Finding Strongly Connected Components in Parallel using $O(\log^2 n)$ Reachability Queries

Warren Schudy[*]
Brown University
115 Waterman Street
Providence, RI
ws@cs.brown.edu

ABSTRACT

We give a randomized (Las-Vegas) parallel algorithm for computing strongly connected components of a graph with n vertices and m edges. The runtime is dominated by $O(\log^2 n)$ multi-source parallel reachability queries; i.e. $O(\log^2 n)$ calls to a subroutine that computes the union of the descendants of a given set of vertices in a given digraph. Our algorithm also topologically sorts the strongly connected components.

Using Ullman and Yannakakis's [23] techniques for the reachability subroutine gives our algorithm runtime $\tilde{O}(t)$ using mn/t^2 processors for any $(n^2/m)^{1/3} \leq t \leq n$. On sparse graphs, this improves the number of processors needed to compute strongly connected components and topological sort within time $n^{1/3} \leq t \leq n$ from the previously best known $(n/t)^3$ [21] to $(n/t)^2$.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*

General Terms

Algorithms, Theory

Keywords

Graph algorithms, Parallel algorithms, Strongly connected components, Topological sort, Transitive closure bottleneck

1. INTRODUCTION

A core component of scheduling is ordering tasks to respect precedence constraints. For example, the foundation of a house must be completed before the walls can be built. Precedence constraints can be expressed as a directed graph where the vertices are tasks and there is an edge from u to v if u must occur before v. If this graph is acyclic, a topological sort (TS) gives one possible order that the tasks can be completed in. If this graph has cycles, one reasonable thing to do is to compute strongly connected components (SCC), compute a topological sort of those components, and deal with the SCCs containing two or more vertices in a problem-dependant manner. Both strongly connected components and topological sort can be computed in linear time using depth-first-search [22].

Some scheduling problems, e.g. those occurring in some physical simulations [16, 15, 18], are too big to process on a single machine and hence require parallel algorithms. In this work we give a parallel algorithm that computes the strongly connected components of a directed graph and a topological sort. On sparse graphs, it requires only $(n/t)^2$ processors for runtime $\tilde{O}(t)$ as long as $n^{1/3} \leq t \leq n$, where $\tilde{O}$ hides $\log n$ factors. For this parameter range on sparse graphs, the best existing algorithm is Spencer's [21], which requires $(n/t)^3$ processors to achieve a runtime of $\tilde{O}(t)$. Our algorithm works by reducing strongly connected components and topological sort to $O(\log^2 n)$ multi-source reachability queries.

Definition 1. A (multi-source) reachability query has input a directed graph and a set of source vertices. Its output is the set of vertices reachable from any of the sources, i.e. the union of their descendants.

Our main result:

THEOREM 1. *There is a randomized (Las-Vegas) parallel algorithm for the topological sort and strongly connected components problems using $p \geq 1$ processors with runtime $O(\tau \log^2 n)$, where τ is the runtime for a multi-source reachability query executed in parallel on p processors. We assume $\tau \geq \Theta((m+n)/p + \log n)$ (the reachability algorithm must read its input).*

We use Ullman and Yannakakis's [23] reachability algorithm, but give a slightly tighter analysis, yielding:

LEMMA 2. *[23] With high probability, reachability can be computed in time $\tilde{O}(t)$ using mn/t^2 processors as long as $(n^2/m)^{1/3} \leq t \leq n$.*

(Practical aside: $(m+n)/t$ processors suffices for any $1 \leq t \leq m$ if the longest finite shortest path is at most t.) Theorem 1 and Lemma 2 imply we can find strongly connected

[*]Work done at Google Inc., Mountain View CA.

Table 1: Summary of results and prior work. Table shows the number of processors needed to handle an instance with m edges, n vertices in time $\tilde{O}(t)$ where t is an input parameter.

