An Optimistic, Parallel Algorithm for the Circuit Value Update Problem

Abstract

Keith H. Randall
`randall@theory.lcs.mit.edu`
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

Many applications—including event-driven logic simulation [1, 2, 5], spreadsheet calculation, and constraint propagation in artificial intelligence programs [6]—maintain a directed acyclic graph (dag) in which each vertex contains a value and each edge represents a data dependency between two values. The value of a vertex is determined by a vertex-specific function of the values of the vertex's immediate predecessors. During the execution of the application, it is common for the value or function at a node to be changed, in which case the effects of that change must be propagated throughout the dag. Since propagating changes is an expensive operation, it is important that as few elements be reevaluated as possible.

We abstract this class of problems as the **circuit value update problem**. We are given a combinational circuit as a dag $G = (V, E)$, where the vertices are either combinational elements or circuit inputs and the edges represent interconnections. Figure 2 illustrates the circuit value update problem using a combinational logic circuit. Each combinational element $v \in V$ computes a function f_v of its inputs to produce a single value $value(v)$ which is fanned out to v's immediate successors in the circuit. In the figure, the functions are the boolean connectives AND, OR, and NOT, represented by traditional logic gates. We presume that the circuit has already been evaluated, as is shown in part (a) of the figure. The circuit value update problem is to update the values as efficiently as possible when one or more of the inputs change. For example, part (b) of the figure shows the values that must be updated when Input C is changed from a 0 to a 1.

We have designed an optimistic, asynchronous, parallel algorithm to perform a circuit value update in $O(W/P + D + \lg W + \lg P)$ expected time, where W and D are the size and embedded depth of the volatile subcircuit, and P is the number of processors. The **volatile** subcircuit is the subcircuit whose inputs can change or glitch as a result of the update, and the **embedded depth** of a subcircuit is its depth measured in the original circuit.

Our optimistic algorithm is extremely simple: we give the pseudocode for it in Figure 1. In a preprocessing phase (not shown), the entire circuit is evaluated and elements are assigned at random to processors. During the update algorithm, elements that need reevaluation are kept in local priority queues on their assigned processors. The priorities of circuit elements are their depths in the circuit. At each time step, each processor removes the shallowest element from its local priority queue and evaluates it. If its value changes, its successors are enqueued on the processors to which they are assigned.

OPTIMISTIC
forall processors $p \in [1 \,..\, P]$
  do $Q[p] \leftarrow$ updated elements assigned to p
while $\exists i$ such that $Q[i] \neq \emptyset$
  do forall processors $p \in [1 \,..\, P]$
    do if $Q[p] \neq \emptyset$
      then $u \leftarrow \textsc{ExtractMin}(Q[p])$
        $new \leftarrow f_u(u\text{'s inputs})$
        if $new \neq value[u]$
        then $value[u] \leftarrow new$
          for $v \in Adj[u]$
            do $\textsc{Insert}(Q[proc[v]], v)$

Figure 1: Pseudocode for the optimistic algorithm. The set of immediate successors of a element u is denoted $Adj[u]$. The processor to which u is assigned is denoted $proc[u]$.

We call this algorithm **optimistic** because it does not wait for all of an element u's predecessors to be correctly evaluated before u is evaluated. The optimistic algorithm assumes (perhaps incorrectly) that all of u's predecessors are evaluated correctly when u is evaluated. If this assumption turns out to be incorrect, then u must be reevaluated later to obtain its correct value. In contrast, a conservative algorithm makes sure that all of u's predecessors are correctly evaluated before u is evaluated. This invariant ensures that each element is evaluated at most once. Maintaining this invariant, however, requires stringent synchronization. Our optimistic algorithm avoids this synchronization cost and thus performs well when the optimistic assumption is often correct.

We prove the following theorem about the behavior of

This research was supported in part by the Advanced Research Projects Agency under Grant N00014-94-1-0985. Keith H. Randall is supported in part by a Department of Defense NDSEG Fellowship.

