Connected Components on a PRAM in Log Diameter Time

Sixue Cliff Liu
Princeton University
Princeton, NJ, USA
sixuel@princeton.edu

Robert E. Tarjan
Princeton University
Princeton, NJ, USA
ret@princeton.edu

Peilin Zhong
Columbia University
New York City, New York, USA
peilin.zhong@columbia.edu

ABSTRACT

We present an $O(\log d + \log\log_{m/n} n)$-time randomized PRAM algorithm for computing the connected components of an n-vertex, m-edge undirected graph with maximum component diameter d. The algorithm runs on an ARBITRARY CRCW (concurrent-read, concurrent-write with arbitrary write resolution) PRAM using $O(m)$ processors. The time bound holds with good probability.[1]

Our algorithm is based on the breakthrough results of Andoni et al. [FOCS'18] and Behnezhad et al. [FOCS'19]. Their algorithms run on the more powerful MPC model and rely on sorting and computing prefix sums in $O(1)$ time, tasks that take $\Omega(\log n/\log\log n)$ time on a CRCW PRAM with poly(n) processors. Our simpler algorithm uses limited-collision hashing and does not sort or do prefix sums. It matches the time and space bounds of the algorithm of Behnezhad et al., who improved the time bound of Andoni et al.

It is widely believed that the larger private memory per processor and unbounded local computation of the MPC model admit algorithms faster than that on a PRAM. Our result suggests that such additional power might not be necessary, at least for fundamental graph problems like connected components and spanning forest.

CCS CONCEPTS

• **Mathematics of computing → Graph algorithms**; • **Theory of computation → Shared memory algorithms**.

KEYWORDS

PRAM; connected components; hashing

ACM Reference Format:
Sixue Cliff Liu, Robert E. Tarjan, and Peilin Zhong. 2020. Connected Components on a PRAM in Log Diameter Time. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20), July 15–17, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3350755.3400249

1 INTRODUCTION

Computing the connected components of an undirected graph is a fundamental problem in algorithmic graph theory, with many applications. Using graph search [32], one can find the connected components of an n-vertex, m-edge graph in $O(m)$ time, which is best possible for a sequential algorithm. But linear time is not fast enough for big-data applications in which the problem graph is of internet scale or even bigger. To find the components of such large graphs in practice requires the use of concurrency.

Beginning in the 1970's, theoreticians developed a series of more-and-more efficient concurrent algorithms. Their model of computation was some variant of the PRAM (parallel random access machine) model. Shiloach and Vishkin [29] gave an $O(\log n)$-time PRAM algorithm in 1982. The algorithm was later simplified by Awerbuch and Shiloach [5]. These algorithms are deterministic and run on an ARBITRARY CRCW PRAM with $O(m)$ processors. There are simpler algorithms and algorithms that do less work but use randomization. Gazit [13] combined an elegant randomized algorithm of Reif [27] with graph size reduction to obtain an $O(\log n)$-time, $O(m/\log n)$-processor CRCW PRAM algorithm. This line of work culminated in the $O(\log n)$-time, $O(m/\log n)$-processor EREW (exclusive-read, exclusive-write) PRAM algorithms of Halperin and Zwick [19, 20], the second of which computes spanning trees of the components as well as the components themselves. On an EREW PRAM, finding connected components takes $\Omega(\log n)$ time [9], so these algorithms minimize both time and work. The $\Omega(\log n)$ lower bound also holds for the CREW (concurrent-read, exclusive-write) PRAM with randomization, a model slightly weaker than the CRCW PRAM [11]. For the PRIORITY CRCW PRAM (write resolution by processor priority), a time bound of $\Omega(\log n/\log\log n)$ holds if there are poly(n) processors and unbounded space, or if there is poly(n) space and any number of processors [6].

The Halperin-Zwick algorithms use sophisticated techniques. Practitioners charged with actually finding the connected components of huge graphs have implemented much simpler algorithms. Indeed, such simple algorithms often perform well in practice [15, 18, 22, 30, 31]. The computational power of current platforms and the characteristics of the problem graphs may partially explain such good performance. Current parallel computing platforms such as MapReduce, Hadoop, Spark, and others have capabilities significantly beyond those modeled by the PRAM [10]. A more powerful model, the MPC (massively parallel computing) model [7] is intended to capture these capabilities. In this model, each processor can have poly(n) (typically sublinear in n) private memory, and the local computational power is unbounded. A PRAM algorithm can usually be simulated on an MPC with asymptotically the same round complexity, and it is widely believed that the MPC model admits algorithms faster than the PRAM model [4, 17, 24]. On the MPC model, and indeed on the weaker COMBINING CRCW PRAM model, there are very simple, practical algorithms that run in $O(\log n)$ time [25]. Furthermore, many graphs in applications have components of small diameter, perhaps poly-logarithmic in n.

[1] To simplify the statements in this paper we assume $m/2 \geq n \geq 2$ and $d \geq 1$. *With good probability* means with probability at least $1 - 1/\text{poly}((m\log n)/n)$; *with high probability* means with probability at least $1 - 1/\text{poly}(n)$.

SPAA '20, July 15–17, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-6935-0/20/07…$15.00
https://doi.org/10.1145/3350755.3400249

These observations lead to the question of whether one can find connected components faster on graphs of small diameter, perhaps by exploiting the power of the MPC model. Andoni et al. [3] answered this question "yes" by giving an MPC algorithm that finds connected components in $O(\log d \log\log_{m/n} n)$ time, where d is the largest diameter of a component. Very recently, this time bound was improved to $O(\log d + \log\log_{m/n} n)$ by Behnezhad et al. [8]. Both of these algorithms are complicated and use the extra power of the MPC model, in particular, the ability to sort and compute prefix sums in $O(1)$ communication rounds. These operations require $\Omega(\log n/\log\log n)$ time on a CRCW PRAM with poly(n) processors [6].[2] These results left open the following fundamental problem in theory:[3]

> *Is it possible to break the* $\log n$ *time barrier for connected components of small-diameter graphs on a PRAM (without the additional power of an MPC)?*

In this paper we give a positive answer by presenting an ARBITRARY CRCW PRAM algorithm that runs in $O(\log d + \log\log_{m/n} n)$ time, matching the round complexity of the MPC algorithm by Behnezhad et al. [8]. Our algorithm uses $O(m)$ processors and space, and thus is space-optimal and nearly work-efficient. In contrast to the MPC algorithm, which uses several powerful primitives, we use only hashing and other simple data structures. Our hashing-based approach also applies to the work of Andoni et al. [3], giving much simpler algorithms for connected components and spanning forest, which should be preferable in practice. While the MPC model ignores the total work, our result on the more fine-grained PRAM model captures the inherent complexities of the problems.

1.1 Computation Models and Main Results

Our main model of computation is the ARBITRARY CRCW PRAM [34]. It consists of a set of processors , each of which has a constant number of cells (words) as the private memory, and a large common memory. The processors run synchronously. In one step, a processor can read a cell in common memory, write to a cell in common memory, or do a constant amount of local computation. Any number of processors can read from or write to the same common memory cell concurrently. If more than one processor writes to the same memory cell at the same time, an arbitrary one succeeds.

Our main results are stated below:

Theorem 1.1 (Connected Components). *There is an ARBITRARY CRCW PRAM algorithm using $O(m)$ processors that computes the connected components of any given graph. With probability $1 - 1/poly((m\log n)/n)$, it runs in $O(\log d \log\log_{m/n} n)$ time.*

The algorithm of Theorem 1.1 can be extended to computing a spanning forest (a set of spanning trees of the components) with the same asymptotic efficiency:

Theorem 1.2 (Spanning Forest). *There is an ARBITRARY CRCW PRAM algorithm using $O(m)$ processors that computes the spanning forest of any given graph. With probability $1 - 1/poly((m\log n)/n)$, it runs in $O(\log d \log\log_{m/n} n)$ time.*

Using the above algorithms as bases, we provide a faster connected components algorithm that is nearly optimal (up to an additive factor of at most $O(\log\log n)$) due to a conditional lower bound of $\Omega(\log d)$ [8, 28].

Theorem 1.3 (Faster Connected Components). *There is an ARBITRARY CRCW PRAM algorithm using $O(m)$ processors that computes the connected components of any given graph. With probability $1 - 1/poly((m\log n)/n)$, it runs in $O(\log d + \log\log_{m/n} n)$ time.*

For a dense graph with $m = n^{1+\Omega(1)}$, the algorithms in all three theorems run in $O(\log d)$ time with probability $1 - 1/\text{poly}(n)$; if $d = \log^{\Omega(1)}_{m/n} n$, the algorithm in Theorem 1.3 runs in $O(\log d)$ time.[4]

Note that the time bound in Theorem 1.1 is dominated by the one in Theorem 1.3. We include Theorem 1.1 here in pursuit of simpler proofs of Theorem 1.2 and Theorem 1.3.

1.2 Related Work and Technical Overview

In this section, we give an overview of the techniques in our algorithms, highlighting the challenges in the PRAM model and the main novelty in our work compared to that of Andoni et al. [3] and Behnezhad et al. [8].

