Brief Announcement: A Tight Distributed Algorithm for All Pairs Shortest Paths and Applications[*]

Qiang-Sheng Hua
Services Computing Technology and System Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, China

Haoqiang Fan
The Institute for Interdisciplinary Information Sciences, Tsinghua University, China

Lixiang Qian, Ming Ai, Yangyang Li, Xuanhua Shi, Hai Jin
Services Computing Technology and System Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, China

ABSTRACT

Given an unweighted and undirected graph, this paper aims to give a tight distributed algorithm for computing the all pairs shortest paths (APSP) under synchronous communications and the $\mathcal{CONGEST}(\mathcal{B})$ model, where each node can only transfer B bits of information along each incident edge in a round. The best previous results for distributively computing APSP need $O(N + D)$ time where N is the number of nodes and D is the diameter [1, 2]. However, there is still a B factor gap from the lower bound $\Omega(N/B + D)$ [1]. In order to close this gap, we propose a multiplexing technique to push the parallelization of distributed BFS tree constructions to the limit such that we can solve APSP in $O(N/B+D)$ time which meets the lower bound. This result also implies a $\Theta(N/B + D)$ time distributed algorithm for diameter. In addition, we extend our distributed algorithm to compute girth which is the length of the shortest cycle and clustering coefficient (CC) which is related to counting the number of triangles incident to each node. The time complexities for computing these two graph properties are also $O(N/B + D)$.

1. INTRODUCTION

Computing all pairs shortest paths (APSP) is a fundamental problem in computer science. For unweighted graphs, APSP can be solved by fast matrix multiplication [3] and the time complexity is $O(N^{\omega} \log N)$ where N is the number of nodes and ω is the exponent of the fast matrix multiplication algorithm[1]. For weighted graphs, the up-to-date fastest algorithm [5] can solve APSP in time $O(N^3/2^{\Omega(\sqrt{\log N})})$. Whether we can design faster APSP algorithms than the above time complexities is an open problem.

However, there is another story when it comes to distributed algorithms. In the distributed algorithms, we consider synchronous communications and each node can only send a bounded size message to each neighbor in a round. Typically, if we restrict each node can only send B bits of information, it's called the $\mathcal{CONGEST}(\mathcal{B})$ model. If we set $B = O(\log N)$, this is the extensively used $\mathcal{CONGEST}$ model used in the distributed computing community [6].

For an undirected and unweighted graph $G = (V, E)$, where $|V| = N$ and $|E| = M$, computing APSP boils down to computing N Breadth-First-Search (BFS) trees. Distributively computing one BFS tree takes $O(D)$ time where D is the graph diameter. So a naive way to distributively computing APSP takes $O(ND)$ time by sequentially computing a distributed BFS tree rooted for each node. Surprisingly, recent elegant results [1, 2] show that APSP can be done in $O(N+D)$ time by scheduling N parallel distributed BFS tree constructions under the $\mathcal{CONGEST}$ model. This result is nearly optimal since the lower bound is $\Omega(N/B + D)$ [1]. However, there's still a B factor gap from the lower bound. A natural question is: Can we design a distributed algorithm for APSP that matches the lower bound under the $\mathcal{CONGEST}(\mathcal{B})$ model? In this paper, we give an affirmative answer to this question.

The basic idea of our distributed algorithm is that we utilize the multiplexing technique to take full advantage of the bandwidth B. Our algorithm divides the bandwidth into multiple channels and divides the nodes into several groups. The algorithm then performs parallel distributed Breadth-First-Search (BFS) processes for the nodes in the corresponding groups on each channel. In order to avoid message transmission collisions on the same channel (i.e., to satisfy the $\mathcal{CONGEST}(\mathcal{B})$ model), we will schedule the starting time of BFS processes in each channel. Since the distributed BFS processes don't interfere on different channels, there will be another dimension of freedom to increase the parallelism of the distributed BFS processes. Therefore,

[1]The state-of-the-art fast matrix multiplication shows this exponent is around 2.372864 [4].

[*]This paper is partly supported by the NSFC under grant No. 61572216 and 61433019, International Science & Technology Cooperation Program of China under grant No. 2015DFE12860, National 863 Hi-Tech Research and Development Program under grant 2014AA01A301.

SPAA '16 July 11-13, 2016, Pacific Grove, CA, USA
© 2016 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4210-0/16/07.
DOI: http://dx.doi.org/10.1145/2935764.2935812

we give a tight distributed algorithm for APSP that meets the lower bound. This algorithm also implies a tight distributed algorithm for diameter since the lower bound for distributively computing diameter is also $\Omega(N/B + D)$ [7]. In addition, our algorithm can be extended to distributively compute the girth and the clustering coefficient (CC) whose time complexities are also $O(N/B + D)$.

2. OUR RESULTS

The main results of this paper are summarized in Table 1. In this paper we mainly focus on distributed algorithms for exactly computing APSP. Furthermore, we can extend our distributed APSP algorithm to compute diameter, girth and clustering coefficient. Our contributions are as follows:

1. We present a distributed algorithm for computing APSP in time $O(N/B + D)$. This improves previous upper bound of $O(N+D)$ in [1, 2]. Our distributed algorithm matches the lower bound $\Omega(N/B + D)$ of computing APSP in [1].

