A Parallel Computation Approach to Topological Sorting

M. C. Er
Department of Computing Science, The University of Wollongong, PO Box 1144, Wollongong 2500, NSW, Australia

A new topological sorting algorithm is formulated using the parallel computation approach. The time complexity of this algorithm is of the order of the longest distance between a source node and a sink node in an acyclic digraph representing the partial orderings between elements. An implementation of this algorithm with an SIMD machine is discussed. To avoid contention for logical resources, a synchronization of all processors is proposed and its performance is also discussed.

INTRODUCTION

The topological sort is a computation of a linearization of a directed acyclic graph subject to the constraints of partial orderings embedded in the graph. A few techniques for computing the topological sorting have been documented in the literature.[1-6] They are all based on the sequential computation approach.

This paper presents a new topological sort based on the parallel computation approach. As we shall see below, this new approach is conceptionally very simple. It can be easily adapted to other applications involving digraph traversal. An implementation of this parallel algorithm with a single instruction stream–multiple data stream machine and its performance are also discussed.

THE ALGORITHM

The topological sorting operates on a digraph of partial orderings. The basic idea behind the parallel computation approach is to traverse in parallel down all links leading from a node, once the node concerned is visited. Inductively, all nodes in the digraph will be visited if all source nodes are visited. If the digraph is acyclic, the parallel traversal will terminate, as the number of nodes in a digraph is finite.

The algorithm is organized in two phases. In the first phase, pairs of nodes which have partial orderings between them are read in. A digraph which represents the partial orderings is set up. In the second phase, the parallel topological sorting takes place. The details of the algorithm are outlined below. Here let N_i be node i in the digraph.

Algorithm (parallel topological sort)

Step 1. Read in pairs of partial orderings and set up a digraph representation of the relationships as follows:

If $a < b$, place a directed link from N_b to N_a
If $a > b$, place a directed link from N_a to N_b

Step 2. Initialize all node values to zero, and find all the source nodes of the digraph (if no source node exists, the digraph is obviously cyclic; exit).

Step 3. Visit all the source nodes, and change their node values to ones.

Step 4. Follow down the directed links from all the nodes N_p just visited and visit all of their successor nodes N_s, in parallel. Update the node value of each successor node N_s as follows: If the node value of $N_s \leq$ the node value of N_p, then update the node value of N_s to 1 + the node value of N_p; otherwise do nothing.

Iterate step 4 until the computation is converged or the node value of a node is assigned a value larger than the total number of nodes in the digraph. The latter obviously signifies that the digraph is cyclic and should be reported accordingly.

Step 5. List all the nodes in ascending order of node values.

This algorithm can be proved correct logically. Since all successor nodes carry node values greater than that of their predecessor nodes, consequently they will be listed last with respect to their predecessor nodes in the linearized form. Furthermore, the digraph traversal is bound to terminate, in the absence of cyclic directed links, as the number of nodes in a digraph and the longest distance between a source node and a sink node are finite.

IMPLEMENTATION

Because of the parallel computation nature of the algorithm, it is attractive to implement it with a multiprocessor multi-memory system. Now, we discuss problems arising in implementing this algorithm.

In recent years, single instruction stream–multiple data stream (SIMD) machines have become increasingly popular.[7-8] Typically, an SIMD machine is a computer system consisting of N processors, N memory modules, a control unit, and an interconnection network. The interconnection network provides a communication facility for the processors and the memory modules; and it may be positioned either between the processors and the memory modules or between N processing elements, where each processing element consists of a processor and its own memory. The control unit broadcasts

CCC-0010-4620/83/0026-0293$01.50

identical instructions to all processors; and thus all active processors execute the same instruction at the same time.

To minimize the memory contention in running the topological sorting algorithm, it is necessary to assign each node of the digraph to a memory module.[9] As such, node values of all nodes can be updated in parallel.

