compared to detect conflicts. Such a scheme can also reduce or eliminate false conflicts, since the tracking granularity can vary to match that of the underlying access patterns of the application. Finally, if an implementation of variable-granularity access tracking is dynamic, it can potentially discover the proper granularity required to minimize false conflicts for a given portion of an application. Of course, for such a scheme to be beneficial the overheads of managing variable granularity would have to be minimized.

In this paper we present preliminary results demonstrating the potential for variable-granularity access tracking in an optimistic parallelization system—specifically for a hardware-based TLS system running speculatively-parallelized SpecINT applications. We show that the largest access tracking granularity that does not incur false violations varies significantly across applications, within applications, and across ranges of memory—from word-size to page size. We also show that if variable-granularity access tracking is supported, the number of memory ranges that must be tracked and compared to detect conflicts can be reduced by an order of magnitude compared to word-level tracking, without increasing false violations.

2. RELATED WORK

Proposed hardware and software support for TLS and TM perform access tracking at a variety of different fixed granularities.

Hardware TLS and TM Many hardware schemes for TLS [7, 23, 24] and TM [1, 8, 18, 21] track accesses at the granularity of the cache-line size of the underlying cache hierarchy (typically 32, 64, or 128 bytes). Others demonstrate the necessity of tracking at finer granularities to reduce false violations [11, 23].

Software TLS The software approach to TLS proposed by Cintra *et al.* [3] tracks accesses at a word granularity, but they show improved performance only for applications with few or no conflicts, having eliminated false conflicts. The software approach to TLS in a Java runtime proposed by Pickett *et al.* [20] tracks accesses at an object granularity. While objects do vary in size, there are cases when object-granularity tracking is unnecessary, and others where tracking at a finer-grain than object-level would be beneficial. Early work by Papadimitriou and Mowry [19] explored a software TLS system that builds on a DSM-like environment, and they experimented with support for a range of access-tracking granularities. They found that the granularity for detecting dependences is a critical performance factor, and demonstrate the necessity of tracking memory at a fine granularity—however they agree that this would result in unacceptable overhead.

Software TM Some software approaches to TM (STMs) maintain read and write sets for each transaction at a word granularity [15, 22]. In contrast, Manassiev *et al.* [13] use a page-level granularity to detect conflicts in their STM system; while the actual page size can be configured for this system, it remains fixed for the duration of an application's execution. Some implementations such as RSTM [14] and DSTM [10] use object-based conflict detection, which as discussed above is indeed a form of variable-granularity access tracking but not necessarily the granularity that matches the access patterns of the application. Other STMs are implemented by locking shared data [5, 9, 10]—in such cases the entities that can be locked can also be done so at a variety of fixed granularities. While our main focus in this paper is on variable-granularity access tracking, the general concept is also applicable to the granularity of locking in lock-based STMs.

Hybrid TM Some TM systems are built on a combination of hardware and software. They use a mixture of the techniques mentioned above, detecting conflicts at the cache-line level [16], word level [4], or a combination of both cache-line level (in the hardware) and object level (in the software) [12].

Bulk Disambiguation Ceze *et al.* [2] propose hashing the read and write addresses of speculative accesses into a representative *signature*. They demonstrate that a signature-based approach can reduce the traffic resulting from conflict detection, but at the cost of an increase in false conflicts due to the conservative nature of signatures. In future work we will be obligated to demonstrate that any variable granularity approach out-performs a signature-based approach.

3. EXPERIMENTAL FRAMEWORK

For this study we use the benchmark and simulation infrastructure developed by Steffan *et al.* [23] for TLS. Several SpecINT benchmarks are profiled for execution time and dependences, then selected loops are transformed for speculative execution. The compiler outputs `C` source code which is then compiled with `gcc` v2.95.2 using the "`-O3`" flag to produce optimized, fully-functional MIPS binaries. A full description of the compilation process can be found in a previous publication [23]. We use a simple in-order single-CPI multiprocessor simulator to analyze the memory access patterns of the benchmark applications. To maintain reasonable simulation time, we truncate the execution of all appropriate benchmarks by fast-forwarding the initialization portion of execution and simulating up to the first billion instructions.

It is important to understand that we are measuring regions of code that were selected for speculative execution based on profile information that assumed a 32-byte granularity for tracking reads and word-level granularity (4-bytes) for tracking writes. Also, we limit this study to only measuring "true" (i.e., Read-After-Write) dependences between speculative threads.

