


Conditionally Optimal Parallel Coloring of Forests

Conference Paper**Author(s):**

[Grunau, Christoph](#) ; Latypov, Rustam; Maus, Yannic; Pai, Shreyas; Uitto, Jara

Publication date:

2023-10

Permanent link:

<https://doi.org/10.3929/ethz-b-000640121>

Rights / license:

[Creative Commons Attribution 4.0 International](#)

Originally published in:

Leibniz International Proceedings in Informatics (LIPIcs) 281, <https://doi.org/10.4230/LIPIcs.DISC.2023.23>

Conditionally Optimal Parallel Coloring of Forests

Christoph Grunau  

ETH Zürich, Switzerland

Rustam Latypov  

Aalto University, Finland

Yannic Maus  

TU Graz, Austria

Shreyas Pai  

Aalto University, Finland

Jara Uitto  

Aalto University, Finland

Abstract

We show the first conditionally optimal deterministic algorithm for 3-coloring forests in the low-space massively parallel computation (MPC) model. Our algorithm runs in $O(\log \log n)$ rounds and uses optimal global space. The best previous algorithm requires 4 colors [Ghaffari, Grunau, Jin, DISC'20] and is randomized, while our algorithm are inherently deterministic.

Our main technical contribution is an $O(\log \log n)$ -round algorithm to compute a partition of the forest into $O(\log n)$ ordered layers such that every node has at most two neighbors in the same or higher layers. Similar decompositions are often used in the area and we believe that this result is of independent interest. Our results also immediately yield conditionally optimal deterministic algorithms for maximal independent set and maximal matching for forests, matching the state of the art [Giliberti, Fischer, Grunau, SPAA'23]. In contrast to their solution, our algorithms are not based on derandomization, and are arguably simpler.

2012 ACM Subject Classification Theory of computation → Massively parallel algorithms

Keywords and phrases massively parallel computation, coloring, forests, optimal

Digital Object Identifier 10.4230/LIPIcs.DISC.2023.23

Related Version *Full Version*: <https://arxiv.org/abs/2308.00355>

Funding *Rustam Latypov*: Academy of Finland, Grant 334238.

Yannic Maus: Austrian Science Fund (FWF), Grant P36280-N.

Shreyas Pai: Academy of Finland, Grant 334238.

1 Introduction

A recent sequence of papers investigates fundamental symmetry-breaking problems such as coloring, maximal independent set and maximal matching on trees [9, 7, 32, 22, 26]. We conclude, simplify and unify this line of work by giving a conceptually simple algorithm for 3-coloring, maximal independent set and maximal matching. We solve the three problems in a unified way by computing a so-called H -decomposition (we discuss these in more detail in Section 1.2). Even though such decompositions are the natural tool for solving the aforementioned problems on trees, computing them efficiently in the MPC model remained outside the reach of previous techniques.

► **Theorem 1.** *There are deterministic $O(\log \log n)$ -round low-space MPC algorithms for 3-coloring, maximal matching and maximal independent set (MIS) on forests. These algorithms use $O(n)$ global space.*



© Christoph Grunau, Rustam Latypov, Yannic Maus, Shreyas Pai, and Jara Uitto;
licensed under Creative Commons License CC-BY 4.0

37th International Symposium on Distributed Computing (DISC 2023).

Editor: Rotem Oshman; Article No. 23; pp. 23:1–23:20



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The runtimes of our algorithms are conditionally optimal, conditioned on the 1 vs 2 cycle conjecture, at least if one restricts to so-called component stable algorithms [23, 15, 35, 41] (see Section 1.4 for a brief discussion about component-stability).

We note that algorithms for maximal matching and maximal independent set matching our guarantees are known from a very recent work [26]. However, their algorithms are quite complicated and technical, and use sophisticated derandomization techniques. Moreover, their techniques inherently cannot be used to color a tree with a small number of colors. Indeed, the 3-coloring problem is considered to be the hardest of the three problems, e.g., once such a coloring is known one can compute an MIS in $O(1)$ rounds. Additionally, a crucial property used in previous MPC algorithms for MIS and maximal matching is that any partial solution can be extended to a solution of the whole graph; a property that does not hold for 3-coloring. The best previous algorithm for coloring trees uses 4 colors and is randomized [22]. If one allows for randomization the single additional color makes the problem significantly easier by the following divide and conquer approach: if one partitions the tree into two parts by letting each node join one of the parts uniformly at random, the connected components induced by each part have logarithmic diameter. Once the diameter is small, one can use $O(\log \log n)$ MPC rounds to color each component independently with two colors in a brute force manner. In the next section we zoom out and present the bigger picture of our work.

1.1 MPC Model and Exponential Speed-Up Over LOCAL Algorithms

The Massively Parallel Computation (MPC) model [30] is a mathematical abstraction of modern frameworks of parallel computing such as Hadoop [43], Spark [44], MapReduce [18], and Dryad [29]. In the MPC model, we have M machines that communicate in all-to-all fashion, in synchronous rounds. In each round, every machine receives the messages sent in the previous round, performs (arbitrary) local computations, and is allowed to send messages to any other machine. Initially, an input graph of n nodes and m edges is arbitrarily distributed among the machines. At the end of the computation, each machine needs to know the output of each node it holds, e.g., their color in the vertex-coloring problem.

The MPC model is typically divided into 3 regimes according to the local space S . The *superlinear* and the *linear* regimes allow for $S = n^{1+\Omega(1)}$ and $S = \tilde{O}(n)$ words¹ of space (memory) per machine. A word is $O(\log n)$ bits and is enough to store a node or a machine identifier from a polynomial (in n) domain. The local space restricts the amount of data a machine initially holds and is allowed to send and receive per round. Both linear and superlinear regimes allow for very efficient algorithms because machines can get a “global view” of the graph in the sense that it can store information for each node of the graph [31, 20, 24]. However, the growing size of most real-world graphs makes it impossible to get such a global view on a single machine and hence research in recent years has focused on the most challenging *low-space* (or *sublinear*) regime with $S = n^\delta$, for some constant $\delta < 1$, where we cannot even store the whole neighborhood of a single node in a single machine. As each machine can only get a local view there are close connections to the LOCAL model of distributed computing that we further elaborate on below.

Furthermore, we focus on the most restricted case of *linear global space*, i.e., $S \cdot M = \Theta(n + m)$. Notice that $\Omega(n + m)$ words are *required* to store the input graph.

¹ The \tilde{O} notation hides polylogarithmic factors.

The LOCAL Model and Graph Exponentiation. The LOCAL model is a classic model of distributed message passing. Each node of an input graph hosts a processor and the nodes communicate along the edges of the graph in synchronous rounds. The local computation, local space, and message sizes are unbounded in this model. Most research in the LOCAL model has focused on symmetry breaking problems like graph colorings, MIS, and maximal matchings. For most of these classic problems $O(\log n)$ -round randomized algorithms are known [36, 1, 38] which can directly be translated to the MPC model. A major focus on recent and current research is to develop *sublogarithmic* MPC algorithms that beat the logarithmic baseline.

In fact, the strong connection between the models also shows up in faster algorithms, as almost all recent MPC algorithms for such problems are MPC-optimized implementations of algorithms that were originally developed for the LOCAL model. The main technique to obtain this speedup is the graph *exponentiation technique* [33]. It allows to gather the T -hop radius neighborhood of a node in $O(\log T)$ MPC rounds. So, as long as these neighborhoods fit the space constraints, after gathering them one can simulate T -round LOCAL algorithms locally to compute the output for each node. Furthermore, for component-stable algorithms, the connection also goes the other way around, that is, an $\Omega(T)$ lower bound on the round complexity in the LOCAL model implies an exponentially lower $\Omega(\log T)$ conditional lower bound in the MPC model. Thus, the holy grail is to obtain this exponential speedup over the LOCAL model. A central open problem in the area is to find an $O(\log \log n)$ round MPC algorithm for the classic MIS problem on general graphs, which enjoys a matching conditional $\Omega(\log \log n)$ -round lower bound.

Unfortunately, we are very far from answering this question. The current state of the art is an $\tilde{O}(\sqrt{\log \Delta} + \log \log \log n)$ -round randomized MPC algorithm [25]. We note that we take into account the new results on network decomposition, which reduce the dependency on n [42, 21]. This result is obtained by combining the graph exponentiation technique with *sparsification* methods [33, 25]. The exponentiation technique is used to simulate Ghaffari's $O(\log \Delta + \text{poly} \log \log n)$ -round MIS algorithm for the LOCAL model [19].