1.2.1 Related Work. Andoni et al. [3] observed that if every vertex in the graph has a degree of at least $b = m/n$, then one can choose each vertex as a *leader* with probability $\Theta(\log(n)/b)$ to make sure that with high probability, every non-leader vertex has at least 1 leader neighbor. As a result, the number of vertices in the contracted graph is the number of leaders, which is $\tilde{O}(n/b)$ in expectation, leading to double-exponential progress, since we have enough space to make each remaining vertex have degree $\tilde{\Omega}(m/(n/b)) = \tilde{\Omega}(b^2)$ in the next round and $\tilde{\Omega}(b^{2^i})$ after i rounds.[5] The process of adding edges is called *expansion*, as it expands the neighbor sets. It can be implemented to run in $O(\log d)$ time. This gives an $O(\log d \log\log_{m/n} n)$ running time.

Behnezhad et al. improved the multiplicative $\log\log_{m/n} n$ factor to an additive factor by streamlining the expansion procedure and double-exponential progress when increasing the space per vertex [8]. Instead of increasing the degree of each vertex *uniformly* to at least b in each round, they allow vertices to have different space budgets, so that the degree lower bound varies on the vertices. They define the *level* $\ell(v)$ of a vertex v to control the *budget* of v (space owned by v): initially each vertex is at level 1 with budget m/n. Levels increase over rounds. Each v is assigned a budget of $b(v) = (m/n)^{c^{\ell(v)}}$ for some fixed constant $c > 1$. The maximal level $L := \log_c \log_{m/n} n$ is such that a vertex at level L must have enough space to find all vertices in its component. Based on this idea, they design an MPC algorithm that maintains the following invariant:

Observation 1.4 ([8]). *With high probability, for any two vertices u and v at distance 2, after 4 rounds, if $\ell(u)$ does not increase then*

[2] Behnezhad et al. also consider the *multiprefix* CRCW PRAM, in which prefix sum (and other primitives) can be computed in $O(1)$ time and $O(m)$ work. A direct simulation of this model on a PRIORITY CRCW PRAM or weaker model would suffer an $\Omega(\log n/\log\log n)$ factor in both time and work, compared to our result.

[3] A repeated matrix squaring of the adjacency matrix computes the connected components in $O(\log d)$ time on a CRCW PRAM, but this is far from work-efficient – the currently best work is $O(n^{2.373})$ [12].

[4] Without the assumption that $m \geq 2n$ for simplification, one can replace the m with $2(m+n)$ in all our statements by creating self-loops in the graph.

[5] We use $\tilde{O}$ to hide polylog(n) factors.

their distance decreases to 1*; moreover, the* skipped *vertex* w *originally between* u *and* v *satisfies* $\ell(w) \le \ell(u)$.[6]

Behnezhad et al. proved an $O(\log d + \log\log_{m/n} n)$ time bound by a potential-based argument on an arbitrary fixed shortest path P_1 in the original graph: in round 1 put 1 coin on each vertex of P_1 (thus at most $d+1$ coins in total); for the purpose of analysis only, when inductively constructing P_{i+1} in round i, every skipped vertex on P_i is removed and passes its coins evenly to its successor and predecessor (if exist) on P_i. They claimed that any vertex v still on P_i in round i has at least $1.1^{i-\ell(v)}$ coins based on the following:

Observation 1.5 (Claim 3.12 in [8]). *For any path* P_i *in round* i *corresponding to an original shortest path, its first and last vertices are on* P_j *in round* j *for any* $j > i$*; moreover, if a vertex on* P_i *does not increase level in* 4 *rounds, then either its predecessor or successor on* P_i *is skipped.*

The first statement is to maintain the connectivity for every pair of vertices in the original graph, and the second statement is to guarantee that each vertex obtains enough coins in the next round. (Observation 1.5 is seemingly obvious from Observation 1.4, however there is a subtle issue overlooked in [8] that invalidates the statement. Their bound is still valid without changing the algorithm by another potential-based argument, which shall be detailed later in this section.)

Since the maximal level is L, by the above claim, a vertex on such a path with length more than 1 in round $R := 8\log d + L$ would have at least $1.1^{8\log d} > d+1$ coins, a contradiction, giving the desired time bound.

Let us first show a counter-example for Observation 1.5 (not for their time bound). Let the original path P_1 be $(v_1, v_2, \ldots, v_s)$. We want to show that the distance between v_1 and v_s is at most 1 after R rounds, so neither v_1 nor v_s can be skipped during the path constructions over rounds. Suppose v_1 does not increase level in 4 rounds, then for v_1 to obtain enough coins to satisfy the claim, v_2 must be skipped. We call an ordered pair (v_i, v_{i+1}) of consecutive vertices on the path *frozen* if v_i is kept and v_{i+1} is skipped in 4 rounds. So (v_1, v_2) is frozen. Let v_3 (any vertex after v_2 suffices) be the *only* vertex on P_1 directly connecting to v_1 after skipping v_2 in 4 rounds. (Observation 1.4 guarantees that a vertex directly connecting to v_1 must exist but cannot guarantee that there is more than 1 such vertex on P_1.) Note that v_3 *cannot* be skipped, otherwise v_1 is isolated from P_1 and thus cannot connect to v_s. Now consider two cases. If v_4 is skipped, then pair (v_3, v_4) is frozen. If v_4 is not skipped, then assume v_4 does not increase level in 4 rounds. For v_4 to obtain enough coins, v_5 must be skipped because v_3 cannot be skipped to pass coins to v_4, so the pair (v_4, v_5) is frozen. Observe that from frozen pair (v_1, v_2), in either case we get another frozen pair, which propagates inductively to the end of P_1. If we happen to have a frozen pair (v_{s-1}, v_s), then v_s must be skipped and isolated from v_1, a contradiction.

Here is a fix to the above issue (formally stated in Lemma 3.10). Note that only the last vertex v on the path can violate the claim that there are at least $1.1^{i-\ell(v)}$ coins on v in round i. Assuming the claim holds for all vertices on the current path, one can always *drop* (to distinguish from skip) v_s and pass its coins to v_{s-1} (which must be kept) if they are a frozen pair. So v_{s-1}, the new last vertex after 4 rounds, obtains enough coins by the induction hypothesis and the second part of Observation 1.4. By the same argument, the resulting path after R rounds has length at most 1. Observe that we dropped $O(R)$ vertices consecutively located at the end of the path, so the concatenated path connecting v_1 and the original v_s has length $O(R)$. Now applying Observation 1.4 to v_1, we get that in 4 rounds, either its level increases by 1 or its successor is skipped. Therefore, in $O(R+L)$ rounds, there is no successor of v_1 to be skipped and the graph has diameter at most 1 by a union bound over all the (original) shortest paths.

1.2.2 *Our Contributions.*

Now we introduce the new algorithmic ideas in our PRAM algorithm with a matching time bound.

The first challenge comes from processor allocation. To allocate different-sized blocks of processors to vertices in each round, there is actually an existing tool called *approximate compaction*, which maps the k distinguished elements in a length-n array one-to-one to an array of length $O(k)$ with high probability [2]. (The vertices to be assigned blocks are *distinguished* and their names are indices in the old array; after indexing them in the new array, one can assign them predetermined blocks.) However, the current fastest (and work-optimal) approximate compaction algorithm takes $O(\log^* n)$ time, introducing a multiplicative factor [14]. To avoid this, our algorithm first reduces the number of vertices to $n/\text{polylog}(n)$ in $O(\log\log_{m/n} n)$ time, then uses approximate compaction to rename the remaining vertices by an integer in $[n/\text{polylog}(n)]$ in $O(\log^* n)$ time. After this, each cell of the array to be compacted owns $\text{polylog}(n)$ processors, and each subsequent compaction (thus processor allocation) can be done in $O(1)$ time [16].

The second challenge is much more serious: it is required by the union bound that any vertex u must connect to *all* vertices within distance 2 with high probability if u does not increase in level. Behnezhad et al. achieve this goal by an algorithm based on constant-time sorting and prefix sum, which require $\Omega(\log n/\log\log n)$ time on an ARBITRARY CRCW PRAM with $\text{poly}(n)$ processors [6]. Our solution is based on hashing: to expand a vertex u, hash all vertices within distance 2 from u to a hash table owned by u; if there is a collision, increase the level of u.[7] As a result, we are able to prove the following result, which is stronger than Observation 1.4 as it holds deterministically and uses only 1 round:

Observation 1.6 (formally stated in Lemma 3.18). *For any two vertices* u *and* v *at distance 2, if* $\ell(u)$ *does not increase then their distance decreases to* 1 *in the next round; moreover, any vertex originally between* u *and* v *has level at most* $\ell(u)$.

The idea of increasing the level immediately after seeing a collision gives a much cleaner proof of Observation 1.6, but might be problematic in bounding the total space/number of processors: a vertex with many vertices within distance 2 can incur a collision and level increase very often. We circumvent this issue by increasing the level of each budget-b vertex with probability $\tilde{\Theta}(b^{-\delta})$ before hashing. Then a vertex v with at least b^{δ} vertices within distance 2 would see a level increase with high probability; if the level does not

[6] For simplicity, we ignore the issue of changing the graph and corresponding vertices for now.