4. THE PROBLEM WITH A UNIFORM COARSE GRAIN APPROACH

To illustrate the problems of tracking accesses at a fixed, uniform, coarse granularity, Figure 1 shows the increase in false conflicts detected relative to a word-level tracking granularity (4-bytes) when tracking at typical cache-line granularities of 32, 64, and 128 bytes. For all applications except for IJPEG the amount of false conflicts increases significantly with grain-size, with PARSER suffering the most. Furthermore, recall that the regions of code that were selected for speculative parallelization were done so with a tracking granularity of 32-byte cache line size in mind—hence these results are conservative since these levels of false conflicts could be tolerated while still improving performance. These results

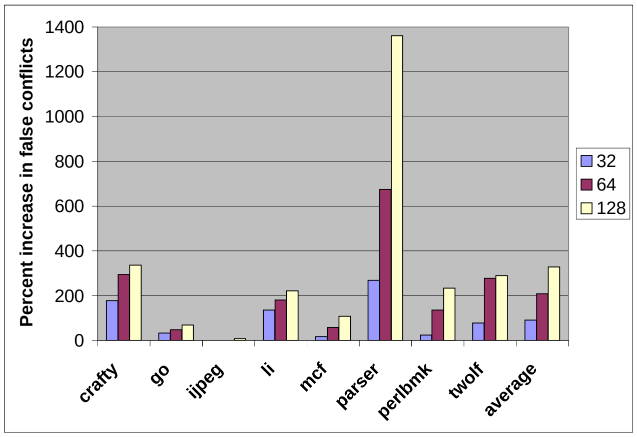

Figure 1: Increase in false conflicts detected relative to word-level tracking granularity (4-bytes) when tracking at typical cache-line granularities of 32, 64, and 128 bytes.

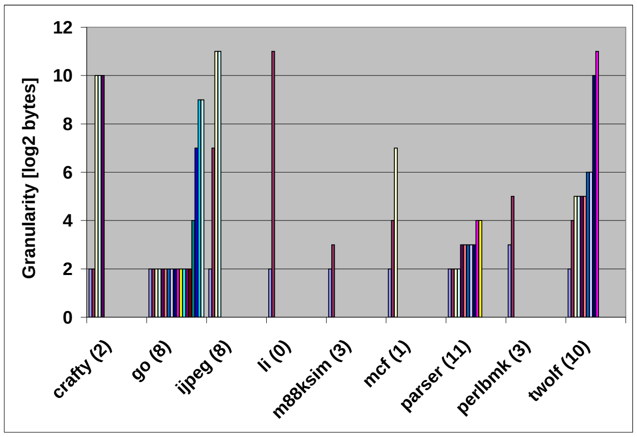

Figure 2: Ideal granularity for all speculatively-parallelized code regions—each bar represents a code region, and the numbers in parentheses count the code regions for which the coarsest granularity (2^{12} bytes, page-level) is ideal.

indicate that cache-line granularity access tracking is non-ideal for most applications, and that word-level tracking is desired in at least part of nearly every application.

5. VARIANCE IN IDEAL GRANULARITY

To measure the potential for varying access tracking granularity, we compute the *ideal granularity*—meaning the coarsest granularity which does not incur any false violations. We begin by measuring the ideal granularity across all speculatively-parallelized code regions in our benchmarks, as shown in Figure 2. Access tracking granularities vary from word-level (4 bytes, or 2 on the y-axis) to page level (4k bytes, or 12 on the y-axis). Each bar represents a code region; to save space, the code regions for which the coarsest granularity (2^{12} bytes, page-level) is ideal are not shown but are instead counted by the numbers in parentheses beside each benchmark. To clarify, a bar with height 4 represents a speculative code region for which the ideal granularity is 2^4 bytes, meaning that there is at least one range in memory for that region for which a tracking granularity of 2^5 bytes would result in a false conflict. Hence for this experiment we are picking only one common ideal granularity per code region.

It is apparent from the figure that no application prefers a single granularity. Most applications have at least one code region that requires the finest granularity access tracking (2^2, 4 bytes) to avoid false conflicts; in particular, GO has a large number of code regions requiring the finest granularity. Tracking accesses at typical cache-line sizes (2^5, 2^6, 2^7 bytes, i.e., 32, 64, or 128 bytes) is sufficient for some code regions, but many require finer-granularity. There are a significant number of code regions that require only page-granularity access tracking, i.e., the region counts shown in parentheses; in particular, PARSER and TWOLF each have a large number of such code regions. This observed high variance in ideal granularity across applications and speculatively parallelized code regions clearly motivates an approach that at least allows the access tracking granularity to vary per code region.