	Problems	**Sparse ($m = \Theta(n)$)**	**Dense ($m = \Theta(n^2)$)**	**General**	**Restrictions**
Coppersmith and Winograd [7]	All	$n^{2.38}/t$	$n^{2.38}/t$	$n^{2.38}/t$	$1 \leq t \leq n^{2.38}$
Coppersmith et al. [8, 6]	SCC	1	1	1	$t = m$
Spencer [21]	All	$(n/t)^3$	$(n/t)^3 + n^2/t$	$(n/t)^3 + m/t$	$1 \leq t \leq m$
Kao et al. [13]	Reach	n^2/t	n^4/t	m^2/t	$t \geq 1$
Ullman et al. [23]	Reach	$(n/t)^2$	n^3/t^2	nm/t^2	$(n^2/m)^{1/3} \leq t \leq n$
This work	SCC, TS	$(n/t)^2$	n^3/t^2	nm/t^2	$(n^2/m)^{1/3} \leq t \leq n$

components and topologically sort using $(n/t)^2$ processors in runtime $\tilde{O}(t)$. Our result is a black-box reduction, so better algorithms for the reachability problem would result in better results for SCC and TS as well.

In the remaining sections, we (2) give additional related work, (3) state our algorithm formally, (4) prove correctness, (5) analyze runtime and (6) conclude. Lemma 2 is proven in Appendix A.

2. RELATED WORK

See Table 1 for a summary of this section.

Breadth-first and depth-first search have many applications in the analysis of directed graphs. Breadth-first search can be used to compute the vertices that are reachable from a given vertex and directed spanning trees. Depth-first search can: solve these problems, determine if a graph is acyclic, topologically sort an acyclic graph and compute strongly connected components (SCCs) [22]. Efforts to parallelize these algorithms have met with mixed success.

Reif [20] shows that if you insist on a particular ordering of the neighboring edges, finding the depth-first-search tree is P-complete and hence unlikely to be in NC. If the search is allowed to explore the children of a vertex in any order, a DFS tree can be found in polylog time with $O(n \cdot n^{2.38})$ processors [1].

Applications of DFS seem easier to solve in parallel than DFS itself. Let *transitive-closure bottleneck problems* [14] refer to breadth-first search and the above applications of BFS and DFS, but not DFS itself. All these transitive-closure bottleneck problems can be solved in polylog time with $n^{2.38}$ processors using matrix-exponentiation techniques to compute the transitive closure of the graph [9, 7]. Spencer [21] gives algorithms for these transitive-closure bottleneck problems that trade-off time and work, with runtime $\tilde{O}(t)$ on $O((n/t)^3 + m/t)$ processors for any $1 \leq t \leq m$.

Breadth-first search and its applications seem easier to parallelize than applications of DFS such as cycle detection. Ullman and Yannakakis [23] give a clever technique for computing reachability and breadth-first search in parallel, with runtime $\tilde{O}(t)$ using $O(mn/t^2)$ processors as long as $(n^2/m)^{1/3} \leq t \leq \sqrt{n}$. As noted in the introduction, we show in Appendix A that (at least for reachability) the upper-bound on t can be relaxed to $t \leq n$. If $t = n$, the number of processors mn/t^2 equals m/t, so the total work done is $\tilde{O}(m)$, which is within logarithmic factors of serial algorithms. We remark that for $1 \leq t \leq n$, if $t \leq n^2/m$, Ullman and Yannakakis's algorithm requires fewer processors than Spencer's, but if $t \geq n^2/m$, Spencer's is more efficient. It follows that Ullman and Yannakakis's algorithm is better for sparse graphs, and Spencer's is better for dense graphs.

If one could compute strongly connected components and topological sort using BFS instead of DFS, one could use Ullman and Yannakakis's better result for that problem on the sparse graphs frequently encountered in applications. Like the present work, Coppersmith, Fleischer, Hendrickson and Pinar [8, 6] give a simple parallel divide and conquer algorithm for computing SCCs using reachability queries. They prove $O(m \log n)$ serial runtime, but their algorithm does not parallelize well in general; for example it has runtime $\Theta(n)$ on the graph without edges. McLendon, Hendrickson, Plimpton and Rauchwerger [16, 15, 18] successfully apply Coppersmith et al's [8, 6] algorithm to an application in scientific computing (discrete ordinates method for radiation transport).

Many special cases of the transitive closure bottleneck problems admit efficient parallel algorithms. There are linear-processor, polylog-time algorithms for finding connected components of *undirected* graphs (see [11] for a survey) and strongly connected components and topological sort of *planar* directed graphs [12, 3, 13]. Kao and Klein [13] also reduce planar reachability to planar topological sort and planar SCC.

Cohen [5] gives a parallel algorithm for estimating the size of the transitive closure of a graph. Like the present work, they use reachability from a prefix of a random permutation of the vertices as part of a divide and conquer step.