From a high-level perspective they break the LOCAL algorithm into $O(\sqrt{\log \Delta})$ phases each of length $T = O(\sqrt{\log \Delta})$. In the beginning of each phase, the graph is subsampled so that the maximum degree of any node is at most $2^{\sqrt{\log \Delta}}$. Then, we can gather the T -hop neighborhood of each node in $O(\log \log \Delta)$ MPC rounds and simulate T rounds of the LOCAL algorithm in a single MPC round. The main benefit of simulating (shorter) phases and subsampling to smaller degree graphs is to reduce the memory resources needed during the exponentiation technique. Unfortunately, this phase-based approach seems to hit a fundamental barrier at $\sqrt{\log \Delta}$ rounds, and it is unclear how to reduce memory usage without it. Due to little progress in improving on this result, recent research has focused on special graph classes such as trees and bounded arboricity graphs.

Symmetry Breaking on Trees and Bounded Arboricity Graphs. Studying low-space MPC algorithms for MIS on trees and forests has been fruitful. This line of work started with a randomized $O(\log^3 \log n)$ round low-space MPC algorithm for MIS and maximal matching on trees [9]. Later, the round complexity was first improved to $O(\log^2 \log n)$ [7] and finally to $O(\log \log n)$ [22], where both algorithms extend to low-arboricity graphs. The $O(\log \log n)$ algorithm is conditionally optimal, at least if one restricts oneself to component-stable algorithms. Finally, a recent work derandomized the $O(\log \log n)$ round algorithm using MPC specific derandomization techniques and thus obtained a deterministic $O(\log \log n)$ round MIS and Maximal Matching algorithm for trees and more generally low-arboricity graphs [26].

While especially the $O(\log \log n)$ round algorithms are quite technical and involved, all of the aforementioned previous algorithms rely on the same fundamental idea. Namely, to interleave graph exponentiation with the computation of partial solutions to rapidly decrease the maximum degree of the remaining graph. Unfortunately, it seems unlikely that such a rapid degree reduction is possible in general graphs; thus it seems that new approaches are necessary in order to get an $O(\log \log n)$ round algorithm for general graphs.

Also, their approach does not work for coloring a forest with a constant number of colors. The main reason is that they critically rely on the fact that any partial solution can be extended to a full solution, which is not the case for coloring a forest with a fixed number of colors.

1.2 Our Technical Contribution

We present a unified solution for 3-coloring, MIS, maximal matching that takes $O(\log \log n)$ rounds. The core technical contribution that unifies these is an efficient algorithm to compute *H-decompositions*.

► **Theorem 2.** *There is a deterministic $O(\log \log n)$ -rounds low-space MPC algorithm that computes a strict *H-decomposition* with $O(\log n)$ layers on forests in $O(n)$ global space.*

H-decompositions were introduced to the area of distributed computing by Barenboim and Elkin [5]. An *H-decomposition* (of a forest) partitions the vertices of the graph into layers such that every node has at most two neighbors² in higher or equal layers. For forests an *H-partition* with $O(\log n)$ layers always exist. In the LOCAL model, such an *H-decomposition* immediately implies an algorithm for 3-coloring in $O(\log n)$ rounds. Essentially, one can iterate through the layers in a reverse order and color all nodes in a layer while avoiding conflicts with the already colored neighbors in higher layers.

The novelty of our approach is not that we use such decompositions to compute a 3-coloring of a forest, in fact, this straightforward approach has made it into the classrooms of many graduate programs of universities, but in the way how we compute it. We detail on our solution in more detail in the nutshell, but the main take-away is as follows. We steer every machine to learn some parts of the graph (to large extent in an uncoordinated fashion) such that every machine can compute a partial *H-decomposition* locally, which we can later unify to a global decomposition. We are not aware of any other MPC algorithm for *H-decompositions* with a similar approach.

Balanced Exponentiation. In order to achieve exponential speedup, our algorithms rely on graph exponentiation. However, there are no known sparsification techniques that can cope with the memory resources that are needed for the classic graph exponentiation technique. Instead, we provide a self-contained exponentiation procedure whose memory overhead is very mild on forests. To explain our procedure, we first need to define a *subtree*. A subtree is a subgraph of a tree, such that if it is removed, the rest of the tree stays connected. A node is *important* if it is contained in a subtree of size $n^{\delta/8}$. We present the following result (the formal statement appears in the full version).

² There are generalizations to higher number of neighbors that are important when dealing with bounded arboricity graphs [37, 6].

Let $0 < k \leq n^{\delta/8}$ be a parameter. There is a deterministic low-space MPC algorithm that, given an n -node forest F , uses $O(\log k)$ rounds in which every important node $v \in F$ discovers its k -hop neighborhood in every direction of the graph, except for at most one.

Given a node $v \in F$, we refer to each of its neighbors $x \in N(v)$ as a *direction* with regard to v . Informally, what node v can discover in direction x is simply the subgraph of F that is connected to v via x , which is uniquely defined, since F is a forest.

This result above may be of independent interest and may be useful to design algorithms for other graph problems. We obtain it by extending the exponentiation technique of a recent work by [3]. Their work designs an exponentiation technique which (almost) equals ours in the special case when k equals the maximum diameter of a component of the forest. In their work it is used in an $O(\log \text{diam})$ -round algorithm to compute the connected components of a forest. Later it has also been used to solve certain dynamic programming tasks on tree-structured data [28], also in time that is logarithmic in the diameter. In the full version we present a more detailed discussion on the similarities and the difference between the exponentiation result in this work and the one in their work, and why our result requires a different analysis. The main benefit of our result is that a flexible choice of k allows the runtime and space to be small, if the required “view” for the nodes is small, which we heavily utilize in our algorithm to compute H -decompositions.

1.3 Our Method in a Nutshell

As mentioned in the previous section, our key technical contribution is to compute a so-called H -decomposition of the input forest F . In particular, the goal is to compute a partition $V(F) = V_1 \sqcup V_2 \sqcup \dots \sqcup V_L$ of the vertices into $L = O(\log n)$ layers such that each node in V_i has at most two neighbors in $\bigcup_{j \geq i} V_j$. There exists a simple peeling algorithm which computes such a partition; iteratively peel off all nodes of degree at most 2 and define V_i as the set of nodes that got peeled off in the i -th iteration. A simple calculation shows that at least half of all the remaining nodes get peeled off in each iteration, and hence we get a decomposition into $O(\log n)$ layers. Moreover, one can determine the iteration in which a node gets peeled off by only looking at its $O(\log n)$ -hop neighborhood. Thus, if we could compute for a given node its entire $O(\log n)$ -neighborhood and store it in a single machine, then we could locally determine the layer of that node with no further communication.

One way to compute the $O(\log n)$ -neighborhood of each node in the MPC model is the well-known graph exponentiation technique. Generally speaking, graph exponentiation allows to learn the 2^i -hop neighborhood of each node in $O(i)$ MPC rounds. Thus, we could in principle hope to learn the $O(\log n)$ -hop neighborhood of each node in just $O(\log \log n)$ rounds. However, one obviously necessary precondition of the graph exponentiation technique is that the $O(\log n)$ -hop neighborhood of each node has size n^δ , as otherwise we cannot possibly store the neighborhood in one machine. This is quite a limiting condition. If the input is for example a star, even the two-hop neighborhood of each node contains $\Omega(n)$ vertices. Moreover, even if each local neighborhood would fit into one machine, the global space required to store all the neighborhoods might still be prohibitively large, especially if one aims for near-linear global space.

Thus, we cannot use the vanilla graph exponentiation technique. Instead, we use the balanced graph exponentiation technique for forests mentioned in the previous section. The output guarantee of the balanced exponentiation algorithm, running in $O(\log \log n)$ rounds, weakens the guarantee that each node sees its $O(\log n)$ -hop in two ways. First, it only gives a guarantee for nodes that are contained in a sufficiently small subtree, namely of size at most n^δ . Second, for each node v in a small subtree, it computes all nodes of distance $O(\log n)$, except for nodes in one direction.

We start by briefly discussing how one can deal with the first shortcoming. If one iteratively removes all nodes that are contained in a subtree of size at most x from F and all nodes of degree at most 2, then all nodes are removed within $O(\log_x(n))$ iterations. This fact was used in similar forms in previous results and for completeness we give a standalone proof (see Lemma 7). Thus, if we repeatedly assign nodes in subtrees of size at most n^δ and nodes of degree at most 2 to one of $O(\log n)$ layers, then after $O(1/\delta)$ iterations, we assigned each node to one of $O((1/\delta) \log n)$ layers. Thus, it intuitively suffices to focus on nodes in subtrees of size at most n^δ . Section 4 gives a formal treatment of this argument.