[7] Hashing also naturally removes the duplicate neighbors to get the desired space bound – a goal achieved by sorting in [3, 8].

increase, there should be at most b^δ vertices within distance 2, thus there is a collision when expanding v with probability $1/\text{poly}(b)$. As a result, the probability of level increase is $\tilde{\Theta}(b^{-\delta})+1/\text{poly}(b) \le b^{-c}$ for some constant $c > 0$, and we can assign a budget of $b^{1+\Omega(c)}$ to a vertex with increased level, leading to double-exponential progress. As a result, the total space is $O(m)$ with good probability since the union bound is over all polylog(n) different levels and rounds, instead of $O(n^2)$ shortest paths.

Suitable combination of the ideas above yields a PRAM algorithm that reduces the diameter of the graph to at most 1 in $O(R) = O(\log d + \log\log_{m/n} n)$ time, with one *flexibility*: the relationship between the level $\ell(v)$ and budget $b(v)$ of vertex v in our algorithm is *not* strictly $b(v) = (m/n)^{c^{\ell(v)}}$ for some fixed constant $c > 1$ as in Behnezhad et al. [8]; instead, we allow vertices with the same level to have *two* different budgets. We show that such flexibility still maintains the key invariant of our algorithm (without influencing the asymptotic space bound): if a vertex is not a root in a tree in the labeled digraph, then its level must be strictly lower than the level of its parent (formally stated in Lemma 3.3).[8] Using hashing and a proper parent-update method, our algorithm does not need to compute the number of neighbors with a certain level for *each* vertex, which is required in [3, 8] and solved by constant-time sorting and prefix sum on an MPC. If this were done by a direct application of (constant-time) *approximate counting* (cf. [1]) on each vertex, then each round would take $\Omega(k)$ time where k is the maximal degree of any vertex, so our new ideas are essential to obtain the desired time bound.

Finally, we note that while it is straightforward to halt when the graph has diameter at most 1 in the MPC algorithm, it is not correct to halt (nor easy to determine) in this case due to the different nature of our PRAM algorithm. After the diameter reaches $O(1)$, to correctly compute components and halt the algorithm, we borrow an idea from [25] to flatten all trees in the labeled digraph in $O(R)$ time, then apply our *slower* connected components algorithm (cf. Theorem 1.1) to output the correct components in $O(\log\log_{m/n} n)$ time, which is $O(R)$ total running time.

2 PRELIMINARIES

2.1 Framework

We formulate the problem of computing connected components concurrently as follows: label each vertex v with a unique vertex $v.p$ in its component. Such a labeling gives a constant-time test for whether two vertices v and w are in the same component: they are if and only if $v.p = w.p$. We begin with every vertex self-labeled ($v.p = v$) and successively update labels until there is exactly one label per component.

The labels define a directed graph (*labeled digraph*) with arcs $(v, v.p)$, where $v.p$ is the *parent* of v. We maintain the invariant that the only cycles in the labeled digraph are self-loops (arcs of the form (v, v)). Then this digraph consists of a set of rooted trees, with v a root if and only if $v = v.p$. Some authors call the root of a tree the *leader* of all its vertices. We know of only one algorithm in the literature that creates non-trivial cycles, that of Johnson and Metaxis [23]. Acyclicity implies that when the parent of a root v changes, the new parent of v is not in the tree rooted at v (for any order of the concurrent parent changes). We call a tree *flat* if the root is the parent of every vertex in the tree. Some authors call flat trees *stars*.

In our connected components and spanning forest algorithms (see the full version of this paper), we maintain the additional invariant that if the parent of a non-root v changes, its new parent is in the same tree as v (for any order of the parent changes). This invariant implies that the partition of vertices among trees changes only by set union; that is, no parent change moves a proper subtree to another tree. We call this property *monotonicity*. Most of the algorithms in the literature that have a correct efficiency analysis are monotone. Liu and Tarjan [25] analyze some non-monotone algorithms. In our faster connected components algorithm (cf. §3), only the preprocessing and postprocessing stages are monotone, which means the execution between these two stages can move subtrees between different trees in the labeled digraph.

2.2 Building Blocks

Our algorithms use three standard and one not-so-standard building blocks, which link (sub)trees, flatten trees, alter edges, and add edges, respectively. (Classic PRAM algorithms develop many techniques to make the graph sparser, e.g., in [13, 19, 20], not denser by adding edges.)

We treat each edge $\{v, w\}$ as a pair of oppositely directed arcs (v, w) and (w, v). A *direct link* applies to a graph arc (v, w) such that v is a root and w is not in the tree rooted at v; it makes w the parent of v. A *parent link* applies to a graph arc (v, w) and makes $w.p$ the parent of v; note that v and $w.p$ are not necessarily roots. Concurrent direct links maintain monotonicity while concurrent parent links do not. We add additional constraints to prevent the creation of a cycle in both cases. Specifically, in the case of parent links, if a vertex is not a root in a tree in the labeled digraph, then its level must be strictly lower than the level of its parent (formally stated in Lemma 3.3).

Concurrent links can produce trees of arbitrary heights. To reduce the tree heights, we use the *shortcut* operation: for each v do $v.p := v.p.p$. One shortcut roughly halves the heights of all trees; $O(\log n)$ shortcuts make all trees flat. Hirschberg et al. [21] introduced shortcutting in their connected components algorithm; it is closely related to the *compress* step in tree contraction [26] and to *path splitting* in disjoint-set union [33].

Our third operation changes graph edges. To *alter* $\{v, w\}$, we replace it by $\{v.p, w.p\}$. Links, shortcuts, and edge alterations suffice to efficiently compute components. Liu and Tarjan [25] analyze simple algorithms that use combinations of our first three building blocks.

To obtain a good bound for small-diameter graphs, we need a fourth operation that adds edges. We *expand* a vertex u by adding an edge $\{u, w\}$ for a neighbor v of u and a neighbor w of v. The key idea for implementing expansion is hashing, which is presented below.

Suppose each vertex owns a block of K^2 processors. For each processor in a block, we index it by a pair $(p, q) \in [K] \times [K]$. For each vertex u, we maintain a size-K table $H(u)$. We choose a random

[8] Our algorithm adopts the framework of *labeled digraph* (or *parent graph*) for computing and representing components, which is standard in PRAM literatures, see §2.

hash function $h : [n] \to [K]$. At the beginning of an expansion, for each graph arc (u, v), we write vertex v into the $h(v)$-th cell of $H(u)$. Then we can expand u as follows: each processor (p, q) reads vertex v from the p-th cell of $H(u)$, reads vertex w from the q-th cell of $H(v)$, and writes vertex w into the $h(w)$-th cell of $H(u)$. For each $w \in H(u)$ after the expansion, $\{u, w\}$ is considered an *added* edge in the graph and is treated the same as any other edge.

The key difference between our hashing-based expansion and that in the MPC algorithms is that a vertex w within distance 2 from u might not be in $H(u)$ after the expansion due to a collision, so crucial to our analysis is the way to handle collisions. All hash functions in this paper are pairwise independent, so each processor doing hashing in each round only needs to read two words, which uses $O(1)$ private memory and time.

3 FASTER CONNECTED COMPONENTS ALGORITHM

In this section we prove Theorem 1.3 by presenting a faster algorithm for connected components.

> Faster Connected Components algorithm: repeat {EXPAND-MAXLINK} until the graph has diameter at most 1 and all trees are flat; run the connected components algorithm from Theorem 1.1 on the remaining graph.

Each iteration of the repeat loop is called a *round*. The *break condition* that the graph has diameter at most 1 and all trees are flat is tested at the end of each round.

To simplify the presentation, we make the following assumption, which is removed in the full version of this paper without influencing the asymptotic running time, number of processors, and success probability.

ASSUMPTION 3.1. *Let $c = 200$, at the beginning of the first round each vertex has a distinct id in $[2m/\log^c n]$ and owns a space block of size $\max\{m/n, \log^c n\}/\log^2 n$.*

The remainder of this section is organized as follows. The detailed method EXPAND-MAXLINK is presented in §3.1. The correctness of the algorithm is deferred to the full version of this paper. In §3.2, we implement EXPAND-MAXLINK on an ARBITRARY CRCW PRAM in $O(1)$ time. We prove that the algorithm uses $O(m)$ processors over all rounds in §3.3. Finally, we show that the graph diameter is at most 1 and all trees are flat after $O(\log d + \log\log_{m/n} n)$ rounds in §3.4. After the graph diameter reaches 1, it is easy to apply Theorem 1.1 to output the connected components of the input graph in $O(\log\log_{m/n} n)$ additional time, giving Theorem 1.3.

3.1 Algorithmic Framework

In this section, we present the algorithmic framework of EXPAND-MAXLINK, the ingredient of each round of Faster Connected Component algorithm.

EXPAND-MAXLINK uses the following three subroutines, which were introduced as building blocks in §2.2:

> ALTER: for each edge $e = \{v, w\}$: replace it by $\{v.p, w.p\}$.
> SHORTCUT: for each vertex u: update $u.p$ to $u.p.p$.
> MAXLINK: repeat {for each vertex v: let $u := \arg\max_{w \in N(v).p} \ell(w)$, if $\ell(u) > \ell(v)$ then update $v.p$ to u} for 2 iterations.

The fourth building block is to expand each vertex v to try to connect to all vertices within distance 2 from v, which corresponds to Steps (3-5). We give detailed explanations of the key concepts and steps after the algorithm.