Akio, Masahiro and Ryozo [2] claim linear-processor, polylog time algorithms for TS and SCC in a Japanese-language journal.

3. ALGORITHMS

The rest of this paper is devoted to our topological sort of strongly connected components (TS/SCC) algorithm. The SCC problem is sufficient for motivating our algorithm and analysis, so we henceforth only occasionally mention that our algorithm returns the SCCs in topologically sorted order.

Notation: let (V,E) be a directed graph with n vertices and m edges. V, n and m refer either to the entire graph or to a subgraph resulting from a recursive step of our algorithm; the meaning should be clear from context.

Definition 2. Vertex u *reaches* vertex v, denoted $u \rightsquigarrow v$, if there is a directed path from u to v. If u reaches v we say u is an *ancestor* of v. If in addition $u \neq v$, we say u is a *strict ancestor* of v. For a vertex set S and vertices $u, v \in S$, let $u \overset{S}{\rightsquigarrow} v$ denote that there is a path from u to v in the subgraph induced by S.

For exposition, we first present the SinglePivot Algorithm algorithm, which is an simple quicksort-like TS/SCC algo-

Algorithm 1 A quicksort-like TS/SCC algorithm similar to previous work (see text).

```
$SinglePivot(V)$:
    • Choose a random pivot vertex $v$
    • Use reachability queries to compute:
        – $B$ = vertices reached from vertex $v$
        – $C$ = vertices that reach and are reached from $v$.   /* $C \subseteq B$ is an SCC. */
    • In parallel, compute $SinglePivot(V \setminus B)$ and $SinglePivot(B \setminus C)$.
    • Return the SCCs from the recursive calls in the following topological order: $SinglePivot(V \setminus B)$, then $C$, and finally $SinglePivot(B \setminus C)$.
```

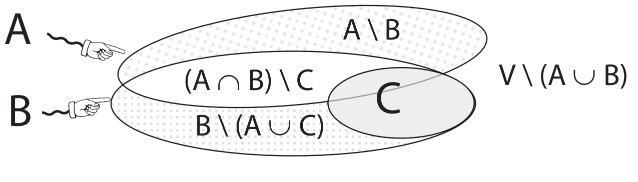

Figure 1: Venn Diagram for MultiPivot Algorithm.

rithm based on the techniques of [21, 8, 6, 4, 10]. All vertices in an SCC have identical ancestor and descendant sets. Therefore given any pivot vertex v, one can divide and conquer, recursing on the vertices reachable from v and the vertices not reachable from v. The SCC containing v is precisely those vertices that are both reachable from v and reach v, so this SCC can be output and removed (this is the base case). See Algorithm 1. Unfortunately the SinglePivot Algorithm and the algorithms of [21, 8, 6, 4, 10] have recursion depth $\Theta(n)$ on a graph with no edges.

To solve this problem, we sample several vertices instead of just one and compute which vertices are reachable from any of the sampled vertices using a multi-source reachability query. We do not know how to choose the sample size a priori, so we simply do a binary search for a sample size that divides the problem evenly. See Algorithm 2 for the definition of the MultiPivot Algorithm and Figure 2 for an illustration. Figure 1 shows the relationship between the sets of the MultiPivot Algorithm. The subtle part of the analysis is showing that the set $B \setminus (A \cup C)$ (as defined in the MultiPivot Algorithm) is not too big.

We prove Theorem 1 via two lemmas. The straightforward proof of the following lemma is in Section 4.

Lemma 3 (Correctness). *The MultiPivot Algorithm correctly computes the SCCs and a topological sort thereof.*

The interesting proof of the following lemma is in Section 5.

