# Parallel Derandomization for Coloring*

Sam Coy
*Department of Computer Science*
*University of Warwick*
Coventry, United Kingdom
S.Coy@warwick.ac.uk

Artur Czumaj
*Department of Computer Science*
*University of Warwick*
Coventry, United Kingdom
A.Czumaj@warwick.ac.uk

Peter Davies-Peck
*Department of Computer Science*
*Durham University*
Durham, United Kingdom
Peter.W.Davies@durham.ac.uk

Gopinath Mishra
*Department of Computer Science*
*National University of Singapore*
Singapore
Gopinath@nus.edu.sg

***Abstract*—Graph coloring problems are among the most fundamental problems in parallel and distributed computing, and have been studied extensively in both settings. In this context, designing efficient *deterministic* algorithms for these problems has been found particularly challenging.**

**In this work we consider this challenge, and design a novel framework for derandomizing algorithms for coloring-type problems in the *Massively Parallel Computation (MPC)* model with sublinear space. We give an application of this framework by showing that a recent $(degree + 1)$-list coloring algorithm by Halldorsson et al. (STOC'22) in the LOCAL model of distributed computation can be translated to the MPC model and efficiently derandomized. Our algorithm runs in $O(\log\log\log n)$ rounds, which matches the complexity of the state of the art algorithm for the $(\Delta + 1)$-coloring problem.**

***Index Terms*—Parallel algorithms, Graph coloring, Derandomization**

## I. Introduction

The *Massively Parallel Computation* (MPC) model, introduced over a decade ago by Karloff et al. [1], is a contemporary standard theoretical model for parallel algorithms. The model has evolved through the successful emulation of parallel and distributed frameworks, including but not limited to MapReduce [2], Hadoop [3], Dryad [4], and Spark [5]. Drawing inspiration from classical models of parallel computation (e.g., PRAM) and distributed models (e.g., CongestedClique), MPC incorporates numerous shared attributes with these established paradigms. We focus on the *sublinear local space* MPC regime, where machines have local space $s = O(n^\phi)$ for any arbitrary fixed constant $\phi \in (0, 1)$, where $n$ is the number of nodes in the graph. Note that $\phi$ is a parameter of the model. This model has attracted a lot of attention recently, see, e.g., [6]–[22]. Recent works have provided algorithms for fundamental graph problems such as connectivity, approximate matching, maximal matching, maximal independent set, and $(\Delta + 1)$ coloring.

While the main focus on MPC algorithms has been typically concentrating on the design of randomized algorithms, an area which has seen a recent surge in interest in the MPC model is the design of *deterministic algorithms* which are as efficient as their randomized counterparts. In related models of distributed computation this task is sometimes impossible: there are lower bounds in the LOCAL model which separate the randomized complexity from the deterministic complexity of several fundamental algorithmic problems (see, e.g., [23]). However, a combination of the global co-ordination and of the computational power of individual machines within the MPC model allows for a broader selection of tools for derandomizing MPC algorithms. For example, a particularly effective tool in this context is the *method of conditional expectations*, which has been recently used to derandomize several algorithms for fundamental problems in the MPC model [8], [13], [15], [24].

In this paper we develop a general derandomization framework, providing a useful tool for translating some class of randomized LOCAL algorithms to deterministic MPC in a black-box manner. Previous works have either been heavily tailored to specific algorithms [8], [13], [15], [24] or have had severe limitations on graph degree [25]. Our approach allows generic derandomization of a large class of algorithms over a much larger degree range.

As an application of our derandomization framework, we consider the task of designing fast deterministic MPC algorithms for graph coloring — one of the most fundamental algorithmic primitives, extensively studied in various settings for several decades. Graph coloring problems have been playing a prominent role in distributed and parallel computing, not

*The research of Sam Coy is supported in part by the Centre for Discrete Mathematics and its Applications (DIMAP), by an EPSRC studentship, and by the Simons Foundation Award No. 663281 granted to the Institute of Mathematics of the Polish Academy of Sciences for the years 2021–2023.

The research of Artur Czumaj is supported in part by the Centre for Discrete Mathematics and its Applications, by EPSRC award EP/V01305X/1, by a Weizmann-UK Making Connections Grant, by an IBM Award, and by the Simons Foundation Award No. 663281 granted to the Institute of Mathematics of the Polish Academy of Sciences for the years 2021–2023.

The work was done when Gopinath Mishra was a Post Doctoral Fellow at the University of Warwick. The research was supported in part by the Centre for Discrete Mathematics and its Applications (DIMAP), by EPSRC award EP/V01305X/1, and by the Simons Foundation Award No. 663281 granted to the Institute of Mathematics of the Polish Academy of Sciences for the years 2021–2023.

only because of their numerous applications, but also since some variants of coloring problems naturally model typical symmetry breaking problems, as frequently encountered in decentralized systems (see, e.g., [26] for an overview of early advances). Parallel graph coloring has been studied since the 1980s [26], [27], and nowadays $(\Delta+1)$-coloring and $(2\Delta-1)$-edge-coloring[1] are considered among the most fundamental graph problems — benchmark problems in the area (throughout the paper, $\Delta$ refers to the maximum degree of the input graph).

In particular, we study the parallel complexity of a natural generalization of the $(\Delta+1)$-coloring problem, the problem of ***(degree+1)-list coloring*** (D1LC). In the D1LC problem, for a given graph $G = (V, E)$, each node has an input palette of acceptable colors of size one more than its degree, and the objective is to find a proper coloring using these palettes. The recent increasing interest in D1LC has been largely caused by the generality and applicability of D1LC, since, for example, given a partial solution to a $(\Delta+1)$-coloring problem, the remaining coloring problem on the uncolored nodes is an instance of D1LC. It also appears as a subproblem in more constrained coloring problems, e.g., as a subroutine in distributed $\Delta$-coloring algorithms (see, e.g., [28]) and in edge-coloring algorithms (see, e.g., [29]).

### A. Our contribution

In this paper, we develop a framework for generic derandomization of LOCAL algorithms in the deterministic low-space MPC model. This framework uses pseudorandom generators and the method of conditional expectations, which are known techniques, but significantly extends prior work in generality and applicability.

Our main derandomization theorem is Theorem 10. Since this theorem involves some technical definitions, we leave the statement to Section IV-C. However, informally, the result shows that any randomized LOCAL algorithm that can be decomposed into short subprocedures with some natural properties can be efficiently derandomized in MPC.

As an application of our framework, we show that D1LC can be solved efficiently deterministically in the *Massively Parallel Computation* (MPC) model with sublinear local space, matching the complexity of the state-of-the-art MPC algorithms for the simpler $(\Delta+1)$-coloring and $(\Delta+1)$-list coloring problems. We first show how to combine the D1LC framework for the LOCAL model due to Halldórsson et al. [30] with the techniques developed in earlier works on the MPC model, to obtain a randomized MPC algorithm for D1LC working in $O(\log\log\log n)$ rounds, w.h.p. Then we apply our novel derandomization framework to derandomize this algorithm.

**Theorem 1** (D1LC)**.** *Let $\phi \in (0,1)$ be an arbitrary constant. There exists a deterministic algorithm that, for every $n$-node graph $G = (V, E)$, solves the* D1LC *problem using $O(\log\log\log n)$ rounds, in the sublinear local space* MPC *model with local space $s = O(n^\phi)$ and global space $O(m + n^{1+\phi})$.*

[1] $(\Delta+1)$-*coloring* problem is to color a graph of maximum degree $\Delta$ using $\Delta+1$ colors; $(2\Delta-1)$-*edge-coloring* problem is to color the edges of a graph of maximum degree $\Delta$ using $2\Delta-1$ colors.

Observe that the bound in Theorem 1 matches the state-of-the-art bound for the complexity of the simpler $(\Delta+1)$-coloring problem in the (sublinear local space) MPC model (see [12] for the randomized bound and [15] for the deterministic bound). Furthermore, the recently developed framework connecting the complexity of LOCAL and sublinear local space MPC algorithms (see [19], [25]), provides some evidence that our upper bound cannot be asymptotically improved, unless the complexity of the $(\Delta+1)$-coloring problem is $(\log\log n)^{o(1)}$ in the LOCAL model. This is because [19], [25] show that for a class of component stable algorithms and conditioned on the so-called 1-vs-2-cycles conjecture, no sublinear local space MPC algorithm can run faster than the logarithm of the complexity of LOCAL algorithms. (Still, even conditioned on the 1-vs-2-cycles conjecture, it might be conceivable that a non-component stable randomized MPC sublinear local space algorithm can solve $(\Delta+1)$-coloring in $o(\log\log\log n)$ rounds, and further, we do not have any good enough LOCAL lower bounds for coloring, and so maybe an $(\log\log n)^{o(1)}$-rounds LOCAL algorithm is possible.) Finally, notice that Roughgarden et al. [31] showed that proving any super-constant lower bound in the sublinear local space MPC for any problem in P would separate NC[1] from P, making any *unconditional* super-constant (strongly sublinear local space) MPC lower bound unlikely.

