Chapter 1

Introduction

Embedded systems sense physical processes and control their behavior, typically through feedback loops. Since physical processes are concurrent, computations that control them must also be concurrent, enabling them to process multiple streams of sensor input and control multiple actuators, all concurrently. Often, such computations need to concurrently read/write shared data objects. Typically, they must also process sensor input and react in a timely manner.

The de facto standard for programming concurrency is the threads abstraction, and the de facto synchronization abstraction is locks. Lock-based concurrency control has significant programmability, scalability, and compositionality challenges [32]. Transactional memory (TM) is an alternative synchronization model for shared in-memory data objects that promises to alleviate these difficulties. With TM, programmers write concurrent code using threads, but organize code that read/write shared objects as transactions, which appear to execute atomically. Two transactions conflict if they access the same object and one access is a write. When that happens, a contention manager (or CM) [27] resolves the conflict by aborting one and allowing the other to proceed to commit, yielding (the illusion of) atomicity. Aborted transactions are re-started, often immediately. In addition to a simple programming model, TM provides performance comparable or superior to highly concurrent fine-grained locking and lock-free approaches [51], and is composable [31]. Multiprocessor TM has been proposed in hardware, called HTM (e.g., [43]), and in software, called STM (e.g., [58]), with the usual tradeoffs: HTM provides strong atomicity [43], has lesser overhead, but needs transactional support in hardware; STM is available on any hardware.

Given STM's programmability, scalability, and compositionality advantages, we consider it for concurrency control in multicore embedded real-time software. Doing so will require bounding transactional retries, as real-time threads, which subsume transactions, must satisfy time constraints. Retry bounds in STM are dependent on the CM policy at hand (analogous to the way thread response time bounds are scheduler-dependent). Thus, real-time CM is logical. Despite the large body of work on contention managers, few results are

known for them on real-time systems. In this proposal, we consider the design of real-time contention managers for real-time multicore systems.

The rest of this Chapter is organized as follows, in Section 1.1, we introduce transactional memory. In Section 1.2, we summarize work on integrating software transactional memory into real-time systems. In Section 1.3, we describe our current research contribution. In Section 1.4, we present an overview of the work we plan to conduct after the preliminary exam in order to achieve the goals of this dissertation and Section 1.5 provides a road map for the rest of the proposal.

1.1 Transactional Memory

Transactional memory (TM) is motivated by Database transactions [26]. In TM, each thread executes a set of transactions when accessing shared memory. A TM transaction consists of a sequence of steps (i.e., reads and/or writes) executed atomically by a thread [30]. Atomicity means that the sequence of steps logically occur at a single instant in time; intermediate states are invisible to other transactions. The difficulty of locks' maintenance and development are the driving motivation for seeking alternate concurrency control methods. Lock-free and wait-free are two alternatives. Lock-free and wait-free have high performance, but significantly complex to write and reason about, and therefore, have largely been limited to a simple data structures - e.g., linked lists, queues, stacks [13–16].

The term "transactional memory" was proposed by Herlihy and Moss [33], where they presented hardware support for lock-free data structures. TM has been provided in hardware (HTM) [28, 33, 42, 47, 50], software (STM) [14, 19, 20, 29, 31, 41, 52, 59, 63] and hybrid TM [17, 36, 45, 60]. Hybrid TM allows STM to improve performance using HTM support. Conflicts between TM threads arise when multiple threads try to access the same object simultaneously, and at least one access is a *write* operation. TM uses *Contention Managers* (CM) to resolve these conflicts. CM decides which transaction to abort and when to restart the aborted transaction in case of conflicts [7, 39, 54, 61].