Lemma 4 (Runtime). *For any $\gamma > 1$, with probability at least $1 - n^{1-\gamma}$, the MultiPivot Algorithm takes time $O(\gamma\tau(\log n)^2)$ on a CRCW PRAM with p processors, where $\tau \geq \Theta((m+n)/p + \log n)$ is the time required for a reachability query.*

Algorithm 2 MultiPivot Algorithm.

```
MultiPivot(V):
    • Permute the vertices in $V$ randomly and assign corresponding indices $1, 2, \ldots n$.
    • Do a binary search for the smallest index $s$ such that the first $s$ vertices together reach vertices that induce a subgraph with at least $(m+n)/2$ vertices plus edges.   /* Uses $O(\log n)$ reachability queries. */
    • Use reachability queries to compute:
        – $A$ = vertices reached from vertex set $\{1, \ldots s-1\}$
        – $B$ = vertices reached from vertex $s$
        – $C$ = vertices that reach and are reached from $s$.   /* $C \subseteq B$ is an SCC. */
    • In parallel, recursively compute the strongly connected components of $V \setminus (A \cup B)$, $A \setminus B$, $B \setminus (A \cup C)$, and $(A \cap B) \setminus C$.
    • Return the SCCs in the following topological order: the SCCs from the recursive call $MultiPivot(V \setminus (A \cup B))$, then $MultiPivot(A \setminus B)$, then the SCC $C$, then $MultiPivot(B \setminus (A \cup C))$, and finally $MultiPivot((A \cap B) \setminus C)$.
```

4. ANALYSIS: CORRECTNESS

Note: this section is straightforward.

Since $C \subseteq B$ by definition of C, we have the following claim (see also Figure 1):

Claim 5. *The sets $\{V \setminus (A \cup B), A \setminus B, C, B \setminus (A \cup C), (A \cap B) \setminus C\}$ form a partition of V.*

Lemma 6. *Every set the algorithm outputs (claimed to be an SCC) satisfies the property that every vertex in it is reachable from every other.*

Proof. Every vertex in C can reach and be reached from s by definition, so therefore any vertex in C can reach any other vertex, via s. □

Lemma 7. *The topological sort has no edges going from a set to another set before it in the order.*

Proof. Recall that the sets are output in the order $V \setminus (A \cup B)$, $A \setminus B$, C, $B \setminus (A \cup C)$, $(A \cap B) \setminus C$. We show in turn that each of these sets has no edges to it from sets to the right of it in the order.

- There are no edges into $V \setminus (A \cup B)$ from the other sets because A and B have no edges leaving them.
- There are no edges into $A \setminus B$ from C, $B \setminus (A \cup C)$, $(A \cap B) \setminus C \subseteq B$ because B has no edges leaving it.
- There are no edges into C from $B \setminus (A \cup C)$, $(A \cap B) \setminus C \subseteq B \setminus C$ because any vertex in B that can reach a vertex in C can also reach s and is therefore in C.
- There are no edges into $B \setminus (A \cup C)$ from $(A \cap B) \setminus C$ because there are no edges leaving A.

□

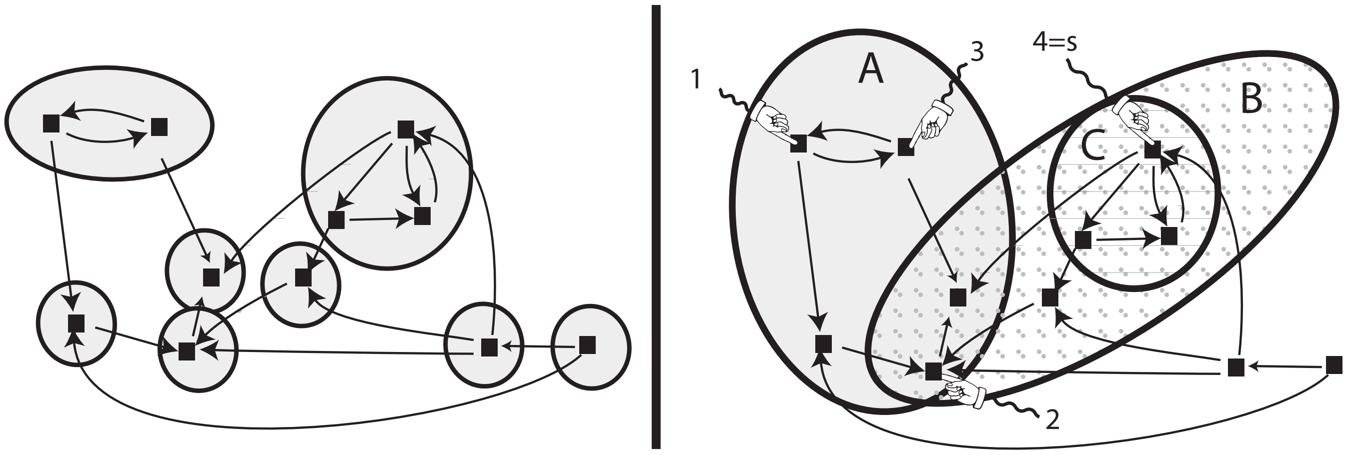

Figure 2: Illustration of MultiPivot Algorithm (right) on a sample graph with SCCs shown (left). The first 3 vertices reach only 6 edges out of 18, but the first four vertices reach 13 out of 18 edges and hence $s = 4$.