*1) Technical contribution:* A natural approach to design efficient MPC algorithms is to simulate a LOCAL algorithm using the so-called graph exponentiation approach (see, e.g., [7], [20], [32], [33]): in the first round each node learns its 2-hop neighborhood, then its 4-hop neighborhood, and so on. In some cases we can even obtain deterministic algorithms in low-space MPC which are exponentially faster than the corresponding *randomized* complexity in LOCAL [25]. However, doing so is often not easy, and faces two main challenges: the first is how to efficiently derandomize algorithms, and the second is that large neighborhoods may not fit onto machines for high-degree instances, which renders challenging many common subroutines in LOCAL algorithms (e.g., in the coloring setting, computing an almost-clique decomposition).

We address the first problem, of derandomization, by developing a general derandomization framework for transferring some class of randomized LOCAL algorithms to deterministic low-space MPC ones. Our framework relies on the use of pseudorandom generators (PRGs) combined with the method of conditional expectations. The approach of using PRGs in derandomization has been used in the past (see, e.g., [15], [25]), but faces a major difficulty in general application. This difficulty is that the stringent constraint of the local space requirement of sublinear local space MPC causes PRGs to fail on a non-trivial proportion of nodes, even if the base randomized procedure succeeds with high probability. Furthermore, attempts to defer or change the outputs of these failed nodes can cause a chain reaction of nodes failing to meet the success requirements of the randomized procedure. To derandomize $(\Delta+1)$-coloring,

[15] addressed this issue by painstakingly analyzing the effects of PRG failures throughout the course of the (highly complex) base randomized algorithm, but this was highly tailored to the specific algorithm and not easily generalizable. [25] did provide a form of general framework, but this required collecting the neighborhoods needed for entire LOCAL algorithms onto machines, and therefore only worked for low values of $\Delta$. These low-$\Delta$ examples sufficed for [25] since the aim was to show that component-stable lower bounds could be surpassed in some cases, but did not aid derandomization of general algorithms.

We overcome these difficulties by providing a suitable framework that formalizes (see Definition 3) a collection of properties that allow us to fully derandomize randomized algorithm using the PRGs. The main technical difference from [25] is that rather than derandomizing full algorithms in one go, we instead decompose them into short subprocedures with certain useful properties. We then have the space necessary to derandomize these subprocedures on individual machines, allowing us to tackle much higher-degree instances.

The framework and its analysis summarized in Theorem 10 form the main technical contribution of our work. We hope that our framework, and in particular, the key lemma (Theorem 10), will prove useful as a powerful black-box derandomization technique in MPC.

We apply this framework to D1LC as our main application. However, for instances with very high degree, we must still reduce the degree before we can fit even 1-hop neighborhoods onto machines. To do so, we employ a *deterministic recursive sparsification* approach similar to [15], [16], where we repeatedly partition an instance of D1LC with maximum degree $\Delta$ into $n^\delta$ D1LC instances, each with maximum degree $\Delta/n^\delta$. Here $\delta$ with $0 \leq \delta \leq \phi$ is to be fixed later, and $\phi$ is our local space parameter, i.e., $s = O(n^\phi)$. All but one of these instances are *valid* D1LC instances and so can be solved using this recursive sparsification if the degree is still too high, and the final instance can only be solved when it is determined which colors are unused in the other instances. In this way, we can reduce the maximum degree of the D1LC instances which we have to solve to an arbitrarily small polynomial in $n$, which is small enough to apply the derandomization framework.

### B. Related work

*Parallel derandomization.:* Our work relies on some sparsification and derandomization techniques developed for parallel and distributed coloring algorithms. Derandomization techniques such as the method of conditional expectations are long-studied and well-understood in classical sequential computation, and were first employed in linear-space MPC (actually, in the mostly equivalent CongestedClique model) by [34]. This was followed by a work extending the techniques to low-space MPC [14]. These works both applied the method of conditional expectations to families of bounded-independence hash functions, a technique for reducing the length of random seeds required for randomized algorithms. However, they are only applicable to algorithms that do not heavily exploit the independence of random choices.

Many algorithms (including all known sublogarithmic LOCAL coloring algorithms) do not appear to have this property, and their analyses effectively use $\Delta$-wise independence or higher, which is too high to efficiently apply families of bounded-independence hash functions. So, some subsequent works on parallel derandomization instead employ pseudorandom generators (PRGs) to reduce the seed space. PRGs are again a well-studied topic in classical computing (see, e.g., [35] for an introduction), and do not have the restriction on independence. The drawback is that PRGs achieving optimal parameters are only known existentially, and computing them requires exponential-time computation — but this is generally permitted in MPC, and even if not, the PRG can be pre-computed sequentially and hard-coded onto MPC machines. [15] used PRG-based derandomization to give an $O(\log\log\log n)$-round deterministic low-space MPC algorithm for $\Delta+1$-coloring, and [25] used it for general derandomization, in order to demonstrate that the method of conditional expectations could be exploited to surpass the component-stable lower bounds of [19]. However, as mentioned, the derandomization results therein were only for low-degree instances.

Implementations of the method of conditional expectations have also recently been used for derandomization in related distributed models, see e.g., [8], [29], [36].

*Parallel and distributed coloring.:* Our application to D1LC continues a long line of research studying the parallel and distributed complexity of graph coloring problems. For the references to earlier work on distributed coloring algorithms we refer to the monograph by Barenboim and Elkin [26] (see also the influential papers by Linial [37], [38]). We will discuss here only more recent advances (and final results) for the four most relevant coloring problems, $(\Delta+1)$-coloring, $(\Delta+1)$-list-coloring, D1LC, and $\Delta$-coloring. After extensive research in the field of distributed computing concerning the $(\Delta+1)$-coloring problem, we understand its complexity for the LOCAL, CongestedClique, and also for the MPC model. For CongestedClique (and also for MPC with linear memory, $s = O(n)$), we now know how to solve $(\Delta+1)$-coloring in a constant number of rounds, see [16], [39]. For the LOCAL model, after a very long line of research, the current state of the art upper bound for randomized algorithm is $\widetilde{O}(\log^2\log n)$ [39], [40]. [2] [3] The best know deterministic algorithm is of $\widetilde{O}(\log^2 n)$ rounds [41] in the LOCAL model.

For the sublinear local space MPC, it is known that the $(\Delta+1)$-coloring algorithm due to Chang et al. [12] can be combined with the network decomposition result of [40] to obtain a randomized $O(\log\log\log n)$-round MPC algorithm, which is currently the state-of-the-art for the problem; this result was derandomized by Czumaj et al. [15]. Furthermore, all algorithms mentioned above for $(\Delta+1)$-coloring can be extended to solve also $(\Delta+1)$-list-coloring.

[2] $\widetilde{O}(f)$ hides a polynomial term in $\log f$.

[3] Through out this paper, $\log^k x$ denotes $(\log x)^k$.

For the D1LC problem, which is a generalization of $(\Delta+1)$-coloring and $(\Delta+1)$-list-coloring, there have not been many comparable bounds until the very recent work of Halldórsson et al. [30]. In particular, D1LC admits a randomized $\widetilde{O}(\log^2 \log n)$-round distributed algorithm in the LOCAL model [30], [41] matching the state-of-the-art complexity for the $(\Delta+1)$-coloring problem [39], [41]. In [30], the authors significantly extended the approaches for $(\Delta+1)$-coloring (in particular, to allow efficient management of nodes of various degrees). As a byproduct, the framework of Halldórsson et al. [30] can be incorporated into a constant-round MPC algorithm assuming the local MPC space is slightly *superlinear*, i.e., $O(n \log^4 n)$ [30, Corollary 2]. We make extensive use of the framework laid out by Halldórsson et al. [30] in their algorithm for LOCAL in the design of our D1LC algorithm. A similar approach has been applied recently for the CONGEST model in [42], [43], solving D1LC in $\widetilde{O}(\log^3 \log n)$ CONGEST rounds, w.h.p. Very recently, a deterministic constant-round algorithm for D1LC has been obtained for the CongestedClique model [44], settling the complexity of D1LC for CongestedClique.

Any omitted content in the current version is presented in the full version [45].