1.2 Real-Time STM

STM concurrency control for real-time systems has been previously studied in [2, 22–24, 40, 53, 55]. [40] proposes a restricted version of STM for uniprocessors. [24] considers STM for distributed uni-processor systems. A higher priority task causes only one retry in a lower priority tasks due to the uniprocessor. [23] bounds response times in distributed multicore systems with STM synchronization. They consider Pfair scheduling, limit to small atomic regions with fixed size, and limit transaction execution to span at most two quanta. [53] presents real-time scheduling of transactions and serializes transactions based on transac-

tions' - not jobs'- deadlines. However, the work does not bound retries and response times, nor establishes tradeoffs against lock-free synchronization. [55] proposes real-time HTM. The retry bound developed in [55] assumes that the worst case conflict between atomic sections of different tasks occurs when the sections are released at the same time. This assumption does not cover the worst case scenario for transactions' interference. [22] presents earliest-deadline CM or ECM. ECM resolves conflicts by aborting the transaction with longer absoluted deadline. [22] derives a number of properties for ECM, upper bounds transactional retry, and compares schedulability of ECM to retry-loop lock-free synchronization [18]. [22], like [55], assumes that the worst case conflict between atomic sections occurs when the sections are released simultaneously. Besides, [22] assumes all transactions have equal lengths. [2] presents extend idea in [22] to bound number of retries and prevent starvation. [2] presents three ideas for CMs. However, work in [2] is still in progress. Provided algorithms might not give the planned results because they are not analyzed. [6] uses timed automate for schedulability analysis of STM.

[44] uses HTM to build single and double linked queue, and limited capacity queue. HTM is used as an alternative synchronization operation to CAS and locks. [44] provides worst case time analysis for the implemented data structures. It experimentally compares the implemented data structures with CAS and lock. [44] reverses the role of TM. Transactions are used to build the data structure, instead of accessing data structures inside transactions. [56] presents an implementation for HTM in a Java chip multiprocessor system (CMP). The used processor is JOP, where worst case execution time analysis is supported.

[3] presents two steps to minimize and limit number of transactional aborts in real-time multiprocessor embedded systems. [3] assumes tasks are scheduled under partitioned EDF. Each task contains at most one transaction. [3] uses multi-versioned STM. In this method, read-only transactions use recent and consistent snapshot of their read sets. Thus, they do not conflict with other transactions and commit on first try. This reduction in abort number comes at the cost of increased memory storage for different versions. [3] uses real-time characteristics to bound maximum number of required versions for each object. Thus, required space is bounded. [3] serializes conflicting transaction in a chronological order. Ties are broken using least laxity and processor identification. [3] does not provide experimental evaluation of its work.

[5] studies the effect of eager versus lazy conflict detection on real-time schedulability. In eager validation, conflicts are detected as soon as they occur. One of the conflicting transactions should be aborted immediately. In lazy validation, conflict detection is delayed to commit time. [5] assumes each task is a complete transaction. [5] proves that synchronous release of tasks does not necessarily lead to worst case response time of tasks. [5] also proves that lazy validation will always result in a longer or equal response time than eager validation. Experiments show that this gap is quite high if higher priority tasks interfere with lower priority ones.

[38]proposes an adaptive scheme to meet deadlines of transactions. This adaptive scheme

collects statistical information about execution length of transactions. A transaction can execute in any of three modes depending on its closeness to deadline. These modes are optimistic, visible read and irrevocable. The optimistic mode defers conflict detection to commit time. In visible read, other transactions are informed that a particular location has been read and subject to conflict. Irrevocable mode prevents transaction from aborting. As a transaction gets closer to its deadline, it moves from optimistic to visible read to irrevocable mode. Deadline transactions are supported by the underlying scheduler by disabling preemption for them. Experimental evaluation shows improvement in number of committed transactions without noticeable degradation in transactional throughput.

