6.895 Theory of Parallel Systems Sean Lie

An Evaluation of Nested Concurrent Transactions

Final Project Proposal

1 Background

Transactional memory is a hardware mechanism that allows the programmer to define atomic regions (called transactions) containing memory accesses to multiple independent addresses. Atomicity requires that a valid execution of multiple regions must produce the same result as a possible serial execution. Instructions are added to the processor ISA to provide the programmer a method of defining these transactional regions. The hardware is responsible for ensuring that every transaction is executed atomically when viewed from the global memory system.

2 Nested Transactions

It is desirable to allow the programmer to make subroutine calls from within a transaction. These subroutines may themselves contain transactions since they may be called from within normal code as well as transactional code. One simple way of dealing with this case is to merge all nested transactions into one large transaction. Merging nested transactions lends itself to a simple hardware implementation. When nested transactions are merged, all inner transactions are treated as being a part of the outermost transaction. All conflicts abort the outermost transaction and, in effect, abort all nested transactions as well.

3 Concurrent Nested Transactions

Though merging nested transactions will always lead to correct program execution, there are practical cases where it can also lead to unnecessary aborts. For example, if a long transaction uses a small transaction at the beginning of its execution to modify a shared data structure, merging the two transactions will have the effect of including the shared data structure in the large transaction. If the shared data structure is not accessed for the remained of the large transaction, merging the two transactions can drastically decrease the parallelism. Therefore, the ability to perform concurrent nested transactions is necessary. A concurrent nested transaction is a transaction nested within another transaction that runs independent of the outer transaction. The inner transaction can abort or commit independent of the outer transaction. When the inner transaction is running, the outer transaction is effectively suspended.

4 Project Proposal

I propose to investigate nested concurrent transactions in greater detail. There are many design problems that need to be addressed. Firstly, it is important to define useful and easy to understand semantics for nested concurrent transactions. Once semantics are defined, the hardware, compiler, and operating system must implement the semantics. I will explore different possibilities for semantics as well as their implantations.

In addition, nested concurrent transactions can be used to achieve goals other than increasing parallelism. For example, they can be used to modify a data structure for logging or debugging purposes since their effects are seen even if the transaction is aborted. I will investigate some of these possibilities as well.

5 Previous Work

Currently, I have a working transactional memory system in a software simulator. The simulator, UVSIM, models an R10K MIPS microprocessor in a shared-memory multi-processor environment similar to the SGI Origin system. The current implementation performs nested transaction merging but does not support concurrent transactions of any kind.

6 Future Work

The first phase of this project will involve adding the necessary mechanisms into UVSIM to support concurrent transactions. With the infrastructure in place, I can experiment with different semantics and their implications on the underlying hardware design. Once the semantics are defined, the necessary hardware changes will be added to UVSIM support nested concurrent transactions.

To evaluate the usefulness as well as the performance of the design, a program that uses nested concurrent transactions will be needed. Either a program will be written or existing code will be modified to serve this purpose. The same will be done to evaluate the other uses of nested concurrent transactions.

7 Scope

The scope of this project is appropriate for this class since it is very scalable. At the base case, some changes are made to UVSIM and one or two programs are written to run on UVSIM. This can clearly be accomplished in one term. On the other hand, if time permits, another implementation can be explored and/or other applications can be written and used for evaluation.

8 Collaboration

Because of the nature of this project, it is desirable to have another student to collaborate with. There is definitely more than enough work for 2 students. Therefore, I have explored the possibility of working with Scottt Ananian. Scott has been working on software transactional memory and is interested in porting some of his codes to use hardware transactions. Currently, we have not decided on the nature of our collaboration but we are actively working on that.

9 References

Herlihy, M. and Moss, J. E. B. [1993]. Transactional Memory: Architectural Support for Lock-Free Data Structures, Proceedings of the 20th Annual International Symposium on Computer Architecture.

Herlihy, M. and Moss, J. E. B. [1992]. Transactional Memory: Architectural Support for Lock-Free Data Structures, Technical Report 92/07, Digital Cambridge Research Lab, Cambridge, Massachusetts.

Rajwar, R. and Goodman, J. R. [2001]. Speculative Lock Elision: Enabling Highly Concurrent Multithreaded Execution, Proceedings of the 34th ACM/IEEE International Symposium on Microarchitecture, 294-305.

Rajwar, R. and Goodman, J. R. [2002]. Transactional Lock-Free Execution of Lock-Based Programs, Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems.

Zhang, L. [2001]. URSIM Reference Manual, Technical Report UUCS-00-015, University of Utah, Salt Lake City, Utah.