PROOF OF LEMMA 3. Claim 5 implies that our output includes every vertex exactly once.

Since vertex s is always in C, each recursive call has at least 1 fewer vertices than its parent so the algorithm terminates.

Lemma 6 shows that we never output an SCC that is too big. Lemma 7 implies that we never output an SCC that is too small and that our topological sort is correct. □

5. ANALYSIS: RUNTIME

Each SCC call uses at most $O(1) + \lg n$ reachability queries for the binary search and then computing the sets A, B and C, which takes time $O(\tau \log n)$. The random permutations can be computed in time $o((1 + \frac{n}{p}) \log n) \leq \Theta((1 + \frac{m+n}{p}) \log n)$ using [19]. Each time the reachability subroutine is called, the number of reached edges must be counted and the source set for the reachability query adjusted. This can be done in time $O(\frac{m+n}{p} + \log n)$ using standard techniques such as parallel prefix computations, for a total time of $O((\frac{m+n}{p} + \log n) \log n) = O(\tau \log n)$ by the assumed lower-bound $\tau = \Omega((m+n)/p + \log n)$. Therefore, each recursive call takes time $O(\tau \log n)$.

All of the required tasks only get easier as the problem is subdivided so the total runtime is bounded by the runtime of the root call multiplied by the depth of the recursion tree.[1] Therefore it remains to show that the depth of the recursion tree is $O(\log n)$ with high probability.

Definition 3. Let $\mu(S)$ denote the number of edges plus vertices in the subgraph induced by vertex set S. Note that $\mu(V) = m + n$.

[1]If this hand-waving bothers you, consider a iterative variant which keeps a list of subgraphs to process. Initially the only subgraph is the whole graph, which is then replaced by several subgraphs as the algorithm progresses. Reachability queries for all the subgraphs can be performed simultaneously using one reachability query: simply remove edges between the subgraphs, and let the overall source set be the union of the source sets of each subgraph.

As in the analysis of quicksort, we show that the problem is divided more or less evenly. Some progress is easy to show:

LEMMA 8. *All vertex sets S in $\{V \setminus (A \cup B),\ A \setminus B,\ (A \cap B) \setminus C\}$ satisfy $\mu(S) \leq \mu(V)/2$.*

PROOF. By the choice of s we know $\mu(A) < (m+n)/2$ and $\mu(A \cup B) \geq (m+n)/2$, which implies $\mu((A \cap B) \setminus C)$, $\mu(A \setminus B) \leq \mu(A) < (m+n)/2$ and $\mu(V \setminus (A \cup B)) \leq (m+n)/2$. □

Definition 4. Define $T(S)$ for vertex set S as follows:

$$T(S) \equiv \{\, (u,v) \in S \times S \mid u \neq v \text{ and } u \overset{S}{\rightsquigarrow} v \,\}$$

The following Lemma, inspired by Spencer's Lemma 5.4 [21], is the core of our analysis:

LEMMA 9. *With probability at least 1/3, $|T(B \setminus (A \cup C))| \leq \frac{3}{4}|T(V)|$.*

We first prove Lemma 9. Then we use Lemmas 8 and 9 and the potential function $\max(|T(V)|, 3/4) \cdot \mu(V)$ to prove Lemma 4.

5.1 Proof of Lemma 9

Definition 5. The *groundbreaker* of a vertex v, denoted $g(v)$, is the ancestor of v with minimum index in the random permutation.

The groundbreaker $g(v)$ of a fixed vertex v is a random variable because the permutation of the vertices is picked randomly.

Definition 6. Let X_v be the set of strict ancestors of v that have the same groundbreaker as v but are not in the same SCC as $g(v)$. Precisely:

$$\begin{aligned} X_v \;=\; \{ & w \neq v : w \rightsquigarrow v \text{ and } g(w) = g(v) \\ & \text{and } w,\ g(v) \text{ are in different SCCs} \} \end{aligned}$$

Let the number of strict ancestors of v be α_v.

