A DIVIDE–AND–CONQUER ALGORITHM FOR IDENTIFYING STRONGLY CONNECTED COMPONENTS*

LISA K. FLEISCHER†, BRUCE HENDRICKSON‡, AND ALİ PINAR§

Abstract. The standard serial algorithm for strongly connected components has linear complexity and is based on depth first search. Unfortunately, depth first search is difficult to parallelize. We describe a divide–and–conquer algorithm for this problem which has significantly greater potential for parallelization. We show the expected serial running time of our algorithm to be $O(|E| \log |V|)$. We also present a variant of our algorithm that has $O(|E| \log |V|)$ worst–case complexity.

Key words. Strongly connected components, divide–and–conquer, parallel algorithm, discrete ordinates method

AMS subject classifications. 05C85, 05C38, 68W10, 68W20

1. Introduction.

A *strongly connected component* of a directed graph is a maximal subset of vertices containing a directed path from each vertex to all others in the subset. The vertices of any directed graph can be partitioned into a set of disjoint strongly connected components. This decomposition is a fundamental tool in graph theory with applications in compiler analysis, data mining, scientific computing and other areas.

The definitive serial algorithm for identifying strongly connected components is due to Tarjan [17] and is built on a depth first search of the graph. For a graph with n vertices and m edges, this algorithm runs in the optimal $O(m)$[1], time, and is widely used in textbooks as an example of the power of depth first search [7].

For large problems, a parallel algorithm for identifying strongly connected components would be useful. One application of particular interest to us is discussed below. Unfortunately, depth first search (DFS) seems to be difficult to parallelize. Reif shows that a restricted version of the problem (lexicographical DFS) is $\mathcal{P}$-Complete [16]. However, Aggarwal and Anderson, and Aggarwal et al. describe randomized NC algorithms for finding a DFS of undirected and directed graphs, respectively [1, 2]. The expected running time of this latter algorithm is $O(\log^7 n)$ and it requires an impractical $n^{2.376}$ processors. To our knowledge, the deterministic parallel complexity of DFS for general, directed graphs is an open problem. Chaudhuri and Hagerup studied the problem for acyclic [5], and planar graphs [11], respectively. More practically, DFS is a difficult operation to parallelize and we are aware of no algorithms or implementations which perform well on large numbers of processors. Consequently, Tarjan's algorithm is not evidently parallelizable.

Alternatively, there exist several parallel algorithms for the strongly connected

* This work was funded by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research and performed at Sandia, a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the U.S. DOE under contract number DE-AC-94AL85000.

†GSIA, Carnegie Mellon University, Pittsburgh, PA 15213, email: lkf@andrew.cmu.edu

‡Parallel Computing Sciences, Sandia National Labs, Albuquerque, NM 87185-1110, email: bah@cs.sandia.gov

§Dept. Computer Science, University of Illinois, Urbana, IL 61801, email: alipinar@cse.uiuc.edu

[1]Function $f(n) = \Theta(g(n))$ if there exist constants $c_2 \geq c_1 > 0$ and N such that for all $n \geq N$, $c_1 g(n) \leq f(n) \leq c_2 g(n)$. If $f(n) = \Omega(g(n))$, then this just implies the existence of c_1 and N. If $f(n) = O(g(n))$, then c_2 and N exist.

components problem (SCC) that avoid the use of depth first search. Gazit and Miller devised an NC algorithm for SCC, which is based upon matrix–matrix multiplication [10]. This algorithm was improved by Cole and Vishkin [6], but still requires $n^{2.376}$ processors and $O(\log^2 n)$ time. Kao developed a more complicated NC algorithm for planar graphs that requires $O(\log^3 n)$ time and $n/\log n$ processors [12]. More recently, Bader has an efficient parallel implementation of SCC for planar graphs [3] which uses a clever *packed–interval* representation of the boundary of a planar graph. When n is much larger than p the number of processors, Bader's approach scales as $O(n/p)$. But Bader's approach does not apply to general graphs.