## II. Preliminaries

For $k \in \mathbb{N}$, $[k]$ denotes the set $\{1, \ldots, k\}$. For $a, b \in \mathbb{N}$, $[a, b]$ denotes the set of integers in $\{a, a+1, \ldots, b\}$. We consider a graph $G = (V, E)$ with $V$ as the node set and $E$ as the edge set with $|V| = n$ and $|E| = m$. The set of neighbors of a node $v$ is denoted by $N(v)$ and the degree of a node $v$ is denoted by $d(v)$. For a node $v$, $\Psi(v)$ denotes the list of colors in the color palette of node $v$ and $p(v)$ denotes the size of $\Psi(v)$. The maximum degree of any node in $G$ is denoted by $\Delta$. As we go on coloring the nodes of the graph $G$, the graph will change and the color palettes of the nodes will also change. Often, we denote the current (rather than the input) graph by $G$. For all graphs we consider, we have $p(v) \geq d(v) + 1$. For a subset $X \subseteq V$, $G[X]$ denotes the subgraph induced by $X$ and $m(X)$ denotes the number of edges in $G[X]$.

*Degree+1 list coloring (D1LC).:* The *degree+1 list coloring (D1LC) problem* is for a given graph $G = (V, E)$ and given color palettes $\Psi(u)$ assigned to each node $u \in V$, such that $|\Psi(u)| \geq d(u) + 1$, the objective to find a proper coloring of nodes in $G$ such that each node is assigned to a color from its color palette and no edge in $G$ is monochromatic.

*Massively Parallel Computation model.:* We consider the *Massively Parallel Computation* (MPC) model, which is a parallel system with some number of machines, each of them having some local space $s$. At the beginning of computation, each machine receives some part of the input, with the constraint that it must fit within its local space. In our case, for the D1LC problem, the input is a set of $n$ nodes, $m$ edges, and $n$ color palettes of total size $O(n+m)$. Hence we will require that the number of machines is $\Omega(\frac{n+m}{s})$, for otherwise the input would not fit the system. The computation on an MPC proceeds in synchronous rounds. In each round, each machine processes its local data and performs an arbitrary local computation on its data without communicating with other machines. At the end of each round, machines can exchange messages. Each message is sent only to a single machine specified by the machine that is sending the message. All messages sent and received by each machine in each round have to fit into the machine's local space. Hence, the total number of messages sent by any machine and received by any machine is bounded by $s$, and the total amount of communication across the whole MPC is bounded by $s$ times the number of machines. At the beginning of the next round, each machine can process all messages received in the previous round. When the algorithm terminates, machines collectively output the solution.

Observe that if a single machine can store the entire input, then any problem (like, e.g., D1LC) can be solved in a single round, since no communication is required. In order for our algorithms to be as scalable as possible, normally one wants to consider graph problems in the *sublinear local space regime*, where local space $s = n^\phi$ for any given constant $\phi \in (0, 1)$. (There has been some research considering also the case when $s = \Theta(n)$, or even when $s = n^{1+\phi}$ (in which case one wants to study the case that $s \ll m$) but we will not consider such setting in the current paper.) We will require that the number of machines is not significantly more than required, specifically that it is $\widetilde{O}(n + \frac{m}{s})$ (note that the optimal amount would be $\widetilde{O}(\frac{n+m}{s})$, but our algorithm requires the ability to assign a machine to each node). A major challenge in the design of MPC algorithms in the sublinear local space regime is that the local space of each machine is (possibly) not sufficient to store all the edges incident to a single node. This constraint naturally requires an MPC algorithm to rely on extensive communication between machines, and most of the techniques known are based on some graph sparsification. It is important to note here that even in the sublinear local space regime, the MPC model is known [21] to be stronger than the PRAM model, e.g., it is known that sorting[4] (and in fact, many related tasks, like prefix sum computation) can be performed in a constant number of rounds, even deterministically, see [21]. Observe that with this tool, we can gather nodes' neighborhoods to contiguous blocks of machines, and learn their degrees, in $O(1)$ rounds, and that we can assume, without loss of generality, that the input can be distributed arbitrarily on the first $\Theta(\frac{n+m}{s})$ MPC machines.

## III. Randomized D1LC Algorithm in MPC

In this section we begin with an overview of the LOCAL algorithm for D1LC due to Halldórsson et al. [30]; a more detailed presentation of this algorithm is deferred to the full version [45]. Next, we argue that if the maximum degree of the input instance is not too large, then, when combined with a result of Czumaj et al. [15], this LOCAL algorithm can be efficiently implemented in exponentially fewer rounds in the MPC model. Since this *randomized* implementation is rather straightforward, we will only sketch it here and defer for more details to the Appendix. However, it is not at all clear that

[4]Here we consider sorting of $N$ objects on an MPC with local space $N^\gamma$ and on $N^{1-\gamma}$ machines, for any constant $\gamma > 0$.

the algorithm of Halldórsson et al. [30] can be efficiently derandomized in MPC. The main contribution of this paper is a derandomization framework and its use for a deterministic MPC algorithm for D1LC.

### A. Overview of LOCAL D1LC algorithm of Halldórsson et al.

The LOCAL algorithm for D1LC due to Halldórsson et al. [30] handles the input graph in "ranges" of degree. It begins by coloring vertices with degrees in the range $[\log^7 n, n]$, followed by vertices with degrees in the range $[\log^7 \log n, \log^7 n]$ vertices in the range $[\log^7 \log \log n, \log^7 \log n]$, and so on, giving $O(\log^* n)$ ranges overall. The coloring algorithm runs in $O(\log^* n)$ rounds for each range. However, for ranges after the first the algorithm does not color all nodes with high probability, leaving a set of "bad" nodes. Some standard "shattering" arguments are used (showing that these bad vertices form small components), and then these bad vertices can be colored using a deterministic D1LC algorithm. The post-shattering coloring step on the second degree range ($[\log^7 \log n, \log^7 n]$) is the bottleneck in the algorithm, and gives the overall complexity of $\widetilde{O}(\log^2 \log n)$ LOCAL rounds. [5]

The algorithm for a single degree range uses an *Almost-Clique Decomposition* approach. First, the algorithm decomposes the vertices into *almost-cliques* (each almost clique is dense internally, with few external neighbors) along with a set of *sparse vertices*, whose 2-hop neighborhoods are missing more than some constant fraction of vertices. The sparse vertices and the dense vertices (vertices in almost-cliques) require different techniques to be colored efficiently. The key in general is to generate *slack* for all vertices, where the slack of a vertex is the number of colors available to a vertex minus its degree. This can be achieved in a number of ways, the most common of which being for neighbors of a vertex to color themselves with the same color, or for neighbors of a vertex to delay coloring themselves until later in the algorithm.

For sparse vertices, slack can usually be generated by performing *color trials*: each vertex nominates itself with constant probability; nominated vertices select a color from their palette uniformly at random; and vertices color themselves with their selected color if none of their neighbors have also selected it. For dense vertices, each almost-clique performs some co-ordination, selecting a "leader" whose palette is similar to many other nodes in the almost-clique. Nodes which are least similar to the leader of their almost-clique are colored first, followed by the nodes which are more similar to the leader of their almost-clique.

### B. Randomized MPC implementation for low-degree graphs

We consider the implementation of the (randomized) algorithm of [30] on a single range of degrees in the low-space MPC model. We will only sketch the approach here (since this is not our main result): we give this partial result to build intuition for which aspects of Theorem 1 are challenging to obtain. Formal explanations as to how we implement the subroutines which are already deterministic in [30] is available in the full version [45], and these arguments are necessary for Theorem 1.

Inspecting the algorithm presented in [30] unveils that, in the worst case, each of its steps either require a node $v$ to send messages to all its neighbors (of size $O(\deg(v))$), or to calculate some value or send some messages based on the full content of the 2-hop neighborhood of $v$. This requires $s \geq \Delta^2$ (or equivalently, for a fixed $s$ we need to have $\Delta \leq \sqrt{s}$), since a single machine has to be able to store $\deg(v)$ messages of size $O(\deg(v))$, and the 2-hop neighborhood of any vertex. Note also that the global space required is $O(m + n^{1+\phi})$. (For implementation details, see the full version [45].

If we have this amount of space relative to our maximum degree, then we can implement 1 round of the LOCAL algorithm of [30] in $O(1)$ rounds of low-space MPC (see the full version [45]) for arguments to this effect). Like the LOCAL algorithm, our MPC implementation would succeed with high probability for all nodes with degrees in $[\log^7 n, \sqrt{s}]$.