> EXPAND-MAXLINK:
> (1) MAXLINK; ALTER.
> (2) For each root v: increase $\ell(v)$ with probability $10 \log n / b(v)^{0.1}$.
> (3) For each root v: for each root $w \in N(v)$: if $b(w) = b(v)$ then hash w into $H(v)$.
> (4) For each root v: if there is a collision in $H(v)$ then mark v as *dormant*. For each vertex v: if there is a dormant vertex in $H(v)$ then mark v as *dormant*.
> (5) For each root v: for each $w \in H(v)$: for each $u \in H(w)$: hash u into $H(v)$. For each root v: if there is a collision in $H(v)$ then mark v as *dormant*.
> (6) MAXLINK; SHORTCUT; ALTER.
> (7) For each root v: if v is dormant and did not increase level in Step (2) then increase $\ell(v)$.
> (8) For each root v: assign a block of size $b_{\ell(v)}$ to v.

Level and budget. The *level* $\ell(v)$ of a vertex v is a non-negative integer that can either remain the same or increase by one during a round. At the beginning of round 1, each vertex v is at level 1 and owns a block of size $b_1 := \max\{m/n, \log^c n\}/\log^2 n$ by Assumption 3.1. During a round, some roots become non-roots by updating their parents. If a vertex remains a root, its level might increase. A root with level ℓ is assigned a block of size $b_\ell := b_1^{1.01^{\ell-1}}$ at the end of the round. Given b, a vertex v has *budget* $b(v) := b$ if the current block owned by v has size b. Each block of size b is partitioned into $\sqrt{b}$ indexed tables, each of size $\sqrt{b}$.

Neighbor set. The edges that define the current graph include: (i) the (altered) original edges corresponding to edge processors, and (ii) the (altered) added edges in the tables over all rounds of all vertices. Any vertex within distance 1 of v (including v) in the current graph is called a *neighbor* of v. For any vertex v, let $N(v)$ be the set of its neighbors. In Step (3) we use the old $N(v)$ when initializing the loop that enumerates $N(v)$. For any vertex set S, define $N(S) := \bigcup_{w \in S} N(w)$, and define $S.p := \{w.p \mid w \in S\}$.

Hashing. At the beginning of a round, one random hash function h is chosen. All neighbor roots of all roots use h to do individual hashing in Step (3). A pairwise independent h suffices, so each processor only reads two words. The hashing in Step (5) uses the same h. For each vertex v, let $H(v)$ be the first table in its block, which will store the added edges incident on v. Step (5) is implemented by storing the old tables for all vertices while hashing new items

(copied from the old $H(v)$ and old $H(w)$ in the block of w) into the new table.

3.2 Implementation

In this section, we show how to implement EXPAND-MAXLINK on an ARBITRARY CRCW PRAM such that any of the first $O(\log n)$ rounds runs in constant time with good probability.

LEMMA 3.2. *With good probability, each of the first $O(\log n)$ rounds can be implemented to run in $O(1)$ time.*

PROOF. The ALTER (cf. Steps (1,6)) applies to all edges in the current graph. Since each edge corresponds to a distinct processor, Step (3) and the ALTER take $O(1)$ time.

Steps (2,4,7) and SHORTCUT take $O(1)$ time as each vertex has a corresponding processor and a collision can be detected using the same hash function to check the same location again: there is a collision in $H(v)$ if a vertex w reads a vertex different from w from the $h(w)$-th cell of $H(v)$ (then w can write a flag to the processor of v to indicate the collision).

In each of the two iterations of MAXLINK, each vertex v updates its parent to a neighbor parent with the highest level if this level is higher than $\ell(v)$. Since a vertex can increase its level by at most 1 in any round (cf. Steps (2,7)), there are $O(\log n)$ different levels. Let each neighbor of v write its parent with level ℓ to the ℓ-th cell of an array of length $O(\log n)$ in the block of v. By the definitions of level and budget, the block of any vertex in any round has size at least $b_1 = \Omega(\log^3 n)$. Therefore, we can assign a processor to each pair of the cells in this array, such that each non-empty cell can determine whether there is a non-empty cell with a larger index in $O(1)$ time. For any non-empty cell, if there is no non-empty cell with a larger index, it must contain a vertex with the maximum level. As a result, Steps (1,6) take $O(1)$ time.

By Step (3), any $w \in H(v)$ has $b(w) = b(v)$, so each $u \in H(w)$ such that $w \in H(v)$ owns a processor in the block of v since $\sqrt{b(v)} \cdot \sqrt{b(w)} = b(v)$ and any vertex in a table is indexed (by its hash value). Therefore, together with collision detection, Step (5) takes $O(1)$ time.

In Step (8), each vertex is assigned a block. The pool of $\Theta(m)$ processors is partitioned into $\Theta(\log^2 n)$ zones such that the processor allocation in round r for vertices with level ℓ uses the zone indexed by (r, ℓ), where $r, \ell \in O(\log n)$ in the first $O(\log n)$ rounds. Since there are $\Theta(m)$ processors in total and all the vertex ids are in $[2m/\log^c n]$ with good probability (cf. Assumption 3.1), we can use $\Theta(m/\log n)$ processors for each different level and apply approximate compaction (cf. §1.2.2 and see the full version of this paper) to index each root in $O(1)$ time with high probability such that the indices of vertices with the same level are distinct, then assign each of them a distinct block in the corresponding zone. Therefore, Step (8) takes $O(1)$ time with good probability by a union bound over all $O(\log n)$ levels and rounds.

Finally, we need to implement the break condition in $O(1)$ time, i.e., to determine whether the graph has diameter at most 1 and all trees are flat at the end of each round. The algorithm checks the following 2 conditions in each round: (i) all vertices do not change their parents nor levels in this round, and (ii) for any vertices v, w, u such that $w \in H(v)$, $u \in H(w)$ before Step (5), the $h(u)$-th cell in $H(v)$ already contains u. Conditions (i) and (ii) can be checked in $O(1)$ time by writing a flag to vertex processor v if they do not hold for v, then let each vertex with a flag write the flag to a fixed processor. If there is no such flag then both Conditions (i) and (ii) hold and the loop breaks. If there is a non-flat tree, some parent must change in the SHORTCUT in Step (6). If all trees are flat, they must be flat before the ALTER in Step (6), then an ALTER moves all edges to the roots. Therefore, if Condition (i) holds, all trees are flat and edges are only incident on roots. Moreover, no level changing means no vertex increase its level in Step (2) and there is no dormant vertex in Step (7). So for each root v, $N(v) = H(v)$ after Step (3) and $N(N(v)) = H(v)$ after Step (5) as there is no collision. By Condition (ii), the table $H(v)$ does not change during Step (5), so $N(v) = N(N(v))$. If there exists root v such that there is another root with distance at least 2 from v, then there must exist a vertex $w \neq v$ at distance exactly 2 from v, so $w \notin N(v)$ and $w \in N(N(v))$, contradicting with $N(v) = N(N(v))$. Therefore, any root is within distance at most 1 from all other roots in its component and the graph has diameter at most 1.

Since each step runs in $O(1)$ time with good probability, the lemma follows. □

3.3 Number of Processors

In this section, we show that with good probability, the first $O(\log n)$ rounds use $O(m)$ processors in total.

First of all, we prove a useful property on levels and roots.

LEMMA 3.3. *If a vertex v is a non-root at any step, then during the execution after that step, v is a non-root, $\ell(v)$ cannot change, and $1 \leq \ell(v) < \ell(v.p)$.*

PROOF. The proof is by an induction on rounds. The lemma clearly holds at the beginning of the first round by definitions.

In Step (1), each iteration of a MAXLINK can only update the parent to a vertex with higher level, which cannot be itself. In Step (2), level increase only applies to roots. The invariant holds after the MAXLINK in Step (6) for the same reasons as above. In SHORTCUT (cf. Step (6)), each vertex v updates its parent to $v.p.p$, which, by the induction hypothesis, must be a vertex with level higher than v if v is a non-root, thus cannot be v. In Step (7), level increase only applies to roots. All other steps and ALTERS do not change the labeled digraph nor levels, giving the lemma. □

Now observe that in the case that a root v with level ℓ increases its level in Step (2) but becomes a non-root at the end of the round, v is not assigned a block of size $b_{\ell(v)}$ in Step (8). Instead, v owns a block of size $b_\ell = b_{\ell(v)-1}$ from the previous round. Since in later rounds a non-root never participates in obtaining more neighbors by maintaining its table in Steps (3-5) (which is the only place that requires a larger block), such *flexibility* in the relationship between level and budget is acceptable.

By the fact that any root v at the end of any round owns a block of size $b_{\ell(v)} = b_1^{1.01^{\ell(v)-1}}$, a non-root can no longer change its level nor budget (cf. Lemma 3.3), and the discussion above, we obtain:

COROLLARY 3.4. *Any vertex v owns a block of size b at the end of any round where $b_{\ell(v)-1} = b_1^{1.01^{\ell(v)-2}} \leq b \leq b_1^{1.01^{\ell(v)-1}} = b_{\ell(v)}$; if v is a root, then the upper bound on b is tight.*

Secondly, we prove two simple facts about MAXLINK.