LEMMA 10. *For any $v \in V$, we have $\mathbf{E}\left[|X_v|\right] \leq \alpha_v/2$ and $0 \leq |X_v| \leq \alpha_v$.*

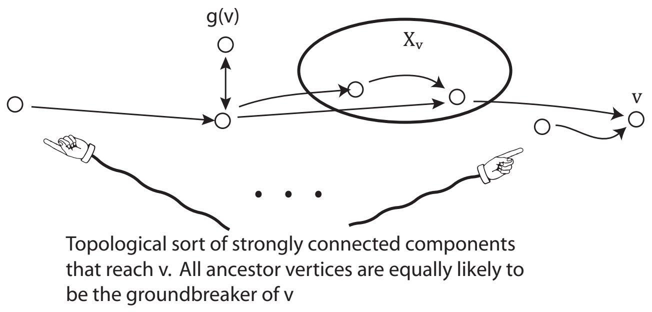
g(v)
X_v
v
Topological sort of strongly connected components that reach v. All ancestor vertices are equally likely to be the groundbreaker of v

Figure 3: Illustration of Lemma 10.

PROOF. See Figure 3. Consider a topological sort of the ancestors of v (with vertices in same SCC adjacent in the sort). The groundbreaker of v is the first ancestor chosen, so all the $\alpha_v + 1$ ancestors (including v itself) have equal probability of groundbreaking. If an ancestor u of v has the same groundbreaker as v but is in a different SCC from $g(v)$, then u must be after $g(v)$ in the topological sort. The expected number of vertices other than v strictly after $g(v)$ in the topological sort is:

$$\frac{1}{\alpha_v+1}\left(0+\sum_{i=1}^{\alpha_v}(i-1)\right) = \frac{1}{\alpha_v+1}\cdot\frac{(\alpha_v-1)(\alpha_v-1+1)}{2} \leq \alpha_v/2.$$

□

Definition 7.

$$Y = \{\,(u,v) \in V \times V \mid u \in X_v\,\}.$$

COROLLARY 11. *We have* $\Pr\left(|Y| \leq \frac{3}{4}|T(V)|\right) \geq 1/3$.

PROOF. Since $\sum_{v\in V}\alpha_v = |T(V)|$, $\mathbf{E}\left[|Y|\right] = \mathbf{E}\left[\sum_v |X_v|\right] \leq |T(V)|/2$ and $0 \leq |Y| \leq |T(V)|$. We use a Markovian argument: the probability of Y being less than $\frac{3}{4}|T(V)|$ subject to these constraints is minimized when $|Y| = \frac{3}{4}|T(V)| + 1$ with probability $2/3$ and $|Y| = 0$ with probability $1/3$. □

To finish the proof of Lemma 9 we argue that $T(B \setminus (A \cup C)) \subseteq Y$, and hence $|T(B\setminus(A\cup C))| \leq \frac{3}{4}|T(V)|$ with probability at least $1/3$ by Corollary 11. To see this, consider some $(w,v) \in T(B \setminus (A \cup C)$. Clearly both w and v are reachable from s but not from $1,\ldots s-1$, so $g(v) = g(w) = s$. The SCC of s is C, so u and v are not in the groundbreaker's SCC by definition. Therefore $w \in X_v$ so $(w,v) \in Y$.

5.2 Putting the pieces together

In this section we use Lemmas 8 and 9 to prove that the recursion tree has depth $O(\log n)$ with high probability, finishing the proof of Lemma 4. It is easy to see that the expected work is small, but for a parallel algorithm we need to show that the maximum depth, not just the average depth, is small. We show every vertex has probability at most $n^{-\gamma}$ of exceeding $O(\gamma \log n)$ depth for any $\gamma > 0$, implying by a union bound that all vertices finish within that depth with probability at least $1 - n^{1-\gamma}$.