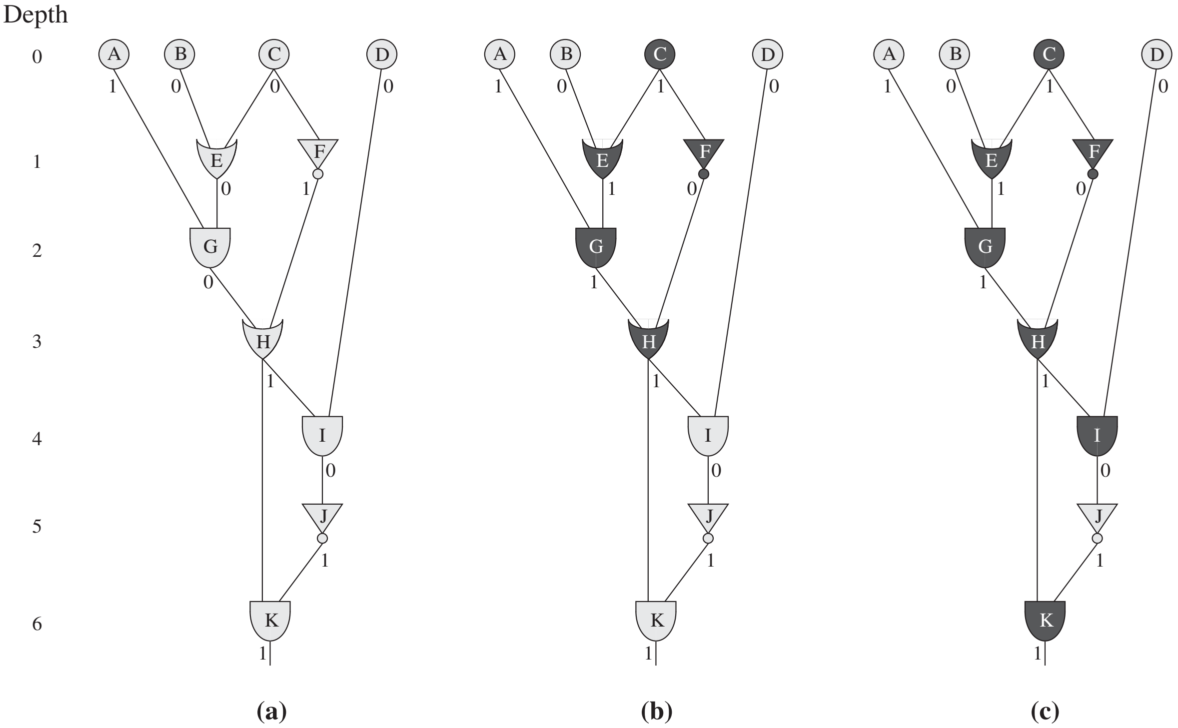

Figure 2: A combinational logic circuit illustrating the circuit value update problem. **(a)** The original circuit whose elements are labeled with their values. The depths of elements are shown on the left side. **(b)** The updated circuit when Input C changes from a 0 to a 1. The shaded elements have an input change value as a result of the update. **(c)** The shaded elements represent the volatile subcircuit. The outputs of Gates H and K may glitch.

the optimistic algorithm:

Theorem 1 *Suppose that the optimistic algorithm run on a computer with P processors performs an update on a bounded-degree circuit $G = (V, E)$, and suppose that W and D are the size and embedded depth, respectively, of the volatile subcircuit. Then, for all $\epsilon > 0$, the probability is at most ϵ that the optimistic algorithm runs in more than $O(W/P + D + \lg W + \lg P + \lg(1/\epsilon))$ time.*

Proof. For a complete proof, we refer the reader to [4]. □

Our optimistic, asynchronous, parallel algorithm is a simple and natural generalization of the serial algorithm, and we find it remarkable that it works as well as it does. To our knowledge, our analysis provides the first analytical bounds on the running time of an optimistic, asynchronous algorithm.

We are optimistic that our proof techniques can be applied to obtain bounds on other optimistic algorithms, such as simulation by time-warping [3, 5, 2]. For example, we are studying the problem of extending our results, which currently are restricted to combinational circuits, to optimistic simulation of sequential circuits.

References

[1] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11):198–206, April 1981.

[2] K. Choi, S. Y. Hwang, and T. Blank. Incremental-in-time algorithm for digital simulation. In *Proc. 25th Design Automation Conference*, pages 501–505, June 1988.

[3] David R. Jefferson. Virtual time. *ACM Transactions of Programming Languages and Systems*, 7(3):404–425, July 1985.

[4] Charles E. Leiserson and Keith H. Randall. Parallel algorithms for the circuit value update problem. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, Santa Barbara, California, July 1995. To appear.

[5] Yukinori Matsumoto and Kazuo Taki. Parallel logic simulation on a distributed memory machine. In *Proceedings of the European Conference on Design Automation*, pages 76–80, 1992.

[6] Patrick Henry Winston. *Artificial Intelligence*. Addison-Wesley, 1992.