LEMMA 3.5. *For any vertex v with parent v' and any $w \in N(v)$ before an iteration of MAXLINK, $\ell(w.p) \geq \ell(v')$ after the iteration; furthermore, if $\ell(w.p) > \ell(v)$ before an iteration, then v must be a non-root after the iteration.*

PROOF. For any $w \in N(v)$, its parent has level $\max_{u \in N(w)} \ell(u.p)$ at least $\ell(v')$ after an iteration of MAXLINK. This implies that if $\ell(w.p) > \ell(v)$ before an iteration, then after that $\ell(v.p)$ is at least the level of the old parent of w which is strictly higher than $\ell(v)$, so v must be a non-root. □

LEMMA 3.6. *For any root v with budget b at the beginning of any round, if there is a root $w \in N(v)$ with at least $b^{0.1}$ neighbor roots with budget b after Step (1), then v either increases level in Step (2) or is a non-root at the end of the round with probability $1 - n^{-5}$.*

PROOF. Assume v does not increase level in Step (2). Let w be any root in $N(v)$ after Step (1). Since each root $u \in N(w)$ with budget b (thus level at least $\ell(v)$ by Corollary 3.4) increases its level with probability $10 \log n / b^{0.1}$ independently, with probability at least $1 - (1 - 10 \log n / b^{0.1})^{b^{0.1}} \geq 1 - n^{-5}$, at least one u increases level to at least $\ell(v) + 1$ in Step (2). Since $u \in N(N(v))$ after Step (1), by Lemma 3.5, there is a $w' \in N(v)$ such that $\ell(w'.p) \geq \ell(v) + 1$ after the first iteration of MAXLINK in Step (6). Again by Lemma 3.5, this implies that v cannot be a root after the second iteration and the following SHORTCUT. □

Using the above result, we can prove the following key lemma, leading to the total number of processors.

LEMMA 3.7. *For any root v with budget b at the beginning of any round, $b(v)$ is increased to $b^{1.01}$ in this round with probability at most $n^{-5} + b^{-0.05}$.*

PROOF. In Step (2), $\ell(v)$ increases with probability $10 \log n / b^{0.1} \leq b^{-0.08}$ when $c \geq 100$, since $b \geq b_1 \geq \log^{c-2} n$. If $\ell(v)$ does increase here then it cannot increase again in Step (7), so we assume this is not the case (and apply a union bound at the end).

If there is a root $w \in N(v)$ with at least $b^{0.1}$ neighbor roots with budget b after Step (1), then v is a root at the end of the round with probability at most n^{-5} by the assumption and Lemma 3.6. So we assume this is not the case.

By the previous assumption we know that at most $b^{0.1}$ vertices are hashed into $H(v)$ in Step (3). By pairwise independency, with probability at most $(b^{0.1})^2/\sqrt{b} = b^{-0.3}$ there is a collision as the table has size $\sqrt{b}$, which will increase $\ell(v)$ (cf. Steps (4,7)).

Now we assume that there is no collision in $H(v)$ in Step (3), which means $H(v)$ contains all the at most $b^{0.1}$ neighbor roots with budget b. By the same assumption, each such neighbor root w has at most $b^{0.1}$ neighbor roots with budget b, so there is a collision in $H(w)$ in Step (3) with probability at most $(b^{0.1})^2/\sqrt{b} = b^{-0.3}$. By a union bound over all the $|H(v)| \leq b^{0.1}$ such vertices, v is marked as dormant in the second statement of Step (4) (and will increase level in Step (7)) with probability $b^{-0.2}$.

It remains to assume that there is no collision in $H(v)$ nor in any $H(w)$ such that $w \in H(v)$ after Step (4). As each such table contains at most $b^{0.1}$ vertices, in Step (5) there are at most $b^{0.2}$ vertices to be hashed, resulting in a collision in $H(v)$ with probability at most $(b^{0.2})^2/\sqrt{b} = b^{-0.1}$, which increases $\ell(v)$ in Step (7).

Observe that only a root v at the end of the round can increase its budget, and the increased budget must be $b^{1.01}$ since the level can increase by at most 1 during the round and $b = b_{\ell(v)}$ at the beginning of the round by Corollary 3.4. By a union bound over the events in each paragraph, $b(v)$ is increased to $b^{1.01}$ with probability at most $b^{-0.08} + n^{-5} + b^{-0.3} + b^{-0.2} + b^{-0.1} \leq n^{-5} + b^{-0.05}$. □

Finally, we are ready to prove an upper bound on the number of processors.

LEMMA 3.8. *With good probability, the first $O(\log n)$ rounds use $O(m)$ processors in total.*

PROOF. Using Lemma 3.7, by a union bound over all $O(n)$ roots, all $O(\log n)$ rounds, and all $O(\log n)$ different budgets (since there are $O(\log n)$ different levels), with probability at least $1 - n^{-3}$, any root v with budget b at the beginning of any round increases its budget to $b^{1.01}$ with probability at most $b^{-0.05}$. We may assume that the $(1-n^{-3})$-probability event always holds since a good-probability result follows from a union bound.

The number of processors for (altered) original edges and vertices are clearly $O(m)$ over all rounds (where each vertex processor needs $O(1)$ private memory to store the corresponding parent, vertex id, hash function, level, and budget). Therefore, we only need to bound the number of processors in blocks that are assigned to a vertex in Step (8) in all $O(\log n)$ rounds. (In the full version of this paper, we show that the overhead in Step (8) is $O(1)$ with high probability.)

For any positive integer ℓ, let n_ℓ be the number of vertices that ever reaches budget b_ℓ during the first $O(\log n)$ rounds. For any vertex v that ever reaches budget b_ℓ, it has exactly one chance to reach budget $b_{\ell+1}$ in a round if v is a root in that round, which happens with probability at most ${b_\ell}^{-0.05}$. By a union bound over all $O(\log n)$ rounds, v reaches budget $b_{\ell+1}$ with probability at most $O(\log n) \cdot {b_\ell}^{-0.05} \leq {b_\ell}^{-0.04}$ when $c \geq 200$, since $b_\ell \geq b_1 \geq \log^{c-2} n$. We obtain $\mathbb{E}[n_{\ell+1} \mid n_\ell] \leq n_\ell \cdot {b_\ell}^{-0.04}$, thus by $b_{\ell+1} = {b_\ell}^{1.01}$, it must be:

$$\mathbb{E}[n_{\ell+1} b_{\ell+1} \mid n_\ell] \leq n_\ell \cdot {b_\ell}^{-0.04} \cdot {b_\ell}^{1.01} = n_\ell b_\ell \cdot {b_\ell}^{-0.03}.$$

By Markov's inequality, $n_{\ell+1} b_{\ell+1} \leq n_\ell b_\ell$ with probability at least $1 - {b_\ell}^{-0.03} \geq 1 - {b_1}^{-0.03}$. By a union bound over all $\ell \in O(\log n)$, $n_\ell b_\ell \leq n_1 b_1$ for all $\ell \in O(\log n)$ with probability at least $1 - O(\log n) \cdot {b_1}^{-0.03} \geq 1 - {b_1}^{-0.01}$, which is $1 - 1/\mathrm{poly}((m \log n)/n)$ by $b_1 = \max\{m/n, \log^c n\}/\log^2 n$ and $c \geq 100$. So the number of new allocated processors for vertices with any budget in any of the first $O(\log n)$ rounds is at most $n_1 b_1$ with good probability.

Recall from Assumption 3.1 and by a direct calculation, $n_1 \cdot b_1 = O(m/\log^2 n)$ with good probability. Therefore, by a union bound over all the $O(\log n)$ different budgets and $O(\log n)$ rounds, with good probability the total number of processors is $O(m)$. □

3.4 Diameter Reduction

Let $R := O(\log d + \log\log_{m/n} n)$ where the constant hidden in O will be determined later in this section. The goal is to prove that $O(R)$ rounds of EXPAND-MAXLINK suffice to reduce the diameter of the graph to $O(1)$ and flatten all trees with good probability.

In a high level, our algorithm/proof is divided into the following 3 stages/lemmas:

Lemma 3.9. *With good probability, after round R, the diameter of the graph is $O(R)$.*

Lemma 3.10. *With good probability, after round $O(R)$, the diameter of the graph is at most* 1.

Lemma 3.11. *With good probability, after round $O(R)$, the diameter of the graph is at most* 1 *and all trees are flat.*

3.4.1 Path Construction. To formalize and quantify the effect of reducing the diameter, consider any shortest path P in the input graph, whose length $|P|$ is at most d. Each ALTER (cf. Steps (1,6)) in each round replaces each vertex on P by its parent, resulting in a path P' of the same length as P. (Note that P' might not be a shortest path in the current graph and can contain loops.) We also add edges to the graph for reducing the diameter of the current graph: for any vertices v and w on path P', if the current graph contains edge (v, w), then all vertices exclusively between v and w can be removed from P', which still results in a valid path in the current graph from the first to the last vertex of P', reducing the length. If all such paths reduce their lengths to at most d', the diameter of the current graph is at most d'. In the following, consider any fixed shortest path P_1 at the beginning of round 1. Formally, we have the following inductive construction of paths for diameter reduction:[9]

Definition 3.12 (path construction). Let all vertices on P_1 be *active*. For any positive integer r, given path P_r with at least 4 active vertices at the beginning of round r, EXPAND-MAXLINK constructs P_{r+1} by the following 7 phases:

(1) The ALTER in Step (1) replaces each vertex v on P_r by $v' := v.p$ to get path $P_{r,1}$. For any v' on $P_{r,1}$, let $\underline{v'}$ be on P_r such that $\underline{v'}.p = v'$.
(2) Let the subpath containing all active vertices on $P_{r,1}$ be $P_{r,2}$.
(3) After Step (5), set i as 1, and repeat the following until $i \geq |P_{r,2}|-1$: let $v' := P_{r,2}(i)$, if $\underline{v'}$ is a root and does not increase level during round r then: if the current graph contains edge $(v', P_{r,2}(i+2))$ then mark $P_{r,2}(i+1)$ as *skipped* and set i as $i+2$; else set i as $i+1$.
(4) For each $j \in [i+1, |P_{r,2}|+1]$, mark $P_{r,2}(j)$ as *passive*.
(5) Remove all skipped and passive vertices from $P_{r,2}$ to get path $P_{r,5}$.
(6) Concatenate $P_{r,5}$ with all passive vertices on $P_{r,1}$ and $P_{r,2}$ to get path $P_{r,6}$.
(7) The ALTER in Step (6) replaces each vertex v on $P_{r,6}$ by $v.p$ to get path P_{r+1}.

