PROJECT FOR CS388P: PARALLEL ALGORITHMS (FALL 2016)

Prioritizing Processor Efficiency Over Time Efficiency

Matthew Hudson Walter Xia

October 16, 2016

Abstract

Our paper is *Time-Work Tradeoffs for Parallel Algorithms, Spencer[3]*. In this paper, Spencer focuses on several open parallel problems that are affected by the transitive closure bottleneck. For each of the below problems, Spencer describes an algorithm that comes closer to work efficiency than existing approaches that utilize transitive closure. However, this increased work efficiency comes at the cost of an asymptotic increase in the span of each algorithm relative to the time optimal solution.

Problems

- Solving Triangular Systems of Linear Equations
- Topological Sort
- Breadth-First Search in Directed Graphs
- Strongly Connected Components
- Single Source Shortest Path

Rather than using a matrix based approach, Spencer utilizes a new data structure called *nearby lists* that serves as the work horse for the subsequent algorithms.

1 INTRODUCTION

Spencer[3] focuses on problems for which we known of no work-time efficient parallel algorithms that provide a solution. That is, there are no known parallel algorithms that are able to solve any of the following problems within a logarithmic factor of the fastest sequential algorithm's work, where work is the product of the running time and the processor count. For the following problems, which were presented in *Karp-Ramachandran[1], Spencer[3]* develops parallel algorithms with a better work efficiency than transitive closure based methods:

- Directed Spanning Tree
- Breadth-First Search

- Topological Sort
- Cycle Detection for Directed Graphs
- Strong Connected Components
- Single Source Shortest Paths with Non-Negative Edges

Spencer's motivation for prioritizing work efficiency over time efficiency stems from the fact that existing parallel transitive closure algorithms for these problems perform significantly worse than their sequential counterparts when run on only a single processor. In an effort to reduce this problem, Spencer's algorithms exhibit a time-work trade-off, that is, the longer they are allowed to run, the less overall work they perform. This phenomenon exists due of the fact that certain speculative computations can be eliminated as time progresses, thus reducing overall work. The first full algorithm which Spencer presents solves topological sort, while the second algorithm solves breadth first search. It was mentioned in passing that the breadth first search algorithm can solve directed spanning trees and the topological sort algorithm can solve cycle detection for directed graphs. Furthermore, Spencer expands upon the breadth first search algorithm by utilizing it to find strongly connected components and extending his breadth first search methods to support non-negative edge weights so as to solve the single source shortest paths problem.

/* Note: We may or may not need these definitions here. */
We next reproduce three definitions given in *Spencer[3]*:

Definition. A vertex v *reaches* a vertex u iff there is a path from v to u. Conversely, u is *reachable* from v iff there is path from v to u.

Definition. Let $G = (V, E)$ be a graph and let $R \subseteq V$. $G[R] = (R, E')$, where $E' \subseteq R \times R$.

Definition. Two vertices u, v are *identified* by replacing them with a single vertex uv. The outgoing edges of both u and v becomes the outgoing edges of uv and the incoming edges of both u and v become the incoming edges of uv.

2 Results and Techniques of Assigned paper

Let $G = (V, E)$, $m = |E|$, and $n = |V|$. The major contributions of *Spencer[3]* are as follows:

- Algorithms that solves topological sort and breadth first search in $O(\frac{n}{\rho}\log^2\rho)$ time with ρ^3 processors on an EREW PRAM such that $\sqrt{\frac{m}{n}} \leq \rho \leq n$.
- A randomized algorithm that solves strongly connected components in a directed graph in $O(\frac{n}{\rho}\log^2\rho\log n)$ expected time and $O(n\rho^2\log^2 p\log n)$ expected work such that $\sqrt{\frac{m}{n}} \leq \rho \leq n$.
- An algorithm that solves single source shortest paths in $O(\frac{n}{\rho}\log n\log(L\rho))$ time and $O(n\rho^2(\log n + \log(L\rho)\log\rho) + m(\log n + \log(L\rho)))$ work such that $\log(L\rho) <= \rho <= n$, where L is the length of the longest edge.

2.1 TECHNIQUES

Definition. The *nearby list*, $NL(v)$, of a vertex v consists of ordered pairs $(u, d(v,u))$ where u is a vertex reachable from v and $d(v,u)$ is the distance from v to u.

Definition. A vertex v is *terminal* if $r(v)$, the nearby radius of v, is infinite and $NL(v)$ contains all reachable vertices from v. A vertex v is *totally terminal* if v is terminal and all vertices in $NL(v)$ is terminal.

In solving topological sort, *Spencer[3]* actually solves the reverse topological sort which can be easily converted to a topological sort by replacing $w(v)$ with $n+1-w(v)$, where $w(v)$ is the label given by the reverse topological sort. *Karp-Ramachandran[1]* presented an algorithm that can compute this topological sort in $O(\log^2 n)$ time and $O(n^3 \log n)$ work by computing the transitive closure, G^*, of G. *Spencer[3]* introduces an algorithm, *Rtopo*, that performs less work by computing parts of G^* instead of the whole thing. This is achieved by the nearby list data structure, whose size is controlled by the parameter ρ. The smallest $w(v)$ is assigned to the totally terminal vertices, and the nearby lists are updated to expose newly formed totally terminal vertices. The details of which we omit here.

Definition. A vertex v is *known* iff the distance from s to v is at most R_k, the known radius.

In solving breadth first search, *Spencer[3]* employs a very similar algorithm to *Rotpo* called *pbfs*. Again, there exists a parallel algorithm that can solve breadth first search in $O(\log^2 n)$ time and $O(n^3 \log n)$ work by calculating the transitive closure G^*. *pbfs* reduced the work by deleting vertices from G_u, the unknown graph consisting of unknown vertices, once it has been known. All the techniques developed for *Rtopo* are applicable to *bpfs*.