2. We introduce three applications of our distributed APSP algorithm: diameter, girth and clustering coefficient. All of them can be computed in time $O(D + N/B)$.

Table 1: Previous works and our results

Task	Precision	Previous Works	This Work
APSP	exact	$O(N)$[1][2] *	$\Theta(D + N/B)$
Girth	exact	$O(N)$[1][2] *	$O(D + N/B)$
CC	exact	$O(N)^{*\dagger}$	$O(D + N/B)$
Diameter	exact	$O(N)$[1][2] *	$\Theta(D + N/B)$

* All these works are based on $\mathcal{CONGEST}$ model. Our work is under the $\mathcal{CONGEST(B)}$ model. Note that if $B = O(\log N)$, $\mathcal{CONGEST(B)}$ model is the same as the $\mathcal{CONGEST}$ model.
† We are unaware of any previous work for computing this graph property under the $\mathcal{CONGEST}$ model. But we can derive this result based on the $O(N)$ time distributed APSP algorithm [1].

3. NAME-INDEPENDENT SHORTEST PATHS

In distributed APSP, $O(NM)$ messages have to be exchanged to construct N BFS trees. Following the distributed BFS approach, we cannot avoid a message complexity of $O(NM)$. But in one time step, at most $O(BM)$ bits can flow over the edges. So if the message size is B bits, the total $O(NBM)$ bits need to be exchanged. Thus, the time complexity will be $O(D + N)$. In order to reduce the time complexity to $O(D + N/B)$ and to meet the $\Omega(D + N/B)$ lower bound in [1], the message size can only be $O(1)$ bits such that we only need to exchange $O(NM)$ bits in the graph. In this case, the time complexity will be $O(D + N/B)$. However, if the message size is $O(1)$, we cannot include the node ID into the message (each node ID will be $O(\log N)$ bits) to distinguish messages from different sources in the BFS process. Thus, our major concern is to avoid sending nodes' IDs.

In this section, we present the method to compute name-independent APSP in $O(D + N/B)$ time. In order to match this bound, we first relabel the nodes to avoid sending nodes' IDs in subsection 3.1. That is, the new ID of each node does not need to be sent any more. For making the best use of the B bandwidth, we divide the bandwidth into multiple channels and perform BFS processes in each channel in subsection 3.2. Considering there may be collisions in each channel (violating the $\mathcal{CONGEST(B)}$ model), we present an algorithm scheduling the starting time of BFS processes to avoid it.

3.1 Relabel the Nodes

Given a spanning tree, we imagine a pebble [1] traversing the tree by DFS (Depth-First-Search), and at each step it records the node it visits. Denote a_k the time a pebble k-th time visits a node. So a_0 is the time a pebble enters node v for the first time and a_k is the time a pebble leaves node v for the last time if v only has k successors. We aim to get an array that records $a_0, \cdots, a_k$ for each node. Denote this array as τ where $\tau(i) = j$ means node j is visited by the pebble at time i.

To get array τ efficiently, we first compute the sizes of subtrees rooted at each node. This allows us to simulate DFS process by the sizes of subtrees. For a node v, if the size of its k-th subtree is s_k, we have $a_k = a_{k-1} + 2s_k$. It is easy for the root of the tree to compute a_k since a_0 is 0, then the root broadcasts a_{k-1} to its k-th successor and this successor takes $a_{k-1} + 1$ as its own a_0. We formulate this algorithm below.

- **Step 1:** Select a node to construct a BFS tree.
- **Step 2:** Each node in the tree computes its subtrees' sizes by convergecasting.
- **Step 3:** The root of the tree computes $a_0, a_1, \cdots, a_k$ where $a_0 = 0$ and $a_k = a_{k-1} + 2s_k$ if the root has k successors, then broadcasting a_{i-1} to its i-th successor where $0 < i \leq k$.
- **Step 4:** The i-th node which receives a_{i-1} sets $a_0 = a_{i-1} + 1$ and does the same operation as Step 3.
- **Step 5:** After all the leaves of the BFS tree finished Step 4, each node gets its local array τ.

We use array τ as the new labels of nodes. Since the new label implies the order of nodes visited in the DFS process, we get the following LEMMA 1 showing the relationship between distance and new labels.

LEMMA 1. *Denote the distance from u to v as $d(u, v)$. For each node, it maintains a local τ where $\tau(x) = y$ means node y is visited by the pebble at time x so that*

$$d(\tau(i), \tau(j)) \leq |i - j|.$$

Observe that nodes do not need to send the distance value in the BFS process. They only need to send a bit as a pebble. The problem is s_v and a_k can be as large as $O(N)$. We use pipelining to address this problem. That is, each node divides s_v into several messages of size B, then it sends the number of B bits at a time, from the least significant bits to the most significant bits. The pipelining takes $O(D + (\log N)/B)$ time. We conclude the time complexity of relabeling the nodes in LEMMA 2.