Our interest in the SCC problem is motivated by the discrete ordinates method for modeling radiation transport. Using this methodology, the object to be studied is modeled as a union of polyhedral finite elements. Each element is a vertex in our graph and an edge connects any pair of elements that share a face. The radiation equations are approximated by an angular discretization. For each angle in the discretization, the edges in the graph are directed to align with the angle. The computations associated with an element can be performed if all its predecessors have been completed. Thus, for each angle, the set of computations are sequenced as a topological sort of the directed graph. A problem arises if the topological sort cannot be completed — i.e. the graph has a cycle. If cycles exist, the numerical calculations need to be modified — typically by using old information along one of the edges in the cycle, thereby removing the dependency. So identifying strongly connected components quickly is essential. Since radiation transport calculations are computationally and memory intensive, parallel implementations are necessary for large problems. Also, since the geometry of the grid can change after each timestep for some applications, the SCC problem must be solved in parallel.

Efficient parallel implementations of the topological sort step of the radiation transport problem have been developed for *structured* grids, oriented grids that have no cycles [4, 8]. Some initial attempts to generalize these techniques to unstructured grids are showing promise [14, 15]. It is these latter efforts that motivated our interest in the SCC problem.

In the next section we describe a simple divide-and-conquer algorithm for finding strongly connected components. In §3 we show that this algorithm has an expected serial complexity of $O(m \log n)$. In §4 we present a modification of our algorithm and show that it has worst case serial complexity of $O(m \log n)$. Our approach has good potential for parallelism for two reasons. First, the divide–and–conquer paradigm generates a set of small problems which can be solved independently by separate processors. Second, the basic step in our algorithm is a reachability analysis, which is similar to topological sort in its parallelizability. So we expect the current techniques for parallelizing radiation transport calculations to enable our algorithm to perform well too. A preliminary version of this work can be found in [9].

2. A Divide–and–Conquer Algorithm. Before describing our algorithm, we introduce some notation. Let $G = (V, E)$ be a directed graph with vertices V and directed edges E. An edge $(i, j) \in E$ is directed from i to j. We denote the set of strongly connected components of G by $SCC(G)$. Thus $SCC(G)$ is a partition of V. We also use $SCC(G, v)$ to denote the (unique) strongly connected component containing vertex v. We denote by $V \setminus X$ the subset of vertices in V which are not in a subset X. The size of vertex set X is denoted $|X|$.

A vertex v is *reachable* from a vertex u if there is a sequence of directed edges $(u, x_1), (x_1, x_2), \ldots, (x_k, v)$ from u to v. We consider a vertex to be reachable from

itself. Given a vertex $v \in V$, the *descendants* of v, $Desc(G, v)$, is the subset of vertices in G which are reachable from v. Similarly, the *predecessors* of v, $Pred(G, v)$, is the subset of vertices from which v is reachable. The set of vertices that is neither reachable from v nor reach v is called the *remainder*, denoted by $Rem(G, v) = V \setminus (Desc(G, v) \cup Pred(G, v))$.

Given a graph $G = (V, E)$ and a subset of vertices $V' \subseteq V$, the *induced subgraph* $G' = (V', E')$ contains all edges of G connecting vertices of V', i.e. $E' = \{(u, v) \in E : u, v \in V'\}$. We will use $\langle V' \rangle = G' = (V', E')$ to denote the subgraph of G induced by vertex set V'. The following lemma is an immediate consequence of the definitions.

LEMMA 2.1. *Let $G = (V, E)$ be a directed graph, with $v \in V$ a vertex in G. Then $Desc(G, v) \cap Pred(G, v) = SCC(G, v)$.*

LEMMA 2.2. *Let G be a graph with vertex v. Any strongly connected component of G is a subset of $Desc(G, v)$, a subset of $Pred(G, v)$ or a subset of $Rem(G, v)$.*

Proof. Let u and w be two vertices of the same strongly connected component in G. By definition, u and w are reachable from each other. The proof involves establishing $u \in Desc(G, v) \iff w \in Desc(G, v)$ and $u \in Pred(G, v) \iff w \in Pred(G, v)$, which then implies $u \in Rem(G, v) \iff w \in Rem(G, v)$. Since the proofs of these two statements are symmetric, we give just the first: If $u \in Desc(G, v)$ then u must be reachable from v. But then w must also be reachable from v, so $w \in Desc(G, v)$. □

