LA-UR-97- - 3254

Approved for public release;
distribution is unlimited

Title: Transactional Memories: A New Abstraction
for Parallel Processing

RECEIVED
DEC 01 1997
OSTI

Author(s): J. H. Fasel, TSA-7
O. M. Lubeck, CIC-19
D. Agrawal, J. L. Bruno, and A. El Abbadi, UC-Santa Barbara

Submitted to: DOE Office of Scientific and Technical Information (OSTI)

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

Los Alamos
NATIONAL LABORATORY

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

**Portions of this document may be illegible
in electronic image products. Images are
produced from the best available original
document.**

Transactional Memories: A New Abstraction for Parallel Programming

Joseph H. Fasel* and Olaf M. Lubeck
Los Alamos National Laboratory

Divyakant Agrawal, John L. Bruno, and Amr El Abbadi
University of California- Santa Barbara

Abstract

This is the final report of a three-year, Laboratory Directed Research and Development (LDRD) project at Los Alamos National Laboratory (LANL). Current distributed memory multiprocessor computer systems make the development of parallel programs difficult. From a programmer's perspective, it would be most desirable if the underlying hardware and software could provide the programming abstraction commonly referred to as sequential consistency—a single address space and multiple threads; but enforcement of sequential consistency limits opportunities for architectural and operating system performance optimizations, leading to poor performance. Recently, Herlihy and Moss have introduced a new abstraction called transactional memories for parallel programming. The programming model is shared memory with multiple threads. However, data consistency is obtained through the use of transactions rather than mutual exclusion based on locking. The transaction approach permits the underlying system to exploit the potential parallelism in transaction processing. We explore the feasibility of designing parallel programs using the transaction paradigm for data consistency and a barrier type of thread synchronization.

Background and Research Objectives

Current distributed memory multiprocessor computer systems make the development of parallel programs difficult. From a programmer's perspective, it would be most desirable if the underlying hardware and software could provide the programming abstraction commonly referred to as *sequential consistency* [11]—a single address space and multiple threads. However, if sequential consistency is ensured by the system, it limits the opportunities for architectural and operating system-related performance optimizations and can lead to rather poor performance. At the other extreme (without sophisticated system support), programmers must explicitly take the underlying architecture into account in developing parallel programs. They must coordinate the various components of their parallel programs to ensure correctness and achieve acceptable performance. This

* Principal Investigator, E-mail: jhf@lanl.gov

coordination involves the use of *send* and *receive* primitives to transfer data between the components of the parallel program running at different nodes and the explicit management of data placement and load balancing. Although this approach can lead to impressive performance gains, programming in this style can be a daunting task. The other extreme approach relies on providing weaker programming abstractions (the notions of *weak ordering* [6, 5], *release consistency* [7, 10], *parallel random-access memory (PRAM)* [14], and *slow memory* [8]) that are particularly conducive to efficient implementations. This approach is restrictive since it is applicable to a limited class of parallel programs, and furthermore code development is considerably more complex.

Recently, Herlihy and Moss [9] have introduced a new abstraction called *transactional memories* that is useful for parallel programming. The programming model is shared memory with multiple threads. However, data consistency is obtained through the use of *transactions* rather than mutual exclusion based on locking. The traditional approach based on locking can lead to unnecessary blocking (if a thread holding a lock is blocked for some reason, e.g., a page fault) and does not afford the underlying system the opportunity to exploit the potential parallelism in accessing shared data. On the other hand, the transaction paradigm (used in databases) does not enforce this sequential execution. Rather, it allows conflicting accesses to be interleaved as long as the overall execution is equivalent to a sequential execution. Unlike mutual exclusion primitives, which need to be explicitly stated and controlled by the programmer, the transaction coordination can be delegated to the system. The transaction approach permits the underlying system to exploit the potential parallelism in transaction processing. We explored the feasibility of designing parallel programs using the transaction paradigm for data consistency and a barrier type of thread synchronization.

Importance to LANL's Science and Technology Base and National R&D Needs

This research is aimed at a problem of crucial importance to high-performance computing and simulation: how to keep the overhead of synchronization on references to data from severely limiting performance.