Given that ideal granularity varies widely per code region, we are motivated to further investigate how ideal granularity varies across accessed ranges of memory. Figure 3 shows the distribution of ideal granularities across accessed ranges of memory, across all code regions for each benchmark—hence each segment labeled “grain X” represents an access-tracking granularity of 2^X bytes. To clarify, we have determined the coarsest access tracking granularity for all accessed ranges of memory such that (i) no false conflicts are incurred and (ii) no two tracking elements overlap.

Again we find that ideal granularity varies widely. The fraction of memory ranges that require finest grain access tracking (4 bytes) is surprisingly small: less than 10% for all benchmarks except for CRAFTY which is 23%. The fraction of memory ranges that can tolerate the coarsest grain access tracking (2^{12} bytes, page-level) significant: more than 70% of tracking elements for CRAFTY and IJPEG, and more than 20% for MCF, PERLBMK, and TWOLF. Typical cache-line size granularities (2^5, 2^6, 2^7 bytes, i.e., 32, 64, or 128 bytes) are the ideal granularity for significant fractions of tracking elements for some applications such as LI and MCF, but are an insignificant fraction of others such as CRAFTY and IJPEG. The distribution of ideal granularities varies widely across benchmarks (and also across code regions according to data we do not show here), motivating an approach that can vary the granularity of access tracking across memory.

6. THE POTENTIAL FOR REDUCING OVERHEAD

Access tracking at a variable granularity for speculative optimization systems can potentially reduce overheads without increasing false conflicts. In particular, using a variable-granularity approach can reduce the number of tracking elements used, which in turn can reduce tracking storage, and the traffic and latency between speculative threads or transactions resulting from conflict detection.

To provide a preliminary measure of this potential benefit, in Figure 4 we provide an estimate of how a variable-granularity access tracking approach can dramatically reduce the number of tracking elements required. In particular we plot the reduction in number of tracking elements

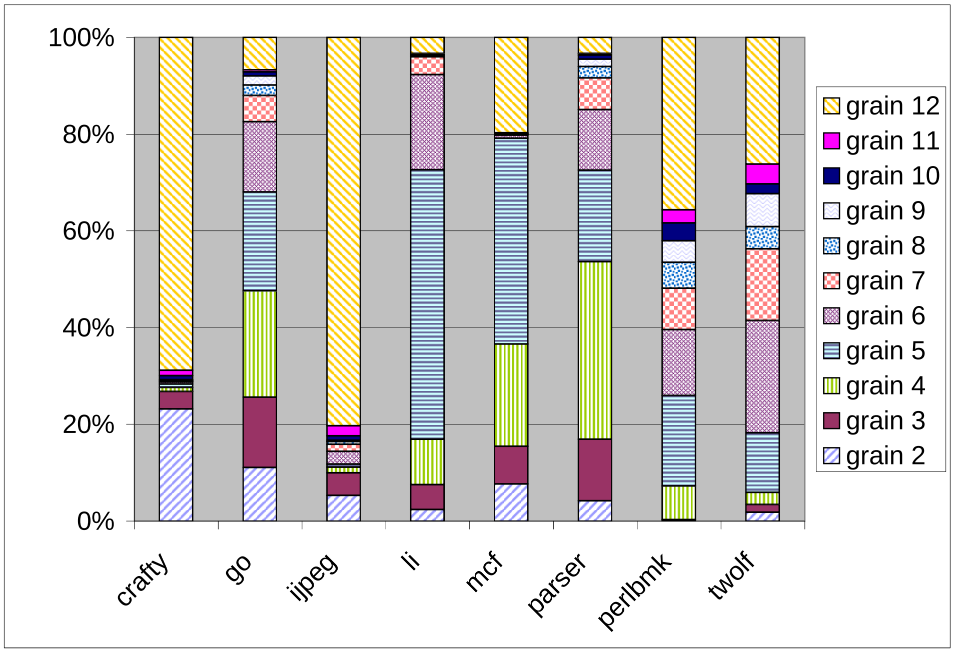

Figure 3: Distribution of ideal granularities across all memory ranges across all speculative code regions for each benchmark. "grain X" means an access-tracking granularity of 2^X bytes.