With this background, we can present our algorithm which we call DCSC (for Divide–and–Conquer Strong Components). The algorithm is sketched in Fig. 2.1. The basic idea is to select a random vertex v, which we will call a *pivot* vertex, and find its descendant and predecessor sets. The intersection of these sets is $SCC(G, v)$ by Lemma 2.1. After this step, the remaining vertices are divided into three sets $Desc(G, v)$, $Pred(G, v)$, and $Rem(G, v)$. By Lemma 2.2, any additional strongly connected component must be entirely contained within one of these three sets, so we can divide the problem and recurse.

```
DCSC(G)
    If G is empty then Return
    Select v uniformly at random from V
    SCC ← Pred(G, v) ∩ Desc(G, v)
    Output SCC.
    DCSC(⟨Pred(G, v) \ SCC⟩)
    DCSC(⟨Desc(G, v) \ SCC⟩)
    DCSC(⟨Rem(G, v)⟩)
```

FIG. 2.1. *A divide–and–conquer algorithm for strongly connected components.*

As mentioned above, this algorithm is amenable to practical parallelization on two levels. First, the recursive invocations are completely independent and so can execute independently. Second, the searches for predecessors and descendants allows for much more parallelism than does a depth first search.

3. Serial Complexity of Algorithm DCSC. The cost of algorithm DCSC can be described by four terms, one each from the three recursive invocations and a fourth from the determination of predecessors and descendants. Let $n(G)$ and $m(G)$ denote the number of vertices and edges in graph G, respectively. We will use n and m to denote the number of vertices and edges in the original graph, when it is clear

from the context. Let $T(G)$ be the run time of the algorithm on a graph G. For a pivot vertex i, let $P_i = Pred(G,i) \setminus SCC(G,i)$, $D_i = Desc(G,i) \setminus SCC(G,i)$ and $R_i = Rem(G,i)$ be the graphs for the recursive calls. The recursive expression for the run time is

$$T(G) = T(R_i) + T(D_i) + T(P_i) + \Theta(m(G) - m(R_i)). \tag{3.1}$$

Clearly, $T(G) = \Omega(m+n)$, since we eventually must look at all vertices and edges. Also, in the worst case $T(G) = O(mn)$, since each iteration takes at most linear time in the number of edges and reduces the number of vertices by at least 1. We show here that the expected behavior of $T(G)$ is $O((m+n)\log n)$. We will accomplish this by showing that each edge is visited $O(\log n)$ times on average. The expected case analysis will require bounds on the sizes of the predecessor and descendant sets. The following two results provide these bounds.

LEMMA 3.1. *For a directed graph G, there is a numbering π of the vertices from 1 to n in which the following is true. For all $v \in V$, all elements $u \in Pred(G,v) \setminus Desc(G,v)$ satisfy $\pi(u) < \pi(v)$; and all elements $u \in Desc(G,v) \setminus Pred(G,v)$ satisfy $\pi(u) > \pi(v)$.*

Proof. If G is acyclic, then a topological sort provides a numbering with this property. If G has cycles, then each strongly connected component can be contracted into a single vertex, and the resulting acyclic graph can be numbered via topological sort. Assume a strongly connected component with k vertices was assigned a number j in this ordering. Assign the vertices within the component the numbers $(j, \ldots, j+k-1)$ arbitrarily and increase all subsequent numbers by $k-1$. □

It is important to note that we do not need to construct an ordering with this property; we just need to know that it exists.

COROLLARY 3.2. *Given a directed graph G and a vertex numbering π from Lemma 3.1, then $n(Pred(G,v) \setminus SCC(G,v)) < \pi(v)$ and $n(Desc(G,v) \setminus SCC(G,v)) \leq n(G) - \pi(v)$ for all vertices v.*

THEOREM 3.3. *Algorithm DCSC has expected time complexity $O((m+n)\log n)$.*

Proof. We assume without loss of generality that $n = O(m)$, since otherwise we can analyze the algorithm separately for each connected component. We will bound the expected number of visits to an edge by $O(\log n)$, and the theorem follows. Edges in the predecessor set P and descendant set D are visited a constant c times at each recursive invocation of DCSC, and edges in R are not visited. Let $X(G)$ be the average number of times an edge is visited in algorithm DCSC. Then

$$X(G) = \frac{1}{m}\{m_p(X(P)+c) + m_d(X(D)+c) + m_r X(R) + m_s c\}, \tag{3.2}$$

where m_p, m_d and m_r, respectively, are the number of edges in the predecessor, descendant and remainder sets, and m_s is the number of edges in the strongly connected component and edges between different components.

