6.895 Theory of Parallel Systems	Project Proposal

Linear time detection of determinacy races

Jeremy Fineman

1 Project Goal

I propose a new algorithm for determinacy race detection which runs in $O(T)$ time, where T is the running time of the Cilk program on one processor. I also extend this algorithm to work with both of the locking algorithms, All-Sets and Brelly, in $O(n^k T)$ and $O(kT)$ time respectively, where k is the maximum number of locks held simultaneously.

I have already developed the new algorithm, and I have done some work on proving its correctness. I have not yet examined space bounds, but I will do that as part of this project. I also intend to implement as new Nondeterminator which takes advantage of this improved algorithm. I will run this new version against the old one with several benchmarks to get experimental results for the improvement in performance.

If time permits, I will look at parallelizing this algorithm.

2 Background

Cilk [2] is a multithreaded programming language in which multiple threads concurrently access shared memory. Generally, Cilk programs are meant to be deterministic. That is to say, the program should generate the same output no matter how the threads are scheduled. Nondeterministic behavior can occur, however, if a thread writes a location while another thread is accessing the same location concurrently.

The SP-bags algorithm for determinacy race detection [6] is a serial algorithm. It executes a Cilk program and given input serially while maintaining data structures to keep track of which procedure instances run logically in parallel. Using disjoint set operations, the SP-bags algorithm can determine whether two procedures operate logically in parallel. Essentially, the SP-bags algorithm looks up the least common ancestor of two procedures to determine whether they are parallel. This algorithm runs in $O(T\alpha(v,v))$ time, where T is the running time of the Cilk program on one processors, α is the functional inverse of Ackermann's function, and v is the number of shared memory locations.

If we want to determine whether two threads are logically in parallel, we do not need to know their least common ancestor. Thus, the SP-bags algorithm maintains a little more structure and does more work than is necessary. Using order maintenance data structures to determine whether two threads are logically parallel, we can detect determinacy races in $O(T)$ time, where T is the running time of the Cilk program on one processor.

3 Order Maintenance

A total order data structure supports three operations:

1. Insert(X,Y): Insert a new element Y immediately following the element X in the total order.
2. Delete(X): Delete an element X.
3. Precedes(X,Y): return true if X precedes Y in the total order.

We can perform any n of these operations in $O(n)$ time [5].

4 Linear time race detection idea

If we maintain two total orders of all the threads in the computation dag, we can determine whether two threads are logically in parallel. We can reuse the rest of the logic (e.g. a *reader* and *writer* shadow space) from the SP-bags algorithm. This new algorithm also runs serially on a Cilk program and given input.

Suppose we have two orders representing our dag. One order represents the serial, depth-first left-to-right execution, where we follow every spawned thread before following the continuation thread. The second order represents a right-to-left execution, where we follow every continuation edge before the corresponding spawn edge. Two threads e_1 and e_2 are logically parallel if and only if $\textsc{Precedes}(e_1, e_2)$ in one order and $\textsc{Precedes}(e_2, e_1)$ in the other. Intuitively, this conclusion makes sense. Whenever a Cilk program performs a **spawn**, the two resulting threads and all their descendants operate logically in parallel. In one order, the spawn thread and its descendants appear before the continuation thread and its descendants. In the other order, this relationship is reversed. Thus, we can determine that two threads are parallel based on both orders.

Since we only need to perform $O(T)$ order maintenance operations, and each of these operations has an $O(1)$ amortized cost, the total running time of this algorithm is $O(T)$.

References

[1] Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list. In *Proceedings of the Tenth European Symposium on Algorithms (ESA)*, pages 152–164, 2002.

[2] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. In *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 207–216, Santa Barbara, California, July 1995.

[3] Guang-Ien Cheng. Algorithms for data-race detection in multithreaded programs. Master's thesis, Massachusetts Institute of Technology, 1998.

[4] Guang-Ien Cheng, Mingdong Feng, Charles E. Leiserson, Keith H. Randall, and Andrew F. Stark. Detecting data races in cilk programs that use locks. In *Proceedinges of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, June 1998.

[5] P. Dietz and D. Sleator. Two algorithms for maintaining order in a list. In *STOC*, 1987.

[6] C. E. Leiserson and M. Feng. Efficient detection of determinacy races in cilk programs. In *Proceedinges of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–11, Newport, Rhode Island, June 1997.