Also, as mentioned earlier, there is already an algorithm in low-space MPC (see, e.g., Czumaj et al. [15]) which colors D1LC instances with polylogarithmic maximum degree in $O(\log \log \log n)$ rounds deterministically. Combining these two observations yields the following:

**Lemma 2.** *Let $\phi \in (0, 1)$ be an arbitrary constant. There exists an algorithm that, for every $n$-node graph $G = (V, E)$ with maximum degree $\Delta = O(\sqrt{n^\phi})$, solves the D1LC problem using $O(\log \log \log n)$ rounds with high probability, in the sublinear local space MPC model with local space $O(n^\phi)$ and global space $O(m + n^{1+\phi})$.*

Two enhancements are required to get from Lemma 2 to our main result, Theorem 1. In Section IV, we give our framework for derandomizing coloring algorithms in MPC, and in Section V we show that we can derandomize the implementation of the algorithm of [30] for the degree range $[\log^7 n, s^c]$ for some suitable constant $c \in (0, 1/2)$. Finally, in Section VI we show that we can, *deterministically*, reduce any input instance to a collection of instances with lower degree. We ensure that the palettes of these instances are mostly distinct, so that only a constant number of instances need to be colored sequentially.

## IV. Framework to Derandomize MPC Algorithms

In this section we present our black-box derandomization technique for coloring problems in MPC. Our framework relies on a collection of suitable properties (normal distributed procedures, see Definition 3) that allow us to fully derandomize randomized algorithm using a combination of pseudorandom generators (PRGs) and the method of conditional expectations.

### A. Normal distributed procedures

Our framework relies on the notion of normal $(\tau, \Delta)$-round randomized distributed procedure. This notion is introduced to combat the issue that a PRG that fits on a machine in low-space

[5] The actual bound obtained by [30] is $O(\log^3 \log n)$, and when this is combined with the result in [41], one gets the round complexity to be $\widetilde{O}(\log^2 \log n)$.

MPC causes more nodes to fail than the underlying random process it is applied to. We wish to defer these failed nodes to deal with them later, but in some procedures this could cause a chain reaction of failures and result in an unsolvable instance. So, Definition 3 captures those procedures for which we will show we can safely defer failed nodes.

Definition 3 is quite technical, but is conceptually fairly simple, so we first explain its meaning before giving the formal notation. In general, the randomized distributed procedures we are interested in are algorithms that run for some number of rounds in a distributed model (in our case, LOCAL), and for which it is shown that upon termination, a particular desirable property (which we call the "strong success property") holds for all nodes with high probability. In coloring problems, this property could be, for example, that all nodes have sufficient *slack* (difference between remaining palette size and number of uncolored neighbors). More generally, this could include properties such as having low remaining degree (i.e. few neighbors that have not yet terminated and given their final output for the graph problem of interest).

The defining property of a *normal* distributed procedure is that we are able to define a "weak success property" that is still sufficient for the overall algorithm to proceed, and is implied by the strong success property *even if an arbitrary subset of nodes are deferred, changing their outputs from the procedure to the special* DEFER *marker*. That is, if we were to run the procedure, causing all nodes to satisfy the strong success property (which will happen with high probability), and then an adversary were able to defer any subset of nodes, effectively nullifying their outputs, we would still be sure to satisfy the weak success property at all nodes. In this case, deferring does not substantially disrupt the overall algorithm, since if we continue to repeat the procedure on the deferred nodes, we will eventually reach a state in which all nodes have the required property to proceed to the next step.

This may seem like a very strong requirement for a randomized distributed procedure: indeed, it can be seen as a special case of the distributed Lovàsz Local Lemma, and much research has been devoted to the more general case where deferring nodes can cause their neighbors to no longer satisfy any success property (see e.g. [36], [46]). However, as we will see, it applies to many useful procedures, and particularly those for coloring problems, where deferring nodes is almost always helpful and provides slack to neighbors.

**Definition 3.** *A **normal** $(\tau, \Delta)$**-round distributed procedure** running on a graph $G$, of maximum degree* at most $\Delta$*, is a procedure in the randomized* LOCAL *model satisfying the following criteria:*

- *The procedure takes $\tau$ rounds of* LOCAL.
- *At the beginning of the procedure, nodes $v$ have $O(\Delta^\tau)$-word sets of input information $\text{IN}_v$ associated with them.*
- *During the procedure, nodes only use information from their $\tau$-hop neighborhood (i.e., from $\text{IN}_v \cup \bigcup_{u \in N^\tau(v)} \text{IN}_u$) and $O(\Delta^{2\tau})$ random bits, and perform $O(\Delta^{8\tau})$ computation.* [6]
- *The output of the procedure is a new $O(\Delta^\tau)$-word output information $\text{OUT}_v$ for each node, from a set of possible outputs* OUT.
- *The procedure has a **"strong success property"** (computable with $O(\Delta^{8\tau})$ computation) that determines whether it has been successful for a particular node, based on the output information of that node's $\tau$-hop neighborhood (formally, $\text{SSP}_v : \text{OUT}^{N^\tau(v)} \to \{T, F\}$).*
- *At the end of the procedure, for any node $v$,*

$$\mathbf{Pr}\left[\text{SSP}_v\left(\prod_{u \in N^\tau(v)} \text{OUT}_u\right)\right] \geq 1 - \frac{1}{2n} \ .$$

- *The procedure also has a **"weak success property"** (also computable with $O(\Delta^{8\tau})$ computation) that extends the output domain of each node to include a special* DEFER *marker (not to be conferred by the procedure itself) indicating that that node will be deferred until the end of the derandomization (formally, $\text{WSP}_v : \{\text{OUT} \cup \text{DEFER}\}^{N^\tau(v)} \to \{T, F\}$).*
- *Denote the set of nodes that do not satisfy the strong success property as $\overline{\text{SSP}}$. The success properties are such that if a node $v$ satisfies $\text{SSP}_v$, and nodes in $\overline{\text{SSP}}$ are deferred, $v$ still satisfies the weak success property. Formally:*

$$\text{SSP}_v\left(\prod_{u \in N^\tau(v)} \text{OUT}_u\right) \implies$$
$$\text{WSP}_v\left(\prod_{u \in N^\tau(v) \setminus \overline{\text{SSP}}} \text{OUT}_u \times \prod_{u \in N^\tau(v) \cap \overline{\text{SSP}}} \text{DEFER}\right) \ .$$

Let us now discuss how Definition 3 can be applied. The intuition behind Definition 3 is that it captures procedures whose output properties are not damaged too much by unsuccessful nodes deferring. As an example, consider Luby's randomized algorithm for maximal independent set [47]. Luby's algorithm, in $O(\log n)$ rounds of LOCAL, produces an output set that is certainly independent, and is maximal with high probability. We can define both success properties (strong and weak) for a node $v$, to be that $v$ has a node within distance 1 in the resulting output set. Note that this only captures maximality, not independence, but that is sufficient since independence is guaranteed by the process, and a "failed" run only violates maximality. A standard analysis of Luby's algorithm would give, as required, that $\mathbf{Pr}\left[\text{SSP}_v\left(\prod_{u \in N^\tau(v)} \text{OUT}_u\right)\right] \geq 1 - \frac{1}{2n}$, i.e., that a node $v$ is successful with high probability. Notice also that only nodes that are not in the output independent set can fail to satisfy SSP, by definition. Therefore, deferring such nodes does not remove any nodes from the output independent set, and so all nodes satisfying $\text{SSP}_v$ also satisfy $\text{WSP}_v$.

[6] $N^\tau(v)$ denotes the set of vertices that are with in $\tau$-hop neighborhood of $v$.

The arguments above imply that Luby's algorithm is a normal $(O(\log n), \Delta)$-round distributed procedure (the other necessary properties are trivial to check). In fact, we had something stronger: we used essentially the same condition for the strong and weak success property (with the only formal difference being that the weak success property has a domain extended to include DEFER tags). This indicates that, in this case, deferring unsuccessful nodes does not hurt us *at all*. This is something we will also see in our main application to D1LC. Definition 3 is more general, though, and allows WSP to be weaker than SSP in addition to incorporating DEFER tags, which can provide some leeway in derandomization.

We will argue (in Section IV-B) that we can only efficiently derandomize normal $(\tau, \Delta)$-round distributed procedures in MPC when $\tau$ is low (ideally constant). This means that, in most applications, it will not suffice to show that entire algorithms for a problem are normal distributed procedures; we must instead consider those algorithms as series of short subroutines, and then show that each of those subroutines is a normal $(\tau, \Delta)$-round distributed procedure. This means defining success properties that capture an appropriate level of progress after each subroutine, not just correctness of the final output. Indeed, the ability to capture subroutines in this way is one of the main generalizations of our approach over that of [25].