The more severe difficulty stems from the fact that there might not be a single node in the forest for which we have stored its entire $O(\log n)$ -neighborhood in one machine. This makes it impossible to locally determine the layer of each node, or even a single one, assigned by the simple peeling process described in the beginning. Instead, each node v locally simulates a conservative variant of the peeling algorithm described above; in each iteration not all the nodes of degree at most 2 are removed, but only those that v has stored in its machine. Note that if v has strictly more than 2 neighbors not stored in its machine, then the conservative peeling algorithm would never peel off v . Moreover, even if v would eventually be peeled off, then there is no guarantee that it happens within the first $O(\log n)$ iterations. However, the fact that v has stored all the nodes in its $O(\log n)$ -hop neighborhood except for nodes in one direction in its machine suffices to show that v gets peeled off within the first $O(\log n)$ iterations (see Lemma 21). Thus, each node v locally computes a layering $V^v = V_1^v \sqcup \dots \sqcup V_L^v$ for some $L = O(\log n)$ such that $v \in V^v$ and each node in V_i^v has at most two neighbors contained in $(\bigcup_{j \geq i} V_j^v) \cup (V(F) \setminus V^v)$. As some nodes might not be assigned to any layer, we refer to such a decomposition as a partial H -decomposition. Note that a node might get assigned to different layers from different nodes. Fortunately, this is not a problem because of the following nice structural property about (partial) H -decompositions: if we are given multiple (partial) H -decompositions, then we can get another (partial) H -decomposition by assigning each node to the smallest layer assigned by any of the H -decompositions. This structural observation allows us to combine the different locally computed (partial) H -decompositions into a single partial H -decomposition where each node in a small subtree is assigned to one of the $O(\log n)$ layers.

Rooted vs. Unrooted Forests. Our results are for unrooted forests, which are indeed more difficult than rooted forests. In fact, the fastest known MPC algorithm to root a forest takes $O(\log \text{diam})$ rounds (and at least on general forests this runtime is conditionally tight) [3]; so rooting the forest does not fit our time budget of $O(\log \log n)$ rounds. Many steps of our algorithm would simplify (or maybe even allow for alternative solutions) if the forest was rooted. For example, in a directed forests, we would not need our balanced exponentiation procedure. One can show that nodes can just exponentiate towards their children until the local memory is full without breaking any global memory bounds. However we would still need our combinatorial algorithm for creating a (global) H -partition. Observe that [3] also contains a $O(\log \text{diam})$ -round 2-coloring algorithm for rooted constant-degree forests, which can be generalized to rooted unbounded-degree forests.

1.4 Further Related Work

Component Stability. Roughly speaking, an MPC algorithm is component-stable, if the outputs of nodes in different components are independent of each other. Low-space component-stable MPC algorithms are closely connected to algorithms in the LOCAL model and this connection was used to lift (unconditional) lower bounds from the LOCAL model into

conditional lower bounds in the MPC model [23]. Under the 1 vs 2 cycle conjecture, this technique turns an $\Omega(T)$ -round lower bound in LOCAL into an $\Omega(\log T)$ lower bound in low-space MPC. This approach was used to establish, among others, $\Omega(\log \log n)$ randomized lower bounds for MIS and maximal matching. Later, the technique was extended to deterministic component-stable algorithms as well [15]. While the assumption of component-stability might seem very natural to MPC algorithms, it is known that component-instability can help. For example, any component-stable algorithm for finding an independent set of size $\Omega(n/\Delta)$ requires $\Omega(\log \log^* n)$ rounds, while there is an $O(1)$ -round algorithm that is not component-stable [15].

log(diam) Algorithms on Forests. There are surprisingly few works with a strict $\log(\text{diam})$ runtime for any graph families in any MPC regimes, where diam refers to the diameter. To our knowledge, the only existing ones are low-space algorithms for forests [3, 28]. The authors of [3] show that connectivity, rooting, and all LCL (locally checkable labeling) problems can be solved on forests in $O(\log \text{diam})$ using optimal global space $O(n)$. The authors of [28] build on top of the works of [3] by introducing a framework to solve dynamic programming tasks and optimization problems, all in time $\log(\text{diam})$ and global space $O(n)$. We note that given a double-logarithmic dependency on n , the connectivity problem can be solved on general graphs (even deterministically) in $O(\log \text{diam} + \log \log n)$ time using linear total space [8, 13]. Also, the 1 vs 2 cycle conjecture directly rules out an $o(\log \text{diam})$ for connectivity.

Symmetry-Breaking on General Graphs. In general graphs, $(\Delta + 1)$ -vertex coloring is an intensively studied symmetry breaking problem, where Δ is the maximum degree of the graph. A series of works [39, 4, 40, 11] for Congested Clique model which is similar to the MPC model with linear local memory has culminated in a deterministic $O(1)$ -round algorithm [17].

In the low-space MPC model, the first algorithm for the problem was randomized and used $O(\log \log \log n)$ rounds with almost linear $\tilde{O}(m)$ global space [11]³. By derandomizing the classic logarithmic-time algorithms, one can obtain an $O(\log \Delta + \log \log n)$ -round algorithm for $(\Delta + 1)$ -coloring, MIS, and maximal matching [14, 17]. For coloring, this was improved to $O(\log \log \log n)$ through derandomizing a tailor-made algorithm [16]. The deterministic algorithms require $n^{1+\Omega(1)}$ global space.

1.5 Outline

We define strict H -decompositions in Section 3, and then we show how to compute them in $O(\log \log n)$ low-space MPC rounds using $O(n \cdot \text{poly}(\log n))$ global space in Section 4. The key subroutine for the algorithm in Section 4 is discussed in Appendix A. In Section 5 we show how to use these decompositions to compute a coloring, MIS, and matching. In Appendix B we show how to reduce the global memory usage of our algorithms from $O(n \cdot \text{poly}(\log n))$ to $O(n)$. Due to space constraints, the balanced exponentiation procedure and the missing proofs of Section 4 appear in the full version. Many of the proofs are omitted due to the page limit and deferred to the full version.

³ The $O(\sqrt{\log \log n})$ runtime stated in the paper is automatically improved to $O(\log \log \log n)$ through developments in network decomposition [42].

2 Preliminaries and Notation

The input graph is an undirected, finite, simple forest $F = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges such that $E \subseteq [V]^2$ and $V \cap E = \emptyset$. For a subset $S \subseteq V$, we use $G[S]$ to denote the subgraph of G induced by nodes in S .

Let $\deg_F(v)$ denote the degree of a node v in F and let Δ denote the maximum degree of F . For node set $S \subseteq V(F)$ and a node $v \in S$ we write $\deg_S(v)$ for the degree of v in $F[S]$. The distance $d_F(v, u)$ between two vertices v, u in F is the number of edges in the shortest $v - u$ path in F ; if no such path exists, we set $d_F(v, u) := \infty$. Sometimes we simply write $\deg(v)$ and $d(v, u)$ if it is clear from context that we refer to the degree and distance in graph F . The greatest distance between any two vertices in F is the diameter of F , denoted by $\text{diam}(F)$.

For each node v and for every $k \in \mathbb{N}$, we denote the k -hop (or k -radius) neighborhood of v as $N^k(v) = \{u \in V : d(v, u) \leq k\}$. Set $N^1(v)$ is simply the set of neighbors of v , to which often refer to as $N(v)$. We often consider sets of nodes S from which we need to remove a single node u . Hence, we use the notation $S \setminus u$ as a shorthand for $S \setminus \{u\}$.

3 Strict H -decompositions

We begin with the formal definition of an H -decomposition, that is, a partition of the graph into layers such that every node has at most two neighbors in higher or equal layers. We also extend the definition to the setting where some nodes remain without a layer.

► **Definition 3** ((Partial) H -Decomposition). *Let F be a forest and $\text{layer} : V(F) \mapsto \mathbb{N} \cup \{\infty\}$. For $i \in \mathbb{N} \cup \{\infty\}$ define $V_i = \{v \in V(F) \mid \text{layer}(v) = i\}$, $V_{\geq i} = \bigcup_{j \geq i} V_j$.*

We say that layer is a partial H -decomposition if $\deg_{V_{\geq i}}(v) \leq 2$ holds for every v with $\text{layer}(v) = i$. We speak of an H -decomposition if $V_\infty = \emptyset$, and $L = \max\{\text{layer}(v) \mid v \in V(F), \text{layer}(v) \neq \infty\}$ is the length of the decomposition.

We also refer to the V_i 's as the *layers* of the (partial) H -decomposition.

Why 3-coloring and strict H -decompositions? H -decompositions were introduced to the area of distributed computing by Barenboim and Elkin [5]. Nowadays, they are a frequent tool in the area and by increasing the degree bound 2 to $(2 + \varepsilon)a$ the concept also extends to graphs with arboricity at most a (this is the original setting considered in [5]). More generally, it can be shown that H -decompositions with $O(\log n)$ layers exist. In the LOCAL model, an H -decomposition of a tree with $O(\log n)$ layers can be computed by iteratively removing nodes of degree 1 (rake) and nodes of degree 2 (compress).