When executing the algorithm in parallel, there is a problem that two or more processors may have a common successor node and thus may try to test and update the node value of this successor node at the same time. To obtain the correct result, the test and update operation could be made a point operation through the use of critical regions or some other mechanism for ensuring mutual exclusion.[10] This, however, results in a performance degradation, since on each node access the processors must co-ordinate. An alternative solution to this is to synchronize execution of the algorithm. If we assume that all processors complete one stage 4 iteration before all processors can start the next iteration, then a common successor node is reached either in a later iteration by the processor chaining down the longer path or at the same iteration by two or more processors chaining down the same distance from the source nodes. In the former case there is no interference. In the latter case there can be interference and only one of them must increment the node value. By postulating a finer degree of synchronization, we can overcome this. Thus, if the algorithm is implemented with an SIMD machine with an interconnection network that can allow any mapping of processors to memory modules including one to many,[11] each active processor fetches the same old node value before starting to update this value. We may formalize the above results in the following theorem.

Theorem 1

The parallel topological sort when implemented with an SIMD machine is free of contention, provided all processors are synchronized and each node value is stored in a memory module.

Proof. The contention may occur when two (or more) processors try to update a common memory with different values simultaneously. Without loss of generality, we shall consider the contention caused by two processors only.

Suppose two nodes, N_i and N_j, have a common successor node N_k. Let V_i, V_j and V_k be the node values of N_i, N_j and N_k, respectively, such that $V_i > V_j \geq V_k$. The contention occurs when processor 1 tests for $V_k \leq V_i$ and then updates V_k to $V_i + 1$, whereas processor 2 simultaneously tests for $V_k \leq V_j$ and then updates V_k to $V_j + 1$. If the test and update operations of these two processors are allowed to interleave, then the resulting V_k is either $V_i + 1$ or $V_j + 1$. In consequence, the parallel topological sort may fail. To prove that this algorithm is free of contention if each node value is stored in a memory module, we need to show: (i) the contention cannot occur if $V_i \neq V_j$; and (ii) if the contention can occur, then $V_i = V_j$.

We prove it by *reductio ad absurdum*. Suppose that the contention does occur even if $V_i \neq V_j$. We assume that this event happens at time t. We further assume, without loss of generality, that each cycle of propagation of node values takes one unit of time. By the basic assumption of the theorem that all processors are synchronized, the chains of propagation of node values leading to V_i and V_j must start at times $(t - V_i)$ and $(t - V_j)$, respectively, from some source nodes. Since $(t - V_i) \neq (t - V_j)$, it contradicts the algorithm. Therefore the contention cannot occur if $V_i \neq V_j$.

If, however, the propagation of node values starts from all the source nodes simultaneously at time t', the propagations of node values from nodes N_i and N_k happen at times $(t' + V_i)$ and $(t' + V_j)$, respectively. If these two propagations reach the same node at the same time, therefore $(t' + V_i) = (t' + V_j)$. That is $V_i = V_j$. Since processors 1 and 2 attempt to update V_k to the same node value $V_i + 1$ $(V_j + 1)$, it follows that only one of the processors needs to update the node value.

Q.E.D.

EXAMPLE

An example will help to put the picture in perspective. The digraph shown in Fig. 1 is adopted from Ref. 2.

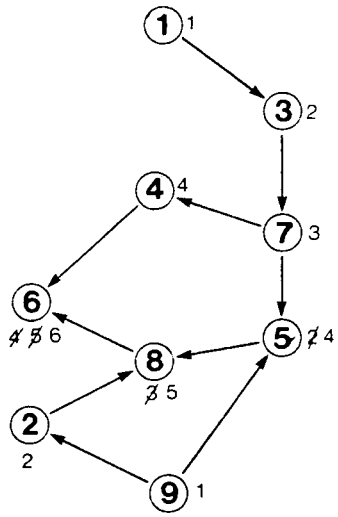

Figure 1. A digraph showing the propagation of node values starting from N_1 and N_9.

The numbers written beside the nodes are node values. Node values that have been crossed out represent superseded values. The propagation of node values starts from nodes N_1 and N_9. The algorithm terminates when the sink node N_6 is assigned a node value equal to 6. Incidentally, the largest node value also equals the longest distance between a source node and a sink node in the digraph.