Scientific Approach and Accomplishments

During this project we implemented the transactional memory using the pessimistic approach with abortion [9] and the optimistic approach [12] on the shared memory platform Maya [3,4,2] that was developed at the University of California-Santa Barbara (UCSB). Maya is a simulation platform capable of simulating the execution of various distributed shared-memory protocols using the parallel discrete-event simulation technique for different architectural platforms. Maya is also capable of executing the distributed shared-memory protocols directly on our network of Sun workstations. We have the Maya prototype running and have obtained some performance data on sequentially consistent distributed shared memories [13], as well as weaker memory systems [1,14]. In the course of experimentation with Maya, we have identified several important synchronization primitives such as the locks and barriers. With the introduction of transactions, locks (for the use of critical sections) are not required. Instead, another abstraction for the invocation of a critical section may be convenient that will initiate a transaction for the critical section and restart the transaction automatically in case it is aborted.

Currently, we have completed Herlihy and Moss's [9] implementation of transactional memory on Maya. In this implementation, each transaction can access its own data item and change the status of the data item appropriately (logically "locking" the item). Whenever such an access cannot be completed due to concurrent operations by other transactions, the transaction issuing the offending operation is aborted unilaterally. The following primitives are available to the users:

BEGIN_TX: tells the underlying memory controller that a transaction is started.

LT (Load Transaction): reads the value of a shared object.

LTX (Load Transaction Exclusive): reads the value of the shared object and indicates that the object is likely to be updated.

ST (Store Transactional): write the value of the shared object. The value is not visible to others until the transaction commits.

VALIDATE: inquiry to determine if the transaction has been aborted.

COMMIT: flush the writes of the transaction to the shared memory.

ABORT: discard the writes of a transaction.

The above primitives can be used in a parallel program to ensure that the view of a distributed shared memory is sequentially consistent. We evaluated several benchmarks in the context of our implementations: the shared counters and the doubly linked list [9]. We

compared these benchmarks with transactional memory and sequential consistency. The results indicate that although transactional memory performs better in some cases, the variance in its performance is quite high. This can be explained due to the restart-based implementation of transactions.

In the context of transactional memories, we experimented with alternative implementations of transactional memories. The pessimistic approach [9] is a reasonable implementation on a bus-based architecture or on a network-based architecture with a very small communication overhead. An alternative approach that was employed on architecture with a higher communication latency is the optimistic approach. A high communication overhead is typical for distributed shared memory computation environments. The optimistic approach avoids communication when a data item is accessed, whenever possible. A footprint on the data item is maintained for the transaction. At commit time, the footprint is validated with the footprints of all concurrent transactions. When a conflict is detected, low priority transactions abort in favor of high priority transactions. Hence a communication is only required at commit time. As an improvement to Herlihy and Moss's approach, instead of aborting the offending transaction when a conflict is detected, we established an ordering on the conflict, called the required commit order. As long as the required commit order is acyclic, *no* transaction needs to be aborted. Transactions do not abort when a conflict exists, but do abort when a conflict materializes. Additional information on the order must be maintained, which is used at commit time, as well as in response to remote data requests.

Unfortunately, our preliminary results indicated that, due to the high cost to implement shared memories and the latency of PVM-based communication, transaction memories have no advantage over traditional approaches of parallel programming. As a consequence, we directed our efforts to implement a low latency implementation of MPI (a standardized message passing interface library similar to PVM) on Meiko CS/2 and a cluster of workstations connected by a high-speed ATM network. We completed this implementation in the summer of 1996 and need to extend our transactional memory implementation in this framework.