Coloring algorithms are particularly amenable to this type of analysis, since they often consist mostly of subroutines to generate *slack*. That is, some short subroutine is performed, after which analysis shows that nodes' remaining palette sizes will be larger than their remaining degrees by some amount. This measure of slack is then the success property for the subroutine. Importantly, deferring nodes (until the end of the entire coloring process) can only ever increase slack, since those deferred nodes are removed from neighbors' neighborhoods, but do not block any colors from neighbors' palettes. So, we can again define both $\text{SSP}_v$ and $\text{WSP}_v$ to be essentially the same property: that node $v$ has at least some amount of slack at the end of a subroutine. As in the example of Luby's algorithm, deferring unsuccessful nodes (in fact, in this case, even successful nodes) cannot hurt $v$ at all, and $\text{SSP}_v \implies \text{WSP}_v$ under any subset of deferrals.

Next, we discuss some known results on pseudorandom generators that we use in our derandomization framework.

### B. PRGs and derandomization

A *Pseudorandom Generator (PRG)* is a function that takes a short *random seed* and produces a longer string of *pseudorandom* bits, which are computationally indistinguishable from truly random bits. We use the following definition from [35] for indistinguishability:

**Definition 4** (Definition 7.1 in [35])**.** *Random variables $X$ and $Y$ taking values in $\{0,1\}^m$ are $(t, \varepsilon)$* ***indistinguishable*** *if for every non-uniform algorithm $T : \{0,1\}^m \to \{0,1\}$ running in time at most $t$, we have $|\Pr[T(X) = 1] - \Pr[T(Y) = 1]| \leq \varepsilon$.*

Let $U_k$ denote a random variable generated uniformly at random from $\{0,1\}^k$. Then pseudorandom generators are defined as follows:

**Definition 5** (PRG, Definition 7.3 in [35])**.** *A deterministic function $\mathcal{G} : \{0,1\}^d \to \{0,1\}^m$ is an $(t, \varepsilon)$* ***pseudorandom generator (PRG)*** *if (1) $d < m$, and (2) $\mathcal{G}(U_d)$ and $U_m$ are $(t, \varepsilon)$ indistinguishable.*

A simple application of the probabilistic method can show the existence of PRGs with optimal parameters:

**Proposition 6** (Proposition 7.8 in [35])**.** *For all $t \in \mathbb{N}$ and $\varepsilon > 0$, there exists a (non-explicit) $(t, \varepsilon)$ pseudorandom generator $\mathcal{G} : \{0,1\}^d \to \{0,1\}^t$ with seed length $d = \Theta(\log t + \log(1/\varepsilon))$.*

As shown in [25], such a PRG can be computed using relatively low space (but exponential computation).

**Lemma 7** (Lemma 35 of [25], arXiv version)**.** *For all $t \in \mathbb{N}$ and $\varepsilon > 0$, there exists an algorithm for computing the $(t, \varepsilon)$ PRG of Proposition 6 in time $\exp(poly(t/\varepsilon))$ and space $poly(t/\varepsilon)$.*

Next, we show how to use PRGs to derandomize normal distributed procedures.

### C. Derandomizing normal distributed procedures

In this section, we will sometimes be working on graphs $G$ which are smaller induced subgraphs of the original graph. So, we use $n_G$ to denote the number of nodes in $G$, as opposed to $n$ which will denote the original number of nodes. This distinction is primarily because the space bounds of the MPC machines are still in terms of $n$ even when working on a smaller graph $G$. Similarly, for a node $v$, $N_G(v)$ denotes the set of neighbors of $v$ in $G$ and $d_G(v)$ denotes the degree of $v$ in $G$.

For any integer $\ell \in \mathbb{N}$, let $G^\ell$ be the $\ell$-power of $G$, that is, $G^\ell$ is the graph having the same node set as $G$ and any pair of nodes at a distance at most $\ell$ in $G$ form an edge in $G^\ell$.

We begin with the following lemma.

**Lemma 8.** *There is a constant $C$ such that, given an $O(\Delta^{8\tau})$-coloring of $G^{4\tau}$, any normal $(\tau, \Delta)$-round distributed procedure on a graph $G$ can be derandomized in $O(\tau)$ rounds of MPC, using $s = O(\Delta^{\tau C})$ local space per machine and global space $O(n_G \Delta^{\tau C})$, with the following properties:*

- *At most $(\frac{1}{2} + \Delta^{-11\tau})n_G$ nodes are deferred.*
- *All non-deferred nodes $v$ satisfy the weak success property.*

*Proof.* First, for each node $v$ we collect the input information of its $8\tau$-hop neighborhood ($\text{IN}_v \cup \bigcup_{u \in N_G^\tau(v)} \text{IN}_u$) to a dedicated machine. This takes $\tau$ rounds, and requires $O(\Delta^{8\tau} \cdot \Delta^\tau) = O(\Delta^{11\tau})$ space per machine and $O(n_G \Delta^{11\tau})$ global space. Our aim is then to simulate the procedure using randomness produced by the $(\Delta^{11\tau}, \Delta^{-11\tau})$ PRG implied by Proposition 6. This PRG has seed length $d = \Theta(\log \Delta)$ and requires $poly(\Delta^{11\tau})$ space to construct and store. We choose $C$ so that this $poly(\Delta^{22\tau})$ term is $O(\Delta^{\tau C})$.

The PRG, when evaluated on a seed, produces a string of $\Delta^{11\tau}$ pseudorandom bits. We use the provided $O(\Delta^{8\tau})$-

coloring of $G^{4\tau}$ to split this string into the input randomness for each node. By definition of a normal $(\tau, \Delta)$-round distributed procedure, each node requires $O(\Delta^{2\tau})$ random bits, and we provide a node colored $i$ in the $O(\Delta^{8\tau})$-coloring with the $i^{th}$ chunk of $O(\Delta^{2\tau})$ bits from the PRG's output. This means that any pair of nodes within distance $4\tau$ receive disjoint chunks of pseudorandom bits.

The output of the PRG under a random seed is $(\Delta^{11\tau}, \Delta^{-11\tau})$ indistinguishable from a uniform distribution. Consider the process of simulating the procedure for all nodes within distance $\tau$ of a node $v$, and then evaluating the strong success property $\text{SSP}_v(\bigcup_{u \in N^\tau_G(v)} \text{OUT}_u)$. By definition of a normal $(\tau, \Delta)$-round distributed procedure, this combined process requires $O(\Delta^{9\tau})$ computation, and depends on the input information and randomness of nodes up to distance $2\tau$ from $v$ (and note that all nodes within this radius receive different chunks of the PRG's output as their pseudorandom bits). This combined process can therefore be run on one MPC machine. Furthermore, it can be considered a non-uniform algorithm using at most $\Delta^{9\tau}$ computation, and so is 'fooled' by the PRG. This means that the output ($T$ or $F$, indicating whether the success property is satisfied) at $v$ differs with probability at most $\Delta^{-11\tau}$ from what it would be under full randomness. That is, the output will be $F$ with probability at most $\frac{1}{2n_G} + \Delta^{-11\tau}$.

The expected number of nodes that do not satisfy the strong success property, when simulating the procedure using the PRG with a random seed, is therefore at most $\frac{1}{2} + n_G\Delta^{-11\tau}$. As this value is an aggregate of functions computable by individual machines, using the method of conditional expectations (as implemented for low-space MPC in [14], [16]) we can deterministically select a seed for the PRG for which the number of nodes which do not satisfy the success property is at most its expectation (i.e., $\frac{1}{2} + n_G\Delta^{-11\tau}$), in $O(1)$ rounds.

Then, we simply mark the nodes which do not satisfy the strong success property as deferred. By Definition 3, all non-deferred nodes still satisfy the weak success property. We therefore meet the conditions of the lemma. □

We can now iterate Lemma 8 in such a way that we can derandomize full algorithms for problems. To do so, we need these problems to satisfy a *self-reducibility property*, which informally means that partial solutions to the problem should be extendable to full solutions. As discussed, D1LC has this property, which is the reason why it emerges naturally in algorithms for $(\Delta+1)$-coloring or $(\Delta+1)$-list coloring, which themselves are not self-reducible.

**Definition 9.** *Consider a graph problem $\mathcal{P}$ in which each graph node $v$ takes some input information $\text{IN}_v$ and must produce some output labelling. We call $\mathcal{P}$* ***self-reducible*** *if it has the following property:*

- *For any graph $G$, any subset $S \subseteq V(G)$, and any valid output labelling on $G$, nodes $v \in S$ can compute in $O(1)$ rounds of* LOCAL *new input information $\widehat{\text{IN}}_v$ such that:*
  - *the graph induced on $S$, with new inputs $\widehat{\text{IN}}$, forms a valid instance of $\mathcal{P}$, and*
  - *replacing the output labelling of nodes in $S$ with any valid output labelling for this induced problem still forms a valid output of the original problem on $G$.*

Now, we can now state our main derandomization theorem.