We use $\max(T(V), 3/4) \cdot \mu(V)$ as our potential function. Declare a call to SCC(V) to be *successful* if all of the nonempty sets V' it recurses on satisfy $\max(T(V'), 3/4)\cdot\mu(V') \leq \frac{3}{4}\max(T(V), 3/4)\cdot\mu(V)$. If $T(V) \geq 1$, we have the probability of success is at least $1/3$ by Lemmas 8 and 9. If $T(V) = 0$, each vertex is isolated so $B\setminus(A\cup C) = \{\}$, hence by Lemma 8 the probability of success in this case is 1. Clearly we have $\min(3/4, T(V)) \cdot m \leq n(n-1)n(n+1) \leq n^4$ (assuming $|V| \geq 2$). By definition $\mu(V') \geq |V'|$ so $3/4\cdot\mu(V') \geq 3/4$ for any non-empty V'. It therefore suffices to show that with probability $1 - n^{-\gamma}$ we have at least $4\log(4n/3)/\log(4/3) \leq 14 \log n$ successes (for n sufficiently large) before the depth of a fixed vertex exceeds $63\gamma \log n$, where log denotes the natural, base-e logarithm. This probability is bounded by the probability that $63\gamma \log n$ independent coin flips, each with probability of success $1/3$, has less than $14 \log n$ successes.

Chernoff bound for any $0 < \delta < 1$ [17]:

$$\Pr\left(Z < (1-\delta)\mu\right) \leq e^{-\mu\delta^2/2}$$

Letting Z be the number of successes, $\mu = 21\gamma\log(n)$, and $\delta = 1/3$, yields

$$\Pr\left(Z < 14\gamma\log(n)\right) \leq n^{-\frac{21}{18}\gamma} \leq n^{-\gamma}$$

Clearly for $\gamma > 1$ we have $14\gamma\log(n) > 14\log n$, so this is sufficient.

6. FUTURE WORK

The most important related open problem is how to answer reachability queries efficiently. For example, consider a constant-degree (sparse) directed graph with longest finite shortest path $\Theta(n)$ and one processor per edge ($m = p = \Theta(n)$). Is it possible to answer a reachability query on this graph in $\tilde{o}(\sqrt{n})$ time?

7. ACKNOWLEDGEMENTS

I would like to thank D. Sivakumar for suggesting the parallel strongly connected components problem, Glencora Borradaile, Maurice Herlihy and Claire Mathieu for advice that dramatically improved the presentation, and Google Inc. for free food.

8. REFERENCES

[1] A. Aggarwal, R. J. Anderson, and M.-Y. Kao. Parallel depth-first search in general directed graphs. *SIAM J. Comput.*, 19(2):397–409, 1990.

[2] T. Akio, M. Masahiro, and N. Ryozo. Parallel topological sorting algorithm. *Transactions of Information Processing Society of Japan*, 45(4):1102–1111, 2004. *SCC algorithm in TIPSJ '99.*

[3] D. Bader. A practical parallel algorithm for cycle detection in partitioned digraphs. Technical Report AHPCC-TR-99-013, Electrical & Computer Eng. Dept., Univ. New Mexico, Albuquerque, NM, 1999.

[4] R. Bloem, H. N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in n log n symbolic steps. *Form. Methods Syst. Des.*, 28(1):37–56, 2006.

[5] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997.

[6] D. Coppersmith, L. Fleischer, B. Hendrickson, and A. Pinar. A divide-and-conquer algorithm for identifying strongly connected components. Technical Report RC23744, IBM Research, 2005.

[7] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *STOC '87: Proceedings of the nineteenth annual ACM Symposium on Theory of Computing*, pages 1–6, New York, NY, USA, 1987. ACM Press.

[8] L. Fleischer, B. Hendrickson, and A. Pinar. On identifying strongly connected components in parallel. In *IPDPS '00: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, pages 505–511, London, UK, 2000. Springer-Verlag.

[9] H. Gazit and G. L. Miller. An improved parallel algorithm that computes the bfs numbering of a directed graph. *Inf. Process. Lett.*, 28(2):61–65, 1988.

[10] R. Gentilini, C. Piazza, and A. Policriti. Computing strongly connected components in a linear number of symbolic steps. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[11] J. Greiner. A comparison of parallel algorithms for connected components. In *SPAA '94: Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, pages 16–25, New York, NY, USA, 1994. ACM Press.

[12] M.-Y. Kao. Linear-processor nc algorithms for planar directed graphs i: Strongly connected components. *SIAM J. Comput.*, 22:431–459, 1993.

[13] M.-Y. Kao and P. N. Klein. Towards overcoming the transitive-closure bottleneck: efficient parallel algorithms for planar digraphs. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 181–192, New York, NY, USA, 1990. ACM Press.