In the LOCAL model, one can 3-color any graph with a given H -decomposition as in Definition 3 with L layers in $O(L + \log^* n)$ rounds. Each layer induces a graph with maximum degree 2. First, use Linial's algorithm to color each layer in parallel with $C = O(1)$ colors. This coloring may contain lots of monochromatic edges between different layers and is only used as a schedule to compute the final 3-coloring. In order to compute that final coloring, iterate through the layers in a decreasing order, and in each layer iterate through the C colors. When processing one of the C color classes, every node picks one color in $\{1, 2, 3\}$ not used by any of its already colored neighbors (at most two).

Optimally, we would like to use the above LOCAL model algorithm as the base for our exponentially faster MPC algorithm. However, even if we were given an H -decomposition for free it is non-trivial to actually use it for 3-coloring a graph if the runtime is restricted

to $O(\log \log n)$ rounds and you only allow for a polylogarithmic memory overhead. Going through the layers in some sequential manner would be way too slow as it would require logarithmically rounds. Still, as the LOCAL algorithm has locality $T = \Theta(\log n)$ the output of a node may depend on the topology in logarithmic distance. In order to achieve a fast MPC algorithm, we want the nodes to use the graph exponentiation technique to learn the part G_v of their T -hop neighborhood that is relevant to determine their output in $O(\log T) = O(\log \log n)$ rounds. We refer to G_v as the predecessor graph of node v . The challenge with the standard H -decomposition as given by Definition 3 is that, even though $G_v \subseteq V_{\geq \text{layer}(v)}$, it may be of size $\Theta(n)$. Hence, even if every node could learn its predecessor graph and store it in its local memory S_v (formally, the memory of every node is stored on some machine), the global space bound can only be upper bounded by $\sum_{v \in V} |G_v| = O(n^2)$, drastically, violating the desired near-linear bound. In order to circumvent this issue we introduce the concept of a strict H -decomposition which is optimized for its usage in the MPC model. The bottom line of this decomposition is that besides the properties of a classic H -decomposition, we also have a set V^{pivot} . The set V^{pivot} induces a graph with maximum degree 2 and hence can be colored with 3 colors in $O(\log^* n)$ rounds with Linial's algorithm [35]. The main gain compared to the classic H -decomposition is, that once we have colored the nodes in V^{pivot} , we can show that the predecessor graph of every node $v \in V \setminus V^{\text{pivot}}$ is of logarithmic size (when considering the same LOCAL model algorithm that colors these nodes layer by layer). Hence, each node can learn its predecessor graph in $O(\log \log n)$ rounds without violating global space constraints. We next present the definition of a strict H -decomposition.

► **Definition 4** ((Partial) Strict H -Decomposition). *Let F be a forest and $\text{layer}: V(F) \mapsto \mathbb{N} \cup \{\infty\}$ be a function. We define $V_{< \infty} = \{v \in V(F) \mid \text{layer}(v) < \infty\}$ and*

$$V^{\text{pivot}} := \{v \in V_{< \infty} \mid \text{layer}(v) \geq \text{layer}(w) \text{ for every } w \in N_F(v)\}.$$

We refer to a (partial) H -decomposition layer as strict if for every $v \in V_{< \infty} \setminus V^{\text{pivot}}$, it holds that

$$|\{w \in N_F(v) \setminus V^{\text{pivot}} \mid \text{layer}(w) = \text{layer}(v)\} \cup \{w \in N_F(v) \mid \text{layer}(w) > \text{layer}(v)\}| \leq 1. \quad (1)$$

That is, the total number of non-pivot neighbors with the same layer and of neighbors with a strictly higher layer is at most 1.

There is some similarity between the definition of a strict H -decomposition and the H -decompositions used in the theory of so called locally checkable labelings [12, 10, 2]. These decompositions iteratively layer degree 1 nodes and paths of length at least ℓ . Our strict H -decomposition is similar to the case when we remove paths of length at least $\ell = 3$ (see Lemma 10). The following lemma is one of the most crucial structural properties of partial H -decompositions that we exploit in the core of our algorithm (see Appendix A).

► **Lemma 5** (Partial Strict H -Decomposition, Closure under taking minimums). *Let F be a forest and $\text{layer}_1, \text{layer}_2: V(F) \rightarrow \mathbb{N} \cup \{\infty\}$ be two partial strict H -decompositions. Let $\text{layer}: V(F) \rightarrow \mathbb{N} \cup \{\infty\}$ with*

$$\text{layer}(v) = \min(\text{layer}_1(v), \text{layer}_2(v))$$

for every $v \in V(F)$. Then, layer is also a partial strict H -decomposition.

23:10 Conditionally Optimal Parallel Coloring of Forests

From a high level point of view, Lemma 5 says that we can independently compute two partial strict H -decompositions, and even though they might contain conflicting layer assignments for certain nodes, we can obtain a unified decomposition, by assigning each node to the smaller layer of the two choices. In fact, this insight also generalizes to more than two (possibly conflicting) decompositions. At the core of our procedure in Appendix A, many nodes (independently) learn large parts of the graph. Then, every node computes a partial decomposition on the parts that it has learned, and in a second step all these partial decompositions are combined, where each node takes the minimum layer that it got assigned in any of the decompositions. Taking the minimum is a very efficient procedure in the MPC model and only requires constant time. The remaining difficulty in Appendix A is to show that nodes learn large enough parts in the graph in order to make very fast global progress, that is, we show that the unified decomposition assigns a layer to a large fraction of the nodes.

4 Strict H -decomposition in MPC

In this section, we present our $O(\log \log n)$ -round MPC algorithm for computing a strict H -decomposition. However, the hardest part of that algorithm, that is, assigning each node that is contained in a small subtree (see definition below) to a layer is deferred to Appendix A. The algorithm in this section uses $n \cdot \text{poly} \log n$ global space. In Appendix B we explain how to extend the algorithm to optimal space.

High Level Overview. For the sake of this high level overview let us first assume that we compute an H -decomposition with $O(\log n)$ layers that may not be strict. Similar to the classic rake & compress algorithm, our algorithm iteratively assigns nodes to layers. After assigning a node to some layer we *remove* it from the graph and continue on the remaining graph, which may actually become disconnected and turn into a forest. In order to present the details of the high level intuition we require the definition of a subtree which is central to our whole approach.

► **Definition 6 (Subtree).** *Let T be a tree. A subtree $T' \subseteq T$ is a connected induced subgraph of T such that $T \setminus T'$ contains at most one component.*

A subtree $T' \subseteq F$ of a forest F is a connected induced subgraph of F such that the number of components of $F \setminus T'$ is not larger than the number of connected components of F .

The definition of a subtree is best understood in a rooted tree, where the subtree rooted at a node v is formed by all its descendants.

In order to assign a layer to all nodes of the graph, we iterate the following two steps until all nodes have received a layer:

1. Assigns a layer to each node contained in a *small subtree* of size $\leq n^{\delta/10}$ ($\text{SUBTREERC}(F)$),
2. Assign a layer to each node of degree ≤ 2 in the remaining graph.

This process can be seen as a generalization of the classic rake and compress procedure, in which one iteratively removes leaves, i.e., subtrees of size 1, and nodes of degree 2. The rake and compress procedure requires $O(\log n)$ iterations to *remove* all nodes of the graph. Our generalized process requires $O(1/\delta) = O(1)$ in order to assign a layer to every node of the graph (see Lemma 7 for $x = n^{\delta/10}$). Note that the lemma statement considers a slightly different process than the one presented in this overview; the difference lies in the fact that we actually want to compute a strict H -decomposition. However, a similar lemma holds for the process of this overview. The main contribution and the main difficulty of

our work lies in the procedure $\text{SUBTREERC}(F)$ as nodes do not know whether they are contained in a small subtree, but still these subtrees can have diameter up to $n^{\delta/10}$, so conditioned on the 1 vs 2 cycle conjecture it is impossible that a single node can learn the whole subtree in $O(\log \log n)$ rounds (we don't prove this formally, but it's very unlikely that such a result holds without breaking the conjecture). We explain the details of the procedure $\text{SUBTREERC}(F)$ in Appendix A.

We continue with our generalized rake and compress statement that shows that a constant number of iterations of the aforementioned process suffice. As we want to compute a strict H -decomposition (see Definition 4), we need to slightly modify Step 2 of the above outline, for which we require further definitions; the details of why $\text{SUBTREERC}(F)$ returns layers that induce a strict H -decomposition are presented in Appendix A.