Since DCSC is a randomized algorithm, we will perform an expected case analysis by summing the cost over all n possible pivot vertices and dividing by n. We will show that the expected value of $X(G)$, $EX(G) \leq a\log n$ for some constant a. As the inductive hypothesis, we will assume this bound holds for all graphs with fewer than n vertices.

$$EX(G) = \frac{1}{n}\sum_{1\leq i\leq n}\frac{1}{m}\{m_p(EX(P_i)+c) + m_d(EX(D_i)+c) + m_r EX(R_i) + m_s c\}$$

$$\leq \frac{1}{n} \sum_{1 \leq i \leq n} \frac{1}{m}(m_p + m_d + m_r + m_s) \max\{EX(P_i) + c, EX(D_i) + c, EX(R_i), c\}$$
$$\leq \frac{1}{n} \sum_{1 \leq i \leq n} \max\{EX(P_i) + c, EX(D_i) + c, EX(R_i)\}$$

Using the inductive hypothesis and Corollary 3.2, we find that

$$EX(n) \leq \frac{1}{n} \sum_{1 \leq i \leq n} \max\{a \log i + c, a \log(n - i) + c, a \log n\}.$$

Using symmetry,

$$EX(n) = \frac{2}{n} \sum_{n/2 \leq i \leq n} \max\{a \log i + c, a \log n\}.$$

The first term dominates when $i > \frac{n}{2^{\frac{c}{a}}}$. Let $d = \frac{1}{2^{\frac{c}{a}}}$.

$$\begin{aligned} EX(n) &= \frac{2}{n} \sum_{n/2 \leq i \leq n} \max\{a \log i + c, a \log n\} \\ &= \frac{2}{n}\Big(\sum_{n/2 \leq i < dn} \max\{a \log i + c, a \log n\} + \sum_{dn \leq i \leq n} \max\{a \log i + c, a \log n\}\Big) \\ &= \frac{2}{n}\Big(\sum_{n/2 \leq i < dn} a \log n + \sum_{dn \leq i \leq n} a \log i + c\Big) \end{aligned} \tag{3.3}$$

To bound the second term, we will split the summation into three pieces, each of span $(1 - d)n/3$.

$$\begin{aligned} \sum_{dn \leq i \leq n} a \log i + c &= \sum_{dn}^{\frac{2d+1}{3}n} (a \log i + c) + \sum_{\frac{2d+1}{3}n}^{\frac{d+2}{3}n} (a \log i + c) + \sum_{\frac{d+2}{3}n}^{n} (a \log i + c) \\ &\leq \sum_{dn}^{\frac{2d+1}{3}n} (a \log(\frac{2d+1}{3}n) + c) + \sum_{\frac{2d+1}{3}n}^{\frac{d+2}{3}n} (a \log(\frac{d+2}{3}n) + c) \\ &\quad + \sum_{\frac{d+2}{3}n}^{n} (a \log n + c) \\ &\leq \frac{(1-d)n}{3}(a \log n + a \log \frac{2d+1}{3} + c) + \frac{(1-d)n}{3}(a \log n + \log \frac{d+2}{3} + c) \\ &\quad + \frac{(1-d)n}{3}(a \log n + c) \\ &\leq (1-d)na \log n + \frac{(1-d)an}{3}(\log \frac{2d+1}{3} + \log \frac{d+2}{3} + \frac{3c}{a}) \end{aligned}$$

We want $\log \frac{2d+1}{3} + \log \frac{d+2}{3} + \frac{3c}{a} \leq 0$, or equivalently,

$$\log \frac{3}{2d+1} + \log \frac{3}{d+2} \geq \frac{3c}{a}$$

$$\frac{9}{(2d+1)(d+2)} \geq (2^{\frac{c}{a}})^3$$
$$\frac{9}{2d^2+3d+2} \geq \frac{1}{d^3}$$
$$9d^3 \geq 2d^2+3d+2$$

which holds for $d = \frac{8}{9}$. Setting $a \geq \frac{c}{\log \frac{9}{8}}$, we find that

$$\sum_{dn \leq i \leq n} a \log i + c \leq (1-d)na \log n \tag{3.4}$$

Combining equations (3.3) and (3.4),

$$\begin{aligned} EX(n) &\leq \frac{2}{n}\Big(\sum_{n/2 \leq i < dn} a \log n + \sum_{dn \leq i \leq n} a \log i + c\Big) \\ &\leq \frac{2}{n}\{\frac{(2d-1)n}{2} a \log n + (1-d)na \log n\} \\ &= \frac{2}{n}\frac{n}{2} a \log n \\ &= a \log n \end{aligned}$$