1.3 Summary of Current Research and Contributions

Contribution of the proposal can be summarized as follows:-

- We investigate and design priority-based contention managers for real-time systems. These contention managers try to preserve time constraints in addition to data accuracy. For this goal, we investigate Earliest Deadline First contention manager (ECM) and present Rate Monotonic Assignment contention manager (RCM). ECM and RCM keeps the logic of the underlying real-time scheduler (i.e., transaction belonging to higher priority job is allowed to commit first).
- We present Length-based contention manager (LCM) which can be used with both G-EDF and G-RMA. LCM is not only concerned with priority of the transactions, but also with the length of the interfering transaction relative to the length of the interfered transaction. LCM achieves better retry cost and response time than ECM and RCM.
- Priority-based with Negative value and First access (PNF) contention manager is introduced. PNF avoids transitive retry effect suffered by ECM, RCM and LCM in case of multiple objects per transaction. PNF tries to optimize processor usage by lower priority of aborted transaction. This way, other tasks can proceed if they do no conflict with others.
- For previous contention managers, we upper bounded their retry cost and response times. We compared their schedulability to identify the conditions to prefer one of the them over the others. We also compared their schedulability against schedulability of lock-free method. We also compared retry cost of previous synchronization techniques.

1.4 Summary of Proposed Post Preliminary-Exam Work

Based on our current research results, we proposed the following work:

- **Analytical and experimental comparison between developed CMs and real-time locking protocols** Lock-free offer numerous advantages over locking protocols, but locking protocols are still of wide use in real-time systems due to simpler programming and analysis than lock-free. Thus, it is desired to compares different CMs against real-time locking protocols. Examples of real-time locking protocols include PCP and its variants [12,35,49,57], multicore PCP (MPCP) [37,48], SRP [1,11], multicore SRP (MSRP) [25], PIP [21], FMLP [8,9,34] and OMLP [4]. OMLP and FMLP are similar, and FMLP was found to be superior to other protocols [10].
- **Contention manager development for nested transactions** Transactions can be nested *linearly*, where each transaction has at most one pending transaction [46]. Nesting can also be done in *parallel* where transactions execute concurrently within the same parent [62]. Linear nesting can be *1) flat:* If a child transaction aborts, then parent also aborts. If a child commits, no effect is taken until the parent commits. Modifications made by child transaction is seen only be the parent . *2) Closed:* Similar to *flat nesting* except that if a child transaction aborts, parent does not have to abort. *3) Open:* If a child transaction commits, its modifications is seen not only by the parent, but also by other non-surrounding transactions. If parent aborts after child commits, child modifications are still valid. It is required to extend the proposed real-time CMs (or develop new ones) to handle some or all types of transaction nesting.
- **Combine both LCM and PNF** LCM is designed to reduce the retry cost of one transaction when it is interfered close to its end of execution. PNF is designed to avoid transitive retry in case of multiple objects per transactions. One goal is to combine benefits of both algorithms.
- **Investigate other criterion for contention managers to further reduced retry cost** Criterion other than or combined with priority, transaction length and first access may be used to produce better contention managers.

1.5 Proposal outline

The rest of this dissertation proposal is organized as follows. Chapter 2 overviews past and related work for real-time concurrency control. Chapter 3 provides our computational model and assumptions. Chapter 4 investigates Earliest Deadline First CM (ECN) and proposes Rate-Monotnic Assignment CM (RCM). We derive upper bounds for retry cost and response time under ECM and RCM. Finally, schedulability is compared between ECM, RCM and lock-free method. Chapter 5 shows how to reduce retry cost of transactions under ECM and RCM using a length-based CM (LCM). Chapter 6 tries to solve transitive retry of transaction under ECM, RCM and LCM in case of multiobjects per transaction. Chapter 7 compares measured retry cost and response time for sets of tasks under previous CMs, as well as, lock-free algorithm. We conclude in Chapter 8.

Chapter 2

Past and Related Work

Chapter 3

Model/Assumptions

Chapter 4

ECM and RCM

Chapter 5

LCM

Chapter 6

PNF

Chapter 7

Experiments

Chapter 8

Conclusions, Contributions, and Proposed Post Preliminary-Exam Work

Bibliography

[1] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3:67–99, 1991.

[2] A. Barros and L.M. Pinho. Managing contention of software transactional memory in real-time systems. In *IEEE RTSS, Work-In-Progress*, 2011.

[3] A. Barros and L.M. Pinho. Software transactional memory as a building block for parallel embedded real-time systems. In *37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 251 –255, 30 2011-sept. 2 2011.

[4] Sanjoy Baruah. Techniques for multiprocessor global schedulability analysis. In *RTSS*, pages 119–128, 2007.