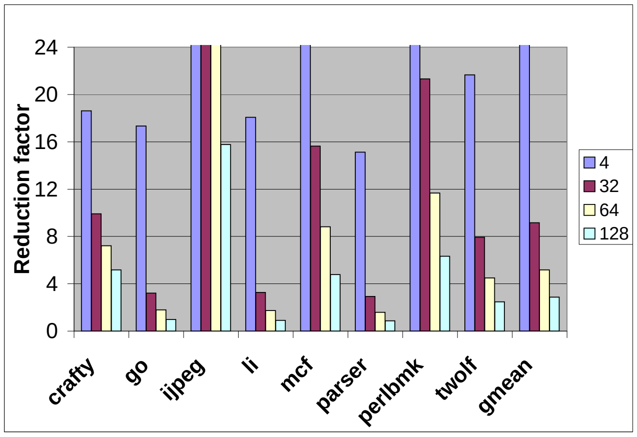

Figure 4: Reduction in number of tracking elements when tracking granularity can vary across accessed ranges of memory (at the ideal granularity), relative to the number of tracking elements for uniform, fixed granularities of 4, 32, 64, and 128 bytes.

when tracking granularity can vary across accessed ranges of memory (at the ideal granularity), relative to the number of tracking elements for uniform, fixed granularities of 4, 32, 64, and 128 bytes (the finest granularity as well as typical cache-line sizes).

Relative to the finest-granularity case (4 bytes), the reduction factor is at least an order of magnitude for every benchmark; For IJPEG, MCF, and PERLBMK the reduction factor is even more dramatic at 458, 51, and 51 times fewer tracking elements respectively (these bars are cropped in the figure). Compared to a 32-byte tracking granularity the reduction factor is still more than 3x for every benchmark. For coarser granularities the reduction factor is still significant, but tracking at these granularities will result in undesirable increases in false conflicts as shown previously in Figure 1. These results are very encouraging, and motivate us to further research implementations of variable-granularity access tracking for speculative optimization systems.

7. IMPLEMENTATION ISSUES

The results in the previous section motivate a variable granularity approach to dependence tracking for speculative parallelism systems. Such an approach could apply to TM or TLS systems, and either hardware or software implementations. We are currently developing variable-granularity implementations of both a hardware-based TLS system and an STM system. For the hardware-based TLS system, a variable-granularity approach promises to reduce conflict-detection traffic, the latency of conflict detection (a key serialization in TLS), and possibly power consumption as well. For the STM system, which is lock-based, a variable-granularity approach promises to greatly reduce the number of locks required saving both space and lock lookup time, while at the same time reducing unnecessary lock-contention for locks that are overly coarse-grain. In both cases, the variable-granularity approach allows the speculative parallelization system to match the access patterns of the application.

In our future work we plan to address the following research questions. Should granularity be determined by memory location or by the code location of the access (i.e., load or store PC)? Can granularity be decided on-the-fly through a dynamic, adaptive system based on iterative sampling? Can a compiler can help by identifying fine or coarse grain accesses? What is the best way to incorporate profile information while limiting its intensity? We will address these questions in our implementations of variable-granularity TLS and STM systems.

8. CONCLUSIONS

In this paper we demonstrated that optimistic parallelization systems that support only a coarse and fixed granularity can suffer a significant number of false conflicts. In contrast, we also showed that systems that support a more variable access tracking granularity can potentially reduce or eliminate false conflicts while at the same time reducing tracking overheads. Different optimistically-parallelized code-regions within an application are amenable to different access tracking granularities ranging from 4 bytes to page-size. The ideal granularity similarly varies widely across ranges of accessed memory. We have shown that using variable granularity for access tracking for optimistic parallelism has significant potential for reducing overhead: in particular, that for every benchmark a variable-granularity approach can reduce the number of tracking elements by an order of magnitude over tracking at a fixed granularity. We believe that these results strongly motivate variable-granularity approaches to access tracking for optimistic parallelization systems, and we are currently developing variable-granularity implementations of both a hardware-based TLS system and an STM system.

9. REFERENCES

[1] C. S. Ananian, K. Asanovic, B. C. Kuszmaul, C. E. Leiserson, and S. Lie. Unbounded Transactional Memory. In *11th International Symposium on High-Performance Computer Architecture (HPCA'05)*, 2005.

[2] L. Ceze, J. Tuck, J. Torrellas, and C. Cascaval. Bulk Disambiguation of Speculative Threads in Multiprocessors. In *ISCA '06: Proceedings of the 33rd Annual International Symposium on Computer Architecture*, 2006.

[3] M. Cintra and D. R. Llanos. Toward Efficient and Robust Software Speculative Parallelization on Multiprocessors. In *PPoPP '03: Proceedings of the Ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2003.

[4] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum. Hybrid Transactional Memory. In *ASPLOS-XII: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 336–346, New York, NY, USA, 2006. ACM.