□

4. A worst case $O(m \log n)$ time algorithm. In this section, we modify the algorithm from §2 to achieve a worst case $O(m \log n)$ time complexity. The basic idea of the modification is to avoid completing long searches for predecessors and descendants. Assume without loss of generality that the predecessor set has fewer edges than the descendant set. If we knew this, we could first identify all the predecessors, and then limit the successor search to only visit predecessor vertices. The intersection of these two searches is a strongly connected component. Once this is done, we can recurse by dividing the graph into two pieces. The first piece is the predecessor set minus the strongly connected component. The second set is the full graph minus the predecessors. By Lemma 2.2, there can be no strongly connected components which cross between these two sets. In practice, we don't know whether we should search for predecessors or successors first. So instead, we will search for both simultaneously and abandon the second search when the first one finishes. The resulting algorithm is sketched in Fig. 4.1.

THEOREM 4.1. *Algorithm WDCSC presented in Fig. 4.1 runs in worst case $O(m \log n)$-time.*

Proof. Let $R(m)$ be the run time of this algorithm on a graph with m edges. We will use p_i , d_i and s_i to denote the total number of edges adjacent to $P_i \setminus SCC(G,i)$, $D_i \setminus SCC(G,i)$ and $SCC(G,i)$, respectively. By symmetry, we can assume $p_i \leq d_i$. Then $R(m)$ can be expressed as

$$R(m) \leq R(p_i) + R(m - p_i - s_i) + \Theta(p_i + s_i) \tag{4.1}$$

since the amount of work in one iteration is proportional to the size of the minimum of the predecessor set and the descendant set. That is, the amount of work is twice $p_i + s_i$ until the algorithm completes one search, and then is at most s_i. Note that by hypothesis $p_i \leq m/2$.

```
WDCSC(G)
    If G is empty then Return
    Select v uniformly at random from V
    Simultaneously search for Pred(G, v) and Desc(G, v)
    If Pred(G, v) finishes first
        Search for Desc(G, v) only in Pred(G, v).
    Else If Desc(G, v) finishes first
        Search for Pred(G, v) only in Desc(G, v).
    SCC ← Pred(G, v) ∩ Desc(G, v)
    Output SCC.
    If Pred(G, v) finished first
        WDCSC(⟨Pred(G, v) \ SCC⟩)
        WDCSC(⟨V \ Pred(G, v)⟩)
    Else If Desc(G, v) finished first
        WDCSC(⟨Desc(G, v) \ SCC⟩)
        WDCSC(⟨V \ Desc(G, v)⟩)
```

FIG. 4.1. *A worst case $O(m \log n)$ time algorithm to find strongly connected components*

We will show that $R(m) = O(m \log m)$ by induction, considering two cases depending on the size of p_i. Since $m = O(n^2)$, this leads to our claimed bound.

Case 1: $p_i + s_i \leq m/2$.

If $p_i + s_i \leq m/2$, then we can upper bound $R(m)$ by substituting $a = p_i + s_i$ in (4.1) to get

$$R(m) \leq R_2(m, a) := R(a) + R(m - a) + \Theta(a)$$

By our assumptions in this case, we have that $1 \leq a \leq m/2$. We show that this recursion is $O(m \log m)$ by first showing that this holds for $a \in \{1, m/2\}$, and then showing that R_2, as a function of a in the range $[1, m-1]$, is convex. Thus, its value in an interval is bounded from above by its value at the endpoints of the interval.

We suppose, by induction, that $R(r) = c_1 r \log r$ for an appropriate constant c_1 for $r < m$, and that $\Theta(r) = c_2 r$. For $a = 1$, we have that $R(m) \leq R(m-1) + \Theta(1)$, which is $O(m \log m)$ by the induction hypothesis. When $a = m/2$, the recursion becomes $R(m) \leq 2R(m/2) + \Theta(m)$, which is well-known to lead to $R(m) = O(m \log m)$. The first derivative of $R_2(m, a)$ with respect to a is

$$c_1(\log a - \log(m - a)) + c_2.$$

The second derivative is

$$c_1(\frac{1}{a} + \frac{1}{m - a}),$$

which is positive for $a \in [1, m-1]$. Thus $R_2(m, a)$ is convex for $a \in [1, m/2]$, and hence $R(m) = O(m \log m)$ in this case.