[14] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. *Handbook of theoretical computer science (vol. A): algorithms and complexity*, pages 869–941, 1990.

[15] W. McLendon, III, B. Hendrickson, S. Plimpton, and L. Rauchwerger. Finding strongly connected components in parallel in particle transport sweeps. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 328–329, 2001.

[16] W. McLendon, III, B. Hendrickson, S. J. Plimpton, and L. Rauchwerger. Finding strongly connected components in distributed graphs. *J. Parallel Distrib. Comput.*, 65(8):901–910, 2005.

[17] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, chapter 4.2, page 66. Cambridge University Press, 2005.

[18] S. Plimpton, B. Hendrickson, S. Burns, and W. McLendon, III. Parallel algorithms for radiation transport on unstructured grids. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 25, Washington, DC, USA, 2000. IEEE Computer Society.

[19] S. Rajasekaran and J. H. Reif. Optimal and sublogarithmic time randomized parallel sorting algorithms. *SIAM J. Comput.*, 18(3):594–607, 1989.

[20] J. H. Reif. Depth-first search is inherently sequential. *Information Processing Letters*, 20(5):229–234, 1985.

[21] T. H. Spencer. Time-work tradeoffs for parallel algorithms. *J. ACM*, 44(5):742–778, 1997.

[22] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

[23] J. D. Ullman and M. Yannakakis. High probability parallel transitive-closure algorithms. *SIAM J. Comput.*, 20(1):100–125, 1991.

APPENDIX

A. REACHABILITY

We only change the analysis of Ullman and Yannakakis [23] slightly, so we only sketch our variant of their work.

Ullman and Yannakakis [23] proved that $\tilde{O}(t)$ time is achievable with $p = nm/t^2$ processors as long as $(n^2/m)^{1/3} \leq t \leq \sqrt{n}$, so to prove Lemma 2 it suffices to analyze the case $\sqrt{n} \leq t \leq n$. We assume that there is only one source vertex by creating a new super-source vertex u.

Let *BasicBFS(G, S, d)* refer to the naive parallel breadth-first-search algorithm on graph G with depth limit d, $|S|$ independent source vertices. This is Ullman and Yannakakis's Basis Rule B2.

Their analysis of the work required in Basis Rule B2 (page 109 of [23]) conservatively assumes that the number of processors is m and hence the work required is $\tilde{O}(m|S| + md)$. We use the tighter bound $\tilde{O}(m|S| + pd)$. A nice consequence of this change is that the algorithm is shown to be efficient when $t = n$.

If a vertex v is reachable from u, there must be a path P from u to v. Consider taking a random sample of vertices S by sampling each vertex with probability $\tilde{O}(1/t)$. One can show that there probably won't be a string of unsampled vertices in the path of length more than t. The key observation of Ullman and Yannakakis [23] is you can then decompose the path into at most $\tilde{O}(n/t)$ hops between sampled vertices, with each hop being a path of length at most t. Hence we have Algorithm 3.

The time spent looking for shortcuts among the distinguished nodes is $\tilde{O}((n/t)(m/p) + t) = \tilde{O}(t^2/t + t) = \tilde{O}(t)$. The number of shortcuts added is at most $(n/t)^2 \leq n^2/(\sqrt{n})^2 = \tilde{O}(m)$ (using $t \geq \sqrt{n}$ by assumption), so G' is not much bigger than G. The time spent searching for paths in G' is $\tilde{O}(m/p + n/t + t) = \tilde{O}(t)$ (using $t \leq n$).

Algorithm 3 Our variant of the reachability algorithm of Ullman and Yannakakis

```
Reachability($u$):
  • Let $t$ be the time budget and $p = nm/t^2$ be the number of processors
  • Take a random sample of *distinguished nodes* of size $\tilde{O}(n/t)$. Let $D$ be the distinguished nodes.
  • Run BasicBFS(G, D, t), and create a new graph $G'$ with extra "shortcut" edges between distinguished nodes that have paths of length at most $t$ in $G$.
  • Run BasicBFS($G'$, $\{u\}$, $|D|+2t$) to determine the nodes reachable from $u$.
  • Check if there are no edges from reachable vertices to unreachable ones. If there are, we were unlucky, so try again with a new set of distinguished nodes.
```