LEMMA 2. *Nodes can be relabeled in $O(D + (\log N)/B)$ time.*

3.2 Multiplexing

Recall that the basic idea of our algorithm is dividing the bandwidth into multiple channels and running the distributed BFS algorithm in each channel. If we divide the nodes into several groups and put them into different channels, there is no collision among the groups. So we only concern the schedule on one channel.

For the sake of generality, let b denote the size of messages used in the BFS process. Since only b bits are needed in one round when building each BFS tree, we divide the bandwidth into $R = \Theta(\min\{B/b, N\})$ channels. We partition the nodes relabeled in the subsection 3.1 into R groups:

$$U_i = \{\tau(j) | \lfloor \frac{i}{R}(2N-1) \rfloor \leq j < \lfloor \frac{i+1}{R}(2N-1) \rfloor\},$$

where U_i denotes the i-th group ($0 \leq i < R$), and $0 \leq j < 2N-1$ denotes the index of nodes in τ. Since a node appears many times in τ, it can be contained in multiple groups. In order to avoid repeatedly computing the distance between two nodes, we only allow each node to perform one BFS process. Note that we only deliver a pebble as the BFS message instead of sending nodes' IDs, so b is $O(1)$. The message's traveling time between two nodes implies the distance between them. If each node knows where the messages originate from, we can figure out the distance by the messages' arriving time. Due to the limit of space, we just give a sketch of the multiplexing technique below.

- **Step 1:** Divide the nodes into R groups U_i where $0 \leq i < R$. Each group corresponds to a channel.
- **Step 2:** In one group, the nodes perform BFS one by one according to their ranks in the group. We set the time interval between two BFS processes as three rounds.
- **Step 3:** The node which receives a BFS message deduces where the message originates from. Since this node could receive duplicated messages from the same source during three continuous rounds, we ignore messages in the latter two rounds.
- **Step 4:** This node computes the distance from the message's originating node to itself.
- **Step 5:** The algorithm is finished if all nodes in the groups finished their BFS process.

We only concern one channel in Steps 2-4. In Step 3, a node may receive several messages from its neighbors in three continuous rounds and these messages originate from one source. Since we only need the messages which travel along the shortest paths, the messages which travel along longer paths must arrive at the latter two rounds and should be ignored. We set time interval as three rounds to avoid ignoring messages originating from different sources so that the messages arriving at different time intervals must originate from different nodes. That is, the first batch of messages a node received must originate from the first node in the group. The second batch of messages which are not ignored must originate from the second node in the group. This allows us to compute the distance between two nodes by messages' arriving time. By using LEMMA 1, we show that there is no collision in one group in LEMMA 3.

LEMMA 3. *Denote BFS_v as a BFS performed by node v. When node j and node k are both in the same group, at no time a node is simultaneously visited by both BFS_j and BFS_k.*

By using LEMMA 2 and LEMMA 3, we conclude the time complexity of name-independent APSP in THEOREM 1.

THEOREM 1. *The name-independent APSP can be solved in $O(D + N/B)$ time by using Relabeling and Multiplexing techniques.*

4. APPLICATIONS

In this section, we introduce three applications using Relabeling and Multiplexing techniques. Due to the limit of space, we directly give the time complexities of these applications.

THEOREM 2. *The diameter can be computed in $O(N/B + D)$ rounds. This matches the lower bound of computing diameter $\Omega(N/B + D)$ in [1].*

THEOREM 3. *Girth can be computed in time $O(D+N/B)$.*

THEOREM 4. *Clustering coefficient of vertices can be computed in time $O(D + N/B)$.*

5. CONCLUSION

In this paper, we studied the distributed computation of APSP on an undirected and unweighted graph under synchronous communications and the $\mathcal{CONGEST}(\mathcal{B})$ model. By dividing the limited B bandwidth into multiple channels and by utilizing the multiplexing technique, we can push the parallelization of the distributed BFS processes to the limit such that the time complexity for distributively computing APSP can be reduced from $O(N+D)$ to $O(N/B+D)$ which meets the lower bound $\Omega(N/B + D)$. This technique might be of independent interest to tackle some other problems under the $\mathcal{CONGEST}(\mathcal{B})$ model.

Our distributed algorithm also implies a tight distributed algorithm for computing diameter whose time complexity is $\Theta(N/B + D)$. In addition, we also extend our distributed algorithm to compute the girth and the clustering coefficient in $O(N/D + B)$ time.

6. REFERENCES

[1] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proc. PODC*, 2012.

[2] David Peleg, Liam Roditty, and Elad Tal. Distributed algorithms for network diameter and girth. In *Proc. ICALP*, 2012.

[3] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.

[4] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. ISSAC*, 2014.

[5] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proc. STOC*, 2014.

[6] David Peleg. Distributed computing: a locality sensitive approach. *SIAM Monographs on discrete mathematics and applications*, 5, 2000.

[7] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proc. SODA*, 2012.