For any vertex v on P_r that is replaced by v' in Phase (1), if v' is not skipped in Phase (3), then let $\overline{v}$ be the vertex replacing v' in Phase (7), and call $\overline{v}$ the *corresponding vertex* of v in round $r+1$.

Lemma 3.13. *For any non-negative integer r, the P_{r+1} constructed in Definition 3.12 is a valid path in the graph and all passive vertices are consecutive from the successor of the last active vertex to the end of P_{r+1}.*

[9]For any $i \in [|P|+1]$, let $P(i)$ be the i-th vertex on P.

Proof. The proof is by an induction on r. Initially, P_1 is a valid path by our discussion on ALTER at the beginning of this section: it only replaces edges by new edges in the altered graph; moreover, the second part of the lemma is trivially true as all vertices are active. Assuming P_r is a valid path and all passive vertices are consecutive from the successor of the last active vertex to the end of the path. We show the inductive step by proving the invariant after each of the 7 phases in Definition 3.12. Phase (1) maintains the invariant. In Phase (2), $P_{r,2}$ is a valid path as all active vertices are consecutive at the beginning of $P_{r,1}$ (induction hypothesis). In Phase (3), if a vertex v is skipped, then there is an edge between its predecessor and successor on the path; otherwise there is an edge between v and its successor by the induction hypothesis; all passive vertices are consecutive from the successor of the last non-skipped vertex to the end of $P_{r,2}$ (cf. Phase (4)), so the invariant holds. In Phase (6), since the first passive vertex on $P_{r,2}$ is a successor of the last vertex on $P_{r,5}$ and the last passive vertex on $P_{r,2}$ is a predecessor of the first passive vertex on $P_{r,1}$ (induction hypothesis), the invariant holds. Phase (7) maintains the invariant. Therefore, P_{r+1} is a valid path and all passive vertices are consecutive from the successor of the last active vertex to the end of P_{r+1}. □

Now we relate the path construction to the diameter of the graph:

Lemma 3.14. *For any positive integer r, the diameter of the graph at the end of round r is $O(\max_{P_r} |P_{r,2}| + r)$.*

Proof. Let P_1 be from s to t. By an induction on the number of ALTERs and Lemma 3.13, the corresponding vertices of s and t are still connected by path P_{r+1} at the end of round r. Note that by Lemma 3.13, P_{r+1} can be partitioned into two parts after Phase (2): subpath $P_{r,2}$ and the subpath containing only passive vertices. Since in each round we mark at most 2 new passive vertices (cf. Phases (3,4)), we get $|P_{r+1}| \leq |P_{r,5}| + 2r \leq |P_{r,2}| + 2r$. If any path P_{r+1} that corresponds to a shortest path in the original graph have length at most d', the graph at the end of round r must have diameter at most d', so the lemma follows. □

It remains to bound the length of any $P_{r,2}$ in any round r, which relies on the following potential function:

Definition 3.15. For any vertex v on P_1, define its *potential* $\phi_1(v) := 1$. For any positive integer r, given path P_r with at least 4 active vertices at the beginning of round r and the potentials of vertices on P_r, define the *potential* of each vertex on P_{r+1} based on Definition 3.12 as follows:

- For each v replaced by $v.p$ in Phase (1), $\phi_{r,1}(v.p) := \phi_r(v)$.
- After Phase (4), for each active vertex v on $P_{r,2}$, if the successor w of v is skipped or passive, then $\phi_{r,4}(v) := \phi_{r,1}(v) + \phi_{r,1}(w)$.
- After Phase (6), for each vertex v on $P_{r,6}$, if v is active on $P_{r,2}$, then $\phi_{r,6}(v) := \phi_{r,4}(v)$, otherwise $\phi_{r,6}(v) := \phi_{r,1}(v)$.
- For each v replaced by $v.p$ in Phase (7), $\phi_{r+1}(v.p) := \phi_{r,6}(v)$.

We conclude this section by some useful properties of potentials.

Lemma 3.16. *For any path P_r at the beginning of round $r \geq 1$, the following holds: (i) $\sum_{v \in P_r} \phi_r(v) \leq d+1$; (ii) for any v on P_r, $\phi_r(v) \geq 1$; (iii) for any non-skipped v on $P_{r,2}$ and its corresponding vertex $\overline{v}$ on P_{r+1}, $\phi_{r+1}(\overline{v}) \geq \phi_r(v)$.*

Proof. The proof is by an induction on r. The base case follows from $\phi(v) = 1$ for each v on P_1 (cf. Definition 3.15) and $|P_1| \le d$. For the inductive step, note that by Definition 3.15, the potential of a corresponding vertex is at least the potential of the corresponding vertex in the previous round (and can be larger in the case that its successor is skipped or passive). This gives (ii) and (iii) of the lemma. For any vertex u on $P_{r,2}$, if both u and its successor are active, then $\phi_r(u)$ is presented for exactly 1 time in $\sum_{v\in P_r}\phi_r(v)$ and $\sum_{v\in P_{r+1}}\phi_{r+1}(v)$ respectively; if u is active but its successor w is skipped or passive, then $\phi_r(u)+\phi_r(w)$ is presented for exactly 1 time in each summations as well; if u and its predecessor are both passive, then $\phi_r(u)$ is presented only in $\sum_{v\in P_r}\phi_r(v)$; the potential of the last vertex on $P_{r,2}$ might not be presented in $\sum_{v\in P_{r+1}}\phi_{r+1}(v)$ depending on i after Phase (3). Therefore, $\sum_{v\in P_{r+1}}\phi_{r+1}(v) \le \sum_{v\in P_r}\phi_r(v)$ and the lemma holds. □

3.4.2 Remaining Proofs: Proof of Lemma 3.9. First of all, we need an upper bound on the maximal possible level:

Lemma 3.17. *With good probability, the level of any vertex in any of the first $O(\log n)$ rounds is at most $L := 1000\max\{2, \log\log_{m/n} n\}$.*

Proof. By Lemma 3.8, with good probability the total number of processors used in the first $O(\log n)$ rounds is $O(m)$. We shall condition on this happening then assume for contradiction that there is a vertex v with level at least L in some round.

If $\log\log_{m/n} n \le 2$, then $m/n \ge n^{1/4}$. By Corollary 3.4, a block owned by v has size at least

$$b_1^{1.01^{2000-2}} \ge b_1^{20} \ge (m/n/\log^2 n)^{20} \ge (n^{1/5})^{20} = n^4 \ge m^2,$$

which is a contradiction as the size of this block owned by v exceeds the total number of processors $O(m)$.

Else if $\log\log_{m/n} n > 2$, then by Corollary 3.4, a block owned by v has size at least

$$b_1^{1.01^{L-2}} \ge b_1^{1.01^{999\log\log_{m/n} n}} \ge b_1^{(\log_{m/n} n)^{10}} \ge b_1^{8\log_{m/n} n}. \quad (1)$$

Whether $m/n \le \log^c n$ or not, if $c \ge 10$, it must be $b_1 = \max\{m/n, \log^c n\}/\log^2 n \ge \sqrt{m/n}$. So the value of (1) is at least $n^4 \ge m^2$, contradiction. Therefore, the level of any vertex is at most L. □

We also require the following key lemma:

Lemma 3.18. *For any root v and any $u \in N(N(v))$ at the beginning of any round, let u' be the parent of u after Step (1). If v does not increase level and is a root during this round, then $u' \in H(v)$ after Step (5).*

To prove Lemma 3.18, we use another crucial property of the algorithm, which is exactly the reason behind the design of maxlink.