[5] C. Belwal and A.M.K. Cheng. Lazy versus eager conflict detection in software transactional memory: A real-time schedulability perspective. *IEEE Embedded Systems Letters*, 3(1):37 –41, march 2011.

[6] C. Belwal and A.M.K. Cheng. Schedulability analysis of transactions in software transactional memory using timed automata. In *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1091 –1098, nov. 2011.

[7] Geoffrey Blake, Ronald G. Dreslinski, and Trevor Mudge. Proactive transaction scheduling for contention management. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 156–167, New York, NY, USA, 2009. ACM.

[8] A. Block, H. Leontyev, B.B. Brandenburg, and J.H. Anderson. A flexible real-time locking protocol for multiprocessors. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 47 –56, aug. 2007.

[9] B.B. Brandenburg and J.H. Anderson. An implementation of the PCP, SRP, D-PCP, M-PCP, and FMLP real-time synchronization protocols in LITMUS-RT. In *RTCSA*, pages 185–194, 2008.

[10] Bjrn Brandenburg and James Anderson. A comparison of the m-pcp, d-pcp, and fmlp on litmus rt. In Theodore Baker, Alain Bui, and Sbastien Tixeuil, editors, *Principles of Distributed Systems*, volume 5401 of *Lecture Notes in Computer Science*, pages 105–124, 2008.

[11] Giorgio C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2004.

[12] Min-Ih Chen and Kwei-Jay Lin. Dynamic priority ceilings: A concurrency control protocol for real-time systems. *Real-Time Systems*, 2:325–346, 1990.

[13] Hyeonjoong Cho, B. Ravindran, and E.D. Jensen. Space-optimal, wait-free real-time synchronization. *IEEE Transactions on Computers*, 56(3):373 –384, March 2007.

[14] Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. On utility accrual processor scheduling with wait-free synchronization for embedded real-time software. In *Proceedings of the ACM symposium on Applied computing*, SAC '06, pages 918–922, New York, NY, USA, 2006. ACM.

[15] Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. Lock-free synchronization for dynamic embedded real-time systems. *ACM Trans. Embed. Comput. Syst.*, 9(3):23:1–23:28, March 2010.

[16] Hyeonjoong Cho, Binoy Ravindran, and E.D. Jensen. A space-optimal wait-free real-time synchronization protocol. In *Proceedings of 17th Euromicro Conference on Real-Time Systems*, ECRTS' 05, pages 79 – 88, July 2005.

[17] Peter Damron, Alexandra Fedorova, Yossi Lev, Victor Luchangco, Mark Moir, and Daniel Nussbaum. Hybrid transactional memory. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XII, pages 336–346, New York, NY, USA, 2006. ACM.

[18] U.M.C. Devi, H. Leontyev, and J.H. Anderson. Efficient synchronization under global edf scheduling on multiprocessors. In *18th Euromicro Conference on Real-Time Systems*, pages 10 pp. –84, 0-0 2006.

[19] Dave Dice and Nir Shavit. Understanding tradeoffs in software transactional memory. In *International Symposium on Code Generation and Optimization*, CGO '07, pages 21 –33, March 2007.

[20] Shlomi Dolev, Danny Hendler, and Adi Suissa. Car-stm: scheduling-based collision avoidance and resolution for software transactional memory. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, PODC '08, pages 125–134, New York, NY, USA, 2008. ACM.

[21] A. Easwaran and B. Andersson. Resource sharing in global fixed-priority preemptive multiprocessor scheduling. In *30th IEEE Real-Time Systems Symposium (RTSS)*, pages 377 –386, dec. 2009.

[22] S. Fahmy and B. Ravindran. On stm concurrency control for multicore embedded real-time software. In *International Conference on Embedded Computer Systems*, SAMOS, pages 1 –8, July 2011.

[23] S.F. Fahmy, B. Ravindran, and E. D. Jensen. On bounding response times under software transactional memory in distributed multiprocessor real-time systems. In *DATE*, pages 688–693, 2009.