A path in a graph is a *degree-2 path* if all of its nodes, including its endpoints have degree 2. The *length of a path* is the number of nodes in the path, e.g., a single node is a path of length 1.

The following lemma is easiest to be understood when setting $x = \ell = 1$ where the process (almost) equals the classic rake & compress process – in fact it consists of a rake step, a compress step, and another rake step – and the theorem shows that it removes $1/3$ of the nodes ($2/3$ of the nodes remain in the graph).

► **Lemma 7** (Generalized rake and compress). *Let $x, \ell \in \mathbb{Z}$. Consider a process on a tree T that consists of the following steps:*

1. *Remove (at least) all subtrees of size $\leq x$ from T , resulting in T_1 ,*
2. *Remove (at least) all nodes contained in a degree-2 path of length at least ℓ from T_1 , resulting in T_2 ,*
3. *Remove (at least) all nodes with degree ≤ 1 from T_2 .*

The number of nodes remaining is at most a $1/(1 + (x + 1)/2\ell) = O(\ell/x)$ fraction of the nodes from T . The degrees of nodes in Step 2) and 3) of the process are with respect to the graph induced by remaining nodes at the respective step.

We now state a lemma for a key subroutine that we will use as black box in this section and dedicate Appendix A to designing an algorithm that proves the lemma.

► **Lemma 8** (SUBTREERC). *Let F be a forest on n vertices. There exists a deterministic MPC algorithm SUBTREERC with $O(n^\delta)$ local space, $0 < \delta < 1$, and $\tilde{O}(n)$ global space which takes F as input and computes in $O(\log \log n)$ rounds a partial strict H -decomposition layer: $V(F) \mapsto [\lceil \log(|V(F)| + 1) \rceil] \cup \{\infty\}$ such that $\text{layer}(v) < \infty$ for every node $v \in V(F)$ contained in a subtree of size $n^{\delta/10}$.*

Our MPC algorithm for computing strict H -decomposition appears in Algorithm 1. We will now prove the correctness and progress guarantees of our algorithm.

► **Lemma 9.** *At the end of each iteration i , we have that layer is a partial strict H -decomposition with at most $(i + 1) \cdot \text{offset}$ layers.*

► **Lemma 10.** *In iteration i , Algorithm 1 correspond to a generalized rake and compress step with $x = n^{\delta/10}$ and $\ell = 3$.*

► **Corollary 11.** *In iteration i , Algorithm 1 correspond to a generalized rake and compress step with $x = 1$ and $\ell = 3$.*

■ **Algorithm 1** Strict H -decomposition.

```

1: Throughout  $V_\infty = \{v \in F \mid \text{layer}(v) = \infty\}$  denotes the set of nodes whose layer equals
    $\infty$ .
2: function STRICTHDECOMP(Forest  $F$ )
3:   Initialize:  $\text{layer}(v) = \infty$  for all  $v \in V(F)$ ;  $\text{offset} \leftarrow \lceil \log(|V(F)| + 1) \rceil + 1$ 
4:   for  $i = 1, 2, \dots, \lceil 10/\delta \rceil$  do
5:      $F_i \leftarrow F[V_\infty]$ 
6:      $\text{layer} \leftarrow i \cdot \text{offset} + \text{SUBTREERC}(F_i, x = n^{\delta/10})$ 
7:     Let  $V_i^{\text{pivot}} \leftarrow \{v \in V_\infty \mid d_{V_\infty}(v) \leq 2, d_{V_\infty}(w) \leq 2 \text{ for all } w \in N(v)\}$ 
8:      $\text{layer}(v) \leftarrow (i + 1) \cdot \text{offset}$  for every node  $v \in V_i^{\text{pivot}}$ 
9:      $\text{layer}(v) \leftarrow (i + 1) \cdot \text{offset}$  for every node  $v \in V_\infty$  with  $\leq 1$  in  $V_\infty$ 
10:  return  $\text{layer}$ 

```

► **Theorem 12.** *Algorithm STRICTHDECOMP(F) (Algorithm 1) applied to some forest F computes a strict H -decomposition of F with $O(\log n)$ layers, uses $O(\log \log n)$ low-space MPC rounds and $\tilde{O}(n)$ global space.*

Proof. By Lemmas 7 and 10, in each iteration, the number of nodes in the forest shrinks by a factor of $O(n^{\delta/10})$. Therefore, after $O(1/\delta)$ iterations of the for loop, the number of nodes with layer ∞ will be zero.

By Lemma 9, in an iteration i , we produce a partial strict H -decomposition with at most $(i + 1) \cdot \text{offset}$ layers and in the next iterations $j > i$, we compute a partial strict H -decomposition of the nodes that received layer ∞ (V_∞) in iteration i . The offset value ensures that the nodes in V_∞ get a higher layer than the nodes in $V \setminus V_\infty$. After $i = O(1/\delta)$ iterations, each node has a layer at most $O(\log n)$ since $(i + 1) \cdot \text{offset} = O(\log n)$, and hence we produce a valid strict H -decomposition.

Lemma 8 ensures that implementing each iteration takes $O(\log \log n)$ low-space MPC rounds and $\tilde{O}(n)$ global space. The theorem follows because there are just $O(1/\delta)$ iterations. ◀

5 Coloring, MIS, and Matching

The following theorem is proven at the end of the section.

► **Theorem 13.** *There is a deterministic $O(\log \log n)$ round algorithm for 3-coloring trees in the low-space MPC model using $\tilde{O}(n)$ words of global space.*

For an input tree F , consider having a strict H -decomposition $\text{layer} : V(F) \rightarrow \mathbb{N}$ described in Definition 4, which we get from Algorithm 1 in $O(\log \log n)$ rounds and $\tilde{O}(n)$ words of global space. We first color the subgraph induced by the nodes in V^{pivot} . Recall that V^{pivot} is the set of nodes that have no neighbor with a higher layer.

Coloring the Pivot Nodes. The subgraph $F[V^{\text{pivot}}]$ has maximum degree 2 each node $v \in V^{\text{pivot}}$ has at most two neighbors in V^{pivot} with the same layer, and no neighbors with higher layer. In order to color $F[V^{\text{pivot}}]$, we first run Linial's $O(\Delta^2)$ -coloring algorithm [34], which requires $O(\log^* n)$ rounds. Since $\Delta(F[V^{\text{pivot}}]) \leq 2$, Linial's algorithm results in an $O(1)$ -coloring which we can convert to a 3-coloring by performing the following: In each round, all nodes with the highest color among their neighbors in V^{pivot} recolor themselves with

the smallest color such that a proper coloring is preserved. Clearly, one color is eliminated in each round and since each node v has at most 2 neighbors in $F[V^{pivot}]$, we achieve a 3-coloring of $F[V^{pivot}]$ in a constant number of rounds.

Coloring the Remaining Nodes. We will now compute a 3-coloring of the nodes in $V \setminus V^{pivot}$. We first orient all edges $e = \{u, v\}$ with $u, v \in V \setminus V^{pivot}$ from u to v if $\text{layer}(u) < \text{layer}(v)$ and arbitrarily if $\text{layer}(u) = \text{layer}(v)$. The following lemma will help us to ensure that we do not create conflicts with the 3-coloring computed on V^{pivot} .

► **Lemma 14.** *Each node in $v \in V \setminus V^{pivot}$ has at most two forbidden colors. If v has an outgoing edge, then it can have at most one forbidden color.*

Proof. Each node in $v \in V \setminus V^{pivot}$ has at most two neighbors in V^{pivot} . This is because nodes in V^{pivot} do not have neighbors in higher layer, so v can only have neighbors in V^{pivot} at the same or higher layer. By Definition 4, v can have at most two such neighbors.

Nodes v with one outgoing edge can have at most one neighbor in V^{pivot} , as otherwise v has three neighbors with same or higher layer, and Definition 4 is violated. So if v has an outgoing edge, it can have at most one forbidden color. ◀

In what follows, each node $v \in V \setminus V^{pivot}$ will remember its at most two forbidden colors due to neighbors in V^{pivot} . Definition 4 also guarantees that all nodes in $V \setminus V^{pivot}$ will have at most one outgoing edge. So the nodes $w \in V \setminus V^{pivot}$ with no outgoing edge pick an arbitrary color that is not forbidden as their final color.

In order to properly color the nodes with exactly one outgoing edge, consider the following centralized procedure: Color the nodes one by one in a greedy manner starting from the highest layer and with an arbitrary order within one layer. Here, greedy means, that a node picks the smallest color that is not forbidden and not used by any of its already colored neighbors. This process computes a proper 3-coloring as each node will have one color used by the neighbor along its outgoing edge, and at most one forbidden color. The output of a node $v \in V \setminus V^{pivot}$ in this centralized procedure only depends on the *directed path of v* obtained by following outgoing edges starting at v . In the following lemma we show that this directed path cannot be too long.