Lemma 3.19. *For any root v and any $u \in N(N(v))$ at the beginning of any round, if v does not increase level in Step (2) and is a root at the end of the round, then $u.p$ is a root with budget $b(v)$ after Step (1).*

Proof. By Lemma 3.3, v is a root during this round. For any $w \in N(v)$ and any $u \in N(w)$, applying Lemma 3.5 for 2 times, we get that $\ell(v) \le \ell(w.p)$ and $\ell(v) \le \ell(u.p)$ after the maxlink in Step (1). If there is a $u \in N(N(v))$ such that $u.p$ is a non-root or $\ell(u.p) > \ell(v)$ before the alter in Step (1), it must be $\ell(u.p.p) > \ell(v)$ by Lemma 3.3. Note that $u.p$ is in $N(N(v))$ after the alter, which still holds before Step (6) as we only add edges. By Lemma 3.5, there is a $w' \in N(v)$ such that $\ell(w'.p) > \ell(v)$ after the first iteration of maxlink in Step (6). Again by Lemma 3.5, this implies that v cannot be a root after the second iteration, a contradiction. Therefore, for any $u \in N(N(v))$, $u.p$ is a root with level $\ell(v)$ (thus budget $b(v)$) after Step (1). □

With the help of Lemma 3.19 we can prove Lemma 3.18:

Proof of Lemma 3.18. For any vertex u, let $N'(u)$ be the set of neighbors after Step (1). First of all, we show that after Step (5), $H(v)$ contains all roots in $N'(N'(v))$ with budget b, where b is the budget of v at the beginning of the round. For any root $w \in N'(v)$, in Step (3), all roots with budget $b(w)$ in $N'(w)$ are hashed into $H(w)$. If there is a collision in any $H(w)$, then v must be dormant (cf. Step (4)) thus increases level in either Step (2) or (7), contradiction. So there is no collision in $H(w)$ for any $w \in N'(v)$, which means $H(w) \supseteq N'(w)$. Recall that $v \in N'(v)$ and we get that all roots with budget $b(w) = b$ from $N'(N'(v))$ are hashed into $H(v)$ in Step (5). Again, if there is a collision, then v must be dormant and increase level in this round. Therefore, $N'(N'(v)) \subseteq H(v)$ at the end of Step (5).

By Lemma 3.19, for any $u \in N(N(v))$ at the beginning of any round, $u' = u.p$ is a root with budget b in $N'(N'(v))$ after Step (1). Therefore, $u' \in H(v)$ at the end of Step (5), giving Lemma 3.18. □

The proof of Lemma 3.9 relies on the following lemma based on potentials:

Lemma 3.20. *At the beginning of any round $r \ge 1$, for any active vertex v on any path P_r, $\phi_r(v) \ge 2^{r-\ell(v)}$.*

Proof. The proof is by an induction on r. The base case holds because for any (active) vertex v on P_1, $\phi_1(r) = 1$ and $r = \ell(v) = 1$. Now we prove the inductive step from r to $r+1$ given that the corresponding vertex $\overline{v}$ of $v \in P_r$ is on P_{r+1} and active.

Suppose v is a non-root at the end of round r. If v is a non-root at the end of Step (1), then $\ell(v.p) > \ell(v)$ after Step (1) by Lemma 3.3, and $\ell(\overline{v}) \ge \ell(v.p) > \ell(v)$; else if v first becomes a non-root in Step (6), then $\overline{v} = v.p$ and $\ell(v.p) > \ell(v)$ after Step (6) by Lemma 3.3. So by the induction hypothesis, $\phi_{r+1}(\overline{v}) \ge \phi_r(v) \ge 2^{r-\ell(v)} \ge 2^{r+1-\ell(\overline{v})}$.

Suppose v increases its level in round r. Let ℓ be the level of v at the beginning of round r. If the increase happens in Step (2), then v is a root after Step (1). Whether v changes its parent in Step (6) or not, the level of $\overline{v} = v.p$ is at least $\ell+1$. Else if the increase happens in Step (7), then v is a root after Step (6). So $\overline{v} = v$ and its level is at least $\ell+1$ at the end of the round. By the induction hypothesis, $\phi_{r+1}(\overline{v}) \ge \phi_r(v) \ge 2^{r-\ell} \ge 2^{r+1-\ell(\overline{v})}$.

It remains to assume that v is a root and does not increase level during round r. By Lemma 3.18, for any $u \in N(N(v))$ at the beginning of round r, the parent u' of u after Step (1) is in $H(v)$ after Step (5). Since v is a root during the round, it remains on $P_{r,2}$ after Phase (2). We discuss two cases depending on whether v is at position before $|P_{r,2}|-1$ or not.

In Phase (3), note that if $v = P_{r,2}(i)$ where $i < |P_{r,2}|-1$, then $P_{r,2}(i+2)$ is the parent of a vertex in $N(N(v))$ after Step (1), which must be in $H(v)$ after Step (5). Therefore, the graph contains edge $(v, P_{r,2}(i+2))$ and $v' := P_{r,2}(i+1)$ is skipped, thus $\phi_{r,4}(v) =$

$\phi_{r,1}(v) + \phi_{r,1}(v')$ by Definition 3.15. Since $i+1 \le |P_{r,2}|+1$, v' is an active vertex on $P_{r,1}$. By the induction hypothesis, $\phi_{r,1}(v') = \phi_r(\underline{v'}) \ge 2^{r-\ell(\underline{v'})}$ (recall that $\underline{v'}$ is replaced by its parent v' in Phase (1)/Step (1)). If $\ell(\underline{v'}) > \ell(v)$, then applying Lemma 3.5 for 2 times we get that v is a non-root after Step (1), a contraction. Therefore, $\phi_{r,1}(v') \ge 2^{r-\ell(\underline{v'})} \ge 2^{r-\ell(v)}$ and $\phi_{r,4}(v) \ge \phi_{r,1}(v) + \phi_{r,1}(v') \ge \phi_r(v) + 2^{r-\ell(v)} \ge 2^{r+1-\ell(v)}$.

On the other hand, if $i \ge |P_{r,2}| - 1$ is reached after Phase (3), it must be $i < |P_{r,2}| + 1$ by the break condition of the loop in Phase (3). Note that $v' := P_{r,2}(i+1)$ is marked as passive in Phase (4), and by Definition 3.15, $\phi_{r,4} = \phi_{r,1}(v) + \phi_{r,1}(v')$. Moreover, since $i+1 \le |P_{r,2}|+1$, v' is an active vertex on $P_{r,1}$. Using the same argument in the previous paragraph, we obtain $\phi_{r,4}(v) \ge 2^{r+1-\ell(v)}$.

By Definition 3.15, after Phase (7), $\phi_{r+1}(\overline{v}) = \phi_{r,6}(v) = \phi_{r,4}(v) \ge 2^{r+1-\ell(v)} = 2^{r+1-\ell(\overline{v})}$. As a result, the lemma holds for any active vertex $\overline{v}$ on P_{r+1}, finishing the induction and giving the lemma. □

Proof of Lemma 3.9. Let $R := \log d + L$, where L is defined in Lemma 3.17. By Lemma 3.17, with good probability, $\ell(v) \le L$ for any vertex v in any of the first $O(\log n)$ rounds, and we shall condition on this happening. By Lemma 3.20, at the beginning of round R, if there is a path P_R of at least 4 active vertices, then for any of these vertices v, it must be $\phi_R(v) \ge 2^{R-\ell(v)} \ge 2^{R-L} \ge d$. So $\sum_{v \in P_R} \phi_r(v) \ge 4d > d+1$, contradicting with Lemma 3.16. Thus, any path P_R has at most 3 active vertices, which means $|P_{R,2}| \le 3$ by Definition 3.12. Therefore, by Lemma 3.14, the diameter of the graph at the end of round R is $O(R)$ with good probability. □

3.4.3 Remaining Proofs: Proof of Lemma 3.11. Based on the graph and any P_R at the beginning of round $R+1$, we need a (much simpler) path construction:

Definition 3.21. For any integer $r > R$, given path P_r with $|P_r| \ge 3$ at the beginning of round r, EXPAND-MAXLINK constructs P_{r+1} by the following:

(1) The ALTER in Step (1) replaces each vertex v on P_r by $v' := v.p$ to get path $P_{r,1}$. For any v' on $P_{r,1}$, let $\underline{v'}$ be on P_r such that $\underline{v'}.p = v'$.
(2) After Step (5), let $v' := P_{r,1}(1)$, if $\underline{v'}$ is a root at the end of round r and does not increase level during round r then: if the current graph contains edge $(v', P_{r,1}(3))$ then remove $P_{r,1}(2)$ to get path $P_{r,2}$.
(3) The ALTER in Step (6) replaces each vertex v on $P_{r,2}$ by $v.p$ to get path P_{r+1}.

For any vertex v on P_r that is replaced by v' in the first step, if v' is not removed in the second step, then let $\overline{v}$ be the vertex replacing v' in the third step, and call $\overline{v}$ the *corresponding vertex* of v in round $r+1$.

An analog of Lemma 3.13 immediately shows that P_r is a valid path for any $r \ge R+1$. The proof of Lemma 3.10 is simple enough without potential:

Proof of Lemma 3.10. By Lemma 3.9, at the beginning of round $R+1$, with good probability, any P_{R+1} has length $O(R)$. We shall condition on this happening and apply a union bound at the end of the proof. In any round $r > R$, for any path P_r with $|P_r| \ge 3$, consider the first vertex v' on $P_{r,1}$ (cf. Definition 3.21). If $\underline{v'}$ is a non-root or increases its level during round r, then by the first 3 paragraphs in the proof of Lemma 3.20, it must be $\ell(\overline{v'}) \ge \ell(\underline{v'}) + 1$. Otherwise, by Lemma 3.18, there is an edge between v' and the successor of its successor in the graph after Step (5), which means the successor of v' on $P_{r,1}$ is removed in the second step of Definition 3.21. Therefore, the number of vertices on P_{r+1} is one less than P_r if $\ell(\overline{v'}) = \ell(\underline{v'})$ as the level of a corresponding vertex cannot be lower. By Lemma 3.17, with good probability, the level of any vertex in any of the $O(\log n)$ rounds cannot be higher than L. As P_{R+1} has $O(R)$ vertices, in round $r = O(R) + L + R = O(R) \le O(\log n)$, the number of vertices on any P_r is at most 2. Therefore, the diameter of the graph after $O(R)$ rounds is at most 1 with good probability. □

Proof of Lemma 3.11. Now we show that after the diameter reaches 1, if the loop has not ended, then the loop must break in $2L + \log_{5/4} L$ rounds with good probability, i.e., the graph has diameter at most 1 and all trees are flat.