**Theorem 10.** *Consider a randomized algorithm $\mathcal{A}$ for a self-reducible problem $\mathcal{P}$ which consists of a series of $k$ ($k = n^{o(1)}$) normal $(\tau, \Delta)$-round distributed procedures (i.e., the final weak success property implies a valid output for $\mathcal{P}$). Then, there is a constant $C$ such that for any $\delta \in (0, 1)$, on any graph $G$ with $\Delta \leq n^{7\delta}$, $\mathcal{A}$ can be derandomized in $O(k\tau + \log^* n_G)$ rounds of* MPC*, using $s = O(n^{7\delta\tau C})$ space per machine and global space $O(n_G \cdot n^{7\delta\tau C})$, with all nodes giving a valid output for $\mathcal{P}$. $\mathcal{A}$ may also contain up to $O(k\tau + \log^* n_G)$ deterministic steps of* LOCAL *or* MPC *adhering to the same space bounds.*

*Proof.* We first note that any deterministic LOCAL or MPC steps contained in $\mathcal{A}$ can simply be run as they are, and do not need any derandomization. For derandomization of the randomized procedure, we begin by computing an $O(\Delta^{8\tau})$-coloring of $G^{4\tau}$ in $O(\tau + \log^* n)$ rounds, by simulating round-by-round the $O(\Delta^2)$-coloring algorithm of Linial [38] on the graph $G^{4\tau}$. Constructing this graph requires collecting $4\tau$-radius balls around each node onto machines, which can be done since we allow $O(n^{7\delta\tau C})$ space per machine.

Then, we apply Lemma 8 to derandomize each of the $k$ normal $(\tau, \Delta)$-round distributed procedures forming $\mathcal{A}$ in order, in each case using $\Delta = n^{7\delta}$ as an upper bound on the maximum degree (even though the actual maximum degree may be significantly lower). The result is that at most $k(\frac{1}{2} + n_G\Delta^{-11\tau})$ nodes are deferred, and the other nodes satisfy the weak success property of the final procedure (have a valid output labelling). This takes $O(k\tau)$ rounds of MPC, and uses $s = O(n^{7\delta\tau C})$ space per machine and global space $O(n_G \cdot n^{7\delta\tau C})$.

Next, since the problem is self-reducible, the deferred nodes can compute, in $O(1)$ rounds, new inputs such that it suffices to solve the problem on the induced graph of deferred nodes with these new inputs. To do so, we recursively apply the above process, again using $\Delta = n^{7\delta}$ as an upper bound on the maximum degree. After $r$ of these recursive applications, the number of remaining deferred nodes is at most $\frac{k}{2} + n(k^r \cdot n^{-11\tau r \cdot 7\delta})$. Taking $r = 1/\delta$ (which is $O(1)$) and $\tau$ to be at least a suitable large constant, we then have $n^{o(1)}$ remaining deferred nodes. We can greedily find valid output labels for them in $O(1)$ further rounds collecting their induced graph onto a single machine, which greedily assigns them valid output labels in any order. Due to the self-reducibility of the problem, we now have a valid output for all nodes for the original problem on $G$.

The total number of rounds used is $O(\tau + \log^* n + kr\tau) = O(k\tau + \log^* n)$, and the space used is $s = O(n^{7\delta\tau C})$ per machine and $O(n_G \cdot n^{7\delta\tau C})$ global space. □

## V. D1LC ALGORITHM WHEN DEGREE IS AT MOST $n^{7\delta}$

In this section we show that our framework from Section IV, when combined with the D1LC algorithm of Halldórsson et al. [30], leads to efficient deterministic MPC algorithm for D1LC when the maximum degree is $n^{7\delta}$. Note that $\delta \in (0, 1)$ is a constant set suitably low relative to the local space parameter $\phi$ and is to be fixed later when we finally prove Theorem 1 in Section VI. Specifically, we show that in $O(\log^* n)$ rounds, with $s = O(n^\phi)$ and $O(n^{1+\phi})$ global space, a subset of the vertices are colored deterministically and the graph induced on the nodes left uncolored will have maximum degree $O(\log^7 n)$ (and will be a D1LC instance, due to the self-reducibility of D1LC). Due to a result of Czumaj et al. [15], this graph can be colored in $O(\log \log \log n)$ rounds.

We presented a brief overview of the D1LC algorithm of Halldórsson et al. in Section III. A more detailed overview (along with pseudocode of the subroutines which we will derandomize here) is in the full version [45]. As well as derandomizing the randomized components of their algorithm using our framework, it is necessary to argue that the deterministic subroutines which their algorithm uses can be implemented in sublinear MPC in exponentially fewer rounds, provided the maximum degree is low enough. As outlined in Section III, this largely follows from arguing that it suffices for the 2-hop neighborhood of each node to be collected: we defer a full explanation to the full version [45].

In this section, graph $G$ is often not clear from the context. So, we use $n_G$ to denote the number of nodes in $G$. For a node $v$, $N_G(v)$ denote the set of neighbors of $v$ in $G$ and $d_G(v)$ denotes the degree of $v$ in $G$. We denote the maximum degree of any node in $G$ by $\Delta_G$. For a node $v$ and $\tau \in \mathbb{N}$, $N^\tau_G(v)$ denotes the nodes in the $\tau$-hop neighborhood of $v$.

### A. Derandomization of key randomized subroutines of [30]

In this section, we consider the randomized subroutines of [30]. Note that all the subroutines succeeds with high probability when the degree of each node is at least $\log^7 n$. Here, we discuss the derandomization of the subroutines by proving that all of them are $(O(1), \Delta)$-round normal distributed procedures in Lemma 11 and arguing that such procedures can be efficiently derandomized by using the framework outlined in the previous section.

**Lemma 11.** *The pre-shattering part of the algorithm of [30] is a series of $O(\log^* \Delta)$ normal $(O(1), \Delta_G)$-round distributed procedures, such that the final weak success property is that all nodes $v$ with degree $d_G(v) \geq \log^7 n$ are properly colored.*

*Proof.* The randomized subroutines used in the pre-shattering part of the algorithm are TRYRANDOMCOLOR, GENERATESLACK, PUTASIDE, SYNCHCOLORTRIAL, and SLACKCOLOR. The pseudocode for all of the above subroutines are in the full version [45]. In the current lemma, our objective is to show that TRYRANDOMCOLOR, GENERATESLACK, PUTASIDE, and SYNCHCOLORTRIAL are all normal $(O(1), \Delta_G)$-round distributed procedures, and SLACKCOLOR consists of a series of $O(\log^* \Delta)$ normal $(O(1), \Delta_G)$-round distributed procedures.

**Algorithm 1:** TRYRANDOMCOLOR(node $v$) from [30]

1. Pick $\psi_v$ u.a.r. from $\Psi(v)$.
2. Send $\psi_v$ to each $u \in N(v)$, receive the set $T = \{\psi_u : u \in N^+(v)\}$, where $N^+(v)$ is the set of neighbours whose colors "conflict" with $v$.
3. If $\psi_v \notin T$ then permanently color $v$ with $\psi_v$.
4. Send and receive permanent colors, and remove the received one from $\Psi(v)$

**Algorithm 2:** GENERATESLACK from [30]

1. $S \leftarrow$ each node $v$ is sampled into $S$ independently with probability $\frac{1}{10}$.
2. For all $v \in S$ in parallel TRYRANDOMCOLOR($v$).

For each of the sub-procedures, we must define strong and weak success properties satisfying the conditions of Definition 3, that capture the notion of success for the subroutine. In each case, our weak success properties will be identical to the corresponding strong success property, other than the extension to deferred nodes. All of our success properties will also deem nodes of degree less then $n^{7\delta}$ to always be successful, regardless of what happens during the randomized process. This means that in this section we will not show any constraints on these low-degree nodes - they will be dealt with afterwards using Lemma 12.

Here, for brevity, we only discuss the pseudocode of TRYRANDOMCOLOR and GENERATESLACK here in Algorithm 1 and Algorithm 2, respectively. We show that both of them are normal $(O(1), \Delta_G)$-round distributed procedures. In the full proof of Lemma 11, discussed in the full version [45], we show the same property for the remaining sub-procedures; this is shown in a very similar fashion.

TRYRANDOMCOLOR: The procedure takes $O(1)$ rounds of LOCAL. Nodes need no other words of input information. Nodes only use information from their neighbors, and each node uses $O(\log \Delta_G)$ random bits to select a color from its palette. Note that the computation is $O(\Delta_G)$. The output information is either a color with which $v$ has permanently colored itself, or FAIL if it does not color itself: this is clearly $O(\Delta_G)$ words of information. We set the success properties $\text{SSP}_v$ and $\text{WSP}_v$ to be that either the slack of $v$ increases from $c \cdot d_G(v)$ for some constant $c$ to $2 \cdot d_G(v)$ [30, Lemma 26], or $d_G(v) < \log^7 n$. Here, for the purposes of $\text{WSP}_v$, deferred neighbors are discounted from $v$'s degree and therefore contribute to its slack. So, deferring neighbors only increases $v$'s slack, and therefore the last condition of Definition 3 is satisfied. This properties are computable in time linear in the degree of a node and based only on the output of the immediate neighbors of a node. The property succeeds for each node with probability $p = \exp(\Omega(s(v)))$ for nodes with $\log^7 n$ and 1 otherwise, so this is with high probability in $n$. So, all necessary conditions are satisfied and TRYRANDOMCOLOR meets Definition 3.