[24] S.F. Fahmy, B. Ravindran, and ED Jensen. Response time analysis of software transactional memory-based distributed real-time systems. In *ACM SAC*, pages 334–338, 2009.

[25] P. Gai, M. Di Natale, G. Lipari, A. Ferrari, C. Gabellini, and P. Marceca. A comparison of mpcp and msrp when sharing resources in the janus multiple-processor on a chip platform. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 189 – 198, may 2003.

[26] Jim Gray. The transaction concept: virtues and limitations (invited paper). In *Proceedings of the seventh international conference on Very Large Data Bases - Volume 7*, VLDB '1981, pages 144–154. VLDB Endowment, 1981.

[27] Rachid Guerraoui, Maurice Herlihy, and Bastian Pochon. Toward a theory of transactional contention managers. In *PODC*, pages 258–264, 2005.

[28] Lance Hammond, Vicky Wong, Mike Chen, Brian D. Carlstrom, John D. Davis, Ben Hertzberg, Manohar K. Prabhu, Honggo Wijaya, Christos Kozyrakis, and Kunle Olukotun. Transactional memory coherence and consistency. In *Proceedings of the 31st annual international symposium on Computer architecture*, ISCA '04, pages 102–, Washington, DC, USA, 2004. IEEE Computer Society.

[29] Tim Harris and Keir Fraser. Language support for lightweight transactions. In *Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications*, OOPSLA '03, pages 388–402, New York, NY, USA, 2003. ACM.

[30] Tim Harris, James Larus, and Ravi Rajwar. *Transactional Memory*. Morgan & Claypool Publishers, 2nd. edition, December 2010.

[31] Tim Harris, Simon Marlow, Simon Peyton Jones, and Maurice Herlihy. Composable memory transactions. *Commun. ACM*, 51:91–100, Aug 2008.

[32] Maurice Herlihy. The art of multiprocessor programming. In *PODC*, pages 1–2, 2006.

[33] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: architectural support for lock-free data structures. In *Proceedings of the 20th annual international symposium on computer architecture*, ISCA '93, pages 289–300, New York, NY, USA, 1993. ACM.

[34] P. Holman and J.H. Anderson. Locking under pfair scheduling. *TOCS*, 24(2):140–174, 2006.

[35] D.K. Kiss. Intelligent priority ceiling protocol for scheduling. In *2011 3rd IEEE International Symposium on Logistics and Industrial Informatics*, LINDI, pages 105 –110, aug. 2011.

[36] Sanjeev Kumar, Michael Chu, Christopher J. Hughes, Partha Kundu, and Anthony Nguyen. Hybrid transactional memory. In *Proceedings of the 11th ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '06, pages 209–220, New York, NY, USA, 2006. ACM.

[37] K. Lakshmanan, D. de Niz, and R. Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *30th IEEE Real-Time Systems Symposium (RTSS)*, pages 469 –478, dec. 2009.

[38] W. Maldonado, P. Marlier, P. Felber, J. Lawall, G. Muller, and E. Riviere. Deadline-aware scheduling for software transactional memory. In *41st International Conference on Dependable Systems Networks (DSN)*, pages 257 –268, june 2011.

[39] Walther Maldonado, Patrick Marlier, Pascal Felber, Adi Suissa, Danny Hendler, Alexandra Fedorova, Julia L. Lawall, and Gilles Muller. Scheduling support for transactional memory contention management. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '10, pages 79–90, New York, NY, USA, 2010. ACM.

[40] J. Manson, J. Baker, et al. Preemptible atomic regions for real-time Java. In *RTSS*, pages 10–71, 2006.

[41] Virendra J. Marathe, William N. Scherer, and Michael L. Scott. Design tradeoffs in modern software transactional memory systems. In *Proceedings of the 7th workshop on Workshop on languages, compilers, and run-time support for scalable systems*, LCR '04, pages 1–7, New York, NY, USA, 2004. ACM.

[42] José F. Martínez and Josep Torrellas. Speculative synchronization: applying thread-level speculation to explicitly parallel applications. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, ASPLOS-X, pages 18–29, New York, NY, USA, 2002. ACM.