For any component, let u be a vertex in it with the maximal level and consider any (labeled) tree of this component. For any vertex v in this tree that is incident with an edge, since the diameter is at most 1, v must have an edge with u, which must be a root. So v updates its parent to a root with the maximal level after a MAXLINK, then any root must have the maximal level in its component since a root with a non-maximal level before the MAXLINK must have an edge to another tree (see the proof of correctness in the full version of this paper). Moreover, if v is a root, this can increase the maximal height among all trees in its component by 1.

Consider the tree with maximal height ξ in the labeled digraph after Step (1). By Lemmas 3.3 and 3.17, with good probability $\xi \le L$. The maximal level can increase by at most 1 in this round. If it is increased in Step (7), the maximal height is at most $\lceil \xi/2 \rceil + 1$ after the MAXLINK in the next round; otherwise, the maximal height is at most $\lceil (\xi+1)/2 \rceil \le \lceil \xi/2 \rceil + 1$. If $\xi \ge 4$, then the maximal height of any tree after Step (1) in the next round is at most $(4/5)\xi$ (the worst case is that a tree with height 5 gets shortcutted to height 3 in Step (6) and increases its height by 1 in the MAXLINK of Step (1) in the next round). Therefore, after $\log_{5/4} L$ rounds, the maximal height of any tree is at most 3.

Beyond this point, if any tree has height 1 after Step (1), then it must have height 1 at the end of the previous round since there is no incident edge on leaves after the ALTER in the previous round, thus the loop must have been ended by the break condition. Therefore, the maximal-height tree (with height 3 or 2) cannot increase its height beyond this point. Suppose there is a tree with height 3, then if the maximal level of vertices in this component does not change during the round, this tree cannot increase its height in the MAXLINK of Step (6) nor that of Step (1) in the next round, which means it has height at most 2 as we do a SHORTCUT in Step (6). So after L rounds, all trees have heights at most 2 after Step (1). After that, similarly, if the maximal level does not increase, all trees must be flat. Therefore, after L additional rounds, all trees are flat after Step (1). By the same argument, the loop must have been ended in the previous round. The lemmas follows immediately from $L = O(R)$ and Lemma 3.10. □

REFERENCES

[1] Miklós Ajtai. 1990. Approximate Counting with Uniform Constant-Depth Circuits. In *Advances In Computational Complexity Theory, Proceedings of a DIMACS Workshop, New Jersey, USA, December 3-7, 1990*. 1–20. https://doi.org/10.1090/dimacs/013/01

[2] Selim G. Akl. 1989. *Design and analysis of parallel algorithms*. Prentice Hall.

[3] Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. 2018. Parallel Graph Connectivity in Log Diameter Rounds. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*. 674–685.

[4] Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. 2019. Massively Parallel Algorithms for Finding Well-Connected Components in Sparse Graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*. 461–470. https://doi.org/10.1145/3293611.3331596

[5] Baruch Awerbuch and Yossi Shiloach. 1987. New Connectivity and MSF Algorithms for Shuffle-Exchange Network and PRAM. *IEEE Trans. Computers* 36, 10 (1987), 1258–1263.

[6] Paul Beame and Johan Håstad. 1989. Optimal bounds for decision problems on the CRCW PRAM. *J. ACM* 36, 3 (1989), 643–670. https://doi.org/10.1145/65950.65958

[7] Paul Beame, Paraschos Koutris, and Dan Suciu. 2017. Communication Steps for Parallel Query Processing. *J. ACM* 64, 6 (2017), 40:1–40:58.

[8] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, and Vahab S. Mirrokni. 2019. Near-Optimal Massively Parallel Graph Connectivity. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*. 1615–1636. https://doi.org/10.1109/FOCS.2019.00095

[9] Stephen A. Cook, Cynthia Dwork, and Rüdiger Reischuk. 1986. Upper and Lower Time Bounds for Parallel Random Access Machines without Simultaneous Writes. *SIAM J. Comput.* 15, 1 (1986), 87–97.

[10] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113. https://doi.org/10.1145/1327452.1327492

[11] Martin Dietzfelbinger, Miroslaw Kutylowski, and Rüdiger Reischuk. 1994. Exact Lower Time Bounds for Computing Boolean Functions on CREW PRAMs. *J. Comput. Syst. Sci.* 48, 2 (1994), 231–254. https://doi.org/10.1016/S0022-0000(05)80003-0

[12] François Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó (Eds.). ACM, 296–303. https://doi.org/10.1145/2608628.2608664

[13] Hillel Gazit. 1991. An Optimal Randomized Parallel Algorithm for Finding Connected Components in a Graph. *SIAM J. Comput.* 20, 6 (1991), 1046–1067. https://doi.org/10.1137/0220066

[14] Joseph Gil, Yossi Matias, and Uzi Vishkin. 1991. Towards a Theory of Nearly Constant Time Parallel Algorithms. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*. 698–710. https://doi.org/10.1109/SFCS.1991.185438

[15] Steve Goddard, Subodh Kumar, and Jan F. Prins. 1994. Connected components algorithms for mesh-connected parallel computers. In *Parallel Algorithms, Proceedings of a DIMACS Workshop, Brunswick, New Jersey, USA, October 17-18, 1994*. 43–58.

[16] Michael T. Goodrich. 1991. Using Approximation Algorithms to Design Parallel Algorithms that May Ignore Processor Allocation (Preliminary Version). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*. 711–722.

[17] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, Searching, and Simulation in the MapReduce Framework. In *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*. 374–383. https://doi.org/10.1007/978-3-642-25591-5_39

[18] John Greiner. 1994. A comparison of parallel algorithms for connected components. In *Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures, SPAA 1994*. ACM, 16–25.

[19] Shay Halperin and Uri Zwick. 1996. An Optimal Randomised Logarithmic Time Connectivity Algorithm for the EREW PRAM. *J. Comput. Syst. Sci.* 53, 3 (1996), 395–416.

[20] Shay Halperin and Uri Zwick. 2001. Optimal randomized EREW PRAM algorithms for finding spanning forests. *Journal of Algorithms* 39, 1 (2001), 1–46.

[21] Daniel S. Hirschberg, Ashok K. Chandra, and Dilip V. Sarwate. 1979. Computing Connected Components on Parallel Computers. *Commun. ACM* 22, 8 (1979), 461–464. https://doi.org/10.1145/359138.359141

[22] Tsan-Sheng Hsu, Vijaya Ramachandran, and Nathaniel Dean. 1997. Parallel implementation of algorithms for finding connected components in graphs. *Parallel Algorithms: Third DIMACS Implementation Challenge, October 17-19, 1994* 30 (1997), 20.

[23] Donald B. Johnson and Panagiotis Takis Metaxas. 1997. Connected Components in O (logˆ3/2 n) Parallel Time for the CREW PRAM. *J. Comput. Syst. Sci.* 54, 2 (1997), 227–242.

[24] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A Model of Computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*. 938–948. https://doi.org/10.1137/1.9781611973075.76

[25] S. Cliff Liu and Robert E. Tarjan. 2019. Simple Concurrent Labeling Algorithms for Connected Components. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*. 3:1–3:20.

[26] Gary L. Miller and John H. Reif. 1985. Parallel Tree Contraction and Its Application. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*. 478–489.

[27] John H Reif. 1984. *Optimal Parallel Algorithms for Graph Connectivity*. Technical Report. HARVARD UNIV CAMBRIDGE MA AIKEN COMPUTATION LAB.

[28] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. 2018. Shuffles and Circuits (On Lower Bounds for Modern Parallel Computation). *J. ACM* 65, 6 (2018), 41:1–41:24. https://doi.org/10.1145/3232536

[29] Yossi Shiloach and Uzi Vishkin. 1982. An O(log n) Parallel Connectivity Algorithm. *J. Algorithms* 3, 1 (1982), 57–67.

[30] Julian Shun, Laxman Dhulipala, and Guy E. Blelloch. 2014. A simple and practical linear-work parallel algorithm for connectivity. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '14, Prague, Czech Republic - June 23 - 25, 2014*. 143–153. https://doi.org/10.1145/2612669.2612692

[31] Stergios Stergiou, Dipen Rughwani, and Kostas Tsioutsiouliklis. 2018. Shortcutting Label Propagation for Distributed Connected Components. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*. 540–546.

[32] Robert E. Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1, 2 (1972), 146–160.

[33] Robert E. Tarjan and Jan van Leeuwen. 1984. Worst-case Analysis of Set Union Algorithms. *J. ACM* 31, 2 (1984), 245–281.

[34] Uzi Vishkin. 1983. Implementation of Simultaneous Memory Address Access in Models That Forbid It. *J. Algorithms* 4, 1 (1983), 45–50.