GENERATESLACK: This takes $O(1)$ rounds of LOCAL and nodes need no other information at the beginning of the procedure. During the procedure, nodes only use information from their neighbors and $\widetilde{O}(\Delta_G)$ random bits (to determine whether the node is sampled and if so, what color is attempted). The output is either the color with which $v$ permanently colored itself, or FAIL. The success properties $\text{SSP}_v$ and $\text{WSP}_v$ are that either that $v$ generates *sufficient* slack, or $d_G(v) < \log^7 n$. By *sufficient* here we mean as described in the appropriate statements [30, Lemmas 10, 11, 13, 15, 17, 18]. These slack expressions are quite complicated: depending on the type of node, a different guarantee on the eventual slack is required. We note, however, that these guarantees all succeed with high probability for nodes with $d_G(v) \geq \log^7 n$ (and again, for lower degree nodes we have defined $\text{SSP}_v$ and $\text{WSP}_v$ to always be satisfied) and are computable using only information in the immediate neighborhood of $v$. Deferring nodes again creates slack and so only helps nodes, so the success properties satisfy the necessary conditions of Definition 3.

□

We have shown that the pre-shattering algorithm of [30] consists of a series of $(O(1), \Delta_G)$-round distributed procedures as required. However, our success properties do not constrain nodes $v$ with $d_G(v) < \log^7 n$, so to reach a full D1LC we must deal with these nodes afterwards. We can do so using the following lemma from [15]:

**Lemma 12** (Lemma 14 of [15])**.** *For any $n$-node graph $G$ with maximum degree $\Delta = \log^{O(1)} n$, there exists an $O(\log\log\log n)$-round deterministic algorithm for computing* D1LC*, using $O(n^\alpha)$ space per machine and $O(n^{1+\alpha})$ global space, for any positive constant $\alpha \in (0, 1)$.*

Now, we can apply the pre-shattering algorithm of [30], followed by the low-degree algorithm of [15]. This fits into the framework of Theorem 10 and can therefore be derandomized:

**Lemma 13.** *There are constants $c, C$ such that, for any constant $\delta > 0$, Algorithm 3 performs* D1LC *deterministically in $O(\log\log\log n)$ rounds of* MPC *on any graph $G$ with $\Delta \leq n^{7\delta}$, using $s = O(n^{7\delta cC})$ space per machine and global space $O(n_G \cdot n^{7\delta cC})$.*

*Proof.* We have shown that the pre-shattering algorithm of [30] consists of a series of $k = O(\log^* n) = O(\log\log\log n)$ $(c, \Delta_G)$-round distributed procedures (for some sufficiently large constant $c$), and some deterministic procedures which we can implement efficiently in MPC (see [48]). We follow this with the $O(\log\log\log n)$-round deterministic low-space MPC procedure of Lemma 12. Setting $\alpha$ (in Lemma 12) sufficiently lower than $\delta$, by Theorem 10, this algorithm can be derandomized in $O(\log\log\log n)$ rounds of MPC using $O(n^{7\delta cC})$ space per machine and $O(n_G \cdot n^{7\delta cC})$ global space. □

**Algorithm 3:** DERANDOMIZEDMIDDEGREECOLOR$(G)$

1 Let $\mathcal{A}$ consist of the pre-shattering randomized LOCAL of [30] followed by the $O(\log\log\log n)$-round deterministic low-space MPC algorithm of [15].
2 Derandomize $\mathcal{A}$, using the success properties from the proof of Lemma 11, by Theorem 10.

## VI. OVERALL DETERMINISTIC D1LC ALGORITHM

In this section we extend our (now derandomized) algorithm for D1LC to handle instances with maximum degree $\Delta \geq n^{7\delta}$. The idea is to reduce our D1LC instance to a collection of instances with lower degree so that constant-radius neighborhoods of any node fit onto a single machine. Instead of guaranteeing that there are not too many instances, we ensure that the sequential dependency between instances is not too long. That is, many of the instances resulting from our decomposition can be colored in parallel; there are only $O(1)$ sets of base-case instances which must be solved sequentially.

We use a recursive structure LOWSPACECOLORREDUCE (Algorithm 4) similar to [15], [16]. The recursive structure in Algorithm 4 relies on a partitioning procedure (Algorithm 5) to divide the nodes and colors in the input instance into *bins*. We can follow the analysis of [16] to analyze the partitioning process, since it is the base case that changed. Lemma 14 provides the important properties of the partitioning.

**Algorithm 4:** LOWSPACECOLORREDUCE$(G)$

1 $G_{\text{mid}}, G_1 \ldots, G_{n^\delta} \leftarrow$ LOWSPACEPARTITION$(G)$.
2 For each $i = 1, \ldots, n^\delta - 1$ in parallel: call LOWSPACECOLORREDUCE$(G_i)$.
3 Update color palettes of $G_{n^\delta}$, call LOWSPACECOLORREDUCE$(G_{n^\delta})$.
4 Update color palettes of $G_{\text{mid}}$.
5 Color $G_{\text{mid}}$ using DERANDOMIZEDMIDDEGREECOLOR$(G_{\text{mid}})$.

**Algorithm 5:** LOWSPACEPARTITION$(G)$

1 Let $G_{\text{mid}}$ be the graph induced by the set of nodes $v$ with $d(v) \leq n^{7\delta}$.
2 Let hash function $h_1 : [n] \to [n^\delta]$ map each node $v$ to a bin $h_1(v) \in [n^\delta]$.
3 Let hash function $h_2 : [n^2] \to [n^\delta - 1]$ map colors $\gamma$ to a bin $h_2(\gamma) \in [n^\delta - 1]$.
4 Let $G_1, \ldots, G_{n^\delta}$ be the graphs induced by bins $1, \ldots, n^\delta$ respectively, minus the nodes in $G_{\text{mid}}$.
5 Restrict palettes of nodes in $G_1, \ldots, G_{n^\delta - 1}$ to colors assigned by $h_2$ to corresponding bins.
6 Return $G_{\text{mid}}, G_1, \ldots, G_{n^\delta}$.

**Lemma 14** (Lemma 4.6 of [16]). *Assume that, at the beginning of a call to* LOWSPACEPARTITION*, we have $d(v) < p(v)$ for all nodes $v$. Then, in $O(1)$ MPC rounds with $O(n^{7\delta})$ local space per machine and $O(n+m)$ global space (over all parallel instances), one can deterministically select hash functions $h_1$, $h_2$ such that after the call,*

- *for any node $v \notin G_{mid}$, $d'(v) < 2d(v)n^{-\delta}$, and*
- *for any node $v$, $d'(v) < p'(v)$.*

*Here $d'(v)$ denotes the degree of $v$ in the subgraph induced by the nodes present in the same bucket as $v$ and $p'(v)$ denotes the number of $v$'s palette colors that are in the same bucket as $v$.*

Now, we are ready to prove our main result (Theorem 1):

*Proof of Theorem 1.* As in [15], calling LOWSPACECOLORREDUCE on our input graph creates a recursion tree of $O(1)$ depth (since each recursive call reduces the maximum degree by a $n^{-\delta}$ factor). It therefore creates $O(1)$ sequential sets of base-case instances to solve concurrently, which in Algorithm 4 are solved by DERANDOMIZEDMIDDEGREECOLOR. Furthermore, each set of concurrent instances has at most $n$ nodes in total, since all nodes are only partitioned into one instance, and each instance has maximum degree $n^{7\delta}$.

By Lemma 13, each such instance $G$ is colored in $O(\log\log\log n)$ rounds using $O(n^{7\delta cC})$ space per machine and $O(n_G \cdot n^{7\delta cC})$ global space. The global space used by all concurrent instances is therefore $O(n^{1+7\delta cC})$. Setting $\delta \leq \frac{\phi}{7cC}$, this is $O(n^{\phi})$ space per machine and $O(n^{1+\phi})$ global space. Since receiving the input and the first call to LOWSPACEPARTITION also requires $O(m)$ global space, the overall space bound is $O(m + n^{1+\phi})$. □

## REFERENCES


[1] H. J. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for MapReduce," in *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, 2010, pp. 938–948.