► **Lemma 15.** *The directed path of a node $v \in V \setminus V^{pivot}$ obtained by following outgoing edges starting at v has length at most $O(\log n)$.*

Proof. Consider a directed edge (u, w) in the directed path of v . If $\text{layer}(u) = \text{layer}(w)$, then w cannot have an outgoing edge as it will have two neighbors in $V \setminus V^{pivot}$ with same or higher layer, violating Definition 4. In other words, if $\text{layer}(u) = \text{layer}(w)$, then the directed path of v ends at w .

Therefore, if we go along the directed path, the layer of the nodes either strictly increases or the path does not continue. Since there are $O(\log n)$ layers in the H -decomposition, the length of a directed path is at most $O(\log n)$. ◀

In our MPC algorithm, the idea is for each node to learn its $O(\log n)$ length directed path by performing graph exponentiation only along the directed edges. Since all nodes with no outgoing edges are already colored with their final color, consider performing the following MPC algorithm only for nodes with one outgoing edge: Each node computes its final color after gathering its directed path by performing $O(\log \log n)$ graph exponentiation steps along directed edges.

23:14 Conditionally Optimal Parallel Coloring of Forests

Proof of Theorem 13. Correctness follows from the fact that each node can recolor itself with its final color when seeing its whole directed path. The runtime follows from the fact that we only perform $O(\log \log n)$ graph exponentiation steps and color the directed paths.

Let us analyze the space usage of our algorithm. Since the length of a directed path stored by each node during the algorithm is at most $O(\log n)$, we do not violate global space. Note that the sequential coloring of frozen layers does not require additional space. Notice that even though each node v is the source of at most one request, multiple nodes may send a request to v . Hence, during graph exponentiation, node v may have to communicate with a large number of nodes in lower layers. To mitigate this issue, we perform a load balancing process by sorting all the at most n requests by the ID of their destination. This can be done deterministically in $O(1)$ rounds. Now, all the requests with destination v lie in consecutive machines, and therefore, we can broadcast the response of v to all these machines in $O(1)$ rounds by creating a constant depth broadcast tree on these machines. Therefore, each step of graph exponentiation can be done in $O(1)$ rounds, which leads to an overall running time of $O(\log \log n)$ rounds. ◀

MIS and Maximal Matching

The maximal independent set and maximal matching algorithms follow from Theorem 13.

► **Theorem 16.** *There is a deterministic $O(\log \log n)$ round MIS algorithm for trees in the low-space MPC model using $\tilde{O}(n)$ words of global space.*

Proof. By Theorem 13, we can color the tree with 3 colors. For all colors i , perform the following. Nodes colored i add themselves to the independent set, and all nodes adjacent to nodes colored i remove themselves from the graph. Clearly this results in a maximal independent set in $O(1)$ rounds and the space requirements are satisfied. ◀

► **Theorem 17.** *There is a deterministic $O(\log \log n)$ round maximal matching algorithm for trees in the low-space MPC model using $\tilde{O}(n)$ words of global space.*

Proof. By Theorem 13, we can color the tree with 3 colors using a H -decomposition. Recall that in the decomposition, each node v with $\text{layer}(v) = i$ has at most two neighbors with layer at least i . Let us define the parent nodes of v . We orient an edge $\{u, v\}$ from v to u if (i) u belongs to a strictly higher layer than v or (ii) u belongs to the same layer and has a higher ID. For all colors i , perform the following. Node v colored i proposes to its highest ID outgoing neighbor u , and u accepts the proposal of the highest ID proposer. If u accepts v 's proposal in which case the edge $\{u, v\}$ joins the matching. If u rejects v 's proposal, it means that u is matched with some other node and then we repeat the same procedure with v 's other possible out-neighbor. Note that when a node joins the matching, it prevents all other incident edges from joining the matching. As a result, all nodes colored i have either joined the matching or they have no out-going edges. After iterating through all color classes, all nodes have either joined the matching or they have no incident edges, implying that all their original neighbors belong to the matching. This results in a maximal matching in $O(1)$ rounds and the space requirements are satisfied. ◀

References

- 1 Noga Alon, László Babai, and Alon Itai. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *Journal of Algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.
- 2 Alkida Balliu, Keren Censor-Hillel, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Locally Checkable Labelings with Small Messages. In *the Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 8:1–8:18, 2021. doi:10.4230/LIPIcs.DISC.2021.8.
- 3 Alkida Balliu, Rustam Latypov, Yannic Maus, Dennis Olivetti, and Jara Uitto. Optimal Deterministic Massively Parallel Connectivity on Forests. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2589–2631, 2023. doi:10.1137/1.9781611977554.ch99.
- 4 Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Efficient Deterministic Distributed Coloring with Small Bandwidth. In *PODC '20: ACM Symposium on Principles of Distributed Computing (PODC)*, pages 243–252, 2020. doi:10.1145/3382734.3404504.
- 5 Leonid Barenboim and Michael Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using nash-williams decomposition. *Distributed Comput.*, 22(5-6):363–379, 2010. doi:10.1007/s00446-009-0088-2.
- 6 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The Locality of Distributed Symmetry Breaking. *Journal of the ACM*, 63(3):20:1–20:45, 2016.
- 7 Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M. Karp, and Jara Uitto. Massively parallel computation of matching and mis in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, pages 481–490, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3293611.3331609.
- 8 Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Łącki, and Vahab Mirrokni. Near-Optimal Massively Parallel Graph Connectivity. In *FOCS*, 2019. doi:10.1109/FOCS.2019.00095.
- 9 Sebastian Brandt, Manuela Fischer, and Jara Uitto. Breaking the Linear-memory Barrier in MPC: Fast MIS on Trees with Strongly Sublinear Memory. *Theoretical Computer Science*, 849:22–34, 2021. doi:10.1016/j.tcs.2020.10.007.
- 10 Yi-Jun Chang. The Complexity Landscape of Distributed Locally Checkable Problems on Trees. In *DISC*, pages 18:1–18:17, 2020. doi:10.4230/LIPIcs.DISC.2020.18.
- 11 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The Complexity of $(\Delta + 1)$ -Coloring in Congested Clique, Massively Parallel Computation, and Centralized Local Computation. In *PODC*, 2019. doi:10.1145/3293611.3331607.
- 12 Yi-Jun Chang and Seth Pettie. A Time Hierarchy Theorem for the LOCAL Model. *SIAM J. Comput.*, 48(1):33–69, 2019. doi:10.1137/17M1157957.
- 13 Sam Coy and Artur Czumaj. Deterministic Massively Parallel Connectivity. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2022. doi:10.1145/3519935.3520055.
- 14 Artur Czumaj, Peter Davies, and Merav Parter. Graph Sparsification for Derandomizing Massively Parallel Computation with Low Space. In *the Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 175–185, 2020. doi:10.1145/3350755.3400282.
- 15 Artur Czumaj, Peter Davies, and Merav Parter. Component Stability in Low-Space Massively Parallel Computation. In *PODC*, 2021. doi:10.1145/3465084.3467903.
- 16 Artur Czumaj, Peter Davies, and Merav Parter. Improved Deterministic $(\Delta + 1)$ -Coloring in Low-Space MPC. In *PODC*, pages 469–479, 2021. doi:10.1145/3465084.3467937.
- 17 Artur Czumaj, Peter Davies, and Merav Parter. Simple, deterministic, constant-round coloring in congested clique and MPC. *SIAM Journal on Computing*, 50(5):1603–1626, 2021. doi:10.1137/20M1366502.
- 18 Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, pages 107–113, 2008. doi:10.1145/1327452.1327492.