Publications

1. Agrawal, D., Choy, M., Leong, H., Singh, A. Maya: A platform for evaluating distributed shared memory systems. *Proc. 4th Workshop on Scalable Shared-Memory Multiprocessors* (1994).
2. Leong, H. V., Agrawal, D. Message logging recovery schemes. *Proc. IEEE Int. Conf. on Distributed Computing Systems* (1994).
3. Agrawal, D., Choy, M., Leong, H., Singh, A. Maya: A simulation platform for parallel architectures and distributed shared memories. *Proc. 8th Workshop on Parallel and Distributed Simulation* (1994).
4. Agrawal, D., Choy, M., Leong, H., Singh, A. Investigating weak memories with Maya. *Proc. 3rd IEEE Int. Symp. on High-Performance Distributed Computing* (1994).
5. Agrawal, D., Choy, M., Leong, H., Singh, A. Mixed consistency: A model for parallel programming. *Proc. ACM Symp. on Principles of Distributed Computing* (1994).
6. Agrawal, D., Choy, M., Leong, H., Singh, A. Prototype developed and installed on INTEL Paragon and on a network of workstations at UCSB, Software.
7. Agrawal, D., El Abbadi, A. A token-based fault-tolerant distributed mutual exclusion algorithm. *Journal of Parallel and Distributed Computing* 24, 164–176 (1995).
8. Agrawal, D., Bruno, J. L., El Abbadi, A., Krishnaswamy, V. Managing concurrent activities in collaborative environments. *Proc. 3rd Int. Conf. on Cooperative Information Systems*, 112–124, (1995).
9. Jones, C. R., Singh, A. L., Agrawal, D. Low-latency MPI for Meiko CS/2 and ATM Clusters. *Proc. Int. Parallel Processing Symp.* (1997).
10. Agrawal, D., Alonso, G., El Abbadi, A., Stanoi, I. Exploiting atomic broadcast in replicated databases. Submitted to European Workshop on Parallel Systems (EuroPar 97).
11. Agrawal, D., El Abbadi, A., Steinke, R. Epidemic algorithms for transaction management in replicated databases. *Proc. ACM Symp. on Principles of Databases* (1997).
12. Agrawal, D., El Abbadi, A., Singh, A. K. Efficient view management at data warehouses. *Proc. ACM Int. Conf. on Management of Data* (1977).
13. Stanoi, I., Agrawal, D., El Abbadi, A. Transaction management using causal broadcasts. Submitted to ACM Symposium on Principles of Distributed Computing (PODC 97).

References

- [1] Ahamad, M., Burns, J. E., Hutto, P. W., Neiger, G. Causal memory. *Proc. 5th International Workshop on Distributed Algorithms*, 9–30 (1991).
- [2] Agrawal, D., Choy, M., Leong, H. V., Singh, A. K. Investigating weak memories using Maya. *Proc. 3rd. Int. Symp. on High-Performance Distributed Computing*, 120–130 (1994).
- [3] Agrawal, D., Choy, M., Leong, H. V., Singh, A. K. Maya: A platform for evaluating memory coherence protocols and parallel architectures. *Proc. Fourth Workshop on Scalable Shared-Memory Multiprocessors*, 1994 (to appear).
- [4] Agrawal, D., Choy, M., Leong, H. V., Singh, A. K. Maya: A simulation platform for distributed shared memories. *Proc. 8th Int. Workshop on Parallel and Distributed Simulation*, 151–155 (1994).
- [5] Adve, S. V., Hill, M. D. Weak ordering—A new definition. *Proc. 17th Ann. Int. Symp. on Computer Architecture*, 2–14 (1990).
- [6] Dubois, M., Scheurich, C., Fayé, A. B. Synchronization, coherence, and event ordering in multiprocessors. *IEEE Computer* **21**, 9–21 (1988).
- [7] Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A., Hennessy, J. L. Memory consistency and event ordering in scalable shared-memory multiprocessors. *Proc. 17th Ann. Int. Symp. on Computer Architecture*, 15–26 (1990).
- [8] Hutto, P. W., Ahamad, M. Slow memory: Weakening consistency to enhance concurrency in distributed shared memories. *Proc. 10th Int. Conf. on Distributed Computing Systems*, 302–309 (1990).
- [9] Herlihy, M. P., Moss, J. E. B. Transactional memory: Architectural support for lock-free data structure. *Proc. 20th Ann. Int. Symp. on Computer Architecture*, 289–300 (1993).
- [10] Keleher, P., Colx, A. L., Zwaenepoel. Lazy release consistency for software distributed shared memory. *Proc. 19th Ann. Int. Symp. on Computer Architecture*, 13–21 (1992).
- [11] Lamport, L. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. on Computers* **28**, 690–691 (1979).
- [12] Leong, H. V. *Exploiting Semantics in Parallel and Distributed Computations*. Ph.D. thesis, Dept. of Comp. Sci., UCSB, 1994.
- [13] Li, K., Hudak, P. Memory coherence in shared virtual memory systems. *ACM Trans. on Computer Systems* **7**, 321–359 (1989).
- [14] Lipton, R. J., Sandberg, J. S. PRAM: A scalable shared memory. Technical Report CS-TR-180-88, Dept. of Comp. Sci., Princeton U. (1988).