[2] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[3] T. White, *Hadoop: The Definitive Guide: Storage and Analysis at Internet Scale*, 4th ed. Sebastopol, CA: O'Reilly Media, 2015.

[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," *SIGOPS Operating Systems Review*, vol. 41, no. 3, pp. 59–72, March 2007.

[5] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'10)*, 2010.

[6] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev, "Parallel algorithms for geometric graph problems," in *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC'14)*, 2014, pp. 574–583.

[7] A. Andoni, Z. Song, C. Stein, Z. Wang, and P. Zhong, "Parallel graph connectivity in log diameter rounds," in *Proceedings of the 59th IEEE Symposium on Foundations of Computer Science (FOCS'18)*, 2018, pp. 674–685.

[8] P. Bamberger, F. Kuhn, and Y. Maus, "Efficient deterministic distributed coloring with small bandwidth," in *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing (PODC'20)*, 2020, pp. 243–252.

[9] P. Beame, P. Koutris, and D. Suciu, "Communication steps for parallel query processing," *J. ACM*, vol. 64, no. 6, pp. 40:1–40:58, 2017.

[10] S. Behnezhad, S. Brandt, M. Derakhshan, M. Fischer, M. Hajiaghayi, R. M. Karp, and J. Uitto, "Massively parallel computation of matching and MIS in sparse graphs," in *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC'19)*, 2019, pp. 481–490.

[11] S. Behnezhad, M. Hajiaghayi, and D. G. Harris, "Exponentially faster massively parallel maximal matching," in *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS'19)*, 2019, pp. 1637–1649.

[12] Y.-J. Chang, M. Fischer, M. Ghaffari, J. Uitto, and Y. Zheng, "The complexity of $(\Delta+1)$ coloring in congested clique, massively parallel computation, and centralized local computation," in *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC'19)*, 2019, pp. 471–480.

[13] S. Coy and A. Czumaj, "Deterministic massively parallel connectivity," in *Proceedings of the 52th Annual ACM Symposium on Theory of Computing (STOC'22)*, 2022, pp. 162–175.

[14] A. Czumaj, P. Davies, and M. Parter, "Graph sparsification for derandomizing massively parallel computation with low space," *ACM Transactions on Algorithms*, vol. 17, no. 2, May 2021.

[15] ——, "Improved deterministic $(\Delta+1)$ coloring in low-space MPC," in *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC'21)*, 2021, pp. 469–479.

[16] ——, "Simple, deterministic, constant-round coloring in congested clique and MPC," *SIAM J. Comput.*, vol. 50, no. 5, pp. 1603–1626, 2021.

[17] A. Czumaj, J. Łącki, A. Mądry, S. Mitrović, K. Onak, and P. Sankowski, "Round compression for parallel matching algorithms," in *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC'18)*, 2018, pp. 471–484.

[18] M. Ghaffari, T. Gouleakis, C. Konrad, S. Mitrović, and R. Rubinfeld, "Improved massively parallel computation algorithms for MIS, matching, and vertex cover," in *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC'18)*, 2018, pp. 129–138.

[19] M. Ghaffari, F. Kuhn, and J. Uitto, "Conditional hardness results for massively parallel computation from distributed lower bounds," in *Proceedings of the 60th IEEE Symposium on Foundations of Computer Science (FOCS'19)*, 2019, pp. 1650–1663.

[20] M. Ghaffari and J. Uitto, "Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation," in *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*, 2019, pp. 1636–1653.

[21] M. T. Goodrich, N. Sitchinava, and Q. Zhang, "Sorting, searching, and simulation in the MapReduce framework," in *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC'11)*, 2011, pp. 374–383.

[22] J. Łącki, S. Mitrović, K. Onak, and P. Sankowski, "Walking randomly, massively, and efficiently," in *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC'20)*, 2020, pp. 364–377.

[23] Y. Chang, T. Kopelowitz, and S. Pettie, "An exponential separation between randomized and deterministic complexity in the LOCAL model," *SIAM J. Comput.*, vol. 48, no. 1, pp. 122–143, 2019.

[24] M. Fischer, J. Giliberti, and C. Grunau, "Improved deterministic connectivity in massively parallel computation," in *Proceedings of the 36th International Symposium on Distributed Computing (DISC'22)*, 2022, pp. 22:1–22:17.

[25] A. Czumaj, P. Davies, and M. Parter, "Component stability in low-space massively parallel computation," in *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC'21)*, 2021, pp. 481–491.

[26] L. Barenboim and M. Elkin, *Distributed Graph Coloring: Fundamentals and Recent Developments*. Morgan & Claypool Publishers, 2013.

[27] H. J. Karloff, *Fast Parallel Algorithms for Graph-theoretic Problems, Matching, Coloring and Partitioning*. University of California, Berkeley, 1985, phD thesis.

[28] M. Fischer, M. M. Halldórsson, and Y. Maus, "Fast distributed Brooks' theorem," in *Proceedings of the 34th ACM-SIAM Symposium on Discrete Algorithms (SODA'23)*, 2023, pp. 2567–2588.

[29] F. Kuhn, "Faster deterministic distributed coloring through recursive list coloring," in *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, 2020, pp. 1244–1259.

[30] M. M. Halldórsson, F. Kuhn, A. Nolin, and T. Tonoyan, "Near-optimal distributed degree+1 coloring," in *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC'22)*, 2022, pp. 450–463.

[31] T. Roughgarden, S. Vassilvitski, and J. R. Wang, "Shuffles and circuits (on lower bounds for modern parallel computation)," vol. 65, no. 6, pp. 41:1–41:24, Nov. 2018.

[32] M. Ghaffari, "An improved distributed algorithm for maximal independent set," in *Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms (SODA'16)*, 2016, pp. 270–277.

[33] C. Lenzen and R. Wattenhofer, "Brief announcement: Exponential speed-up of local algorithms using non-local communication," in *Proceedings of the 29th ACM Symposium on Principles of Distributed Computing (PODC'10)*, 2010, pp. 295–296.

[34] K. Censor-Hillel, M. Parter, and G. Schwartzman, "Derandomizing local distributed algorithms under bandwidth restrictions," *Distributed Computing*, vol. 33, no. 3, pp. 349–366, 2020.

[35] S. P. Vadhan, "Pseudorandomness," *Foundations and Trends in Theoretical Computer Science*, vol. 7, no. 1-3, pp. 1–336, 2012.

[36] M. Ghaffari, D. G. Harris, and F. Kuhn, "On derandomizing local distributed algorithms," in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, 2018, pp. 662–673.

[37] N. Linial, "Distributive graph algorithms — global solutions from local data," in *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science (FOCS'87)*, 1987, pp. 331–335.

[38] ——, "Locality in distributed graph algorithms," *SIAM J. Comput.*, vol. 21, no. 1, pp. 193–201, 1992.

[39] Y.-J. Chang, W. Li, and S. Pettie, "An optimal distributed $(\Delta + 1)$-coloring algorithm?" in *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC'18)*, 2018, pp. 445–456.

[40] V. Rozhoň and M. Ghaffari, "Polylogarithmic-time deterministic network decomposition and distributed derandomization," in *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC'20)*, 2020, pp. 350–363.

[41] M. Ghaffari and C. Grunau, "Faster deterministic distributed MIS and approximate matching," in *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, B. Saha and R. A. Servedio, Eds. ACM, 2023, pp. 1777–1790.

[42] M. M. Halldórsson, A. Nolin, and T. Tonoyan, "Overcoming congestion in distributed coloring," in *Proceedings of the 41st ACM Symposium on Principles of Distributed Computing (PODC'22)*, 2022, pp. 26–36.

[43] M. Ghaffari and F. Kuhn, "Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition," in *Proceedings of the 62nd IEEE Symposium on Foundations of Computer Science (FOCS'21)*, 2021, pp. 1009–1020.

[44] S. Coy, A. Czumaj, P. Davies, and G. Mishra, "Optimal $(degree + 1)$-coloring in Congested Clique," in *Proceedings of the 50th International Colloquium on Automata, Languages and Programming (ICALP'23), to appear*, 2023.

[45] ——, "Fast parallel degree+1 list coloring," *CoRR*, vol. abs/2302.04378, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2302.04378

[46] P. Davies, "Improved distributed algorithms for the Lovász local lemma and edge coloring," in *Proceedings of the 34th ACM-SIAM Symposium on Discrete Algorithms (SODA'23)*, 2023, pp. 4273–4295.

[47] M. Luby, "A simple parallel algorithm for the maximal independent set problem," *SIAM J. Comput.*, vol. 15, no. 4, pp. 1036–1053, 1986.

[48] S. Coy, A. Czumaj, P. Davies, and G. Mishra, "Fast parallel degree+1 list coloring," 2023.