Definition. A Las Vegas algorithm is a probabilistic algorithm which can detect when it has made a mistake and try again, thereby guaranteeing a correct solution.

Definition. A vertex v is *big* iff it reaches more than $\frac{n}{8}$ vertices, otherwise the vertex is small.

In solving strongly connected components, Spencer again utilizes pbfs. The algorithm utilizes divide and conquer in order to separate subsets of SCC's and subgraphs of nodes which have not yet been placed in a SCC. He first applies a procedure called "split" which removes any fully explored SCC's, where utilizes uniform randomness to select a node v in one of the subgraphs in order to compute its SCC and then separates the nodes which were reachable from v but not in its SCC by adding these nodes to a new subgraph for further processing. At this point, Spencer's algorithm utilizes the divide and conquer technique by recursively calling itself on each of the induced subgraphs until all of the SCC's have been found. Although this solution is correct with high probability, before termination, Spencer utilizes *Ullman's[**?**]* method of verifying the SCC's and rerunning the algorithm if the the verification fails in order to transform his algorithm into a Las Vegas Algorithm. Since the true running time of the algorithm is probabilistic, Spencer utilizes a distinction between "large" and "small" vertices along with Chebyshev's Inequality to lower bound the probability of success by a constant factor, thereby allowing him to treat the necessary number of repeated runs as a constant and absorbing this factor into the big O.With these bounds in place,the algorithm has an expected $O(\frac{n}{\rho} * log^2(\rho) * log(n))$ running time and $O(n\rho^2 \log^2 \rho \log n)$ expected work.

In solving Single Source Shortest Paths, Spencer employs a method very similar to pbfs, but adds additional complexity in order to account for the edge weights.

3 Significant Related Results

The three most centrally related papers to *Spencer[3]* are: *Karp-Ramachandran[1]*, *Ullman-Yannakakis[4]*, and *Paul[2]*.

Karp-Ramachandran[1] is a survey of the accumulated theory of parallel algorithms up to the 1990s. This survey focused on the PRAM as its main computational model and starts off with an overview of efficient parallel algorithms. After establishing the fundamentals, the survey explores how the different variants PRAM models handle concurrent reads and write, in addition provided a brief introduction to other parallel models different from the PRAM. Finally, the survey discussed the class of problem known as NC, problems that have parallel solutions in polylog time and perform polynomial work. It is actually here in which *Spencer[3]* makes use of *Karp-Ramachandran[1]*, where the survey listed the aforementioned six graph problems that have no known efficient solution due to the Transitive Closure Bottleneck. We believe that without *Karp-Ramachandran[1]*, there would be no *Spencer[3]*, since his entire paper is stemmed from this part of the survey. There is also the fact that *Spencer[3]'s Rtopo* algorithm depends on an efficient simulation of CREW on EREW to maintain the necessary data structures for its nearby lists.

Ullman-Yannakakis[4] introduces a high-probability parallel algorithm that finds the transitive closure from a single source in $\tilde{O}(n^{\epsilon})$ time with $\tilde{O}(mn^{1-2\epsilon})$ processors such that $m \geq n^{2-3\epsilon}$. Their algorithm also finds the transitive closure for all pairs in $\tilde{O}(n^{\epsilon})$ time with $\tilde{O}(mn^{1-\epsilon})$ processors such that $m \geq n^{2-2\epsilon}$. Both require $0 \leq \epsilon \leq \frac{1}{2}$. The probability that their algorithm fails to find a path is at most $2^{-c\alpha}$, where α is a positive constant and c is some time multiplier. Their algorithm is the first in which time is sub-linear and work is less than $M(n)$. Our main interest in *Ullman-Yannakakis[4]* is the fact that they have already established an algorithm exhibiting the time-work trade off that solves breadth first search. Their algorithm is high-probability, and we were interested in seeing a different approach to the same problem we are facing in *Spencer[3]*. In addition, their was extended into the context of single source shortest paths and strongly connected components, highlighting the fact that the true novelty in *Spencer[3]* is it's introduction of nearby lists.

Paul[2] introduces parallel techniques that search, insert, and delete from 2-3 trees in $O(\log n + \log k)$ time on the EREW, where n is the number of leaves and k is the number of searches, inserts or deletes we want to perform in parallel. He seemed to be the first to design a fast tree algorithm. This results presented by *Paul[2]* were used by *Spencer[3]* at several key areas in his algorithms. In particular, *Rtopo* needed a variation of the search algorithm to find all the totally terminal vertices to be deleted and *SSSP* required a parallel extension of 2-3 trees to search, insert, and delete many items in parallel. We decided to unravel some of the details presented in Paul that Spencer decided to abstract away from his paper so that we may have a better understanding of his algorithms.

4 Presentation Topics

Here each team member should cite the paper from which they will present technical results in their presentation, and should indicate the exact theorem(s)/lemma(s) whose proof(s) they will present.

5 Further Research Directions

Give an overview of the current open problems and future research directions related to the work in the assigned paper.

References

[1] R. M. Karp and V. Ramachandran. A survey of parallel algorithm for shared-memory machines. *Handbook in Theoretical Computer Science*, pages 35–57, 1990.

[2] H. Paul, U. Vishkin, and H. Wagener. Parallel dictionaries in 2-3 trees. pages 597–609, 1983.

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

[4] J. D. Ullman and M. Yannakakis. High-probablility parallel transitive closure algorithms. *SIAM Journal on Computing*, 20(1):100–125, Feb. 1991.