- 19 Mohsen Ghaffari. An Improved Distributed Algorithm for Maximal Independent Set. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 270–277, 2016.
- 20 Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrovic, and Ronitt Rubinfeld. Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover. In *PODC*, pages 129–138, 2018. doi:10.1145/3212734.3212743.
- 21 Mohsen Ghaffari, Christoph Grunau, Bernhard Haeupler, Saeed Ilchi, and Václav Rozhoň. Improved Distributed Network Decomposition, Hitting Sets, and Spanners, via Derandomization. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2532–2566, 2023. doi:10.1137/1.9781611977554.ch97.
- 22 Mohsen Ghaffari, Christoph Grunau, and Ce Jin. Improved MPC Algorithms for MIS, Matching, and Coloring on Trees and Beyond. In *DISC*, 2020. doi:10.4230/LIPICs.DISC.2020.34.
- 23 Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds. In *FOCS*, pages 1650–1663, 2019. doi:10.1109/FOCS.2019.00097.
- 24 Mohsen Ghaffari and Ali Sayyadi. Distributed Arboricity-Dependent Graph Coloring via All-to-All Communication. In *ICALP*, pages 142:1–142:14, 2019. doi:10.4230/LIPICs.ICALP.2019.142.
- 25 Mohsen Ghaffari and Jara Uitto. Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation. In *SODA*, 2019. doi:10.1137/1.9781611975482.99.
- 26 Jeff Giliberti, Manuela Fischer, and Christoph Grunau. Deterministic massively parallel symmetry breaking for sparse graphs. *CoRR*, abs/2301.11205, 2023. doi:10.48550/arXiv.2301.11205.
- 27 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In Takao Asano, Shin-ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, *Algorithms and Computation*, pages 374–383, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- 28 Chetan Gupta, Rustam Latypov, Yannic Maus, Shreyas Pai, Simo Särkkä, Jan Studený, Jukka Suomela, Jara Uitto, and Hossein Vahidi. Fast dynamic programming in trees in the mpc model, 2023. arXiv:2305.03693.
- 29 Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. *ACM SIGOPS Operating Systems Review*, pages 59–72, 2007. doi:10.1145/1272996.1273005.
- 30 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *SODA*, 2010. doi:10.1137/1.9781611973075.76.
- 31 Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: A Method for Solving Graph Problems in MapReduce. In *SPAA*, pages 85–94, 2011. doi:10.1145/1989493.1989505.
- 32 Rustam Latypov and Jara Uitto. Deterministic 3-coloring of trees in the sublinear MPC model. *CoRR*, abs/2105.13980, 2021. arXiv:2105.13980.
- 33 Christoph Lenzen and Roger Wattenhofer. Brief Announcement: Exponential Speed-Up of Local Algorithms Using Non-Local Communication. In *PODC*, 2010. doi:10.1145/1835698.1835772.
- 34 Nathan Linial. Distributive Graph Algorithms – Global Solutions from Local Data. In *FOCS*, 1987. doi:10.1109/SFCS.1987.20.
- 35 Nathan Linial. Locality in Distributed Graph Algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.
- 36 Michael Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal on Computing*, 15:1036–1053, 1986. doi:10.1137/0215074.
- 37 Crispin Nash-Williams. Decomposition of Finite Graphs Into Forests. *Journal of the London Mathematical Society*, s1-39:12, 1964. doi:10.1112/jlms/s1-39.1.12.

- 38 Öjvind Johansson. Simple Distributed $\Delta + 1$ -coloring of Graphs. *Information Processing Letters*, pages 229–232, 1999. doi:10.1016/S0020-0190(99)00064-2.
- 39 Merav Parter. $(\delta + 1)$ coloring in the congested clique model. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *International Colloquium on Automata, Languages, and Programming, (ICALP)*, volume 107, pages 160:1–160:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.160.
- 40 Merav Parter and Hsin-Hao Su. Randomized $(\delta + 1)$ -coloring in $o(\log^* \delta)$ congested clique rounds. In *32nd International Symposium on Distributed Computing (DISC)*, volume 121, pages 39:1–39:18, 2018. doi:10.4230/LIPIcs.DISC.2018.39.
- 41 Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and Circuits (On Lower Bounds for Modern Parallel Computation). *Journal of the ACM*, 2018. doi:10.1145/3232536.
- 42 Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *STOC*, pages 350–363, 2020. doi:10.1145/3357713.3384298.
- 43 Tom White. *Hadoop: The Definitive Guide*. O’Reilly Media, Inc., 2009.
- 44 Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *the Proceedings of the SENIX Conference on Hot Topics in Cloud Computing (HotCloud)*, 2010. doi:10.5555/1863103.1863113.

A Massively Parallel Subtree Rake and Compress

This section is dedicated to designing an algorithm that proves Lemma 8, which states that we can compute in $O(\log \log n)$ rounds a partial strict H -decomposition that assigns each node contained in a subtree of size $O(n^{\delta/10})$ to one of $O(\log n)$ layers. We restate the lemma.

► **Lemma 8 (SUBTREERC).** *Let F be a forest on n vertices. There exists a deterministic MPC algorithm SUBTREERC with $O(n^\delta)$ local space, $0 < \delta < 1$, and $\tilde{O}(n)$ global space which takes F as input and computes in $O(\log \log n)$ rounds a partial strict H -decomposition layer: $V(F) \mapsto [\lceil \log(|V(F)| + 1) \rceil] \cup \{\infty\}$ such that $\text{layer}(v) < \infty$ for every node $v \in V(F)$ contained in a subtree of size $n^{\delta/10}$.*

Our algorithm critically relies on the balanced graph exponentiation technique mentioned in Section 1.2 and explained in detail in the full version. In the following definition, you should think about U as being the set of nodes that v has stored in its local memory after the balanced graph exponentiation. We refer to U as good if it contains all nodes within distance $O(\log n)$ of v , except for potentially one direction, for which no node is contained in U .

► **Definition 18** (U is a good subset for v). *Let F be a forest, $U \subseteq V(F)$ and $v \in V(F)$. We say that U is a good subset for v if*

1. $v \in U$,
2. $|N_F(v) \setminus U| \leq 1$, i.e., v has at most one neighbor in F not in U ,
3. $N_F(w) \subseteq U$ for every $w \in U \setminus \{v\}$ with $d_F(v, w) \leq 3L$ where $L := \lceil \log(|U| + 1) \rceil$.

In Appendix A.1, we give a peeling algorithm that takes as input a set U and computes a partial strict H -decomposition with $O(\log n)$ layers by repeatedly peeling off low-degree vertices contained in U . Moreover, if U is good for v , then v gets assigned to one of the $O(\log n)$ layers. This peeling algorithm will later be simulated without any further communication on the machine that has stored the set U in its memory.

23:18 Conditionally Optimal Parallel Coloring of Forests

Using the balanced graph exponentiation technique, we can compute a collection of sets U_1, U_2, \dots, U_k such that for each node v contained in a subtree of size at most $n^{\delta/10}$ there exists some subset U_j that is good for v . In particular, in the full version, we prove the following statement.

► **Lemma 19** (Lemma from Balanced Exponentiation). *Let F be a forest on n vertices. There exists a deterministic low-space MPC algorithm with $O(n^\delta)$ local space, $0 < \delta < 1$, and $O(n \cdot \text{poly}(\log n))$ global space which takes F as input and computes in $O(\log \log n)$ rounds a collection of non-empty sets $U_1, U_2, \dots, U_k \subseteq V(F)$ such that*

1. (Local Space) $|U_j| = O(n^\delta)$ for every $j \in [k]$,
2. (Global Space) $\sum_{j=1}^k |U_j| = O(n \cdot \text{poly}(\log n))$ and
3. for every $v \in V(F)$ which is contained in a subtree of size at most $n^{\delta/10}$, there exists a $j \in [k]$ such that U_j is a good subset for v (see Definition 18).

Moreover, the algorithm also computes for each U_j the forest $F[U_j]$ induced by vertices in U_j and stores it on a single machine.

Our final MPC algorithm for proving Lemma 8 first computes a collection of sets U_1, U_2, \dots, U_k using Lemma 19. Then, the machine storing U_j locally simulates the peeling algorithm of Appendix A.1 with input U_j . As a result, we obtain one partial strict H -decomposition for each set U_j . These partial strict H -decompositions are then combined into one partial strict H -decomposition by assigning each node to the smallest layer assigned by any of the partial strict H -decompositions. More details can be found in Appendix A.2.

A.1 The Conservative Peeling Algorithm

Algorithm 2 computes a partial strict H -decomposition by repeatedly removing low-degree vertices contained in U .

■ **Algorithm 2** Conservative Peeling Algorithm.

```

1: function CONSERVATIVEPEELING(Forest  $F$ , Subset  $U \subseteq V(F)$ )
2:    $V_{\geq 1} \leftarrow V(F)$ , layer:  $V(F) \mapsto \mathbb{N} \cup \{\infty\}$ ,  $L \leftarrow \lceil \log(|U| + 1) \rceil$ 
3:   for  $i = 1, 2, \dots, L$  do
4:     We define  $N_{\geq i}(v) := N_{F[V_{\geq i}]}(v)$  for every  $v \in V_{\geq i}$ .
5:      $V_i^{pivot} \leftarrow \{v \in V_{\geq i} \cap U \mid N_{\geq i}(v) \subseteq U \text{ and } \forall w \in N_{\geq i}(v) \cup \{v\}: |N_{\geq i}(w)| \leq 2\}$ .
6:      $V_i \leftarrow V_i^{pivot} \cup \{v \in V_{\geq i} \cap U \mid |N_{\geq i}(v) \setminus V_i^{pivot}| \leq 1\}$ 
7:      $V_{\geq i+1} \leftarrow V_{\geq i} \setminus V_i$ 
8:     layer( $v$ )  $\leftarrow i$  for every  $v \in V_i$ 
9:   layer( $v$ )  $\leftarrow \infty$  for every  $v \in V_{\geq L+1}$ 
10:  return layer

```

If we would just be interested in computing a partial H -decomposition instead of a strict one, then we could replace Algorithm 2 with the single line $V_i \leftarrow \{v \in V_{\geq i} \cap U \mid |N_{\geq i}(v)| \leq 2\}$.