A linearized arrangement of the nodes in ascending order of node values is 1 9 2 3 7 4 5 8 6.

ANALYSIS

The parallel topological sort relies on the propagation of node values from all source nodes to all sink nodes. Let D_{max} be the maximum distance between a source node and a sink node in a digraph concerned. Obviously, D_{max} equals the maximum node value assigned to a sink node.

The number of cycles of propagations is thus equal to D_{max}. Therefore, the running time of the algorithm is $O(D_{max})$. In the normal digraph context, $D_{max} < N$. However, in the worst case when all nodes line up as a chain, $D_{max} = N$.

The actual performance of this implementation may be less than optimum. The synchronized mode of operation we proposed above for avoiding contention for logical resources carries a penalty. The synchronization forces all processors to take the time of the processor handling the node with the most successor nodes. Clearly, some processors must wait, and this is a waste of processor power.

If, however, the synchronization is abandoned, Theorem 1 is no longer valid. Two or more processors may try to update the node value of the same node concurrently with different values. In order to preserve the correctness of the node value, only one processor at a time is allowed to be in the test and update cycle. In other words, this test and update cycle on a node value must appear like a point operation; or equivalently mutual exclusion must be achieved over access to a node value. The processors accessing a common successor node thus contend to update that node value and clearly some processors must wait. Such a contention for logical resources will result in performance degradation.[12,13]

On the balance, the synchronization approach seems to offer a better compromise.

CONCLUDING REMARKS

The parallel topological sort is based on a very simple computational concept: the propagation of node values from all source nodes to all sink nodes in a given digraph. This algorithm may be implemented with an SIMD machine, and takes advantage of its parallel architecture. The time complexity of the parallel topological sorting turns out to be of the order of the longest distance between a source node and sink node in an acyclic digraph. It has been shown that the synchronization approach or the point operation approach for avoiding contention caused by two or more processors accessing a common successor node degrade the performance somewhat. Of course, the parallel topological sorting algorithm could also be implemented with a conventional single-processor single-memory computer system. The resulting sequential computation is effectively a simulation of the parallel computation.

Acknowledgements

The author is indebted to the referee for his helpful and valuable comments. This research was supported by the Research Grants Committee under grant 05-143-105.

REFERENCES

1. A. B. Kahn, Topological sorting of large networks. *Communications of ACM* **5**, 558–562 (1962).
2. D. E. Knuth, Fundamental algorithms. *The Art of Computer Programming 1*, Addison-Wesley, Reading, Mass. (1973).
3. D. E. Knuth and J. L. Szwarcfiter, A structured program to generate all topological sorting arrangements. *Information Processing Letters* **2**, 153–157 (1974).
4. E. M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, New Jersey (1977).
5. Y. L. Varol and D. Rotem, An algorithm to generate all topological sorting arrangements. *The Computer Journal* **24**, 83–84 (1981).
6. N. Wirth, *Algorithms + Data Structures = Programs*, Prentice-Hall, New Jersey (1977).
7. H. J. Siegel, Interconnection networks for SIMD machines. *Computer* **12**, 57–65 (June 1979).
8. H. J. Siegel, A model of SIMD machines and a comparison of various interconnection networks. *IEEE Transactions on Computers* **c-28**, 907–917 (1979).
9. D. L. Lawrie, The prime memory system for array access. *IEEE Transactions on Computers* **c-31**, 435–442 (1982).
10. P. Brinch Hansen, *Operating System Principles*, Prentice-Hall, New Jersey (1973).
11. D. H. Lawrie, Access and alignment of data in an array processor. *IEEE Transactions on Computers* **c-24**, 1145–1155 (1975).
12. F. Baskett and A. J. Smith, Interference in multiprocessor computer systems with interleaved memory. *Communications of ACM* **19**, 327–334 (1976).
13. D. P. Bhandarkar, Analysis of memory interference in multiprocessors. *IEEE Transactions on Computers* **c-24**, 897–908 (1975).

Received June 1982