Efficient on the fly maintenance of series-parallel relationships

Jeremy T. Fineman
MIT Laboratory for Computer Science
200 Technology Square
Cambridge, MA 02139

Singapore-MIT Alliance
NSF Grant ACI-032497

November 3, 2003

***Abstract*—A series-parallel directed acyclic graph, or *SP-dag*, contains nodes that are either in series or logically in parallel. We present a data structure and algorithm to efficiently determine, in a single serial walk of the dag, whether two nodes are logically in parallel. We also present a modified version of this algorithm to detect parallel threads in any (parallel or serial) execution of a Cilk dag.**

The techniques we present in this paper depend on an order-maintenance data structure inspired by Dietz and Sleator. This data structure supports inserts and queries in *O(1)* amortized time. We maintain two complementary total-orders of the dag. If two nodes have the same relationship in both orders, then they operate in series. If they have different relationships in both orders, then they operate logically in parallel. The algorithm we use allows us to maintain both orders on a single, serial walk of the dag. Our algorithm takes *O(T)* time, where *T* is the time to execute a serial walk of the dag.

The Dietz and Sleator order-maintenance structure does not support concurrent operations. Given the work-first property of the Cilk scheduler with a bounded number of steals (with high probability), we can maintain separate order structures for each processor in addition to a global order structure. Concurrent operations are only required in the global order structure on a steal. We prove that a Cilk program modified with our algorithm has a running time bounded to within a constant factor of the original program.

Determinacy race detection depends on knowledge of the SP relationships of a parallel-program. We will apply the serial algorithm mentioned above to a determinacy race detector in Cilk. We will run benchmarks to compare the running time of this implementation to that of the current Nondeterminator, which relies on least common ancestor lookups.

[Full Text Not Available]