We first show that Algorithm 2 indeed computes a partial strict H -decomposition.

► **Lemma 20.** *Let F be a forest and $U \subseteq V(F)$. Let layer: $V(F) \mapsto \mathbb{N} \cup \{\infty\}$ be the mapping computed by Algorithm 2 when given F and U as input. Then, layer is a partial strict H -decomposition as defined in Definition 4.*

Next, we show that if U is a good subset for v , as defined in Definition 18, then v gets assigned to one of the $O(\log n)$ layers.

► **Lemma 21.** *Let F be a forest, $U \subseteq V(F)$ and $v \in V(F)$. Let $\text{layer}: V(F) \mapsto \mathbb{N} \cup \{\infty\}$ be the mapping computed by Algorithm 2 when given F and U as input. If U is a good subset for v (see Definition 18), then $\text{layer}(v) \leq L := \lceil \log(|U| + 1) \rceil$.*

Finally, we show that we can locally simulate Algorithm 2 by only knowing the forest induced by vertices in U and the degree of each node U in the original forest.

► **Lemma 22 (Local Sequential Simulation).** *Let F be an arbitrary forest and $U \subseteq V(F)$ be a non-empty subset. Let $\text{layer}: V(F) \mapsto \mathbb{N} \cup \{\infty\}$ be the mapping computed by Algorithm 2 when given F and U as input. There exists a sequential algorithm running in $O(|U|)$ space with the following guarantee: The input of the algorithm is the forest $F[U]$ and the degree $\deg_F(u)$ of each node $u \in U$ in the forest F . The algorithm outputs for each node $v \in U$ its layer $\text{layer}(v)$.*

A.2 Subtree Rake and Compress

Algorithm 3 computes a partial strict H -decomposition with $O(\log n)$ layers where each node in a subtree of size at most x is assigned to one of the layers. We later set $x = n^{\delta/10}$. The correctness follows from the key structural property that partial strict H -decompositions are closed under taking minimums (Lemma 5).

■ **Algorithm 3** SUBTREERC Algorithm.

-
- 1: **function** SUBTREERC(forest F , $x \in \mathbb{N}$)
 - 2: Let $U_1, U_2, U_3, \dots, U_k \subseteq V(F)$ such that for every node $v \in V(F)$ contained in a subtree of size at most x in F , there exists some $j \in [k]$ such that U_j is a good subset for v (see Definition 18)
 - 3: $\text{layer}_j \leftarrow \text{ConservativePeeling}(F, U_j)$ for every $j \in [k]$ ▷ $\text{layer}_j: V(F) \mapsto \mathbb{N} \cup \{\infty\}$
 - 4: $\text{layer}(v) = \min_{j \in [k]} \text{layer}_j(v)$ ▷ $\text{layer}: V(F) \mapsto \mathbb{N} \cup \{\infty\}$
 - 5: **return** layer
-

► **Lemma 23.** *The algorithm above computes a partial H decomposition $\text{layer}: V(F) \mapsto [\lceil \log(|V(F)| + 1) \rceil] \cup \{\infty\}$ such that $\text{layer}(v) < \infty$ for every node $v \in V(F)$ contained in a subtree of size at most x .*

Proof. Lemma 20 states that layer_j is a strict partial H -decomposition for every $j \in [k]$. Hence, Lemma 5 implies that layer is also a strict H -decomposition. Moreover, for every node $v \in V(F)$ contained in a subtree of size at most x in F , there exists some $j \in [k]$ such that U_j is a good subset for v . Thus, Lemma 21 gives that $\text{layer}_j(v) < \infty$ and therefore $\text{layer}(v) < \infty$. ◀

We are now ready to prove Lemma 8.

Proof of Lemma 8. We first run the balanced exponentiation algorithm of Lemma 19 which runs in $O(\log \log n)$ rounds and needs $\tilde{O}(n)$ global space. As a result, we obtain a collection of non-empty subsets $U_1, U_2, \dots, U_k \subseteq V(F)$ satisfying the three properties stated in Lemma 19. In particular, for each $j \in [k]$, there exists one machine which has stored $F[U_j]$. As $|U_j| = O(n^\delta)$ and F is a forest, $F[U_j]$ indeed fits into one machine. Moreover, one can compute in $O(1)$ rounds for each node $v \in V(F)$ its degree $\deg_F(v)$ and store $\deg_F(v)$ for every node $v \in U_j$ in the same machine as we store $F[U_j]$ using standard MPC primitives [27]. Let $\text{layer}_j \leftarrow \text{ConservativePeeling}(F, U_j)$. Lemma 22 implies that we can compute

$\text{layer}_j(u)$ for every node $u \in U_j$ locally on the machine that stores $F[U_j]$ without any further communication. Then, in $O(1)$ rounds we can compute $\text{layer}(v) = \min_{j \in [k]} \text{layer}_j(v)$ for every $v \in V$ using the fact that we can sort N items in $O(1)$ rounds in the low-space MPC model with $\tilde{O}(N)$ global space [27]. In more detail, we create one tuple $(v, \text{layer}_j(v))$ for every $j \in [k]$ and $u \in U_j$ and one tuple (v, ∞) for every node $v \in V(F)$. Then, we sort the tuples according to the lexicographic order. Given the sorted tuples, it is straightforward to determine $\text{layer}(v)$ for every $v \in V$. As $\sum_{j=1}^k |U_j| = \tilde{O}(n)$, it follows that the algorithm needs $\tilde{O}(n)$ global space. It thus remains to argue about the correctness, which directly follows from the third property of Lemma 19 and Lemma 23. ◀

B Coloring, MIS, Matching, and H -decomposition with Optimal Space

In this section we show how to obtain optimal global space by equipping the algorithm from Theorems 13, 16, and 17 with suitable pre- and processing steps that free additional space.

► **Theorem 1.** *There are deterministic $O(\log \log n)$ -round low-space MPC algorithms for 3-coloring, maximal matching and maximal independent set (MIS) on forests. These algorithms use $O(n)$ global space.*

Proof. We perform the standard procedure of iteratively putting in layer i nodes of degree at most 2 for $i = 1$ to $O(\log \log n)$. This removes $O(\text{poly } \log n)$ fraction of the nodes since each iteration layers a constant fraction of the nodes. Therefore, the new number of nodes is $n' = n / \text{poly } \log n$, and an MPC algorithm using $\tilde{O}(n')$ global space uses $O(n)$ words of global space.

So we freeze these initial $O(\log \log n)$ layers obtain G' remaining graph with $n' = n / \text{poly } \log n$ nodes. Then we apply Theorem 13 to compute a 3-coloring in G' in $O(\log \log n)$ rounds and $O(n)$ global space. Finally we complete the solution on the nodes in the frozen layers one layer at a time taking an additional $O(\log \log n)$ rounds.

The claim for MIS and maximal matching follows by the proofs of Theorem 16 and Theorem 17 respectively after computing the H -decomposition and the 3-coloring. ◀

Using a similar preprocessing step, we can also show that a strict H -decomposition of Theorem 12 can be computed with optimal global space.

► **Theorem 2.** *There is a deterministic $O(\log \log n)$ -rounds low-space MPC algorithm that computes a strict H -decomposition with $O(\log n)$ layers on forests in $O(n)$ global space.*

Proof. Same as above, iteratively putting in layer i the pivot nodes and nodes of degree 1 as in Algorithm 1 of Algorithm 1 for $i = 1$ to $O(\log \log n)$. By Corollary 11 and using Lemma 7 with $x = 1$ and $\ell = 3$, we get that each iteration layers a constant fraction of nodes, which implies that $O(\text{poly } \log n)$ fraction of the nodes are removed after $O(\log \log n)$ iterations.

Now we have a partial strict H -decomposition if we assign layer ∞ to the remaining nodes. These nodes form a graph G' with $n' = n / \text{poly } \log n$ nodes, and so we can compute a strict H -decomposition on G' using Algorithm 1 in $O(\log \log n)$ rounds and $\tilde{O}(n') = O(n)$ global space. Therefore, we have computed a strict H -decomposition of G in $O(\log \log n)$ rounds and in $O(n)$ global space. ◀