[5] K. Fraser. Practical Lock Freedom, 2003. Ph.D. Thesis. Also available as Technical Report UCAM-CL-TR-579.

[6] H. Chafi and J. Casper and B. D. Carlstrom and A. McDonald and C. Cao Minh and W. Baek and C. Kozyrakis and K. Olukotun. A Scalable, Non-blocking Approach to Transactional Memory. In *13th International Symposium on High-Performance Computer Architecture (HPCA) 2007*, February 2007.

[7] L. Hammond, M. Willey, and K. Olukotun. Data speculation support for a chip multiprocessor. In *ASPLOS-VIII: Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 58–69, New York, NY, USA, 1998. ACM.

[8] L. Hammond, V. Wong, M. Chen, B. D. Carlstrom, J. D. Davis, B. Hertzberg, M. K. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun. Transactional Memory Coherence and Consistency. In *ISCA '04: Proceedings of the 31st Annual International Symposium on Computer Architecture*, 2004.

[9] T. Harris and K. Fraser. Language support for Lightweight Transactions. In *OOPSLA '03: Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2003.

[10] M. Herlihy, V. Luchangco, M. Moir, and W. N. Scherer, III. Software Transactional Memory for Dynamic-sized Data Structures. In *PODC '03: Proceedings of the 22nd Annual Symposium on Principles of Distributed Computing*, 2003.

[11] V. Krishnan and J. Torrellas. A Chip-Multiprocessor Architecture with Speculative Multithreading. *IEEE Trans. Comput.*, 48(9):866–880, 1999.

[12] S. Kumar, M. Chu, C. J. Hughes, P. Kundu, and A. Nguyen. Hybrid Transactional Memory. In *PPoPP '06: Proceedings of the eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 209–220, New York, NY, USA, 2006. ACM.

[13] K. Manassiev, M. Mihailescu, and C. Amza. Exploiting Distributed Version Concurrency in a Transactional Memory Cluster. In *PPoPP '06: Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2006.

[14] V. J. Marathe, M. F. Spear, C. Heriot, A. Acharya, D. Eisenstat, W. N. Scherer III, and M. L. Scott. Lowering the overhead of software transactional memory. In *ACM SIGPLAN Workshop on Transactional Computing.* Jun 2006.

[15] J. C. Marek Olszewski and J. G. Steffan. JudoSTM: a Dynamic Binary Rewriting Approach to Software Transactional Memory. In *International Conference on Parallel Architectures and Compilation Techniques*, 2007.

[16] C. C. Minh, M. Trautmann, J. Chung, A. McDonald, N. Bronson, J. Casper, C. Kozyrakis, and K. Olukotun. An Effective Hybrid Transactional Memory System with Strong Isolation Guarantees. In *ISCA '07: Proceedings of the 34th annual International Symposium on Computer Architecture*, 2007.

[17] M. Moir. Hybrid Transactional Memory, Jul 2005.

[18] K. E. Moore, J. Bobba, M. J. Moravan, M. D. Hill, and D. A. Wood. LogTM: Log-based Transactional Memory. In *12th International Symposium on High-Performance Computer Architecture (HPCA'06)*, Feb 2006.

[19] S. Papadimitriou and T. Mowry. Exploring Thread-Level Speculation in Software: The Effects of Memory Access Tracking Granularity, Jul 2001. CMU Technical Report CMU-CS-01-145.

[20] C. J. F. Pickett and C. Verbrugge. Software thread level speculation for the Java language and virtual machine environment. In *LCPC'05: Proceedings of the 18th International Workshop on Languages and Compilers for Parallel Computing*, volume 4339 of *LNCS: Lecture Notes in Computer Science*, pages 304–318, Oct. 2005.

[21] R. Rajwar, M. Herlihy, and K. Lai. Virtualizing Transactional Memory. In *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture*, 2005.

[22] B. Saha, A.-R. Adl-Tabatabai, R. L. Hudson, C. C. Minh, and B. Hertzberg. McRT-STM: A High Performance Software Transactional Memory System for a Multi-Core Runtime. In *PPoPP '06: Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2006.

[23] J. G. Steffan, C. Colohan, A. Zhai, and T. C. Mowry. The STAMPede Approach to Thread-Level Speculation. *ACM Trans. Comput. Syst.*, 23(3):253–300, 2005.

[24] T. N. Vijaykumar, S. Gopal, J. E. Smith, and G. Sohi. Speculative Versioning Cache. *IEEE Transactions on Parallel and Distributed Systems*, 12(12):1305–1317, 2001.