Case 2: $p_i + s_i > m/2$.

In this case, we can bound $R(m)$ by

$$\begin{aligned} R(m) &= R(p_i) + R(m - p_i - s_i) + \Theta(p_i + s_i) \\ &\leq 2R(m/2) + \Theta(m) \end{aligned}$$

since $p_i \leq m/2$. As before, we have a well–known recursion whose solution is $R(m) = O(m \log m)$.
□

5. Conclusions. The simple divide–and–conquer algorithm described in 2 has recently been implemented by McLendon, et al. [13], and good parallel performance is observed for problems arising from the radiation transport application described in §1. Several open questions remain. Despite our conviction that our divide–and–conquer approach is amenable to effective parallelization, we have not shown that it leads to an improved NC algorithm. Specifically, the reachability analyses at the core of our method can be solved in NC via Gazit and Miller's matrix product approach [10], but their technique can be used to find strongly connected components directly. Another question of interest to us is the potential for a linear time algorithm for strongly connected components that does not rely on depth first search. Any such algorithm could be a candidate for an improved parallel approach.

Acknowledgments. We benefited from general discussions about algorithms for parallel strongly connected components with Steve Plimpton, Will McLendon and David Bader. We are also indebted to Bob Carr, Cindy Phillips, Bill Hart and Sorin Istrail for discussions about the analysis.

REFERENCES

[1] A. Aggarwal and R. J. Anderson, *A random NC algorithm for depth first search*, Combinatorica, 8 (1988), pp. 1–12.
[2] A. Aggarwal, R. J. Anderson, and M.-Y. Kao, *Parallel depth-first search in general directed graphs*, SIAM J. Comput., 19 (1990), pp. 397–409.
[3] D. A. Bader, *A practical parallel algorithm for cycle detection in partitioned digraphs*, Tech. Rep. Technical Report AHPCC-TR-99-013, Electrical & Computer Engng. Dept., Univ. New Mexico, Albuquerque, NM, 1999.
[4] R. S. Baker and K. R. Koch, *An S_n algorithm for the massively parallel CM-200 computer*, Nuclear Science and Engineering, 128 (1998), pp. 312–320.
[5] P. Chaudhuri, *Finding and updating depth-first spanning trees of acyclic digraphs in parallel*, The Computer Journal, 33 (1990), pp. 247–251.
[6] R. Cole and U. Vishkin, *Faster optimal prefix sums and list ranking*, Information and Computation, 81 (1989), pp. 334–352.
[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press and McGraw-Hill, Cambridge, MA, 1990.
[8] M. R. Dorr and C. H. Still, *Concurrent source iteration in the solution of 3-dimensional, multigroup discrete ordinates neutron-transport equations*, Nuclear Science and Engineering, 122 (1996), pp. 287–308.
[9] L. K. Fleischer, B. Hendrickson and A. Pinar, *On Identifying Strongly Connected Components in Parallel*, in Solving Irregularly Structured Problems in Parallel: 7th International Symposium, Irregular '00, Lecture Notes in Computer Science", Vol. 1800, pp. 505–512, Springer-Verlag, 2000.
[10] H. Gazit and G. L. Miller, *An improved parallel algorithm that computes the BFS numbering of a directed graph*, Inform. Process. Lett., 28 (1988), pp. 61–65.
[11] T. Hagerup, *Planar depth-first search in $O(\log n)$ parallel time*, SIAM J. Comput., 19 (1990), pp. 678–704.
[12] M.-Y. Kao, *Linear-processor NC algorithms for planar directed graphs I: Strongly connected components*, SIAM J. Comput., 22 (1993), pp. 431–459.
[13] W. McLendon III, B. Hendrickson, S. Plimpton, and L. Rauchwerger. *Identifying strongly connected components in parallel.* In *Proc. 10th SIAM Conf. Parallel Processing for Sci. Comput.*, March 2001.
[14] S. Pautz. Personal Communication, October 1999.
[15] S. Plimpton. Personal Communication, May 1999.
[16] J. H. Reif, *Depth-first search is inherently sequential*, Inform. Process. Lett., 20 (1985), pp. 229–234.

[17] R. E. TARJAN, *Depth first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

This research was sponsored in part by National Science Foundation (NSF) grant no. CCR-0122581.