[43] Austen McDonald. *Architectures for Transactional Memory*. PhD thesis, Stanford University, June 2009.

[44] Fadi Meawad, Martin Schoeberl, Karthik Iyer, and Jan Vitek. Real-time wait-free queues using micro-transactions. In *Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems*, JTRES '11, pages 1–10, New York, NY, USA, 2011. ACM.

[45] Chi Cao Minh, Martin Trautmann, JaeWoong Chung, Austen McDonald, Nathan Bronson, Jared Casper, Christos Kozyrakis, and Kunle Olukotun. An effective hybrid transactional memory system with strong isolation guarantees. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 69–80, New York, NY, USA, 2007. ACM.

[46] J. Eliot B. Moss and Antony L. Hosking. Nested transactional memory: Model and architecture sketches. *Science of Computer Programming*, 63(2):186 – 201, 2006.

[47] Jeffrey Oplinger and Monica S. Lam. Enhancing software reliability with speculative threads. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, ASPLOS-X, pages 184–196, New York, NY, USA, 2002. ACM.

[48] R. Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *ICDCS*, pages 116–123, 2002.

[49] Ragunathan Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[50] Ravi Rajwar and James R. Goodman. Transactional lock-free execution of lock-based programs. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, ASPLOS-X, pages 5–17, New York, NY, USA, 2002. ACM.

[51] Bratin Saha, Ali-Reza Adl-Tabatabai, et al. McRT-STM: a high performance software transactional memory system for a multi-core runtime. In *PPoPP*, pages 187–197, 2006.

[52] Bratin Saha, Ali-Reza Adl-Tabatabai, and Quinn Jacobson. Architectural support for software transactional memory. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 185–196, Washington, DC, USA, 2006. IEEE Computer Society.

[53] T. Sarni, A. Queudet, and P. Valduriez. Real-time support for software transactional memory. In *RTCSA*, pages 477–485, 2009.

[54] William N. Scherer, III and Michael L. Scott. Advanced contention management for dynamic software transactional memory. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, PODC '05, pages 240–248, New York, NY, USA, 2005. ACM.

[55] M. Schoeberl, F. Brandner, and J. Vitek. RTTM: Real-time transactional memory. In *ACM SAC*, pages 326–333, 2010.

[56] M. Schoeberl and P. Hilber. Design and implementation of real-time transactional memory. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 279 –284, 31 2010-sept. 2 2010.

[57] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175 –1185, sep 1990.

[58] Nir Shavit and Dan Touitou. Software transactional memory. In *PODC*, pages 204–213, 1995.

[59] Tatiana Shpeisman, Vijay Menon, Ali-Reza Adl-Tabatabai, Steven Balensiefer, Dan Grossman, Richard L. Hudson, Katherine F. Moore, and Bratin Saha. Enforcing isolation and ordering in stm. In *Proceedings of ACM SIGPLAN conference on Programming language design and implementation*, PLDI '07, pages 78–88, New York, NY, USA, 2007. ACM.

[60] Arrvindh Shriraman, Michael F. Spear, Hemayet Hossain, Virendra J. Marathe, Sandhya Dwarkadas, and Michael L. Scott. An integrated hardware-software approach to flexible transactional memory. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 104–115, New York, NY, USA, 2007. ACM.

[61] Michael F. Spear, Luke Dalessandro, Virendra J. Marathe, and Michael L. Scott. A comprehensive strategy for contention management in software transactional memory. In *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '09, pages 141–150, New York, NY, USA, 2009. ACM.

[62] H. Volos, A. Welc, A.R. Adl-Tabatabai, T. Shpeisman, X. Tian, and R. Narayanaswamy. Nepaltm: design and implementation of nested parallelism for transactional memory systems. *ECOOP 2009–Object-Oriented Programming*, pages 123–147, 2009.

[63] Richard M. Yoo and Hsien-Hsin S. Lee. Adaptive transaction scheduling for transactional memory systems. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, SPAA '08, pages 169–178, New York, NY, USA, 2008. ACM.